Submitted to *INFORMS Journal on Computing*
manuscript (Please, provide the mansucript number!)

Constraint Aggregation in Column Generation for Resource-Constrained Set-Covering Models

Daniel Porumbel
CEDRIC, CNAM (Conservatoire National des Arts et Métiers) Paris, daniel.porumbel@cnam.fr

François Clautiaux
Université de Bordeaux, Institut de Mathématiques de Bordeaux (UMR CNRS 5251), INRIA Bordeaux Sud-Ouest, francois.clautiaux@math.u-bordeaux.fr

We propose an aggregation method to reduce the size of column generation models for a class of set-covering problems in which the feasible subsets depend on a resource constraint. The aggregation relies on a correlation between the resource consumption of the elements and the corresponding optimal dual values. The aggregated model obtained allows to find good quality lower bounds more rapidly than the original CG algorithm. The speedup is due to less primal and dual variables in the master, and to an aggregated pricing sub-problem. To guarantee optimaly, we designed an algorithm that iteratively refines the aggregation until the CG optimum is reached. Computational results proving the usefulness of our methods are reported.

1. Introduction

Column generation (CG) is a widespread technique for optimizing linear programs (LPs) with prohibitively-many columns (dual constraints). Without loss of generality, we consider that the objective function is minimized. A well-known drawback of CG is that it may converge very slowly, and the classical lagrangian lower bound does not provide high-quality lower bounds very rapidly. The last decades have seen a surge of interest in stabilization methods that speed-up the convergence and reduce the number of CG iterations [2, 4, 5, 6, 13, 16].

A different approach for optimizing LPs of large size consists of aggregating some of the problem data. Generally speaking, the idea of LP aggregation has a long history in optimization. The goal is usually to transform LPs with high degree of detail into coarser LPs of smaller size. For example, one can aggregate close time instants [15], nearby locations, related scheduling tasks—see [12] for more examples and numerous references. A fact that

motivates our work is that, more recently, aggregation techniques have been successfully applied in CG: they can reduce degeneracy in the master LP, reduce the number of dual variables and produce a stabilization effect [1, 7, 8].

We present an aggregated CG model that produces a valid lower bound for the initial CG optimum. Our model describes an aggregated dual polytope that is included in the original dual polytope. The main idea behind the proposed aggregation can be described as follows. Recall that the dual space in set-covering CG is defined by dual variables y_i that correspond to elements i of a ground set I. In our resource-constrained context, we associate a resource consumption w_i (*e.g.*, weight, length) to each $i \in I$ and all feasible subsets of I have a total resource consumption between some C^- and C^+. We exploit the following assumption: the optimum primal solution consists of subsets with tight resource constraints, and so, we expect a correlation between y_i and w_i, $\forall i \in I$. Given a partition of I into $k \geq 1$ groups, the proposed aggregation forces the dual values y_i in each group to follow an affine function of w_i. This leads to a smaller aggregated CG model with $2k$ dual variables. By iteratively splitting groups, the aggregated model is continuously refined until its optimum reaches the optimum of the non-aggregated CG model.

The remainder is organized as follows. Section 2 recalls the classical CG method and discusses the most related aggregation work. Section 3 describes the aggregated model for a fixed k. Section 4 presents a convergent algorithm that computes a sequence of aggregated dual polytopes by iteratively breaking groups. In Section 5, we show that our linear aggregation has better theoretical properties than a simpler equality aggregation. Section 6 presents computational experiments, followed by conclusions in the last section.

2. Set-Covering Column Generation and Related Work

2.1. CG Models with Resource Constraints and Dynamic Programming Pricing

We first introduce the set-covering models considered throughout this paper. Such models are very general and can be used to solve cutting and packing, vehicle routing, employee scheduling problems, and many real-life problems.

Let $I = \{1, \ldots, n\}$ be a ground set. We formulate a master set-covering integer LP (ILP) with a prohibitively-large set $\mathcal{A}$ of columns $\mathbf{a} = [a_1 \ldots a_n]^\top$ defined by all extreme solutions of a given sub-problem. The classical set-covering problem requires finding the minimum number of configurations needed to cover each $i \in I$. In fact, this study considers a more general set-covering version: each element $i \in I$ has to be covered at least

b_i times ($b_i \in \mathbb{Z}_+$) and each configuration $\mathbf{a} \in \mathcal{A}$ has a cost μ_a (often depending on the total resource consumption of the elements of $\mathbf{a}$). We use primal decision variables λ_a to indicate the number of selections of columns $\mathbf{a} \in \mathcal{A}$, leading to a well-known classical ILP $\big\{\min \sum_{\mathbf{a}\in\mathcal{A}} \mu_a\lambda_a \text{ s.t. } \sum_{\mathbf{a}\in\mathcal{A}} a_i\lambda_a \geq b_i, \forall i \in \{1,\ldots,n\}, \lambda_a \in \mathbb{Z}_+, \forall \mathbf{a} \in \mathcal{A}\big\}$.

We consider the linear relaxation of this ILP, replacing $\lambda_a \in \mathbb{Z}_+$ with $\lambda_a \geq 0, \forall \mathbf{a} \in \mathcal{A}$. The dual LP is written using a vector $\mathbf{y} = [y_1\ y_2\ \ldots\ y_n]^\top \in \mathbb{R}^n_+$ of dual variables and a possibly exponential number of constraints.

$$\left.\begin{array}{ll} \max \mathbf{b}^\top\mathbf{y} & \\ \mathbf{a}^\top\mathbf{y} \leq \mu_a, \ \forall \mathbf{a} \in \mathcal{A} & \\ y_i \geq 0, \qquad i \in 1,\ldots,n & \end{array}\right\} \mathcal{P} \tag{2.1}$$

This is the main *dual set-covering* (`DSCvr`) formulation over polytope $\mathcal{P}$. We can write the above LP more compactly as $\texttt{DSCvr}(\mathcal{P}) = \max\big\{\mathbf{b}^\top\mathbf{y} : \mathbf{y} \in \mathcal{P}\big\}$. Its optimum value $\text{OPT}(\texttt{DSCvr}(\mathcal{P}))$ is the CG optimum, also noted OPT_{CG}. To determine OPT_{CG}, a CG algorithm dynamically generates only a subset of constraints of $\mathcal{P}$ (primal columns). From a dual point of view, the CG can be seen as a cutting-plane method that iterates two steps: (i) consider the current dual polytope $\mathcal{P}^{\text{outer}} \supset P$ described by the constraints generated so far and find an optimal $\mathbf{y} \in \mathcal{P}^{\text{outer}}$; (ii) pricing subproblem: generate a new valid constraint of $\mathcal{P}$ violated by $\mathbf{y}$ and add it to the description of $\mathcal{P}^{\text{outer}}$—or report $\text{OPT}(\texttt{DSCvr}(\mathcal{P}^{\text{outer}})) = \text{OPT}_{\text{CG}}$ if no such constraint can be found.

The pricing CG sub-problem seeks a configuration of most negative reduced cost:

$$\min_{\mathbf{a}\in\mathcal{A}} \big(\mu_a - \mathbf{y}^\top\mathbf{a}\big) \tag{2.2}$$

Given current dual solution $\mathbf{y} \in \mathbb{R}^n_+$, this pricing sub-problem can be defined as follows: select a_i times each $i \in \{1,\ldots,n\}$ so as to maximize the *profit* $\mathbf{y}^\top\mathbf{a}$ *minus* the *cost* μ_a under the resource constraints $C^- \leq \mathbf{w}^\top\mathbf{a} \leq C^+$, where $\mathbf{w} = [w_1\ w_2 \ldots w_n]^\top \in \mathbb{Z}^n_+$ is the vector of resource consumptions and $C^-, C^+ \in \mathbb{Z}_+$ are the two-sided bounds on total consumption.

A key point in CG is the running time of the pricing solution method. If C^+ is bounded, the pricing can often be solved in polynomial time by dynamic programing (DP). For this, we define a profit function $P_{\max}$ that maps any state $(c,i) \in \{0,\ldots,C^+\} \times \{1,\ldots,n\}$ to the maximum value $P_{\max}(c,i)$ of profit $\mathbf{y}^\top\mathbf{a}_{ci}$ over all configurations $\mathbf{a}_{ci}$ that satisfy $\mathbf{w}^\top\mathbf{a}_{ci} = c$ and that only use elements of $\{1,\ldots,i\}$. We start with $P_{\max}(0,0) = 0$ and we determine

the values of $P_{\max}(c,i)$ for all reachable states $(c,i) \in \{C^-,\dots,C^+\} \times \{1,\dots,n\}$ using a recursion such as:

$$P_{\max}(c,i) = \max_{\substack{r\in\{0,\dots,b_i\}\\ r\le c/w_i}} \Big\{ P_{\max}(c - r\cdot w_i, i-1) + r\cdot y_i \Big\} \tag{2.3}$$

In most practical cases, the cost μ_a only depends on the total resource consumption of $\mathbf{a}$, and so, μ_a can be computed separately as a preprocessing. By slightly abusing notations, we thus write $\mu_a = \mu(\mathbf{w}^\top \mathbf{a})$. The best reduced cost is thus attained in a state (c^*, n) such that: $P_{\max}(c^*,n) - \mu(c^*) = \max_{c\in\{C^-,\dots,C+\}} P_{\max}(c,n) - \mu(c)$.

Depending on application-specific features, this DP approach is subject to at least two classical generalizations. First, the state space can be increased to account for additional information, *e.g.*, the current vertex visited in routing problems (see examples in [17, §4.2.3.2]), more resources in vector-packing, etc. Secondly, some transitions between states (values of r in (2.3)) might not be valid, *e.g.*, in bin packing with conflicts some pairs of elements cannot be selected together.

The construction of the $O(C^+ \times n)$ DP states requires an asymptotic running time of $O(n_b C^+)$, where $n_b = \sum_{i=1}^{n} b_i$ is the number of individualized elements. We use n_b instead of n: the elements with demand multiplicities $b_i > 1$ can be selected up to b_i times.

2.2. Related Work in Aggregation Methods

Prohibitively-large ILPs arising in extended formulations are often optimized by approximation. For instance, this can be done by restricting the ILP to a subset of variables and/or constraints, leading to a primal approximation that can be iteratively refined, typically by column-and-row generation, see [18] for a generic view of those methods.

Another way of obtaining a tractable model consists of aggregating constraints or variables. For instance, this is done statically by [20], who define a smaller model whose size depends on a given parameter. An interesting conclusion of [20] is that small values of this parameter are often sufficient to obtain excellent bounds. However this method is static and does not converge toward the optimum of the initial model.

In the context of CG where the master is a set covering/partitioning model, aggregation methods are used to group together elements that are either similar, or often appear together in subproblem solutions. The latter property is used in the DCA algorithm proposed in [1, 7, 8]. This aggregation approach starts from the following restriction of the set of feasible columns. Given a partition of the ground set I, all elements of the same

group are only allowed to arise *together* in any column. The aggregated master LP only contains *compatible columns*, *i.e.*, columns that contain either all or none of the elements of a group. The *primal* covering constraints of the elements of a group are all replaced by a unique representative aggregated constraint. From a dual perspective, a dual aggregated variable represents a group of original dual variables and its value is equal to the sum of the actual dual values in the group. When the pricing subproblem is called, the dual variables are disaggregated and the initial subproblem is solved. The column produced by the subproblem is added to the restricted master program if it is compatible with the current partition of I, or put aside otherwise. The convergence is realized by iteratively updating the partition. At each iteration, the aggregated dual polytope includes the original dual polytope, and so, its optimum is an upper bound for the sought CG optimum.

An aggregation approach that produces a dual polytope inside the original dual polytope consists of enforcing the dual values of "similar" elements to be equal. Such an equality aggregation makes all elements of a group equivalent, *i.e.*, they can be freely exchanged in an aggregated column with no impact on the column feasibility. This has the advantage of reducing the size of the pricing sub-problem (all dual variables of similar elements become one) and of stabilizing the CG process. A recent example of such exchanges can be found in [10], although this is not used explicitly to aggregate the LP.

The latter type of methods relies on the fact that some elements are often almost equivalent in the pricing problem (*e.g.*, consider two similar-size items in cutting-stock). However, if the sub-problem has more complex resource constraints to satisfy, the equivalence of different elements may be less obvious. Such situations can require a more complex correlation between the dual value of an element and its resource consumption. For example, [5] proved that the dual solution vector is always non-decreasing for the cutting-stock problem (when elements are sorted by non-decreasing resource consumption). More generally, a study of optimal dual solutions for cutting-stock [3] show that restricting the dual values to follow a piecewise linear function of the resource consumption leads to optimal dual values in a large majority of the cases. In what follows, we exploit this feature to propose a new type of aggregation.

3. The Aggregated Model for Fixed k

We now present the aggregated model for a fixed k and propose an aggregated CG method based on an aggregated pricing. Given a partition of the elements into k groups, we enforce

the dual values in each group to follow an affine function of their resource consumption. This can be done by adding new linear inequalities to the original dual LP. Consequently, this leads to a dual restriction and thus any dual bound obtained with the new model is also a valid dual bound for the original problem. Based on these restrictions, we eventually reduce both the number of dual variables and dual constraints (primal columns).

3.1. The Aggregated Model : from Dimension n to Dimension $2k$

Let $G_k = \{I^1, I^2, \dots, I^k\}$ be a partition of $I = \{1, \dots, n\}$ and note $n_j = |I^j|$. Given elements I^j of a group $j \in \{1, \dots, k\}$, let $\mathbf{y}^j$, $\mathbf{w}^j$, $\mathbf{b}^j$ and $\mathbf{a}^j$ be the n_j-dimensional column vectors related to dual variables, resource consumptions, demands and, respectively, coefficients of some configuration $\mathbf{a} \in \mathcal{A}$. Without restricting generality, we consider the elements ordered such that $\mathbf{y}$ is the concatenation of k group components: $\mathbf{y}^\top = [y_1 \;\; y_2 \dots y_n] = [(\mathbf{y}^1)^\top \dots (\mathbf{y}^k)^\top]$.

The linear restriction is the following: given any group $j \in \{1, \dots, k\}$, the values of the dual variables $y_1^j, y_2^j, \dots, y_{n_j}^j$ have to follow an affine function of their resource consumptions $w_1^j, w_2^j, \dots, w_{n_j}^j$. Formally, for each group j, we impose that there exists $\alpha^j, \beta^j \in \mathbb{R}$ such that $y_i^j = \alpha^j w_i^j + \beta^j$, $\forall i \in I^j$. This restriction can be implemented by introducing $2k$ additional variables $\alpha^j, \beta^j, j = 1, \dots, k$, and linking these variables to the $\mathbf{y}$ variables in (2.1). We obtain the *restricted dual polytope* in $\mathbb{R}^n_+ \times \mathbb{R}^{2k}$:

$$\left.\begin{array}{ll} \max \mathbf{b}^\top \mathbf{y} & \\ \mathbf{a}^\top \mathbf{y} \le \mu_a, & \forall \mathbf{a} \in \mathcal{A} \\ y_i^j = w_i^j \alpha^j + \beta^j, & \forall j \in 1, \dots, k, i \in I^j \\ y_i^j \ge 0, & \forall j \in 1, \dots, k, i \in I^j \\ \alpha^j, \beta^j \in \mathbb{R}, & \forall j \in 1, \dots, k \end{array}\right\} \mathcal{P}_k^{y,\alpha,\beta} \tag{3.1}$$

PROPOSITION 3.1. *The projection* $proj_y(\mathcal{P}_k^{y,\alpha,\beta})$ *of* $\mathcal{P}_k^{y,\alpha,\beta}$ *onto the variables* $\mathbf{y}$ *verifies* $proj_y(\mathcal{P}_k^{y,\alpha,\beta}) \subseteq \mathcal{P}$.

Proof: Observe that all constraints of $\mathcal{P}$ in (2.1) are still in place in (3.1). □

We cannot generally state $proj_y(\mathcal{P}_k^{y,\alpha,\beta}) = \mathcal{P}$, because only the vectors $\mathbf{y} \in \mathcal{P}$ with the suitable group-wise linear structure do represent valid solutions of (3.1).

We now rewrite (3.1) using only variables α^j and β^j. We first re-write the objective function. For each group j, $\mathbf{y}^j$ can be written as a linear combination of $\mathbf{w}^j$ and $\mathbf{1}_j$ (vector $[1\ 1 \dots\ 1]^\top$ with n_j elements): $\mathbf{y}^j = \mathbf{w}^j\alpha^j + \mathbf{1}_j\beta^j$.

$$\mathbf{b}^\top\mathbf{y} = \sum_{j=1}^{k} \left(\mathbf{b}^j\right)^\top \mathbf{y}^j = \sum_{j=1}^{k} \left(\mathbf{b}^j\right)^\top \left(\mathbf{w}^j\alpha^j + \mathbf{1}_j\beta^j\right) = \sum_{j=1}^{k} \left(\left(\mathbf{b}^j\right)^\top \mathbf{w}^j\right)\alpha^j + \left(\left(\mathbf{b}^j\right)^\top \mathbf{1}_j\right)\beta^j \quad (3.2)$$

The first constraints of (3.1) can be written $\sum_{j=1}^{k} \left(\mathbf{a}^j\right)^\top \mathbf{y}^j \leq \mu_a, \mathbf{a} \in \mathcal{A}$. For each j, we have:

$$\left(\mathbf{a}^j\right)^\top \mathbf{y}^j = \left(\mathbf{a}^j\right)^\top \left(\mathbf{w}^j\alpha^j + \mathbf{1}_j\beta^j\right) = \left(\left(\mathbf{a}^j\right)^\top \mathbf{w}^j\right)\alpha^j + \left(\left(\mathbf{a}^j\right)^\top \mathbf{1}_j\right)\beta^j \quad (3.3)$$

We are now ready to express model (3.1) with variables α^j and β^j only. To simplify the writing in (3.2)-(3.3), we use the following notational shortcuts.

Definition 3.1. Given a configuration $\mathbf{a} \in \mathcal{A}$ and a group j, we define:

– $c_a^j = \left(\mathbf{a}^j\right)^\top \mathbf{w}^j$: *total resource consumption* of the elements of I^j selected in $\mathbf{a}$. Observe that this is the coefficient of variable α^j in (3.3);

– $N_a^j = \left(\mathbf{a}^j\right)^\top \mathbf{1}_j$: *total number of elements of* I^j selected in $\mathbf{a}$. This is the coefficient of variable β^j in (3.3);

– $w_{\min}^j = \min_{i \in I^j} w_i^j$ and $w_{\max}^j = \max_{i \in I^j} w_i^j$: respectively the lowest and highest resource consumption of an element of group j.

We substitute (3.2)-(3.3) in model (3.1) and we reformulate the non-negativity constraints, and so, we obtain an equivalent model in the space $\mathbb{R}^{2k}$:

$$\left.\begin{array}{ll} \max \sum_{j=1}^{k} \left(\left(\mathbf{b}^j\right)^\top \mathbf{w}^j\right)\alpha^j + \left(\left(\mathbf{b}^j\right)^\top \mathbf{1}_j\right)\beta^j & \\ \sum_{j=1}^{k} c_a^j\alpha^j + N_a^j\beta^j \leq \mu_a, \ \forall \mathbf{a} \in \mathcal{A} & \\ w_{\min}^j\alpha^j + \beta^j \geq 0 & \forall j \in 1, \dots, k \\ w_{\max}^j\alpha^j + \beta^j \geq 0 & \forall j \in 1, \dots, k \\ \alpha^j, \beta^j \in \mathbb{R}, & \forall j \in 1, \dots, k \end{array}\right\} \mathcal{P}_k^{\alpha,\beta} \quad (3.4)$$

Proposition 3.2. *There is a bijection between the solutions of* $\mathcal{P}_k^{y,\alpha,\beta}$ *and* $\mathcal{P}_k^{\alpha,\beta}$*:* $proj_{\alpha,\beta}(\mathcal{P}_k^{y,\alpha,\beta}) = \mathcal{P}_k^{\alpha,\beta}$.

Proof: The first constraint of $\mathcal{P}_k^{y,\alpha,\beta}$ in (3.1) is equivalent to the first constraint of $\mathcal{P}_k^{y,\alpha,\beta}$, because the latter constraint is obtained from the former by simply writing $y_i^j = w_i^j\alpha^j + \beta^j$ and applying Def. 3.1. The non-negativity constraint $y_i^j \geq 0$ of $\mathcal{P}_k^{y,\alpha,\beta}$ is also equivalent to the

last two constraints in (3.4): $w_i^j\alpha^j+\beta^j\geq 0, \forall i\in I^j \Longleftrightarrow w_{\min}^j\alpha^j+\beta^j\geq 0$ and $w_{\max}^j\alpha^j+\beta^j\geq 0$, as $w_{\min}^j\alpha^j\leq w_i^j\alpha^j\leq w_{\max}^j\alpha^j (\forall j\in\{1,\ldots,k\})$. □

The new model (3.4) reduces the number of dual variables, which may reduce degeneracy and speed-up the algorithm for the restricted master LP (see *e.g.* [7]).

3.2. Aggregated Model of Minimum Size: Less Variables and Less Constraints

The new linearity restrictions reduced the number of variables from n in the initial model (2.1) to $2k$ in the last model (3.4). Furthermore, some configurations of $\mathcal{A}$ become redundant under these linearity restrictions, and so, we can also reduce the number of possible constraints in (3.4).

For any configuration $\mathbf{a}=[a_1\ a_2\ldots a_n]\in\mathcal{A}$, the associated constraint in (3.4) only uses coefficients N_a^j and c_a^j (calculated using Def. 3.1), *i.e.*, it does not need all n individual values $a_1,\ a_2,\ldots,a_n$. As such, we do not need to express configurations $\mathbf{a}\in\mathcal{A}$ as vectors in $\mathbb{Z}_+^n$, but as aggregated $\mathbb{Z}_+^{2k}$ vectors of the form $\overline{\mathbf{a}}=[c_a^1\ N_a^1\ c_a^2\ N_a^2\ \ldots\ c_a^k\ N_a^k]$. An *aggregated configuration* $\overline{\mathbf{a}}$ corresponds to configuration $\mathbf{a}\in\mathcal{A}$ if $c_a^j=\left(\mathbf{a}^j\right)^\top\mathbf{w}^j$ and $N_a^j=\left(\mathbf{a}^j\right)^\top\mathbf{1}_j$, $\forall j\in\{1,\ldots,k\}$. There might exist more than one disaggregated configuration $\mathbf{a}\in\mathcal{A}$ for the same aggregated configuration $\overline{\mathbf{a}}$, as several configurations $\mathbf{a}$ might generate the same values $c_a^j,\ N_a^j,\ \forall j\in\{1,\ldots,k\}$. Therefore, many constraints in (3.4) are redundant.

Definition 3.2. Given group $j\in\{1,\ldots,k\}$, the set R^j of *feasible resource consumptions* is defined via : $R^j=\{c^j\in\{0,\ldots,C^+\}:\ \exists\mathbf{a}\in\mathcal{A}$ such that $c^j=c_a^j\}$.

Definition 3.3. We introduce notations $N^-(j,c^j)$, $N^+(j,c^j)$ and $\mathcal{A}_k$:

– $N^-(j,c^j)$ and $N^+(j,c^j)$: the minimum and respectively the maximum N_a^j value (number of selected elements, see Def. 3.1) over all valid $\mathbf{a}\in\mathcal{A}$ such that $c_a^j=c^j$. These values are referred to as the minimum and maximum cardinality coefficients for group j and feasible resource consumption $c^j\in R^j$;

– $\mathcal{A}_k$: the set of dominant (non-redundant) configurations $\mathbf{a}\in\mathcal{A}$ such that $N_a^j=N^+(j,c_a^j)$ or $N_a^j=N^-(j,c_a^j)$ *for any* group $j\in\{1,\ldots,k\}$.

By replacing $\mathcal{A}$ with $\mathcal{A}_k$ in model (3.4), we obtain a new model:

$$\left.\begin{array}{ll}\max\sum_{j=1}^k\left(\left(\mathbf{b}^j\right)^\top\mathbf{w}^j\right)\alpha^j+\left(\left(\mathbf{b}^j\right)^\top\mathbf{1}_j\right)\beta^j & \\ \sum_{j=1}^k c_a^j\alpha^j+N_a^j\beta^j\leq\mu_a, & \forall\mathbf{a}\in\mathcal{A}_k \\ w_{\min}^j\alpha^j+\beta^j\geq 0 & \forall j\in 1,\ldots,k \\ w_{\max}^j\alpha^j+\beta^j\geq 0 & \forall j\in 1,\ldots,k \\ \alpha^j,\beta^j\in\mathbb{R} & \forall j\in 1,\ldots,k\end{array}\right\}\mathcal{P}_k \tag{3.5}$$

Obviously, if two configurations $\mathbf{a}, \mathbf{a}' \in \mathcal{A}$ lead to the same values of N_a^j and c_a^j, only one constraint has to be explicitly considered. This new LP can be referred to as the *Dual Set-Covering* (`DSCvr`) LP over polytope $\mathcal{P}_k$ and written $\texttt{DSCvr}(\mathcal{P}_k) = \max\{\sum_{j=1}^{k}((\mathbf{b}^j)^\top \mathbf{w}^j)\alpha^j + ((\mathbf{b}^j)^\top \mathbf{1}_j)\beta^j : [\alpha^1 \dots \alpha^k, \beta^1 \dots \beta^k]^\top \in \mathcal{P}_k\}$. We now show below that $\mathcal{P}_k = \mathcal{P}_k^{\alpha,\beta}$, and so, any solution of $\mathcal{P}_k$ can be rewritten as a solution of the initial LP (2.1).

PROPOSITION 3.3. *Any solution of $\mathcal{P}_k$ can be written as a valid solution of $\mathcal{P}$ in* (2.1).

Proof: Proposition 3.1 states that $proj_y(\mathcal{P}_k^{y,\alpha,\beta}) \subseteq \mathcal{P}$. Proposition 3.2 shows that $\mathcal{P}_k^{y,\alpha,\beta}$ is equivalent to $\mathcal{P}_k^{\alpha,\beta}$. It is enough to show that $\mathcal{P}_k = \mathcal{P}_k^{\alpha,\beta}$.

Using the notations from Def. 3.1 and 3.3, we observe that the values of variables α^j and β^j have to respect one of the two inequalities below for any $\mathbf{a} \in \mathcal{A}$ and $j \in \{1, \dots, k\}$:

1. $c_a^j\alpha^j + N_a^j\beta^j \leq c_a^j\alpha^j + N^+(j, c_a^j)\beta^j$, if $\beta^j \geq 0$;
2. $c_a^j\alpha^j + N_a^j\beta^j \leq c_a^j\alpha^j + N^-(j, c_a^j)\beta^j$, if $\beta^j < 0$.

These inequalities show that any constraint of (3.4) associated to a configuration outside $\mathcal{A}_k$ is dominated by a constraint of some $\mathbf{a} \in \mathcal{A}_k$, *i.e.*, with $N_a^j = N^+(j, c_a^j)$ or $N_a^j = N^-(j, c_a^j)$. All configurations of $\mathcal{A} \setminus \mathcal{A}_k$ are dominated, and so, $\mathcal{P}_k = \mathcal{P}_k^{\alpha,\beta}$. □

3.3. Computational Method: Optimizing $\texttt{DSCvr}(\mathcal{P}_k)$ by CG

3.3.1. Preprocessing Stage To solve $\texttt{DSCvr}(\mathcal{P}_k)$, one first needs to compute the cardinality coefficients $N^+(j, c^j)$, $N^-(j, c^j)$, $\forall j \in \{1, \dots, k\}$, $\forall c^j \in R^j$ from Def. 3.3. For this, we propose a coefficient calculation preprocessing stage, executed only once in advance for each group j. The goal is to find the maximum and minimum number of elements that can be selected from I^j so as to consume a total resource amount of c^j, $\forall c^j \in R^j$. This task is similar to the pricing sub-problem (2.2) for the non-aggregated problem (Section 2.1). It is enough to replace y_i with 1 in (2.3) and to apply an analogous DP twice, *i.e.*, once with a minimization and once with a maximization objective. Such a DP scheme only discovers reachable states for feasible c^j values, *i.e.*, it constructs the set R^j at the same time. Considering all groups $j \in \{1, \dots, k\}$ together, this coefficient calculation stage has the same complexity as the pricing sub-problem DP algorithm for the initial model (2.1).

3.3.2. Optimization by Aggregated CG and Aggregated Pricing For $k = 1$, $\texttt{DSCvr}(\mathcal{P}_k)$ has only two variables and it is possible to enumerate all its non-dominated constraints. When $k > 1$, (3.5) is optimized using CG by iteratively solving the associated pricing sub-problem. This pricing can be solved by disaggregation, *i.e.*, express the current

solution $[\alpha^1 \dots \alpha^k \ \beta^1 \dots \beta^k]^\top$ of (3.5) as a solution $\mathbf{y} \in \mathbb{R}^n$ of (2.1) and apply the classical non-aggregated pricing. This would ignore the fact that the dual values in a same group are correlated with the resource consumption in (3.5).

We now show how the pricing can be solved more efficiently without disaggregation. Iteratively, for each current solution $[\alpha^1 \dots \alpha^k \ \beta^1 \dots \beta^k]^\top$of (3.5), the CG algorithm solves an aggregated version of the sub-problem (2.2): find the *aggregated configuration* $\overline{\mathbf{a}} = [c_a^1 \ N_a^1 \ c_a^2 \ N_a^2 \ \dots \ c_a^k \ N_a^k]$ that maximizes a profit $\sum_{j=1}^k \alpha^j c_a^j + \beta^j N(j, c_a^j)$ minus a cost $\mu(\sum_{j=1}^k c_a^j)$. Formally, the aggregated pricing is written in variables $c^1, c^2, \dots, c^k$ as follows:

$$\begin{aligned}
\max \ & \textstyle\sum_{j=1}^k \alpha^j c^j + \beta^j N(j, c^j) - \mu(\sum_{j=1}^k c_a^j) \\
& C^- \le \textstyle\sum_{j=1}^k c^j \le C^+ \\
& N(j, c^j) = \begin{cases} N^+(j, c^j) & \text{if } \beta^j \ge 0 \\ N^-(j, c^j) & \text{if } \beta^j < 0 \end{cases} && \forall j \in \{1, \dots, k\} \\
& c^j \in R^j && \forall j \in \{1, \dots, k\}
\end{aligned} \tag{3.6}$$

where R^j is the set of feasible resource consumptions (computed only once in advance during the preprocessing stage from Section 3.3.1).

The decision variables $c^1, c^2, \dots, c^k$ are sufficient to represent a solution for (3.6), since any $N(j, c^j)$ term can be deduced from c^j and β^j. All β^j represent input data: we choose from the beginning to use either $N(j, c^j) = N^+(j, c^j)$ or $N(j, c^j) = N^-(j, c^j)$, depending on the sign of β^j. Let us denote $p_c^j = \alpha^j c + \beta^j N(j, c)$ the potential profit that can be obtained with a resource amount of c for group j. The problem (3.6) is rewritten using binary decision variables x_c^j such that x_c^j is 1 only when group j uses a total resource amount of c (for all $j \in \{1, \dots, k\}, c \in R^j$).

$$\begin{aligned}
\max \ & \textstyle\sum_{j=1}^k \sum_{c \in R^j} p_c^j x_c^j - \mu(\sum_{j=1}^k \sum_{c \in R^j} c^j x_c^j) \\
& C^- \le \textstyle\sum_{j=1}^k \sum_{c \in R^j} c^j x_c^j \le C^+ \\
& \textstyle\sum_{c \in R^j} x_c^j = 1 && \forall j \in \{1, \dots, k\} \\
& x_c^j \in \{0, 1\} && \forall j \in \{1, \dots, k\}, c \in R^j
\end{aligned}$$

The resulting aggregated pricing is a multiple-choice variant of the non-aggregated pricing from Section 2.1. The non-aggregated dynamic programming (DP) from Section 2.1 can be extended to an aggregated DP: we define a DP state to each pair $(c, j) \in \{1, \dots, C^+\} \times \{1, \dots, k\}$ and we calculate for each such state a profit function $P_{\max}(c, i)$. In both DP

Algorithm 1: Iterative Inner Dual Approximation

$k \leftarrow 1$, $G_1 \leftarrow \{I\}$ *// initial partition with one group*;
$\mathcal{A}_1 \leftarrow$`initialConstr()` *// all constraints with 2 variables can be generated*;
repeat
  `calcCardCoefs`(G_k) *// cardinality coefs $N^-(c^j, j)$, $N^+(c^j, j)$, see §3.3.1*
  $\text{lb}_{G_k}, \mathcal{A}_k \leftarrow$`aggregCGoptim`$(\mathcal{A}_k)$
  $\text{ub}_{G_k} \leftarrow$ `upBound`$(\mathcal{A}_k, \text{lb}_{G_k})$ *// optional upper bound of OPT_{CG}*
  $G_{k+1} \leftarrow$ `groupSplit`(G_k)
  $\mathcal{A}_{k+1} \leftarrow$ `liftConstr`$(\mathcal{A}_k, G_{k+1})$ *// lift constraints from $\mathcal{P}_k$ to $\mathcal{P}_{k+1}$ (Sec. 4.3)*
  $k \leftarrow k + 1$
until *a stopping condition is reached*;

schemes, the costs can be considered pre-defined functions $\mu : \{0, 1, \ldots, C^+\}$ that do not need to be calculated by DP. The complexity of this aggregated DP does not longer depend on the number $n_b = \sum_{i=1}^{n} b_i$ of individualized elements; this complexity factor is reduced, once the initial preprocessing (Section 3.3.1) is performed.

4. The Convergent Algorithm

Section 3.3.2 describes an aggregated CG method that optimizes (3.5) for a fixed partition $G_k = \{I^1, I^2, \ldots, I^k\}$. The resulting value, hereafter noted lb_{G_k} (or simply lb_k when the exact structure of G_k is not essential) is a lower bound of the sought OPT_{CG}. We now describe how this bound can be iteratively improved to compute OPT_{CG}.

Algorithm 1 provides the general steps of our Iterative Inner Dual Approximation (2IDA) method. The main idea is to iteratively break the groups into smaller subgroups and incrementally refine $\mathcal{P}_k$, similarly to what is done in [8]. In the worst case, $\mathcal{P}_k$ may become equivalent to $\mathcal{P}$ in the last iteration. We need to explain how to split our groups to obtain a new partition (Sec. 4.1), how to compute an (optional) upper bound (Sec. 4.2), and how to make use of the previous iteration to avoid constructing each $\mathcal{P}_{k+1}$ from scratch (Sec. 4.3).

4.1. Split Strategies

2IDA eventually converges towards OPT_{CG} regardless of the way the groups are split: after enough iterations, all groups can be reduced to a size of 1 or 2, leading to an aggregated polytope $\mathcal{P}_k$ equivalent to $\mathcal{P}$. However, the split decisions are crucial for the practical effectiveness of 2IDA.

We present two split strategies. The first uses no information on the structure of $\mathcal{P}_k$, whereas the second uses an estimation of the optimal dual solution.

4.1.1. Basic Split Method to Maintain Regular Group Sizes This split method considers only groups associated to segments of $[0, C^+]$ with a regular length such as $\frac{1}{2}C^+$, $\frac{1}{4}C^+$, $\frac{1}{8}C^+$, etc. The very first split operation takes the midpoint $\frac{C^+}{2}$ and breaks the unique group of $k=1$ into: $I^1 = \{i \in I:\ w_i \leq \frac{C^+}{2}\}$ and $I^2 = \{i \in I:\ w_i > \frac{C^+}{2}\}$. At each next iteration, we choose to split a group j^* that maximizes the *resource consumption spread*, *i.e.*, such that $w^{j^*}_{\max} - w^{j^*}_{\min} = \max_{j \in \{1,\ldots,k\}} (w^j_{\max} - w^j_{\min})$. As for the case $k=1$ above, this group j^* is split by cutting its associated segment in two equal regular segments. If any of these two new segments contains all elements of I^{j^*}, we iterate the same operation on this smaller segment; this can be repeated until we obtain two non-empty sub-groups. This method is general, and does not exploit any information from the current 2IDA step.

4.1.2. Split Strategies that Make $\mathcal{P}_k$ Evolve Towards Better Solutions This method aims at breaking linearity restrictions that are not compatible with a higher-quality solution $\mathbf{y^u}$ at current k, *i.e.*, so as to allow $\mathcal{P}_{k+1}$ to contain solutions closer to $\mathbf{y^u}$.

DEFINITION 4.1. Given a solution $\mathbf{y} \in \mathcal{P}_k$ and a solution $\mathbf{y^u} \in \mathbb{R}^n_+$ we say that direction $\mathbf{y} \to \mathbf{y^u}$ is *an open direction* if there exists $\epsilon > 0$ such that $\mathbf{y}_\epsilon = \mathbf{y} + \epsilon(\mathbf{y^u} - \mathbf{y}) \in \mathcal{P}$. If in addition $\mathbf{b}^\top \mathbf{y^u} > \mathbf{b}^\top \mathbf{y}$, we say that $\mathbf{y} \to \mathbf{y^u}$ is *an improving open direction.*

Obviously, a linear restriction excludes an interesting solution $\mathbf{y^u}$ from the current $\mathcal{P}_k$ if the values in $\mathbf{y^u}$ do not follow an affine function in a given group. Therefore, we choose to split the group for which the "distance" between $\mathbf{y^u}$ and an affine function is the largest. The details of the computation of this distance are discussed in Appendix A. They are not essential for the theoretical description of 2IDA; for now, the above guidelines are enough to discuss all main 2IDA ideas.

We still need to say how to find such high-quality solutions $\mathbf{y^u}$. For example, one can take $\mathbf{y^u}$ as (i) the optimal solution returned by an upper bound routine (§A.1), or (ii) any other high-quality solution generated by problem-specific methods (§A.2 for cutting-stock).

4.2. An Upper Bound Generated From the $\mathcal{P}_k$ Optimum

The optional upper bounding routine of 2IDA takes $\mathbf{y^*_k}$ (optimal solution of $\texttt{DSCvr}(P_k)$ expressed in dimension $\mathbb{R}^n_+$) as input and returns an upper bound solution $\mathbf{y^u}$ such that:
(a) if $\mathbf{b}^\top \mathbf{y^*_k} < \mathrm{OPT}_{\mathrm{CG}}$, the direction $\mathbf{y^*_k} \to \mathbf{y^u}$ is open

(b) if $\mathbf{b}^\top \mathbf{y}^*_\mathbf{k} = \mathrm{OPT}_{\mathrm{CG}}$, then $\mathbf{b}^\top \mathbf{y}^\mathbf{u} = \mathrm{OPT}_{\mathrm{CG}}$.

We hereafter call such $\mathbf{y}^\mathbf{u}$ an *outside reference solution*. The first property (a) is useful for guiding the above group split heuristic (Sec. 4.1.2). Property (b) allows to stop the 2IDA iterative process as soon as $\mathbf{b}^\top \mathbf{y}^*_\mathbf{k} = \mathrm{OPT}_{\mathrm{CG}}$.

Given an optimal solution $\mathbf{y}^*_\mathbf{k}$ of $\texttt{DSCvr}(\mathcal{P}_k)$, let $\mathcal{P}^u \supset \mathcal{P}$ be the polytope determined by the $\mathbf{y}^*_\mathbf{k}$-*tight constraints* of $\mathcal{P}$:

$$\mathcal{P}^u = \left\{ \mathbf{y} \in \mathbb{R}^n_+ : \mathbf{a}^\top \mathbf{y} \leq \mu_a, \forall \mathbf{a} \in \mathcal{A} \text{ such that } \mathbf{a}^\top \mathbf{y}^*_\mathbf{k} = \mu_a \right\} \tag{4.1}$$

PROPOSITION 4.1. *An optimal solution* $\mathbf{y}^\mathbf{u}$ *of polytope* $\mathcal{P}^u$ *(over objective function* $\mathbf{b}^\top \mathbf{y}$*) satisfies the following:*
(1) upper bounding: $OPT_{CG} \leq \mathbf{b}^\top \mathbf{y}^\mathbf{u}$*;*
(2) open direction: if $\mathbf{y}^*_\mathbf{k}$ *is* ***not*** *an optimal solution of* $\texttt{DSCvr}(\mathcal{P})$*, then* $\mathbf{b}^\top \mathbf{y}^*_\mathbf{k} < \mathbf{b}^\top \mathbf{y}^\mathbf{u}$ *and* $\mathbf{y}^*_\mathbf{k} \rightarrow \mathbf{y}^\mathbf{u}$ *is an improving open direction;*
(3) optimality proving: if $\mathbf{b}^\top \mathbf{y}^*_\mathbf{k} = OPT_{CG}$*, then* $\mathbf{y}^\mathbf{u}$ *also satisfies* $\mathbf{b}^\top \mathbf{y}^\mathbf{u} = OPT_{CG}$*.*

Proof: Property (1) actually follows from the $\mathcal{P}^u$ definition (4.1). Since $\mathcal{P}^u$ is constructed from a subset of the constraints of $\mathcal{P}$, we directly have $\mathcal{P}^u \supset \mathcal{P}$ and so, $\mathbf{b}^\top \mathbf{y}^\mathbf{u} \geq \mathrm{OPT}_{\mathrm{CG}}$.

We now prove (2). First, the non-optimality of $\mathbf{y}^*_\mathbf{k}$ directly shows that $\mathbf{b}^\top \mathbf{y}^*_\mathbf{k} < \mathrm{OPT}_{\mathrm{CG}} \leq \mathbf{b}^\top \mathbf{y}^\mathbf{u}$. To prove that the direction $\mathbf{y}^*_\mathbf{k} \rightarrow \mathbf{y}^\mathbf{u}$ is open, let us suppose the opposite: there exists an arbitrarily small $\epsilon > 0$ such that $\mathbf{y}^*_\mathbf{k} + \epsilon(\mathbf{y}^\mathbf{u} - \mathbf{y}^*_\mathbf{k}) \notin \mathcal{P}$. As such, there is some $\mathbf{a} \in \mathcal{A}$ for which $\mathbf{a}^\top \mathbf{y}^*_\mathbf{k} \leq \mu_a$ and $\mathbf{a}^\top \mathbf{y}^*_\mathbf{k} + \epsilon \mathbf{a}^\top (\mathbf{y}^\mathbf{u} - \mathbf{y}^*_\mathbf{k}) > \mu_a$. It would imply that $\mathbf{a}^\top \mathbf{y}^*_\mathbf{k} = \mu_a$ and $\mathbf{a}^\top (\mathbf{y}^\mathbf{u} - \mathbf{y}^*_\mathbf{k}) > 0$, and so, $\mathbf{a}^\top \mathbf{y}^\mathbf{u} > \mathbf{a}^\top \mathbf{y}^*_\mathbf{k} = \mu_a$, *i.e.*, $\mathbf{y}^\mathbf{u}$ would violate the $\mathbf{y}^*_\mathbf{k}$-tight constraint $\mathbf{a}^\top \mathbf{y} \leq \mu_a$. This is impossible, since $\mathbf{y}^\mathbf{u} \in \mathcal{P}^u$ satisfies all $\mathbf{y}^*_\mathbf{k}$-tight constraints in (4.1).

The last point (3) comes from the construction of $\mathcal{P}^u$. If $\mathbf{y}^*_\mathbf{k}$ is optimal in $\mathcal{P}$, the optimal basis of $\mathbf{y}^*_\mathbf{k}$ only contains $\mathbf{y}^*_\mathbf{k}$-tight constraints, *i.e.*, it only uses constraints from $\mathcal{P}^u$. □

We optimize over $\mathcal{P}^u$ by CG: $\mathcal{P}^u$ has exactly the same structure as $\mathcal{P}$ in (2.1), but it has a smaller set of (only $\mathbf{y}^*_\mathbf{k}$-tight) constraints. The separation (pricing) problem for $\mathcal{P}^u$ requires finding a configuration $\mathbf{a} \in \mathcal{A}$ with $\mathbf{a}^\top \mathbf{y}^*_\mathbf{k} = \mu_a$ that maximizes $\mathbf{y}^\top \mathbf{a}$ for the current dual solution $\mathbf{y}$. This separation approach is not new in general LP [11, § 2.3], but we are not aware of other applications in CG. We can formulate it as the classical $\mathcal{P}$ pricing from Sec. 2.1, but with a modified lexicographic objective: first maximize the $\mathbf{y}^*_\mathbf{k}$-profit $(\mathbf{y}^*_\mathbf{k})^\top \mathbf{a}$, and, subject to this, maximize the $\mathbf{y}$-profit $\mathbf{y}^\top \mathbf{a}$. In the original dynamic programming (DP) recursion (2.3), $P_{\max}(c, i)$ represents the maximum $\mathbf{y}$-profit $\mathbf{y}^\top \mathbf{a}_{ci}$ over all configurations

$\mathbf{a}_{ci}$ that satisfy $\mathbf{w}^\top \mathbf{a}_{ci} = c$ and that only use elements of $\{1,\dots,i\}$. We now replace $P_{\max}$ with a function $P_{\max}^{\mathbf{y}_\mathbf{k}^*,\mathbf{y}}$ that maximizes $M \cdot (\mathbf{y}_\mathbf{k}^*)^\top \mathbf{a}_{ci} + \mathbf{y}^\top \mathbf{a}_{ci}$ over the same $\mathbf{a}_{ci}$, using a value of M sufficiently large to make any $\mathbf{y}_\mathbf{k}^*$-profit value more important than any $\mathbf{y}$-profit.

The resulting $\mathcal{P}^u$ pricing requires the same asymptotic running time as the classical $\mathcal{P}$ pricing: we only change the profit function from $P_{\max}$ to $P_{\max}^{\mathbf{y}_\mathbf{k}^*,\mathbf{y}}$, but we use the same DP routine and state set. However, $\mathcal{P}^u$ has far less constraints than $\mathcal{P}$, and so, the CG convergence can be faster on $\mathcal{P}^u$. We still need to prove that this new profit function $P_{\max}^{\mathbf{y}_\mathbf{k}^*,\mathbf{y}}$ does lead the DP pricing from Sec. 2.1 to $\mathbf{y}_\mathbf{k}^*$-tight constraints.

PROPOSITION 4.2. *Given a dual solution* $\mathbf{y}$ *and a* $\texttt{DSCvr}(\mathcal{P}_k)$ *optimum* $\mathbf{y}_\mathbf{k}^*$ *expressed in dimension* $\mathbb{R}^n$*, the above hierarchical profit function* $P_{max}^{\mathbf{y}_\mathbf{k}^*,\mathbf{y}}$ *leads the dynamic program from Sec. 2.1 to a configuration* $\mathbf{a} \in \mathcal{A}$ *that maximizes* $\mathbf{y}^\top \mathbf{a}$ *subject to* $(\mathbf{y}_\mathbf{k}^*)^\top \mathbf{a} = \mu_a$.

Proof: First, since $\mathbf{y}_\mathbf{k}^* \in \mathcal{P}$, there is *no* feasible configuration $\mathbf{a}$ such that $\mathbf{a}^\top \mathbf{y}_\mathbf{k}^* > \mu_a$. Furthermore, $\mathbf{y}_\mathbf{k}^*$-tight configurations such that $\mathbf{a}^\top \mathbf{y}_\mathbf{k}^* = \mu_a$ do exist (otherwise, a solution $\mathbf{y}_\mathbf{k}^* + \epsilon[1\dots1]^\top$ could be feasible and better than $\mathbf{y}_\mathbf{k}^*$), and so, we only need to prove these $\mathbf{y}_\mathbf{k}^*$-tight constraints can be effectively discovered by DP.

For a sufficiently large M, $P_{\max}^{\mathbf{y}_\mathbf{k}^*,\mathbf{y}}$ leads this DP routine to any maximum $\mathbf{y}_\mathbf{k}^*$-profit that could be reached by the same DP on $\mathbf{y}_\mathbf{k}^*$ (optimizing $\mathbf{a}^\top \mathbf{y}_\mathbf{k}^*$ over all $\mathbf{a} \in \mathcal{A}$). The correctness of this DP routine guarantees that $P_{\max}^{\mathbf{y}_\mathbf{k}^*,\mathbf{y}}$ surely leads to a $\mathbf{y}_\mathbf{k}^*$-tight constraint. □

4.3. From lb_k to lb_{k+1} in Two Steps: Lift $\mathcal{P}_k$ to $\mathbb{R}^{2k+2}$ and Fully Optimize $\texttt{DSCvr}(\mathcal{P}_{k+1})$

After solving $\texttt{DSCvr}(\mathcal{P}_k)$ at iteration k, Alg. 1 splits a group j^* and generates (sub-)groups j_1 and j_2. A new model (3.5), associated to a new polytope $\mathcal{P}_{k+1}$ has to be optimized. We use the columns already generated at iteration k to warm-start the CG at the next iteration $k+1$, *i.e.*, the CG routine from Sec. 3.3.2 is not run from scratch at iteration $k+1$. First, the $\mathcal{P}_k$ constraints generated so far are lifted from dimension $\mathbb{R}^{2k}$ to dimension $\mathbb{R}^{2k+2}$. The optimal solution of $\texttt{DSCvr}(\mathcal{P}_k)$ is also lifted as follows: set $\alpha^{j_1} = \alpha^{j_2} = \alpha^{j^*}$ and $\beta^{j_1} = \beta^{j_2} = \beta^{j^*}$ for j_1 and j_2, and keep unchanged α_j and β_j for all $j \in \{1,2,\dots,j^*-1,j^*+1,\dots,k-1,k\}$.

More formally, we note $\mathcal{A}_k^{\text{tight}}$ the set of $\mathbf{y}_\mathbf{k}^*$-tight constraints of $\mathcal{P}_k$ generated while determining the optimum solution $\mathbf{y}_\mathbf{k}^*$ of $\texttt{DSCvr}(\mathcal{P}_k)$. Our approach first constructs a dual polytope $\mathcal{P}'_{k+1}$ only by lifting such $\mathcal{P}_k$ constraints. $\mathcal{A}'_{k+1} = \left\{\mathbf{a}' \in \mathcal{A}_{k+1} : \exists \mathbf{a} \in \mathcal{A}_k^{\text{tight}},\ \text{s. t. } c_a^j = c_{a'}^j,\ \ N_a^j = N_{a'}^j\ \forall j \neq j^*, j \in \{1,\dots,k\}\right\}$, *i.e.*, we restrict the set $\mathcal{A}_{k+1}$ of non-dominated $\mathcal{P}_{k+1}$ constraints (see Def 3.3, Sec. 3.2) to those that can be lifted from $\mathcal{A}_k$.

We optimize $\mathcal{P}'_{k+1}$ by CG, solving the same pricing as for $\mathcal{P}_{k+1}$ but over $\mathcal{A}'_{k+1}$ instead of $\mathcal{A}_{k+1}$. In fact, we use the aggregated pricing version (*i.e.*, the multi-choice dynamic program from Sec. 3.3.2), but with only two decision levels j_1 and j_2, leading to a much faster algorithm (but applied once for each lifted constraint). When no such constraint can be found, we turn to the original pricing for $\mathcal{P}_{k+1}$. Since $\mathcal{A}'_{k+1} \subset \mathcal{A}_{k+1}$, we have $\mathcal{P}'_{k+1} \supset P_{k+1}$, and so, $\text{OPT}(\texttt{DSCvr}(\mathcal{P}'_{k+1})) \geq \text{OPT}(\texttt{DSCvr}(\mathcal{P}_{k+1})) \geq \text{OPT}(\texttt{DSCvr}(\mathcal{P}_k))$. If $\text{OPT}(\texttt{DSCvr}(\mathcal{P}'_{k+1})) = \text{OPT}(\texttt{DSCvr}(\mathcal{P}_k))$, we can directly state $\text{lb}_{k+1} = \text{lb}_k$ only using lifted constraints. Experiments suggest that lb_{k+1} can sometimes be computed this way in almost neglectable time.

To summarize, the iterative 2IDA method actually optimizes $\texttt{DSCvr}(\mathcal{P}_{k+1})$ in two steps: (1) it first lifts $\mathcal{P}_k$ constraints to generate and (very rapidly) optimize $\mathcal{P}'_{k+1} \supset \mathcal{P}_{k+1}$ and (2) it fully optimizes $\mathcal{P}_{k+1}$ using the standard aggregated CG from Sec. 3.3.2. The full algorithmic template of the $\texttt{DSCvr}(\mathcal{P}_{k+1})$ optimization is presented in Algorithm 2.

5. Theoretical Property of the Linear Aggregation

At each step k, 2IDA solves a relaxation of the initial *primal* problem. This relaxation comes from the fact that the aggregation leads to non-valid configurations implicitly added to the master LP. Instead of only using valid columns from subproblem solutions, the aggregated model can implicitly use linear combinations of valid columns and non-valid exchange vectors associated to the new linearity restrictions.

These particular exchange vectors, hereafter called *linearity-based exchange vectors* influence the aggregated model as follows. Starting from the linearity restriction $y_i^j = \alpha^j w_i^j + \beta^j$, we observe that y_i^j can also be expressed without terms α^j and β^j, as a linear function of any two variables among $y_1^j, y_2^j, \ldots, y_{n_j}^j$; let us chose $y_{\min}^j$ and $y_{\max}^j$, corresponding to the elements of lowest and (respectively) largest resource consumption in I^j. Given a column involving y_i^j, one can replace y_i^j with a convex combination of $y_{\min}^j$ and $y_{\max}^j$ and obtain an artificial column that is valid in the aggregated model but not necessarily in the original one. Such replacements are referred to as an exchange process.

An interesting property of our aggregated model is that these *linearity-based* exchange processes generate artificial columns that cannot violate the resource constraints, although they can have fractional (infeasible) coefficients.

PROPOSITION 5.1. *Let* $\mathbf{a}$ *be a feasible configuration, and* $\mathbf{e}$ *be a linearity-based exchange vector. Any linear combination* $\hat{\mathbf{a}} = \mathbf{a} + \psi\mathbf{e}$ *does verify* $C^- \leq \hat{\mathbf{a}}^\top \mathbf{w} \leq C^+$, $\forall \psi \in \mathbb{R}$.

Algorithm 2: Two-Step CG for $\texttt{DSCvr}(\mathcal{P}_{k+1})$: optimize over $\mathcal{P}'_{k+1} \supset \mathcal{P}_k$ then over $\mathcal{P}_{k+1}$

Data: Optimal solution $[\alpha^1 \dots \alpha^k \ \beta^1 \dots \beta^k]$ of $\mathcal{P}_k$, existing constraints $\mathcal{A}_k^{\text{tight}}$

Result: $\text{lb}_{k+1} = \text{OPT}(\texttt{DSCvr}(\mathcal{P}_{k+1}))$

Step 1: lift $\mathcal{P}_k$ to $\mathcal{P}'_{k+1} \supset \mathcal{P}_{k+1}$

Lift aggregated solution $[\alpha^1 \dots \alpha^k \ \beta^1 \dots \beta^k] \in \mathcal{P}_k$ to the space of $\mathcal{P}_{k+1}$;

– $\alpha^{j_1}, \alpha^{j_2} \leftarrow \alpha^{j^*}$ and $\beta^{j_1}, \beta^{j_2} \leftarrow \beta^{j^*}$ *// break j^* into (unlinked) groups j_1, j_2* ;

– keep unchanged the values α^j and β^j for all $j \neq j^*$;

repeat

  for $\mathbf{a} \in \mathcal{A}_k^{tight}$ **do**

    – given dual solution $[\alpha \ \beta]$, solve the aggregated multiple-choice pricing variant (Sec. 3.3.2) with 2 elements (j_1 and j_2) and two-sided capacities $C^- - \sum_{\substack{1 \le j \le k \\ j \neq j^*}} c_a^j$ and $C^+ - \sum_{\substack{1 \le j \le k \\ j \neq j^*}} c_a^j$ to lift $\mathbf{a}$ to some $\mathbf{a}' \in \mathcal{A}_{k+1}$;

    – $\mathcal{A}'_{k+1} \leftarrow \mathcal{A}'_{k+1} \cup \{\mathbf{a}'\}$;

  – optimize over current $\mathcal{P}'_{k+1}$ described by configurations $\mathcal{A}'_{k+1}$ only

  – update $\text{OPT}(\texttt{DSCvr}(P'_{k+1}))$ and the current dual solution $[\alpha, \beta]$

until *no configuration* $\mathbf{a}'$ *of negative reduced cost can be found*;

if $\text{OPT}(\texttt{DSCvr}(\mathcal{P}'_{k+1})) = \text{lb}_k$ **return** lb_k;

Step 2: standard $\mathcal{P}_{k+1}$ optim

repeat

  – given dual solution $[\alpha, \beta]$, solve the aggregated multiple-choice pricing variant (Sec. 3.3.2) on $k+1$ levels and generate a new configuration $\mathbf{a}$;

  – $\mathcal{A}_{k+1} \leftarrow \mathcal{A}_{k+1} \cup \{\mathbf{a}\}$;

  – optimize current $\mathcal{P}_{k+1}$ described by above $\mathcal{A}_{k+1}$ and update $[\alpha, \beta]$;

until *no configuration* $\mathbf{a}$ *of negative reduced can be found*;

return $\text{OPT}(\texttt{DSCvr}(\mathcal{P}_{k+1}))$;

Proof: Since $\mathbf{a}$ is feasible, it does satisfy $C^- \le \mathbf{a}^\top \mathbf{w} \le C^+$. Thus, it is enough to show that $\mathbf{a}$ and $\hat{\mathbf{a}}$ have the same resource consumption.

We focus on the exchange process restricted to a given group I^j. Without loss of generality, we consider that $y^j_{\min}$ and $y^j_{\max}$ are (respectively) the first and the last element of I^j. We examine how any other element y^j_i of $\mathbf{y}^j$ can be written as a combination of $y^j_{\min}$ and $y^j_{\max}$. First, we use a classical substitution to determine $\alpha^j = \frac{y^j_{\max} - y^j_{\min}}{w^j_{\max} - w^j_{\min}}$ and $\beta^j = \frac{w^j_{\min} y^j_{\max} - w^j_{\max} y^j_{\min}}{w^j_{\min} - w^j_{\max}}$.

Replacing this in $y_i^j = \alpha^j w_i^j + \beta^j$, we obtain $y_i^j = \frac{y^j_{\max} - y^j_{\min}}{w^j_{\max} - w^j_{\min}} w_i^j + \frac{w^j_{\min} y^j_{\max} - w^j_{\max} y^j_{\min}}{w^j_{\min} - w^j_{\max}}$. The exchange process $\mathbf{a} + \psi\mathbf{e} \longrightarrow \hat{\mathbf{a}}$ (restricted to group j) can be written as follows:

$$\begin{bmatrix} a^j_{\min} \\ \vdots \\ a^j_i \\ \vdots \\ a^j_{\max} \end{bmatrix} + \psi \begin{bmatrix} \frac{w^j_{\max} - w^j_i}{w^j_{\max} - w^j_{\min}} \\ \vdots \\ -1 \\ \vdots \\ \frac{w^j_i - w^j_{\min}}{w^j_{\max} - w^j_{\min}} \end{bmatrix} \longrightarrow \begin{bmatrix} a^j_{\min} + \psi \frac{w^j_{\max} - w^j_i}{w^j_{\max} - w^j_{\min}} \\ \vdots \\ a_i - \psi \\ \vdots \\ a^j_{\max} + \psi \frac{w^j_i - w^j_{\min}}{w^j_{\max} - w^j_{\min}} \end{bmatrix},$$

where ψ can be positive or negative, which may respectively decrease or increase the coefficient of a_i in the resulting artificial column $\hat{\mathbf{a}}$. Observe that the magnitude of ψ needs to be limited (*e.g.*, $\psi \leq a_i$) to keep $\mathbf{a} + \psi\mathbf{e}$ valid in the aggregated model; however, the theorem holds for any $\psi \in \mathbb{R}$.

Note that in $\mathbf{a} + \psi\mathbf{e}$, the only coefficients to be modified are related to $a^j_{\min}$, $a^j_{\max}$ and a_i. In the original column, the total resource consumption of these three elements is $a^j_{\min} w^j_{\min} + a^j_i w_i + a^j_{\max} w^j_{\max}$. In $\mathbf{a} + \psi\mathbf{e}$, this resource consumption becomes:

$$(a^j_{\min} + \frac{w^j_{\max} - w^j_i}{w^j_{\max} - w^j_{\min}}\psi) w^j_{\min} + (a^j_i - \psi) w^j_i + (a^j_{\max} + \frac{w^j_i - w^j_{\min}}{w^j_{\max} - w^j_{\min}}\psi) w^j_{\max}$$

This simplifies to $a^j_{\min} w^j_{\min} + a^j_i w^j_i + a^j_{\max} w^j_{\max}$, which means that the resource consumption is the same in $\mathbf{a}$ and $\mathbf{a} + \psi\mathbf{e}$, *i.e.*, the initial resource consumption never change by applying *linearity-based exchange vectors.* □

Let us compare the linear aggregation with a simpler aggregation that only imposes equality constraints, *e.g.*, consider fixing $\alpha^j = 0, \forall j \in \{1, \ldots, k\}$. We show below that the artificial columns generated by equality aggregation may violate resource constraints. Indeed, an equality aggregation $y_i = y_j$ would make a_i and a_j interchangeable in any column $\mathbf{a}$. This would lead to a classical (equality-based) exchange vector with two non-zero elements at positions i and j that can be combined with valid columns to produce artificial columns as follows:

$$[\ldots a_i \ldots a_j \ldots]^\top + \psi \cdot [\ldots 1 \ldots -1 \ldots]^\top \longrightarrow [\ldots a_i + \psi \ldots a_j - \psi \ldots]^\top,\ \forall \psi \in [-a_i, a_j] \quad (5.1)$$

PROPOSITION 5.2. *Configurations produced by exchange vectors resulting from an equality aggregation as in* (5.1) *can violate the resource constraints.*

Proof: It is sufficient to give an example of an artificial column that violates the capacity constraint. Take $C^- = 8$, $C^+ = 10$ and an instance with two elements such that $w_1 = 8$ and $w_2 = 3$. Column $[0, 3]^\top$ is valid; by applying the exchange vector $[1, -1]^\top$ with $\psi = 3$ in (5.1), one obtains artificial column $[3, 0]^\top$ with total resource consumption 24. To find an artificial column that violates the minimum resource constraint, an analogous example can be produced by taking column $[1, 0]^\top$ and $\psi = -1$. □

6. Numerical Evaluation

We tested our aggregation methodology on three cutting-stock problems that cover all features of the general dual LP (2.1): different costs μ_a of configurations $\mathbf{a} \in \mathcal{A}$, different demands b_i, as well as two-sided limits C^- and C^+ on the total feasible consumption.

6.1. Problem Definitions, Instances and Experimental Conditions

We first recall the cutting-stock context. We are given a set $I = \{1, 2, \ldots, n\}$ of items such that each item $i \in I$ has weight w_i. Each $i \in I$ is ordered b_i times and a feasible solution consists of a set of valid cutting patterns (configurations). A valid cutting pattern is a subset I of maximum total weight C. The resulting LP is the Gilmore-Gomory model from [9], which is a special case of model (2.1). We consider the following problem variants:

The standard cutting-stock problem (CSP) asks to minimize the number of patterns, *i.e.*, it uses the same cost $\mu_a = 1$ for any pattern $\mathbf{a} \in \mathcal{A}$, regardless of its weight. This corresponds to $C^- = 0$ and $C^+ = C$ in the models from Sec. 2.

The multiple-length CSP (ML-CSP) is a CSP variant in which the patterns do not have all the same cost. We consider three bins (rolls) with lengths (widths) $C_0 = C$, $C_1 = 0.9C$ and $C_2 = 0.8C$ of costs 1, 0.9 and respectively 0.8. The cost μ_a of a pattern $\mathbf{a} \in \mathcal{A}$ is given by the cost of the smallest bin (roll) that can hold a total length (width) of $\mathbf{w}^\top \mathbf{a}$, *e.g.*, if $\mathbf{w}^\top \mathbf{a} = 0.85C$, then $\mu_a = 0.9$.

The low-waste CSP (LW-CSP) is a CSP variant that imposes two limits C^- and C^+ on the total length of feasible patterns; the cost is always $\mu_a = 1$, $\forall \mathbf{a} \in \mathcal{A}$. One can see $C^+ = C$ as the fixed width of a roll and $C^+ - C^-$ as a maximum acceptable waste. Such waste limits can arise in industry when it is not possible to recycle pieces of waste larger than $C^+ - C^-$. If a solution uses more than b_i times some item i, then such non-demanded items are not considered as waste, because they are potentially useful for future orders.

Table 1 General characteristics of the CSP instances considered in this paper. Columns 2, 3, 4 and 5 respectively indicate (the interval of) the number of items, the capacity, the demand value and the item weight.

Name	n	C	avg. $\mathbf{b}$	avg. $\mathbf{w}$ span	description
`vb20`[a]	20	10000	$[10, 100]$	$[1, \frac{1}{2}C]$	25 random instances [19]
`vb50-c1`[a]	50	10000	$[50, 100]$	$[1, \frac{3}{4}C]$	20 random instances [19]
`vb50-c2`[a]	50	10000	$[50, 100]$	$[1, \frac{1}{2}C]$	20 random instances [19]
`vb50-c3`[a]	50	10000	$[50, 100]$	$[1, \frac{1}{4}C]$	20 random instances [19]
`vb50-c4`[a]	50	10000	$[50, 100]$	$[\frac{1}{10}C, \frac{1}{2}C]$	20 random instances [19]
`vb50-c5`[a]	50	10000	$[50, 100]$	$[\frac{1}{10}C, \frac{1}{4}C]$	20 random instances [19]
`vbInd`[a]	[5,43]	[4096,200000]	$[1, 300]$	$[\frac{1}{10}C, \frac{1}{2}C]$	Industrial instances [19]
`m01`	100	100	1	$[1, C]$	1000 random instances [3];
`m20`	100	100	1	$[\frac{20}{100}C, C]$	1000 random instances [3];
`m35`	100	100	1	$[\frac{35}{100}C, C]$	1000 random instances [3];
`Hard`	≈ 200	100000	$1-3$	$[\frac{20}{100}C, \frac{35}{100}C]$	Instances known to be very hard

[a] These first 7 instance sets correspond (respectively) to the following sets of files `CSTR20b50c[1-5]*`, `CSTR50b50c1*`, `CSTR50b50c2*`, `CSTR50b50c3*`, `CSTR50b50c4*`, `CSTR50b50c5*`, `CSTR*p*`, all publicly available at `www.math.u-bordeaux1.fr/~fvanderb/data/randomCSPinstances.tar.Z` or at `www.math.u-bordeaux1.fr/~fvanderb/data/industrialCSPinstances.tar.Z` (for the last set).

We generated ML-CSP and LW-CSP instances by adding specific parameters to existing standard CSP instances. We consider a dozen of CSP benchmark sets each having up to 1000 individual instances (accounting for a total number of 3127 instances). The technical characteristics of these CSP instances are provided in Table 1.

We implemented both 2IDA and a classical non-stabilized CG algorithm based on dynamic programming (DP) for pricing the columns. A similar implementation approach has been used for *all* DP routines of 2IDA, *i.e.*, for the preprocessing stage (Sec. 3.2), for the aggregated pricing used to optimize over $\mathcal{P}_k$ (Sec. 3.3.2) and for the pricing needed by the optional upper bounds (Sec. 4.2). We used a 2.27GHz (Intel Core i3) CPU, with the `gnu g++` compiler on Linux. The master LPs are solved using Cplex 12.3 and Ilog Concert libraries. Detailed results (instance by instance) are also publicly available at `cedric.cnam.fr/~porumbed/csp/`.

6.2. Detailed Evaluation on Standard Cutting-Stock

We distinguish two 2IDA versions depending on the use of the optional intermediate upper bounds from Sec. 4.2. When such upper bounds are available, they are used to guide

the split strategy (see below) or to prove optimality for lower k values, *i.e.*, as soon as $\lceil \text{lb}_k \rceil = \lceil \text{ub}_k \rceil$.

However, the convergence can also be realized by executing a pure CG process in the end, *i.e.*, we turn to classical CG when $\lceil \text{lb}_k \rceil + 1 = \lceil \text{ub}_k \rceil$ or if $k = 10$. This setting was determined empirically: for some instances, 2IDA exhibits a fast convergence only when it takes the right split decisions at each step. When it cannot take such optimal split decisions, the CG process run in the end leads a faster convergence with the current implementation.

We use the split strategy guided by reference solutions from Sec. 4.1.2. In the first 2IDA iterations, this reference solution is given by a so-called *dual-feasible function* (DFF) with a piecewise linear form [3]. As soon as 2IDA requires more intervals than the number of pieces of the DFF, this reference solution is given by the intermediate upper bound (when available). All technical details of this strategy are specified in Appendix A. We can also turn to the simpler split method from Sec. 4.1.1 when the value of k exceeds the number of intervals of the DFF and when no upper bound is available (pure 2IDA).

Table 2 presents the computing effort (CPU time and sub-problems calls) for two 2IDA versions: a "full 2IDA" version that does use intermediate upper bounds ("yes" in Column 2) and a "pure 2IDA" version that does *not* use them ("no" in Column 2). The indicated upper bounding computing effort includes both the intermediate upper bounds and the upper bounds calculated during the final CG process.

The first conclusion is that, in 7 of 11 benchmarks, 2IDA converges more rapidly when intermediate upper bounds are used. Recall that these upper bounds are generally faster than the upper bounds generated by a stand-alone CG, because they are obtained by optimizing over a polytope $\mathcal{P}^u$ with far less constraints than $\mathcal{P}$ (Sec. 4.2).

The lower bounds are often faster than the upper bounds, at least for small k values. This is due to the fact that one can often conclude $\text{lb}_{k+1} = \text{lb}_k$ in very short time by lifting $\mathcal{P}_k$ constraints to construct $\mathcal{P}_{k+1}$ (Sec. 4.3).

Since 2IDA switches to classical CG after a number of iterations, the difference between the upper and lower bounding time can become even larger. This difference can be significant, most notably for `vb50-3` and `vb50-5`, where the lower bounding stops at $k = 1$ and classical CG is run immediately because $\lceil \text{lb}_k \rceil + 1 = \lceil \text{ub}_k \rceil$. In these cases, most of the time is actually spent on proving that lb_1 is optimal. The construction of $\mathcal{P}_1$ is not carried out by CG, since the model has only 2 variables and it is faster to directly generate all constraints. This also explains why the number of pricing calls is 0 for `vb50-3` and `vb50-5`.

Table 2 Summarized results of two 2IDA versions: pure 2IDA ("no" in Column 2) and 2IDA with intermediate upper bounds ("yes" in Column 2). Columns 3-5 provide the total CPU time of the lower bounding calculations and of *all* upper bounding calculations (both for Sec. 4.2 and for the classical CG steps run at the end, e.g., at $k = 10$). The last 6 columns present the minimum, average and maximum number of pricing calls in each part of 2IDA.

Instance set	Intermediate upper bounds	Avg. CPU [s]			LB pricing calls			UB pricing calls		
		Total	LB	UB	min	avg	max	min	avg	max
vb20	yes	9161	3162	5999	0	58	158	22	31	59
	no	11958	8029	3929	22	87	150	19	29	49
vb50-1	yes	39827	18620	21207	5	72	120	50	107	212
	no	33719	16556	17163	14	75	129	52	107	149
vb50-2	yes	89432	31087	58345	0	56	133	74	150	210
	no	75928	37987	37941	45	81	196	95	139	188
vb50-3	yes	44068	3745	40323	0	0	0	60	80	94
	no	100279	59671	40608	12	26	57	60	80	94
vb50-4	yes	66498	28142	38356	56	106	148	109	136	179
	no	43011	24907	18104	68	113	165	98	127	167
vb50-5	yes	28490	2910	25580	0	0	0	58	62	72
	no	58700	33095	25606	16	30	45	58	62	72
vbInd	yes	7580	670	6910	0	15	65	4	28	89
	no	3800	1883	1917	5	41	74	0	16	70
m01	yes	818	84	734	0	6	51	55	157	321
	no	1394	628	766	14	27	74	72	149	317
m20	yes	423	135	288	3	11	47	46	117	218
	no	1256	529	728	13	21	51	81	186	448
m35	yes	207	68	139	3	7	17	19	64	176
	no	963	572	393	12	16	30	34	116	305
hard	yes	117908	45575	72333	7	8	10	175	211	278
	no	539086	467083	72004	31	37	45	175	208	278

6.2.1. Comparing the Intermediate Lower Bounds of 2IDA and CG We here compare two series of bounds: (i) the 2IDA lower bounds combining both the optima of $\mathcal{P}_1,\ \mathcal{P}_2,\ \mathcal{P}_3, \ldots$ as well as the intermediate Lagrangian bounds obtained during the CG optimization of these polytopes $\mathcal{P}_1,\ \mathcal{P}_2,\ \mathcal{P}_3, \ldots$; (ii) the Lagrangian bounds produced by a stand-alone CG process. In both cases, we could use the Farley bound, which is a well-known customization of the Lagrangian bound for the case $\mu_a = 1, \forall a \in \mathcal{A}$ (see [19, §. 3.2], [13, §. 2.1] or [2, § 1.2]).

Table 3 reports a comparison constructed by the following protocol. We first run the classical CG on each instance. This produces for each instance a reference computing time T_{CG}. Then, we run for the same instance both CG and "full 2IDA" using a time limit of $p \cdot T_{\mathrm{CG}}$, where $p \in \{5\%, 10\%, 20\%, 30\%, 40\%\}$. For each such p, we note L_p the best lower bound obtained after $p \cdot T_{\mathrm{CG}}$ time. If 2IDA is stopped during step k, L_p is either lb_k or the best Lagrangian dual bound obtained during step k. Table 3 reports the evolution of $L_p/\mathrm{OPT}_{\mathrm{CG}}$, as p goes from 5% to 40%.

Table 3 Comparison of intermediate lower bounds for CG an 2IDA. We report the average ratio $L_p/\mathbf{OPT_{CG}}$ for each instance set for p=5% to p=40% of the time $T_{\mathbf{CG}}$. For example, row vb20 of column "5%" shows that if both methods are run on instances of data set vb20, and stopped after $5\% \cdot T_{\mathbf{CG}}$, CG reports a lower bound of 0.41 $\mathbf{OPT_{CG}}$ and 2IDA reports 0.64 $\mathbf{OPT_{CG}}$.

	Percentage of CG time T_{CG} needed for full convergence									
	5%		10%		20%		30%		40%	
Instance set	CG	2IDA	CG	2IDA	CG	2IDA	CG	2IDA	CG	2IDA
vb20	0.41	0.64	0.55	0.95	0.70	0.99	0.85	0.99	0.94	0.99
vb50-1	0.29	0.88	0.42	0.97	0.65	0.98	0.79	0.98	0.84	0.99
vb50-2	0.49	1.00	0.70	1.00	0.87	1.00	0.97	1.00	0.98	1.00
vb50-3	0.44	1.00	0.57	1.00	0.65	1.00	0.88	1.00	0.98	1.00
vb50-4	0.52	1.00	0.66	1.00	0.76	1.00	0.94	1.00	0.98	1.00
vb50-5	0.52	1.00	0.60	1.00	0.65	1.00	0.79	1.00	0.97	1.00
vbInd	0.52	0.64	0.70	0.87	0.71	0.93	0.82	0.93	0.88	0.99
m01	0.41	0.94	0.51	0.95	0.65	0.97	0.73	0.97	0.77	0.98
m20	0.49	0.74	0.55	0.92	0.66	0.95	0.75	0.98	0.79	0.98
m35	0.64	0.47	0.64	0.80	0.69	0.92	0.73	0.96	0.75	0.98
hard	0.89	1.00	0.89	1.00	0.89	1.00	0.89	1.00	0.94	1.00

We can draw the following conclusions from Table 3:

– Within $5\% \cdot T_{\mathrm{CG}}$ time, 2IDA reaches $\mathrm{OPT_{CG}}$ for all instances vb50-{2,3,4,5}. For the same computing time, the Lagrangian CG lower bound represents approximately $\frac{\mathrm{OPT_{CG}}}{2}$ for instances vb50-{2,3,4,5}.

– Within $20\% \cdot T_{\mathrm{CG}}$ time, 2IDA returns lower bounds in $[0.92 \cdot \mathrm{OPT_{CG}}, \mathrm{OPT_{CG}}]$ for all instance sets. Within the same time limit, the Lagrangian CG bound belongs to $[0.65 \cdot \mathrm{OPT_{CG}},\ 0.89 \cdot \mathrm{OPT_{CG}}]$.

– Within $40\% \cdot T_{\mathrm{CG}}$ time, 2IDA closes the gap almost completely, reporting lower bounds in $[0.98 \cdot \mathrm{OPT_{CG}}, \mathrm{OPT_{CG}}]$.

– There is only one cell of Table 3 in which the classical Lagrangian bound is better: row m35 and column $p = 5\%$. The m35 instances have only weights larger than $\frac{C}{3}$, and so, all columns have only two non-zero coefficients. The first approximations of the dual values are likely to be weak.

6.2.2. Final Convergence of 2IDA and CG on CSP The main purpose of 2IDA is to produce quality lower bounds very rapidly. However, the fact that almost optimal dual bounds are produced in the first iterations hints that the method can help proving optimality in a faster way.

Table 4 compares the CPU time and the number of pricing calls (generated columns) used to converge by either method. 2IDA is faster on 2402 instances out of 3127 (76%), and at least as good in 2544 cases (81%). Note that the most difficult instances are **hard**, for which 2IDA converges 3 times more rapidly. The success of 2IDA on these instances can

Table 4 The computing effort required by 2IDA and CG to fully converge. Columns 2-3 report the average time. Columns "2IDA vs CG (CPU)" report how many times 2IDA (1) needs less time than CG (column $\prec$), (2) needs a similar (difference $< 5\%$) time (column $\simeq$), and (3) needs mores time than CG (column $\succ$). The last two columns compare the average number of generated columns (pricing calls). For 2IDA, this includes both the lower bounding process (Column 7 in Table 2) and the upper bounding process (Column 10 in Table 2). All times and column numbers represent averages over all instances in each set.

	Avg. CPU Time [ms]		2IDA vs CG (CPU)			Avg. nb columns	
Instance set	2IDA	CG	$\prec$	$\simeq$	$\succ$	2IDA	CG
vb20	9161	11653	10	4	11	89	63
vb50-1	39827	50164	14	2	4	179	125
vb50-2	89432	125399	17	0	3	206	223
vb50-3	44068	106356	20	0	0	80	159
vb50-4	66498	87604	19	1	0	242	199
vb50-5	28490	85680	20	0	0	62	160
vbInd	7580	14252	10	1	6	43	51
m01	818	849	549	43	408	161	191
m20	423	620	793	49	158	128	173
m35	207	379	920	42	38	71	134
hard	117908	376112	10	0	0	219	768

be explained by several factors. The first iterations produce a good lower bound, because we exploit the structure of a good dual solution provided by DFFs. Regarding the upper bounds, recall we optimize a polytope $\mathcal{P}^u$ (Sec. 4.2) by only generating $\mathbf{y}^*_\mathbf{k}$*-tight constraints*, inducing a form of implicit stabilization around those good initial solutions.

We finally recall an essential key for the speed of the lower bounding process for large k. The incremental construction of polytope $\mathcal{P}_{k+1}$ from $\mathcal{P}_k$ (Sec. 4.3) can considerably reduce the time needed to optimize $\mathcal{P}_{k+1}$. In several cases, 2IDA computes lb_{k+1} from lb_k in neglectable time (see also instance by instance results at `cedric.cnam.fr/~porumbed/csp/`).

6.3. Experiments on Multiple-Length Cutting Stock

We here consider the ML-CSP problem, as defined in Sec. 6.1, *i.e.*, with three available roll widths (bin sizes) $0.8C$, $0.9C$ and C of costs 0.8, 0.9 and respectively 1.

Table 5 reports the results of a *full 2IDA* run that uses both the intermediate upper bounds from Sec. 4.2 and the split strategy guided by upper bounds (Sec. 4.1.2). All reported times are cumulative, except the one in the last column that corresponds to a stand-alone run of pure CG. The upper bounds can stop the 2IDA sooner; more exactly, since all considered costs μ_a are multiples of 0.1, 2IDA can stop as soon as $\lceil 0.1 \cdot \text{lb}_k \rceil = \lceil 0.1 \cdot \text{ub}_k \rceil$, and so, it can report the ILP optimum $10 \cdot \lceil 0.1 \cdot \text{ub}_k \rceil$ even at $k = 1$ (see figures in bold, or also more results at `cedric.cnam.fr/~porumbed/csp/`).

Table 5 2IDA with intermediate upper bounds and split decisions guided by upper bounds (Sec. 4.1.2) on `multiple length cutting stock`. We only report results on the first instance of each set, but the same trends have been observed on most instances (see `cedric.cnam.fr/~porumbed/csp/` for results on more instances).

Instance	$\sum_{i\in I} b_i \frac{w_i}{C}$	k=1		k=2		k=3		pure CG
		lb $[T_s]$	ub $[T_s]$	lb $[T_s]$	ub $[T_s]$	lb $[T_s]$	ub $[T_s]$	OPT_{CG} $[T_s]$
m01-1	48.62	48.62 [0.02]	54 [0.2]	48.62 [0.2]	54 [0.2]	48.62 [0.3]	54 [0.3]	49 [0.5]
m02-1	55.24	55.24 [0.01]	63.16 [0.1]	55.24 [0.1]	63.16 [0.1]	55.24 [0.1]	63.16 [0.1]	56.27 [0.2]
m03-1	66.49	66.49 [0.01]	84 [0.03]	66.49 [0.04]	84 [0.04]	66.5[0.05]	84 [0.05]	70.2 [0.1]
vb20p01	294.94	294.94 [0.5]	475.5 [3.8]	294.95 [9.1]	399.37 [9.9]	294.97 [10]	373.75 [11]	295.13 [16]
vb50-1p01	935.32	935.47 [1.8]	2262 [4.3]	935.52 [12]	2158 [13]	935.67 [14]	2158 [14]	937.61 [119]
vb50-2p01	730.65	730.65 [3.1]	766.87 [76]	730.65 [90]	766.87 [90]	730.65 [95]	766.87 [96]	736.5 [90]
vb50-3p01	328.94	328.94 [5.4]	**328.99** [114]					328.94 [190]
vb50-4p01	672.74	672.74 [2.8]	708.06 [77]	672.74 [93]	708.06 [93]	672.74 [97]	708.06 [98]	672.74 [206]
vb50-5p01	393.95	393.95 [39]	**393.95** [57]					393.95 [151]
vbInd30p0	89.45	89.45 [0.7]	**89.45** [7.2]					89.45 [15]
hard1	55.39	55.39 [3.2]	74.474 [315]	55.39 [666]	74.474 [667]	55.39 [678]	74.474 [680]	55.39 [872]

In Table 6 below, we turn off the intermediate upper bounds and we report lb_k (with $k \in \{1,2,3\}$) for a *pure 2IDA* method using the basic split method from Sec. 4.1.1. These lower bounds are also compared to the lower bounds reached by CG within similar time. For such low values $k \leq 3$, this *pure 2IDA* bound is very fast, because it only solves highly-aggregated pricing problems with k variables. Even the 2IDA bound for $k = 1$ is far better than the Lagrangian bounds obtained with much higher computing times.

Table 6 Comparison of the first three 2IDA lower bounds with the Lagrangian bounds reported by CG within a similar (slightly larger) time on `multiple length CSP`.

Instance	$\sum_{i\in I} b_i \frac{w_i}{C}$	k=1		k=2		k=3		ML-CSP
		$\text{lb}_{\text{2IDA}}[T_s]$	$\text{lb}_{\text{CG-lagr}}[T_s]$	$\text{lb}_{\text{2IDA}}[T_s]$	$\text{lb}_{\text{CG-lagr}}[T_s]$	$\text{lb}_{\text{2IDA}}[T_s]$	$\text{lb}_{\text{CG-lagr}}[T_s]$	OPT_{CG}
m01-1	48.62	48.62 [0.02]	15.87 [0.02]	48.62 [0.02]	15.87 [0.02]	48.62 [0.04]	20.08 [0.04]	49.00
m02-1	55.24	55.24 [0.02]	40.31 [0.03]	55.24 [0.03]	52.16 [0.04]	55.24 [0.05]	54.89 [0.06]	56.27
m01-3	66.49	66.49 [0.01]	25.87 [0.02]	66.49 [0.03]	29.48 [0.04]	67.297[0.04]	32.47 [0.05]	70.2
vb20-p01	294.94	294.94 [0.5]	122.47 [0.7]	294.94[6.2]	284.90 [6.2]	294.99[6.6]	287.01 [6.8]	295.13
vb50-1p01	935.32	935.47[1.5]	214.06 [2.9]	935.51[3.1]	256.35 [3.6]	935.52[4.4]	258.63 [4.4]	937.61
vb50-2p01	730.65	730.65[3.0]	66.55 [3.1]	730.65[5.2]	145.49 [5.3]	730.67[8.2]	200.75 [8.7]	736.5
vb50-3p01	328.94	328.94[5.4]	73.56 [5.8]	328.94[12]	82.44 [13]	328.94[13]	89.91 [14]	328.94
vb50-4p01	672.74	672.74[2.5]	140.64 [2.7]	672.74[4.8]	199.47 [5.6]	672.74[4.9]	199.47 [5.6]	672.74
vb50-5p01	393.95	393.95[3.9]	115.53 [4.1]	393.95[11]	134.19 [12]	393.95[12]	142.12 [13]	393.95
vbInd30p0	89.45	89.45[0.5]	55.42 [0.7]	89.45[1.1]	55.38 [1.1]	89.45[1.9]	41.57 [1.3]	89.45
hard-1	55.39	55.39[1.0]	40.39 [1.3]	55.39[272]	53.08 [272]	55.39[491]	55.24 [491]	55.39

In both 2IDA versions in Tables 5-6, lower bound at $k = 1$ is often equal to $\sum_{i\in I} b_i \frac{w_i}{C}$ (see Column 2 of both Tables). This is not surprising. The dual solution $y_i = \frac{w_i}{C}$ $(\forall i \in I)$ can be very close to optimal in ML-CSP. While this dual solution is always feasible in both ML-CSP and pure CSP, it can more easily be dominated by other solutions in the case of pure CSP. In ML-CSP, there are numerous no-waste patterns $\mathbf{a}$ with $\mathbf{w}^\top \mathbf{a} \in \{0.8C, 0.9C, C\}$,

which yield many constraints that can be written $\mathbf{y}^\top \mathbf{a} \leq \frac{1}{C}\mathbf{w}^\top \mathbf{a}$. The more numerous these constraints are, the more difficult it is to find dual solutions that dominate that $y_i = \frac{w_i}{C}$, *i.e.*, we get closer to a CSP version in which the cost of any pattern $\mathbf{a}$ is $\frac{1}{C}\mathbf{w}^\top \mathbf{a}$ so that $y_i = \frac{w_i}{C}$ is optimal.

To conclude the ML-CSP experiments, the 2IDA lower bounds for $k \in \{1, 2, 3\}$ are relatively better on ML-CSP than on pure CSP. On the other hand, the fact that the linear dual solution $y_i = \frac{w_i}{C}$ is nearly-optimal reduces the interest in using numerous intervals. As such, the ML-CSP lower bound does not grow very spectacularly when k increases.

6.4. Experiments on Low-Waste Cutting Stock

We here evaluate 2IDA on LW-SCP (Sec. 6.1), using the following maximum waste:

– 0 for the `m01` instances (*i.e.*, $C^- = C^+ = C = 100$). The instances `m20` and `35` were not used because they contain only large items (at least $\frac{20}{100}C$ or respectively $\frac{35}{100}C$), and so, they are often infeasible under such no-waste constraint;

– 2 for all `vb50` instances (*i.e.*, $C^- = 99998$);

– $0.5\%C$ for `vbInd` instances; for this set, we select the only three instances with $n \geq 30$;

– 4, for the `hard` instances (*i.e.*, $C^- = 196$).

Table 7 compares the 2IDA bounds with the Lagrangian CG bounds. We used a pure 2IDA version with no intermediate upper bounds based on the basic group split method from Sec. 4.1.1. The 2IDA bounds clearly outperform the Lagrangian bounds. More exactly, even the first 2IDA bounds for $k = 1$ (Column 2) are usually larger than the Lagrangian bounds reported after 2 or 3 times more computing time (Column 7).

The only exception arises for the `hard` instances, where CG converges more rapidly. Recall (Sec. 6.2.2) that, for the same instances of pure CSP, 2IDA converges more rapidly. The difference comes from the fact that the LW-CSP 2IDA uses neither split methods guided by DFFs nor upper bounds to stop 2IDA earlier. While we could speed-up the convergence by designing LW-CSP-customized split methods, the main goal of the paper is not to present very refined "competition" LW-CSP results, but to describe a rather generic aggregation approach for such resource-constrained problems.

An interesting property of LW-CSP is that it is the only CSP version for which the dual solution $y_i = \frac{w_i}{C}$ is not feasible. More generally, none of the DFFs that proved to be very effective for pure CSP [3] can be used for LW-CSP. To our knowledge, 2IDA is the only method that can produce high-quality bounds so rapidly for such CSP version.

Table 7 Results on `Low-waste CSP` (LW-CSP). Columns 2, 4 and 6 respectively provide the lower bound returned by 2IDA at the end of iterations $k \in \{1,2,3\}$. Columns 3, 5, and 7 respectively report the Lagrangian bound obtained using a similar (slightly larger) CPU time. The two values in the last column indicate the CG optimum for LW-CSP and respectively for pure CSP.

Instance	k=1		k=2		k=3		$\mathrm{OPT}_{\mathrm{CG}}$
	lb_{2IDA} [T_{s}]	$\mathrm{lb}_{\text{CG-lagr}}$[$T_{\mathrm{s}}$]	lb [T_{s}]	$\mathrm{lb}_{\text{CG-lagr}}$[$T_{\mathrm{s}}$]	lb [T_{s}]	$\mathrm{lb}_{\text{CG-lagr}}$[$T_{\mathrm{s}}$]	LW-CSP/CSP
`m01-1`	49 [0.02]	52 [0.03]	53 [0.03]	49 [0.04]	53 [0.06]	35 [0.07]	54/54
`m01-2`	infeasible for current conditions ($C^- = C^+ = C = 100$)						
`m01-3`	50 [0.01]	24 [0.02]	54 [0.03]	47 [0.04]	54 [0.05]	41 [0.06]	55/54
`vb50-1p01`	940*[1.0]	418 [1.1]	940*[1.9]	572 [2.0]	950*[2.5]	571 [2.6]	1219/939
`vb50-1p02`	896 [1.7]	329 [1.8]	898 [3.1]	508 [3.3]	905*[4.3]	537 [4.5]	1000/898
`vb50-1p03`	989*[1.2]	319 [1.2]	989*[2.2]	528 [2.4]	989*[3.1]	473 [3.1]	1200/928
`vb50-2p01`	731 [1.7]	266 [1.9]	731 [2.8]	405 [3.0]	731 [4.4]	585 [4.5]	754/737
`vb50-2p02`	679 [2.1]	337 [2.3]	679 [3.5]	463 [3.5]	679 [5.2]	453 [5.3]	682/679
`vb50-2p03`	560 [2.0]	288 [2.1]	560 [3.5]	327 [3.7]	560 [6.1]	358 [6.2]	560/560
`vb50-3p01`	329 [2.7]	103 [2.7]	329 [5.3]	157 [5.5]	329 [5.7]	170 [5.8]	329/329
`vb50-3p02`	280 [2.8]	104 [2.8]	280 [5.8]	148 [5.8]	280 [11]	139 [11]	280/280
`vb50-3p03`	317 [2.7]	138 [3.0]	317 [6.2]	181 [6.3]	317 [7.1]	173 [7.2]	317/317
`vb50-4p01`	673 [1.4]	282 [1.5]	673 [2.4]	371 [2.5]	673 [2.6]	392 [2.7]	683/673
`vb50-4p02`	640 [1.5]	331 [1.5]	640 [2.5]	378 [2.5]	640 [2.7]	340 [2.7]	659/640
`vb50-4p03`	774 [1.1]	380 [1.2]	774 [1.7]	444 [1.9]	774 [2.0]	559 [2.2]	861/775
`vb50-5p01`	394 [1.9]	216 [2.1]	394 [4.0]	240 [4.0]	394 [4.8]	173 [5.1]	394/394
`vb50-5p02`	408 [1.9]	244 [2.0]	408 [4.8]	193 [4.9]	408 [5.5]	132 [5.6]	408/408
`vb50-5p03`	345 [2.1]	179 [2.3]	345 [4.4]	193 [4.5]	345 [4.9]	182 [4.9]	345/345
`vbInd30p0`	90 [0.3]	21 [0.3]	90 [0.5]	20 [0.6]	90 [0.9]	73 [1.0]	90/90
`vbIndd43p20`	29 [0.02]	15 [0.05]	29 [0.04]	15 [0.05]	29 [0.06]	13 [0.08]	36/29
`vbIndd43p21`	40 [0.09]	24 [0.2]	40 [0.3]	33 [0.3]	40 [1.2]	9 [1.2]	40/40
`hard-1`	56 [0.5]	47 [0.6]	56 [214]	60[162]	60 [296]	60[162]	60/57
`hard-2`	56 [0.5]	47 [0.6]	56 [157]	58[145]	58 [225]	58[145]	58/56
`hard-3`	55 [0.5]	47 [0.6]	55 [175]	59[163]	58 [234]	59[163]	59/55

* Remark that these 2IDA bounds are higher than the CG optimum $\mathrm{OPT}_{\mathrm{CG}}$ for standard CSP (refer to last column).

7. Conclusions and Perspectives

We described an aggregation method for computing dual bounds in column generation more rapidly than using classical Lagrangian bounds. The new method proceeds by constructing an *inner* approximation of the dual polytope and converges towards the optimum of the original model. Several computational techniques are used to incrementally construct a sequence of inner polytopes, by lifting constraints from iteration k to iteration $k+1$. The numerical results show that, besides generating fast dual bounds, the method can often reach the optimum of the original set-covering model more rapidly than classical column generation.

The proposed aggregation relies on a linear correlation between the dual values of the elements and their resource consumption. Several open questions and research directions are possible from here; they could address cases such as: multiple resource constraints

(*e.g.*, as in *vector-packing*); configuration costs determined both by resource data and by external factors (*e.g.*, conflicts between items in *bin-packing with conflicts*, route durations in *capacitated vehicle routing*, distances to clients in *capacitated facility location*, etc.); other set-covering problems in which one can find a correlation between some numerical characteristic of the elements and their dual values, *e.g.*, for sub-problems with more complex constraints solved by a generic ILP solver.

References

[1] P. Benchimol, G. Desaulniers, and J. Desrosiers. Stabilized dynamic constraint aggregation for solving set partitioning problems. European Journal of Operational Research, 2012.

[2] 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.

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

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

[5] J. V. de Carvalho. Using extra dual cuts to accelerate column generation. INFORMS Journal on Computing, 17(2):175–182, 2005.

[6] O. Du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen. Stabilized column generation. Discrete Mathematics, 194:229–237, 1999.

[7] I. Elhallaoui, A. Metrane, F. Soumis, and G. Desaulniers. Multi-phase dynamic constraint aggregation for set partitioning type problems. Mathematical Programming, 123(2):345–370, 2010.

[8] I. Elhallaoui, D. Villeneuve, F. Soumis, and G. Desaulniers. Dynamic aggregation of set-partitioning constraints in column generation. Operations Research, 53(4):632–645, 2005.

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

[10] T. Gschwind and S. Irnich. Dual inequalities for stabilized column generation. Discussion Paper 1407, Gutenberg School of Management and Economics, 2014.

[11] A. N. Letchford and A. Lodi. Primal cutting plane algorithms revisited. Mathematical Methods of Operations Research, 56(1):67–81, 2002.

[12] I. Litvinchev and V. Tsurkov. Aggregation in large-scale optimization, volume 83 of Applied Optimization. Springer, 2003.

[13] M. Lübbecke and J. Desrosiers. Selected topics in column generation. Operations Research, 53:1007–1023, 2005.

[14] G. S. Lueker. Bin packing with items uniformly distributed over intervals $[a, b]$. In 24th Annual Symposium on Foundations of Computer Science, pages 289–297. IEEE, 1983.

[15] R. Macedo, C. Alves, J. V. de Carvalho, F. Clautiaux, and S. Hanafi. Solving the vehicle routing problem with time windows and multiple routes exactly using a pseudo-polynomial model. European Journal of Operational Research, 214(3):536–545, 2011.

[16] R. Marsten, W. Hogan, and J. Blankenship. The BOXSTEP method for large-scale optimization. Operations Research, 23(3):389–405, 1975.

[17] D. Porumbel. Ray projection for optimizing polytopes with prohibitively many constraints in set-covering column generation. *Mathematical Programming*, 2014 (doi 10.1007/s10107-014-0840-7).

[18] R. Sadykov and F. Vanderbeck. Column generation for extended formulations. Electronic Notes in Discrete Mathematics, 37:357–362, 2011.

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

[20] M. V. Vyve and L. A. Wolsey. Approximate extended formulations. Mathematical Programming, 105(2-3):501–522, 2006.

Appendix A: Customizing the Group Split Strategies

While the way group are split has no theoretical impact on our study, it is essential for the practical effectiveness of 2IDA and it deserves further analysis. In Section 4.1, we mentioned:

1. A basic split method (Sec. 4.1.1) that only maintains the regularity of the generated groups. This method has the advantage that it can be applied on absolutely any problem.
2. A splitting strategy (Sec. 4.1.2) that aims at making the aggregated polytope cover a *reference solution* that has a higher quality than the current lower bound.

We here develop this latter strategy in greater detail; we present two methods. The first one (Sec. A.1) uses the (optional) 2IDA upper bound as *reference solution* to guide the splits. The second method (Sec. A.2) is only applicable to CSP and ML-CSP and it takes a dual *reference solution* constructed from dual-feasible functions. These functions are well-acknowledged to provide CSP lower bounds [3, 14] in very short time, *i.e.*, they only require applying a (often piece-wise linear) function on the item weights.

A.1. Guiding the Splits by Upper Bound Solutions

We assume that the elements within each group are sorted by increasing weight. The goal is to determine: (i) a group $j^* \in \{1,\dots,k\}$ to split and (ii) a split point i^* such that the first i^* elements of group j^* are assigned to the first (sub-)group and the other $n_{j^*} - i^*$ elements to the second (sub-group). These decisions rely on a comparison of the current optimal solution $\mathbf{y^*_k}$ of $\mathcal{P}_k$ to an outside reference solution such that $\mathbf{y^*_k} \rightarrow \mathbf{y^u}$ is an improving open direction (see Def. 4.1).

This can be done by computing the difference between $\mathbf{y^*_k}$ and $\mathbf{y^u}$ over segments $[i_1, i_2]$ of each group j. More exactly, considering fixed solutions $\mathbf{y^*_k}$ and $\mathbf{y^u}$, we define the operator $\Delta_j(i_1,i_2) = \sum_{i=i_1}^{i_2} b_i^j \cdot (\mathbf{y^*_k} - \mathbf{y^u})_i^j$, where $(\mathbf{y^*_k} - \mathbf{y^u})_i^j$ is the i^{th} element of vector $\mathbf{y^*_k} - \mathbf{y^u}$ restricted to the elements of group j.

The proposed split method is fully specified by Algorithm 3. We focus on the case $\Delta_j(1,n_j) > 0$, *i.e.*, we consider a global positive "trend" over the whole interval $[1,n_j]$. The negative case $\Delta_j(1,n_j) \leq 0$ is symmetric and can be reduced to the positive one using a simple inversion (Lines 2-4). Algorithm 3 determines the best split point $i^*(j)$ of each group j in Lines 5-12. For this, it first determines two indices i_a and i_b such that there is has a "negative trend" over the segment $[i_a,i_b]$, *i.e.*, by decreasing $\mathbf{y^*_k}$ over $[i_a,i_b]$ and increasing it over $[1,i_a-1]$ and $[i_b+1,n_j]$, $\mathbf{y^*_k}$ could get closer to $\mathbf{y^u}$. Technically, we set $i_a = \min\{i \in \{1,\dots,n_j\} :\ \Delta_j(1,i-1) \geq 0, \Delta_j(i,i) < 0\}$ and $i_b = \max\{i \in \{i_a,\dots,n_j\} :\ \Delta_j(i_a,i) < 0\}$. If $\Delta_j(1,i_a-1) > \Delta_j(i_b+1,n_j)$, we consider that the best split option for j is $[1,n_j] \rightarrow [1,i_a-1],\ [i_a,n_j]$, and so, $i^*(j) = i_a - 1$; otherwise, we choose $i^*(j) = i_b$. The interest in splitting j at $i^*(j)$ is quantified by a heuristic score $h(j)$ initially defined by $\max\big(\Delta_j(1,i_a-1), \Delta_j(i_b+1,n_j)\big)$ as described above.

Furthermore, we multiply this $h(j)$ score by the weight spread $w^j_{\max} - w^j_{\min}$, so as to discourage splitting groups with similar weights (Line 13). We finally multiply by 2 the interest of splitting extremal groups 1 and k (Line 15) because the smallest and the largest items can have a high importance, *e.g.*, in CSP, many small items will have a 0 dual value and large items will have a dual value equal to 1 and this has a strong influence on the the way all other groups are determined.

A.2. Guiding the Splits by Solutions Obtained from Dual-Feasible Functions

On CSP and ML-CSP, one can obtain a high-quality dual solution by taking $y_i = f(w_i),\ \forall i \in I$, where f is a dual feasible function. We recall that $f : [0,C] \rightarrow [0,1]$ is a *dual-feasible function* (DFF) if and only if the following holds: $\sum_{i\in I} a_i w_i \leq C \implies \sum_{i\in I} a_i f(w_i) \leq 1$ for any index set I, and any values $a_i \in \mathbb{Z}^+$ and $w_i > 0$.

All classical DFFs surveyed in [3] have a piece-wise linear form. An example of DFF is the identity function $f(x) = \frac{x}{C}$ which yields dual feasible solution $y_i = \frac{w_i}{C}$. However, most DFFs are given by staircase functions. The DFF-based split method proposed here identifies k intervals of $[0,C]$ before starting the construction of $\mathcal{P}_k$. These k intervals are chosen so that DFF f is linear over each of them. For this value of k, the generated aggregated polytope does include the solution $y_i = f(w_i)$, because f is linear over all groups defined this way. If f yields a dual solution $\mathbf{y}$ that is optimum in $\mathcal{P}$, then $\texttt{DSCvr}(\mathcal{P}_k)$ also contains this $\mathcal{P}$ optimum. If f does not lead to an optimal solution, it may give a good starting point for 2IDA. To be sure we guide the splits using a reasonable reference solution, for each instance we choose from [3] the DFF with maximum 10 intervals that yields the highest objective value.

Algorithm 3: Group Split Routine Guided by an Upper Bound Reference Solution

Data: $\mathbf{y_k^*}$ and $\mathbf{y^u}$

Result: (i) group j^* to split, (ii) the number i^* of elements of the first sub-group

for $j \leftarrow 1$ **to** k **do**
  if $\Delta_j(1, n_j) < 0$ **then**
    $\mathbf{y_k^*} \leftarrow -\mathbf{y_k^*}$ // *simple inversion to reduce the*
    $\mathbf{y^u} \leftarrow -\mathbf{y^u}$ // *negative case to a positive case*
  $i_a \leftarrow \min\{i \in \{1, \ldots, n_j\} : \Delta_j(1, i-1) \geq 0, \Delta_j(i, i) < 0\}$
  $i_b \leftarrow \max\{i \in \{i_a, \ldots, n_j\} : \Delta_j(i_a, i) < 0\}$
  if $\Delta_j(1, i_a - 1) > \Delta_j(i_b + 1, n_j)$ **then**
    $i^*(j) \leftarrow i_a - 1$ // *Obtain intervals* $[1, i_a - 1]$ *and* $[i_a, n_j]$
    $h(j) \leftarrow \Delta_j(1, i_a - 1)$
  else
    $i^*(j) \leftarrow i_b$ // *Obtain intervals* $[1, i_b]$ *and* $[i_b + 1, n_j]$
    $h(j) \leftarrow \Delta_j(i_b + 1, n_j)$
  $h(j) \leftarrow h(j) \cdot (w^j_{\max} - w^j_{\min})$ // *Discourage splitting groups with similar weights*
  if $j = 1$ *or* $j = k$ **then**
    $h(j) \leftarrow h(j) * 2$ // *Encourage splitting extremal groups*
  if $h(j) > h(j^*)$ **then**
    $j^* \leftarrow j$ // *Initially, set* $j^* = 0$ *and* $h(j^*) = -\infty$
return $(j^*, i^*(j^*))$