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 (CG) 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 resulting aggregated model allows to find high-quality lower (dual) bounds more rapidly than standard CG. The speed-up is due to less primal and dual variables in the master, and to an aggregated pricing subproblem. To guarantee optimality, 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 we minimize an objective (cost) function. A well-known drawback of CG is that it may converge rather slowly, i.e., as [2, §1.2] put it, the standard CG can be desperately slow. As such, in the first CG iterations, the classical Lagrangian lower bound is usually quite far from the optimum. 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 technique to reduce the size of large-scale LPs 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 motivation of our work comes from the fact that, more recently, aggregation techniques have proved to be very successful 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. We now present the general main idea. Recall that a set-covering problem is defined on a ground set $I$ and that the CG model uses a dual variable $y_i$ for each $i \in I$. In our resource-constrained context, each $i \in I$ is associated to a resource consumption $w_i$ (e.g., weight) and all feasible subsets of $I$ (columns) are required to consume a total resource amount between some $C^-$ and $C^+$. We exploit the following assumption: all columns selected by the optimum master solution have very tight resource constraints, and so, we expect a correlation between $y_i$ and $w_i$ at optimality ($\forall i \in I$). 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 $a = [a_1 \ldots a_n]^\top$ defined by all extreme solutions of a given subproblem. 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 $a \in \mathcal{A}$ has a cost $\mu_a$ (often depending on the total resource consumption of the elements of $a$). We use primal decision variables $\lambda_a$ to indicate the number of selections of columns $a \in \mathcal{A}$, leading to a well-known classical ILP
\[
\begin{align*}
\min & \sum_{a \in \mathcal{A}} \mu_a \lambda_a \\
\text{s.t.} & \sum_{a \in \mathcal{A}} a_i \lambda_a \geq b_i, \forall i \in \{1, \ldots, n\}, \lambda_a \in \mathbb{Z}_+, \forall a \in \mathcal{A}.
\end{align*}
\]

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

\[
\begin{align*}
\max & \ b^\top y \\
\text{s.t.} & \quad a^\top y \leq \mu_a, \forall a \in \mathcal{A} \\
& \quad y_i \geq 0, \quad i \in \{1, \ldots, n\}
\end{align*}
\]

This is the main dual set-covering (DSCvr) formulation over polytope $\mathcal{P}$. We can write (2.1) more compactly as $\text{DSCvr}(\mathcal{P}) = \max \{ b^\top y : y \in \mathcal{P} \}$. Its optimum value $\text{OPT}(\text{DSCvr}(\mathcal{P}))$ is referred to as the CG optimum and is noted $\text{OPT}_\text{CG}$. To determine $\text{OPT}_\text{CG}$, a CG algorithm dynamically generates a subset of the 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 $\mathcal{P}_\text{outer} \supset \mathcal{P}$ described by the constraints generated so far and find an optimal $y \in \mathcal{P}_\text{outer}$; (ii) pricing subproblem: generate a new valid constraint of $\mathcal{P}$ violated by $y$ and add it to the description of $\mathcal{P}_\text{outer}$—or report $\text{OPT}(\text{DSCvr}(\mathcal{P}_\text{outer})) = \text{OPT}_\text{CG}$ if no such constraint can be found.

The pricing CG subproblem seeks a configuration of most negative reduced cost:

\[
\min_{a \in \mathcal{A}} \mu_a - y^\top a,
\]

where $y \in \mathbb{R}_n^+$ is current dual solution. This pricing subproblem can be defined as follows: select $a_i$ times each $i \in \{1, \ldots, n\}$ so as to maximize the profit $y^\top a$ minus the cost $\mu_a$ under the resource constraints $C^- \leq w^\top a \leq C^+$, where $w = [w_1 \ w_2 \hdots \ w_n]^\top \in \mathbb{Z}_n^+$ 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 algorithm. If $C^+$ is bounded, the pricing subproblem can generally be solved in polynomial time by dynamic programming (DP). For this, we define a profit function $P_{\text{max}}$ that maps any state $(c, i) \in \{0, \ldots, C^+\} \times \{1, \ldots, n\}$ to the maximum value $P_{\text{max}}(c, i)$ of profit $y^\top a_{ci}$ over all configurations $a_{ci}$ that satisfy $w^\top a_{ci} = c$ and that only use elements of $\{1, \ldots, i\}$. Starting with $P_{\text{max}}(0, 0) = 0$, one
can determine $P_{\text{max}}(c, i)$ for all reachable states $(c, i) \in \{C^-, \ldots, C^+\} \times \{1, \ldots, n\}$ using a recursion such as:

$$P_{\text{max}}(c, i) = \max_{r \in \{0, \ldots, b_i\}} \{P_{\text{max}}(c - r \cdot w_i, i - 1) + r \cdot y_i\}$$

(2.3)

In most practical cases, the cost $\mu_a$ only depends on the total resource consumption of $a$, and so, $\mu_a$ can be determined separately as a preprocessing. By slightly abusing notations, we can write $\mu_a = \mu(w^T a)$. The best reduced cost is thus attained in a state $(c^*, n)$ such that:

$$P_{\text{max}}(c^*, n) - \mu(c^*) = \max_{c \in \{C^-, \ldots, C^+\}} P_{\text{max}}(c, n) - \mu(c).$$

Depending on application-specific features, there are at least two widespread generalizations of this DP scheme. 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 $C^+ \times n$ table of DP states can be constructed in $O(nbC^+)$ time, where $nb = \sum_{i=1}^n b_i$ is the number of individualized elements. We use $nb$ instead of $n$: the elements with demand multiplicities $b_i > 1$ can be selected up to $b_i$ times, and so, $r$ can vary from 0 to $b_i$ in (2.3).

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, 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 master covering constraints of all elements of a group are 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 original dual values in the group. When the pricing subproblem is called, the dual variables are disaggregated and the original subproblem is solved. The column produced by the subproblem is added to the restricted master program if it is compatible with the current partition, 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 subproblem (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 explicitly used 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 subproblem has more complex resource constraints, the equivalence of different elements may be less obvious. Such situations 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 \( I = \{1, \ldots, n\} \) into \( k \) groups, we force
the dual values in each group to follow a linear function of their resource consumption. This is implemented by adding new linear equalities to the dual LP. This restricts the dual LP, and so, the optimum of the resulting model is a lower bound for the original problem (fractional or continuous). We will also prove that certain dual constraints (primal columns) are redundant in this new model, so as eventually reduce both the number of dual variables and dual constraints.

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

Let $G_k = \{I^1, I^2, \ldots, I^k\}$ be a partition of $I = \{1, \ldots, n\}$ and note $n_j = |I^j|$. Given elements $I^j$ of a group $j \in \{1, \ldots, k\}$, let $y^j, w^j, b^j$ and $a^j$ be the $n_j$-dimensional column vectors related to dual variables, resource consumptions, demands and, respectively, coefficients of some configuration $a \in A$. Without restricting generality, we consider the elements $I^j$ indexed such that we can write $[y^1 y^2 \ldots y^n] = [(y^1)^\top \ldots (y^k)^\top]$.

The linear restriction is implemented as follows: given any group $j \in \{1, \ldots, k\}$, the values of the dual variables $y^{j_1}, y^{j_2}, \ldots, y^{j_{n_j}}$ are written as a linear function of their resource consumptions $w^{j_1}, w^{j_2}, \ldots, w^{j_{n_j}}$. Formally, we impose that there exists $\alpha^j, \beta^j \in \mathbb{R}$ such that 

$$y^j_i = \alpha^j w^j_i + \beta^j, \quad \forall i \in I^j.$$ 

This restriction requires $2k$ additional variables $\alpha^j, \beta^j$ with $j \in \{1, \ldots, k\}$ that are linked to the $y$ variables in (2.1). We obtain the first version of a restricted dual polytope in $\mathbb{R}_+^n \times \mathbb{R}^{2k}$:

$$
\max \ b^\top y \\
\begin{aligned}
& a^\top y \leq \mu_a, & \forall a \in A \\
y^j_i = w^j_i \alpha^j + \beta^j, & \forall j \in \{1, \ldots, k\}, i \in I^j \\
y^j_i \geq 0, & \forall j \in \{1, \ldots, k\}, i \in I^j \\
\alpha^j, \beta^j \in \mathbb{R}, & \forall j \in \{1, \ldots, k\}
\end{aligned}
$$

$$
(3.1)
$$

PROPOSITION 3.1. The projection $\text{proj}_y(P^{y,\alpha,\beta}_k)$ of $P^{y,\alpha,\beta}_k$ onto the variables $y$ verifies $\text{proj}_y(P^{y,\alpha,\beta}_k) \subseteq P$.

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

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

---

1. Observe we slightly abused notations to simplify the indexes. To lighten the text in such contexts, we prefer to write $i \in I^j$ to mean $i \in \{1,2,\ldots,|I^j|\}$. 
We can reformulate (3.1) using only variables $\alpha^j$ and $\beta^j$. We first rewrite the objective function. For each group $j$, $y^j$ can be written as a linear combination of $w^j$ and $1_j$ (vector $[1\ 1\ \ldots\ 1]^T$ with $n_j$ elements): $y^j = w^j \alpha^j + 1_j \beta^j$.

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

We now address the first constraints of (3.1). We rewrite $a^T w = \sum_{j=1}^k (a^j)^T y^j$ and

$$\left( a^j \right)^T y^j = \left( a^j \right)^T \left( w^j \alpha^j + 1_j \beta^j \right) = \left( (a^j)^T w^j \right) \alpha^j + \left( (a^j)^T 1_j \right) \beta^j \quad (3.3)$$

for each $j \in \{1, \ldots, k\}$. We are now ready to express model (3.1) with variables $\alpha^j$ and $\beta^j$ only. To simplify the writing in (3.3), we use the following notational shortcuts.

**Definition 3.1.** Given a configuration $a \in A$ and a group $j$, we define:

- $c_a^j = (a^j)^T w^j$: total resource consumption of the elements of $I^j$ selected in $a$. Observe that this is the coefficient of variable $\alpha^j$ in (3.3);

- $N_a^j = (a^j)^T 1_j$: total number of elements of $I^j$ selected in $a$. This is the coefficient of variable $\beta^j$ in (3.3);

- $w_{\min}^j = \min_{i \in I^j} w_i^j$ and $w_{\max}^j = \max_{i \in I^j} w_i^j$: extremal resource consumptions in group $j$.

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

$$\begin{array}{l}
\max \sum_{j=1}^k \left( (b^j)^T w^j \right) \alpha^j + \left( (b^j)^T 1_j \right) \beta^j \\
\sum_{j=1}^k c_a^j \alpha^j + N_a^j \beta^j \leq \mu_a, \ \forall a \in A \\
w_{\min}^j \alpha^j + \beta^j \geq 0, \ \forall j \in \{1, \ldots, k\} \\
w_{\max}^j \alpha^j + \beta^j \geq 0, \ \forall j \in \{1, \ldots, k\} \\
\alpha^j, \beta^j \in \mathbb{R}, \ \forall j \in \{1, \ldots, k\}
\end{array} \quad (3.4)$$

**Proposition 3.2.** There is a bijection between the solutions of $P_{k}^{y,\alpha,\beta}$ and $P_{k}^{\alpha,\beta}$:

$$\text{proj}_{\alpha,\beta}(P_{k}^{y,\alpha,\beta}) = P_{k}^{\alpha,\beta}.$$

**Proof:** The first constraint of $P_{k}^{y,\alpha,\beta}$ in (3.1) is equivalent to the first constraint of $P_{k}^{y,\alpha,\beta}$: it is enough to substitute (3.3) in (3.1) and to apply Def. 3.1 to obtain the first constraint of $P_{k}^{y,\alpha,\beta}$. The non-negativity constraint $y_i^j \geq 0$ of $P_{k}^{y,\alpha,\beta}$ is also equivalent to the last two constraints in (3.4): $w_i^j \alpha^j + \beta^j \geq 0, \forall i \in I^j \iff w_{\min}^j \alpha^j + \beta^j \geq 0$ and $w_{\max}^j \alpha^j + \beta^j \geq 0$, as $w_{\min}^j \alpha^j \leq w_i^j \alpha^j \leq w_{\max}^j \alpha^j \ (\forall j \in \{1, \ldots, k\})$. \qed

The new model (3.4) uses less 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

We reduced the number of variables from \( n \) in the initial model (2.1) to \( 2k \) in the last model (3.4). We will show below that certain configurations of \( \mathcal{A} \) become redundant under the new linearity restrictions, so as to also reduce the number constraints.

For any configuration \( a = [a_1 \ldots a_n] \in \mathcal{A} \), 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 \( a_1, \ldots, a_n \). As such, we do not need to express configurations \( a \in \mathcal{A} \) as vectors in \( \mathbb{Z}_+^n \), but as aggregated \( \mathbb{Z}_+^{2k} \) vectors of the form \( \bar{a} = [c_1^a, N_a^1, c_a^2, N_a^2, \ldots, c_a^k, N_a^k] \). Such aggregated configuration \( \bar{a} \) corresponds to all \( a \in \mathcal{A} \) that satisfy \( c_a^j = (a^j)^\top w^j \) and \( N_a^j = (a^j)^\top 1_j, \forall j \in \{1, \ldots, k\} \).

The aggregated model can be optimized only using these aggregated configurations.

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

**Definition 3.3.** Given \( j \in \{1, \ldots, k\} \) and \( c^j \in R^j \), we define:

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

- \( \mathcal{A}_k \): the set of dominant configurations \( a \in \mathcal{A} \) such that \( N_a^j = N^+(j, c_a^j) \) or \( N_a^j = N^-(j, c_a^j) \) for any group \( j \in \{1, \ldots, k\} \). We will later show (Prop. 3.3) that any configuration outside \( \mathcal{A}_k \) generates a weaker constraint than a configuration inside \( \mathcal{A}_k \).

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

\[
\begin{align*}
\max & \sum_{j=1}^k \left( (b^j)^\top w^j \right) \alpha^j + \left( (b^j)^\top 1_j \right) \beta^j \\
\sum_{j=1}^k & c_a^j \alpha^j + N_a^j \beta^j \leq \mu_a, \forall a \in \mathcal{A}_k \\
w_{\min}^j \alpha^j + \beta^j & \geq 0, \forall j \in \{1, \ldots, k\} \\
w_{\max}^j \alpha^j + \beta^j & \geq 0, \forall j \in \{1, \ldots, k\} \\
\alpha^j, \beta^j & \in \mathbb{R}, \forall j \in \{1, \ldots, k\}
\end{align*}
\]

(3.5)

Obviously, if two configurations \( a, a' \in \mathcal{A}_k \) yield \( N_a^j = N_{a'}^j \), \( c_a^j = c_{a'}^j \), \forall j \in \{1, \ldots, k\} \), only one of them has to be explicitly considered. This model can be referred to as the Dual Set-Covering (DSCvr) LP over \( \mathcal{P}_k \) and written \( \text{DSCvr}(\mathcal{P}_k) = \max \{ \sum_{j=1}^k ((b^j)^\top w^j) \alpha^j + ((b^j)^\top 1_j) \beta^j : [\alpha^1 \ldots \alpha^k, \beta^1 \ldots \beta^k]^\top \in \mathcal{P}_k \} \). We will see that \( \mathcal{P}_k = \mathcal{P}_k^{\alpha, \beta} \).

**Proposition 3.3.** Any solution of \( \mathcal{P}_k \) can be written as a valid solution of \( \mathcal{P} \) in (2.1).
**Proof:** Prop. 3.1 states that \( \text{proj}_{\mathcal{P}}(\mathcal{P}_{k}^{\alpha,\beta}) \subseteq \mathcal{P} \). Prop. 3.2 shows that \( \mathcal{P}_{k}^{\alpha,\beta} \) is equivalent to \( \mathcal{P}_{k}^{\alpha,\beta} \). It is enough to show that \( \mathcal{P}_{k}^{\alpha,\beta} = \mathcal{P} \). Recall that above \( \mathcal{P}_{k} \) is constructed from \( \mathcal{P}_{k}^{\alpha,\beta} \) by replacing \( \mathcal{A} \) with \( \mathcal{A}_{k} \) in (3.4). We will show that any configuration \( \mathbf{a} \in \mathcal{A} \setminus \mathcal{A}_{k} \) is dominated by a configuration \( \mathbf{a}' \in \mathcal{A}_{k} \), i.e., \( \mathbf{a} \) generates a weaker constraint than \( \mathbf{a}' \).

Using the notations from Def. 3.3, we observe that \( \mathbf{a} \notin \mathcal{A}_{k} \Rightarrow \exists j \in \{1,\ldots,k\} \) such that \( N^{-}(j,c_{j}^{\alpha}) < N_{a_{j}}^{j} < N^{+}(j,c_{j}^{\beta}) \). As such, \( \mathbf{a} \) generates a constraint in which the \( j^{\text{th}} \) term \( c_{j}^{\alpha} + N^{-}(j,c_{j}^{\alpha}) \beta_{j} \) is sandwiched by \( c_{j}^{\beta} + N^{+}(j,c_{j}^{\beta}) \beta_{j} \) and \( c_{j}^{\alpha} + N^{-}(j,c_{j}^{\alpha}) \beta_{j} \). The constraint of \( \mathbf{a} \) is thus weaker than the constraint generated by some \( \mathbf{a}' \) that does verify \( N_{a'_{j}}^{j} = N^{+}(j,c_{j}^{\alpha}) \) or \( N_{a'_{j}}^{j} = N^{-}(j,c_{j}^{\alpha}) \). By applying this for all groups \( j \) with above properties, we obtain a configuration \( \mathbf{a}' \in \mathcal{A}_{k} \), i.e., satisfying all conditions from Def. 3.3. □

### 3.3. Computational Method: Aggregated CG to Optimize \( \text{DSCvr}(\mathcal{P}_{k}) \)

#### 3.3.1. Preprocessing Stage

To construct the aggregated constraints of \( \mathcal{P}_{k} \) in (3.5), one first needs to compute all cardinality coefficients \( N^{+}(j,c_{j}^{\alpha}) \) and \( N^{-}(j,c_{j}^{\beta}) \) from Def. 3.3 for all \( c_{j} \in \mathbf{R}_{j} \), \( \forall j \in \{1,\ldots,k\} \). This is done in a preprocessing stage, executed only once in advance. We need to find the maximum and the minimum number of elements of \( I_{j} \) that consume a total resource amount of \( c_{j} \) for all \( c_{j} \in \mathbf{R}_{j} \). This can be solved by applying the dynamic programming (DP) scheme from Sec 2.1 with a profit of 1 for each element of \( I_{j} \). More exactly, to determine \( N^{+}(j,c_{j}^{\alpha}) \) and \( N^{-}(j,c_{j}^{\beta}) \) for all \( c_{j} \in \mathbf{R}_{j} \), \( \forall j \in \{1,\ldots,k\} \). This is done in a preprocessing stage, executed only once in advance. We need to find the maximum and the minimum number of elements of \( I_{j} \) that consume a total resource amount of \( c_{j} \) for all \( c_{j} \in \mathbf{R}_{j} \). This can be done by applying the dynamic programming (DP) scheme from Sec 2.1 with a profit of 1 for each element of \( I_{j} \). More exactly, to determine \( N^{+}(j,c_{j}^{\alpha}) \) and \( N^{-}(j,c_{j}^{\beta}) \) for all \( c_{j} \in \mathbf{R}_{j} \), \( \forall j \in \{1,\ldots,k\} \). This DP scheme only discovers reachable states for feasible \( c_{j} \) values, i.e., it constructs the set \( \mathbf{R}_{j} \) at the same time for each group \( j \). Considering all groups \( j \in \{1,\ldots,k\} \) together, the preprocessing has the same complexity as the initial DP algorithm for \( \mathcal{P} \) pricing.

#### 3.3.2. Optimization by Aggregated CG and Aggregated Pricing

For \( k = 1 \), \( \text{DSCvr}(\mathcal{P}_{k}) \) 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 subproblem. This pricing could be solved by disaggregation, i.e., express the current solution \( [\alpha^{1} \ldots \alpha^{k} \beta^{1} \ldots \beta^{k}]^\top \) of (3.5) as a solution \( \mathbf{y} \in \mathbf{R}^{n} \) of (2.1) and apply the classical non-aggregated pricing. This would ignore the fact that the dual values are correlated with the resource consumption in each group.

We now show how the pricing can be solved without disaggregation. Given current solution \( [\alpha^{1} \ldots \alpha^{k} \beta^{1} \ldots \beta^{k}]^\top \) of (3.5), the aggregated version of the subproblem (2.2) is the...
following; find the aggregated configuration \( \mathbf{a} = [c_1^1 N_a^1 c_2^2 N_a^2 \ldots c_k^k N_a^k] \) that maximizes the profit \( \sum_{j=1}^{k} \alpha_j c_j + \beta j N(j, c_j) \) minus the cost \( \mu(\sum_{j=1}^{k} c_j) \). Formally, the aggregated pricing can be expressed in variables \( c^1, c^2, \ldots, c^k \) as follows:

\[
\max \sum_{j=1}^{k} \alpha_j c_j + \beta j N(j, c_j) - \mu(\sum_{j=1}^{k} c_j)
\]

\[
C^- \leq \sum_{j=1}^{k} c_j \leq C^+
\]

\[
N(j, c_j) = \begin{cases} 
N^+(j, c_j) & \text{if } \beta j \geq 0 \\
N^-(j, c_j) & \text{if } \beta j < 0
\end{cases} \forall j \in \{1, \ldots, k\}
\]

\[
c_j \in R^j \forall j \in \{1, \ldots, k\}
\]

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

The decision variables \( c^1, c^2, \ldots, c^k \) are sufficient to represent a solution for (3.6), since all \( N^+ \) and \( N^- \) values are determined from \( c_j \) and \( \beta j \) (in the preprocessing stage). All \( \beta j \) represent input data and 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 \( \beta j \). Finally, let us write \( p^j_c = \alpha_j c + \beta j N(j, c) \) and reformulate (3.6) using binary decision variables \( x^j_c \) such that \( x^j_c = 1 \) if and only if group \( j \in \{1, \ldots, k\} \) uses a total resource amount of \( c \in R^j \).

\[
\max \sum_{j=1}^{k} \sum_{c \in R^j} p^j_c x^j_c - \mu(\sum_{j=1}^{k} \sum_{c \in R^j} c x^j_c)
\]

\[
C^- \leq \sum_{j=1}^{k} \sum_{c \in R^j} c x^j_c \leq C^+
\]

\[
\sum_{c \in R^j} x^j_c = 1 \forall j \in \{1, \ldots, k\}
\]

\[
x^j_c \in \{0, 1\} \forall j \in \{1, \ldots, k\}, c \in R^j
\]

The resulting aggregated pricing (for \( P_k \)) is a multiple-choice variant of the non-aggregated pricing (for \( P \)). The standard dynamic programming (DP) from Sec. 2.1 can be extended to an aggregated DP, e.g., in the same way the standard knapsack DP scheme can be extended to solve multiple-choice knapsack problems. In both DP versions, the costs can be considered pre-defined functions \( \mu : \{0, 1, \ldots, C^+\} \rightarrow \mathbb{R}^+ \) 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 \( n_b \) complexity factor is reduced, once the initial preprocessing (Sec. 3.3.1) is performed.

4. The Convergent Algorithm

Sec. 3.3.2 above presented 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 \( \text{lb}_{G_k} \) (or simply \( \text{lb}_k \)
Algorithm 1: Iterative Inner Dual Approximation

\[ k \leftarrow 1, \quad G_1 \leftarrow \{I\} \quad \text{// initial partition with only one group} \]
\[ A_1 \leftarrow \text{initialConstr()} \quad \text{// all constraints with 2 variables can be generated} \]

repeat
\[ \text{calcCardCoefs}(G_k) \quad \text{// calculate all } N^-, N^+ \text{ values (Sec. 3.3.1)} \]
\[ \text{lb}_k, A_k, y^*_k \leftarrow \text{lbByAggregCG}(A_k) \]
\[ G_{k+1} \leftarrow \text{groupSplit}(G_k) \]
\[ A_{k+1} \leftarrow \text{liftConstr}(A_k, G_{k+1}) \quad \text{// lift constraints from } \mathcal{P}_k \text{ to } \mathcal{P}_{k+1} \text{ (Sec. 4.2)} \]
\[ \text{ub}_k \leftarrow \text{upBound}(A_k, y^*_k) \quad \text{// optional upper bound of OPT}_{CG} \text{ (Sec. 4.3)} \]
\[ k \leftarrow k + 1 \]
until a stopping condition is reached \quad \text{// e.g., } \lceil \text{lb}_{G_k} \rceil = \lceil \text{ub}_{G_k} \rceil \text{ if } \mu_a \in \mathbb{Z}, \forall a \in A \]

when the exact structure of \( G_k \) is not essential) is a lower bound of the sought \( \text{OPT}_{CG} \).

We now describe how this bound can be iteratively improved to compute \( \text{OPT}_{CG} \).

Alg. 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, the convergence is realized when \( \mathcal{P}_k \) becomes equivalent to \( \mathcal{P} \) in the last iteration. We further explain:

- how to split groups to obtain a new partition \( G_{k+1} \) from \( G_k \) (Sec. 4.1);
- how to determine \( \text{lb}_{k+1} \) from \( \text{lb}_k \) without optimizing \( \mathcal{P}_{k+1} \) from scratch (Sec. 4.2);
- how to compute an (optional) upper bound \( \text{ub}_k \) from \( \text{lb}_k \) (Sec. 4.3);

4.1. Split Methods

2IDA eventually converges towards \( \text{OPT}_{CG} \) 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 first discuss in Sec. 4.1.1 a simple-but-general method that uses no information on the structure of \( \mathcal{P}_k \). We continue in Sec. 4.1.2 with a more refined split strategy that is guided by an estimated 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 \( \frac{1}{2} C^+, \frac{1}{4} C^+, \frac{1}{8} C^+, \frac{1}{16} C^+ \), etc. The very first split operation takes the segment \([0, C^+]\) of the unique group
4.1.2. Split Methods that Make $P_k$ Evolve Towards Better Solutions

We now consider we are given some $y^u \in \mathbb{R}^n$ that dominates the best solution $y^*_k$ of $P_k$ (i.e., $b^T y^u > b^T y^*_k$). The goal is to break linearity restrictions that are incompatible with $y^u$, so as to make $P_k$ evolve to a polytope $P_{k+1}$ that does contain solutions closer to $y^u$.

**Definition 4.1.** Given a solution $y \in P_k$ and some $y^u \in \mathbb{R}_+^n$, we say that direction $y \to y^u$ is an open direction if there exists $\epsilon > 0$ such that $y_\epsilon = y + \epsilon(y^u - y) \in P$. If in addition $b^T y_\epsilon > b^T y$, we say that $y \to y^u$ is an improving open direction.

If $y^*_k \to y^u$ is an improving open direction, then all $y^*_k + \epsilon(y^u - y)$ are excluded from the current $P_k$ only because $y^u$ is non-linear over some $j \in \{1, \ldots, k\}$. We choose to split in two a group $j^*$ for which $y^u - y^*_k$ can be “well approximated” by a 2-part linear function. By splitting $j^*$ accordingly into two parts, $y^*_k$ can evolve into some $y^*_{k+1}$ that is linear over both parts. The calculations that determine this split decision are further discussed in App. A (Alg. 3), but they are not essential for the theoretical description of 2IDA. For now, the above ideas are enough to present all other 2IDA components.

We still need to say how to find such $y^u$. We propose two methods: (i) if an upper bound solution is available, use it as $y^u$ (see App. A.1), or (ii) take $y^u$ as any other high-quality solution generated by problem-specific methods (see App. A.2 for cutting-stock).

4.2. From $lb_k$ to $lb_{k+1}$ in Two Steps: Lift $P_k$ to $\mathbb{R}^{2k+2}$ and Fully Optimize $\text{DSCvr}(P_{k+1})$

After solving $\text{DSCvr}(P_k)$ 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 $P_{k+1}$ has to be optimized. We use the columns already generated at iteration $k$ to warm-start the CG at iteration $k+1$, i.e., the standard aggregated CG from Sec. 3.3.2 is not run from scratch at each iteration. First, the $P_k$ constraints generated so far are lifted from dimension $\mathbb{R}^{2k}$ to dimension $\mathbb{R}^{2k+2}$. The optimal solution of $\text{DSCvr}(P_k)$ 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 $\alpha_j$ and $\beta_j$ for all $j \in \{1, \ldots, k\} \setminus \{k^*\}$.
Let us note $A_k^{\text{tight}}$ the set of $y_k^*$-tight constraints of $P_k$ generated while optimizing $\text{DSCvr}(P_k)$. Our approach first constructs a polytope $P'_{k+1}$ only by lifting such $P_k$ constraints: we restrict set $A_{k+1}$ (recall Def 3.3, Sec. 3.2) to $A_{k+1}^{\text{tight}} = \{a' \in A_{k+1} : \exists a \in A_k^{\text{tight}}, \text{ s. t. } c_a^j = c_{a'}^j, N_a^j = N_{a'}^j \forall j \neq j^*, j \in \{1, \ldots, k\}\}$. We optimize $P'_{k+1}$ by CG, solving the same pricing as for $P_{k+1}$ but over $A_{k+1}'$ instead of $A_{k+1}$.

For this, we can use the multi-choice aggregated pricing Sec. 3.3.2 with only two decision levels $j_1$ and $j_2$—all other column terms are taken from the lifted constraint. This leads to a much faster algorithm, applied once for each lifted constraint (Step 1 in Alg. 2 below).

<table>
<thead>
<tr>
<th>Algorithm 2: Two-Step CG for $\text{DSCvr}(P_{k+1})$: optimize over $P'<em>{k+1} \supset P_k$ then over $P</em>{k+1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data:</strong> Optimal solution $[\alpha^1 \ldots \alpha^k \beta^1 \ldots \beta^k]$ of $P_k$, existing constraints $A_k^{\text{tight}}$</td>
</tr>
<tr>
<td><strong>Result:</strong> $\text{lb}<em>{k+1} = \text{OPT}(\text{DSCvr}(P</em>{k+1}))$</td>
</tr>
<tr>
<td>Lift aggregated solution $[\alpha^1 \ldots \alpha^k \beta^1 \ldots \beta^k] \in P_k$ to the space of $P_{k+1}$;</td>
</tr>
<tr>
<td>$\alpha^{j_1}, \alpha^{j_2} \leftarrow \alpha^<em>$ and $\beta^{j_1}, \beta^{j_2} \leftarrow \beta^</em>$ // break $j^*$ into (unlinked) groups $j_1$, $j_2$;</td>
</tr>
<tr>
<td>keep unchanged the values $\alpha^j$ and $\beta^j$ for all $j \neq j^*$;</td>
</tr>
<tr>
<td>repeat</td>
</tr>
<tr>
<td>for $a \in A_k^{\text{tight}}$ do</td>
</tr>
<tr>
<td>- 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 $C_- = \sum_{1 \leq j \leq k \atop j \neq j^<em>} c_{a}^j$ and $C_+ = \sum_{1 \leq j \leq k \atop j \neq j^</em>} c_{a}^j$ to lift $a$ to some $a' \in A_{k+1}$;</td>
</tr>
<tr>
<td>- $A'<em>{k+1} \leftarrow A'</em>{k+1} \cup {a'}$;</td>
</tr>
<tr>
<td>- optimize over current $P'<em>{k+1}$ described by configurations $A'</em>{k+1}$ only</td>
</tr>
<tr>
<td>- update $\text{OPT}(\text{DSCvr}(P'_{k+1}))$ and the current dual solution $[\alpha, \beta]$</td>
</tr>
<tr>
<td>until no configuration $a'$ of negative reduced cost can be found;</td>
</tr>
<tr>
<td>if $\text{OPT}(\text{DSCvr}(P'_{k+1})) = \text{lb}_k$ return $\text{lb}_k$;</td>
</tr>
<tr>
<td>repeat</td>
</tr>
<tr>
<td>- 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 $a$;</td>
</tr>
<tr>
<td>- $A_{k+1} \leftarrow A_{k+1} \cup {a}$;</td>
</tr>
<tr>
<td>- optimize current $P_{k+1}$ described by above $A_{k+1}$ and update $[\alpha, \beta]$;</td>
</tr>
<tr>
<td>until no configuration $a$ of negative reduced cost can be found;</td>
</tr>
<tr>
<td>return $\text{OPT}(\text{DSCvr}(P_{k+1}))$;</td>
</tr>
<tr>
<td>Step 1: lift $P_k$ to $P'_{k+1} \supset P_k$</td>
</tr>
</tbody>
</table>
| Step 2: standard $P_{k+1}$ optim
After determining $\text{OPT(DSCvr}(P'_{k+1}))$ this way, we turn to the original pricing for $P_{k+1}$. Since $A'_{k+1} \subset A_{k+1}$, we have $P'_{k+1} \supset P_{k+1}$, and so, $\text{OPT(DSCvr}(P'_{k+1})) \geq \text{OPT(DSCvr}(P_{k+1})) \geq \text{OPT(DSCvr}(P_{k}))$. If $\text{OPT(DSCvr}(P'_{k+1})) = \text{OPT(DSCvr}(P_{k}))$, we can directly state $lb_{k+1} = lb_k$ only using lifted constraints. Experiments suggest that $lb_{k+1}$ can sometimes be computed this way in almost neglectable time.

To summarize, 2IDA actually optimizes $\text{DSCvr}(P_{k+1})$ in two steps: (1) it first lifts $P_k$ constraints to (rapidly) optimize $P'_{k+1} \supset P_{k+1}$ and (2) it fully optimizes $P_{k+1}$ using the standard aggregated CG from Sec. 3.3.2. The complete algorithmic template of the $\text{DSCvr}(P_{k+1})$ optimization is presented in Alg. 2.

4.3. An Upper Bound Generated From the $P_k$ Optimum

We finish the 2IDA description with the optional upper bounding routine. It takes $y_k^*$ (optimal solution of $\text{DSCvr}(P_k)$ expressed in dimension $\mathbb{R}^n_+$) as input and returns an upper bound solution $y_u$ such that:

(a) if $b^\top y_k^* < \text{OPT}_{CG}$, then the direction $y_k^* \rightarrow y_u$ is open (see Def. 4.1);
(b) if $b^\top y_k^* = \text{OPT}_{CG}$, then $b^\top y_u = \text{OPT}_{CG}$.

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 $b^\top y_k^* = \text{OPT}_{CG}$.

Since $y_k^* \in P$ (recall Prop. 3.3), exists $a \in A$ such that $a^\top y_k^* > \mu_a$. On the other hand, $P$ does contain constraints such that $a^\top y_k^* = \mu_a$: if there were no such constraints, a solution $y_k^* + \epsilon[1\ldots1]^\top$ could be feasible in $P_k$ and better than $y_k^*$. We can thus define a polytope $P_u \supset P$ delimited only by $y_k^*$-tight constraints:

$$P_u = \{y \in \mathbb{R}^n_+ : a^\top y \leq \mu_a, \forall a \in A \text{ such that } a^\top y_k^* = \mu_a\}$$

**Proposition 4.1.** If $y_u$ is an optimal solution of polytope $P_u$ (over objective function $b^\top y$), $y_u$ satisfies the following:

1. upper bounding: $\text{OPT}_{CG} \leq b^\top y_u$;
2. open direction: if $y_k^*$ is not an optimal solution of $\text{DSCvr}(P)$, then $b^\top y_k^* < b^\top y_u$ and $y_k^* \rightarrow y_u$ is an improving open direction;
3. optimality proving: if $b^\top y_k^* = \text{OPT}_{CG}$, then $y_u$ also satisfies $b^\top y_u = \text{OPT}_{CG}$.

**Proof:** Property (1) actually follows from the $P_u$ definition (4.1). Since $P_u$ is constructed from a subset of the constraints of $P$, we directly have $P_u \supset P$, and so, $b^\top y_u \geq \text{OPT}_{CG}$. 
We now prove (2). First, the non-optimality of \( y_k^* \) directly shows that \( b^Ty_k^* < \text{OPT}_{CG} \leq b^Ty^u \). To prove that the direction \( y_k^* \rightarrow y^u \) is open, let us suppose the opposite: there exists an arbitrarily small \( \epsilon > 0 \) such that \( y_k^* + \epsilon(y^u - y_k^*) \notin \mathcal{P} \). As such, there is some \( a \in \mathcal{A} \) for which \( a^Ty_k^* \leq \mu_a \) and \( a^Ty_k^* + ca^T(y^u - y_k^*) > \mu_a \). This would imply that \( a^Ty_k^* = \mu_a \) and \( ca^T(y^u - y_k^*) > 0 \), and so, \( a^Ty^u > a^Ty_k^* = \mu_a \), i.e., \( y^u \) would violate the \( y_k^* \)-tight constraint \( a^Ty \leq \mu_a \). This is impossible, because \( y^u \in \mathcal{P}^u \) satisfies all \( y_k^* \)-tight constraints in (4.1).

We now prove (3). When \( y_k^* \) is optimal in \( \mathcal{P} \), any solution better than \( y_k^* \) would be cut off by a \( y_k^* \)-tight constraint in \( \mathcal{P} \), and so would be in \( \mathcal{P}^u \).

\( \Box \)

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 \( y_k^* \)-tight) constraints. The separation (pricing) problem for \( \mathcal{P}^u \) requires finding a configuration \( a \in \mathcal{A} \) with \( a^Ty_k^* = \mu_a \) that maximizes \( y^Ta \) for the current dual solution \( y \). Such separation problems are not new in general linear programming [11, § 2.3], but we propose a new solution method specific to CG applications. This solution method relies on the dynamic programming (DP) ideas from Sec. 2.1, but it uses a modified lexicographic objective: first maximize the \( y_k^* \)-profit \( (y_k^*)^Ta \), and, subject to this, maximize the \( y \)-profit \( y^Ta \).

In the original DP recursion (2.3), \( P_{\text{max}}(c, i) \) represents the maximum \( y \)-profit \( y^Ta_{c,i} \) over all \( a_{c,i} \in \mathcal{A} \) that satisfy \( w^Ta_{c,i} = c \) and that only use elements of \( \{1, \ldots, i\} \). We now replace \( P_{\text{max}} \) with a function \( P_{\text{max}}^{y_k^*y} \) that maximizes \( M \cdot (y_k^*)^Ta_{c,i} + y^Ta_{c,i} \) over the same \( a_{c,i} \), using a value of \( M \) sufficiently large to make any \( y_k^* \)-profit value larger than any \( y \)-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 \( P_{\text{max}} \) to \( P_{\text{max}}^{y_k^*y} \), but we use the same DP routine and the same states. 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 show that this new profit function \( P_{\text{max}}^{y_k^*y} \) does lead the DP pricing from Sec. 2.1 to \( y_k^* \)-tight constraints.

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

\( \text{Proof:} \) As explained when we defined \( \mathcal{P}^u \) in (4.1), configurations \( a \in \mathcal{A} \) such that \( a^Ty_k^* = \mu_a \) do exist. We only need to prove these \( y_k^* \)-tight constraints can be effectively found by DP.

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

5. Theoretical Property of the Linear Aggregation

Up to now, we have described 2IDA on the dual formulation (2.1) of the master LP:

$$\begin{array}{ll}
\min & \sum_{a \in A} \mu_a \lambda_a \\
\text{s.t.} & \sum_{a \in A} a_i \lambda_a \geq b_i, \forall i \in \{1, \ldots, n\}, \lambda_a \in \mathbb{R}_+, \forall a \in A
\end{array}$$

At each step $k$, the proposed aggregation restricts the dual LP and relaxes the master LP. Indeed, by aggregating dual variables, one implicitly allows the use of some non-valid primal columns in this master LP. More exactly, all aggregated models can implicitly use linear combinations of valid columns and non-valid exchange vectors resulting from the new linearity restrictions.

Recall that an exchange vector is any vector $e \in \mathbb{R}^n$ such that if $a$ is a feasible column, then one can consider that any $a + \phi e \geq 0$ is also a feasible column ($\forall \phi \in \mathbb{R}$ such that $a + \phi e \geq 0$). We investigate below the linearity-based exchange vectors resulting from our aggregation.

By considering the linear restriction $y^j_i = \alpha^j w^j_i + \beta^j$, $\forall i \in I^j$, we observe that $y^j_i$ can also be expressed without terms $\alpha^j$ and $\beta^j$, as a linear function of any two variables among $y^j_1, y^j_2, \ldots, y^j_n$; let us chose $y^j_{\text{min}}$ and $y^j_{\text{max}}$, corresponding to the elements of lowest and (respectively) largest resource consumption in $I^j$. Given a column involving $y^j_i$, one can replace $y^j_i$ with a linear combination of $y^j_{\text{min}}$ and $y^j_{\text{max}}$ and obtain an artificial column that is valid in the aggregated model but not necessarily in the original one. Such replacement (exchange) process can be formalized using a linearity-based exchange vector.

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

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

**Proof:** Since $a$ is feasible, it does satisfy $C^- \leq a^\top w \leq C^+$. Thus, it is enough to show that $a$ and $\hat{a}$ 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 $y^j_{\text{min}}$ and $y^j_{\text{max}}$ are (respectively) the first and the last element of $I^j$. We examine how other elements $y^j_i$ of $y^j$ can be written as a combination of $y^j_{\text{min}}$ and $y^j_{\text{max}}$.

First, we use a classical substitution to determine $\alpha^j = \frac{y^j_{\text{max}} - y^j_{\text{min}}}{w^j_{\text{max}} - w^j_{\text{min}}} \text{ and } \beta^j = \frac{w^j_{\text{min}} y^j_{\text{max}} - w^j_{\text{max}} y^j_{\text{min}}}{w^j_{\text{min}} - w^j_{\text{max}}}.$
Revisiting the change process $a + \psi e \rightarrow \hat{a}$ (restricted to group $j$) can be written as follows:

$$
\begin{bmatrix}
\alpha_i \min \\
\vdots \\
\alpha_i \\
\vdots \\
\alpha_i \max
\end{bmatrix} + \psi
\begin{bmatrix}
w_j \max - w_j \min \\
w_j \max - w_j \min \\
w_j \max - w_j \min \\
\vdots \\
w_j \max - w_j \min
\end{bmatrix}
\rightarrow
\begin{bmatrix}
\alpha_i \min + \psi w_j \max - w_j \min \\
\vdots \\
\alpha_i \max + \psi w_j \max - w_j \min
\end{bmatrix},
$$

(5.1)

where $\psi$ can be positive or negative, which may respectively decrease or increase the coefficient of $a_i$ in the resulting artificial column $\hat{a}$. Observe that the magnitude of $\psi$ needs to be limited (e.g., $\psi \leq a_i$) to keep $a + \psi e \geq 0$; however, the theorem holds for any $\psi \in \mathbb{R}$.

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

$$(a_i \min + \frac{w_i \max - w_i \min}{w_i \max - w_i \min} \psi) w_i \min + (a_i - \psi) w_i \max + (a_i \max + \frac{w_i \max - w_i \min}{w_i \max - w_i \min} \psi) w_i \max$$

This simplifies to $a_i \min w_i \min + a_i \max w_i \max$, which means that the resource consumption is the same in $a$ and $a + \psi e$, i.e., the initial resource consumption can not change by applying linearity-based exchange vectors as in (5.1).

Let us compare the linear aggregation with a simpler aggregation that only imposes equality constraints, e.g., consider fixing $a_i = 0, \forall j \in \{1, \ldots, k\}$. 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 $a$. This would lead to a classical (equality-based) exchange vector with two non-zero elements at positions $i$ and $j$. The associated exchange process would be:

$$
[\ldots a_i \ldots a_j \ldots] \top + \psi \cdot [\ldots 1 \ldots -1 \ldots] \top \rightarrow [\ldots a_i + \psi \ldots a_j - \psi \ldots] \top, \ \forall \psi \in [-a_i, a_j]
$$

(5.2)

**Proposition 5.2.** Columns produced by equality-based exchange vectors in (5.2) can violate the resource constraints.

**Proof:** It is sufficient to give an example of an artificial column that violates the resource constraints. 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 (5.2), 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 \(\psi = -1\).

\(\square\)

6. **Numerical Evaluation**

We performed an experimental analysis on three cutting-stock problems that cover all features of the general dual LP (2.1): different costs \(\mu_a\) of configurations \(a \in A\), 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. A feasible solution consists of a set of valid cutting patterns (configurations), i.e., subsets of \(I\) 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 \(a \in A\), 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 \(\mu_a\) of a pattern \(a \in A\) is given by the cost of the smallest bin (roll) that can hold a total length (width) of \(w^\top a\), e.g., if \(w^\top a = 0.85C\), 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 \(\mu_a = 1\), \(\forall a \in A\). 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.

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.
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 values and the item weights.

<table>
<thead>
<tr>
<th>Name</th>
<th>n</th>
<th>C</th>
<th>avg. b</th>
<th>avg. w span</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vb20(^a)</td>
<td>20</td>
<td>10000</td>
<td>[10,100]</td>
<td>[1, (\frac{1}{2}C)]</td>
<td>25 random instances [19]</td>
</tr>
<tr>
<td>vb50-c1(^a)</td>
<td>50</td>
<td>10000</td>
<td>[50,100]</td>
<td>[1, (\frac{3}{4}C)]</td>
<td>20 random instances [19]</td>
</tr>
<tr>
<td>vb50-c2(^a)</td>
<td>50</td>
<td>10000</td>
<td>[50,100]</td>
<td>[1, (\frac{1}{2}C)]</td>
<td>20 random instances [19]</td>
</tr>
<tr>
<td>vb50-c3(^a)</td>
<td>50</td>
<td>10000</td>
<td>[50,100]</td>
<td>[1, (\frac{1}{4}C)]</td>
<td>20 random instances [19]</td>
</tr>
<tr>
<td>vb50-c4(^a)</td>
<td>50</td>
<td>10000</td>
<td>[50,100]</td>
<td>[(\frac{1}{10}C, \frac{3}{4}C)]</td>
<td>20 random instances [19]</td>
</tr>
<tr>
<td>vb50-c5(^a)</td>
<td>50</td>
<td>10000</td>
<td>[50,100]</td>
<td>[(\frac{1}{10}C, \frac{1}{4}C)]</td>
<td>20 random instances [19]</td>
</tr>
<tr>
<td>vbInd(^a)</td>
<td>[5,43]</td>
<td>[4096, 200000]</td>
<td>[1,300]</td>
<td>[1, (\frac{1}{4}C)]</td>
<td>industrial instances [19]</td>
</tr>
<tr>
<td>m01</td>
<td>100</td>
<td>100</td>
<td>1</td>
<td>[1, (C)]</td>
<td>1000 random instances [3];</td>
</tr>
<tr>
<td>m20</td>
<td>100</td>
<td>100</td>
<td>1</td>
<td>[(\frac{20}{100}C, C)]</td>
<td>1000 random instances [3];</td>
</tr>
<tr>
<td>m35</td>
<td>100</td>
<td>100</td>
<td>1</td>
<td>[(\frac{35}{100}C, C)]</td>
<td>1000 random instances [3];</td>
</tr>
<tr>
<td>Hard</td>
<td>(\approx 200)</td>
<td>100000</td>
<td>1−3</td>
<td>[(\frac{20}{100}C, \frac{35}{100}C)]</td>
<td>instances known to be very hard</td>
</tr>
</tbody>
</table>

\(^a\)These first 7 instance sets correspond (respectively) to the following sets of files CSTR20b50\(c[1-5]^{*}\), CSTR50b50\(c[1-5]^{*}\), 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 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 \(P_k\) (Sec. 3.3.2) and for the pricing needed to calculate optional upper bounds (Sec. 4.3). We used a 2.27GHz (Intel Core i3) CPU, with the gnu g++ compiler on Linux. The master LPs are solved using Cplex 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. 4.3. When such upper bounds are available, they can be very useful to guide split decisions (Sec. 4.1.2) or to prove optimality for low \(k\) values, i.e., as soon as \([lb_k] = [ub_k]\).

However, the convergence can also be ensured by launching a pure CG process in the end, i.e., we turn to classical CG when \([lb_k] + 1 = [ub_k]\) or if \(k = 10\). This setting was de-
2IIDA does not take very fortunate split decisions before $k$ becomes 10, this pure CG process run in the end ensures a faster convergence with the current implementation.

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

<table>
<thead>
<tr>
<th>Instance</th>
<th>Intermediate upper bounds</th>
<th>Total Avg. CPU [s]</th>
<th>LB pricing calls</th>
<th>UB pricing calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>vb20</td>
<td>yes</td>
<td>9.16 3.16 6.00</td>
<td>0 58 158</td>
<td>22 31 59</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>11.96 8.03 3.93</td>
<td>22 87 150</td>
<td>19 29 49</td>
</tr>
<tr>
<td>vb50-1</td>
<td>yes</td>
<td>39.8 18.6 21.2</td>
<td>5 72 120</td>
<td>50 107 212</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>33.7 16.6 17.2</td>
<td>14 75 129</td>
<td>52 107 149</td>
</tr>
<tr>
<td>vb50-2</td>
<td>yes</td>
<td>89.4 31.1 58.3</td>
<td>0 56 133</td>
<td>74 150 210</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>75.9 38.0 37.9</td>
<td>45 81 196</td>
<td>95 139 188</td>
</tr>
<tr>
<td>vb50-3</td>
<td>yes</td>
<td>44.1 3.8 40.3</td>
<td>0 0 0</td>
<td>60 80 94</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>100.3 59.7 40.6</td>
<td>12 26 57</td>
<td>60 80 94</td>
</tr>
<tr>
<td>vb50-4</td>
<td>yes</td>
<td>66.5 28.1 38.4</td>
<td>56 106 148</td>
<td>109 136 179</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>43.0 24.9 18.1</td>
<td>68 113 165</td>
<td>98 127 167</td>
</tr>
<tr>
<td>vb50-5</td>
<td>yes</td>
<td>28.5 2.9 25.6</td>
<td>0 0 0</td>
<td>58 62 72</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>58.7 33.1 25.6</td>
<td>16 30 45</td>
<td>58 62 72</td>
</tr>
<tr>
<td>vbInd</td>
<td>yes</td>
<td>7.58 0.67 6.91</td>
<td>0 15 65</td>
<td>4 28 89</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>3.80 1.88 1.92</td>
<td>5 41 74</td>
<td>0 16 70</td>
</tr>
<tr>
<td>m01</td>
<td>yes</td>
<td>0.81 0.08 0.73</td>
<td>0 6 51</td>
<td>55 157 321</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>1.39 0.63 0.77</td>
<td>14 27 74</td>
<td>72 149 317</td>
</tr>
<tr>
<td>m20</td>
<td>yes</td>
<td>0.42 0.14 0.29</td>
<td>3 11 47</td>
<td>46 117 218</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>1.26 0.53 0.73</td>
<td>13 21 51</td>
<td>81 186 448</td>
</tr>
<tr>
<td>m35</td>
<td>yes</td>
<td>0.21 0.07 0.14</td>
<td>3 7 17</td>
<td>19 64 176</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>0.96 0.57 0.39</td>
<td>12 16 30</td>
<td>34 116 305</td>
</tr>
<tr>
<td>hard</td>
<td>yes</td>
<td>117.9 45.6 72.3</td>
<td>7 8 10</td>
<td>175 211 278</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>539.1 467.1 72.0</td>
<td>31 37 45</td>
<td>175 208 278</td>
</tr>
</tbody>
</table>

Table 2 presents the computing effort (CPU time and subproblems calls) of two 2IIDA versions: a full 2IIDA 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 by the pure CG process launched in the end.

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 of a stand-alone pure CG, because they are obtained by optimizing over a polytope \( P^u \) with far less constraints than \( P \) (Sec. 4.3).

2IDA computes the lower bounds more rapidly than the upper bounds, at least for small \( k \) values. This is due to the fact that it can often conclude \( lb_{k+1} = lb_k \) in very short time by lifting \( P_k \) constraints to construct \( P_{k+1} \) (Sec. 4.2).

The fact that 2IDA can switch to pure CG in the end further increases the difference between the upper and lower bounding time. This difference can be very visible for instances \( vb50-3 \) and \( vb50-5 \), where the lower bounding stops at \( k = 1 \) and the final CG process is immediately launched because \( \lceil lb_1 \rceil + 1 = \lceil ub_1 \rceil \). In these cases, most of the time is actually spent on proving that \( lb_1 \) is optimal. The construction of \( P_1 \) 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 \( vb50-5 \).

6.2.1. The evolution of the 2IDA and CG Lower Bounds Along the Search We here compare two series of bounds: (i) the 2IDA lower bounds combining both the optima of \( P_1, P_2, P_3, \ldots \) as well as the intermediate Lagrangian bounds obtained during the CG optimization of these polytopes \( P_1, P_2, P_3, \ldots \); (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 \( \mu_a = 1, \forall a \in A \) (see [19, \S. 3.2], [13, \S. 2.1] or [2, \S. 1.2]).

Table 3 reports a comparison generated 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 \( p \cdot T_{CG} \), where \( p \in \{5\%, 10\%, 20\%, 30\%, 40\% \} \). For each such \( p \), we note \( L_p \) the best lower bound obtained after \( p \cdot T_{CG} \) 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 \( L_p/OPT_{CG} \), as \( p \) goes from 5\% to 40\%. 
Table 3 Comparison of intermediate lower bounds for CG an 2IDA. We report the average ratio $L_p/OPT_{CG}$ for each instance set for $p=5\%$ to $p=40\%$ of the time $T_{CG}$. For example, row $vb20$ of column “5\%” shows that if both methods are run on instances of data set $vb20$, and stopped after $5\% \cdot T_{CG}$, CG reports a lower bound of 0.41 $OPT_{CG}$ and 2IDA reports 0.64 $OPT_{CG}$.

<table>
<thead>
<tr>
<th>Instance set</th>
<th>Percentage of CG time $T_{CG}$ needed to fully converge</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5%</td>
</tr>
<tr>
<td></td>
<td>CG 2IDA</td>
</tr>
<tr>
<td>$vb20$</td>
<td>0.41 0.64</td>
</tr>
<tr>
<td>$vb50-1$</td>
<td>0.29 0.88</td>
</tr>
<tr>
<td>$vb50-2$</td>
<td>0.49 1.00</td>
</tr>
<tr>
<td>$vb50-3$</td>
<td>0.44 1.00</td>
</tr>
<tr>
<td>$vb50-4$</td>
<td>0.52 1.00</td>
</tr>
<tr>
<td>$vb50-5$</td>
<td>0.52 1.00</td>
</tr>
<tr>
<td>$vbInd$</td>
<td>0.52 0.64</td>
</tr>
<tr>
<td>$m01$</td>
<td>0.41 0.94</td>
</tr>
<tr>
<td>$m20$</td>
<td>0.49 0.74</td>
</tr>
<tr>
<td>$m35$</td>
<td>0.64 0.47</td>
</tr>
<tr>
<td>$hard$</td>
<td>0.89 1.00</td>
</tr>
</tbody>
</table>

We can draw the following conclusions from Table 3:

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

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

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

– 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 standard 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: (1) 2IDA needs less time than CG (column ≺), (2) needs a similar (difference < 5%) time (column ≃), and (3) needs more time than CG (column ≻). 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.

<table>
<thead>
<tr>
<th>Instance set</th>
<th>Avg. CPU Time [s]</th>
<th>2IDA vs CG (CPU)</th>
<th>Avg. nb columns</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2IDA</td>
<td>CG</td>
<td>≺</td>
</tr>
<tr>
<td>vb20</td>
<td>9</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>vb50-1</td>
<td>40</td>
<td>50</td>
<td>14</td>
</tr>
<tr>
<td>vb50-2</td>
<td>89</td>
<td>125</td>
<td>17</td>
</tr>
<tr>
<td>vb50-3</td>
<td>44</td>
<td>106</td>
<td>20</td>
</tr>
<tr>
<td>vb50-4</td>
<td>66</td>
<td>88</td>
<td>19</td>
</tr>
<tr>
<td>vb50-5</td>
<td>28</td>
<td>86</td>
<td>20</td>
</tr>
<tr>
<td>vbInd</td>
<td>8</td>
<td>14</td>
<td>10</td>
</tr>
<tr>
<td>m01</td>
<td>0.8</td>
<td>0.8</td>
<td>549</td>
</tr>
<tr>
<td>m20</td>
<td>0.4</td>
<td>0.6</td>
<td>793</td>
</tr>
<tr>
<td>m35</td>
<td>0.2</td>
<td>0.4</td>
<td>920</td>
</tr>
<tr>
<td>hard</td>
<td>118</td>
<td>377</td>
<td>10</td>
</tr>
</tbody>
</table>

be explained by two factors. The first iterations produce a good lower bound solution \( y^*_k \), because we exploit the structure of a good dual solution provided by DFFs (see App. A.2). Regarding the intermediate upper bounds, they are calculated (recall Sec. 4.3) by only generating \( y^*_k \)-tight constraints, which induced a form of implicit stabilization around \( y^*_k \).

Finally, let us emphasize the main reason lying behind the speed of the lower bound for large \( k \). The incremental construction of polytope \( P_{k+1} \) from lifted \( P_k \) constraints (Sec. 4.2) can considerably reduce the time needed to optimize \( P_{k+1} \). In several cases, the lifted \( P_k \) constraints are sufficient to allow 2IDA conclude \( \text{lb}_{k+1} = \text{lb}_k \) in almost 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) 0.8\( C \), 0.9\( C \) 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. 4.3 and the split method guided by upper bounds (App. A.1). 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 \( \mu_a \) 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/).
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.

In both 2IDA versions in Tables 5-6, the lower bound at $k = 1$ is often equal to $\sum_{i \in I} b_i w_i$ (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 $a$ with $w^\top a \in \{0.8C, 0.9C, C\}$,
which yield many constraints that can be written $y^\top a \leq \frac{1}{C} w^\top a$. Observe these constraints are verified with equality by $y_i = \frac{w_i}{C}$. The more numerous these constraints are, the more difficult it is to find dual solutions that dominate $y_i = \frac{w_i}{C}$, i.e., we get closer to a CSP version in which the cost of any pattern $a$ is $\frac{1}{C} w^\top a$ and $y_i = \frac{w_i}{C}$ is optimal.

An interesting property of ML-CSP is that one can no longer compute lower bounds using all DFFs that proved to be very effective for pure CSP [3]. Indeed, since the dual constraints do not always have a right-hand side coefficient of 1, some DFFs do not produce dual feasible solutions (see also App. A.2). In fact, the dual solution $y_i = \frac{w_i}{C}$ discussed above is not necessarily feasible for any cost function $\mu$, e.g., it would not be feasible if the roll of width $0.8C$ had cost 0.7. On the other hand, 2IDA would still be applicable for different cost functions; in fact, to our knowledge, it is the only method that can produce high-quality bounds so rapidly for such CSP variants.

### 6.4. Experiments on Low-Waste Cutting Stock

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

- 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 and with the simple 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, CG is slower than 2IDA. 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.
<table>
<thead>
<tr>
<th>Instance</th>
<th>Instance</th>
<th>Instance</th>
<th>Instance</th>
<th>Instance</th>
<th>Instance</th>
<th>Instance</th>
<th>Instance</th>
<th>Instance</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>k=1</td>
<td>k=2</td>
<td>k=3</td>
<td>OPT CG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m01-1</td>
<td>49 [0.02]</td>
<td>52 [0.03]</td>
<td>53 [0.03]</td>
<td>49 [0.04]</td>
<td>53 [0.06]</td>
<td>35 [0.07]</td>
<td></td>
<td></td>
<td>LW-CSP/CSP</td>
</tr>
<tr>
<td>m01-2</td>
<td>50 [0.01]</td>
<td>24 [0.02]</td>
<td>54 [0.03]</td>
<td>47 [0.04]</td>
<td>54 [0.05]</td>
<td>41 [0.06]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m01-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>55/54</td>
</tr>
<tr>
<td>vbInd30p0</td>
<td>90 [0.3]</td>
<td>21 [0.3]</td>
<td>90 [0.5]</td>
<td>20 [0.6]</td>
<td>90 [0.9]</td>
<td>73 [1.0]</td>
<td>90/90</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vbInd43p20</td>
<td>29 [0.02]</td>
<td>15 [0.05]</td>
<td>29 [0.04]</td>
<td>15 [0.05]</td>
<td>29 [0.06]</td>
<td>13 [0.08]</td>
<td></td>
<td></td>
<td>36/29</td>
</tr>
<tr>
<td>vbInd43p21</td>
<td>40 [0.09]</td>
<td>24 [0.2]</td>
<td>40 [0.3]</td>
<td>33 [0.3]</td>
<td>40 [1.2]</td>
<td>9 [1.2]</td>
<td></td>
<td></td>
<td>40/40</td>
</tr>
<tr>
<td>hard-1</td>
<td>56 [0.5]</td>
<td>47 [0.6]</td>
<td>56 [214]</td>
<td>60 [162]</td>
<td>60 [236]</td>
<td></td>
<td></td>
<td></td>
<td>60/57</td>
</tr>
<tr>
<td>hard-3</td>
<td>55 [0.5]</td>
<td>47 [0.6]</td>
<td>55 [175]</td>
<td>59 [163]</td>
<td>58 [234]</td>
<td></td>
<td></td>
<td></td>
<td>59/55</td>
</tr>
</tbody>
</table>

*Remark that these 2IDA bounds are higher than the CG optimum OPT CG [CSP] for standard CSP (refer to last column). Standard lower bounds for CSP (e.g., the DFFs) could not produce lower bounds better than OPT CG [CSP].

In this exceptional case, the CG fully converged before the end of step $k = 2$ of 2IDA.

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 subproblems with more complex constraints solved by a generic ILP solver.

References


Appendix A: Customizing the Group Split Methods

While the way groups are split has no theoretical impact on our study, it is essential for the practical effectiveness of 2IDA and it deserves further analysis. In Sec. 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 to absolutely any problem.

2. A splitting strategy (Sec. 4.1.2) that aims at making the aggregated polytope cover a reference solution that is better 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 as reference solution the 2IDA upper bound (when available). The second method (Sec. A.2) is only applicable to CSP and LW-CSP and it uses as reference solution the dual solution constructed from dual-feasible functions. These functions are well-acknowledged to provide CSP lower bounds [3, 14] very rapidly, 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 \( j^* \in \{1, \ldots, k \} \) 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 \( y^*_k \) of \( P_k \) to an outside reference solution. This reference is given by an upper bound solution \( y^u \) calculated as described in Sec. 4.3. The main idea is that \( y^*_k \rightarrow y^u \) is an improving open direction (see Def. 4.1), i.e., there is no \( P \) constraint that can block a (sufficiently small) advance from \( y^*_k \) to \( y^u \).

We need to identify and break aggregation links that do block such advance. This can be done by evaluating the difference between \( y^*_k \) and \( y^u \) over segments \([i_1, i_2]\) of each group \( j \). More exactly, we define the operator \( \Delta_j(i_1, i_2) = \sum_{i=i_1}^{i_2} b^j \cdot (y^u - y^*_k)_i \), where \((y^u - y^*_k)_i\) is the \( i \)th element of vector \((y^u - y^*_k)\) restricted to the elements of group \( j \).

The pseudo-code of this split method is presented in Alg. 3 and commented upon next. We develop on the case \( \Delta_j(1, n_j) > 0 \), i.e., an advance along \( y^*_k \rightarrow y^u \) would generate a positive (increasing) “trend” over the elements of group \( j \). The negative case \( \Delta_j(1, n_j) \leq 0 \) is symmetric and can be reduced to the positive one using the simple inversion from Lines 2-4. In Lines 5-12, Alg. 3 determines the best split point \( i^*(j) \) of each group \( j \). For this, it first determines two indices \( i_a \) and \( i_b \) such that there is a “negative trend” over the segment \([i_a, i_b]\), i.e., by decreasing \( y^*_k \) over \([i_a, i_b]\) and increasing it over \([1, i_a - 1]\) and \([i_b + 1, n_j]\), \( y^*_k \) could advance towards \( y^u \). Technically, we set \( i_a = \min\{i \in \{1, \ldots, n_j\} : \Delta_j(1, i - 1) \geq 0, \Delta_j(i, i) < 0\} \) and \( i_b = \max\{i \in \{i_a, \ldots, n_j\} : \Delta_j(i_a, i) < 0\} \). If \( \Delta_j(1, i_a - 1) > \Delta_j(i_b + 1, n_j) \), we consider that the best split option is to cut group \( j \) into intervals \([1, i_a - 1]\) and \([i_a, n_j]\), leading to \( i^*(j) = i_a - 1 \); 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\( (\Delta_j(1, i_a - 1), \Delta_j(i_b + 1, n_j)) \) as described above.

Furthermore, we multiply this \( h(j) \) score by the weight spread \( w^j_{\max} - w^j_{\min} \) of the group \( j \), 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 (resp. large) items will have a dual value of 0 (resp. 1) at optimality 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

We recall that \( f : [0, C] \rightarrow [0, 1] \) is a dual-feasible function (DFF) if and only if the following holds for any index set \( I \) and any values \( a_i \in \mathbb{Z}^+ \) and \( w_i > 0 \ ( \forall i \in I \)):

\[
\sum_{i\in I} a_i w_i \leq C \implies \sum_{i\in I} a_i f(w_i) \leq 1.
\]

This property ensures that \( y_i = f(w_i) \) is a dual feasible solution for CSP and for LW-CSP (considering \( C^+ = C \) and any \( C^- \in \{1, \ldots, C^+\} \)). This dual solution is not necessarily feasible for ML-CSP or for other CSP versions in which the configurations costs are not always 1.

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 produces the dual feasible solution \( y_i = \frac{w_i}{C} \). However, most DFFs are given by staircase
Algorithm 3: Group Split Routine Guided by an Upper Bound Reference Solution

Data: $y_k^*$ and $y^u$

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

1. for $j \leftarrow 1$ to $k$
do
  2. if $\Delta_j(1,n_j) < 0$ then
  3.      $y_k^* \leftarrow -y_k^*$ \hspace{1cm} // simple inversion to reduce the
  4.      $y^u \leftarrow -y^u$ \hspace{1cm} // negative case to a positive case
  5.      $i_a \leftarrow \min \{i \in \{1, \ldots, n_j\} : \Delta_j(1,i-1) \geq 0, \Delta_j(i,i) < 0\}$
  6.      $i_b \leftarrow \max \{i \in \{i_a, \ldots, n_j\} : \Delta_j(i_a,i) < 0\}$
  7.      if $\Delta_j(1,i_a-1) > \Delta_j(i_b+1,n_j)$ then
  8.          $i^*(j) \leftarrow i_a - 1$ \hspace{1cm} // Obtain intervals $[1, i_a - 1]$ and $[i_a, n_j]$
  9.          $h(j) \leftarrow \Delta_j(1, i_a - 1)$
  10. else
  11.      $i^*(j) \leftarrow i_b$ \hspace{1cm} // Obtain intervals $[1, i_b]$ and $[i_b + 1, n_j]$
  12.      $h(j) \leftarrow \Delta_j(i_b + 1, n_j)$
  13.      $h(j) \leftarrow h(j) \cdot (w_{\max}^j - w_{\min}^j)$ \hspace{1cm} // Discourage splitting groups with similar weights
  14. if $j = 1$ or $j = k$ then
  15.      $h(j) \leftarrow h(j) \cdot 2$ \hspace{1cm} // Encourage splitting extremal groups
  16. if $h(j) > h(j^*)$ then
  17.      $j^* \leftarrow j$ \hspace{1cm} // Initially, set $j^* = 0$ and $h(j^*) = -\infty$

18. return $(j^*, i^*(j^*))$

functions. To be sure we guide the splits using a reasonable reference solution, we always choose from [3] the DFF $f$ with maximum 10 intervals (pieces) that yields the highest objective value.

The proposed DFF-based split method identifies $k$ intervals of $[0,C]$ before starting the construction of $P_k$. Specifically, these $k$ intervals are chosen so that $f$ is linear over each of them. When 2IDA reaches this value of $k$, it generates an aggregated polytope $P_k$ that does include the solution $y_i = f(w_i)$, because $f$ is linear over all groups of $P_k$. The 2IDA bound calculated this way dominates the DFF bound associated to $f$. If $y_i = f(w_i)$ is an optimum dual solution in $P$, then $\text{DSCvr}(P_k)$ also contains this $P$ optimum. If $f$ does not lead to an optimal solution, it may always give a good starting point for 2IDA.