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

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

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 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 dual model is a restriction of the original one, and it can be rapidly optimized to obtain a feasible dual solution. A primal bound can also be obtained by restricting the set of columns to those saturated by the dual feasible solution obtained by aggregation. The convergence is realized by iterative disaggregation until the gap is closed by the bounds. Computational results show the usefulness of our method for different cutting-stock problems. An important advantage is the fact that it can produce high-quality dual bounds much faster than the traditional lagrangian bound used in stabilized column generation.

1. Introduction

Column generation (CG) is a widespread technique for optimizing linear programs (LPs) with prohibitively-many variables. Without loss of generality, we consider a minimization objective 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 of CG and reduce the number of iterations [1, 2, 4, 5, 6, 7, 8, 11, 14, 15].

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 [13], nearby locations, related scheduling tasks. Our work is in part motivated by the recent progress of aggregation techniques in CG. These techniques can limit degeneracy in the master LP, reduce the number of dual variables and produce a stabilization effect [1, 7, 8].

We propose a new so-called *linear dual aggregation* that leads to a reduced-size CG model whose optimum is a valid lower bound for sought non-aggregated CG optimum: the aggregated model is a restriction of the original dual model. We now present the general main idea. A covering problem is defined on a ground set I and the CG model uses a dual variable y_i for each $i \in I$. In a 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^+. Our model exploits the following assumption: when the resource constraints play a major role, the optimal solution selects many configurations with tight resource constraints (*e.g.*, filled bins, patterns without waste), 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 a linear function of w_i. This leads to a smaller aggregated CG model with $2k$ dual variables, which are the parameters of the affine functions for each group. The aggregated model is iteratively refined by splitting groups, until it can be proved that it has reached the optimum of the non-aggregated CG model. To ensure a faster convergence, we intertwine the aggregation calculations with a fast CG process that generates an upper bound for each value of k.

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-partition. Section 4 presents the convergent approach that iteratively breaks groups, computing a lower and an upper bound for each k. In Section 5, we show that our linear aggregation has better theoretical properties than a simpler equality aggregation. Section 6 presents numerical tests, followed by conclusions in the last section; the appendix presents the incremental calculations used to iterate from k to $k+1$ as fast as possible.

2. Column Generation for Covering Problems and Related Work

2.1. CG Models with Resource Constraints and Dynamic Programming Pricing

We first introduce the 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 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 specific subproblem. In this paper, as stated above, we consider that the subproblem includes a resource constraint $C^- \leq \mathbf{w}^\top \mathbf{a} \leq C^+$. The basic set-covering problem requires finding the minimum number of configurations needed to cover each $i \in I$. However, we here consider the most general multi-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, which depends 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, whose linear relaxation can be stated as follows: $\big\{\min \sum_{\mathbf{a}\in\mathcal{A}} \mu_a \lambda_a : \sum_{\mathbf{a}\in\mathcal{A}} a_i \lambda_a \geq b_i, \forall i \in \{1, \ldots, n\}, \lambda_a \geq 0, \forall \mathbf{a} \in \mathcal{A}\big\}$. Its dual can be 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 covering problem* (`DCvr`) formulation over polytope $\mathcal{P}$. We will write (2.1) as $\texttt{DCvr}(\mathcal{P}) = \max\big\{\mathbf{b}^\top \mathbf{y} : \mathbf{y} \in \mathcal{P}\big\}$. Its optimum value $\mathrm{OPT}(\texttt{DCvr}(\mathcal{P}))$ is referred to as the CG optimum and it is denoted by $\mathrm{OPT}_{\mathrm{CG}}$. To determine $\mathrm{OPT}_{\mathrm{CG}}$, a CG algorithm dynamically generates a subset of the constraints of $\mathcal{P}$ (primal columns). In the dual space, the CG can be seen as a cutting-plane method that iterates two steps: (i) find the optimal solution $\mathbf{y}$ of the current dual polytope $\mathcal{P}^{\mathrm{out}} \supset P$ described by the constraints generated so far ; and (ii) pricing subproblem: generate a new valid constraint of $\mathcal{P}$ violated by $\mathbf{y}$ and add it to $\mathcal{P}^{\mathrm{out}}$, or report $\mathrm{OPT}(\texttt{DCvr}(\mathcal{P}^{\mathrm{out}})) = \mathrm{OPT}_{\mathrm{CG}}$ if no such constraint exists. *The pricing CG subproblem* seeks a configuration of most negative reduced cost:

$$\min\big\{\mu_a - \mathbf{y}^\top \mathbf{a} : \mathbf{a} \in \mathcal{A}\big\}, \tag{2.2}$$

where $\mathbf{y} \in \mathbb{R}^n_+$ is the current dual solution. As such, the pricing subproblem asks to 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. A key point in CG is the asymptotic running time needed to solve this pricing sub-problem.

In a resource-constrained context, all configurations $\mathbf{a} \in \mathcal{A}$ satisfy $C^- \leq \mathbf{w}^\top \mathbf{a} \leq C^+$, where $\mathbf{w} = (w_1, w_2, \ldots, w_n)^\top \in \mathbb{Z}^n_+$ is a vector of resource consumptions and $C^-, C^+ \in \mathbb{Z}_+$

are the two-sided bounds on the total consumption. In many such cases, the fastest pricing algorithms use a pseudo-polynomial dynamic programming (DP) approach with a complexity depending on C^+. For this, one defines a profit function $\phi_{\max}$ that maps a DP state $(c,i) \in \{0,\dots,C^+\} \times \{1,\dots,n\}$ to the maximum value $\phi_{\max}(c,i)$ of profit $\mathbf{y}^\top \mathbf{a}$ over all configurations $\mathbf{a}$ that satisfy $\mathbf{w}^\top \mathbf{a} = c$ and that only use elements of $\{1,\dots,i\}$. Starting with $\phi_{\max}(0,0) = 0$, one can determine $\phi_{\max}(c,i)$ for *all* reachable states $(c,i) \in \{C^-,\dots,C^+\} \times \{1,\dots,n\}$ using a recursion such as:

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

Regarding the costs μ_a of configurations $\mathbf{a} \in \mathcal{A}$, they often depend substantially on the total resource consumption $\mathbf{w}^\top \mathbf{a}$ of $\mathbf{a}$. Restricting to this case, the cost μ_a can be determined separately as a preprocessing for each feasible value of resource consumption. By slightly abusing notations, we can write $\mu_a = \mu(\mathbf{w}^\top \mathbf{a})$. The best reduced cost is thus attained in a state (c^*, n) such that: $\phi_{\max}(c^*,n) - \mu(c^*) = \max_{c \in \{C^-,\dots,C+\}} \phi_{\max}(c,n) - \mu(c)$.

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 [16, §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 $O(C^+ \times n)$ states of DP can be constructed in $O(C^+ \times n_b)$ time, 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, and so, r can vary from 0 to b_i in (2.3).

2.2. Related Work in Aggregation Methods

If an ILP is too large to be directly tackled by generic solvers, it is often optimized by outer approximation: start with an ILP with fewer constraints/variables and progressively enrich it with new constraints by solving a separation problem. Cutting planes and CG are among the most famous examples of such methods. Given this outer approach, the convergence of CG is ensured by a sequence of dual infeasible solutions, *i.e.*, before the last iteration, all intermediate dual solutions in CG are dual infeasible.

Another way of obtaining a tractable model consists of aggregating constraints or variables. In some cases, this offers the advantage of constructing an inner approximation, in

the sense that the aggregated models only contain feasible solutions—see also [16, §1.1.1.1] for comparisons between inner and outer approximations. For instance, a static aggregation is presented by [18], which defines a smaller model whose size depends on a given parameter. An interesting conclusion of [18] is that small values of this parameter are often sufficient to obtain excellent dual bounds. However this method is static and does not converge toward the optimum of the initial model.

In the context of CG for covering or partitioning problems, aggregation methods are used to group together elements that are either similar, or that appear very often together in columns (in subproblem solutions). The latter property is used in the DCA algorithm proposed in [1, 7, 8]. This aggregation approach starts by restricting the set of feasible columns as follows. 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 current dual polytope (for the restricted master) includes the original dual polytope (for the full master). Thus, the optimum of the current dual program is an upper bound for the sought CG optimum.

A possible 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 [9], 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 articles in cutting-stock).

However, if the resource has a more complex impact on the subproblem, the "equivalence" of different elements may be less obvious. Such situations require a more refined 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, an empirical 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 with few pieces leads to nearly-optimal dual solutions 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 a Fixed Partition of I

We here consider a fixed partition of $I = \{1, \dots, n\}$ into k groups. The goal of the linear aggregation is to enforce the dual values in each group to follow a linear function with regards to the resource consumptions of the elements. For this, we add a new linear equality for each of the k groups, constructing a restricted dual LP. As such, the optimum of the resulting model is a lower bound for the original problem. 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. This allows to solve the aggregated model using a smaller master problem and a smaller aggregated pricing subproblem.

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

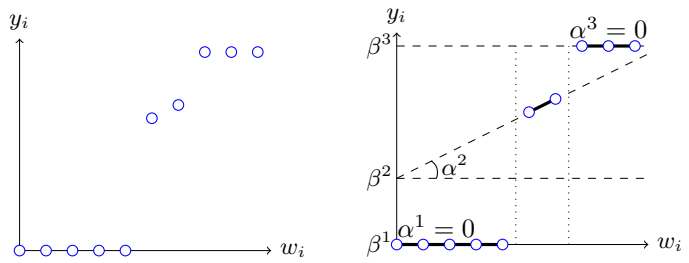


Figure 1 **Expressing all dual values by a group-wise linear function with $k = 3$ intervals.**

Note. The figure on the left plots some dual values y_i as a function of the consumption value w_i. The right figure shows a three-wise linear function that can underline such y_i values, with solid lines inside each group.

Let $G_k = \{I^1, I^2, \dots, I^k\}$ denote the partition of I and $n_j = |I^j|$ the cardinality of group j. The linear restriction is implemented as follows: given any group $j \in \{1, \dots, k\}$, the values of the dual variables y_i for all $i \in I^j$ are written as an affine function of their resource consumptions w_i. Formally, we impose that there exists $\alpha^j, \beta^j \in \mathbb{R}$ such that $y_i =$

$\alpha^j w_i + \beta^j$, $\forall i \in I^j$, as illustrated in Fig. 1. By linking the $2k$ new variables $\alpha^1, \alpha^2, \ldots, \alpha^k$ and $\beta^1, \beta^2, \ldots, \beta^k$ to the $\mathbf{y}$ variables in (2.1), we obtain the first *restricted dual polytope* in space $\mathbb{R}^n_+ \times \mathbb{R}^{2k}$:

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

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

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

In general, we have $proj_y(\mathcal{P}_k^{y,\alpha,\beta}) \subsetneq \mathcal{P}$, because only the vectors $\mathbf{y} \in \mathcal{P}$ with the suitable group-wise affine structure do represent valid solutions of (3.1).

Model (3.1) can be reformulated using only variables α^j and β^j. We first rewrite the objective function, by decomposing $\sum_{i=1}^n b_i y_i$ into k sums of the form $\sum_{i \in I^j} b_i y_i$.

$$\mathbf{b}^\top \mathbf{y} = \sum_{j=1}^k \sum_{i \in I^j} b_i y_i = \sum_{j=1}^k \sum_{i \in I^j} b_i \left(w_i \alpha_j + \beta_j\right) = \sum_{j=1}^k \left(\sum_{i \in I^j} b_i w_i\right) \alpha_j + \left(\sum_{i \in I^j} b_i\right) \beta_j \tag{3.2}$$

We now address the first constraints of (3.1). We decompose as above $\mathbf{a}^\top \mathbf{y}$ into $\sum_{j=1}^j \sum_{i \in I^j} a_i y_i$. For each $j \in \{1, 2, \ldots, k\}$, we rewrite:

$$\sum_{i \in I^j} a_i y_i = \sum_{i \in I^j} a_i \left(w_i \alpha_j + \beta_j\right) = \left(\sum_{i \in I^j} a_i w_i\right) \alpha^j + \left(\sum_{i \in I^j} a_i\right) \beta_j \tag{3.3}$$

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

DEFINITION 3.1. Given a configuration $\mathbf{a} \in \mathcal{A}$ and a group j, we introduce the following notations. Observe c_a^j and N_a^j are resp. the coefficients of variables α^j and β^j in (3.3).

- $c_a^j = \sum_{i \in I^j} a_i w_i$: *total resource consumption* of the elements of I^j selected in $\mathbf{a}$.
- $N_a^j = \sum_{i \in I^j} a_i$: *total number of elements of* I^j selected in $\mathbf{a}$.
- $w^j_{\min} = \min_{i \in I^j} w_i$ and $w^j_{\max} = \max_{i \in I^j} w_i$: 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 $\mathbb{R}^{2k}$:

$$\left.\begin{array}{ll} \max \sum_{j=1}^{k} \left(\sum_{i\in I^j} b_i w_i\right) \alpha_j + \left(\sum_{i\in I^j} b_i\right) \beta_j & \\ \sum_{j=1}^{k} c_a^j \alpha^j + N_a^j \beta^j \leq \mu_a, \ \forall \mathbf{a} \in \mathcal{A} & \\ w_{\min}^j \alpha^j + \beta^j \geq 0 & \forall j \in \{1,\ldots,k\} \\ w_{\max}^j \alpha^j + \beta^j \geq 0 & \forall j \in \{1,\ldots,k\} \\ \alpha^j, \beta^j \in \mathbb{R}, & \forall j \in \{1,\ldots,k\} \end{array}\right\} \mathcal{P}_k^{\alpha,\beta} \tag{3.4}$$

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 $\mathcal{P}_k^{\alpha,\beta}$ in (3.4): it is enough to substitute (3.3) in (3.1) and to apply Def. 3.1 to obtain the first constraint of $\mathcal{P}_k^{\alpha,\beta}$. The non-negativity constraint $y_i \geq 0$ of $\mathcal{P}_k^{y,\alpha,\beta}$ is also equivalent to the last two constraints in (3.4): $y_i = w_i\alpha^j + \beta^j \geq 0, \forall i \in I^j \Longleftrightarrow w_{\min}\alpha^j + \beta^j, w_{\max}\alpha^j + \beta^j \geq 0$.□

The new model (3.4) uses less dual variables, which reduces the potential oscillations of these variables and can speed-up the algorithm to solve the restricted master LP.

3.2. Restricting to extremal non-dominated configurations

So far we have built a master problem with less variables. We now show that many constraints become redundant in this model. This will allow us to reduce the number of variables in the subproblem as well. For any configuration $\mathbf{a} = (a_1,\ldots,a_n) \in \mathcal{A}$, the associated constraint in (3.4) only uses aggregated coefficients N_a^j and c_a^j. 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 $\overline{\mathbf{a}} = (c_a^1, N_a^1, c_a^2, N_a^2, \ldots, c_a^k, N_a^k)$. Such an *aggregated configuration* $\overline{\mathbf{a}}$ represents all $\mathbf{a} \in \mathcal{A}$ that satisfy $c_a^j = \sum_{i\in I^j} a_i w_i$ and $N_a^j = \sum_{i\in I^j} a_i$ *for all* $j \in \{1,\ldots,k\}$ (see also Def. 3.1).

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. 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 $\mathbf{a} \in \mathcal{A}$ such that $c_a^j = c^j$. These two values are referred to as the cardinality coefficients of feasible resource consumption $c^j \in R^j$;

– $\mathcal{A}_k = \{a \in \mathcal{A} : N_a^j = N^+(j,c_a^j)$ or $N_a^j = N^-(j,c_a^j), \forall j \in \{1,\ldots,k\}\}$.

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

$$\left.\begin{array}{rl} \max \sum_{j=1}^{k} \left(\sum_{i\in I^j} b_i w_i\right) \alpha_j + \left(\sum_{i\in I^j} b_i\right) \beta_j & \\ \sum_{j=1}^{k} c_a^j \alpha^j + N_a^j \beta^j \leq \mu_a, \ \forall \mathbf{a} \in \mathcal{A}_k & \\ w_{\min}^j \alpha^j + \beta^j \geq 0 \qquad \forall j \in \{1,\ldots,k\} & \\ w_{\max}^j \alpha^j + \beta^j \geq 0 \qquad \forall j \in \{1,\ldots,k\} & \\ \alpha^j, \beta^j \in \mathbb{R} \qquad \forall j \in \{1,\ldots,k\} & \end{array}\right\} \mathcal{P}_k \tag{3.5}$$

Obviously, if two configurations $\mathbf{a}, \mathbf{a}' \in \mathcal{A}_k$ 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 Covering Problem* (`DCvr`) LP over $\mathcal{P}_k$ and written $\texttt{DCvr}(\mathcal{P}_k) = \max\{\sum_{j=1}^{k} \left(\sum_{i\in I^j} b_i w_i\right) \alpha_j + \left(\sum_{i\in I^j} b_i\right) \beta_j : (\alpha^1,\ldots,\alpha^k,\beta^1,\ldots,\beta^k)^\top \in \mathcal{P}_k\}$. We finish by formally showing that $\mathcal{P}_k$ can be projected onto a polytope that is included in $\mathcal{P}$.

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 $proj_y(\mathcal{P}_k^{y,\alpha,\beta}) \subseteq \mathcal{P}$. Prop. 3.2 shows that $\mathcal{P}_k^{y,\alpha,\beta}$ is equivalent to $\mathcal{P}_k^{\alpha,\beta}$. It is enough to show that $\mathcal{P}_k^{\alpha,\beta} = \mathcal{P}_k$. 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.*, the constraint of $\mathbf{a}$ is weaker than that of $\mathbf{a}'$.

Using the notations from Def. 3.3, we observe that $\mathbf{a} \notin \mathcal{A}_k \implies \exists j \in \{1,\ldots,k\}$ such that $N^-(j, c_a^j) < N_a^j < N^+(j, c_a^j)$. As such, $\mathbf{a}$ yields a constraint in which the $j^{\texttt{th}}$ term $c_a^j \alpha^j + N_a^j$ is sandwiched by $c_a^j \alpha^j + N^+(j, c_a^j)\beta^j$ and $c_a^j \alpha^j + N^-(j, c_a^j)\beta^j$. The constraint of $\mathbf{a}$ is thus weaker than the constraint of some $\mathbf{a}'$ that does verify $N_{a'}^j = N^+(j, c_a^j)$ or $N_{a'}^j = N^-(j, c_a^j)$. By applying this for all groups j with above properties, we obtain a configuration $\mathbf{a}' \in \mathcal{A}_k$ that yields a constraint dominating the constraint of $\mathbf{a}$. □

3.3. Computational Method: Aggregated CG to Optimize $\texttt{DCvr}(\mathcal{P}_k)$

3.3.1. Pre-processing calculation of the values $N^+(j, c^j)$ and $N^-(j, c^j)$ To construct the aggregated constraints of $\mathcal{P}_k$ in (3.5), one first needs to compute all values $N^+(j, c^j)$ and $N^-(j, c^j)$ for all $c^j \in R^j, \forall j \in \{1,\ldots,k\}$. This is done in a preprocessing stage, executed only once. To determine $N^+(j, c^j)$ and $N^-(j, c^j)$ for all $c^j \in R^j$, we respectively replace y_i with 1 or -1 in (2.3) and maximize $\phi_{\max}$ as in Sec. 2.1. This DP scheme constructs the set R^j at the same time for each group j. Considering all groups $j \in \{1,\ldots,k\}$ together, the preprocessing has the same pseudo-polynomial complexity as the initial DP algorithm.

3.3.2. Aggregated column generation Model (3.5) is optimized using CG. The pricing problem could be solved by projecting the current solution into the original space, but this would not benefit from the simpler structure of the model. Actually, the pricing problem 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* $\bar{\mathbf{a}} = (c_a^1, N_a^1, c_a^2, N_a^2, \ldots, c_a^k, N_a^k)$ that maximizes the profit $\sum_{j=1}^k \alpha^j c_a^j + \beta^j N(j, c_a^j)$ minus the cost $\mu\big(\sum_{j=1}^k c_a^j\big)$. Formally, the aggregated pricing can be expressed using $c^1, c^2, \ldots, c^k$ as decision variables:

$$\begin{aligned} \max\ & \textstyle\sum_{j=1}^k \alpha^j c^j + \beta^j N(j, c^j) - \mu(\sum_{j=1}^k c^j) \\ & C^- \le \textstyle\sum_{j=1}^k c^j \le C^+ \\ & c^j \in R^j\ \forall j \in \{1, \ldots, k\}, \end{aligned} \tag{3.6}$$

where R^j is the set of feasible resource consumptions, and $N(j, c^j) = N^+(j, c^j)$ if $\beta^j \ge 0$ or $N(j, c^j) = N^-(j, c^j)$ otherwise. Note that it is sufficient to determine the values of c^j, since all N^+ and N^- values are determined during preprocessing, all β^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 β^j.

We now reformulate (3.6) using binary decision variables x_c^j such that $x_c^j = 1$ if and only if group j uses a total resource amount of $c \in R^j$ $(\forall j \in \{1, \ldots, k\})$. Using notational shortcut $p_c^j = \alpha^j c + \beta^j N(j, c)$, (3.6) becomes

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

The resulting aggregated pricing subproblem is a multiple-choice variant of the non-aggregated pricing. The standard dynamic programming (DP) from Sec.2.1 can be extended to an aggregated DP. All columns with the same resource consumption and number of elements per group are aggregated into one column. This may reduce certain calculations, *e.g.*, there is no longer need to scan all $n_b = \sum_{i=1}^n b_i$ individualized elements for each $c \in \{0, 1, \ldots, C^+\}$ as in Sec 2.1, but it is enough to scan R^j for all $j \in \{1, 2, \ldots k\}$.

4. The Convergent Algorithm

We have 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 lb_k is a lower bound of the sought $\mathrm{OPT}_{\mathrm{CG}}$. We now describe how this bound can be iteratively improved to compute $\mathrm{OPT}_{\mathrm{CG}}$.

Algorithm 1: Iterative Inner Dual Approximation

$k \leftarrow 1, G_1 \leftarrow \{I\}, \mathcal{A}_1 \leftarrow$`initialConstr()` *// write all constraints with 2 variables*

repeat

- `calcCardCoefs`(G_k) *// calculate all N^-, N^+ values (Sec. 3.3.1)*
- $\text{lb}_k, \mathcal{A}_k, \mathbf{y^*_k} \leftarrow$`aggregatedCG`$(\mathcal{A}_k)$
- $G_{k+1} \leftarrow$ `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. 4.2)*
- $\text{ub}_k \leftarrow$ `ubCgProcess`$(\mathcal{A}_k, \mathbf{y^*_k})$ *// optional upper bound of OPT_{CG} (Sec. 4.3)*
- $k \leftarrow k+1$

until *a stopping condition is reached* *// e.g., $\lceil lb_k \rceil = \lceil ub_k \rceil$ if $\mu_a \in \mathbb{Z}, \forall \mathbf{a} \in \mathcal{A}$*

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 $\mathcal{P}_k$. 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 lb_{k+1} from lb_k without optimizing $\mathcal{P}_{k+1}$ from scratch (Sec. 4.2) ;

– how to compute an upper bound ub_k from lb_k (Sec. 4.3), to allow 2IDA to stop earlier by closing the gap between lb_k and ub_k.

4.1. Group Split Operators

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 a model $\mathcal{P}_k$ equivalent to $\mathcal{P}$. However, the split decisions are crucial for the practical effectiveness of 2IDA. We first present (Sec. 4.1.1) a simple method, which can be used in a basic setting of 2IDA. The second operator (Sec. 4.1.2) is more refined and needs the knowledge of a reference solution that indicates a direction of evolution for the optimum $\mathcal{P}_k$ solution.

4.1.1. Basic Dichotomic Split Method This method considers only groups associated to segments of $[0, C^+]$ with a regular length of the form $\frac{1}{2^\ell}C^+$ ($\ell \in \mathbb{N}$). The initial split operation takes the segment $[0, C^+]$ of the unique group $k = 1$ and breaks it into two sub-groups $I^1 = \left\{i \in I : \ w_i \leq \frac{C^+}{2}\right\}$ and $I^2 = \left\{i \in I : \ w_i > \frac{C^+}{2}\right\}$. If any of these two sub-groups is empty, we break again the other sub-group in two; this can be repeated until we obtain two proper non-empty sub-groups. At each 2IDA iteration k, we choose to split a group j^*

that maximizes the *resource consumption spread*, *i.e.*, $w^{j^*}_{\max} - w^{j^*}_{\min} = \max_{j\in\{1,\dots,k\}} (w^j_{\max} - w^j_{\min})$. Group j^* is split by breaking its associated segment into two equal segments, as above.

4.1.2. Split Methods that Make $\mathcal{P}_k$ Evolve Towards Better Solutions In the remainder, to lighten the notation, we will work in the original space $\mathbb{R}^n$. Let $\mathcal{P}^y_k = proj_y(\mathcal{P}^{\alpha,\beta}_k)$. According to Proposition 3.3, this polyedral set is included in $\mathcal{P}$. We now consider that we are given a possibly infeasible solution $\mathbf{y^u} \in \mathbb{R}^n$ of higher quality than the optimum solution $\mathbf{y^*_k}$ of $\mathcal{P}^y_k$ (*i.e.*, $\mathbf{b}^\top\mathbf{y^u} > \mathbf{b}^\top\mathbf{y^*_k}$). For now we just assume that this solution is a good approximation for the optimum of the original non-aggregated problem. The goal is thus to remove the linearity restrictions that are the most violated by $\mathbf{y^u}$, so as to make $\mathcal{P}^y_k$ evolve to a polytope $\mathcal{P}^y_{k+1}$ that can contain solutions closer to $\mathbf{y^u}$.

DEFINITION 4.1. Given a solution $\mathbf{y} \in \mathcal{P}^y_k$ and some $\mathbf{y^u} \in \mathbb{R}^n_+$ we say that direction $\mathbf{y} \to \mathbf{y^u}$ is *an open direction* if there exists $\epsilon > 0$ such that $(\mathbf{y} + \epsilon(\mathbf{y^u} - \mathbf{y})) \in \mathcal{P}$. 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.* □

If $\mathbf{y^*_k} \to \mathbf{y^u}$ is an improving open direction, then all $\mathbf{y^*_k} + \epsilon(\mathbf{y^u} - \mathbf{y^*_k})$ are excluded from the current $\mathcal{P}^y_k$ only because $\mathbf{y^u}$ is non-linear over some $j \in \{1,\dots,k\}$. We choose to split a group j^* over which $\mathbf{y^u} - \mathbf{y^*_k}$ is well approximated by a 2-part linear function. By splitting j^* accordingly into two parts, $\mathbf{y^*_{k+1}}$ can become 2-part linear over group j^*, as in Fig. 2.

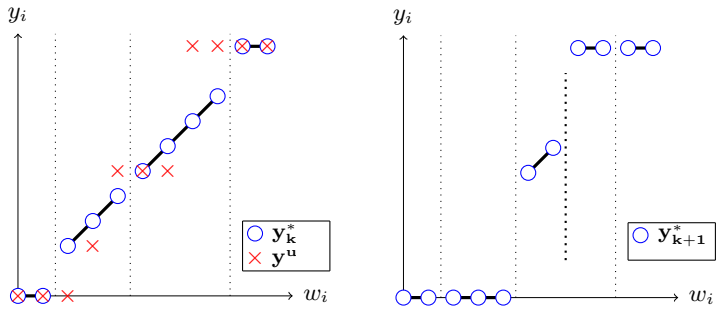


Figure 2 A split operation guided by a better solution $\mathbf{y^u}$.

Note. The left figure plots the values of a current solution $\mathbf{y^*_k}$ (blue circles) and of a reference solution $\mathbf{y^u}$ (red crosses) using 4 groups, delimited by dotted lines. The right figure plots possible values for $\mathbf{y^*_{k+1}}$ after splitting along the thicker line the group number $j^* = 3$, whose difference with $\mathbf{y^u}$ is the largest.

The exact calculations behind this split method are further discussed in App. A.1 (Alg. 3), but the above ideas are enough to present all other 2IDA components. We only say that $\mathbf{y^u}$ can be obtained from: (i) an upper bound solution if available (see App. A.1.1) or (ii) a solution generated by problem-specific methods (see App. A.1.2 for cutting-stock).

4.2. Fast Calculation of lb_{k+1} From lb_k by Lifting $\mathcal{P}_k$ Constraints

After solving $\texttt{DCvr}(\mathcal{P}_k)$ at step 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 lift the constraints 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 $\mathcal{P}_k$ constraints generated so far are lifted from dimension $\mathbb{R}^{2k}$ to dimension $\mathbb{R}^{2k+2}$.

The optimal solution of $\texttt{DCvr}(\mathcal{P}_k)$ 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 $j \in \{1,\dots,k\} \setminus \{k^*\}$. We recall that $\mathbf{y^*_k} \in \mathbb{R}^n$ is the optimal solution of $\texttt{DCvr}(\mathcal{P}_k)$ expressed in dimension n.

In a first step, our approach uses a polytope $\mathcal{P}'_{k+1}$ with $\mathcal{P}'^{y}_{k+1} \supset \mathcal{P}^y_k$ constructed by lifting $\mathcal{P}_k$ constraints satisfied with equality by $\mathbf{y^*_k}$. More exactly, we consider (constraint) set:

$$\mathcal{A}'_{k+1} = \big\{\mathbf{a}' \in \mathcal{A}_{k+1} : \exists \mathbf{a} \in \mathcal{A}_k, \mathbf{a}^\top \mathbf{y^*_k} = \mu_a, \text{ s.t. } c^j_a = c^j_{a'}, \;\; N^j_a = N^j_{a'} \;\forall j \in \{1,\dots,k\} \setminus \{k^*\}\big\}.$$

To optimize over $\mathcal{P}'_{k+1}$ by CG, we solve the same aggregated pricing as for $\mathcal{P}_{k+1}$ but with $\mathcal{A}'_{k+1}$ instead of $\mathcal{A}_{k+1}$. In practice, the pricing routine scans the generated configurations $\mathbf{a}$ satisfying $\mathbf{a}^\top \mathbf{y^*_k} = \mu_a$, recopies all their coefficients except those of group j^*, and computes the best coefficients for new groups j_1 and j_2. For this, it uses the multi-choice aggregated dynamic program from Sec. 3.3.2 with only two decision levels j_1 and j_2.

After determining $\text{OPT}(\texttt{DCvr}(\mathcal{P}'_{k+1}))$ this way, it turns to the original aggregated pricing for $\mathcal{P}_{k+1}$. Since $\mathcal{A}'_{k+1} \subset \mathcal{A}_{k+1}$, we have $\text{OPT}(\texttt{DCvr}(\mathcal{P}'_{k+1})) \geq \text{OPT}(\texttt{DCvr}(\mathcal{P}_{k+1})) \geq \text{OPT}(\texttt{DCvr}(\mathcal{P}_k))$. If $\text{OPT}(\texttt{DCvr}(\mathcal{P}'_{k+1})) = \text{OPT}(\texttt{DCvr}(\mathcal{P}_k))$, we can directly state $\text{lb}_{k+1} = \text{lb}_k$ only using lifted constraints. Our computational experiments show that lb_{k+1} can often be computed this way in almost negligible time.

To summarize, 2IDA actually optimizes $\texttt{DCvr}(\mathcal{P}_{k+1})$ in two steps: (1) it first lifts $\mathcal{P}_k$ constraints to find the best solution of $\mathcal{P}'_{k+1} \supset \mathcal{P}_{k+1}$ and (2) it seeks the best solution of $\mathcal{P}_{k+1}$ using the aggregated CG from Sec. 3.3.2. App. A presents the complete algorithm for determining $\texttt{DCvr}(\mathcal{P}_{k+1})$, followed by complexity and acceleration considerations.

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

We finish the 2IDA description with the optional upper bounding. It allows to stop 2IDA earlier by closing the gap between lb_k and ub_k, but it is also useful to guide the split operator (Sec 4.1.2). To avoid solving non-aggregated models from scratch, we propose a CG process based on a smaller problem that exploits the current optimum $\mathbf{y^*_k} \in \mathcal{P}^y_k$.

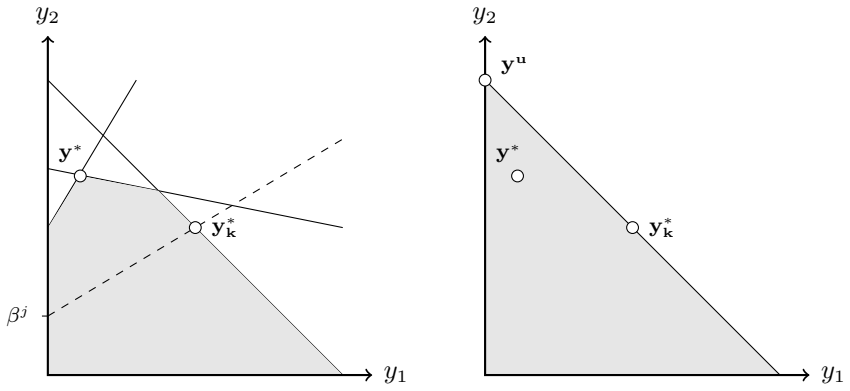


Figure 3 Computing an improving open direction.

Note. The picture is a projection on the plan defined by y_1, y_2 of the same group j for given values of α^j, β^j. On the left-hand figure, $\mathcal{P}$ is the grey polyhedron, while $\mathcal{P}_k$ is the segment of the dashed line between the point $(0, \beta^j)$ and $\mathbf{y^*_k}$. On the right hand figure, only the constraint that is tight for $\mathbf{y^*_k}$ is kept. The grey polyhedron is now $\mathcal{P}^u$. The optimal solution for the corresponding problem is $\mathbf{y^u}$, which defines an improving direction to go from $\mathbf{y^*_k}$ to $\mathbf{y^*}$.

An important observation is that $\mathbf{y^*_k}$ always verifies to equality at least one original constraint $\mathbf{a} \in \mathcal{A}$. If there were no constraints $\mathbf{a} \in \mathcal{A}$ such that $\mathbf{a}^\top \mathbf{y^*_k} = \mu_a$, then $\mathbf{y^*_k}$ would be dominated by a solution $\mathbf{y^*_k} + \epsilon(1, \dots, 1)^\top \in \mathcal{P}^y_k$ for a small-enough ϵ. We use this observation to restrict the CG algorithm in such a way that it generates only such $\mathbf{y^*_k}$-tight constraints. This plays a stabilization role and reduces the number of possible subproblem solutions by a wide range. We now define the corresponding polytope $\mathcal{P}^u \supset \mathcal{P}$, delimited only by $\mathbf{y^*_k}$-*tight constraints*:

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

Let $\mathbf{y^u} \in \mathbb{R}^n_+$ denote an optimal solution for $\max\{\mathbf{b}^\top \mathbf{y} : \mathbf{y} \in \mathcal{P}^u\}$. In Fig. 3, we give an example to illustrate the notions of $\mathbf{y^*_k}$-tight constraints and $\mathcal{P}^u$. We now show in Proposition 4.1 how $\mathbf{y^u}$ is useful for our method. Property (2) is useful for guiding the above group split heuristic (Sec. 4.1.2), while property (3) ensures that 2IDA stops whenever $\mathbf{y^*_k}$ is optimal for the original problem.

PROPOSITION 4.1. *Let* $\mathbf{y^u} \in \mathbb{R}^n_+$ *be an optimal solution of* $\max\{\mathbf{b}^\top \mathbf{y} : \mathbf{y} \in \mathcal{P}^u\}$. *Then* $\mathbf{y^u}$ *verifies the following properties:*

(1) upper bounding: $OPT_{CG} \leq \mathbf{b}^\top \mathbf{y^u}$;

(2) open direction: if $\mathbf{y^*_k}$ *is* ***not*** *an optimal solution of* $\texttt{DCvr}(\mathcal{P})$, *then* $\mathbf{b}^\top \mathbf{y^*_k} < \mathbf{b}^\top \mathbf{y^u}$ *and* $\mathbf{y^*_k} \to \mathbf{y^u}$ *is an improving open direction;*

(3) optimality proving: if $\mathbf{b}^\top \mathbf{y^*_k} = OPT_{CG}$, *then* $\mathbf{y^u}$ *also satisfies* $\mathbf{b}^\top \mathbf{y^u} = OPT_{CG}$.

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, $\mathbf{b}^\top \mathbf{y^u} \geq \mathrm{OPT}_{\mathrm{CG}}$.

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

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

We compute $\mathbf{y^u}$ by CG: $\mathcal{P}^u$ has exactly the same structure as $\mathcal{P}$ in (2.1), but it has a significantly smaller set of (only $\mathbf{y}^*_\mathbf{k}$-tight) constraints. The 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 $\mathbf{y}$. To solve this problem, we use the dynamic programming ideas from Sec. 2.1, but with a modified lexicographic objective: first maximize $(\mathbf{y}^*_\mathbf{k})^\top \mathbf{a}$, and, subject to this, maximize the $\mathbf{y}$-profit $\mathbf{y}^\top \mathbf{a}$. The resulting $\mathcal{P}^u$ pricing requires the same asymptotic running time as the classical $\mathcal{P}$ pricing.

Since $\mathcal{P}^u$ has far less constraints than $\mathcal{P}$, the CG convergence is generally faster on $\mathcal{P}^u$. Furthermore, the availability of a dual feasible solution $\mathbf{y}^*_\mathbf{k}$ will be used to stabilize the CG process using a `trust region` technique, *i.e.*, by puting a box around $\mathbf{y}^*_\mathbf{k}$ in the first stage of the CG process. The practical implementation will be discussed in Section 6.1.1.

5. Theoretical Properties of the Linear Aggregation

Up to now, we have described 2IDA on the dual formulation (2.1) of the *master* LP: $\big\{ \min \sum_{\mathbf{a} \in \mathcal{A}} \mu_a \lambda_a : \sum_{\mathbf{a} \in \mathcal{A}} a_i \lambda_a \geq b_i, \forall i \in \{1, \ldots, n\}, \lambda_a \in \mathbb{R}_+, \forall \mathbf{a} \in \mathcal{A} \big\}$. By restricting the dual LP, our aggregation actually relaxes this master LP. Indeed, by adding linearity restrictions in the dual LP, one implicitly introduces artificial primal columns in this master LP. More exactly, the aggregated master model can implicitly use linear combinations of valid columns and *exchange vectors* associated to the new linearity restrictions.

Sec. 5.1 below shows how our linear aggregation yields implicit artificial columns that:

(i) preserve the feasibility of the resource constraint, *although*

(ii) the coefficients of the new columns can be fractional, and so, infeasible.

We further show (Sec. 5.2) that a more classical equality aggregation generate artificial columns that are infeasible with regards to *both* these criteria. As such, the master relaxation produced by equality aggregation is weaker than the one obtained by linear aggregation (it has more infeasible artificial columns).

5.1. The case of the linearity aggregation

Given the linear restriction $y_i = \alpha^j w_i + \beta^j,\ \forall i \in I^j$, we observe that any y_i with $i \in I^j$ can be expressed without the terms α^j and β^j, as a linear function of two variables $y^j_{\min}$ and $y^j_{\max}$ (associated to the elements of lowest and resp. largest resource consumption in I^j). Consider a selected column $\mathbf{a}$ with $a_i > 0$ for some $i \in I^j$. Since y_i can be written as a linear combination of $y^j_{\min}$ and $y^j_{\max}$, $\mathbf{a}$ is equivalent to a modified column $\hat{\mathbf{a}}$ in which the i^{th} term is replaced by a combination of $a^j_{\min}$ and $a^j_{\max}$; such a process is formalized through a *linearity-based exchange vector*, as described below, see (5.1). An interesting property of our linear aggregation is that the modified column cannot violate the resource constraints, although it can be infeasible because its coefficients can be fractional.

PROPOSITION 5.1. *Let $\mathbf{a} \in \mathbb{R}^n_+$ be a feasible solution of subproblem (2.2), and $\mathbf{e}$ be a linearity-based exchange vector. For any $\psi \in \mathbb{R}$, $\hat{\mathbf{a}} = \mathbf{a} + \psi\mathbf{e}$ verifies $C^- \le \hat{\mathbf{a}}^\top \mathbf{w} \le C^+$.*

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 $y^j_{\min}$ (resp. $y^j_{\max}$) is the first (resp. last) element of I^j, associated to values $w^j_{\min}$ (resp. $w^j_{\max}$) and $a^j_{\min}$ (resp. $a^j_{\max}$). We examine how other elements y_i with $i \in I^j$ can be written as a combination of $y^j_{\min}$ and $y^j_{\max}$. Using classical algebraic manipulations of $y^j_{\min} = \alpha^j w^j_{\min} + \beta^j$ and $y^j_{\max} = \alpha^j w^j_{\max} + \beta^j$, one can first determine $\alpha^j = \frac{y^j_{\max} - y^j_{\min}}{w^j_{\max} - w^j_{\min}}$, followed by $\beta^j = \frac{w^j_{\max} y^j_{\min} - w^j_{\min} y^j_{\max}}{w^j_{\max} - w^j_{\min}}$.

Replacing this in $y_i = \alpha^j w_i + \beta^j$ for $i \in I^j$, we obtain $y_i = \frac{w^j_{\max} - w_i}{w^j_{\max} - w^j_{\min}} y^j_{\min} + \frac{w_i - w^j_{\min}}{w^j_{\max} - w^j_{\min}} y^j_{\max}$, and so, the exchange process $\mathbf{a} + \psi\mathbf{e} \longrightarrow \hat{\mathbf{a}}$ over group j can be written as:

$$\begin{pmatrix} a^j_{\min} \\ \vdots \\ a_i \\ \vdots \\ a^j_{\max} \end{pmatrix} + \psi \begin{pmatrix} \frac{w^j_{\max} - w_i}{w^j_{\max} - w^j_{\min}} \\ \vdots \\ -1 \\ \vdots \\ \frac{w_i - w^j_{\min}}{w^j_{\max} - w^j_{\min}} \end{pmatrix} \longrightarrow \begin{pmatrix} a^j_{\min} + \psi \frac{w^j_{\max} - w_i}{w^j_{\max} - w^j_{\min}} \\ \vdots \\ a_i - \psi \\ \vdots \\ a^j_{\max} + \psi \frac{w_i - w^j_{\min}}{w^j_{\max} - w^j_{\min}} \end{pmatrix}, \tag{5.1}$$

where ψ can be positive or negative, which may respectively decrease or increase the i^{th} coefficient 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 $\mathbf{a} + \psi\mathbf{e} \geq \mathbf{0}$; 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 $\mathbf{a}$, the total resource consumption of these three elements is $a^j_{\min} w^j_{\min} + a_i w_i + a^j_{\max} w^j_{\max}$. In $\mathbf{a} + \psi\mathbf{e}$, this resource consumption becomes:

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

This simplifies to $a^j_{\min} w^j_{\min} + a_i w_i + a^j_{\max} w^j_{\max}$, which means that the resource consumption is the same in $\mathbf{a}$ and $\mathbf{a} + \psi\mathbf{e}$, *i.e.*, the initial resource consumption can *not* change through the exchange process (5.1). □

5.2. The case of the equality aggregation

Let us compare the linear aggregation with a simpler aggregation that only imposes equality constraints, *i.e.*, equivalent to 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 $y_i = y_{i'}$ would make a_i and $a_{i'}$ interchangeable in any column $\mathbf{a}$. This would lead to an *equality-based* exchange vector with two non-zero components at positions i and i', see other examples in [10]. The associated exchange process would be:

$$(..., a_i, ..., a_{i'}, ...)^\top + \psi \cdot (..., 1, ..., -1, ...)^\top \longrightarrow (..., a_i + \psi, ..., a_{i'} - \psi, ...)^\top, \ \forall \psi \in [-a_i, a_{i'}] \quad (5.2)$$

Proposition 5.2. *Given feasible configuration $\mathbf{a} \in \mathbb{R}^n_+$, an equality-based exchange vectors $\mathbf{e}$ can lead* (5.2) *to columns violating the resource constraint $C^- \leq (\mathbf{a} + \psi\mathbf{e})^\top \mathbf{w} \leq C^+$.*

Proof: It is sufficient to give an example of an artificial column that violates the resource constraints. Take $C^- = 8$, $C^+ = 10$, $I = \{1, 2\}$, $w_1 = 8$ and $w_2 = 3$. Column $\mathbf{a} = (0, 3)^\top$ is valid; by applying the exchange vector $\mathbf{e} = (1, -1)^\top$ with $\psi = 3$ in (5.2), one obtains artificial column $(3, 0)^\top$ with resource consumption 24. An artificial column that violates the minimum resource constraint can be produced by taking $\mathbf{a} = (1, 0)^\top$ and $\psi = -1$. □

6. Numerical Evaluation

We here perform an evaluation of 2IDA on three cutting-stock variants (Sec. 6.2, 6.3, and resp. 6.4) that cover different features of the general LP (2.1): different configuration costs μ_a, different demands, and two-sided limits C^- and C^+ on the total feasible consumption.

Table 1 General characteristics of the CSP instances considered in this paper.

Name	n	C	avg. demand $\mathbf{b}$	avg. $\mathbf{w}$ span	description
`wäscher`	57-239	10000	1	$[1, \frac{1}{2}C]$	17 hard `bin-packing` instances [20][a]
`hard–sch`	≈ 200	100000	$[1,3]$	$[\frac{20}{100}C, \frac{35}{100}C]$	the ten hardest `scholl` instances [17]
`vb50c1`	50	10000	$[50,100]$	$[1, \frac{3}{4}C]$	20 random instances [19][b]
`vb50c2`	50	10000	$[50,100]$	$[1, \frac{1}{2}C]$	20 random instances [19][b]
`vb50c3`	50	10000	$[50,100]$	$[1, \frac{1}{4}C]$	20 random instances [19][b]
`vb50c4`	50	10000	$[50,100]$	$[\frac{1}{10}C, \frac{1}{2}C]$	20 random instances [19][b]
`vb50c5`	50	10000	$[50,100]$	$[\frac{1}{10}C, \frac{1}{4}C]$	20 random instances [19][b]
`vb20`	20	10000	$[10,100]$	$[1, \frac{1}{2}C]$	25 random instances [19][b]
`vbInd`	[5,43]	[4096,200000]	$[1,300]$	$[\frac{1}{10}C, \frac{1}{2}C]$	17 industrial instances [19][b]
`m01`	100	100	1	$[1,100]$	1000 random instances [3];
`m20`	100	100	1	$[20,100]$	1000 random instances [3];
`m35`	100	100	1	$[35,100]$	1000 random instances [3];

[a] According to `http://or.dei.unibo.it/library/bpplib`, these instances are "the ones considered as the most difficult".
[b] These instances can be found in the archive `www.math.u-bordeaux1.fr/~fvanderb/data/randomCSPinstances.tar.Z` resp. in files `CSTR50b50c1*`, `CSTR50b50c2*`, `CSTR50b50c3*`, `CSTR50b50c4*`, `CSTR50b50c5*`, `CSTR20b50c*`, `CSTR*p*`.

We first map our general notation to the setting of a `Cutting-Stock Problem` (CSP). We consider a set $I = \{1, 2, \ldots, n\}$ of articles with weights (sizes) w_i that need to be cut (produced) b_i times from rolls (bins) of capacity C ($\forall i \in I$). A feasible solution is composed of *cut patterns* (configurations), *i.e.*, subsets of I with a maximum total weight of C.

The pure CSP simply asks to minimize the number of patterns (consider $\mu_a = 1$ for any pattern $\mathbf{a} \in \mathcal{A}$ in the models from Sec. 2). We use 12 CSP benchmark sets, each with up to 1000 individual instances (with a total number of 3169 instances). The technical characteristics of these CSP instances are provided in Tab. 1.

All times reported below have been obtained on a (Xeon E5-268) processor clocked at 2.50GHz by non-parallel programs compiled by `gnu g++` with code optimization option `-O3` on Linux Suse, kernel version 3.19 (Cplex version 12.6).

6.1. Implementation of 2IDA and of a Stand-Alone `Stabilized CG`

6.1.1. 2IDA with Intermediate Upper Bounds using Trust Region Stabilization

Unless otherwise specified, we use the 2IDA version with the upper bounds from Sec. 4.3. For each value of k, our method first generates a lower bound lb_k and then an upper bound ub_k. The upper bound solution for each k is found using a CG process. This CG process is stabilized by exploiting the lower bound solution $\mathbf{y}^*_\mathbf{k}$. First, recall (Sec. 4.3) that each ub_k is generated only by $\mathbf{y}^*_\mathbf{k}$-saturated constraints. Furthermore, we introduce a *trust region* method to stabilize the CG process around $\mathbf{y}^*_\mathbf{k}$, similarly to what is done in [4]. More exactly, this CG process keeps the current solution in a box around $\mathbf{y}^*_\mathbf{k}$ during the

first iterations. The length of the box edge is fixed at $4/\sum b_i$: this way, the upper bound is at maximum four units larger than lb_k. At the end of the CG process, we remove the box and obtain the actual upper bound ub_k.

For large values of k, the aggregated model tends to be as hard as the non-aggregated one. For best numerical results, 2IDA can eventually converge earlier by launching a final CG phase in any of the following conditions:

(a) k reaches a cut-off value $k_{\max}$ (this value is 10 for pure CSP in Section 6.2) *or*

(b) $\lceil \mathrm{lb}_k \rceil + 1 = \lceil \mathrm{ub}_k \rceil$ *or*

(c) the CPU time for computing lb_k is more than a third of the time for computing ub_k.

We do not stabilize the final CG phase, since it takes profit from all columns generated by the CG process, and so, it often proves to be quite short.

We use the split methods guided by reference solutions from Sec. 4.1.2. For the pure CSP, the first 2IDA iterations use a Dual-Feasible Function f (see [3]) as reference solution. As soon as 2IDA needs more intervals than the number of pieces of f, this reference solution is given by the upper bound ub_k. The exact details of this strategy are specified in App. A.1.

6.1.2. Stand-Alone `Stabilized CG` We compare 2IDA with a stand-alone `Stabilized CG` algorithm based on dynamic programming for pricing. The following techniques, inspired from [15], have been used to stabilize this stand-alone CG:

– *penalty functions* Instead of optimizing the classical dual objective value $\mathbf{b}^\top \mathbf{y}$, we add to this objective a step-wise linear penalty [15, eq. (23)]. For all instances `vb*` and `wäscher`, the penalty of $\mathbf{y}$ is given by a function $p\big(|\mathbf{y} - \mathbf{y}_{\text{last}}|_1\big)$ where $\mathbf{y}_{\text{last}}$ is the dual solution obtained at the previous iteration. For `vb*`, we define $p(x) = 0$ for $x < 0.002n$ or $p(x) = 1.5x$ otherwise. For `wäscher`, the penalty is milder: $p(x) = 0$ for $x < 0.01n$ or $p(x) = 0.5x$ otherwise. The approach is different for instances `m*` and `hard`: the penalty is 0 over all $\mathbf{y}$ inside the box $[\frac{\mathbf{y}_{\text{last}}}{10}, 2\mathbf{y}_{\text{last}}]$ and increases with slope 20 as $\mathbf{y}$ gets out of this box (proportional to the largest deviation from the box boundary).

– *dual solution smoothing* Instead of applying the pricing on the current dual solution $\mathbf{y}$, we apply it on the smoothed dual solution $\alpha\hat{\mathbf{y}} + (1-\alpha)\mathbf{y}$, where $\hat{\mathbf{y}}$ is the previous smoothed dual solution [15, eq. (23)]. We start with $\alpha = 0.2$ (or 0.5 for `vb*` and `wäscher`), but we gradually decrease it at each misprice (for each new constraint separating $\hat{\mathbf{y}}$ but *not* separating $\mathbf{y}$), eventually finishing at $\alpha = 0$ (misprice impossible).

We report in Tab. 2 the speedups obtained on the CSP by `Stabilized CG` compared to the regular CG in terms of the number of iterations, *i.e.*, $\frac{\text{iters}(CG\text{-NoStab})}{\text{iters}(CG\text{-Stab})}$. All further comparisons will be made with this stabilized version of CG.

Table 2 Reduction of the number of iterations $\left(\frac{\text{iters}(CG\text{-NoStab})}{\text{iters}(CG\text{-Stab})}\right)$ in CG due to the stabilization techniques implemented.

Instance	wäscher	hard–sch	vb50c1	vb50c2	vb50c3	vb50c4	vb50c5	vb20	vbInd	m01	m20	m35
Speedup	1.20	1.16	1.12	1.13	1.10	1.08	1.05	1.13	1.13	1.19	1.44	2.27

Regarding the dual bounds obtained along the search, 2IDA calculates them by solving to optimality the aggregated models, while `Stabilized CG` uses the Farley bound, which is a well-known customization of the Lagrangian bound [2, 11, 19].[1]

6.2. The Standard Cutting-Stock Problem

Table 3 The computing effort required by 2IDA and `Stabilized CG` to fully converge.

	Avg. CPU Time [s]		Avg. nr. of pricing calls		2IDA vs CG (CPU)		
Instance set	2IDA	CG	2IDA (lb_k, ub_k, final CG)	CG_{STAB}	$\prec$	$\simeq$	$\succ$
wäscher	78	117	479 (7+452+20)	480	16	1	0
hard-sch	31	77	227 (8+217+2)	663	10	0	0
vb50c1	5.2	13	124 (14+19+91)	112	17	1	2
vb50c2	22	35	177 (20+130+27)	197	19	1	0
vb50c3	15	29	82 (5+77+0)	144	20	0	0
vb50c4	16	23	181 (30+95+56)	185	20	0	0
vb50c5	11	22	73 (7+66+0)	152	20	0	0
vb20	2.1	3.2	52 (15+21+16)	56	22	0	3
vbInd	2.2	2.7	40 (12+26+2)	45	13	1	3
m01	0.11	0.09	203 (8+187+8)	161	574	42	384
m20	0.04	0.04	150 (8+119+23)	120	424	89	487
m35	0.02	0.01	107 (6+93+8)	59	36	25	939

Columns 2-3 and 4-5 report the average time and resp. the number of pricing calls. For 2IDA, this includes three types of pricing subproblems (indicated in parenthesis):
- the aggregated pricing sub-problems of the lower bounding routine (lb_k), included those obtained by lifting;
- the pricing calls for the upper bounding routine CG process (ub_k);
- the pricing calls of the final CG phase (final CG).

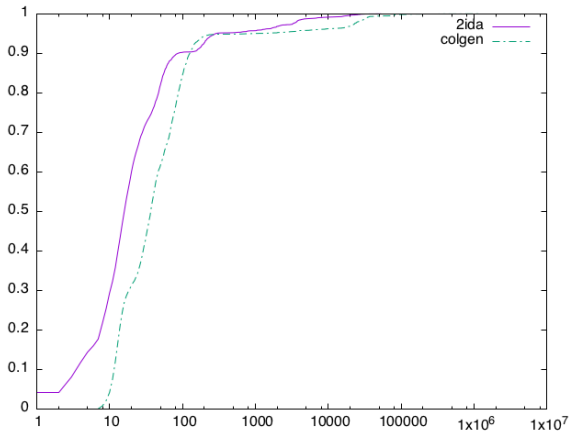
The last 3 columns "2IDA vs CG (CPU)" report how many times 2IDA needs less time than CG ($\prec$) ; needs a similar (difference $< 5\%$) time ($\simeq$); needs mores time than CG ($\succ$).

Tab. 3 shows summarized CSP results obtained by 2IDA and `Stabilized CG`. Results on individual instances are publicly available at `cedric.cnam.fr/~porumbed/csp/` or in the companion paper. Regarding the CPU time, Columns 2-3 show that 2IDA realizes a speed-up of more than 2.4 for `wäscher` and 1.5 for `hard-sch` compared to `Stabilized`

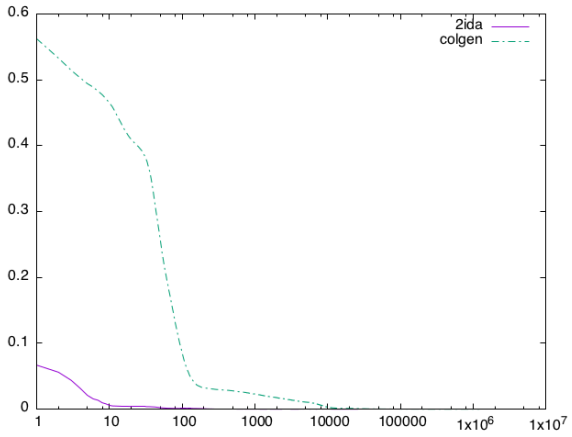
[1] The value of this bound is $\frac{\mathbf{b}^\top \mathbf{y}}{1-M_{\text{RedCst}}}$, where $\mathbf{y}$ and M_{RedCst} are the dual solution and resp. the minimum reduced cost at current iteration. If the costs μ_a of patterns $\mathbf{a} \in \mathcal{A}$ are not fixed to 1, the bound value becomes $\frac{\mathbf{b}^\top \mathbf{y}}{1-M_{\text{RedCst}} \cdot min_{\mathbf{a}\in\mathcal{A}}\mu_a}$.

`CG`. In terms of iterations (pricing calls), Columns 4-5 show that 2IDA has the potential to converge on the hardest instances using 2 or 3 times fewer columns (aggregated or non-aggregated). The last number in Column 4 also shows that the final CG phase of 2IDA is relatively short in general. In other words, the intermediate lower and upper bounds lb_k and ub_k are not very far from closing the gap before launching the final CG phase.

While 2IDA offers no clear advantage on the smallest instances `m01`, `m20`, `m35`, this is in part because these instances can be solved in less than 0.1 seconds and 2IDA involves more pre-processing that `Stabilized CG`. One should also notice that `Stabilized CG` is actually very well stabilized on these instances, *e.g.*, with a speed-up of 2.27 for the `m35` instances (recall Tab. 2).



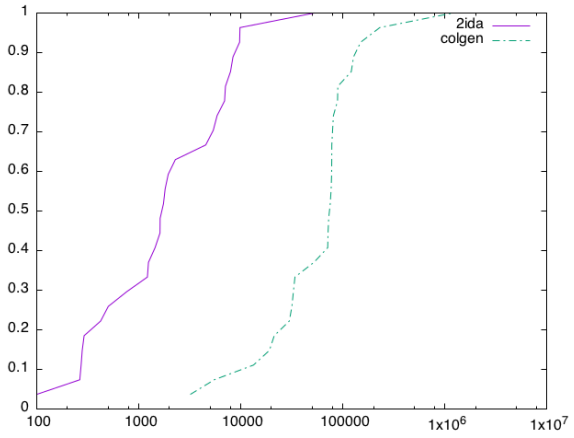
(a) Proportion of instances solved vs. time in milliseconds (log. scale)



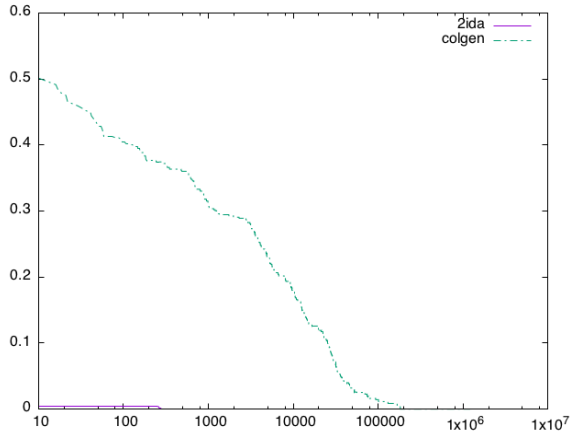
(b) Average relative gap between the dual bound and the optimum vs. time in milliseconds (log. scale)

Figure 4 Comparison of 2IDA and `Stabilized CG` for CSP on all instances.

Fig. 4a (resp. Fig. 5a) reports the percentage of all instances (resp. of all the hardest instances) that are solved along the time by 2IDA and `Stabilized CG`. This confirms that 2IDA solves the smallest instances quickly, and is also able to solve the most difficult ones before `Stabilized CG`. When restricting to the most difficult instances `wäscher` and `hard-sch` in Fig. 5a, it becomes clear that 2IDA outperforms `Stabilized CG`. Fig. 4b compares the relative gap between the optimum and the dual bounds produced by 2IDA and CG along the time, considering all instances. One can see that the `Stabilized CG` needs 1000 more time than 2IDA (on average) to obtain a relative gap close to 0.01. This makes 2IDA a choice candidate to obtain excellent dual bounds in a short time. Once again, restricting the experiments to the hardest instances (Fig. 5b) clearly shows that 2IDA outperforms `Stabilized CG` when the problem gets more difficult.



(a) Proportion of instances solved vs. time in milliseconds (log. scale)



(b) Average relative gap between the dual bound and the optimum vs. time in milliseconds (log. scale)

Figure 5 Comparison of 2IDA and `Stabilized CG` for CSP (hardest instances : `wäscher` and `hard-sch`)

6.3. The Multiple-Length Cutting Stock

The `Multiple-Length CSP` (`Mult Len`-CSP) is a CSP variant in which the patterns do *not* have all the same cost. We here consider two classes of instances: (i) one with two bin sizes $0.7C$ and C of costs 0.6 and resp. 1, and (ii) a second one in which we add a third bin size $0.2C$ of cost 0.1. In both cases, the cost μ_a of a pattern $\mathbf{a} \in \mathcal{A}$ is evaluated by the smallest bin that can hold $\mathbf{w}^\top \mathbf{a}$, *e.g.*, if $\mathbf{w}^\top \mathbf{a} = 0.43C$, then $\mu_a = 0.6$. A `Mult Len`-CSP particularity is that one *cannot* compute lower bounds using the Dual Feasible Functions that proved so effective in pure CSP [3]. Even the dual solution $y_i = \frac{w_i}{C}$ is not feasible here.

We consider the 2IDA implementation from Sec. 6.1.1, except that we used the group splitting based on upper bounds from Sec. 4.1.2 without Dual Feasible Functions (DFFs). As for the CSP, the iterative upper bounds ub_k can stop 2IDA sooner: since all considered costs μ_a are multiples of 0.1, 2IDA can stop as soon as $\lceil 10 \cdot \mathrm{lb}_k \rceil = \lceil 10 \cdot \mathrm{ub}_k \rceil$.

Tab. 4 and 5 report an execution of 2IDA and `Stabilized CG` resp. on the above two `Mult Len`-CSP classes of instances. They compare the CPU time used by 2IDA to find the lower bounds lb_1, $\mathrm{lb}_2, \ldots$ (*e.g.*, in Columns 2 or 5) with the time required by `Stabilized` CG to reach the same bound quality (*e.g.*, Columns 4 or 7). We only show the first instance of each set, but similar trends show up across most instances; summaries obtained from all instances are reported in Fig. 6, see also `cedric.cnam.fr/~porumbed/mlcsp/` or the companion paper.

These tables show that 2IDA can find high-quality lower bounds very rapidly: the 2IDA bounds for $k = 1$ could be generated 10 times more rapidly than by CG (compare Columns

Table 4 Results of 2IDA (fixing the number of iterations to $k_{\max} = 2$) and `Stabilized CC` on several instances of `Mult Len`-CSP with $\mu_{0.7C} = 0.6$ and $\mu_C = 1$.

Instance	k=1 2IDA		Time used by CG to reach	k=2 2IDA		Time used by CG to reach	OPT_{CG} $[T_s]$
	lb_1 $[T_s]$	ub_1 $[T_s]$	lagr. bnd. $\geq lb_1$	lb_2 $[T_s]$	opt $[T_s]$	lagr. bnd. $\geq lb_2$	
`wäscher-1`	24 [0.22]	112.6 [0.64]	21	24.1 [3.1]	24.1 [14]	21	24.1 [21]
`hard-sch-1`	47.5 [1.1]	198.6 [5.4]	60	47.7 [70]	51.5 [132]	60	51.5 [85]
`vb50c1-1`	802.7 [0.56]	2453.8 [1.1]	8.6	803.4 [3.2]	866.3 [4.3]	8.6	866.3 [10]
`vb50c2-1`	626.4 [0.85]	2488 [1.4]	13	626.7 [4.4]	672.3 [7.2]	13	672.3 [17]
`vb50c3-1`	282 [1.1]	2369 [2.6]	43	282 [9.5]	282 [20]	43	282 [43]
`vb50c4-1`	576.7 [0.69]	2444.3 [1.3]	18	576.9 [3.5]	579.6 [7.0]	19	579.6 [19]
`vb50c5-1`	337.7 [0.95]	2288.4 [1.9]	33	337.7 [9.2]	337.7 [14]	33	337.7 [33]
`vb20-1`	252.9 [0.22]	938.6 [0.41]	1.3	254.9 [1.7]	265.6 [1.8]	1.3	265.6 [1.3]
`vbInd30p0`	76.7 [0.22]	703 [0.5]	3.1	76.7 [1.7]	76.7 [2.5]	3.1	76.7 [3.1]
`m01-1`	41.7 [≈ 0]	93.6 [0.01]	0.1	41.7 [0.02]	49.3 [0.14]	0.1	49.3 [0.13]
`m02-1`	47.4 [≈ 0]	98.6 [0.01]	0.09	48.4 [0.02]	56.6 [0.1]	0.1	56.6 [0.11]
`m03-1`	58.3 [≈ 0]	100 [≈ 0]	0.03	62.6 [0.01]	73.9 [0.06]	0.03	73.9 [0.04]

For each k, we provide three fields:
– the lower bound lb_k and the cumulative CPU time at iteration k (Columns 2, 5),
– the upper bound and the cumulative CPU time at iteration k (Columns 3, 6),
– the CPU time needed by `Stabilized CC` to derive a Lagrangian bound of the same quality as lb_k (Columns 4, 7).

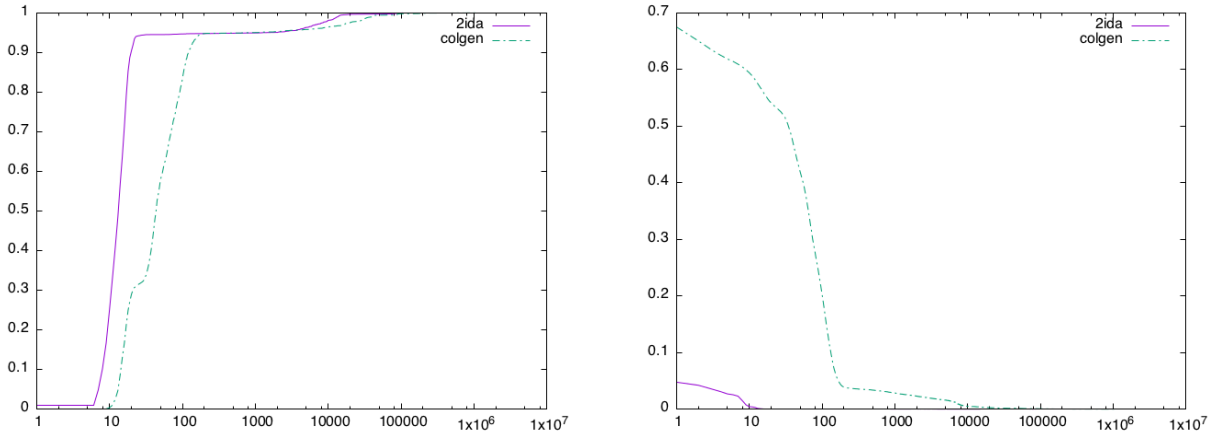
Table 5 Results of 2IDA (fixing the number of iterations to $k_{\max} = 3$) and `Stabilized CC` on `Mult Len`-CSP with $\mu_{0.2C} = 0.1$, $\mu_{0.7C} = 0.6$ and $\mu_C = 1$.

Instance	k=1 2IDA		equiv. CG	k=2 2IDA		equiv. CG	k=3 2IDA		equiv. CG	OPT_{CG} $[T_s]$
	lb_1 $[T_s]$	ub_1 $[T_s]$	T_s^{lagr}	lb_2 $[T_s]$	ub_2 $[T_s]$	T_s^{lagr}	lb_3 $[T_s]$	opt $[T_s]$	T_s^{lagr}	
`wäscher-1`	14.5 [0.16]	111.2 [0.94]	9.2	17.5 [5.7]	102.8 [6.8]	9.6	22.4 [11]	23.8 [21]	15	23.8 [16]
`hard-sch-1`	47.5 [0.97]	198.6 [5.4]	71	47.7 [70]	197.2 [74]	71	47.9 [75]	51.5 [136]	71	51.5 [85]
`vb50c1p-1`	538.9 [0.46]	2469.7 [0.94]	8.2	744 [4.5]	2457.3 [4.8]	9.5	764.7 [7.9]	866.3 [9.2]	9.5	866.3 [9.5]
`vb50c2p-1`	377.7 [0.75]	2489.3 [1.4]	13	633.6 [4.3]	2287 [5.4]	15	638.6 [6.1]	672.3 [9.0]	15	672.3 [16]
`vb50c3p-1`	166 [1.0]	2359.7 [2.7]	27	171.1 [13]	2359.7 [13]	30	171.1 [25]	238.8 [34]	30	238.8 [39]
`vb50c4p-1`	373.6 [0.66]	2500.1 [1.3]	16	543.8 [4.4]	2224.6 [5.5]	18	544.3 [14]	567.1 [17]	18	567.1 [18]
`vb50c5p-1`	200 [0.88]	2381.3 [2.2]	24	210.1 [6.6]	2157.3 [7.7]	25	210.2 [14]	277.1 [19]	25	277.1 [29]
`vb20-1`	172.2 [0.15]	931.7 [0.29]	1.0	225.8 [0.94]	855.2 [1.0]	1.3	243.1 [1.1]	255.4 [1.4]	1.3	255.4 [1.3]
`vbInd30p0`	47 [0.18]	672.1 [0.56]	2.1	50.2 [1.1]	625.6 [1.3]	2.2	51.2 [1.5]	63.8 [2.2]	2.2	63.8 [2.4]
`m01-1`	26.5 [≈ 0]	99.1 [0.01]	0.11	39.9 [0.02]	99.1 [0.03]	0.11	39.9 [0.04]	49.3 [0.15]	0.11	49.3 [0.14]
`m02-1`	47.4 [≈ 0]	99.6 [≈ 0]	0.09	48.4 [0.01]	99.6 [0.01]	0.09	48.4 [0.01]	56.6 [0.07]	0.09	56.6 [0.09]
`m03-1`	58.3 [≈ 0]	100 [≈ 0]	0.04	62.6 [0.01]	100 [0.02]	0.04	62.9 [0.02]	73.9 [0.06]	0.04	73.9 [0.04]

The interpretation of columns is the same as in Tab. 4. Results on more instances are publicly available on-line: `cedric.cnam.fr/~porumbed/mlcsp/`

2 and 4). To our knowledge, 2IDA is the only method that can produce lower bounds of this quality so rapidly for `Mult Len`-CSP, as DFFs can not be used.

Fig. 6 shows that 2IDA is generally faster than `Stabilized CG`, *e.g.*, it converges more rapidly for the instances needing more than 10000ms in Fig. 6a. The difference is more obvious in Fig. 6b, which compares the evolution of the gap between the current best lower bound and the optimal solution. This confirms that 2IDA can rapidly report nearly-



(a) Proportion of instances solved vs. time in milliseconds (log. scale)

(b) Average relative gap between the dual bound and the optimum vs. time in milliseconds (log. scale)

Figure 6 Comparison of 2IDA and `Stabilized CG` for `Mult Len`-CSP

optimal dual bounds; the most difficult task remains to prove the optimality of a feasible dual solution.

6.4. A Brief Study on Low-Waste Cutting Stock

The `Low Waste`-CSP imposes two limits C^- and C^+ on the total length of feasible patterns. One can see $C^+ = C$ as the fixed width of a roll and $C^+ - C^-$ as a maximum acceptable waste. Such constraints can arise in industry when it is not possible to recycle pieces of waste larger than $C^+ - C^-$. Overproduction is allowed to help feasibility.

Tab. 6 compares the 2IDA bounds with the Lagrangian CG bounds on `Low Waste`-CSP. We used a straightforward version of 2IDA with no intermediate upper bounds based on the basic dichotomic group split method from Sec. 4.1.1. Although we could speed-up the convergence by designing customized `Low Waste`-CSP split operators, the main goal of the paper is *not* to present a refined powerhouse for solving `Low Waste`-CSP, but to validate our generic aggregation approach for different resource-constrained problems.

Even with this simple setting, the 2IDA bounds clearly outperform the Lagrangian bounds computed by `Stabilized CG` within the same running time. More precisely, even the first 2IDA bounds for $k = 1$ (Column 3) are usually larger than the Lagrangian bounds reported after 2 or 3 times more computing time (Column 6).

6.5. General Experimental Conclusions

Let us conclude our experimental analysis with a few 2IDA trends. For the pure CSP, 2IDA produces high-quality solutions $\mathbf{y}^*_\mathbf{k}$ by exploiting the structure of known Dual Feasible

Table 6 `Low Waste-CSP` **results on all benchmarks that can not be solved in less than 1 second.**

Instance	max waste	k=1		k=2		$\mathrm{OPT}_{\mathrm{CG}}$
	$C^+ - C^-$	$\mathrm{lb}_1^{\mathrm{2IDA}}$ [T_s]	$\mathrm{lb}_{\text{CG-lagr}}[T_\mathrm{s}]$	$\mathrm{lb}_2^{\mathrm{2IDA}}$ [T_s]	$\mathrm{lb}_{\text{CG-lagr}}[T_\mathrm{s}]$	LW-CSP/CSP
`wäscher-1`	2	28 [0.06]	16 [0.08]	28 [0.65]	7 [0.66]	28/28
`wäscher-2`	2	23 [0.02]	6 [0.04]	23 [0.25]	13 [0.26]	24/23
`wäscher-3`	2	14 [0.02]	7 [0.02]	14 [0.13]	5 [0.14]	15/14
`hard-sch-1`	2	56 [0.22]	47 [0.23]	**59**[a] [5.6]	16 [5.7]	60/57
`hard-sch-2`	2	56 [0.15]	48 [0.19]	**58**[a] [7.7]	22 [7.7]	58/56
`hard-sch-3`	2	55 [0.2]	47 [0.23]	**57**[a] [5.4]	22 [5.4]	59/55
`vb50c1p-1`	3	**940**[a] [0.44]	851 [0.44]	**954**[a] [0.87]	CG ended [b]	1131/939
`vb50c1p-2`	3	896 [0.53]	542 [0.55]	897 [1.1]	697 [1.1]	938/898
`vb50c1p-3`	3	**939**[a][0.41]	681 [0.41]	**964**[a] [0.85]	CG ended[b]	1160/928
`vb50c2p-1`	3	731 [0.77]	136 [0.77]	731 [1.6]	705 [1.6]	737/737
`vb50c2p-2`	3	679 [0.65]	289 [0.67]	679 [1.2]	567 [1.2]	680/679
`vb50c2p-3`	3	560 [0.69]	284 [0.69]	560 [1.5]	507 [1.5]	560/560
`vb50c3p-1`	3	329 [1.0]	173 [1.1]	329 [3.4]	208 [3.4]	329/329
`vb50c3p-2`	3	280 [0.83]	143 [0.89]	280 [2.7]	82 [2.7]	280/280
`vb50c3p-3`	3	317 [0.82]	167 [0.89]	317 [2.4]	60 [2.4]	317/317
`vb50c4p-1`	3	673 [0.59]	471 [0.6]	673 [1.3]	612 [1.3]	673/673
`vb50c4p-2`	3	640 [0.46]	366 [0.47]	640 [0.93]	457 [0.94]	640/640
`vb50c4p-3`	3	774 [0.35]	530 [0.36]	774 [0.69]	648 [0.69]	827/775
`vb50c5p-1`	3	394 [0.76]	188 [0.78]	394 [2.0]	394 [2.0]	394/394
`vb50c5p-2`	3	408 [0.55]	128 [0.56]	408 [2.7]	CG ended[b]	408/408
`vb50c5p-3`	3	345 [0.92]	69 [0.93]	345 [1.6]	345 [1.7]	345/345

Columns 3 and 5 report the lower bound returned by 2IDA at the end of iterations 1 and resp. 2. Columns 4 and 6 resp. report the Lagrangian bound obtained by a stand-alone CG using a similar (slightly larger) CPU time. Last column indicates the CG optimum for LW-CSP and respectively for pure CSP.

[a] Remark that these 2IDA bounds are higher than the CG optimum $\mathrm{OPT}_{\mathrm{CG}}$[CSP] for standard CSP from the last column. Standard lower bounds for CSP (*e.g.*, the DFFs) could *never* produce lower bounds of such quality.
[b] In these exceptional cases, the CG fully converged before the end of step $k = 2$ of 2IDA.

Functions (DFFs). In fact, $\mathbf{y}^*_\mathbf{k}$ is at least as good as the solution related to the best DFF that is linear over the same k intervals, see App. A.1.2. However, for the multiple-length CSP, we cannot use DFFs and this does not prevent 2IDA to find high quality dual solutions for $k = 1$ or $k = 2$. This shows that 2IDA does not necessarily rely too much on the quality of an input initial dual solution.

Our method needs three types of sub-problems : one to generate a lower bound solution $\mathbf{y}^*_\mathbf{k}$, one for the intermediate upper bound ub_k, and one traditional CG subproblem for the final CG phase. Only the last one is as difficult as a classical CG pricing. As such, even when 2IDA uses more columns that CG, it may converge in a faster way. The incremental construction of $\mathcal{P}_{k+1}$ from lifted $\mathcal{P}_k$ constraints (Sec. 4.2) reduces considerably the time needed to optimize $\mathcal{P}_{k+1}$ and this is a key technique for fast lower-bounding in 2IDA.

Convergence without upper bounds remains rather difficult. The most efficient way to converge is to intertwine the aggregation approach with a CG process for each k. However, this CG process does exploit information obtained by aggregation: the fact that $\mathbf{y}^*_\mathbf{k}$ belongs to the boundary of the dual polytope $\mathcal{P}$, proved to be very useful in stabilizing the CG

process. The source of stabilisation is twofold: (i) we use only $\mathbf{y_k^*}$-saturated constraints (Sec. 4.3) and (ii) the first iterations of this CG process keep the dual solution in a box around $\mathbf{y_k^*}$ (Sec. 6.1.1). Even when the convergence is eventually ensured by launching a final CG phase, this final CG is generally quite short (see last number in Column 4, Tab. 3).

7. Conclusions and Perspectives

We described an aggregation method that computes a sequence of dual bounds in column generation models. The approach is designed for covering problems in which the feasibility and the cost of configurations mainly depend on a unique resource. It relies on an expected linear correlation between the dual values y_i and the resource consumptions w_i of the elements. At each iteration k, it computes an optimal solution of a restricted dual LP in which the dual values have to follow an affine function over k pre-defined intervals. The slope and y-intercepts of these affine functions are the variables of this new smaller dual LP.

Besides producing a lower bound, the proposed method can also compute an upper bound for each k, by intertwining the aggregation with a column generation process. The upper bound solution is first sought in the proximity of the feasible dual solution constructed by aggregation. We use different computational techniques to incrementally calculate each new lower or upper bound by re-using already-generated constraints; experiments show that the resulting method has a much higher speed-up potential than stabilized column generation for several versions of the cutting-stock problem.

Further work will focus on other ideas for anticipating the form of optimal dual solutions. If an aggregation method can rapidly generate nearly-optimal dual feasible solutions, this can be useful both for producing high-quality lower bounds *and* for stabilizing a column generation process (e.g., using trust region methods). The optimality can thus be ensured through the convergence of both dual feasible and dual infeasible solutions. In light of these observations, it will be interesting to determine strategies to aggregate in the case of multiple resources (e.g. demands and distances in vehicle routing) or consider configuration costs determined by both the resource consumption and external factors.

References

[1] P. Benchimol, G. Desaulniers, and J. Desrosiers. Stabilized dynamic constraint aggregation for solving set partitioning problems. European Journal of Operational Research, 223(2):360–371, 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] T. Gschwind and S. Irnich. Dual inequalities for stabilized column generation. Discussion Paper 1407, Gutenberg School of Management and Economics, 2014.

[10] D. Liang and W. E. Wilhelm. A generalization of column generation to accelerate convergence. Mathematical Programming A, 122:349–378, 2010.

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

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

[13] 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.

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

[15] A. Pessoa, R. Sadykov, E. Uchoa, and F. Vanderbeck. In-out separation and column generation stabilization by dual price smoothing. In 12th International Symposium on Experimental Algorithms, pages 354–365. Springer, 2013.

[16] D. Porumbel. Ray projection for optimizing polytopes with prohibitively many constraints in set-covering column generation. *Mathematical Programming*, 155(1):147–197, 2016.

[17] A. Scholl, R. Klein, and C. Jurgens. BISON: a fast hybrid procedure for exactly solving the one-dimensional bin-packing problem. Computers and Operations Research, 24:627–645, 1997.

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

[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] G. Wäscher and T. Gau. Heuristics for the integer one-dimensional cutting stock problem: A computational study. Operations-Research-Spektrum, 18(3):131–144, 1996.

Appendix A: On the update from P_k to P_{k+1}

While the exact incremental calculations used to update P_k to P_{k+1} have a limited theoretical impact on our study, they are essential for the practical speed of 2IDA and they deserve further analysis. We here provide the pseudo-code of the routine from Sec. 4.2. Recall that it starts building $\mathcal{P}_{k+1}$ by lifting $\mathcal{P}_k$ constraints.

Algorithm 2: A Two-Step CG for $\texttt{DCvr}(\mathcal{P}_{k+1})$: optimize over $\mathcal{P}'^{y}_{k+1} \supset \mathcal{P}^{y}_{k}$, then over $\mathcal{P}^{y}_{k+1}$

Data: Optimal solution $(\alpha^1, \ldots, \alpha^k, \beta^1, \ldots, \beta^k)$ of $\mathcal{P}_k$ equivalent to $\mathbf{y}^*_{\mathbf{k}} \in \mathcal{P}^{y}_{k}$

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

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

- Lift aggregated solution $(\alpha^1, \ldots, \alpha^k, \beta^1, \ldots, \beta^k) \in \mathcal{P}_k$ 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 two groups j_1, j_2 ;*
  - keep unchanged the values α^j and β^j for all $j \neq j^*$;
- **repeat**
  - **for** $\mathbf{a} \in \mathcal{A}_k : \mathbf{a}^\top \mathbf{y}^*_{\mathbf{k}} = \mu_a$ **do**
    - given current $[\alpha\ \beta]$, solve the aggregated multiple-choice pricing (Sec. 3.3.2) with 2 levels (j_1,j_2) and capacities $C^- - \sum_{j \neq j^*} c^j_a$ and $C^+ - \sum_{j \neq j^*} c^j_a$ to lift $\mathbf{a}$ to $\mathbf{a}' \in \mathcal{A}_{k+1}$;
    - $\mathcal{A}'_{k+1} \leftarrow \mathcal{A}'_{k+1} \cup \{\mathbf{a}'\}$;
  - optimize over current $\mathcal{P}'_{k+1}$ described by configurations $\mathcal{A}'_{k+1}$ only
  - update $\text{OPT}(\texttt{DCvr}(P'_{k+1}))$ and the current dual solution $[\alpha, \beta]$
- **until** *no configuration* $\mathbf{a}'$ *of negative reduced cost can be found*;

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

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

- **repeat**
  - given current $[\alpha, \beta]$, solve the aggregated multiple-choice pricing (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** $\text{OPT}(\texttt{DCvr}(\mathcal{P}_{k+1}))$;

Observe that Step (1) of Algorithm 2 generates new configurations $\mathbf{a}'$ by reusing coefficients (c_a^j, N_a^j) from non-split groups $j \neq j^*$ of already-generated configurations $\mathbf{a} \in \mathcal{A}_k$. For any such $\mathbf{a}$, the lifted configuration $\mathbf{a}'$ is computed by determining the coefficients of the new sub-groups j_1 and j_2. This is carried out using a dynamic programm significantly faster than the original one from Section 3.3.2. Besides only using two decision levels (j_1 and j_2), it also uses a residual capacity reduced from C^+ to $C^+ - \sum_{j\in\{1,\dots,k\}-\{j^*\}} c_a^j$ (it removes the amount consumed by lifted values from $\mathbf{a} \in \mathcal{A}_k$).

A.1. Customizing the Group Split Methods

While the group split operator has no great impact in theory, in practice it is important to take the best split decisions. In Sec. 4.1, we briefly presented:

1. A dichotomic split operator (Sec. 4.1.1) that only maintains the regularity of the generated groups.
2. A splitting strategy (Sec. 4.1.2) that aims at making the current polytope cover a *reference solution* of better quality than the current lower bound.

We here develop this latter strategy in greater detail, using as *reference solution* the 2IDA upper bound in Sec. A.1.1 and resp. a dual solution constructed from dual-feasible functions (DFFs) in Sec. A.1.2 (for CSP and `Low Waste`-CSP only). These DFFs are well-acknowledged for their speed in pure CSP [3, 12], *i.e.*, they only require applying a (often piece-wise linear) function on the item weights.

A.1.1. Guiding the Split Operator using 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 remaining $n_{j^*} - i^*$ elements to the second (sub-)group. These decisions rely on a comparison of the current optimal solution $\mathbf{y}^*_{\mathbf{k}}$ of $\mathcal{P}^y_k$ to an outside reference solution. This reference is given by an upper bound solution $\mathbf{y}^{\mathbf{u}}$ (see Sec. 4.3) that indicates a direction of evolution that $\mathbf{y}^*_{\mathbf{k}}$ can follow to reach a better $\mathbf{y}^*_{\mathbf{k+1}}$.

The main idea is that that $\mathbf{y}^*_{\mathbf{k}} \to \mathbf{y}^{\mathbf{u}}$ is an improving open direction (see Def. 4.1), *i.e.*, there is no legitimate $\mathcal{P}$ constraint that can block a (sufficiently small) advance from $\mathbf{y}^*_{\mathbf{k}}$ to $\mathbf{y}^{\mathbf{u}}$. We need to identify and break aggregation restrictions that do block such advances. This can be done by evaluating the difference between $\mathbf{y}^*_{\mathbf{k}}$ and $\mathbf{y}^{\mathbf{u}}$ over segments $[i_1, i_2] = \{i_1, i_1+1, \dots i_2\}$ of each group I^j (we only use groups with continuous consecutive indices). More exactly, we define the operator $\Delta_j(i_1, i_2) = \sum_{i=i_1}^{i_2} b_i \cdot (\mathbf{y}^{\mathbf{u}} - \mathbf{y}^*_{\mathbf{k}})_i$.

Alg. 3 gives the pseudo-code of this split operator. We describe it on the case $\Delta_j(1, n_j) > 0$, *i.e.*, the advance $\mathbf{y}^*_{\mathbf{k}} \to \mathbf{y}^{\mathbf{u}}$ generates a positive trend ("push") on the elements of group j. The negative case $\Delta_j(1, n_j) \leq 0$ is symmetric and can be reduced to the positive with the simple inversion from Lines 2-4. If $\Delta_j(i_a, i_b)$ were strictly positive for all $i_a, i_b \in \{1, 2, \dots n_j\}$, *no* linearity constraint could stop the increasing trend $\mathbf{y}^*_{\mathbf{k}} \to \mathbf{y}^{\mathbf{u}}$ over group j. The goal is then to break groups j such that the trend $\mathbf{y}^*_{\mathbf{k}} \to \mathbf{y}^{\mathbf{u}}$ is positive over some $[1, i_a]$, negative over some $[i_a, i_b]$ and (possibly) positive over $[i_b, n_j]$.

Technically, Alg. 3 determines $i_a = \min\{i \in \{1,\dots,n_j\} :\ \Delta_j(1, i-1) \geq 0, \Delta_j(i, i) < 0\}$ and $i_b = \max\{i \in \{i_a,\dots,n_j\} :\ \Delta_j(i_a, i) < 0\}$. The split point $i^*(j)$ is either $i_a - 1$ (to cut group j into intervals $[1, i_a - 1]$ and $[i_a, n_j]$) or i_b. If $\Delta_j(1, i_a - 1) > \Delta_j(i_b + 1, n_j)$, we use $i^*(j) = i_a - 1$ and otherwise we use $i^*(j) = i_b$.

The interest in splitting j at $i^*(j)$ is quantified by a heuristic score $h(j)$ initially defined by $\max\big(\Delta_j(1, i_a - 1), \Delta_j(i_b + 1, n_j)\big)$. Then, we multiply $h(j)$ by the weight spread $w^j_{\max} - w^j_{\min}$, so as to discourage splitting

Algorithm 3: Group Split Operator 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

1 **for** $j \leftarrow 1$ **to** k **do**
2 **if** $\Delta_j(1, n_j) < 0$ **then**
3 $\mathbf{y^*_k} \leftarrow -\mathbf{y^*_k}$ *// simple inversion to reduce the*
4 $\mathbf{y^u} \leftarrow -\mathbf{y^u}$ *// negative case to a positive case*
5 $i_a \leftarrow \min\left\{i \in \{1, \ldots, n_j\} :\ \Delta_j(1, i-1) \geq 0, \Delta_j(i, i) < 0\right\}$
6 $i_b \leftarrow \max\left\{i \in \{i_a, \ldots, n_j\} :\ \Delta_j(i_a, i) < 0\right\}$
7 **if** $\Delta_j(1, i_a - 1) > \Delta_j(i_b + 1, n_j)$ **then**
8 $i^*(j) \leftarrow i_a - 1$ *// Choose sub-groups* $[1, i_a - 1]$ *and* $[i_a, n_j]$ *for group* j
9 $h(j) \leftarrow \Delta_j(1, i_a - 1)$
10 **else**
11 $i^*(j) \leftarrow i_b$ *// Choose sub-groups* $[1, i_b]$ *and* $[i_b + 1, n_j]$ *for group* j
12 $h(j) \leftarrow \Delta_j(i_b + 1, n_j)$
13 $h(j) \leftarrow h(j) \cdot (w^j_{\max} - w^j_{\min})$ *// Discourage splitting groups with similar weights*
14 **if** $j = 1$ *or* $j = k$ **then** $h(j) \leftarrow h(j) \cdot 2$ *// Encourage splitting extremal groups*
15 **if** $h(j) > h(j^*)$ **then** $j^* \leftarrow j$ *// Initially,* j^* *was* 0 *and* $h(j^*)$ *was* $-\infty$
16 **return** $(j^*, i^*(j^*))$

groups with similar weights (Line 13). We finally multiply by 2 the interest of splitting extremal groups 1 and k (Line 14), because the smallest and the largest items can have a higher 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 way all other groups are determined.

A.1.2. Guiding the Split Operator using Dual-Feasible Functions We recall that $f : [0, C] \to [0, 1]$ is a *dual-feasible function* (DFF) if and only if

$$\textstyle\sum_{i \in I} a_i w_i \leq C \implies \sum_{i \in I} a_i f(w_i) \leq 1$$

holds for any index set I, any $a_i \in \mathbb{Z}^+$ and $w_i > 0$ ($\forall i \in I$). This ensures that a dual solution $y_i = f(w_i)$ is feasible in CSP or `Low Waste`-CSP, but not in `Mult Len`-CSP (because the patterns do not have all the same cost 1).

All classical DFFs surveyed in [3] have a piece-wise linear form. A well-known 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 functions. To guide the split operator using high-quality reference solutions, we always choose from [3] the DFF f with 10 intervals (pieces) that yields the highest objective value.

The proposed DFF-based split operator identifies $k_D \leq 10$ intervals of $[0, C]$ before starting the construction of $\mathcal{P}_k$,*i.e.*, these k_D intervals are chosen so that f is linear over each of them. When 2IDA reaches iteration $k = k_D$, the polytope $\mathcal{P}^y_k$ has to include the solution $y_i = f(w_i)$, because f is linear over all groups of $\mathcal{P}^y_k$. As such, the 2IDA bound $\mathrm{lb}_k = \mathrm{OPT}(\texttt{DCvr}(\mathcal{P}_k))$ dominates the DFF bound $\sum_{i \in I} b_i f(w_i)$ associated to f.