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

A Numerically-Exact Algorithm for the Bin Packing Problem

Roberto Baldacci
Engineering Management and Decision Sciences, College of Science and Engineering, Hamad Bin Khalifa University, Doha P.O. Box 34110, Qatar

Stefano Coniglio
University of Southampton, School of Mathematical Sciences, University Road, SO17 1BJ, Southampton, United Kingdom

Jean-François Cordeau
Chair in Logistics and Transportation, HEC Montréal, 3000 chemin de la Côte-Sainte-Catherine, Montréal H3T 2A7, Canada

Fabio Furini
Department of Computer, Control, and Management Engineering "Antonio Ruberti", Sapienza University, 00185 Roma, Italy

We propose a numerically-exact algorithm for solving the Bin Packing Problem (BPP) based on a branch-price-and-cut framework combined with a pattern-enumeration method. Key to the algorithm is a novel technique for the computation of numerically-safe dual bounds for the widely-adopted set covering reformulation of the BPP (tightened with additional valid inequalities) with a precision that is higher than the one of general-purpose floating-point solvers. Our branch-price-and-cut algorithm also relies on an exact integer (fixed-point) label setting algorithm for solving the pricing problem associated with the tightened set covering formulation.

To the best of our knowledge, ours is the first algorithm for the BPP that can be proven to be numerically exact. Extensive computational results on instances affected by notorious numerical difficulties, those of the Augmented Non-IRUP (ANI) class, show that our exact algorithm outperforms the state-of-the-art not numerically-exact algorithms based on branch-and-cut-and-price that rely on a set-covering formulation of the BPP.

Key words: bin packing, numerical precision, branch-price-and-cut, dynamic programming.
History: August 25, 2022

1. Introduction

Given a set N of n items, a positive integer weight w_j associated with each item $j \in N$, and an unlimited number of identical bins of integer capacity W, the *Bin Packing Problem* (BPP) asks for finding the minimum number of bins that are needed to pack all the items. The BPP is one of the fundamental problems in Combinatorial Optimization and Operations Research and, while it has been extensively studied for decades, it is still very challenging to solve to optimality. The BPP occurs in countless areas ranging from job scheduling to logistics and in very diverse applications such as truck or container loading with weight capacity constraints, digital content management in cloud storage, and event seating. For more details, we refer the reader to the survey by Delorme et al. (2016).

Most of the state-of-the-art algorithms for the BPP are based on solving a Set Covering (SC) formulation featuring an exponential number of variables, each associated with a subset of items, called a *pattern*, of total weight no larger than W. Letting $\mathscr{P} = \left\{ S \subseteq N : \sum_{j \in S} w_j \leq W \right\}$ be the collection of all patterns and letting the binary variable ξ_S be equal to 1 if and only if pattern $S \in \mathscr{P}$ is featured in the solution, the SC formulation reads

$$\text{(SC)} \qquad \min \sum_{S \in \mathscr{P}} \xi_S \tag{1a}$$

$$\text{s.t.} \sum_{S \in \mathscr{P}(j)} \xi_S \geq 1 \qquad j \in N \tag{1b}$$

$$\xi_S \in \{0,1\} \qquad S \in \mathscr{P}, \tag{1c}$$

where $\mathscr{P}(j) \subset \mathscr{P}$ is the collection of patterns containing item $j \in N$.

Let LSC be the Linear Programming (LP) relaxation of the SC formulation. The problem underlying this relaxation is usually called the Fractional Bin Packing Problem (FBPP); see, e.g., Karmarkar and Karp (1982), Eisenbrand et al. (2013), Caprara et al. (2015), Coniglio et al. (2019). Let $z(\text{LSC})$ be the optimal solution value of LSC and let $z(\text{BPP})$ be the optimal solution value of the BPP. Any BPP instance such that $z(\text{BPP}) = \lceil z(\text{LSC}) \rceil$ is said to enjoy the *Integer Round-Up Property* (IRUP) (Kartak et al. 2015, Delorme et al. 2016). While this property holds for many BPP instances, several non-IRUP instances are known in the literature (Caprara et al. 2015). The *Modified Integer Round-Up Property* (MIRUP) is a weaker version of the IRUP that holds if the difference between $z(\text{BPP})$ and $\lceil z(\text{LSC}) \rceil$ is no larger than 1 (i.e., if $\lceil z(\text{LSC}) \rceil \leq z(\text{BPP}) \leq \lceil z(\text{LSC}) \rceil + 1$). The MIRUP has been proven to hold when the number of different item sizes is no larger than 7 (see Eisenbrand et al. (2013)), and it is conjectured to hold in general (Scheithauer and Terno 1997). While, on the IRUP instances, computing $z(\text{LSC})$ suffices to certify (or disprove) the optimality of a given solution, non-IRUP instances require more sophisticated methods to produce a strong bound to do so. This situation is exacerbated by the fact that classical approaches for solving the SC formulation of the BPP are known to suffer from numerical errors on certain classes of non-IRUP instances. The Augmented Non-IRUP (ANI) instances proposed by Delorme et al. (2016) are one such class. As noted by Pessoa et al. (2020, 2021), these numerical errors may lead to computing incorrect bounds. Such an event can be catastrophic for the exactness of the solution algorithm, as it may lead to constructing an invalid enumeration tree as a result of the application of invalid pruning operations.

In the majority of the state-of-the-art algorithms for the BPP, the SC formulation is solved with a Branch-Price-and-Cut (BPC) algorithm as originally proposed by Vance et al. (1994). At each node of the BPC tree, the LP relaxation is solved by a Column-and-Row-Generation (CRG) method following the original work of Gilmore and Gomory (1961). In it, the patterns are generated by solving a pricing problem which, at the root node of the BPC tree, coincides with an instance of the Knapsack Problem (KP). With only a few exceptions (see Belov and Scheithauer (2006), Pessoa et al. (2020, 2021)), branching is carried out following the classical rule proopsed by Ryan and Foster (1981), according to which two items are forced to be packed into the same bin or into different ones. A relatively new addition to the state of the art is strengthening the SC formulation with additional *master cuts* such as the *subset-row inequalities of cardinality three* (SR3), a family of rank-1 Chvátal-Gomory cuts first proposed by Jepsen et al. (2008) that are valid for set covering and set partitioning problems and that were introduced into the BPP literature by Wei et al. (2020). Such cuts are valid also for set covering problems such as (1) as, in it, Constraints (1b) are tight w.l.o.g. in any optimal solution. Another new addition to the state of the art is *pattern enumeration*. Originally proposed by Baldacci et al. (2008) for the Vehicle Routing Problem (VRP) and used in the context of the BPP by Pessoa et al. (2021), the technique coincides with enumerating all the patterns whose reduced cost is no larger than the gap between the value of the incumbent and $z(\text{LSC})$ (possibly strengthened by master cuts). Such an enumeration leads to building a *reduced SC problem* which can then be solved to optimality by a branch-and-bound method without generating any futher patterns. For more details on state-of-the-art BPC methods for the BPP, we refer the reader to Wei et al. (2020) for the `EXM` algorithmn and to Pessoa et al. (2020, 2021) for the general-purpose BPC algorithm `VRPSolver`, which can tackle variants of the VRP, of which the BPP is a special case. For another family of methods (the most recent one being algorithm `NF-F`) we refer the reader to de Lima et al. (2022). Such methods rely on a *network-flow framework* designed to solve the integer counterpart to a Dantzig-Wolfe decomposition via a path-flow and arc-flow model, and can be adapted to solve instances of the BPP. They rely on arc-flow formulations of pseudo-polynomial size which enjoy strong linear relaxations. Key to the efficiency of the method is a novel asymmetric branching scheme where a collection of smaller arc-flow subproblems are solved by a general Mixed Integer Linear Programming (MILP) solver. Branching is performed by selecting a subset of arcs and by either imposing all of them to be absent from the solution or by imposing that at least one be present in it.

Crucially, all the state-of-the-art algorithms we mentioned (`EXM`, `VRPsolver`, and `NF-F`) rely either on numerically-unsafe bounds obtained by a floating-point LP solver combined with a floating-point pricing algorithm (this is the case of `EXM`) or on the adoption of a floating-point MILP solver in a key step of the algorithm (this is the case of both `VRPsolver` and `NF-F`: in the former, the reduced SC problem is solved as a MILP with `Cplex`, whereas in the latter the left child node generated after a branching operation is always solved as an arc-flow MILP formulation with `Gurobi`). As a consequence, none of them is guaranteed to produce numerically-exact solutions whose enumeration tree constitutes a mathematically exact proof of optimality of the solution the methods find.

1.1. Review on numerically-safe approaches

The solution methods presented in the literature to avoid numerical inaccuracies inherent in the floating-point computations are based on a *pure rational* approach, a *safe floating point* approach, or a *hybrid* approach. In addition, the methods can also be classified based on the type of mathematical formulations addressed: general compact Mixed-integer linear programming (MILP) models or extended models, the latter characterized by an exponential number of variables. In the pure rational approach (Applegate et al. 2007, Wunderling 1996, Gleixner et al. 2012, 2016), all arithmetic operations are performed over rational numbers. As a result of this line of research, rational solvers such as `QSopt_ex` (Applegate et al. 2007) and `SoPlex` (Wunderling 1996, Gleixner et al. 2012, 2016) are available for the exact solution of LP problems. Regarding *safe floating point* approaches dealing with compact MILP models, Cook et al. (2009), Cornuéjols et al. (2013), and Fukasawa and Goycoolea (2011) addressed numerical safety in the context of cutting plane generation, whereas Steffy and Wolter (2012) and Neumaier and Shcherbina (2004) tackled the issue of obtaining safe bounds for MILPs. The works of Cook et al. (2011, 2013) deal with the design of full exact MILP solvers, where hybrid approaches combining exact rational LP solvers and safe floating point dual bounds are used. To the best of our knowledge, the two only works addressing the computation of safe dual bounds in the context of extended models were proposed by Held et al. (2012) and Fukasawa and Poirrier (2017). The work of Held et al. (2012) introduces numerically safe bounds within a column generation based framework in the context of the graph coloring problem. Fukasawa and Poirrier (2017) introduce five methods to obtain safe dual bounds for the LP relaxations of the Capacitated VRP (CVRP). The first method is based on the scaling approach proposed by Held et al. (2012). The next three methods are derived from ideas by Applegate et al. (2007) for the traveling salesman problem and adapted to accommodate a column generation scheme for the CVRP. The last method is based on a specific Lagrangian relaxation of the CVRP formulation. Both Held et al. (2012) and Fukasawa and Poirrier (2017) also discussed safe floating point implementations of the dynamic programming procedure employed in the pricing step to derive numerically-exact pricing algorithms.

Even if we restricted ourselves to a BPC method designed to employ the scaling technique of Held et al. (2012), as done by Pessoa et al. (2021) (see Proposition 2 in this paper for a generalization), we would obtain bounds that are worse than their infinite-precision counterpart by an additive error proportional to the number of variables and the fixed-point precision of the pricing algorithm—see Equation (2) in Held et al. (2012). In some instances, such an error can be non-negligible, and it can lead to prohibitively large search trees. This is, in particular, the case of the ANI instances, where (as we will show in this paper) having access to a numerically safe dual bound obtained with a precision higher than the one offered by commercial LP solvers can lead to enormous computational advantages.

1.2. Contributions of this paper

In this paper, we propose a novel technique for the computation of *numerically safe* LP bounds within an *extended numerical precision* higher than the one of state-of-the-art commercial solvers. The contributions of this paper can be summarized as follows:

- We present the first numerically-exact algorithm for the BPP.
- We propose a technique based on the combined adoption of a rational (infinite-precision) LP solver (`SoPlex`, Gleixner et al. (2016)) with a faster, floating-point one (`Gurobi`) and on an exact integer (fixed-point) pricing algorithm that we develop *ex novo* for this work. To the best of our knowledge, rational solvers have never been successfully used before in the context of a column generation method, and ours is the first numerically-exact algorithm to be proposed in the literature for solving the BPP.
- We design a new exact integer (fixed-point) pricing algorithm which relies on, among other features, a new fathoming rule; the algorithm is designed in such a way that it can also perform *pattern enumeration*; differently from not numerically-safe approaches such as the one of Pessoa et al. (2021), after the enumeration phase we solve the *reduced SC problem* to optimality by a further call to a simplified version of our BPC algorithm in which the pricing algorithm is disabled, thus guaranteeing the lack of numerical errors.
- Thanks to extensive computational results, we validate the effectiveness of our numerically-exact algorithm as well as the relevance of its components on BPP instances affected by notorious numerical difficulties.
- The framework on which our algorithm is based can be of interest for finding numerically-exact solutions to a large array of combinatorial optimization problems that are typically tackled via a BPC method featuring a set covering/partitioning problem as the master problem.

The remainder of the paper is organized as follows. In Section 2, we introduce the BPC method that forms the basis of our algorithm. Section 3 presents the techniques we propose for the computation of extended-precision numerically safe dual bounds and describes the corresponding algorithm based on CRG. Section 4 describes our exact integer (fixed-point) label-setting algorithm designed to solve the pricing problem and illustrates how it can be tailored to perform pattern enumeration. Section 5 presents and analyzes the results of our algorithm on a set of BPP instances from the literature. Section 6 concludes the paper and indicates future research directions.

2. Branch-price-and-cut method

We outline in this section the BPC method that forms the basis for our numerically-exact algorithm.

Let $\mathscr{T} = \{T \subseteq N : |T| = 3\}$ be the collection of all triplets of items in N and, for each $T \in \mathscr{T}$, let $\mathscr{P}(T) = \{S \in \mathscr{P} : |S \cap T| \geq 2\}$ be the subset of patterns with at least two items in T. The SR3 constraints read

$$\text{(SR3)} \qquad \sum_{S \in \mathscr{P}(T)} \xi_S \leq 1, \quad T \in \mathscr{T}.$$

We refer to LSC (the LP relaxation of the SC formulation) tightened by these constraints as LSCT, where “T” stands for “triplets”, and call $z(\text{LSCT})$ its optimal solution value.

The value of $z(\text{LSCT})$ is computed by a CRG method. At each iteration, the restricted master problem (RMP)—a formulation featuring only a subset of the variables and constraints of LSCT—is solved with the primal simplex algorithm to obtain a primal and a dual solution. Then, a *column generation* step takes place, and the pricing problem is solved in order to find negative reduced cost columns. If any are found, they are added to the RMP and a new column generation iteration is carried out. If not, a *row generation* step takes place, and SR3 constraints are separated in a cutting plane fashion and added to the RMP. The cutting plane algorithm terminates when no violated SR3s are found, and one or more iterations of the column-and-row generation method takes place. The computation stops when both the column and the row generation algorithms terminate without finding new columns or rows. Due to their small cardinality, the SR3 inequalities are separated by complete enumeration. The generation of columns requires, on the contrary, an efficient *ad hoc* pricing algorithm, which we describe in Section 4.1.

Let us consider a generic node of the BPC tree in which Ryan-Foster (RF) branching is applied. Due to RF branching, a node with a fractional solution is split into two new nodes by imposing that a chosen pair of items (the RF pair) either be packed in the same bin (“same branch”) or into different bins (“different branch”). In the same branch case, the pair of items is merged into a *superitem* with a weight equal to the sum of the weights of the individual items, and every already-generated pattern that contains only one of the two items is discarded. In the different branch case, a *conflict* is introduced between the two items to prevent the selection of patterns that contain both, and every already-generated pattern that violates such a constraint is discarded. While in the pricing problem, the same branch case is handled by simply considering the superitem *in lieu* of the two individual items, in the different branch case the new conflict constraint must be enforced explicitly when a new pattern is created (see Section 4.1 for more details).

We denote by MP the master problem associated with the node at hand and by $z(\text{MP})$ its optimal solution value. Let also RMP be the corresponding restricted master problem. Let $\overline{\mathscr{T}} \subseteq \mathscr{T}$ be the set of triplets whose SR3 constraints have been generated so far and let $\overline{N} = \{1, 2, \dots, \overline{n}\}$ be a set of $\overline{n}$ items (possibly involving superitems) with integer weights $\overline{w}_j$ for all $j \in \overline{N}$. Let $G = (\overline{N}, E)$ be a conflict graph whose edges represent conflicts between the items. Let $\overline{\mathscr{P}}$ be the set of patterns generated so far. For each $j \in \overline{N}$, let $\pi_j \geq 0$ be the dual variable corresponding to Constraint (1b) associated with item j and let $\rho_T \leq 0$ be the dual variable corresponding to the SR3 constraint associated with a triplet T belonging to $\overline{\mathscr{T}}$. For each pattern S, let $\overline{\mathscr{T}}(S) = \{T \in \overline{\mathscr{T}} : |T \cap S| \geq 2\}$ be the set of triplets in $\overline{\mathscr{T}}$ with at least two items in S. The LP dual of the RMP reads as follows:

$$\text{(RMPD)} \qquad \max \sum_{j \in \overline{N}} \pi_j + \sum_{T \in \overline{\mathscr{T}}} \rho_T \tag{2a}$$

$$\text{s.t.} \sum_{j \in S} \pi_j + \sum_{T \in \overline{\mathscr{T}}(S)} \rho_T \leq 1 \qquad S \in \overline{\mathscr{P}} \tag{2b}$$

$$\pi_j \geq 0 \qquad j \in \overline{N} \tag{2c}$$

$$\rho_T \leq 0 \qquad T \in \overline{\mathscr{T}}. \tag{2d}$$

The pricing problem coincides with an instance of the *Knapsack Problem with Triplets and Conflicts* (KP-T-C), whose nonempty solutions coincide with the following set of patterns $\widehat{\mathscr{P}}$:

$$\widehat{\mathscr{P}} = \left\{ S \subseteq \overline{N} : S \neq \emptyset, \sum_{j \in S} \overline{w}_j \leq W, \text{ and } \{j_1, j_2\} \notin E \text{ for all } j_1, j_2 \in S \right\}.$$

The reduced cost of a pattern $S \in \widehat{\mathscr{P}}$ with respect to a dual solution $(\boldsymbol{\pi}, \boldsymbol{\rho})$ and a parameter b (equal to 1 here—different values will be used later on in the paper) is

$$\overline{c}_S(\boldsymbol{\pi}, \boldsymbol{\rho}, b) = b - \sum_{j \in S} \pi_j - \sum_{T \in \overline{\mathscr{T}}(S)} \rho_T.$$

The KP-T-C asks for a pattern $S' \in \widehat{\mathscr{P}}$ of minimum reduced cost. If $\overline{c}_{S'}(\boldsymbol{\pi}, \boldsymbol{\rho}, 1) < 0$, the pattern is added to the RMP; otherwise the solution $(\boldsymbol{\pi}, \boldsymbol{\rho})$ is proven to be a feasible dual solution to the MP.

3. Extended-precision numerically safe dual bounds and two-phase column generation method

We aim at computing a numerically safe dual bound with a precision that is higher than the one allowed by floating-point LP solvers (usually not higher than 10^{-9}) by relying on both a floating-point LP solver and an infinite-precision (rational) one.

To achieve this, we introduce a procedure that extends the dual-variable scaling technique of Held et al. (2012) (which applies to an RMP with nonnegative dual variables) to the case where the RMP also features dual variables that are nonpositive. Since floating-point solvers would approximate the value of the dual variables of the RMP by floating-point numbers, such a procedure is *necessary* as, if not handled with care, these floating-point numbers may lead to two issues: (i) incorrect calculations within the pricing algorithm and (ii) the generation of a column which would not enter the basis due to its reduced cost being smaller than the optimality tolerance of the solver. In our algorithm, both are circumvented by relying on the extended scaling procedure (described in this section) and by combining it with an exact integer (fixed-point) pricing algorithm (described in Section 4.1).

Our CRG method works in two phases. In the first phase, the RMP is always reoptimized via the floating-point LP solver and the scaling procedure is combined with the pricing algorithm to generate patterns with a reduced cost that lies within the floating-point solver's optimality tolerance ϵ_0. The second phase starts as soon as the pricing algorithm finds a pattern S' whose reduced cost is strictly smaller (in absolute value) than ϵ_0. Since the floating-point solver would not be able to bring the corresponding $\xi_{S'}$ variable into the basis when reoptimizing the RMP, at the beginning of this phase we switch to the rational (infinite-precision) solver and use it until the CRG procedure halts (halting is guaranteed by Proposition 1 below). This two-phase method allows us to compensate for the weaker dual bound (illustrated in Proposition 2) that the scaling procedure inevitably leads to. With it, we achieve a higher numerical precision than the floating-point solver would allow for while also benefiting for many iterations from the efficiency of this solver. This way, we limit the number of calls to the computationally more cumbersome rational solver and, overall, achieve a good trade-off between numerical precision and computational efficiency.

3.1. Extended dual scaling technique for the computation of numerically safe dual bounds

Let $(\boldsymbol{\pi}^{\text{float}}, \boldsymbol{\rho}^{\text{float}})$ be a floating-point dual solution to the RMP of value z^{float} computed by a floating-point LP solver. Since solvers typically do not provide a bound on the accuracy of their solutions (Held et al. 2012), $(\boldsymbol{\pi}^{\text{float}}, \boldsymbol{\rho}^{\text{float}})$ can be affected by numerical errors (it can be infeasible, suboptimal, or both). Even if this were not the case, truncation errors may still be made in the pricing algorithm due to the floating-point nature of $(\boldsymbol{\pi}^{\text{float}}, \boldsymbol{\rho}^{\text{float}})$ (as the precision available to the pricer may just be insufficient to carry out the necessary arithmetic operations on $(\boldsymbol{\pi}^{\text{float}}, \boldsymbol{\rho}^{\text{float}})$ without errors). As a consequence, the pricing algorithm may attest the dual feasibility of $(\boldsymbol{\pi}^{\text{float}}, \boldsymbol{\rho}^{\text{float}})$ (thus declaring $\overline{c}_S(\boldsymbol{\pi}^{\text{float}}, \boldsymbol{\rho}^{\text{float}}, 1) \geq 0$ for all $S \in \widehat{\mathscr{P}}$) even if the solution is not dual feasible and $z^{\text{float}} > z(\text{MP})$ or deem $(\boldsymbol{\pi}^{\text{float}}, \boldsymbol{\rho}^{\text{float}})$ infeasible even if it is not. In the first case, the risk is of computing the invalid lower bound $\lceil z^{\text{float}} \rceil > z(\text{BPP})$, while, in the second case, the CRG algorithm could enter an infinite loop.

To overcome these issues, we extend the technique proposed in Held et al. (2012) to the case where the RMP features nonnegative as well as nonpositive dual variables. We (safely) approximate the (possibly inaccurate) floating-point dual solution $(\boldsymbol{\pi}^{\text{float}}, \boldsymbol{\rho}^{\text{float}})$ by introducing the following *scaled integer* dual solution $(\boldsymbol{\pi}^{\text{int}}, \boldsymbol{\rho}^{\text{int}})$:

$$\pi_j^{\text{int}} = \lfloor K\pi_j^{\text{float}} \rfloor, j \in \overline{N}, \quad \text{and } \rho_T^{\text{int}} = \lfloor K\rho_j^{\text{float}} \rfloor, T \in \overline{\mathscr{T}},$$

where $K > 0$ is a positive scaling factor and the following *diminished* dual solution $(\boldsymbol{\pi}^{\text{dim}}, \boldsymbol{\rho}^{\text{dim}})$:

$$\pi_j^{\text{dim}} = \frac{1}{K}\pi_j^{\text{int}},\ j \in \overline{N}, \quad \text{and} \quad \rho_T^{\text{dim}} = \frac{1}{K}\rho_T^{\text{int}},\ T \in \overline{\mathscr{T}}.$$

If K is a power of 10, i.e., $K = 10^k$ for some $k \in \mathbb{N}$, then $\boldsymbol{\pi}^{\text{dim}} = \text{trunc}(\boldsymbol{\pi}^{\text{float}}, k)$ due to $\boldsymbol{\pi}^{\text{float}} \geq \mathbf{0}$ where $\text{trunc}(x, k)$ is the truncation of the floating-point number x to the k-th decimal figure. Differently, since $\boldsymbol{\rho}^{\text{float}} \leq \mathbf{0}$, $\boldsymbol{\rho}^{\text{dim}}$ does *not* coincide with the truncation of $\boldsymbol{\rho}^{\text{float}}$ to the k-th decimal figure, as $\text{trunc}(\boldsymbol{\rho}^{\text{float}}, k) = \lceil \frac{1}{K}\boldsymbol{\rho}^{\text{float}} \rceil$. Thus, $\boldsymbol{\pi}^{\text{dim}} = \text{trunc}(\boldsymbol{\pi}^{\text{float}}, k) \leq \boldsymbol{\pi}^{\text{float}}$ whereas $\boldsymbol{\rho}^{\text{dim}} \leq \boldsymbol{\rho}^{\text{float}} \leq \text{trunc}(\boldsymbol{\rho}^{\text{float}}, k)$. By construction, if $(\boldsymbol{\pi}^{\text{int}}, \boldsymbol{\rho}^{\text{int}})$ satisfies

$$\sum_{j \in S} \pi_j^{\text{int}} + \sum_{T \in \overline{\mathscr{T}}(S)} \rho_T^{\text{int}} \leq K, \quad \forall S \in \widehat{\mathscr{P}},$$

then $(\boldsymbol{\pi}^{\text{dim}}, \boldsymbol{\rho}^{\text{dim}})$ satisfies dual Constraints (2b) and, thus, it is dual feasible for the RMP and $z^{\text{dim}} = \sum_{j \in \overline{N}} \pi_j^{\text{dim}} + \sum_{T \in \overline{\mathscr{T}}} \rho_T^{\text{dim}}$ is a valid lower bound on $z(\text{MP})$ ($z^{\text{dim}} \leq z(\text{MP})$).

After computing the scaled integer dual vectors $(\boldsymbol{\pi}^{\text{int}}, \boldsymbol{\rho}^{\text{int}})$, we use them for solving the pricing problem in order to compute one or more patterns by minimizing, rather than the *floating-point reduced cost* $\overline{c}_S(\boldsymbol{\pi}^{\text{float}}, \boldsymbol{\rho}^{\text{float}}, 1)$, the *scaled integer reduced cost* $\overline{c}_S(\boldsymbol{\pi}^{\text{int}}, \boldsymbol{\rho}^{\text{int}}, K)$. Notice that $\overline{c}_S(\boldsymbol{\pi}^{\text{int}}, \boldsymbol{\rho}^{\text{int}}, K)$ is equal to the *diminished reduced cost* $\overline{c}_S(\boldsymbol{\pi}^{\text{dim}}, \boldsymbol{\rho}^{\text{dim}}, 1)$ multiplied by K. This way, the pricing algorithm works solely with fixed-point (integer) numbers. Assuming that $(\boldsymbol{\pi}^{\text{int}}, \boldsymbol{\rho}^{\text{int}})$ and the result of any intermediate calculations that the pricer carries out can be represented by the integer data type that is being used, truncation errors are completely avoided and the pricer always produces an exact solution.

Let $I_{\min} < 0$ and $I_{\max} > 0$ be the minimum and maximum integer values, respectively, that can be represented by the data type used for storing integer values. Standard implementations (including ours), rely on 64-bit signed integers, for which $I_{\min} = -9,223,372,036,854,775,808$ and $I_{\max} = 9,223,372,036,854,775,807$ (inclusive). For an error-free computation of the scaled integer reduced cost $\overline{c}_S(\boldsymbol{\pi}^{\text{int}}, \boldsymbol{\rho}^{\text{int}}, K)$, we need to ensure

$$I_{\min} \leq \sum_{j \in S} \pi_j^{\text{int}} + \sum_{T \in \overline{\mathscr{T}}(S)} \rho_T^{\text{int}} \leq I_{\max}.$$

For this to hold, we must guarantee that the largest scaled integer reduced cost value that can be computed be less than or equal to $I_{\max}$ and that the smallest value be greater than or equal to $I_{\min}$. Since $\boldsymbol{\pi}^{\text{int}} \geq \mathbf{0}$ and $\boldsymbol{\rho}^{\text{int}} \leq \mathbf{0}$, we can restrict ourselves to the following sufficient condition:

$$\sum_{j\in S} \pi_j^{\text{int}} \leq I_{\max} \text{ and } \sum_{T\in\overline{\mathscr{T}}(S)} \rho_T^{\text{int}} \geq I_{\min}.$$

Let $\hat{p}$ and $\hat{q}$ be two upper bounds on, respectively, the number of superitems and the number of SR3 inequalities associated with the RMP and assume $(\boldsymbol{\pi}^{\text{float}}, |\boldsymbol{\rho}^{\text{float}}|) \in [0,1]\times[0,1]$ (we observed that this is always the case in our numerical experiments). The occurrence of integer overflows is explicitly checked in our code; in case of their occurrence, the algorithm is terminated without returning a solution; in our experiments, such a case was never witnessed. By construction, this implies that no entry of $(\boldsymbol{\pi}^{\text{int}}, \boldsymbol{\rho}^{\text{int}})$ is larger than K. To avoid integer overflows during the solution of the pricing problem, K must therefore be chosen such that the following holds:

$$K \leq \min\left\{\frac{I_{\max}}{\hat{p}}, -\frac{I_{\min}}{\hat{q}}\right\}. \tag{3}$$

From a convergence perspective, the definition of $(\boldsymbol{\pi}^{\text{dim}}, \boldsymbol{\rho}^{\text{dim}})$ is motivated by the following proposition:

Proposition 1 *Consider a CRG method with a floating-point LP solver with tolerance ε_0, where the pricing problem is always solved with the integer dual variables $(\boldsymbol{\pi}^{int}, \boldsymbol{\rho}^{int})$ without numerical errors and where the CRG stops whenever $\min_{S\in\widehat{\mathscr{P}}}\{\overline{c}_S(\boldsymbol{\pi}^{dim}, \boldsymbol{\rho}^{dim}, 1)\} \geq -\epsilon_0$. Then, if the reduced costs calculated by the CRG algorithm are in an absolute error no larger than ϵ_0, the CRG algorithm terminates after finitely many iterations.*

Proof. By construction, we have $\pi_j^{\text{float}} \geq \pi_j^{\text{dim}}$ for all $j \in \overline{N}$ and $\rho_T^{\text{float}} \geq \rho_T^{\text{dim}}$ for all $T \in \overline{\mathscr{T}}$. Thus:

$$\begin{aligned}\frac{1}{K}\overline{c}_S(\boldsymbol{\pi}^{\text{int}}, \boldsymbol{\rho}^{\text{int}}, K) = \overline{c}_S(\boldsymbol{\pi}^{\text{dim}}, \boldsymbol{\rho}^{\text{dim}}, 1) = 1 - \sum_{j\in S}\pi_j^{\text{dim}} - \sum_{T\in\overline{\mathscr{T}}(S)}\rho_T^{\text{dim}} \geq \\ 1 - \sum_{j\in S}\pi_j^{\text{float}} - \sum_{T\in\overline{\mathscr{T}}(S)}\rho_T^{\text{float}} = \overline{c}_S(\boldsymbol{\pi}^{\text{float}}, \boldsymbol{\rho}^{\text{float}}, 1)\end{aligned} \tag{4}$$

holds for each $S \in \widehat{\mathscr{P}}$. By assumption, if $(\boldsymbol{\pi}^{\text{dim}}, \boldsymbol{\rho}^{\text{dim}})$ violates the dual constraint associated with a pattern S of diminished reduced cost $\overline{c}_S(\boldsymbol{\pi}^{\text{dim}}, \boldsymbol{\rho}^{\text{dim}}, 1) < 0$, then $\overline{c}_S(\boldsymbol{\pi}^{\text{dim}}, \boldsymbol{\rho}^{\text{dim}}, 1) < -\epsilon_0$ holds. Due to (4), this implies $\overline{c}_S(\boldsymbol{\pi}^{\text{float}}, \boldsymbol{\rho}^{\text{float}}, 1) < -\epsilon_0$. Therefore, such a constraint is violated by the floating-point solution $(\boldsymbol{\pi}^{\text{float}}, \boldsymbol{\rho}^{\text{float}})$ returned by the floating-point LP solver. Since its reduced cost is within the solver's tolerance ϵ_0, when the column corresponding to pattern S is added to the RMP, $(\boldsymbol{\pi}^{\text{float}}, \boldsymbol{\rho}^{\text{float}})$ becomes numerically infeasible for the floating-point solver. This guarantees that, if the reduced costs calculated by the LP solver are in an absolute error no larger than ϵ_0, the CRG method will not enter an infinite loop. Moreover, given a dual solution $(\boldsymbol{\pi}^{\text{float}}, \boldsymbol{\rho}^{\text{float}})$ and the corresponding diminished dual solution $(\boldsymbol{\pi}^{\text{dim}}, \boldsymbol{\rho}^{\text{dim}})$, for each S belonging to the set of patterns $\overline{\mathscr{P}} \subseteq \widehat{\mathscr{P}}$ contained in the RMP we have $\overline{c}_S(\boldsymbol{\pi}^{\text{dim}}, \boldsymbol{\rho}^{\text{dim}}, 1) \geq \overline{c}_S(\boldsymbol{\pi}^{\text{float}}, \boldsymbol{\rho}^{\text{float}}, 1) \geq -\varepsilon_0$. Hence, because $\widehat{\mathscr{P}}$ contains only finitely many patterns, the CRG method terminates after finitely many iterations. Q.E.D.

In order for the CRG algorithm to converge, K must be chosen in accordance with the precision of the pricing algorithm (as indicated in Condition (3)) as well as with the *reduced-cost tolerance for optimality* (e.g., `CPXPARAM_Simplex_Tolerances_Optimality` and `OptimalityTol` parameters for `Cplex` and `Gurobi` LP solvers, respectively) or precision ε_0 of the floating-point LP solver (as indicated in Proposition 1). Somewhat oddly, Held et al. (2012) fail to mention the latter condition even though it is clearly necessary also for the case they consider, where the SC only involves nonnegative variables.

While, as Proposition 1 shows, the CRG algorithm is guaranteed to terminate if the tolerance ϵ_0 of the floating-point LP solver is sufficiently good, working with the diminished dual solution $(\boldsymbol{\pi}^{\text{dim}}, \boldsymbol{\rho}^{\text{dim}})$ rather than with $(\boldsymbol{\pi}^{\text{float}}, \boldsymbol{\rho}^{\text{float}})$ can lead to a bound z^{dim} that is weaker than $z(\text{MP})$. The difference between the two can be bounded according to the following proposition:

Proposition 2 *Let $(\boldsymbol{\pi}^{float}, \boldsymbol{\rho}^{float})$ be an (error-free) optimal dual feasible solution to the MP of value $z(MP)$ and let $(\boldsymbol{\pi}^{dim}, \boldsymbol{\rho}^{dim})$ be the associated diminished solution of value z^{dim}. Then, $z(MP) - \frac{|\overline{N}|+|\overline{\mathscr{T}}|}{K} \leq z^{dim} \leq z(MP)$.*

Proof. For any $\alpha, \beta \in \mathbb{R}$ such that $\beta = \lfloor K\alpha \rfloor$, we have $\alpha - \frac{1}{K}\beta \leq \frac{1}{K}$; hence, we deduce $0 \leq \pi_j^{\text{float}} - \pi_j^{\text{dim}} \leq \frac{1}{K}$ for all $j \in \overline{N}$ and $0 \leq \rho_T^{\text{float}} - \rho_T^{\text{dim}} \leq \frac{1}{K}$ for all $T \in \overline{\mathscr{T}}$. Since

$$z^{\text{float}} - z^{\text{dim}} = \sum_{j \in \overline{N}} (\pi_j^{\text{float}} - \pi_j^{\text{dim}}) + \sum_{T \in \overline{\mathscr{T}}} (\rho_T^{\text{float}} - \rho_T^{\text{dim}}),$$

we derive

$$0 \leq z^{\text{float}} - z^{\text{dim}} \leq \frac{|\overline{N}| + |\overline{\mathscr{T}}|}{K}.$$

The claim follows since $z^{\text{float}} = z(\text{MP})$. Q.E.D.

The proposition shows that $\frac{1}{K}$ should be as small as possible as, if $\frac{1}{K} \geq \varepsilon_0$, the final dual bound obtained when the scaling procedure is adopted would be weaker than it could be.

To achieve a higher numerical precision than ε_0, one could, in principle, adopt a rational (infinite-precision) LP solver instead of a floating-point one at each iteration of the CRG method. In such a case, K would be selected purely based on the fixed-point precision of the pricing algorithm. As known, though, infinite-precision LP solvers suffer from significantly higher running times than their floating-point counterparts, and their adoption at every stage of the CRG method would be impractical (this is indeed confirmed by our preliminary computational experiments). For this reason, we introduce a two-phase CRG algorithm to efficiently handle the dual bound calculations with a value of K with $\frac{1}{K} \ll \varepsilon_0$. In the first phase, we rely on the more efficient but inaccurate floating-point LP solver until the pricing algorithm generates a pattern S with $\overline{c}_S(\boldsymbol{\pi}^{\mathrm{dim}}, \boldsymbol{\rho}^{\mathrm{dim}}, 1) < -\varepsilon_0$. After this, phase two starts and we switch to the rational solver until the MP is solved. This way, we limit the number of calls to the rational solver and achieve a good efficiency as well as a high numerical accuracy.

3.2. Two-phase column generation algorithm

We now describe in more detail the two-phase CRG algorithm we propose to solve the MP at a generic node of the enumeration tree. Before doing so, we introduce dual bounds and fathoming rules that we leverage to improve the overall computational efficiency of the method.

3.2.1. Lower bounds and fathoming rules To achieve, if possible, an early termination of the CRG algorithm at a generic node of the enumeration tree before it naturally converges, we rely on an extension to the case with SR3 constraints of the classical bound that holds for set covering problems that is due to Farley (1990) and also adopt two (classical) fathoming rules (or termination criteria).

The extension of Farley's bound is described by the following proposition:

Proposition 3 *Let $(\boldsymbol{\pi}^{dim}, \boldsymbol{\rho}^{dim})$ be a (not necessarily feasible nor optimal) diminished dual solution of value z^{dim} to the RMP at a generic iteration of the CRG procedure. Let $\overline{c}_{\min} = \min_{S\in\widehat{\mathscr{P}}}\{\overline{c}_S(\boldsymbol{\pi}^{dim}, \boldsymbol{\rho}^{dim}, 1)\}$. If $\overline{c}_{\min} < 0$, the value*

$$LB_F = \frac{z^{dim}}{1 - \overline{c}_{\min}}$$

is a valid lower bound on the optimal solution value $z(MP)$.

Proof. By assumption, we have

$$\overline{c}_S(\boldsymbol{\pi}^{\mathrm{dim}}, \boldsymbol{\rho}^{\mathrm{dim}}, 1) = 1 - \sum_{j\in S} \pi_j^{\mathrm{dim}} - \sum_{T\in\overline{\mathscr{T}}(S)} \rho_T^{\mathrm{dim}} \geq \overline{c}_{\min}, \quad \forall S \in \widehat{\mathscr{P}},$$

or, equivalently,

$$\sum_{j\in S} \pi_j^{\mathrm{dim}} + \sum_{T\in\overline{\mathscr{T}}(S)} \rho_T^{\mathrm{dim}} \leq 1 - \overline{c}_{\min}, \quad \forall S \in \widehat{\mathscr{P}}.$$

Since $1 - \overline{c}_{\min} > 0$, we deduce

$$\sum_{j\in S} \frac{\pi_j^{\mathrm{dim}}}{1 - \overline{c}_{\min}} + \sum_{T\in\overline{\mathscr{T}}(S)} \frac{\rho_T^{\mathrm{dim}}}{1 - \overline{c}_{\min}} \leq 1, \quad \forall S \in \widehat{\mathscr{P}}.$$

This is to say that $\bar{c}_S(\frac{\boldsymbol{\pi}^{\text{dim}}}{1-\bar{c}_{\min}}, \frac{\boldsymbol{\rho}^{\text{dim}}}{1-\bar{c}_{\min}}, 1) \geq 0$ holds for all $S \in \widehat{\mathscr{P}}$. Therefore, the dual solution $(\frac{\boldsymbol{\pi}^{\text{dim}}}{1-\bar{c}_{\min}}, \frac{\boldsymbol{\rho}^{\text{dim}}}{1-\bar{c}_{\min}})$ of value $\frac{z^{\text{dim}}}{1-\bar{c}_{\min}}$ is a feasible solution to the dual of the MP and, thus, z^{dim} is a valid lower bound on $z(MP)$. Q.E.D.

Interestingly, LB_F turns out to be tighter than the classical lower bound on z(MP) known in the literature as the *Lagrangian bound* (see Lübbecke and Desrosiers 2005), which reads:

$$LB_L = z^{\text{dim}} + UB\,\bar{c}_{\min}.$$

The validity of LB_L follows directly from the theory of linear programming. Indeed, since UB is an upper bound on the optimal solution value of the MP, the value of z^{dim} cannot be reduced by more than UB times the smallest reduced cost $\bar{c}_{\min}$. Thus, we have $z^{\text{dim}} + UB\,\bar{c}_{\min} \leq z(\text{MP}) \leq z^{\text{dim}}$. We also provide an alternative derivation of the Lagrangian bound which, to the best of our knowledge, is new. Consider an extension of the SC formulation obtained by adding to it the redundant constraint $\sum_{S \in \overline{\mathscr{P}}} \xi_S \leq UB$, where $UB \leq n$ is an upper bound on the number of bins featured in an optimal solution. The dual of such a problem reads:

$$\max_{\pi \geq 0, \rho \leq 0, \gamma \leq 0} \left\{ \sum_{j \in \overline{N}} \pi_j + \sum_{T \in \overline{\mathscr{T}}} \rho_T + UB\gamma : \quad \sum_{j \in S} \pi_j + \sum_{T \in \overline{\mathscr{T}}(S)} \rho_T + \gamma \leq 1, \;\; S \in \overline{\mathscr{P}} \right\}.$$

Given a floating-point solution $(\boldsymbol{\pi}^{\text{float}}, \boldsymbol{\rho}^{\text{float}})$, let $\gamma^{\text{float}} = \min_{S \in \overline{\mathscr{P}}} \{1 - \sum_{j \in S} \pi_j^{\text{float}} - \sum_{T \in \overline{\mathscr{T}}(S)} \rho_T^{\text{float}}\}$. If $\gamma^{\text{float}} \geq 0$, $(\boldsymbol{\pi}^{\text{float}}, \boldsymbol{\rho}^{\text{float}})$ is dual feasible (with infinite precision). If not, the invalid dual bound $\sum_{j \in \overline{N}} \pi_j^{\text{float}} + \sum_{T \in \overline{\mathscr{T}}} \rho_T^{\text{float}}$ can be corrected by adding to it the (negative) term $UB\gamma^{\text{float}}$. As, by construction, $\gamma^{\text{float}} = \bar{c}_{\min}$, such an updated bound coincides with the Lagrangian bound.

The following proposition holds.

Proposition 4 *LB_F dominates LB_L, i.e., $LB_F \geq LB_L$.*

Proof. We aim to show that

$$LB_L = z^{\text{dim}} + UB\,\bar{c}_{\min} \leq \frac{z^{\text{dim}}}{1-\bar{c}_{\min}} = LB_F.$$

Since $\bar{c}_{\min} < 0$, we divide both sides by $\bar{c}_{\min} < 0$, obtaining

$$\frac{z^{\text{dim}}}{\bar{c}_{\min}} + UB \geq \frac{z^{\text{dim}}}{\bar{c}_{\min}(1-\bar{c}_{\min})}.$$

Grouping by z^{dim}, we have that

$$UB \geq z^{\text{dim}} \left(\frac{1}{\bar{c}_{\min}(1-\bar{c}_{\min})} - \frac{1}{\bar{c}_{\min}} \right) = \frac{z^{\text{dim}}}{1-\bar{c}_{\min}} = LB_F.$$

As the latter inequality is always satisfied thanks to Proposition 3, the claim follows. Q.E.D.

We remark that this result holds even for set covering problems where LB_F is the classical bound due to Farley. To the best of our knowledge, this was not observed before.

In our algorithm, LB_F is combined with the following two fathoming rules, which leverage either the optimality of the incumbent upper bound UB (Fathoming 1) or the optimality of the lower bound $\lceil z^{\text{dim}} \rceil$ (Fathoming 2):

Fathoming Rule 1 *If $\lceil LB_F \rceil = UB$, the subproblem associated with the current node of the enumeration tree is fathomed by the upper bound UB.*

Fathoming Rule 2 *Let z^{dim} be the solution value of the RMP at a generic iteration of the CRG method. If $\lceil LB_F \rceil = \lceil z^{dim} \rceil$, then $\lceil z^{dim} \rceil$ is a valid lower bound on the optimal solution value of the subproblem associated with the current node of the enumeration tree.*

3.3. Outline of the two-phase column generation algorithm

The algorithm we use to solve the MP at a generic node of the enumeration tree is summarized by Algorithm 1. It takes as input the initial numerical precision ε_0 associated with the LP solver, the value of K, the sets $\overline{\mathscr{P}}$ and $\overline{\mathscr{T}}$ of, respectively, patterns and SR3s associated with the current RMP, and the incumbent value UB. It returns the updated sets $\overline{\mathscr{P}}$ and $\overline{\mathscr{T}}$, the value of the lower bound z^{dim} and (these values will be used for the pattern enumeration method described below) the value of z^{int}, the integer scaled dual vector $(\boldsymbol{\pi}^{\text{int}}, \boldsymbol{\rho}^{\text{int}})$, and the diminished dual vector $(\boldsymbol{\pi}^{\text{dim}}, \boldsymbol{\rho}^{\text{dim}})$. If $z^{\text{dim}} = +\infty$, the subproblem associated with the BPC node is fathomed by Fathoming Rule 1.

The algorithm starts by setting ε to the numerical precision ε_0 of the floating-point LP solver (phase I). The variable 'optimality' takes value 'true' if the current lower bound is proven to be optimal and 'false' otherwise, whereas the variable 'fathomed' takes value 'true' if the current subproblem is fathomed and 'false' otherwise. At each iteration of the main loop (line 2), the algorithm solves the RMP using the LP solver with precision ε, and computes the diminished and scaled integer dual vectors $(\boldsymbol{\pi}^{\text{dim}}, \boldsymbol{\rho}^{\text{dim}})$ and $(\boldsymbol{\pi}^{\text{int}}, \boldsymbol{\rho}^{\text{int}})$ (lines 3-4). Then, the pricing problem is solved using $(\boldsymbol{\pi}^{\text{int}}, \boldsymbol{\rho}^{\text{int}})$ followed by the computation of the scaled integer reduced cost $\bar{c}_{\text{min}} = \min_{S \in \overline{\mathscr{P}}} \{\bar{c}_S(\boldsymbol{\pi}^{\text{int}}, \boldsymbol{\rho}^{\text{int}}, K)\}$ associated with the set of generated patterns $\mathscr{P}'$ (lines 5-6). Fathoming Rules 1 and 2 are then tested (lines 7–8); if the lower bound is fathomed by Fathoming Rule 2, the value of the lower bound z^{dim} is set to LB_F. The sign of $\bar{c}_{\text{min}}$ is then checked to verify the convergence of the algorithm (line 9). If $\bar{c}_{\text{min}} \geq 0$, the SR3 inequalities are separated (line 10). If none are found, optimality is proven (line 12); otherwise, the corresponding set of SR3 inequalities $\overline{\mathscr{T}}$ is updated (line 14). If $\bar{c}_{\text{min}} < 0$, the set of patterns $\overline{\mathscr{P}}$ is updated (line 17). The algorithm then checks if the current precision ε of the LP solver is still sufficient to include (one or more) of the newly generated patterns of set $\mathscr{P}'$ in the basis at the next iteration (line 18). If this is not the

Algorithm 1: Two-phase column generation algorithm

```
Input: $\varepsilon_0$, $K$, $\overline{\mathscr{P}}$ and $\overline{\mathscr{T}}$, $UB$
Output: $\overline{\mathscr{P}}$, $\overline{\mathscr{T}}$, $z^{\mathrm{int}}$, $z^{\mathrm{dim}}$, $(\boldsymbol{\pi}^{\mathrm{int}}, \boldsymbol{\rho}^{\mathrm{int}})$, $(\boldsymbol{\pi}^{\mathrm{dim}}, \boldsymbol{\rho}^{\mathrm{dim}})$
begin
       // Initialization (phase I starts)
1      $\varepsilon \leftarrow \varepsilon_0$, optimality $\leftarrow$ false, fathomed $\leftarrow$ false;
       // Main loop
2      while optimality = false and fathomed = false do
           // Solve the RMP
3          Solve the RMP using solver $LP(\varepsilon)$. Let $(\boldsymbol{\pi}^{\mathrm{float}}, \boldsymbol{\rho}^{\mathrm{float}})$ be the RMP dual solution;
           // Compute scaled integer and diminshed dual solutions
4          Compute the scaled integer dual solution $(\boldsymbol{\pi}^{\mathrm{int}}, \boldsymbol{\rho}^{\mathrm{int}})$ and the diminished solution
             $(\boldsymbol{\pi}^{\mathrm{dim}}, \boldsymbol{\rho}^{\mathrm{dim}})$;
           // Solve the pricing problem
5          Compute $S' \in \arg\min_{S\in\widehat{\mathscr{P}}}\{\overline{c}_S(\boldsymbol{\pi}^{\mathrm{int}}, \boldsymbol{\rho}^{\mathrm{int}}, K)\}$ and let $\mathscr{P}' \subset \widehat{\mathscr{P}}$ be the set of generated patterns
             (see §4.1);
           // Computation of $\overline{c}_{min}$
6          $\overline{c}_{min} \leftarrow \overline{c}_{S'}(\boldsymbol{\pi}^{\mathrm{dim}}, \boldsymbol{\rho}^{\mathrm{dim}}, 1)$;
           // Apply Fathoming rule 1
7          if $\lceil LB_F \rceil = UB$ then fathomed $\leftarrow$ true, $z^{\mathrm{dim}} \leftarrow +\infty$; break
           // Apply Fathoming rule 2
8          if $\lceil LB_F \rceil = \lceil z^{dim} \rceil$ then optimality $\leftarrow$ true, $z^{\mathrm{dim}} \leftarrow LB_F$; break
           // Check convergence
9          if $\overline{c}_{min} \geq 0$ then
               // SR3 separation
10             Separate SR3 by complete enumeration. Let $\mathscr{T}' \subseteq \mathscr{T} \setminus \overline{\mathscr{T}}$ be the set of generated triplets;
11             if $|\mathscr{T}'| = 0$ then
12                 optimality $\leftarrow$ true;
13             else
                   // Update the set of SR3
14                 $\overline{\mathscr{T}} \leftarrow \overline{\mathscr{T}} \cup \mathscr{T}'$;
15             end
16         else
               // Update the set of patterns
17             $\overline{\mathscr{P}} \leftarrow \overline{\mathscr{P}} \cup \mathscr{P}'$;
               // Check LP numerical precision
18             if $\overline{c}_{min} \geq -\varepsilon$ then
                   // Set precision to infinite-precision (phase II starts)
19                 $\varepsilon \leftarrow 0$;
20             end
21         end
22     end
23 end
```

case, the precision parameter ε is set to 0, in which case the rational (infinite-precision) solver is adopted from the next main iteration until convergence is established (phase II).

Since the maximum nonpositive scaled integer reduced cost $\overline{c}_S(\boldsymbol{\pi}^{\text{int}}, \boldsymbol{\rho}^{\text{int}}, K)$ that can be computed is equal to -1, the maximum nonpositive diminished reduced cost $\overline{c}_S(\boldsymbol{\pi}^{\text{dim}}, \boldsymbol{\rho}^{\text{dim}}, 1)$ that can be computed is equal to $-\frac{1}{K}$. If K is chosen such as $K = \frac{1}{\varepsilon_0}$ (where $\varepsilon_0 > 0$ is the numerical precision of the LP solver), then the maximum negative diminished reduced cost value computed by the pricing algorithm is no larger than $-\frac{1}{\varepsilon_0}$ and, therefore, Algorithm 1 works within the numerical precision of the LP solver. Dual bounds with an extended numerical precision higher than ε_0 are obtained for any $K > \frac{1}{\varepsilon_0}$.

4. Pricing algorithm, pattern enumeration, node fathoming, and branching scheme

In this section, we provide a description of the exact integer (fixed-point) pricing algorithm we propose and of the pattern enumeration method, together with an illustration of how we solve the reduced SC problem to optmality exactly thanks to a the simplified version of our BPC algorithm. We also introduce a fathoming procedure to be applied at the root node of the enumeration tree, and describe the branching procedures we adopt.

4.1. Pricing algorithm

The pricing algorithm follows the scheme of a classical Dynamic Programming (DP) algorithm for the KP (see Martello and Toth 1990), where $|\overline{N}|$ stages are considered and each stage $j \in \overline{N}$ is associated with a subset of the set of feasible patterns $\widehat{\mathscr{P}}$ involving the first j items. The algorithm takes as input an integer dual solution $(\boldsymbol{\pi}, \boldsymbol{\rho})$ and two user-defined parameters $\hat{z} \in \mathbb{Z}$ and $\Delta \in \mathbb{N}$. If $\hat{z} < 0$, the algorithm produces as output the most negative reduced cost pattern together with at most $\Delta - 1$ additional negative reduced cost patterns (if any). If $\hat{z} \geq 0$, pattern enumeration is performed (see Subsection 4.2 for a detailed description) and the algorithm generates at most Δ patterns with a reduced cost less than or equal to $\hat{z}$.

We call a *label* a tuple $L = (j, X, \omega, \overline{c}, \mathbf{X}, \mathscr{R})$ representing a *partial* pattern, where: j is the last item considered in the label; $X \subseteq \{1, \dots, j\}$ is the set of items contained in the partial pattern; ω is the total weight of X, i.e., $\omega = \sum_{h \in X} \overline{w}_h$; $\overline{c}$ is the reduced cost of the partial pattern; $\mathbf{X}$ is the set of items in $\{j+1, \dots, \overline{n}\}$ that can be individually added to set X resulting in a feasible pattern and $\mathscr{R}$ is the set of $|\overline{\mathscr{T}}|$ binary resources associated with the triplets in the set $\overline{\mathscr{T}}$, namely, $\mathscr{R} = \{\sigma_1, \sigma_2, \dots, \sigma_{|\overline{\mathscr{T}}|}\}$ and $\sigma : \widehat{\mathscr{P}} \to \{0, 1\}$ for all $\sigma \in \mathscr{R}$. With a slight abuse of notation, we refer to the attributes j, X, ω, $\overline{c}$, $\mathbf{X}$, and $\mathscr{R}$ of a label L by $j(L)$, $X(L)$, $\omega(L)$, $\overline{c}(L)$, $\mathbf{X}(L)$, and $\mathscr{R}(L)$.

The algorithm starts from the initial label $L_0 = (0, \emptyset, 0, 1, \overline{N}, \{0\}_{\sigma \in \mathscr{R}})$. A label $L = (j, X, \omega, \overline{c}, \mathbf{X}, \mathscr{R})$ represents a complete pattern consisting of the set of items in X of reduced cost equal to $\overline{c}$ if $\mathbf{X} = \emptyset$. If $\mathbf{X} \neq \emptyset$, let $j^* = \arg\min\{h : h \in \mathbf{X}\}$ be the item of minimum index in $\mathbf{X}$. Given a label L, a *label extension* rule is used to create two new labels L_1 and L_2 by packing

j^* in the former and discarding it in the latter. For a label L, let $\widehat{\mathscr{P}}(L) \subseteq \widehat{\mathscr{P}}$ be the set of all feasible patterns with items in $\mathbf{X}(L)$ that can be used to complete the partial pattern $X(L)$, i.e., $\widehat{\mathscr{P}}(L) = \{S' \in \overline{N} \mid S' = S \setminus X(L) \text{ for some } S \in \widehat{\mathscr{P}}, X(L) \subset S\}$. Given a label $L = (j, X, \omega, \bar{c}, \mathbf{X}, \mathscr{R})$ and a pattern $S \in \widehat{\mathscr{P}}(L)$, we denote by $(X(L), S)$ the pattern $(X(L) \cup S) \in \widehat{\mathscr{P}}$ obtained by joining the partial pattern $X(L)$ with its completion S.

The algorithm relies on fathoming and dominance rules to reduce the number of labels and accelerate its execution. The following property holds:

Property 1 *Given a label $L = (j, X, \omega, \bar{c}, \mathbf{X}, \mathscr{R})$, let $lb(L)$ be a lower bound on the reduced cost of any pattern $(X(L), S) \in \widehat{\mathscr{P}}$ that can be obtained by completing the partial pattern $X(L)$ with some $S \in \widehat{\mathscr{P}}(L)$. If $lb(L) \geq \hat{z}$, then the label L cannot lead to any pattern of reduced cost smaller than $\hat{z}$ and, therefore, it can be discarded.*

Let $KP(B, \alpha, \boldsymbol{\beta}, \boldsymbol{\gamma})$ be an instance of the KP (in minimization version) asking for a minimum-profit subset of items of $B = \{1, 2, \ldots, |B|\}$ fitting into a bin of capacity $\alpha \in \mathbb{Z}_+$ with profits $\boldsymbol{\beta} \in \mathbb{R}_-^{|B|}$ and weights $\boldsymbol{\gamma} \in \mathbb{Z}_+^{|B|}$. Let $z(KP(B, \alpha, \boldsymbol{\beta}, \boldsymbol{\gamma}))$ be its optimal solution value. We define the following fathoming rule (also used by Wei et al. (2020)).

DP Fathoming Rule 1 *Given a label $L = (j, X, \omega, \bar{c}, \mathbf{X}, \mathscr{R})$, the value $lb_1(L) = \bar{c}(L) + z(KP(\{j+1, \ldots, n\}, c - \omega(L), \boldsymbol{\beta}, \boldsymbol{\gamma}))$ with $\boldsymbol{\beta} = (-\pi_{j+1}, \ldots, -\pi_n)$ and $\boldsymbol{\gamma} = (\overline{w}_{j+1}, \ldots, \overline{w}_n)$ provides a valid lower bound on the reduced cost of any pattern $(X(L), S) \in \widehat{\mathscr{P}}$, $S \in \widehat{\mathscr{P}}(L)$.*

Let $z(KP_{LP}(B, \alpha, \boldsymbol{\beta}, \boldsymbol{\gamma}))$ be the optimal solution value of the LP relaxation of the aforementioned KP problem. We introduce the following new fathoming rule:

DP Fathoming Rule 2 *Given a label $L = (j, X, \omega, \bar{c}, \mathbf{X}, \mathscr{R})$, the value $lb_2(L) = \bar{c}(L) + z(KP_{LP}(B, c - \omega(L), \boldsymbol{\beta}, \boldsymbol{\gamma}))$ provides a valid lower bound on the reduced cost of any pattern $(X(L), S) \in \widehat{\mathscr{P}}$, $S \in \widehat{\mathscr{P}}(L)$, where:*

- *B is equal to $\mathbf{X}(L)$ (we assume that the indices in B are defined consecutively).*
- *$\boldsymbol{\beta} = (-\pi_{\nu(1)}, \ldots, -\pi_{\nu(|\mathbf{X}(L)|)})$ and $\boldsymbol{\gamma} = (\overline{w}_{\nu(1)}, \ldots, \overline{w}_{\nu(|\mathbf{X}(L)|)})$, where, for each $h \in B$, $\nu(h)$ is the original index of the item in $\mathbf{X}(L)$.*

It is worth noting that, differently from Fathoming Rule 1, Fathoming Rule 2 explicitly considers the conflict constraints associated with the items in $X(L)$ and the set of remaining items $\mathbf{X}(L)$. Therefore, no dominance relation exists between the two rules. Clearly, at the root node of the BPC tree where no conflict constraints are present, Fathoming Rule 1 dominates Fathoming Rule 2.

To implement the pricing algorithm, we associate with an item $j \in \overline{N}$ a *bucket* $\mathscr{L}_j$ containing all labels $L = (j, X, \omega, \bar{c}, \mathbf{X}, \mathscr{R})$. The set of buckets is implemented by means of a doubly-linked list stored in one contiguous pre-allocated segment of memory, where each node of the list, in addition to the link fields, contains the label data. Whenever the maximum allocation size is reached by the algorithm, the entire algorithm terminates prematurely. In order to reduce the space requirements, the sets X and $\mathbf{X}$ and the resource vector $\mathscr{R}$ of a label are encoded by using a bit-mask representation where every single bit of a computer word is used separately to indicate whether an element is included in the set or not, and classical bit-set operations are used to implement the basic set operations.

The algorithm returns a set of patterns $\mathscr{P}' \subseteq \widehat{\mathscr{P}}$ containing the most negative reduced cost pattern and at most $\Delta - 1$ additional patterns with a negative reduced cost.

To speed up the computations, the lower-bounding functions $lb_1(.)$ and $lb_2(.)$ adopted in Fathoming Rules 1 and 2 are efficiently computed in the following way. For the computation of the bounding function $lb_1(.)$, we precompute the values of the recursive function $g(j,\omega) = z(KP(\{j,\dots,\overline{n}\},\omega,(\overline{w}_j,\dots,\overline{w}_{\overline{n}}),(-\pi_j,\dots,-\pi_{\overline{n}})))$ for all $j \in \overline{N}$, $\omega = 0,\dots,W$ using dynamic programming (Martello and Toth 1990). This allows for computing the corresponding term in the bounding function $lb_1(L)$ as $g(j(L)+1, W-\omega(L))$ with an overall number of operations equal to $O(|I|)$. For computing the bounding function $lb_2(.)$, we precompute the component-wise ratio between the profit vector $-\boldsymbol{\pi}$ and the weight vector $\overline{\boldsymbol{w}}$ and sort the corresponding values in non-increasing order. This allows for computing the bounding function $lb_2(L)$ in $O(|\overline{N}|)$ operations.

To further reduce the number of generated labels, we also rely on the following dominance rule, also used by Wei et al. (2020):

Dominance Rule 1 *Let L_1 and L_2 be two labels with $j(L_1) = j(L_2)$ and $\omega(L_1) \le \omega(L_2)$. If*

$$\overline{c}(L_1) - \sum_{\substack{\sigma \in \mathscr{R}: \\ \sigma(L_1)=1, \sigma(L_2)=0}} \rho(\sigma) \le \overline{c}(L_2) - \sum_{j \in \mathbf{X}(L_2) \setminus \mathbf{X}(L_1)} \pi_j,$$

then label L_1 dominates label L_2 and, thus, label L_2 can be discarded.

The correctness of this rule follows from the observation that the feasible extensions of L_1 are also feasible extensions for L_2 and that the reduced cost of L_1 cannot exceed that of L_2.

4.2. Exact integer pattern enumeration algorithm

A classical result of integer programming (often called *reduced-cost fixing* and dating back to the seminal paper on the Traveling Salesman Problem (TSP) by Dantzig et al. (1954)) is that, given an upper bound UB and the optimal solution value LB of the LP relaxation of a binary integer program, any variable with a reduced cost larger than the difference $UB - LB$ between the UB and the LB cannot take a positive value in an optimal integer solution. Baldacci et al. (2008) were the first to show how this result can be leveraged in the context of a column generation method for the VRP to design a so-called *route enumeration* method (the same method is used by Contardo and Martinelli (2014) and Pessoa et al. (2020)).

The idea to run the CG algorithm until convergence and, then, apply a column enumeration algorithm (a variant of a list-based dynamic programming such as the one we presented previously) tasked with building all the columns with a reduced cost no larger than $UB - LB$. Such columns are used to populate a *Reduced* Master Problem (ReMP) which is then solved as an integer programming problem with a state-of-the-art MILP solver.

Translating this into the context of the BPP, we have a *pattern enumeration* method. Given a feasible (but not necessarily optimal) dual solution $(\boldsymbol{\pi}^{\text{float}}, \boldsymbol{\rho}^{\text{float}})$ of value LB of the dual of LSCT and a valid upper bound UB, any optimal integer solution to SC of cost less than UB cannot contain any pattern $S \in \mathscr{P}$ whose reduced cost computed with respect to the dual solution $(\boldsymbol{\pi}^{\text{float}}, \boldsymbol{\rho}^{\text{float}})$ is greater than or equal to $UB - LB$. Let $\widetilde{\mathscr{P}} \subset \mathscr{P}$ be the set of all the patterns that can be part of a solution that improves on UB (i.e., the set of all patterns of reduced cost strictly less than $UB - LB$). After building $\widetilde{\mathscr{P}}$ by enumeration, the BPP can be solved to optimality by formulating as ReMP a *reduced SC formulation* with explicit integrality constraints that features a variable per pattern in $\widetilde{\mathscr{P}}$ and then solving it with a MILP solver. This way, any optimal solution to the reduced SC formulation is, by construction, an optimal solution to the BPP.

In the context of the BPP, a pattern enumeration method is used implicitly by Pessoa et al. (2021) when casting the BPP as an instance of the VRP and solving it with the `VRPSolver` algorithm that the authors propose. Crucially, though, their method (but also those we mentioned before for the VRP) relies on a floating-point MILP solver for solving the reduced SC formulation and, thus, the validity of the solution it finds *cannot be certified* due to the floating-point errors that can affect it.

To circumvent the numerical issue and guarantee the construction of solutions to the reduced SC formulation that are exact, in our algorithm we rely on the numerically exact BPC method we proposed also for solving the reduced SC formulation. In particular, we apply the pattern enumeration method at the root node of the BPC tree after the computation of the root lower bound based on Algorithm 1. The pricing algorithm is invoked with the final scaled integer solution $(\boldsymbol{\pi}^{\text{int}}, \boldsymbol{\rho}^{\text{int}})$ of value z^{int} to perform pattern enumeration, where Dominance Rule 1 is not used.

Rather than just solving the reduced SC formulation, we solve a *reduced-and-restricted* SC formulation in order to find solutions of value *strictly better* than UB. This is motivated by the fact that, in many instances of the BPP, it is often the case that the UB found at the root node already coincides with the optimal solution value of the problem (this is the case, e.g., of the `ANI` instances). The reduced-and-restricted SC formulation is obtained by imposing an objective cut thanks to which only solutions of value no larger than $UB-1$ are sought. This way, while any optimal solution to such a reduced-and-restricted SC formulation is an optimal solution to the BPP, the infeasibility of the formulation proves that no such solution exists, thus allowing to conclude that UB is the optimal solution value of the problem.

When performing pattern enumeration in our BPC algorithm, we set the parameter Δ of Algorithm 1 to a user-defined parameter Δ_{PE} whose value is dictated by the amount of memory that is available (Δ_{PE} is set equal to 500,000 for the experiments reported in Section 5). This way, we restrict the size of the reduced set of patterns $\widetilde{\mathscr{P}}$ to $|\widetilde{\mathscr{P}}| \leq \Delta_{PE}$. If $|\widetilde{\mathscr{P}}|$ is too large for the amount of memory that is available and $|\widetilde{\mathscr{P}}| > \Delta_{PE}$, the reduced-and-restricted SC formulation is discarded and we resort to BPC. On the contrary, if $|\widetilde{\mathscr{P}}| < \Delta_{PE}$, the reduced-and-restricted SC formulation is solved to optimality using a version of our BPC algorithm in which the pricing problem is disabled, i.e., the BPC algorithm is turned into a branch-and-cut (BC) algorithm. This has several advantages, as it allows for the separation of additional cuts such as the SR3 cuts without impacting the complexity of the pricing algorithm, and also allows for alternative branching rules such as binary branching.

4.3. Node fathoming procedure based on the full exploration of the first level of the enumeration tree and branching scheme

Our experiments revealed that, after solving the root node either in the BPC or in the BC algorithms, the adoption of a RF branching scheme is not always very effective because (as it is known for BPP instances) it may lead to very large enumeration trees. For this reason, we enhance our branching scheme at the root node by adopting a fathoming procedure based on an *full 1-level tree exploration* (to which we refer as F1LTE) of the enumeration tree.

In the BPC case, the F1LTE is utilized to find a RF branching pair leading to a dual bound (the smallest of the bounds calculated with the CRG procedure in each of the two child nodes created after branching) equal to UB. Finding such a pair provides a certificate of optimality of the current solution, allowing the BPC algorithm to terminate at the root node. In the BC case, we use the F1LTE to find a RF pair leading to two subproblems both of which are infeasible. Thanks to the presence of the objective cut in the formulation, finding such a pair of items guarantees that no solution of value strictly better than UB exists, thus allowing the BC algorithm to terminate. If the fathoming procedure is not successful, in both the BPC and the BC algorithms a standard RF branching is carried out by selecting a pair of items fractionally covered with value closer to 0.5.

As shown in more details in Section 5, our experiments revealed that the F1LTE procedure is computationally very effective in the BC algorithm and that, if it fails, the RF branching scheme still allows to solve a good number of instances.

5. Computational experiments

We assess in this section the computational performance of our numerically-exact hybrid BPC and pattern-enumeration algorithm, to which we refer as `BCCF` (named after the authors of this paper). The main goal of this computational study is assessing the role that each of the key components of `BCCF` plays towards solving BPP instances to optimality in a numerically exact way. For comparison purposes, we contrast the computational behavior of `BCCF` with three state-of-art BPP algorithms from the literature—`EXM`, `VRPsolver`, and `NF-F`, even though none of them is numerically exact due to either employing non-safe bounds (`EXM`) or resorting to a MILP solver to carry out certain operations (`VRPsolver` and `NF-F`).

5.1. BPP benchmark sets

Among the many instances proposed for the BPP (an extensive description of which is provided by Delorme et al. (2016)), in this computational study we focus on non-IRUP instances. As mentioned in Section 1, $\lceil z(\text{LSC})\rceil \leq z(\text{BPP}) \leq \lceil z(\text{LSC})\rceil + 1$ holds (stricly in most cases) on every non-IRUP instance. Therefore, computing $z(\text{LSC})$ with a CG method does not suffice to calculate $z(\text{BPP})$ and more sophisticated techniques (the generation of SR3 constraints, branching, or pattern enumeration) are needed. In particular, if $\lceil z(\text{LSC})\rceil = z(\text{LSC})$ holds, then any strictly positive bound improvement over $z(\text{LSC})$ suffices to calculate $z(\text{BPP})$.

In our study, we focus on the instances of the `ANI` class, which comprises 250 instances with a number of items n ranging from 201 to 1,002 and a capacity c ranging from 2,500 to 80,000. These instances enjoy two properties: they satisfy $z(\text{BPP}) = \lceil z(\text{LSC})\rceil + 1$ with $z(\text{LSC}) = \lceil z(\text{LSC})\rceil$ and, for each of them, an optimal solution can be found by running the best-fit algorithm—the challenge for this class is therefore proving the optimality of the greedy solution.

Crucially (as indicated before), these instances are particularly challenging due to suffering from numerical difficulties that have been observed to arise when solving the LSC and LSCT in several papers, see, e.g., (Pessoa et al. 2021). In particular, while $z(\text{LSC}) = \lceil z(\text{LSC})\rceil = z(\text{BPP}) - 1$ holds in infinite-precision arithmetic, it is very often the case that $z(\text{LSC}) < \lceil z(\text{LSC})\rceil$, implying that a strictly positive bound improvement may over $z(\text{LSC})$ not suffice to calculate $z(\text{BPP})$.

5.2. Setup

We rely on two LP solvers in our implementation: `Gurobi` 9.5.1 (Gurobi Optimization, LLC 2022) and `SoPlex` 5.0.1 (Wunderling 1996, Gleixner et al. 2012, 2016). `Gurobi`'s LP solver is used in Phase I of Algorithm 1 and in the enumeration trees associated with the BPC and the BC approaches. `SoPlex` is used in Phase II of Algorithm 1, set to work in infinite-precision. At the root node, we found to be computationally advantageous to set the initial value of ε_0 to 10^{-6}, use `Gurobi` as long as $\varepsilon > 0$, and switch to `SoPlex` when $\varepsilon = 0$. In the enumeration trees associated with the BPC and the BC approaches, Phase II of Algorithm 1 is not used, and we set the value of ε_0 to 10^{-9}, so that `Gurobi` works within its smallest optimality tolerance. At the root note of the BPC tree, we rely on the greedy *best-fit* algorithm to build in a short amount of time an approximate solution (Dósa and Sgall 2014).

By construction, on the `ANI` instances the LP bound of the classical formulation of the BPP featuring $O(n(n+1))$ binary variables (which can be computed in closed form in $O(n)$ as $\frac{1}{c}\sum_{j\in N} w_j$) coincides with z(LSC) (Delorme et al. 2016). Improving on this bound by either branching or by cut generation is therefore key to solving these instances. In order to obtain a bound as strong as possible, in our experiments we separate the SR3 constraints not only at the root note but also at every node of the BPC tree. As the computing time needed to separate SR3 constraints increases w.r.t. the number of items and the number of generated patterns, in our experiments we limit their number to no more than 50. Since, as explained before, the greedy heuristic is always able to find an optimal solution for these instances, we do not rely on any additional heuristic methods for solving them. However, `BCCF` is equipped with different heuristics, including: 1) a *diving heuristic* based on a depth-first heuristic search in the BP tree that relies on a limited backtracking as a diversification and limited discrepancy search method, similar to those of Sadykov and Vanderbeck (2013) and Wei et al. (2020); 2) the greedy *best-fit* algorithm; 3) an LP-based heuristic by which the patterns with ξ_S larger than a given threshold (up to a certain maximum cardinality) are added to the solution and the still uncovered items are packed greedily with the best-fit algorithm.

The experiments with `BCCF` are run on the IRIDIS 5.0 High Performance Computing Facility of the University of Southampton, relying on a cluster of compute nodes equipped with dual 2.0 GHz Intel Skylake processors and 192 GB of DDR2 using a single thread per experiment. Our source code is written in C/C++ and compiled with gcc 11.1.0 with the -o3 optimization flag. In line with the other computational studies in the literature, all the experiments are run within a time limit of 3,600 seconds per instance.

5.3. Comparison with state-of-the-art not numerically-exact algorithms

For comparson purposes, we begin our analysis by contrasting `BCCF` with the following three state-of-the-art BPP algorithms, none of which is numerically-exact:

- `EXM`: BPC algorithm proposed by Wei et al. (2020), run sequentially on an Intel Xeon 3.10 GHz equipped with 8 GB of RAM. The algorithm is not numerically exact due to relying on a non-safe Lagrangian bound and a non-safe floating-point pricing algorithm.
- `VRPsolver`: BPC algorithm proposed by Pessoa et al. (2021), run in parallel with 8 computations at a time on a 2 Deca-core Ivy-Bridge Haswell Intel Xeon e5-2680 v4 server running at 2.50 GHz with 128 GB of RAM. The algorithm computes safe dual bounds during the enumeration tree

Table 1 Number of instances solved exactly to optimality by `BCCF` and average computing time for the `ANI` class, compared with the results obtained with `EXM`, `VRPsolver`, and `NF-F`.

		BCCF		EXM		VRPsolver		NF-F	
Group	inst.	opt	time [s]	solved	time [s]	solved	time [s]	solved	time [s]
`ANI-201`	50	50	13.6	50	13.9	50	16.7	50	3.0
`ANI-402`	50	50	308.2	47	436.2	50	96.0	50	24.9
`ANI-600`	50	25	1,931.5	0	tl	3	3,512.5	50	140.7
`ANI-801`	50	3	3,352.7	0	tl	0	tl	49	393.2
`ANI-1002`	50	0	tl	0	tl	–	–	43	1,302.5
tot	250	128		97		103		242	

based on a safe bound obtained with a procedure similar to the one of Held et al. (2012) (whose details, though, are not given in the paper), but is also uses a non-safe general MILP solver to solve the restricted master problem. As in the paper no detailed results are reported to indicate for which instances the MILP solver was used, it is not possible to verify how many of the solutions found by `VRPsolver` are numerically exact.

• `NF-F`: algorithm based on the *network-flow framework* proposed by de Lima et al. (2021) to solve the integer counterpart to a Dantzig-Wolfe decomposition via a path-flow and arc-flow model, which the authors also use to solve BPP instances; it is run on a computer equipped with an Intel Xeon E3-1245 v5 at 3.50GHz 768 and 32GB RAM, with a single-thread limit. The algorithm computes safe dual bounds based on the scaling technique proposed by Held et al. (2012), but is also uses a non-safe general MILP solver as a key ingredient in the branching scheme to fathom the left child at each level of the enumeration tree, as well as in each leaf node.

The results obtained on the `ANI` instances are reported in Table 1, aggregated by the number of items n into five groups: `ANI-201` (201 items), `ANI-402` (402 items), `ANI-600` (600 items), `ANI-801` (801 items), and `ANI-1002` (1002 items). Each group comprises 50 instances.

The table reports the number of instances solved by `BCCF` (column “opt”) by constructing an exact proof of optimality and the average computing time in seconds (column “time [s]”). For the other algorithms, it reports the number of instances for which each algorithm terminated with a solution that it claimed to be optimal (column “solved”) and the corresponding computing time (column “time[s]”). In line with other works in the BPP literature, the average is computed across all the instances, including those for which the time limit of 3,600 seconds is reached. In the table, an entry is marked with “tl” whenever the corresponding method reached the time limit for all of the instances in the corrsponding group. While the computational environments in which the different algorithms were run is heterogenous and, thus, the solution times of the algorithms cannot be directly compared with absolute precision, in practice and given the type of instances, if an instance cannot be solved within the imposed time limit, there is limited chances to solve the instance within a reasonable (limited) computing time. Therefore, our comparison with respect to the literature will be based on the number of instances solved exactly to optimality by BCCF and on the number of instances for which the other algorithms terminated with a solution they deemed to be optimal.

It is worth noting that, in our preliminary experiments with Algorithm 1 and in line with what is reported by Pessoa et al. (2020), we observed that on the instances with 600 items or more and with Gurobi working within its smallest optimality tolerance (10^{-9}), the pricing problem finds columns with a negative reduced cost that the LP solver does not include in the basis due to the absolute value of the reduced cost being smaller than the optimality tolerance the solver adopts (OptimalityTol). The final solution obtained by the CRG algorithm is therefore not dual feasible. Thus, the numerically safe bound obtained by the scaling procedure can be rather weak. Therefore, the results of those algorithms that employ MILP solvers relying on a finite-precision LP algorithm, such as VRPsolver and NF-F, can be particularly affected by numerical errors when attempting to solve these instances.

As Table 1 shows, BCCF solves 128 instances out of the 250 instances of the ANI class. The algorithm outperforms both EXM and VRPsolver and, on the instances with up to 402 items, it also solves the same instances solved by NF-F. For the larger instances with 600 and 801 items, BCCF largely outperforms EXM and VRPsolver by solving 25 instances with 601 items and 3 with 802 items. No instance with 1002 items is solved by either BCCF, EXM or VRPsolver. NF-F solves (albeit without a guarantee of numerical exactness) almost all the instances with more than 600 items. Indeed, the solutions it produces are not certified to be optimal within an infinite precision, a property that is enjoyed by the solutions produced by BCCF.

5.4. Analysis of the role played by the different components of our algorithm

Tables 2 and 3 illustrate the impact of the different components of BCCF. In both tables, the instances are grouped as in Table 1 and the values reported are average values calculated over the corresponding group of instances.

Under the heading "Phase I", Table 2 reports the number of columns ("col") and the computing time in seconds ("time [s]") of Phase I (Gurobi-based computation) and Phase II (SoPlex-based computation) of Algorithm 1. Under the heading "SR3", the table indicates how many instances are solved to optimality thanks to the generation of SR3 cuts ("opt") and the time spent by separating them ("time [s]"). It is worth mentioning that the generation of SR3 constraints is not carried out in Phase II as preliminary computational results indicated that it was not effective. Under the heading "Phase II", the table reports the number of columns ("col"), the computing time in seconds ("time [s]"), and the number of times ("#") Phase II (SoPlex-based computation)

Table 2 Impact of the different components of BCCF (first part)

		Phase I		SR3		Phase II			Pattern Enumeration		
Group	inst.	col	time [s]	opt	time [s]	#	col	time [s]	#	col	time [s]
ANI-201	50	279.7	1.6	41	1.1	9	121.7	0.8	1	25,235.0	31.5
ANI-402	50	489.5	21.5	18	20.8	32	324.1	4.8	22	2,219.5	0.5
ANI-600	50	660.1	111.6	0	117.0	47	475.6	13.9	25	2,145.0	0.9
ANI-801	50	827.4	360.2	0	340.1	19	537.0	28.0	3	1,809.3	1.3
ANI-1002	50	1,300.5	848.6	0	775.0	18	786.7	80.1	-	-	-
Tot	250			59		125			51		

of Algorithm 1 takes place. The "Pattern Enumeration" section of the table reports the number of instances for which the pattern enumeration was successfully run (i.e., it managed to generate the full set of columns within the reduced gap without exceeding the memory limitations) ("#"), the number of patterns it generates ("col"), and the computing time spent by the label-setting algorithm.

Overall, the table shows that, while relying on a much less efficient solver than Phase I does, Phase II is fairly fast due to the generation of a much smaller (by about an order of magnitude) number of columns. It also shows that the SR3 constraints are useful mostly on the smaller instances, whereas it clearly indicates that the pattern enumeration procedure is responsible for solving a large number of the more challenging instances.

Table 3 provides details about the effectiveness of adopting the F1LTE fathoming procedure within BCCF as well as on the number of instances that are solved by solving the reduced SC problem with regular branching operations and not at the root note thanks to F1LTE. More specifically, under the heading "F1LTE", the table reports the number of instances solved to optimality ("opt") thanks to F1LTE (for both the BPC and BC algorithms), the number of (fractional) pairs of item considered ("pair"), and the number of branches ("branches") on which strong branching was applied. The F1LTE procedure is applied either at the root node of the BC algorithm used for solving the reduced SC problem obtained after pattern enumeration or at the root node of the BPC algorithm which is run whenever the pattern-enumeration procedure fails to build the reduced SC problem due to memory limitations. Under the heading "RF branching", the table reports the number of instances solved to optimality by either the BC or the BPC algorithm with RF branching ("opt") beyond the root node (i.e., when the F1LTE procedure is unsuccessful, the number of nodes ("nodes"), and the computing time ("time [s]").

Our results indicate that the average number of fractional pairs present in the optimal LP solution of the root node, while substantial, decreases w.r.t the number of items and that the actual number of pairs explored by F1LTE when successful is much smaller, ranging from 1/4th to 1/10th

Table 3 Impact of the different components of BCCF (second part)

		F1LTE				RF branching		
Group	inst.	opt	time [s]	pairs	branches	opt	nodes	time [s]
ANI-201	50	8	16.8	405.4	44.3	1	40.0	54.7
ANI-402	50	25	232.2	207.2	53.6	7	5,788.0	1,484.4
ANI-600	50	25	1,386.9	83.8	12.7	0		
ANI-801	50	3	2,852.7	76.3	11.3	0		
ANI-1002	50	-	2,556.1			0		
Tot	250	61				8		

of the total number of pairs. The table clearly shows that the F1LTE fathoming procedure is very effective as it is responsible for solving 61 instances out of the 128 that BCCF solves to optimality. It also shows that the RF branching procedure is still useful, as it is responsible for solving 8 instances.

Based on Tables 2 and 3, we now analyze the results obtained by BCCF on each individual group of instances.

• ANI-201: BCCF solves 41 of the 50 instances with 201 items at the root node thanks to the separation of the SR3 inequalities. For all such instances, $z(\text{LSCT})$ takes values between 65.0000007774641 and 65.0000276484105 (with a precision of $K \simeq 10^{13}$). Since, with infinite precision, we have $z(\text{LSC}) = 65$ and $z(\text{BPP}) = 66$, such an improvement suffices to prove the optimality of the greedy solution on each of these 41 instances. A total of 8 instances are solved at the root node of either the BC or the BPC tree by F1LTE, whereas one instance is solved thanks to RF branching in 54.7 seconds.

• ANI-402: BCCF manages to solve 18 instances with 402 items at the root node thanks to the separation of SR3 inequalities. The pattern enumeration procedure was successfully run on 22 instances. A total of 25 instances are solved by F1LTE, and 7 instances are solved by RF branching outside of the root node.

• ANI-600: BCCF solves 25 of the 50 instances with 600 items, all of which thanks to the pattern enumeration procedure and the F1LTE procedure applied at the root node of the BC algorithm.

• ANI-801: BCCF solves 3 instances of the 50 instances with 801 items, namely, 801_40000_NR_0, 801_40000_NR_11, and 801_40000_NR_27, all of which thanks to the pattern enumeration and the F1LTE procedure. On them, the pattern-enumeration procedure takes less than two seconds, generating approximatively 1800 columns per instance. For all these instances, the F1LTE procedure proves the infeasiblity of the reduced SC problem in 2,852 seconds on average without branching.

5.5. Ablative study of the effectiveness of the main components of our algorithm

The effectiveness of the main components of BCCF are summarized in Table 4. The results are obtained with three variants of BCCF where one of the following components is disabled: separation

Table 4 **Summary of the performance of different versions of BCCF.**

		BCCF		No SR3		No F1LTE		No Patt. Enum.	
Group	inst.	opt	time [s]	opt	time [s]	opt	time [s]	opt	time [s]
ANI-201	50	50	13.6	50	112.3	50	16.5	50	7.7
ANI-402	50	50	308.2	45	494.2	47	427.0	44	695.3
ANI-600	50	25	1931.5	25	1851.5	19	2347.7	0	tl
ANI-801	50	3	3352.7	3	3392.4	3	3358.7	0	tl
ANI-1002	50	0	tl	0	tl	0	tl	0	tl
Tot		128		123		119		94	

Table 5 **Impact of varying the scaling factor K on the ANI-600 and ANI-801 instances**

		Phase II	Pattern Enumeration			
K	opt	time [s]	#	time [s]	cols	labels
$2^{44} \simeq 10^{13}$	128	12.3	51	1.3	2610.1	321,763.8
$2^{37} \simeq 10^{11}$	123	11.8	50	6.0	32,495.8	1,023,669.1
$2^{30} \simeq 10^{09}$	92	5.4	16	12.4	116,741.4	2,656,733.9

of SR3 cuts (“No SR3”), F1LTE fathoming procedure (“No F1LTE”), or pattern enumeration (“No Patt. Enum.”).

The table shows that each component helps to improve the performance of BCCF. In particular, it shows that the pattern enumeration method combined with the computation of extended-precision safe dual bounds plays the most important role as it allows for solving 34 more instances than the version of BCCF in which such a feature is disabled. In particular, this feature is responsible for solving all 25 instances with 600 items that the full version of BCCF manages to solve.

5.6. Analysis of the impact of the scaling factor K

We now analyze the impact of varying the scaling factor K affecting the numerical precision of the dual bounds calculated within BCCF (see Proposition 2) and its impact on the effectiveness of the main components of the algorithm.

A breakdown of the performance of BCCF on the instances of the ANI class with 600 and 801 items is reported in Table 5. The table indicates the number of instances solved to optimality (“opt”) and the average time spent in Phase II (“time [s]”) as a function of the value of the precision parameter K. It also focuses on the pattern enumeration method, indicating the number of instances for which the method was run successfully without exceeding the memory limitations (“#”), the average time spent in the procedure (“time [s]”), and the average number of patterns (“cols”) and labels (“labels”) that it generates for different values of the scaling factor K.

Table 5 clearly shows the positive impact of increasing the value of parameter K affecting the numerical precision. With the larger K, BCCF solved 36 more instances than with the lower K. With a higher numerical precision, improved dual bounds can be computed and the pattern enumeration procedure, whose effectiveness relies on the final restricted gap, becomes more effective. This is testified by the final average number of patterns generated by varying the value of K. It must be noted, though, that the performance in terms of number of labels generated by the DP algorithm used to enumerate the patterns depends not only on the final restricted gap but also on the values of the dual variables associated with the final dual bound. In practice, we observe that the distribution of the dual variables strongly affects the number of labels generated by the DP algorithm, and enumerating patterns based on a lower restricted gap does not necessarily result in a smaller number of DP labels. Moreover, we observed that `SoPlex` may suffer from numerical instabilities for higher values of K, leading to an abnormal termination of the Phase II (this happens when, after building an "optimal" floating-point solution, `SoPlex` tries to build an optimal rational one by a rational implementation of the simple method in which the floating-point solution is used to provide a starting basic solution). For these reasons, in our experiments a few instances solved with a lower value of K cannot be solved with the larger value of K.

Overall, the table illustrates how crucial the calculation of (safe) dual bounds with a precision (10^{-13}) higher than what commercial solvers allow for (10^{-9}) is, validating the relevance of the two-phase column-and-row generation method we proposed.

5.7. Optimal rational solutions to the FBPP

As a byproduct of our work, we exhibit the first (to our knowledge) optimal rational dual solutions to the LP relaxation of the SC formulation of the BPP — the FBPP — for 189 instances of the `ANI` class.

The solutions are certified to be exact by a rational implementation of the pricing algorithm (in its lookup-table version, which suffices for solving the pricing problem when no SR3 constraints have been generated and no branching operations took place). Such a version, while computationally inefficient due to operating in rational arithmetic, allows for manipulating dual variables with no restriction on their precision (i.e., with $K = \infty$). As such, whenever an optimal dual solution is given as input to the algorithm, the latter can certify its optimality with infinite precision.

The solution can be found at the address `https://github.com/stefanoconiglio/FBPP_dual_rational_solutions`. The repository contains solutions to 50/50 instances with 201 items, 50/50 instances with 402 items, 50/50 instances with 600 items, 24/50 instances with 801 items, and 15/50 instances with 1002 items.

6. Conclusions

We proposed the first numerically-exact algorithm for the bin packing problem (BPP). Key to the method is the calculation of extended-precision numerically safe dual bounds with a two-phase column-and-row generation method based on a dual-variable scaling technique that combines a floating-point (finite precision) linear programming solver with a rational (infinite precision) one. To the best of our knowledge, rational solvers have never been successfully used before in the context of a column generation method. The method relies on a suitably-designed exact integer (fixed-point) pricing algorithm which is also designed to perform pattern enumeration, thereby generating a reduced master (set covering) problem containing a superset of the columns (patterns) that are featured in an optimal BPP solution.

Our numerically-exact algorithm was tested on the class of `ANI` instances (which are affected by notorious numerical difficulties) and its performance compared with state-of-the-art not numerically-exact algorithms for the BPP. The results we obtained show that the our exact algorithm outperforms the state-of-the-art not numerically-exact algorithms based on branch-and-cut-and-price that rely on a set-covering formulation of the BPP (`EXM` and `VRPsolver`). Thanks to the extended precision in which our algorithm works, all the solutions it produces on the instances we tested are certified to be optimal within an infinite precision, a property that is not enjoyed by the solutions produced by the other algorithms.

The framework we introduced with our algorithm can be of interest for the design of numerically-exact algorithms of a similar nature to solve a large array of combinatorial optimization problems that are typically tackled via a BPC technique featuring a set covering/partitioning problem as the master problem.

Among many interesting possibilities, future works include the design of a family of modified master problems designed to generate a set of dual variables with a distribution that is conducive to generating a small number of patterns within the pattern-enumeration method, possibly at the cost of worsening the dual bound by a small additive term.

References

Applegate, D. L., Cook, W., Dash, S. and Espinoza, D. G. (2007), Exact solutions to linear programming problems, *Operations Research Letters* **35**(6), 693–699.

Baldacci, R., Christofides, N. and Mingozzi, A. (2008), An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts, *Mathematical Programming* **115**, 351–385.

Belov, G. and Scheithauer, G. (2006), A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting, *European Journal of Operational Research* **171**(1), 85–106.

Caprara, A., Dell'Amico, M., Díaz-Díaz, J. C., Iori, M. and Rizzi, R. (2015), Friendly bin packing instances without integer round-up property, *Mathematical Programming* **150**(1), 5–17.

Coniglio, S., D'Andreagiovanni, F. and Furini, F. (2019), A lexicographic pricer for the fractional bin packing problem, *Operations Research Letters* **47**(6), 622–628.

Contardo, C. and Martinelli, R. (2014), A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints, *Discrete Optimization* **12**, 129–146.

Cook, W., Dash, S., Fukasawa, R. and Goycoolea, M. (2009), Numerically safe Gomory mixed-integer cuts, *INFORMS Journal on Computing* **21**(4), 641–649.

Cook, W., Koch, T., Steffy, D. E. and Wolter, K. (2013), A hybrid branch-and-bound approach for exact rational mixed-integer programming, *Mathematical Programming Computation* **5**(3), 305–344.

Cook, W., Koch, T., Steffy, D. and Wolter, K. (2011), *An Exact Rational Mixed-Integer Programming Solver*, Vol. 6655 of *Lecture Notes in Computer Science*.

Cornuéjols, G., Margot, F. and Nannicini, G. (2013), On the safety of Gomory cut generators, *Mathematical Programming Computation* **5**(4), 345–395.

Dantzig, G., Fulkerson, R. and Johnson, S. (1954), Solution of a large-scale traveling-salesman problem, *Journal of the Operations Research Society of America* **2**(4), 393–410.

de Lima, V. L., Iori, M. and Miyazawa, F. K. (2021), New exact techniques applied to a class of network flow formulations, *in* M. Singh and D. P. Williamson, eds, 'Integer Programming and Combinatorial Optimization', Springer International Publishing, Cham, pp. 178–192.

de Lima, V. L., Iori, M. and Miyazawa, F. K. (2022), Exact solution of network flow models with strong relaxations, *Mathematical Programming* pp. 1–34.

Delorme, M., Iori, M. and Martello, S. (2016), Bin packing and cutting stock problems: Mathematical models and exact algorithms, *European Journal of Operational Research* **255**(1), 1 – 20.

Dósa, G. and Sgall, J. (2014), Optimal analysis of best fit bin packing, *in* 'International Colloquium on Automata, Languages, and Programming', Springer, pp. 429–441.

Eisenbrand, F., Pálvölgyi, D. and Rothvoß, T. (2013), Bin packing via discrepancy of permutations, *ACM Transactions on Algorithms (TALG)* **9**(3), 1–15.

Farley, A. A. (1990), A note on bounding a class of linear programming problems, including cutting stock problems, *Operations Research* **38**(5), 922–923.

Fukasawa, R. and Goycoolea, M. (2011), On the exact separation of mixed integer knapsack cuts, *Mathematical Programming* **128**(1), 19–41.

Fukasawa, R. and Poirrier, L. (2017), Numerically safe lower bounds for the capacitated vehicle routing problem, *INFORMS Journal on Computing* **29**(3), 544–557.

Gilmore, P. C. and Gomory, R. E. (1961), A linear programming approach to the cutting-stock problem, *Operations Research* **9**(6), 849–859.

Gleixner, A. M., Steffy, D. E. and Wolter, K. (2012), Improving the Accuracy of Linear Programming Solvers with Iterative Refinement, ZIB-Report 12-19, Zuse Institute Berlin. Accepted for publication in proceedings of ISSAC 2012: 37th International Symposium on Symbolic and Algebraic Computation.

Gleixner, A. M., Steffy, D. E. and Wolter, K. (2016), Iterative refinement for linear programming, *INFORMS Journal on Computing* **28**(3), 449–464.

Gurobi Optimization, LLC (2022), 'Gurobi Optimizer Reference Manual'.
URL: *https://www.gurobi.com*

Held, S., Cook, W. J. and Sewell, E. C. (2012), Maximum-weight stable sets and safe lower bounds for graph coloring, *Mathematical Programming Computation* **4**(4), 363–381.

Jepsen, M., Petersen, B., Spoorendonk, S. and Pisinger, D. (2008), Subset-row inequalities applied to the vehicle-routing problem with time windows, *Operations Research* **56**(2), 497–511.

Karmarkar, N. and Karp, R. M. (1982), An efficient approximation scheme for the one-dimensional bin-packing problem, *in* '23rd Annual Symposium on Foundations of Computer Science (SFCS 1982)', p. 312–320.

Kartak, V. M., Ripatti, A. V., Scheithauer, G. and Kurz, S. (2015), Minimal proper non-IRUP instances of the one-dimensional cutting stock problem, *Discrete Applied Mathematics* **187**, 120–129.

Lübbecke, M. and Desrosiers, J. (2005), Selected topics in column generation, *Operations Research* **53**(6), 1007–1023.

Martello, S. and Toth, P. (1990), *Knapsack Problems: Algorithms and Computer Implementations*, John Wiley & Sons, Chichester, New York.

Neumaier, A. and Shcherbina, O. (2004), Safe bounds in linear and mixed-integer linear programming, *Mathematical Programming* **99**(2), 283–296.

Pessoa, A. A., Sadykov, R., Uchoa, E. and Vanderbeck, F. (2020), A generic exact solver for vehicle routing and related problems, *Mathematical Programming* **183**(1), 483–523.

Pessoa, A., Sadykov, R., Uchoa, E., Pessoa, A., Sadykov, R. and Uchoa, E. (2021), Solving bin packing problems using VRPSolver models, *SN Operations Research Forum* **2**, 20.

Ryan, D. and Foster, B. (1981), *An integer programming approach to scheduling*, Wren A, ed. Computer Scheduling of Public Transport: Urban Passenger Vehicle and Crew Scheduling (North-Holland, Amsterdam).

Sadykov, R. and Vanderbeck, F. (2013), Bin packing with conflicts: a generic branch-and-price algorithm, *INFORMS Journal on Computing* **25**(2), 244–255.

Scheithauer, G. and Terno, J. (1997), Theoretical investigations on the modified integer round-up property for the one-dimensional cutting stock problem, *Operations Research Letters* **20**(2), 93–100.

Steffy, D. E. and Wolter, K. (2012), Valid linear programming bounds for exact mixed-integer programming, *INFORMS Journal on Computing* **25**(2), 271–284.

Vance, P. H., Barnhart, C., Johnson, E. L. and Nemhauser, G. L. (1994), Solving binary cutting stock problems by column generation and branch-and-bound, *Computational Optimization and Applications* **3**(2), 111–130.

Wei, L., Luo, Z., Baldacci, R. and Lim, A. (2020), A new branch-and-price-and-cut algorithm for one-dimensional bin-packing problems, *INFORMS Journal on Computing* **32**(2), 428–443.

Wunderling, R. (1996), Paralleler und Objektorientierter Simplex-Algorithmus, PhD thesis, Technische Universität Berlin.