Exactly solving packing problems with fragmentation

M. Casazza, A. Ceselli,
Università degli Studi di Milano, Dipartimento di Informatica,
OptLab – Via Bramante 65, 26013 – Crema (CR) – Italy
{marco.casazza, alberto.ceselli}@unimi.it

April 13, 2015

In packing problems with fragmentation a set of items of known weight is given, together with a set of bins of limited capacity; the task is to find an assignment of items to bins such that the sum of items assigned to the same bin does not exceed its capacity. As a distinctive feature, items can be split at a price, and fractionally assigned to different bins. Arising in diverse application fields, packing with fragmentation has been investigated in the literature from both theoretical, modeling, approximation and exact optimization points of view.

We improve the theoretical understanding of the problem, and we introduce new models by exploiting only its combinatorial nature. We design new exact solution algorithms and heuristics based on these models. We consider also variants from the literature arising while including different objective functions and the option of handling weight overhead after splitting. We present experimental results on both datasets from the literature and new, more challenging, ones. These show that our algorithms are both flexible and effective, outperforming by orders of magnitude previous approaches from the literature for all the variants considered. By using our algorithms we could also assess the impact of explicitly handling split overhead, in terms of both solutions quality and computing effort.

Keywords: Bin Packing, Item Fragmentation, Mathematical Programming, Branch&Price

1 Introduction

Logistics has always been a benchmark for combinatorial optimization methodologies, as practitioners are traditionally familiar with the competitive advantage granted by optimized systems. As an example, Vehicle Routing Problems (VRP) [6] are popular since decades for operational planning. Indeed, as more problems are understood from a computational point of view, more details are required to be included in state-of-the art models, driving research for new solution methods in a virtuous cycle.

The Bin Packing Problem with Item Fragmentation (BPPIF) has been introduced to provide such an additional level of detail. As in the traditional Bin Packing Problem (BPP), a set of items of known weight is given, together with a set of bins of limited capacity; the task is to find an assignment of items to bins such that the sum of items assigned to the same bin does not exceed its capacity. However, unlike in BPPs, items can be split at a price, and fractionally assigned to different bins.

The BPPIF has first been introduced to model message transmission in community TV networks, VLSI circuit design [16] and preemptive scheduling on parallel machines with setup times/setup costs. It also proved to suitably model telecommunication optimization, where items correspond to data transfer requests, bins correspond to transmission channels, and a packet switching network dimensioning must be carried out [23]. It has also been employed in fully optical network planning problems [21]: connection requests are given, that can be split among different transmission channels to fully exploit bandwidth, but each split is known to introduce delays and loss of energy in the transmission process.

The BPPIF arises also as tactical counterpart of the Split Delivery Vehicle Routing Problem (SDVRP) [2], in which the VRP models are enriched by allowing multiple visits to each customer, in order to split its transportation request between multiple vehicles and thus better exploit vehicle capacities and decrease the routing costs: BPPIF models allow to find optimal fleet sizes.

We also mention that, from a methodological point of view, there is currently a strong concern in tackling problems in which different kind of decisions need to be taken, some being combinatorial while others continuous in nature. It is the case, for instance, of simultaneous routing and recharge planning of electrical vehicles [20], or rebalancing in bike sharing systems [15]. The BPPIF is one of the most fundamental problems in this family.

From a computational complexity point of view, the BPPIF is NP-Hard, and has been firstly tackled in [19]: the authors studied its computational complexity and discussed the approximation properties of traditional BPP heuristics. In [22] they improved their results, presenting dual asymptotic fully polynomial time approximation schemes that represent the

state-of-the-art in approximately solving BPPIFs. Similar models have been introduced in the context of memory allocation problems by [5] and considered in [12]: BPPs are presented in which items can be split, but each bin can contain at most k item fragments; the theoretical complexity is discussed for different values of k, and simple approximation algorithms are given. Such results have been refined in [11], where the authors provide efficient polynomial-time approximation schemes, and consider also dual approximation schemes.

BPPs are in general appealing benchmarks for decomposition and column generation algorithms [13]: surveys like [8] contributed to make packing models and algorithms popular in the operations research community. State-of-the-art decomposition algorithms can now successfully tackle involved BPPs [9, 18]. Indeed, we previously tackled a particular BPPIF in which a fixed number of bins is given, and a solution needs to be found that minimizes the number of item fragmentations [4]. We proposed a branch-and-price algorithm that allows to solve instances with up to 20 items in one hour of computing time.

However, many other BPPIF variants have been discussed in the literature. In fact, for what concerns capacity consumption, two versions of the BPPIF can be found: the simpler BPPIF with Size Preserving Fragmentation (BPPSPF), that is the variant described above where splitting introduces no overhead, and the BPP with Size Increasing Fragmentation (BPPSIF), in which the weight of each fragment is increased by a certain amount after splitting. Instead, for what concerns the objective of the optimization, the BPPIF arises in the literature in both fragmentations-minimization form as in [21] and [4], or in bin-minimization form as in [22], in which an upper limit on the total number of fragmentations is imposed, and a solution minimizing the number of used bins is required. To the best of our knowledge, no exact approach has been proposed so far for any of these variants.

In this paper we first investigate on further BPPIF properties. These yield compact models that avoid the use of fractional variables. We design both heuristics and new exact algorithms relying on these models that (a) are more flexible, as can be applied to all variants of BPPIF described above, including both bin-minimization and fragmentation-minimization objectives, and both size preserving and size increasing variants and (b) are more effective, as when applied to the variant discussed in [4] are orders of magnitude faster, thus solving in minutes instances one order of magnitude larger than those of [4]. Exploiting our new tools, we also present a computational comparison on BPPIF models, assessing the impact of overhead handling on solution values and computing hardness. Preliminary results were presented in [3].

For the ease of exposition, in Section 2 we consider the bin minimization BPPSPF, its mathematical programming model and a few theoretical properties on the structure of optimal solutions, and in Section 3 we detail our new exact algorithm to solve it. Then, in Section 4 we discuss on how to extend it in order to tackle all the BPPIF variants listed above. In Section

5 we present our experimental analysis. Finally, in Section 6 we summarize our results and collect some brief conclusions.

2 Models

In the following we formalize the *bin minimization BPPSPF (bm-BPPSPF)*. Then, we recall and exploit some properties on the structure of optimal solutions to get an improved formulation that avoids the use of fractional variables, thus yielding better computational behavior. We also discuss dominance between bounds. To clearly distinguish theorems from the literature by new ones, the former are always marked with their reference at the beginning of the claim.

2.1 Mathematical formulation

We are given a set of items I and a set of bins B. Let w_i be the weight of each item $i \in I$ and let C be the capacity of each bin. Each item has to be fully packed, but may be split into fragments and fractionally assigned to different bins. The sum of the weights of the (fragments of) items packed into a single bin must not exceed the capacity C.

The bm-BPPSPF can be stated as the problem of packing all the items into the minimum number of bins by performing at most F fragmentations. It can be formalized as follows:

$$
\begin{aligned}
(BM) \qquad & \min \quad \sum_{j \in B} u_j && && (2.1)\\
& \text{s.t.} \quad \sum_{j \in B} x_{ij} = 1 && \forall i \in I && (2.2)\\
& \sum_{i \in I} w_i \cdot x_{ij} \leq C \cdot u_j && \forall j \in B && (2.3)\\
& \sum_{i \in I, j \in B} z_{ij} - |I| \leq F && && (2.4)\\
& x_{ij} \leq z_{ij} && \forall i \in I, \forall j \in B && (2.5)\\
& 0 \leq x_{ij} \leq 1 && \forall i \in I, \forall j \in B && (2.6)\\
& z_{ij} \in \{0,1\} && \forall i \in I, \forall j \in B && (2.7)\\
& u_j \in \{0,1\} && \forall j \in B && (2.8)
\end{aligned}
$$

where each variable x_{ij} represents the fraction of item i packed into bin j, each binary variable z_{ij} is 1 if any fragment of item i is packed into bin j, and each binary variable u_j is 1 if bin j is open.

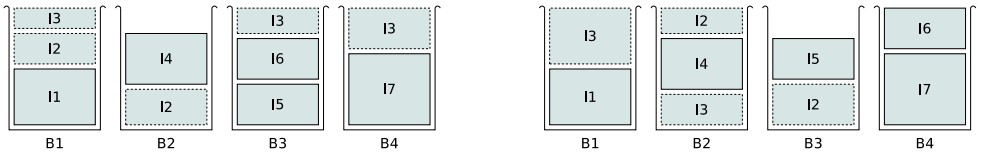

Figure 1: Example of packing with item fragmentation: 7 items are packed into 4 bins. Items 2 and 3 are fragmented in a non-primitive solution (left) and in a primitive one (right).

The objective function (2.1) minimizes the number of open bins. Constraints (2.2) ensure that each item is fully packed. Constraints (2.3) have a double effect: they forbid the assignment of items to bins that are not open, and ensure that the capacity of each open bin is not exceeded. Constraints (2.5) enforce consistency between variables, so that no fragment x_{ij} of each item i is packed into bin j unless z_{ij} is set to 1. Constraint (2.4) ensures that the packing is performed with at most F fragmentations. In fact, as observed in [4],

Observation 2.1 ([4])**:** given any BPPIF solution, the number of fragmentations is equal to the overall number of fragments minus the number of items.

Now, we refer to the definition of *primitive solution* given in [22] as a feasible packing such that (a) each bin contains at most two fragmented items and (b) each item is fragmented at most once. According to the literature

Theorem 2.1 ([22])**:** any instance of bm-BPPSPF has an optimal solution which is primitive.

In Figure 1 an example of both primitive and non-primitive solutions are reported.

It is easy to prove that such a structure directly influences the cost of a packing by representing a primitive solution through the construction of a *Bin Packing Graph (BPG)*, which is a graph having one vertex for each bin and one edge between each pair of bins sharing a fragmented item. In Figure 2, the BPGs corresponding to the solutions of Figure 1 are depicted.

Following the notation of [4] we define as *chain* each maximal path in the BPG of a primitive solution. For instance, the primitive solution reported in Figure 1 yields a BPG composed by two chains: bins B1-B2-B3, and bin B4. It is easy to observe that

Observation 2.2: the cost of a primitive solution is the sum of the length of all the chains in its BPG.

In fact, the length of a chain is the number of bins in the corresponding path of the BPG. Hence, the sum of all chain lengths is the overall number of bins used in a certain solution, and thus the cost of the packing. Furthermore, according to [4],

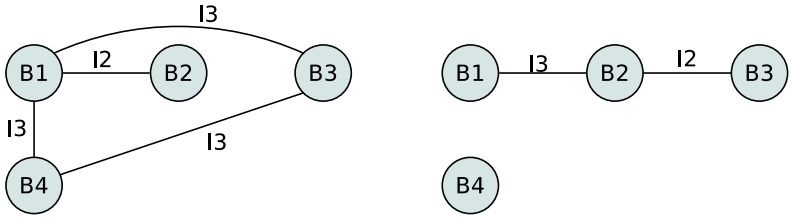

Figure 2: Example of Bin Packing Graphs of non primitive (left) and primitive (right) solutions in Figure 1.

Proposition 2.1 ([4])**:** let a partitioning of the set of items be given, in which each class corresponds to the subset of items packed into bins of the same chain. Then, a feasible primitive solution can be obtained using the *Next-Fit with Item Fragmentation (NF$_f$)* algorithm of [19] on each class independently.

This method does not affect the cost of the solution, since the length of each chain is fixed and thus the overall number of used bins does not change.

By exploiting these properties, we can model the bm-BPPSPF as the problem of optimally packing items into chains instead of bins (Figure 3).

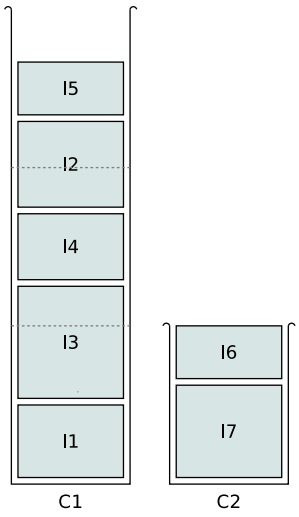

Figure 3: Chain representation of the primitive solution in Figure 1.

Let K be a set of chains. Let l_k be a variable representing the length of each $k \in K$; that is, each $k \in K$ includes a set of l_k bins, involves $l_k - 1$ item splits and provides an overall

capacity of $l_k \cdot C$. Model (2.1) – (2.8) can be reformulated as follows:

$$
\begin{aligned}
(BMC) \qquad & \min && \sum_{k \in K} l_k && && (2.9)\\
& \text{s.t.} && \sum_{k \in K} z_{ik} = 1 && \forall i \in I && (2.10)\\
& && \sum_{k \in K} l_k - v_k \leq F && && (2.11)\\
& && \sum_{i \in I} w_i \cdot z_{ik} \leq C \cdot l_k && \forall k \in K && (2.12)\\
& && v_k \leq l_k && \forall k \in K && (2.13)\\
& && z_{ik} \in \mathbb{B} && \forall i \in I, \forall k \in K && (2.14)\\
& && l_k \in \mathbb{N} && \forall k \in K && (2.15)\\
& && v_k \in \mathbb{B} && \forall k \in K && (2.16)
\end{aligned}
$$

where each binary variable z_{ik} is set to 1 if a fragment of item i is packed into chain k, and each binary variable v_k is set to 1 if chain k contains at least one item.

The objective function (2.9) minimizes the number of used bins. Each item can be split among bins in the same chain, but no fractional assignment of items to different chains is allowed: constraints (2.10) impose that each item is fully packed in a single chain. Constraints (2.12) ensure that the capacity of each chain is not exceeded, and constraint (2.11) guarantees that at most F fragmentations are performed. Constraints (2.13) enforce consistency between variables, so that a chain is used only if its length is at least one.

We remark that a solution of BMC encodes no information on which items are split, nor on which items are packed into the same bin of each chain. In fact, due to Proposition 2.1, it is possible to obtain a feasible solution of bm-BPPSPF starting from any feasible solution of BMC with post processing, by applying the NF_f algorithm on each chain independently.

2.2 Extended formulation

From a continuous relaxation point of view, neither formulation (2.1) – (2.8) nor formulation (2.9) – (2.16) offers a significant lower bound: in the first model, it is possible to fix $|B| = |I|$, and pack each item i into bin $j = i$ fixing each $u_i = w_i/C$. The corresponding objective function value would be $\sum_j u_j = \sum_i w_i/C$, yielding a trivial lower bound. Likewise in the chain model, by fixing $|K| = |I|$ and $l_i = w_i/C$.

Therefore, we propose a reformulation of the problem obtained through Dantzig-Wolfe

decomposition [7]. Let $z_k = (z_{1k}, z_{2k}, \ldots, z_{|I|k})$ and $w = (w_1, w_2, \ldots, w_{|I|})$. Let, for each k in K,

$$\Omega_k = \left\{ (z_k, v_k, l_k) \in \mathbb{B}^{|I|} \times \mathbb{B} \times \mathbb{N} \mid w^T \cdot z_k \leq C \cdot l_k \wedge v_k \leq l_k \right\}$$

be the set of feasible integer points with respect to constraints (2.12) – (2.16). We relax integrality conditions, but replace each Ω_k with the convex hull of its P_k extreme integer points

$$\Gamma_k = \left\{ (\bar{z}_k^1, \bar{v}_k^1, \bar{l}_k^1), \ldots, (\bar{z}_k^{P_k}, \bar{v}_k^{P_k}, \bar{l}_k^{P_k}) \right\}$$

and then we impose

$$(z_k, u_k, l_k) = \sum_{p \in P_k} y_k^p \cdot (\bar{z}_k^p, \bar{v}_k^p, \bar{l}_k^p) \tag{2.17}$$

with $y_k^p \geq 0$ for each $k \in K$, $p \in \Gamma_k$, and $\sum_{p \in \Gamma_k} y_k^p = 1$ for each $k \in K$. That is, each point is represented as a linear convex combination of points in Γ_k, and variables y represent coefficients in such a combination.

The model obtained by replacing in the continuous relaxations of formulation BMC the vectors (z_k, u_k, l_k) as indicated in (2.17), and by making explicit the vector indices is

$$\min \quad \sum_{k \in K} \sum_{p \in \Gamma_k} y_k^p \cdot \bar{l}_k^p \tag{2.18}$$

$$\text{s.t.} \quad \sum_{k \in K} \sum_{p \in \Gamma_k} y_k^p \cdot \bar{z}_{ik}^p = 1 \qquad \forall i \in I \tag{2.19}$$

$$\sum_{k \in K} \sum_{p \in \Gamma_k} y_k^p \cdot (\bar{l}_k^p - \bar{v}_k^p) \leq F \tag{2.20}$$

$$\sum_{p \in \Gamma_k} y_k^p = 1 \qquad \forall k \in K \tag{2.21}$$

$$y_k^p \geq 0 \qquad \forall k \in K, \forall p \in \Gamma_k \tag{2.22}$$

Constraints (2.19) can be relaxed in $\geq$ form, as an optimal solution always exists in which no item is assigned to bins more than once. Constraints (2.21) can be relaxed in $\leq$ form by observing that an empty pattern with $\bar{l}_k^p = 0$, $\bar{v}_k^p = 0$ and $\bar{z}_{ik}^p = 0$ always exists for each $k \in K$; in fact selecting such a pattern is equivalent to setting all the corresponding y_k^p variables to 0. From this relaxation we also observe that constraint (2.20) can be rewritten as

$$\sum_{k \in K} \sum_{p \in \Gamma_k} y_k^p \cdot (\bar{l}_k^p - 1) \leq F.$$

We also observe that since bins are identical, so are the sets Γ_k. Therefore, we consider a

single representative $\Gamma = \bigcup_{k \in K} \Gamma_k$, and aggregate constraints (2.21) as

$$\sum_{p \in \Gamma} y^p \leq |K|. \tag{2.23}$$

Furthermore, constraints (2.23) can be removed from the model since it always exists an optimal solution in which at most $|K|$ patterns are selected. After a rewriting in canonical form, we obtain the following Master Problem (MP):

$$\min \quad \sum_{p \in \Gamma} y^p \cdot \bar{l}^p \tag{2.24}$$

$$\text{s. t.} \quad \sum_{p \in \Gamma} y^p \cdot \bar{z}_i^p \geq 1 \qquad \forall i \in I \tag{2.25}$$

$$-\sum_{p \in \Gamma} y^p \cdot (\bar{l}^p - 1) \geq -F \tag{2.26}$$

$$y^p \geq 0 \qquad \forall p \in \Gamma \tag{2.27}$$

Observation 2.3: The lower bound provided by the MP dominates that given by the continuous relaxation of model BMC.

The observation directly follows from the Dantzig-Wolfe decomposition principle. We further observe that, although the continuous relaxations of models BM and BMC are equivalent, their Dantzig-Wolfe decompositions are not. In particular,

Proposition 2.2: the lower bound provided by the MP dominates that obtained through Dantzig-Wolfe decomposition of model BM;

a proof is sketched in subsection 4.1. As discussed in Section 5, we found the lower bound provided by MP to outperform the other ones also from an experimental point of view.

3 Algorithms

Straightly solving the MP would be impractical, as it would require to consider a tableau with $|\Gamma|$ columns; therefore, we recur to column generation techniques: we start with a Restricted Master Problem (RMP) involving a small subset of columns (see Subsection 3.1), we solve it to optimality, and we use dual information to search for variables having negative reduced cost; to this aim we propose an exact combinatorial algorithm for a particular variant of the 0-1 Knapsack Problem (KP) (see Subsection 3.2). If any negative reduced cost variable is found, it is added to the RMP and the column generation process is repeated, otherwise the optimal

RMP solution is optimal for the MP as well, and therefore the corresponding value is retained as a valid lower bound for the bm-BPPSPF. If the final RMP solution is integer, then it is also optimal for the bm-BPPSPF; otherwise, in order to find a proven global optimum, we enter a search tree by performing branching operations (see Subsection 3.4) .

3.1 Initialization

In order to reduce heading-in effects, we populate the RMP with two sets of columns. The first one consists of chains of different length that pack all the items; as a side effect, this ensures the starting RMP to be feasible. The second one is composed by polynomial families of dual cuts; as a side effect, they help to prevent stability issues and to speedup the overall column generation process.

Subset-Sum columns Our heuristic initialization approach is based on the iterative generation of columns obtained by solving Subset-Sum problems: the algorithm (see Pseudocode 3.1) generates at each iteration a set of chains having the same length, and packs the items by minimizing the residual capacity of each chain. The starting length of the chains is set to one, while the maximum allowed length is set to F.

The packing relies on a *Subset-Sum (SS)* Procedure that takes in input a set of items J, their weights w and a capacity Q, and returns the set of items $\bar{J} \subseteq J$ of largest overall weight not exceeding Q.

function INITRMP(I, w, K, C)
  for $k = 1 \dots F$ **do**
    $J \leftarrow I$
    do
      $\bar{J} \leftarrow$SS$(J, w, k \cdot C)$
      add $\bar{J}$ to RMP as a new column
      $J \leftarrow J \setminus \bar{J}$
    while $J \neq \emptyset$
  end for
end function

Pseudocode 3.1: RMP initialization alorithm

In our algorithm, the SS Procedure exploits a simple dynamic programming recursion, as described in [17]. It is easy to observe that this approach always produces a set of columns forming a feasible RMP solution. In particular, during iteration $k = 1$, the initialization algorithm packs all the items in chains of single bins, thereby creating a BPPSPF solution with no fragmentations.

Dual cuts for BPPSPF Then we restrict the dual space of the MP, in order to obtain optimal MP solutions faster. We exploit a variant of the family of dual cuts proposed in [24] for the Cutting-Stock Problem. Let λ and μ be the vector of non negative dual variables corresponding to constraints (2.19) and (2.20), respectively. We formulate the following

Proposition 3.1: for each pair of subsets S and T of I such that $\sum_{i \in S} w_i \leq \sum_{i \in T} w_i$, no optimal dual solution violates the following inequalities

$$\sum_{i \in S} \lambda_i \leq \sum_{i \in T} \lambda_i \tag{3.1}$$

In fact, if any such inequality were violated, all basic columns of the MP representing chains including T can be replaced by columns where items in T are removed and replaced by items in S, as these still encode feasible chains. The reduced cost of these new columns would be negative, thus contradicting the optimality of the solution.

Intuitively, these inequalities encode the following condition: an optimal solution always exists, in which subsets of small size yield less dual contribution to the reduced costs. Moreover, the linear combination of dual cuts (3.1) and columns in the RMP, gives origin to new patterns that may avoid the generation of further columns.

From an implementation point of view, each dual cut is represented by a column p in which $z_i^p = 1$ for each $i \in S$ and $z_i^p = -1$ for each $i \in T$. These valid dual cuts are in exponential number. Very recent contributions showed that in particular cases their dynamic generation is appealing [14]. Instead, we found it useful to focus on sets S and T of small cardinality, considering two cases in which $|S| = |T| = 1$, and $|S| = 2$ and $|T| = 1$, and we add the corresponding columns to the RMP before the execution of the column generation process.

3.2 Pricing problem

For each $p \in \Gamma$, the reduced cost of variable y^p is computed as

$$\pi^p = \bar{l}^p - \sum_{i \in I} \lambda_i \cdot \bar{z}_i^p + \mu \cdot (\bar{l}^p - 1).$$

The pricing problem, that is the problem of finding the most negative reduced cost column,

can be stated as follows:

$$\begin{aligned} \pi^* = \min_{p\in\Gamma} \quad & \bar{l}^p - \sum_{i\in I} \lambda_i \cdot \bar{z}_i^p + \mu \cdot (\bar{l}^p - 1) && (3.2)\\ \text{s.t.} \quad & \sum_{i\in I} w_i \cdot \bar{z}_i^p \leq C \cdot \bar{l}^p \\ & 0 \leq \bar{l}^p - 1 \leq F \\ & \bar{z}_i^p \in \mathbb{B} && \forall i \in I \\ & \bar{l}^p \in \mathbb{N} \end{aligned}$$

Let us state the objective function of the pricing problem (3.2) in maximization form, and collect the coefficients of terms z_i^p and l^p

$$\pi^* = -\max_{p\in\Gamma} \sum_{i\in I} \lambda_i \cdot \bar{z}_i^p - (\mu+1) \cdot \bar{l}^p - \mu.$$

That is, λ_i and $(\mu+1)$ represent the prize for packing item i and the cost for using each bin in the current chain, respectively. Therefore, the pricing problem is a variant of the 0-1 Knapsack Problem (KP) [17]. It aims to find an optimal tradeoff between the cost of using bins and the profit of including items, respecting a single aggregated capacity constraint and an upper bound on the available capacity. We indicate such a variant as the *Variable Size KP* (VSKP).

To solve the VSKP we develop the following ad-hoc pseudo-polynomial time algorithm based on the well-known dynamic programming approach for the 0-1 KP [17]: let $M(\bar{I}, c)$ be the cost of an optimal VSKP solution in which only items $\bar{I} \subseteq I$ are allowed to be selected, and exactly c units of capacity are consumed. The values $M(\bar{I}, c)$ can be recursively computed as follows:

$$M(\bar{I} \cup \{i\}, c) = -(\mu+1) \cdot \lceil c/C \rceil + \begin{cases} M(\bar{I}, c), \text{ if } w_i > c \\ \max\left\{M(\bar{I}, c); M(\bar{I}, c - w_i) + \lambda_i\right\}, \text{ otherwise} \end{cases}$$

where $M(\emptyset, c) = -\mu$ for each $0 \leq c \leq F \cdot C$. Thus, the final cost of an optimal VSKP is

$$\pi^* = \max_{0\leq c\leq F\cdot C} \{M(I, c)\}$$

It is easy to keep track of the subset of items forming an optimal solution by storing which arguments yield maxima in the above expression. The complexity of the overall procedure is $O(F \cdot C \cdot |I|)$.

As multiple pricing strategy, after preliminary experiments we found out to be beneficial

adding $|K|$ columns at each column generation iteration, corresponding to the values $M(I, k{\cdot}C)$ for $k = 1 \dots |K|$.

3.3 Primal Heuristics

When the column generation process is over, the optimal MP solution can be fractional. In that case we run primal (upper bounding) heuristics: if the upper and lower bounds match, global optimality for the bm-BPPSPF is proved. In order to find good integer solutions quickly, we developed the following Iterative Subset-Sum Heuristic (ISSH), that is built on the SSH heuristic for fragmentations-minimization BPPSPF (fm-BPPSPF) proposed in [4]. The idea is to iteratively fix the number of open bins and pack all the items by solving a fm-BPPSPF, until a feasible solution is reached, that packs all items with a number of fragmentations not exceeding F. The algorithm is detailed in Pseudocode 3.2.

function ISSH(I, w, F, C)
  $B \leftarrow \lceil \sum_{i \in I} w_i / C \rceil$
  loop
    $P, \tilde{F} \leftarrow$ SSH(I, B, w, C)
    if $\tilde{F} \leq F$ **then**
      return P, B
    else
      $B \leftarrow B + 1$
    end if
  end loop
end function

Pseudocode 3.2: ISSH heuristic

The ISSH makes use of the SSH procedure [4], that takes as arguments the set of items I, the number of bins B, the vector of weights w and the capacity C, and returns a set of chains P and the number of fragmentations $\tilde{F}$. The algorithm always ends with a feasible solution for the bm-BPPSPF, since it is always possible to find a feasible packing that uses $|I|$ bins.

Additionally, we considered several general purpose MILP heuristics to be run on MP fractional solutions, that are included in the framework we used in our implementation; more details are reported in Section 5.

3.4 Branch and bound

When the optimal MP solution is fractional, and upper and lower bounds do not match, we check which integrality constraints are not satisfied, and we explore a search tree by means of

branching.

In our case, branching is particularly involved, as the MP is prone to symmetries. We devised the following binary branching rule, in which chains are progressively defined and an integer solution is enforced through the assignment of items to the chains.

Let us suppose that a particular *head item* is defined for each chain, and that at a certain node of the branching tree, items in a set $H \subseteq I$ are selected to be *head items* of $|H|$ chains. At the root node, $H = \emptyset$; then, recursively, a branching item is either selected to be assigned to one of the chains identified by an item in H, or becomes a new head item, thereby identifying an additional chain.

Phase 1 - Item assignment If $H = \emptyset$, we directly skip to Phase 2. Otherwise, let $\tilde{y}^p$ be the values of each variable y^p in a fractional MP solution and, for each $i \in I$ and $h \in H$

$$t_{ih} = \sum_{p \in \Gamma} \bar{z}_i^p \cdot \bar{z}_h^p \cdot \tilde{y}^p$$

be a coefficient that indicates how much item i is packed with head item h in the fractional solution. We search for an item $\hat{i}$ and a head item $\hat{h}$ corresponding to the most fractional assignment in the current fractional solution, that is

$$(\hat{h}, \hat{i}) \in \underset{h \in H, i \in I}{\operatorname{argmin}} \left\{ \left| t_{ih} - \frac{1}{2} \right| \right\}.$$

If $t_{\hat{i}\hat{h}}$ is fractional, then we perform binary branching: we enforce $\hat{i}$ to be always packed with $\hat{h}$ in one branch, while we forbid $\hat{i}$ to be packed with $\hat{h}$ in the other. If instead $t_{\hat{i}\hat{h}}$ is integer, we proceed to Phase 2.

Phase 2 - Chain selection If no fractional assignment of items to chains defined by H can be found, then it also holds that no column in the MP having a fixed head item is fractionally selected. In fact, if it were, such fractionally selected columns should be identical, but in our MP it is never profitable to generate the same column twice. However, it is still possible that fractional solutions arise due to splitting in additional chains for which no head item is fixed.

We check this condition as follows. We search for the most fractionally selected MP variable, that is we identify

$$\hat{p} \in \underset{p \in \Gamma}{\operatorname{argmin}} \left\{ \left| \tilde{y}^p - \frac{1}{2} \right| \right\}.$$

If $\tilde{y}^{\hat{p}}$ is integer, then a full integer solution is found: the incumbent is possibly updated and

the branching node is fathomed.

Otherwise, an arbitrary item $\hat{i}$ is selected, such that $\tilde{z}_{\hat{i}}^{\hat{p}} = 1$ and $\hat{i} \notin H$. Then we add $\hat{i}$ to the set H, we initialize $I_{\hat{i}} = \{\hat{i}\}$, and we restart branching from Phase 1.

Pricing implementation Our branching strategy changes the nature of the pricing problem. Let I_h be the set of items forced to be packed with head item h, let $W_h = \sum_{i \in I_h} w_i$ be their sum of weights, and let $\bar{I}_h$ be the set of items whose packing with h is forbidden. Let I_0 be the set of items which are not forced to be packed to any head item, such that

$$I_0 = I \setminus \bigcup_{h \in H} I_h.$$

We deal with additional constraints introduced in both branching phases by solving $|H| + 1$ VSKPs. The first VSKP aims at finding the chain without head item yielding the most negative reduced cost:

$$\pi_0 = - \max_{0 \leq c \leq F \cdot C} \{M(I_0, c)\}.$$

Then, we solve a VSKP for each $h \in H$, each searching for the chain with head item h yielding the most negative reduced cost:

$$\pi_h = - \sum_{i \in I_h} \lambda_i - \max_{0 \leq c \leq F \cdot C - W_h} \{M(I_0 \setminus \bar{I}_h, c)\}.$$

that is, we solve a VSKP where the available capacity is decreased by the weights of the items fixed in I_h, forbidding the selection of items in $\bar{I}_h$ and decreasing the final reduced cost by the sum of the prizes of the items in I_h.

We experimentally observed that, although the number of VSKP subproblems increases as the depth of the branching tree increases, the overall number of chains remains limited. Additionally, the solutions of the $|H| + 1$ VSKPs yield well diversified columns, that are in turn useful to perform more effective multiple pricing, thus speeding up the column generation process.

In particular, after preliminary experiments, we set a different multiple pricing strategy in the inner nodes of the branching tree. That is, at each iteration of column generation we add to the RMP (a) one column, corresponding to the packing defining the value of π_0 and (b) $|H|$ columns, each corresponding to the packing defining the value of $i\pi_h$ for each $h \in H$, provided they have negative reduced cost.

4 Tackling BPPIF variants

Several BPPIF variants arise in the literature and in practical applications. The main features changing among them are the possibility of handling overhead in item weights after each split and the objective to be optimized.

In the following we first describe how to handle these features in chain based models. Then we discuss on how to adapt our algorithms to obtain a unified framework.

4.1 Extending Models

First, still owing to practical applications, packing an item may introduce an overhead in its weight. This is the typical case of transmissions over packed-switching networks, in which data are split into packets that need additional headers (and trailers) to be delivered, as stated in [22].

Let ϵ be a constant representing an amount of overhead, that we suppose to satisfy $\epsilon \leq \min_{i \in I} w_i$. The BPPIF with size-increasing fragmentation (BPPSIF) is the variant of BPPIF in which a weight ϵ is attached to all fragments packed into bins, including those fragments corresponding to unfragmented items. We first observe that according to the literature,

Proposition 4.1 ([22])**:** each solution of bin-minimization BPPSIF has a primitive solution.

Thus we adapt the chain-based BPPIF model BMC (2.9)–(2.16), by changing constraint (2.12) into

$$\sum_{i \in I} (w_i + \epsilon) \cdot z_{ik} + \epsilon \cdot (l_k - 1) \leq C \cdot l_k \qquad \forall k \in K, \tag{4.1}$$

where the term $\epsilon \cdot (l_k - 1)$ is the overhead given by the chain fragmentations.

A corresponding extended formulation can also be obtained, by performing the reformulation steps described on (2.18)–(2.22), and changing only the definition of sets Γ^k. Indeed, our BPPSPF models can be obtained as a special case of BPPSIF ones by setting $\epsilon = 0$.

We remark that other overhead policies may be pertinent depending on the practical application, as in [19]. The methodology may still be valid depending on the existence of a primitive optimal solution.

Second, bin-minimization is not the only objective function discussed in the literature. In particular BPPIFs arise in applications, in which the number of available bins is fixed, and the number of fragmented items has instead to be minimized [21]. Although different policies may be considered for penalizing fragmentations, appealing ones turn out to be equivalent. For instance, according to [4],

Proposition 4.2 ([4])**:** an optimal solution for the fragmentations minimization BPPIF always exists, that is primitive,

and therefore

Observation 4.1: the problem of minimizing the number of item fragments is equivalent to that of minimizing the overall number of fragmentations,

as each item is fragmented at most once, and hence the number of fragments is equal to the number of items plus the number of fragmentations.

We therefore consider the size preserving fragmentation-minimization BPPIF (fm-BPPSPF) problem modeled in [4]:

$$\min \quad \sum_{i \in I, j \in B} z_{ij} - |I| \tag{4.2}$$

$$(FM) \quad \text{s.t.} \quad 2.2, 2.5, 2.6, 2.7$$

$$\sum_{i \in I} w_i \cdot x_{ij} \leq C \qquad \forall j \in B \tag{4.3}$$

The objective function (4.2) minimizes the number of fragmentations, and equivalently the number of fragments. Constraints (4.3) ensure that the capacity of each bin is not exceeded.

In principle, this model is compact and can thus be directly managed by a generic solver. However, according to the experimental results obtained in [4], such an approach fails even on very small instances. The branch-and-price algorithm of [4] allows to solve about 40% more instances than CPLEX, but fails as the size of the instances increases.

Indeed we could obtain substantially better results by adapting the chain based methods detailed in the previous section. Still without loss of optimization power, by exploiting Proposition 4.2 we consider only primitive solutions, and the corresponding BPG. As before, let l_k be the length of any chain k in the BPG, that is in turn the number of fragmentations in the chain plus one. The fm-BPPSPF can therefore be modeled with a chain formulation, in which the objective function is to minimize the overall number of fragmentations:

$$\min \quad \sum_{k \in K} l_k - v_k \tag{4.4}$$

$$(FMC) \quad \text{s.t.} \quad 2.10, 2.12, 2.13 - 2.16$$

$$\sum_{k \in K} l_k \leq |K| \tag{4.5}$$

The objective function (4.4) minimizes the overall number of fragmentations by minimizing

the sum of the costs of the chains in the corresponding BPG. Constraints (4.5) ensure that all items are packed into at most $|K|$ bins.

As the compact models, also the chain model can be directly solved by a generic solver, but as before it is easy to prove that the continuous relaxations of both models FM and FMC do not offer significant lower bounds. Therefore, we adapt the Dantzig-Wolfe decomposition proposed in Subsection 2.2. In fact, by replacing in the continuous relaxations of formulation FMC the vectors (z_k, v_k, l_k) as indicated in (2.17), and by making explicit the vector indices, we obtain the following formulation:

$$\begin{aligned} \min \quad & \sum_{k \in K} \sum_{p \in \Gamma_k} y_k^p \cdot (\bar{l}_k^p - \bar{v}_k^p) \\ \text{s.t.} \quad & 2.19, 2.21, 2.22 \\ & \sum_{k \in K} \sum_{p \in \Gamma_k} y_k^p \cdot \bar{l}_k^p \leq |K| \end{aligned}$$

that after applying the same reformulation steps performed for model (2.18) – (2.22), yields the following:

$$\begin{aligned} & \min \quad \sum_{p \in \Gamma} y^p \cdot (\bar{l}^p - 1) && (4.6) \\ (F-MP) \quad & \text{s.t.} \quad 2.25, 2.27 \\ & \qquad -\sum_{p \in \Gamma} y^p \cdot \bar{l}^p \geq -|K| && (4.7) \end{aligned}$$

The objective function (4.6) minimizes the overall cost, given as the sum of fragmentations performed in each chain. Constraint (4.7) ensures that items are packed into at most $|K|$ bins.

As far as the quality of bounds is concerned, we can prove the following.

Proposition 4.3: The lower bound given by F-MP dominates that obtained by Dantzig-Wolfe decomposition of model FM.

The latter has been introduced in [4] as a basis for an exact branch-and-price algorithm: it consists of an extended formulation with one column for each feasible assignment pattern of items to *bins*. Intuitively, given an optimal MP solution $\tilde{y}^p$, we can run NF(f) on each pattern $p \in \Gamma$ of length l_p, build l_p assignment patterns of items to bins, corresponding to the l_p bins in the NF(f) solution, and select each of them for a value $\tilde{y}^p$. This solution is feasible for the extended formulation of [4]. Indeed, we experimentally observed it to be often suboptimal, thereby providing weaker bounds.

We remark that model F-MP, including (4.6), (2.25), (2.27) and (4.7), differs from model

MP, including (2.24)–(2.27), only for the exchange between the objective function and the constraint on the available resource: the first model limits the number of bins while minimizing the number of fragmentations. The second does the opposite. Indeed the dominance proof among bounds readily extends to bm-BPPIF models.

For completeness, we also mention that

Proposition 4.4: an optimal solution to fm-BPPSIF always exists, that is primitive.

The proof is similar to that of Proposition 4.2, additionally observing that a non-primitive optimal solution would just introduce more overhead than a corresponding primitive one. Therefore, a chain-based modeling of fm-BPPSIF can be obtained by combining the steps described above.

4.2 Extending algorithms

Overhead handing nicely fits in our framework. In fact, Constraint (4.1) can be easily transposed into the pricing problem and solved as a variant of VSKP in which bins capacity is $\hat{C} = C - \epsilon$, and a capacity of ϵ is set as consumed since the beginning. Due to Constraint (4.1) the weight of each item i is then $\hat{w}_i = w_i + \epsilon$. Once again, when $\epsilon = 0$, the pricing algorithm is exactly the one described in Section 3.2.

We could successfully adapt our column generation techniques also to fragment-minimization problems. In this case, the models differ in the objective function of the pricing problem: let λ and μ be respectively the vector of non negative dual variables corresponding to constraints (2.25) and (4.7); the reduced cost of each variable y^p is

$$\min_{p \in \Gamma} \bar{l}^p - 1 - \sum_{i \in I} \lambda_i \cdot \bar{z}_i^p + \mu \cdot \bar{l}^p.$$

By rewriting the objective function in maximization form, and by collecting the coefficients of terms $\bar{z}_i^p$ and $\bar{l}^p$, we obtain

$$\max \sum_{i \in I} \lambda_i \cdot \bar{z}_i^p - (\mu + 1) \cdot \bar{l}^p + 1.$$

that is still a VSKP, and can therefore be solved as explained in section 3.2.

Every other detail of the algorithm remains the same, with the only exception that instead of using ISSH as primal heuristic, it is more appropriate to directly include SSH as proposed in [4] when optimizing fragmentation-minimization problems.

5 Experimental results

We implemented our algorithms in C++, using the framework SCIP [1] version 3.0.2, keeping the default options but forcing single thread execution. In particular, that includes a full suite of general purpose primal heuristics. The LP subproblems were solved using the simplex algorithm implemented in CPLEX 12.4 [10]: the framework automatically switches between primal and dual methods. We refer to our exact branch-and-price algorithm as BPA in the remainder.

As a benchmark we considered the branch-and-cut implemented in CPLEX 12.4, using the mathematical programming models described in Section 2 and Section 4, and keeping again default settings besides forcing single thread execution. All the tests have been performed on a PC equipped with an Intel(R) Core2 Duo CPU E6850 at 3.00 GHz and 4 GB of memory.

As a first check, we tested our BPA on the dataset for the fm-BPPSPF introduced in [4], that consists of 180 instances divided in 18 classes that differ on instance size, distribution of the weights and amount of residual capacity in the bins. We included three benchmark algorithms: CPLEX using the compact model, CPLEX using the chain-based model, and the branch-and-price algorithm described in [4]. A time limit of one hour was given to each run.

In Table 1 we report for each solving method the number of instances solved to proven optimality within the time limit (S), the average duality gap on the remaining ones computed as $(UB - LB)/UB$ (Gap) and the average computing time (t). Our BPA outperforms the three benchmark algorithms by far, solving all the instances in fractions of second. It is also interesting to observe that chain-based models allow to obtain always better results when CPLEX is employed: on a few classes of instances, CPLEX using the chain-based models performs better than the algorithm of [4].

5.1 Dataset generation

Indeed, the design of a dataset being challenging, statistically significant and fair at the same time turned out to be an issue on its own. First, the results of [4] indicate that the ratio between item weights and capacity, and the amount of residual capacity are the features influencing most the computational behavior of both general purpose solvers and ad-hoc algorithms. Second, the behavior of general purpose solvers is strongly influenced by the number of available bins $|B|$. On the contrary, our pricing algorithms have a pseudo-polynomial time complexity in the capacity value C, and therefore such a parameter can influence the performance of BPA. Therefore a full control on all these parameters is needed to detect regularities.

Hence, we created a new dataset as follows. We considered instances including $|I| = 20, 50$

	Class		CPLEX			CPLEX chain model			Branch-and-price			BPA		
|I|	w.	cap.	S.	Gap(%)	t(s)	S.	Gap(%)	t(s)	S.	Gap(%)	t(s)	S.	Gap(%)	t(s)
10	big	tight	6	20.8	427.2	10	0.0	0.2	10	0.0	137.4	10	0.0	0.0
10	free	tight	10	0.0	36.3	10	0.0	0.0	10	0.0	0.4	10	0.0	0.0
10	small	tight	10	0.0	0.8	10	0.0	0.0	10	0.0	0.1	10	0.0	0.0
10	big	loose	10	0.0	140.4	10	0.0	0.4	10	0.0	0.2	10	0.0	0.0
10	free	loose	10	0.0	0.0	10	0.0	0.0	10	0.0	0.0	10	0.0	0.0
10	small	loose	10	0.0	0.0	10	0.0	0.0	10	0.0	0.0	10	0.0	0.0
15	big	tight	0	17.4	-	10	0.0	62.0	1	12.5	0.3	10	0.0	0.1
15	free	tight	0	56.7	-	10	0.0	8.4	6	16.7	6.0	10	0.0	0.1
15	small	tight	7	55.6	1,090.2	10	0.0	1.3	10	0.0	0.5	10	0.0	0.0
15	big	loose	0	65.0	-	8	37.5	1,231.1	10	0.0	5.0	10	0.0	0.1
15	free	loose	10	0.0	0.1	10	0.0	0.0	10	0.0	0.1	10	0.0	0.0
15	small	loose	10	0.0	0.0	10	0.0	0.0	10	0.0	0.0	10	0.0	0.0
20	big	tight	0	29.9	-	2	29.2	2,525.0	0	12.5	-	10	0.0	0.2
20	free	tight	0	82.3	-	1	57.4	2,891.4	3	16.7	124.7	10	0.0	0.1
20	small	tight	0	86.7	-	10	0.0	97.8	10	0.0	2.3	10	0.0	0.1
20	big	loose	0	93.3	-	0	63.3	-	3	16.7	1,822.9	10	0.0	0.1
20	free	loose	10	0.0	0.1	10	0.0	0.1	10	0.0	0.1	10	0.0	0.1
20	small	loose	10	0.0	0.0	10	0.0	0.0	10	0.0	0.1	10	0.0	0.0

Table 1: Solving fm-BPPSPF to proven optimality.

or 100 items. The capacity C of each bin was always fixed to 1000. Then we considered three types of weight distributions: let $\bar{w}$ and $\underline{w}$ be respectively upper and lower ends of the range of possible weight values

large: draw integers from a uniform distribution between $\underline{w} = 0.5 \cdot C$ and $\bar{w} = 0.9 \cdot C$

small: draw integers from a uniform distribution between $\underline{w} = 0.1 \cdot C$ and $\bar{w} = 0.5 \cdot C$

free: draw integers from a uniform distribution between $\underline{w} = 0.1 \cdot C$ and $\bar{w} = 0.9 \cdot C$

We initially considered fragmentations-minimization. With respect to the residual capacity, we considered two types of instances by changing the number of available bins:

tight: $|B| = \lceil \frac{\underline{w}+\bar{w}}{2} \cdot |I| \rceil$, that is the minimum expected number of bins needed to fractionally pack all the items,

loose: 10% of expected residual space, that is $|B| = \lceil \frac{1.0}{0.9} \cdot \frac{\underline{w}+\bar{w}}{2} \cdot |I| \rceil$.

Finally, the actual weights of tight instances were generated as follows. For the first $|I| - 1$ items we set the weights to be a random integer value chosen between $\underline{w}$ and $\bar{w}$, while the last item weight was given by the difference between $C \cdot |B|$ and the sum of the previously generated weights. The generation was repeated until the last weight was between $\underline{w}$ and $\bar{w}$. The actual weights of loose instances were generated similarly, but setting the weight of the last item to the difference between $\frac{1.0}{0.9} \cdot C \cdot |B|$ and the sum of the previously generated weights.

On bin-minimization tests, we fixed the maximum number of allowed fragmentations to $F = F^*/2$, where F^* is the optimal solution of the corresponding fragmentations-minimization instance. In fact, we found out instances with higher values of F to be trivial for our algorithms.

We created an instance class for each combination of instance size, weight distribution and residual capacity, and generated ten instances for each class, obtaining a dataset of 180 instances. We remark that, in this way, each class contains instances having homogeneous $|I|$, $|B|$ and C values.

5.2 Root lower bound

In a first round of experiments we compared the efforts for obtaining a lower bound for bm-BPPSPF problems, stopping at the root node of the branching tree with either CPLEX using the compact model, CPLEX using the chain-based model, and our BPA. In Table 2 we report, for each instance class and for each method, the time spent at the root node, and the average gap $(B^* - LB)/B^*$ between the corresponding lower bound LB and the best known bm-BPPSPF solution B^*. The results show that the chain-based model is in general the fastest

way to get a lower bound, but the BPA always gets the best gap, which usually is also the value of B^*.

	Class		Compact model		Chain model		BPA	
|I|	w.	cap.	Gap(%)	t(s)	Gap(%)	t(s)	Gap(%)	t(s)
20	big	tight	7.8	0.5	7.8	0.0	0.0	0.2
20	free	tight	9.1	0.3	9.1	0.0	0.0	0.1
20	small	tight	14.3	0.2	14.3	0.0	0.0	0.1
20	big	loose	16.7	0.6	16.7	0.1	0.0	0.1
20	free	loose	9.8	0.5	9.8	0.0	0.0	0.1
20	small	loose	0.0	0.2	0.0	0.0	0.0	0.1
50	big	tight	12.5	4.5	12.5	0.3	0.0	0.7
50	free	tight	3.8	4.0	3.8	0.3	0.4	1.5
50	small	tight	3.1	3.2	3.1	0.2	0.0	0.9
50	big	loose	20.0	5.6	20.0	0.4	0.0	0.8
50	free	loose	5.7	5.2	5.7	0.3	0.0	0.8
50	small	loose	0.0	3.1	0.0	0.2	0.0	0.9
100	big	tight	12.5	8.5	12.5	2.9	0.0	3.6
100	free	tight	2.0	8.0	2.0	2.2	0.0	7.8
100	small	tight	0.0	6.7	0.0	1.4	0.0	4.3
100	big	loose	21.3	9.2	21.3	3.0	0.0	4.6
100	free	loose	5.0	8.5	5.0	2.0	0.0	6.7
100	small	loose	0.0	6.3	0.0	1.5	0.0	14.1

Table 2: Computing lower bounds at the root node

5.3 Root upper bound

In a second round of experiments we compared the quality of the upper bounds for bm-BPPSPF obtained at the root node of the branching tree by CPLEX heuristics using either compact or chain-based models, and by our BPA. In Table 3 we report for each instance class and for each method, the average gap $(UB - B^*)/UB$ between the corresponding best upper bound UB and the best known bm-BPPSPF solution B^*. The results show that at root node, our algorithm always gets a better upper bound, which is usually already the optimal solution. For what concerns the execution of CPLEX, chain-based models clearly give better upper bounds than compact ones. BPA always offers the most accurate results.

5.4 Solving bm-BPPSPF to proven optimality

Third, we ran a comparison between CPLEX on both compact models and chain-based ones, and our BPA in solving bm-BPPSPF instances to proven optimality. A time limit of one hour was given to each run.

In Table 4 we report for each solving method the number of instances solved to proven optimality within the time limit (S), the average duality gap on the remaining ones computed as $(UB - LB)/UB$ (Gap), and the average computing time (t).

Our BPA solves all the 180 instances, while CPLEX solves only 21 and 29 of them, using the compact and the chain-based models, respectively. This test confirms the results of [4] on the hardness of BPPIF for generic solvers. Chain-based models perform better than compact ones, always yielding a smaller duality gap.

5.5 Solving Size-Increasing variants

In the last round of experiments we assessed the impact of size-increasing features both on the computational behavior of our algorithms and on the final solution costs. The analysis has been performed using BPA. No time limit was given to these test, in order to always obtain a global optimal solution. Nevertheless, no test exceeded one hour of computation. The overhead ϵ was set to be 1% of the bin capacity.

In Table 5 we report the average gap between the optimal solutions values obtained on the bm-BPPIF with the size preserving model (SP) and the ones obtained with the size-increasing variant (SI), computed as (SI - SP)/SP.

From the results we observe that overhead mildly worsens the quality of the solutions, while the execution times are actually insensitive of the overhead management policy.

As a computational stress test, we repeated the experiment by increasing the overhead ϵ to 10% of the bin capacity, although in practical applications such a large overhead would not be meaningful. In Table 6 we report our results.

Still, no optimization required more than one hour of computing time. It is first interesting to note that more aggressive overhead settings do not necessarily result in more difficult problems from the computing time point of view. The solutions quality, instead, is highly penalized by the large overhead: solutions are in a few cases more than 30% worse than their size-preserving counterpart. We also observed that the impact of high overhead increases as the average item size decrease. In fact, for small items, an overhead of $\epsilon = 0.1 \cdot C$ means an average growth of one third of each item.

We finally performed the same test on the fm-BPPIF. When setting large overhead ($\epsilon = 0.1 \cdot C$), the fraction of infeasible instances was too large to provide any meaningful result, since for most instances the sum of the items weight plus the overhead of each item was already greater than the available capacity. Therefore, in Table 7 we report the results for small overhead ($\epsilon = 0.01 \cdot C$) only.

All instances having tight capacity resulted infeasible by design: it is however interesting to

	Class		Compact model		Chain model		BPA	
|I|	w.	cap.	Gap(%)	t(s)	Gap(%)	t(s)	Gap(%)	t(s)
20	big	tight	19.4	0.5	6.1	0.0	0.6	0.2
20	free	tight	2.4	0.3	7.0	0.0	0.0	0.1
20	small	tight	7.2	0.2	0.0	0.0	0.0	0.1
20	big	loose	50.0	0.6	3.2	0.1	0.0	0.1
20	free	loose	29.5	0.5	9.7	0.0	0.0	0.1
20	small	loose	6.3	0.2	1.3	0.0	0.0	0.1
50	big	tight	81.4	4.5	5.8	0.3	0.0	0.7
50	free	tight	20.9	4.0	11.5	0.3	0.4	1.5
50	small	tight	21.7	3.2	9.2	0.2	3.1	0.9
50	big	loose	100.0	5.6	0.2	0.4	0.0	0.8
50	free	loose	100.0	5.2	12.0	0.3	1.4	0.8
50	small	loose	11.5	3.1	8.0	0.2	0.0	0.9
100	big	tight	100.0	8.5	23.3	2.9	0.0	3.6
100	free	tight	58.1	8.0	48.2	2.2	1.1	7.8
100	small	tight	57.8	6.7	19.4	1.4	3.2	4.3
100	big	loose	100.0	9.2	10.0	3.0	0.0	4.6
100	free	loose	100.0	8.5	17.9	2.0	1.8	6.7
100	small	loose	30.9	6.3	36.8	1.5	1.3	14.1

Table 3: Computing upper bounds at the root node

	Class		Compact model			Chain model			BPA		
|I|	w.	cap.	S.	Gap(%)	t(s)	S.	Gap(%)	t(s)	S.	Gap(%)	t(s)
20	big	tight	0	10.2	-	1	8.0	3,448.9	10	0.0	0.2
20	free	tight	0	9.1	-	0	9.1	-	10	0.0	0.1
20	small	tight	1	14.3	335.9	0	14.3	-	10	0.0	0.1
20	big	loose	0	16.7	-	0	16.7	-	10	0.0	0.1
20	free	loose	0	9.8	-	4	9.0	0.1	10	0.0	0.1
20	small	loose	10	0.0	0.2	10	0.0	0.1	10	0.0	0.1
50	big	tight	0	24.1	-	0	12.5	-	10	0.0	0.7
50	free	tight	0	6.0	-	0	3.8	-	10	0.0	1.6
50	small	tight	0	6.3	-	0	6.3	-	10	0.0	0.9
50	big	loose	0	36.0	-	0	20.0	-	10	0.0	0.8
50	free	loose	0	7.4	-	0	5.7	-	10	0.0	0.8
50	small	loose	10	0.0	22.2	10	0.0	0.3	10	0.0	0.9
100	big	tight	0	75.6	-	0	12.5	-	10	0.0	3.5
100	free	tight	0	8.4	-	0	2.3	-	10	0.0	11.2
100	small	tight	0	5.9	-	0	3.2	-	10	0.0	9.4
100	big	loose	0	100.0	-	0	21.3	-	10	0.0	4.6
100	free	loose	0	9.2	-	0	5.0	-	10	0.0	7.6
100	small	loose	0	3.8	-	4	3.2	1,321.6	10	0.0	53.2

Table 4: Solving bm-BPPSPF to proven optimality.

	Class		No overhead	Fragment overhead	
|I|	w.	cap.	t(s)	Gap(%)	t(s)
20	big	tight	0.2	2.0	0.1
20	free	tight	0.1	0.0	0.1
20	small	tight	0.1	0.0	0.1
20	big	loose	0.1	0.0	0.1
20	free	loose	0.1	2.7	0.1
20	small	loose	0.1	0.0	0.1
50	big	tight	0.7	0.0	1.0
50	free	tight	1.6	0.0	1.9
50	small	tight	0.9	3.3	1.1
50	big	loose	0.8	0.0	0.9
50	free	loose	0.8	3.3	0.8
50	small	loose	0.9	0.0	2.5
100	big	tight	3.5	0.0	5.8
100	free	tight	11.2	2.0	11.2
100	small	tight	9.4	3.3	9.0
100	big	loose	4.6	0.0	5.7
100	free	loose	7.6	1.9	7.4
100	small	loose	53.2	3.3	52.8

Table 5: Solving bm-BPPSIF with small overhead.

	Class		No overhead	Fragment overhead	
|I|	w.	cap.	t(s)	Gap(%)	t(s)
20	big	tight	0.2	18.5	0.1
20	free	tight	0.1	18.2	0.1
20	small	tight	0.1	28.6	0.1
20	big	loose	0.1	5.6	0.1
20	free	loose	0.1	26.1	0.1
20	small	loose	0.1	30.0	0.1
50	big	tight	0.7	12.5	1.2
50	free	tight	1.6	23.5	1.1
50	small	tight	0.9	35.6	0.8
50	big	loose	0.8	4.0	0.8
50	free	loose	0.8	22.6	0.6
50	small	loose	0.9	31.3	1.0
100	big	tight	3.5	11.1	16.2
100	free	tight	11.2	23.7	8.3
100	small	tight	9.4	36.7	12.6
100	big	loose	4.6	3.3	5.2
100	free	loose	7.6	22.5	4.0
100	small	loose	53.2	34.3	20.0

Table 6: Solving bm-BPPSIF with large overhead.

note that infeasibility is detected quickly by our algorithms. In the remaining instances, the penalties are in a few cases very high, confirming the behavior on bm-BPPSPF. The computing time showed to be not an issue: all instances were solved within five minutes, even if by introducing overhead the required CPU time slightly increased.

	Class		No overhead	Fragment overhead	
|I|	w.	cap.	t(s)	Gap(%)	t(s)
20	big	tight	0.2	-	0.1
20	free	tight	0.1	-	0.1
20	small	tight	0.1	-	0.0
20	big	loose	0.1	5.0	0.1
20	free	loose	0.1	40.0	0.1
20	small	loose	0.0	0.0	0.0
50	big	tight	8.4	-	0.9
50	free	tight	4.3	-	0.5
50	small	tight	1.5	-	0.3
50	big	loose	0.8	0.0	2.5
50	free	loose	0.7	15.0	1.2
50	small	loose	0.2	0.0	0.2
100	big	tight	235.7	-	6.1
100	free	tight	252.6	-	3.4
100	small	tight	35.6	-	1.4
100	big	loose	5.0	0.0	15.0
100	free	loose	5.5	0.0	10.5
100	small	loose	0.8	0.0	0.8

Table 7: Solving fm-BPPSIF with small overhead.

6 Conclusions

In this paper we tackled several variants of BPPIF with a unified approach: after collecting and deriving some properties of BPPIF solutions, we proposed new mathematical programming models that avoid the use of fractional variables. Then, by using Dantzig-Wolfe decomposition we also introduced an extended formulation and exploited column generation with ad-hoc pricing algorithms, dual cuts, heuristics and implicit enumeration to design a new exact branch-and-price algorithm.

That algorithm proved to be first of all flexible, as minor or no modifications are needed to adapt it to bin-minimization or fragmentations-minimization, and to size preserving or size increasing overhead management policies.

Our experimental campaign revealed that state-of-art general purpose solvers like CPLEX find it beneficial to use our new models, but still fail in optimizing even instances of very small

size, regardless on the BPPIF variant considered. Instead, our algorithms proved to be very effective, being able to solve to proven optimality all the instances in our datasets in minutes of computation for any BPPIF variant, thus outperforming both CPLEX and previous approaches from the literature.

Finally, having such an effective tool, we could perform an experimental comparison on overhead handling, showing that overhead can be tackled explicitly without additional computing effort, and with limited increase in solution costs, unless its amount becomes a substantial share of the overall available capacity.

Of course, other BPPIF variants might be pertinent in practice, as the minimization of bins including fragmented items, the minimization of fragmented items, or more involved overhead management schemes. We are currently investigating on how far our theoretical results and algorithms might be extended.

References

[1] T. Achterberg. Scip: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1 – 41, 2009.

[2] C. Archetti, N. Bianchessi, and M.G. Speranza. Branch-and-cut algorithms for the split delivery vehicle routing problem. *European Journal of Operational Research*, 238(3):685–698, 2014.

[3] M. Casazza and A. Ceselli. Improved algorithms for bin packing problems with item fragmentation. In *Contribution at EURO XXVI, Rome – July 2013*, 2013.

[4] M. Casazza and A. Ceselli. Mathematical programming algorithms for bin packing problems with item fragmentation. *Computers & Operations Research*, 46(0):1 – 11, 2014.

[5] F. Chung, R. Graham, J. Mao, and G. Varghese. Parallelism versus memory allocation in pipelined router forwarding engines. *Theory of Computer Systems*, 39(6):829 – 849, 2006.

[6] J.F. Cordeau, G. Laporte, M.W.P. Savelsbergh, and D. Vigo. *Vehicle Routing*, volume 14 of *Handbooks in Operations Research and Management Science*, pages 367 – 428. Elsevier, 2007.

[7] G.B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations research*, 8:101–111, 1960.

[8] J.M.V. de Carvalho. Lp models for bin packing and cutting stock problems. *European Journal of Operational Research*, 141(2):253 – 273, 2002.

[9] M. Dell'Amico, J. C. Díaz Díaz, and M. Iori. The bin packing problem with precedence constraints. *Operations Research*, 60:1491–1504, 2012.

[10] CPLEX development team. Ibm ilog cplex optimization studio: Cplex user's manual - version 12 release 4. Technical report, IBM corp., 2011.

[11] L. Epstein, A. Levin, and R. van Stee. Approximation schemes for packing splittable items with cardinality constraints. *Algorithmica*, 62(1 – 2):102 – 129, 2012.

[12] L. Epstein and R. van Stee. Improved results for a memory allocation problem. *Theory of Computing Systems*, 48(1):79 – 92, 2011.

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

[14] T. Gschwind and S. Irnich. Dual inequalities for stabilized column generation revisited. Technical Report Working Paper n. 1407, Gutenberg School of Management and Economics, Mainz Univ., 2014.

[15] S. C. Ho and W. Y. Szeto. Solving a static repositioning problem in bike-sharing systems using iterated tabu search. *Transportation Research Part E: Logistics and Transportation Review*, 69:180–198, 2014.

[16] C.A. Mandal, P.P. Chakrabarti, and S. Ghose. Complexity of fragmentable object bin packing and an application. *Computers and Mathematics with Applications*, 35(11):91––97, 1998.

[17] S. Martello and P. Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., New York, NY, USA, 1990.

[18] M. A. Alba Martínez, F. Clautiaux, M. Dell'Amico, and M. Iori. Exact algorithms for the bin packing problem with fragile objects. *Discrete Optimization*, 10(3):210–223, 2013.

[19] N. Menakerman and R. Rom. Bin packing with item fragmentation. In *Proceedings of the 7th International Workshop on Algorithms and Data Structures, in: Lecture Notes in Computer Science*, pages 313–324, 2001.

[20] M. Schneider, A. Stenger, and D. Goeke. The electric vehicle-routing problem with time windows and recharging stations. *Transportation Science*, 48(4):500–520, 2014.

[21] S. Secci, A. Ceselli, F. Malucelli, A. Pattavina, and B. Sansò. Direct optimal design of a quasi-regular composite-star core network. In *IEEE Proc. of DRCN*, pages 7–10, 2007.

[22] H. Shachnai, T. Tamir, and O. Yehezkely. Approximation schemes for packing with item fragmentation. In Thomas Erlebach and Giuseppe Persinao, editors, *Approximation and Online Algorithms*, volume 3879 of *Lecture Notes in Computer Science*, pages 334–347. Springer Berlin / Heidelberg, 2006.

[23] N. Skorin-Kapov. Routing and wavelength assignment in optical networks using bin packing based algorithms. *European Journal of Operational Research*, 177(2):1167–1179, 2007.

[24] José M. Valério de Carvalho. Using extra dual cuts to accelerate column generation. *INFORMS Journal on Computing*, 17(2):175–182, 2005.