Discrete Applied Mathematics 163 (2014) 73–86

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

Lower and upper bounds for the Bin Packing Problem with Fragile Objects

François Clautiaux [a], Mauro Dell'Amico [b], Manuel Iori [b,*], Ali Khanafer [a]

[a] Université de Lille 1, INRIA Lille Nord Europe, Parc de la Haute Borne, 59655 Villeneuve d'Ascq, France

[b] University of Modena and Reggio Emilia, Via Amendola 2, 42122 Reggio Emilia, Italy

ARTICLE INFO

Article history:
Received 16 February 2011
Received in revised form 23 November 2011
Accepted 17 April 2012
Available online 18 May 2012

Keywords:
Bin Packing Problem
Fragile Objects
Variable Neighborhood Search
Column generation
Dual cuts

ABSTRACT

We are given a set of items, each characterized by a weight and a fragility, and a large number of uncapacitated bins. Our aim is to find the minimum number of bins needed to pack all items, in such a way that in each bin the sum of the item weights is less than or equal to the smallest fragility of an item in the bin. The problem is known in the literature as the Bin Packing Problem with Fragile Objects, and appears in the telecommunication field, when one has to assign cellular calls to available channels by ensuring that the total noise in a channel does not exceed the noise acceptance limit of a call.

We propose several techniques to compute lower and upper bounds for this problem. For what concerns lower bounds, we present combinatorial techniques with guaranteed worst case and a more complex bound based on a column generation algorithm. We also present a technique to compute, in a fast heuristic way, dual information that is used to strengthen the convergence of the column generation. For what concerns upper bounds, we present a large set of constructive heuristics followed by a Variable Neighborhood Search algorithm. Our heuristic techniques are aimed at both computing upper bounds and strengthening the behavior of the lower bounds in a *matheuristic* fashion. Extensive computational tests show the effectiveness of the proposed algorithms.

1. Introduction

We are given n items, each having weight w_j and fragility f_j ($j = 1, \ldots, n$), and a large number of uncapacitated bins. Our aim is to find the minimum number of bins needed to pack all items, in such a way that in each bin the sum of the item weights is less than or equal to the smallest fragility of an item in the bin. More formally, let $J(i)$ denote the set of items assigned to a bin i in a given solution. The solution is feasible if

$$\sum_{j \in J(i)} w_j \leq \min_{j \in J(i)} \{f_j\} \tag{1}$$

for any bin i. The problem is known as the *Bin Packing Problem with Fragile Objects* (BPPFO), and was introduced by Bansal et al. [2]. Fig. 1 reports a simple BPPFO example with eight items (left part of the figure: the white bars denote the weights, the white plus the gray bars denote the fragilities) and the corresponding optimal solution (right part).

The BPPFO is clearly NP-complete, because it generalizes the classical *Bin Packing Problem* (BPP). In the BPP we are given n items of weight w_j ($j = 1, \ldots, n$) and a large number of bins of capacity C, with the aim of packing the items in the

* Corresponding author. Tel.: +39 0522 522653; fax: +39 0522 522312.
E-mail addresses: francois.clautiaux@univ-lille1.fr (F. Clautiaux), mauro.dellamico@unimore.it (M. Dell'Amico), manuel.iori@unimore.it (M. Iori), Ali.Khanafer@inria.fr (A. Khanafer).

0166-218X/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2012.04.010

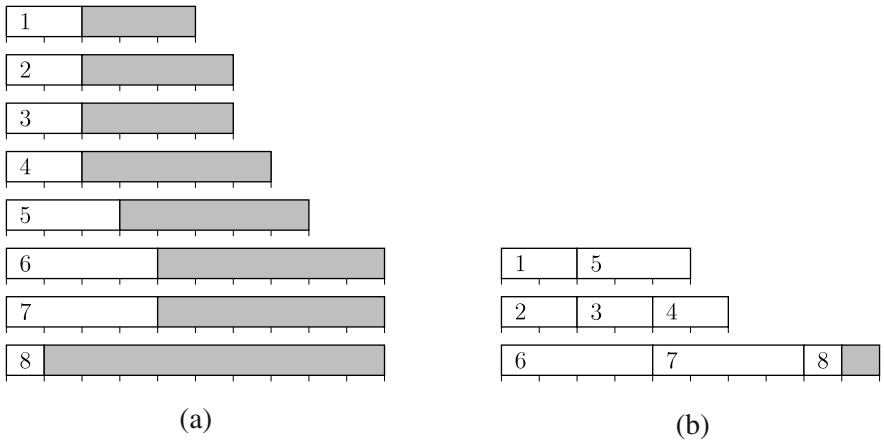

Fig. 1. (a) A simple BPPFO instance and (b) its optimal solution.

minimum number of bins without exceeding the bin capacity. The BPP can thus be seen as a particular BPPFO where all item fragilities are set to C.

The BPPFO arises in the telecommunication field and in particular in the allocation of cellular calls to frequency channels (see Bansal et al. [3] and Chan et al. [7]). In *Code Division Multiple Access* (CDMA) systems, a limited number of frequency channels is given. Each channel has a capacity much larger than the bandwidth requirement of a single call, so it is possible to assign many calls to the same channel. However, such assignment may produce interferences among the calls sharing the same channel, and may result in a loss of quality of the communication. In particular, each call is characterized by a certain *noise* that it produces, and by a certain *tolerance* with respect to the total noise in the channel. The call has to be assigned to a frequency channel in which the total noise does not exceed its tolerance. To model the above telecommunication problem as a BPPFO, it is enough to associate each frequency channel to a bin, and each call to an item having weight equal to the call noise and fragility equal to the call tolerance. The solution of the BPPFO gives the minimum number of frequency channels required to allocate all calls.

The literature on the BPPFO is still small. Bansal et al. [3] present approximation schemes and a probabilistic analysis. They consider approximations both with respect to the number of bins and to the fragility of a bin. They present results for the general BPPFO and for a special case, denoted the frequency allocation problem, in which weight and fragility are strictly correlated one to the other. Chan et al. [7] consider the on-line version of the BPPFO in which an item is available only after the previous item has been packed. They study two cases in which the ratio between the maximum and the minimum fragility is respectively bounded or unbounded, and present algorithms with asymptotic competitive ratios.

The literature on the BPP is, on the other hand, very large. We refer the interested reader to the surveys by Coffman et al. [9], Valério de Carvalho [28] and Clautiaux et al. [8]. For what concerns the use of optimization techniques in general frequency assignment problems on networks, we refer to the survey by Aardal et al. [1].

This paper is devoted to the presentation of algorithms to compute lower and upper bounds for the BPPFO. We are particularly interested in the use of heuristic techniques, not only for providing valid upper bounds (as usually happens), but also for strengthening the behavior of the lower bounding techniques. This can be seen as a *matheuristic* technique (see, e.g., Maniezzo et al. [20] and Hansen et al. [15]), in the sense that the heuristic algorithms are used to speed up the computation and/or improve the robustness of the optimal techniques.

The reminder of the paper is organized as follows. To clearly state the problem we start by introducing in Section 2 some notation, a compact mathematical formulation and a basic preprocessing technique. In Section 3 we provide a set of lower bounds that exploit the combinatorial structure of the problem and we assess their worst case performances. In Section 4 we propose a large set of upper bounds, including constructive heuristics, local search procedures and a Variable Neighborhood Search algorithm. In Section 5 we present a set covering formulation and provide a column generation approach to solve the linear relaxation of this formulation. We also describe a way to speed up the convergence of the column generation, by making use of some dual information that is computed quickly and heuristically. The algorithms are tested on a large set of instances in Section 6 and conclusions are finally drawn in Section 7.

2. Some preliminaries

Throughout the paper we suppose that w_j and f_j are positive integers ($j = 1, \ldots, n$). We also suppose, without loss of generality, that $w_j \leq f_j$ ($j = 1, \ldots, n$), and that items are sorted according to non-decreasing values of fragility, breaking ties by non-increasing values of weight. When no confusion arises, we use equivalently the terms object and item.

We define here two combinatorial optimization problems that are important because they are at the basis of several algorithms that will be discussed in the next sections:

- In the 0–1 *Knapsack Problem* (KP01), we are given n items with profit p_j and weight w_j ($j = 1, \ldots, n$) and a single bin of capacity C, with the aim of determining a subset of items of largest total profit whose total weight does not exceed C;
- In the 0–1 *Knapsack Problem with Fragile Objects* (KP01FO), we are given n items with profit p_j, weight w_j and fragility f_j ($j = 1, \ldots, n$) and a single uncapacitated bin, with the aim of determining a subset of items of largest total profit whose total weight does not exceed the fragility of any item in the bin (according to (1)).

We solve the KP01 through the efficient procedure COMBO by Martello et al. [21], and the KP01FO through a commercial software applied to the mathematical model described in Section 5.1.

2.1. A compact mathematical model

We define y_i as a binary variable taking value 1 *if item i is the item with smallest fragility in the bin in which it is packed*, 0 otherwise ($i = 1, \ldots, n$). We also define x_{ji} as a binary variable taking value 1 *if item j is assigned to the bin having item i as item with smallest fragility* (bin i for short in the following), 0 otherwise ($i = 1, \ldots, n, j = i + 1, \ldots, n$). (Remind that due to the items ordering $f_i \leq f_j$ for $i < j$, so variables x_{ji} have no sense for $j < i$.) The BPPFO can be modeled as the following *integer linear program* (ILP):

$$\min \sum_{i=1}^{n} y_i \tag{2}$$

$$y_j + \sum_{i=1}^{j-1} x_{ji} = 1 \quad j = 1, \ldots, n \tag{3}$$

$$\sum_{j=i+1}^{n} w_j x_{ji} \leq (f_i - w_i) y_i \quad i = 1, \ldots, n \tag{4}$$

$$x_{ji} \leq y_i \quad i = 1, \ldots, n,\ j = i + 1, \ldots, n \tag{5}$$

$$y_i \in \{0, 1\} \quad i = 1, \ldots, n \tag{6}$$

$$x_{ji} \in \{0, 1\} \quad i = 1, \ldots, n,\ j = i + 1, \ldots, n. \tag{7}$$

Constraints (3) impose that either an item is the one of smallest fragility in its bin, or it is assigned to a bin containing an item with smaller fragility. Constraints (4) require that the sum of the weights of the items packed in a bin does not exceed the smallest fragility in the bin (recall items are sorted according to non-decreasing values of fragility). Constraints (5) are used to tighten the model linear relaxation.

We spend a few words to notice the fact that Model (2)–(7) shares some similarities with that proposed by Ceselli and Righini [6] for the *Ordered Open-End Bin Packing Problem* (OOEBPP). In the OOEBPP the input consists of a BPP instance and of a total order among the items in the instance. The last (according to the input order) item among those packed in the same bin is not forced to fit completely within the bin capacity, but it is allowed to fill it by just one unit.

Despite the similarities in the models, the BPPFO and the OOEBPP are quite different problems in practice. The rationale behind the OOEBPP is to have a very large last item in each bin, so as to save space for other items. In the BPPFO, on the contrary, the capacity (i.e., fragility) constraint has to be completely satisfied, and it is quite usual to have similar items, especially in terms of fragility, packed together in the same bin. There is a single case in which the two problems are equivalent: an OOEBPP instance with weights $\overline{w}_j$, such that $\overline{w}_j \geq \overline{w}_{j+1}, j = 1, \ldots, n-1$, and total order $1, 2, \ldots, n$ is equivalent to a BPPFO instance with weights $w_j = \overline{w}_{n-j+1}$ and fragilities $f_j = w_j + C - 1,\ j = 1, \ldots, n$.

2.2. A preprocessing technique

In the BPP a classical preprocessing idea (see, e.g., Boschetti and Mingozzi [5]) is to compute the subset of items of maximal weight that can be packed together with a given item in a bin. Let w^* be the residual space in the bin after the given item, say j, has been packed in the bin with the subset of items of maximal weight. If $w^* > 0$, then one knows that in any feasible solution there will be a residual space in the bin containing item j. Hence w_j can be lifted to $w_j + w^*$.

Unfortunately this reasoning does not apply directly to the BPPFO. We can prove this by means of a simple counterexample with two items: item 1 has $w_1 = 2$ and $f_1 = 4$, and item 2 has $w_2 = 2$ and $f_2 = 5$. The two items can be clearly packed in a single bin. If we preprocessed item 2 as described above, we would obtain a residual space $w^* = f_2 - w_1 - w_2 = 1$. But if we set $w_2 = w_2 + w^* = 3$, then we would lose the optimal solution.

We can, however, adapt the above idea to the BPPFO as follows.

Proposition 1. *If w^* is the minimum residual space in any bin containing item j, then the following transformation can be applied to a BPPFO instance without affecting its optimal solution value:*

$$f_j \mapsto f_j - w^*. \tag{8}$$

Proof. Suppose, by contradiction, that there is a set S of items containing item j that can be feasibly packed in a bin when the fragility of j is f_j, but cannot be packed after the fragility of j is updated as in (8). Define $\widetilde{w}$ the total weight of the items in this set. The infeasibility implies $\widetilde{w} > f_j - w^*$. Hence the residual space left by this set in a bin would be $(\min_{k\in S}\{f_k\} - \widetilde{w}) \leq (f_j - \widetilde{w}) < f_j - f_j + w^* = w^*$, which contradicts the fact that w^* is the minimum remaining space in any bin containing item j. □

Proposition 1 can be implemented as follows. We first define the *compatible set* of an item j as $\gamma(j) = \{k \in \{1, \ldots, n\}, k \neq j : w_k + w_j \leq \min\{f_k, f_j\}\}$, for $j = 1, \ldots, n$. Then for any item j we solve a particular KP01FO in which: (i) the item set is $\gamma(j) \cup \{j\}$, (ii) item j is forced to belong to the solution, and (iii) the objective function minimizes the residual space (i.e., w^*) in the bin (see Section 5.1 for further implementation details). If $w^* > 0$ we update the fragility using (8).

Note that if all items in $\gamma(j) \cup \{j\}$ can be packed in a single bin, then we pack them and remove them from the instance without any need to solve the KP01FO. Note also that an update occurring to an item may change the compatible sets of other items and thus lead to further improvements. For this reason, we re-execute the procedure just outlined until no improvement is found for any item.

3. Combinatorial lower bounds

We provide a set of fast lower bounding algorithms that are derived from observations on the combinatorial structure of the BPPFO. In the following we use L both to define a lower bounding procedure and the value produced by the procedure itself.

To evaluate the performance that a lower bounding procedure may guarantee, we define its *worst case performance ratio* as a real value r such that $r \leq L(I)/z(I)$ for any instance I of the problem considered, with $z(I)$ being the optimal solution value of instance I and $L(I)$ the value produced by the lower bounding procedure on such instance. We say that a worst case performance ratio is *tight* if the value r is reached for at least one instance. If the context is sufficiently clear, we use L instead of $L(I)$ and z instead of $z(I)$.

3.1. Fractional lower bounds

The first simple lower bound we consider is obtained by relaxing the fragility requirements, so as to convert a BPPFO instance into a BPP one. To do this, we compute the maximum fragility $f_{\max} = \max_{j=1,\ldots,n}\{f_j\}$, and set the bin capacity C to $f_{\max}$. Any lower bound for the BPP instance obtained in this way is a valid lower bound for the original BPPFO instance. In particular the immediate bound from the BPP continuous relaxation is:

$$L_0 = \left\lceil \sum_{j=1}^{n} \frac{w_j}{f_{\max}} \right\rceil. \tag{9}$$

We note that the worst case performance ratio of L_0 is arbitrarily bad. It is enough to consider a BPPFO instance with n items, the first $n-1$ having weight 1 and fragility 1 and the last one having weight 1 and fragility n. The optimal solution has value $z = n$ because each item has to be packed alone in a bin, whereas $L_0 = 1$. When n tends to infinity, the worst case of L_0 tends to 0.

The result above was improved by Chan et al. [7], who proves that the following is a valid lower bound:

$$L_1 = \left\lceil \sum_{j=1}^{n} \frac{w_j}{f_j} \right\rceil. \tag{10}$$

Also for L_1 we prove that the worst case performance ratio is arbitrarily bad. It is enough to consider an instance with n items having weight n^{j-1} and fragility n^j, $j = 1, \ldots, n$. Each item in this instance has to be packed alone in a bin, whereas

$$L_1 = \sum_{j=1}^{n} \frac{n^{j-1}}{n^j} = \sum_{j=1}^{n} \frac{1}{n} = 1.$$

When n tends to infinity, the worst case of L_1 tends to 0.

The result above may be further improved using the so called *fractional lower bound*, presented by Bansal et al. [3] and outlined in Algorithm 1. The idea is to pack items in bins according to a non-decreasing fragility order. Whenever an item j does not fit entirely into a bin, only a portion of j is packed in the current bin, then a new bin having fragility equal to f_j is opened and filled with the remaining portion. We define L_2 the lower bound value produced by Algorithm 1.

Proposition 2. *The worst case performance ratio of L_2 is $1/2$ and the value is tight.*

Proof. The proof is somehow implicit in Bansal et al. [3], but for sake of clarity we report it here in a more concise form. We construct a heuristic solution using $2L_2$ bins starting from the assignments of items to bins obtained using Algorithm 1. The

input : n items sorted by non-decreasing fragility (breaking ties by non-increasing weight)
output: Lower bound value L_2

$L_2 = 1$;
$f_{res} = f_1 - w_1$;
for $j = 2$ **to** n **do**
  if $w_j \leq f_{res}$ **then**
    $f_{res} = f_{res} - w_j$;
  else
    $w_{res} = w_j - f_{res}$;
    $L_2 = L_2 + 1$;
    $f_{res} = f_j - w_{res}$;
  end
end
return L_2;

Algorithm 1: Fractional lower bound (L_2)

first bin has at most one fractional item. If this is the case, we remove the two portions of the item from the first and the second bin, and pack this item alone into a new bin. We reiterate with the second bin, that now has at most one fractional item, and so on. For each of the L_2 bins used by Algorithm 1, we open at most one new bin, so the heuristic solution uses $2L_2$ bins and the worst case performance ratio is at most $L_2/2L_2 = 1/2$.

To see that the performance is tight, it is enough to consider an instance with an even number of items, each having weight equal to $C/2 + \varepsilon$ (with ε being a small positive value) and fragility equal to C. The optimal solution uses $z = n$ bins, whereas $L_2 = n/2 + 1$. When n tends to infinity, the ratio between L_2 and z tends to $1/2$. □

3.2. Lower bounds based on incompatibilities among items

We associate to our BPPFO instance a *Conflict Graph* $G = (V, E)$ constructed as follows. Each vertex $j \in V$ is associated to item j ($j = 1, \ldots, n$). An edge (j, k) belongs to E if the two items j and k cannot be packed together in the same bin, i.e., if $w_j + w_k > \min\{f_j, f_k\}$. Using a notation common to other articles in the literature, we say that these two items are *in conflict*.

We first note that the chromatic number of G is a valid lower bound value for the original problem. Hence, any valid lower bound on the chromatic number is also a valid BPPFO lower bound. We use this observation as follows: let $K \subseteq V$ denote a clique in G, then

$$L_k = |K| \tag{11}$$

is a valid BPPFO lower bound. It is easy to see that the worst case performance of L_K is arbitrarily bad. This can be obtained by considering a BPPFO instance with no conflicts at all. For any solution value $z(I)$, the value of L_K is always 1.

We improve the above observation by using a *constrained packing lower bound* (L_{cp} in the following). This bound has been originally developed by Gendreau et al. [11] for the *Bin Packing Problem with Conflicts* (BPPC), which is a particular BPP in which some prespecified pairs of items may be in conflict and cannot be packed in the same bin. The algorithm used to determine L_{cp} starts by computing a maximal clique K of G through the standard greedy algorithm and initializing a bin for each item in K. Then it fractionally assigns the items in $V \setminus K$ to the opened bins, by solving a transportation problem that takes into consideration both conflicts and weights. All items (or fractions of items) that do not fit in the opened bins are stored in new bins by computing the classical continuous BPP lower bound (i.e., by disregarding conflicts and allowing to fraction the items). Let q denote the value produced by the continuous lower bound. We obtain

$$L_{\text{cp}} = |K| + q. \tag{12}$$

We adapt L_{cp} to our problem by converting the original BPPFO instance into a BPPC one. This can be done by (1) constructing the conflict graph G as outlined above and (2) setting $f_j = f_{\max}$ for $j = 1, \ldots, n$.

Simple improvements on the computation of L_{cp} have been proposed in the literature. In particular, Fernandes Muritiba et al. [10] use an improved heuristic algorithm instead of the standard greedy to build K, whereas Khanafer et al. [19] use the fact that some items in $V \setminus K$ may be in conflict to impose an upper bound on the total flow that can be sent to any of the $|K|$ initialized bins. In our experiments we use the implementation by Khanafer et al. [19]. Note that $L_{\text{cp}} \geq L_k$ when the two bounds use the same algorithm to compute the clique (as happens in our case, where we use the greedy algorithm by Johnson [18]). However, also the worst case performance of L_{cp} is arbitrarily bad. It is enough to consider a BPPFO instance with no conflicts at all and a large value of $f_{\max}$.

4. Upper bounds

To produce a heuristic solution of good quality within a limited computational effort, we use greedy upper bounding techniques, described in Section 4.1, and a Variable Neighborhood Search, described in Section 4.2.

4.1. Greedy heuristics

We developed a large set of greedy heuristics that are mainly obtained by modifying well-known heuristics from the BPP and the Vertex Coloring literature, so as to fit the BPPFO constraints.

The X-Fit heuristic

The classical *First Fit* algorithm was developed by Johnson [17] for the BPP and works as follows: sort items according to a given order, pack the first item in the first bin, then pack each subsequent item in the first feasible (with respect to capacity constraint) bin, if any, otherwise open a new bin and pack the item in such bin. Adapting the First Fit to the BPPFO simply requires to replace the capacity constraint with the fragility requirement. This can be done by (i) setting the capacity of a bin to the fragility of the first item packed in the bin, and (ii) updating the fragility of a bin whenever a new item is packed in it.

We also adapted to the BPPFO the other classical *Best Fit*, *Worst Fit* and *Next Fit* algorithms. We obtained in total 12 heuristic solutions by considering 3 different orderings of the items, namely: (i) non-decreasing values of f_j, breaking ties by non-increasing values of w_j; (ii) non-increasing values of w_j, breaking ties by non-decreasing values of f_j; (iii) non-decreasing values of the ratio f_j/w_j. We denote U_{xf} the minimum upper bound value produced by these attempts.

The knapsack-based heuristic

The idea that we used in Section 2.2 for preprocessing may also be used in a simpler iterative fashion to produce a feasible solution. In particular, at each iteration let j be the item with smallest fragility among the items still to be packed, and let $\gamma(j)$ be its compatible set. We solve a KP01 with item set $\gamma(j)$, items profits equal to their weights and bin capacity equal to $f_j - w_j$. We pack in a bin j and the items belonging to the optimal KP01 solution we just found. We then reiterate with the remaining items, until all items are packed. We denote U_{kp} the resulting upper bound value. Note that by selecting at each iteration the item j with smallest fragility we are allowed to solve a KP01 instead of a KP01FO.

The clique-based heuristic

In Section 3.2 we obtained a valid lower bound by constructing the conflict graph G and greedily computing a maximal clique K of G. We extend this approach and also compute a valid upper bound value as follows. We select the first item $j \in K$, determine its compatible set $\gamma(j)$ and find the subset $\overline{\gamma}(j)$ of items of largest total weight that can be packed side by side with j (by solving a KP01FO). We pack j and $\overline{\gamma}(j)$ in a bin and then reiterate with the next item in the clique, until all $|K|$ items have been processed. If all items in the instance have been packed in the $|K|$ bins, then we have found an optimal solution. If instead some items are still unpacked, we pack them in a First Fit fashion. We denote U_{cb} the resulting upper bound value.

The merging heuristic

The basic idea behind this heuristic strategy is to iteratively merge a pair of items into a unique composite item, until no more merging exists. The final set of items obtained in this way represents a heuristic solution in which each composite item corresponds to a feasible packing of a bin. We proceed as follows: (i) we select two *compatible* items j and k such that $w_j + w_k \leq \min\{f_j, f_k\}$ and merge them into a new item, say ℓ, of size $w_\ell = w_j + w_k$ and fragility $f_\ell = \min\{f_j, f_k\}$; (ii) we repeat the process until no more feasible merging is possible. We obtain two heuristic solutions by attempting two different strategies for the selection of the pair of compatible items to be merged: (i) select the pair (j, k) for which $|f_j - f_k|$ is a minimum; (ii) select the pair (j, k) for which $(\min\{f_j, f_k\} - w_j - w_k)$ is a minimum. We denote U_{mg} the resulting upper bound value.

4.2. A Variable Neighborhood Search

Modern metaheuristic and matheuristic methods build upon the simple concept of local search, where a solution σ is improved by iteratively moving to a solution σ' selected among those which are "close" to σ. This concept of vicinity is formalized by means of a *neighborhood* $N(\sigma)$, which is a set containing solutions that can be obtained from σ with a simple modification. During the years the "closeness" requirement has been relaxed, and large neighborhoods have been used, allowing the starting solution to be modified at a large extent. These important changes in the solution have been adopted in several frameworks. They are called *destructive methods* in the Strategic Oscillation by Glover and Laguna [14] and *ruin phase* in the Ruin and Recreate concept by Schrimpf et al. [25]. They are also present in Markov chains optimization, see Martin et al. [22], *Iterated Descent* by Baum [4] (the precursor of *Iterated Local Search*, see, e.g., Stützle [26]) and *Variable Neighborhood Search* (VNS) by Mladenović and Hansen [23].

In particular VNS uses a set of neighborhoods $N_h(\sigma)$ whose size increases according to a parameter h. The value of h is strategically grown so that the next solutions are more and more distant from the original one. The parameter is reset to its smallest value when some quality or feasibility criteria are met. VNS algorithms have been used profitably in a huge number of combinatorial optimization problems (see, e.g., Hansen et al. [16] for a recent survey).

We adopt this scheme, using a set of *destructive* neighborhoods that may destroy the entire solution but a single bin. We start by computing a heuristic solution, say σ, using $U(\sigma)$ bins, by means of the greedy heuristics of Section 4.1. We then

(i) remove h bins from σ and (ii) reassign the corresponding items in a way that possibly violates the fragility requirements but uses $U(\sigma) - 1$ bins, thus producing a new solution, say σ', belonging to $N_h(\sigma)$. The sketch is outlined in Algorithm 2. *Perturbation* and *Local Search* steps are described in details in the following.

input : A BPPFO instance
output: A heuristic solution σ using $U(\sigma)$ bins

Starting Solution: Compute a (feasible) heuristic solution σ;
$h = 1$;
repeat
  $t = 1$;
  repeat
    Perturbation: Generate a new (possibly infeasible) solution $\sigma' \in N_h(\sigma)$ using $U(\sigma) - 1$ bins;
    Local Search: Apply improvement methods to σ' so as to obtain a (hopefully feasible) solution σ'';
    $t = t + 1$;
  until $(t \leq n_{iter}$ **and** σ'' is infeasible$)$;
  if $(\sigma''$ is feasible$)$ **then**
    $\sigma = \sigma''$;
    $h = 1$;
  else
    $h = h + 1$;
    if $(h > h_{\max})$ **then** $h = 1$;
  end
until (a stopping condition is met);
return σ;

Algorithm 2: A Variable Neighborhood Search algorithm for the BPPFO

The new solution σ' is possibly infeasible, because the sum of the weights of the items associated to one or more of the $U(\sigma) - 1$ bins may exceed the fragility of the most fragile item in the bins. We then try to minimize the total weight excess by using a set of local search algorithms (to be described below). The new solution obtained after the local search application is denoted by σ''.

If we managed to restore feasibility in all bins of σ'', then we obtained a new heuristic solution using one bin less. We thus update the incumbent solution and restart with $h = 1$. Otherwise, we reiterate the process until a maximum number of n_{iter} iterations has been elapsed. If σ'' remains infeasible after n_{iter} iterations, then we increase h by one unit, so as to perform a search in a larger solution space. Whenever h exceeds a given limit $h_{\max}$, we set again $h = 1$. On the basis of computational outcome, we set n_{iter} to 3 and $h_{\max}$ to the number of bins in the incumbent solution minus one.

The algorithm is halted whenever it finds an upper bound equal to the lower bound or after 4 CPU minutes have been elapsed. We found this time limit to be the best compromise between solution quality and speed after running the VNS for 15 min on each instance. The idea of moving in a search space that contains infeasible solutions while trying to restore feasibility has been used in a number of successful heuristic implementations (see, e.g., Talbi [27]). This proved to be profitable also for the BPPFO, because it allows to quickly move from a solution to another by temporarily disregarding the fragility requirements (which may be particularly strict), hence facilitating local search.

The perturbation method

As briefly discussed above, the aim of our perturbation method is to modify the incumbent solution σ so as to generate a new solution σ' belonging to the neighborhood $N_h(\sigma)$. This neighborhood is the set of feasible and infeasible solutions that can be obtained from σ by means of (i) the removal of h bins and (ii) the assignment of the items originally packed in the removed bins to $U(\sigma) - 1$ bins including the bins of the unremoved items. In step (ii) the fragility constraints are disregarded to give emphasis to solutions using no more than $U(\sigma) - 1$ bins.

In particular σ' is constructed as follows: the $U(\sigma) - h$ bins that remain in σ after the removal of the h bins are copied directly into σ'; $h - 1$ new empty bins are opened in σ'; the $U(\sigma) - 1$ bins obtained in this way are filled with the removed items by eventually accepting violations of the fragility requirements.

We attempt different strategies to perform steps (i) and (ii) above, so as to obtain efficient diversification methods. We recall $J(i)$ denotes the set of items assigned to a bin i. The first step is obtained by removing h bins:

(i.1) with probability proportional to $\min_{j \in J(i)}\{f_j\}$;
(i.2) with probability proportional to $\left(\min_{j \in J(i)}\{f_j\} - \sum_{j \in J(i)} w_j\right)$;
(i.3) in a random way.

On the basis of computational outcome we use a two-level objective function to redistribute the removed items. Suppose we need to evaluate the packing of an item j to a bin i. Our objective function, denoted $\phi(j, i)$, computes at the first level the number of items in conflict with j in bin i, i.e., $\left|\left\{k \in J(i) : w_j + w_k > \min\{f_j, f_k\}\right\}\right|$. Ties are broken at the second level,

where $\phi(j, i)$ computes the eventual weight excess in the bin after packing j. Packing j in i is feasible only when both levels of $\phi(j, i)$ take value zero. The different policies we adopted to perform the second step of our perturbation method are:

(ii.1) *best-best*: at each iteration consider all assignments of an item j to a bin i, and choose the one minimizing $\phi(j, i)$;
(ii.2) *first-best*: at each iteration choose the item j having minimum fragility among the removed items, and pack it in the bin i minimizing $\phi(j, i)$;
(ii.3) *first-first*: as in *first-best* but scan the bins according to non-increasing values of fragility and stop as soon as a feasible assignment for the item j is found, if any, otherwise choose the bin i minimizing $\phi(j, i)$;
(ii.4) *random*: assign each item to a random bin.

After evaluating the computational impact of the combinations of these different policies, we produced a VNS configuration that works as follows. Recall the VNS performs $n_{iter} = 3$ iterations with the same value of h. During the first iteration it uses policy ((i.1), (ii.1)) to perturb the incumbent, during the second iteration ((i.1), (ii.3)) and during the third ((i.3), (ii.3)). Policies (i.2), (ii.2) and (ii.4) are not used in the final VNS configuration because of poor computational performance.

Improvement through local search

The solution obtained after the execution of the perturbation method is usually infeasible, because there might be some weight excess in one or more bins. We try to restore feasibility by means of a local search procedure that *swaps* sets of items between pairs of bins. In particular, let $\widetilde{W}(i) = \max\left\{0, \sum_{j\in J(i)} w_j - \min_{j\in J(i)}\{f_j\}\right\}$ be the weight excess associated to bin i. Let also i_1 and i_2 be two bins, such that at least one of the two has some weight excess. Our procedure, denoted *swap* (ℓ_1, ℓ_2), attempts to exchange ℓ_1 items packed in i_1 with ℓ_2 items packed in i_2. If this provides a reduction in the maximum weight excess between the two bins (i.e., if this reduces $\max\{\widetilde{W}(i_1), \widetilde{W}(i_2)\}$), then we say that the move is *improving*.

During any swap (ℓ_1, ℓ_2) execution we keep the bins sorted by non-decreasing values of $\widetilde{W}(i)$ and attempt first to swap items between the first and the second bin, then between the first and the third and so on. Quick checks on the weights of the items and the weight excesses of the bins are performed so as to attempt only those swaps that can be improving. Note that any time a swap is attempted we may need to re-evaluate the fragility of the involved bins, but this can be done quickly, by keeping the list of items packed in a bin sorted according to non-increasing fragility. We operate in a *first improvement* policy, i.e., as soon as an improving move is found we perform it and reiterate.

We implemented swaps (1, 0), (1, 1), (1, 2), (2, 1) and (2, 2). After preliminary computational evaluations we adopted a strategy that attempts only the swaps (1, 0), (1, 1), (1, 2) and (2, 1), and performs them in non-decreasing order of their complexity, starting from (1, 0). The local search phase is halted when all the weight excess has been removed from the bins or when no improving move is found for any type of swap.

5. A column generation algorithm

We present a model that builds upon the classical decomposition method by Gilmore and Gomory [12,13]. We define a *pattern* as a feasible combination of items. We describe the pattern, say p, by a column $(a_{1p}, \ldots, a_{jp}, \ldots, a_{np})^T$, where a_{jp} takes value 1 if item j is in pattern p, 0 otherwise. Let P be the set of all valid patterns, i.e., the set of patterns p for which

$$\sum_{j=1}^{n} w_j a_{jp} \leq \min_{j=1,\ldots,n}\{f_j a_{jp}\}. \tag{13}$$

Let also z_p be a binary variable taking value 1 if pattern p is used, 0 otherwise ($p \in P$). The BPPFO can be modeled as the following Set Covering problem:

$$\min \sum_{p\in P} z_p \tag{14}$$

$$\sum_{p\in P} a_{jp} z_p \geq 1 \quad j = 1, \ldots, n \tag{15}$$

$$z_p \in \{0, 1\} \quad \forall p \in P. \tag{16}$$

Constraints (15) impose that each item j is packed in at least one bin. Note that we can use "$\geq$" instead of "$=$", because if an item is packed in more that one bin then we can remove such item from all bins but one. Indeed this removal would decrease the left hand side of (13) but would not decrease the corresponding right hand side.

As the number of possible patterns may be very large, even solving the linear relaxation of Model (14)–(16) (defined *master problem* in the following) may be difficult. We approach this problem by means of a column generation method. We initialize the model with a subset $\widetilde{P} \subseteq P$ of patterns, and obtain the so-called *restricted master problem*. We replace the integrality requirements (16) with $z_p \geq 0, \ \forall p \in P$. Note that we are allowed to drop the conditions $z_p \leq 1$ since redundant (indeed we can always replace a solution in which there exists a $z_p > 1$ with a better one having $z_p = 1$). We finally associate dual variables π_j ($j = 1, \ldots, n$) to Constraints (15).

We operate in an iterative way. We solve the linear model just outlined and check if a pattern (i.e., a column) with negative reduced cost exists. If it exists, then we add it to the restricted master problem and reiterate, otherwise we proved the optimality of the (eventually fractional) solution obtained. The reduced cost of a pattern p is defined by

$$\bar{c}_p = 1 - \sum_{j=1}^{n} \pi_j a_{jp}.$$

A pattern is added to the model if it satisfies the fragility requirement (13) and has a negative reduced cost. The existence of such pattern can thus be determined by solving a KP01FO (see Section 2.1) with objective function

$$\max \sum_{j=1}^{n} \pi_j a_{jp}.$$

In Section 5.1 we present the ILP we implemented to solve the KP01FO just described. In Section 5.2 we propose a way to improve the column generation behavior by means of dual cuts. Note that in case the column generation terminates with a fractional solution, we do not enter a branching scheme but content us with a valid lower bound. Branching may affect both the solution of the pricing subproblem and the improvement with dual cuts, nevertheless is an interesting future research direction.

5.1. A solution of the pricing subproblem

We solve the KP01FO problem arising in the pricing phase by means of an ILP model that is derived from Model (2)–(7) of Section 2.1. Let us recall that items are sorted by non-decreasing values of f_j, breaking ties by non-increasing values of w_j. We define β_j as a binary variable taking value 1 *if item j is the item with smallest fragility in the bin*, 0 otherwise ($j = 1, \ldots, n$). We also define α_j as a binary variable taking value 1 *if item j is packed in the bin but is not the item with smallest fragility*, 0 otherwise ($j = 1, \ldots, n$). We obtain:

$$\max \sum_{j=1}^{n} \pi_j(\alpha_j + \beta_j) \tag{17}$$

$$\sum_{j=1}^{n} \beta_j = 1 \tag{18}$$

$$\sum_{j=1}^{n} \beta_j(f_j - w_j) - \sum_{j=1}^{n} w_j \alpha_j \geq 0 \tag{19}$$

$$\alpha_k + \sum_{j=k}^{n} \beta_j \leq 1 \quad k = 1, \ldots, n \tag{20}$$

$$\alpha_j \in \{0, 1\} \quad j = 1, \ldots, n \tag{21}$$

$$\beta_j \in \{0, 1\} \quad j = 1, \ldots, n. \tag{22}$$

Constraints (18) impose the existence of just one item with smallest fragility inside the bin. Constraints (19) impose that the sum of the weights of the items in the bin does not exceed the fragility of the smallest item in the bin. Constraints (20) impose that if an item k is packed in the bin but is not the item with smallest fragility (i.e., if $\alpha_k = 1$), then no other item having larger fragility can be the item with smallest fragility in the bin. Note that the latter constraints also guarantee that $\alpha_j + \beta_j \leq 1$ for $j = 1, \ldots, n$. Note also that it is enough to replace the objective function (17) with the minimization of the left hand side of Constraint (19) to apply Proposition 1 of Section 2.2.

5.2. Stabilizing column generation with dual cuts

One of the main issues in column generation is a long tail convergence, during which the value of the optimum is only marginally improved. Several methods have been proposed in the literature to deal with this issue. One of the most recent and promising ones is based on the notion of *dual cuts* and was introduced by Valério de Carvalho [29]. The idea is to restrict the dual space of the Set Covering problem by adding dual cuts (primal columns) to the restricted master problem, to exclude dual solutions that are dominated by others while still preserving the optimal primal solution.

Let us describe briefly the original idea and then generalize it to the BPPFO. Although the idea has been proposed for the cutting stock problem, for sake of clarity we summarize it here as if it were designed for the BPP. Valério de Carvalho [29] defines a dual cut for the Set Covering formulation of the BPP as

$$\sum_{\ell \in S} \pi_\ell \leq \pi_k$$

for any item k and any subset of items S such that $w_k \geq \sum_{\ell \in S} w_\ell$. This dual cut is added to the reduced master problem as a primal column, say q, having cost 0 and entries

$$a_{jq} = \begin{cases} 1 & j \in S, \\ -1 & j = k, \\ 0 & \text{otherwise.} \end{cases} \tag{23}$$

This column may be useful for the solution of the Set Covering formulation in the following way. Suppose the current reduced master problem includes a pattern p that contains item k but does not contain set S. Suppose also that at the next iteration the pattern to be added to the restricted master problem is $p' = p \cup S \setminus \{k\}$. To detect p', the classical column generation approach would be forced to call an algorithm for the solution of the slave problem (i.e., a KP01 when solving a BPP). The approach improved by dual cuts can use instead the original column corresponding to p in conjunction with the new column q built as in (23), hence resulting in p'. In practice the new pattern is built starting from the old one, using the new column, whose cost is 0, to remove k and add S. It is important to note that the new patterns obtained using the dual cut are feasible because the cut satisfies the condition $w_k \geq \sum_{\ell \in S} w_\ell$.

The rationale behind this idea is that by adding dual cuts we can save calls to the algorithm for the solution of the slave problem. Since solving the slave problem may be difficult and time consuming (as is the case for both the BPP and the BPPFO), the overall time to convergence may decrease.

The notion of dual cuts can be extended to the BPPFO as follows.

Proposition 3. *For a given item k, if there exists a subset S of items such that $\sum_{\ell \in S} w_\ell \leq w_k$ and $\min_{\ell \in S} f_\ell \geq f_k$, then*

$$\sum_{\ell \in S} \pi_\ell \leq \pi_k \tag{24}$$

is a valid dual cut for (14)–(16).

Proof. Let p be a valid pattern containing item k but not the set S. We need to prove that by using (24) we can create a new pattern p' that is still feasible. Since p is feasible we have

$$\sum_{j \in p \setminus \{k\}} w_j + w_k \leq \min \left\{ \min_{j \in p \setminus \{k\}} \{f_j\}, f_k \right\}.$$

By replacing k with the set S in pattern p, we obtain a new pattern p' such that

$$\sum_{j \in p \setminus \{k\}} w_j + \sum_{\ell \in S} w_\ell \leq \min \left\{ \min_{j \in p \setminus \{k\}} \{f_j\}, \ \min_{\ell \in S} \{f_\ell\} \right\},$$

hence

$$\sum_{j \in p'} w_j \leq \min_{j \in p'} \{f_j\},$$

which means that p' is feasible. □

There is an exponential number of cuts (24) that can be added to the restricted master problem, hence one has to design an efficient strategy to include only the useful ones. In our implementation we use a quick heuristic strategy, by limiting our search to subsets of promising cuts. In particular, we generalize to the BPPFO the cuts of type I and II introduced in [29] for the BPP.

The cuts of *type* I are obtained by choosing sets S that consist of a single item. In this way we obtain a set of $O(n^2)$ inequalities of the type:

$$\pi_j \leq \pi_k,$$

for all $j, k = 1, \ldots, n$, such that $w_k \geq w_j$ and $f_k \leq f_j$.

The cuts of *type* II are obtained by choosing sets S that contain exactly two items. The resulting inequalities are:

$$\pi_j + \pi_\ell \leq \pi_k,$$

for all $i, \ell, k = 1, \ldots, n$, such that $w_k \geq w_j + w_\ell$ and $f_k \leq \min\{f_j, f_\ell\}$. The number of cuts of type II can be as large as $O(n^3)$, but, practically speaking, it turns out to be much smaller due to the restrictive condition on fragility. The cuts are all added to the restricted master at the beginning of the column generation algorithm. The computational impact of these two types of dual cuts is evaluated in details in the next section.

6. Computational results

We implemented all algorithms in C++ and ran them on an Intel Xeon 2.40 GHz with 6 GB of memory. Since, according to our knowledge, this is the first work in which computational results are presented for the BPPFO, we devote Section 6.1 to describe a challenging benchmark set that we generated. The overall results that we obtained on such benchmark are then given in Section 6.2.

6.1. A benchmark set

It is of interest to determine in which way the fragility constraint can affect the difficulty of an instance. To this aim, we constructed a large set of BPPFO instances, starting from a classical BPP benchmark and attempting several ways to create the fragility of the items.

As BPP benchmark we used the classical *set* 1 by Scholl et al. [24], which has been used in almost all recent papers presenting heuristic or exact computational results for the BPP. This set can be downloaded from the ESICUP website http://www.fe.up.pt/~esicup and consists of instances with number of items $n \in \{50, 100, 200, 500\}$, bin capacity $C \in \{100, 120, 150\}$ and item weights uniformly generated in the intervals [1, 100], [20, 100] and [30, 100]. For each configuration of n, C and item weights, 20 instances were generated, giving in total 720 instances. We reduced our computational tests to the first five instances out of the original 20 of each configuration, and we disregarded the case with $n = 500$, because, as it will be seen clearly in the following, the smaller instances already proved to be difficult. Hence, we consider in total 135 BPP instances.

To generate the BPPFO benchmark, we take the item weights from the BPP instances and generate fragilities according to different strategies. We particularly focus on the relation that lies between the fragility of an item and its weight, because this can strongly influence the resulting difficulty of the problem. In particular we created different configurations using different correlations between fragility and weight:

Uncorrelated instances: we set $w_{\max} = \max_{j=1,\dots,n}\{w_j\}$ and generated fragilities into different sub-intervals of the interval $[w_{\max}, k_2C]$, with k_2 being a positive integer. More in details, we set f_j as an integer value selected with uniform distribution in the interval $[w_{\max} + k_1(k_2C - w_{\max})/5, k_2C]$, for $j = 1, \dots, n$. We tested the values $k_1 = 0, 1, \dots, 4$ and $k_2 = 1, 2, \dots, 5$;

Weakly correlated instances: we set f_j as an integer value selected with uniform distribution in the interval $[k_1w_j, k_2C]$, for $j = 1, \dots, n$. We tested the values $k_1 = 1, 2, \dots, 5$ and $k_2 = k_1, k_1 + 1, \dots, 5$;

Strongly correlated instances: we set $f_j = k_1w_j$, for $j = 1, \dots, n$. We tested the values $k_1 = 2, 3, \dots, 5$.

We obtained 49 different configurations in total. We attempted to solve the resulting BPPFO instances with $n = 100$ by means of the ILP Model (2)–(7), that we implemented using Cplex 12 and ran with a time limit of 5 CPU minutes. Strongly correlated instances are very easy and can all be solved by the model within the given time limit, with a single exception for an instance with $k_1 = 4$. Uncorrelated instances are instead more difficult, although those with both low values of k_1 and k_2, as well as those with both high values of k_1 and k_2, are easier than those that are in the middle. Weakly correlated instances are even more difficult, with three configurations for which not even a fifth of the instances could be solved to optimality by the ILP.

We continued our study by also attempting the solution of the above instances through the preprocessing scheme of Section 2.2, followed by Model (2)–(7). Also in this case the time limit was set to 5 CPU minutes, including both times of preprocessing and model. The use of the preprocessing improved the behavior of the compact model on the uncorrelated and weakly correlated instances that are characterized by both small values of k_1 and k_2. On the more difficult instances, however, the decrease in the fragility of the items was not large enough to motivate the time spent for its use. It is thus an interesting future research direction to try to devise more enhanced preprocessing techniques.

On the basis of the above results we selected the five most difficult classes, that seem to represent a challenging test bed:

*Class*1: uncorrelated with $k_1 = 1$ and $k_2 = 3$;
*Class*2: uncorrelated with $k_1 = 1$ and $k_2 = 5$;
*Class*3: weakly correlated with $k_1 = 1$ and $k_2 = 5$;
*Class*4: weakly correlated with $k_1 = 3$ and $k_2 = 3$;
*Class*5: weakly correlated with $k_1 = 3$ and $k_2 = 4$.

To encourage further research on the problem, the $5 \times 135 = 675$ BPPFO instances that we obtained in this way have been made publicly available in the web site http://www.or.unimore.it/resources.htm.

6.2. Overall results

Due to the large number of instances addressed in the following we mainly report aggregate results, and refer again to http://www.or.unimore.it/resources.htm for the detailed computational results on each instance.

The results of the lower bounding procedures of Section 3 are summarized in Table 1. Each line in the table gives average values over 45 instances. The first two columns give the number of items (n) and the class (cl) of the group of 45 instances. Column U_{vns} reports the average upper bound value produced by the VNS. For each lower bound, say L_x, we give the average percentage gap, computed as $\%g(L_x) = 100(U_{\text{vns}} - L_x)/U_{\text{vns}}$. We also give the CPU time in seconds (sec) elapsed by all combinatorial lower bounds and by the column generation algorithm. The column generation was run with no time limit, initializing the restricted master with n columns, each containing a single item j ($j = 1, \dots, n$) and with the dual cuts of types I and II. At each iteration the solution of Model (17)–(22) is halted as soon as a negative reduced cost column is found, if any.

Table 1
Results of the lower bounds.

n	cl	U_{vns}	Combinatorial lower bounds					Column generation	
			$\%g(L_0)$	$\%g(L_1)$	$\%g(L_{cp})$	$\%g(L_2)$	*sec*	$\%g(L_{cg})$	*sec*
50	1	13.31	36.15	6.44	37.47	2.68	0.10	1.11	6.96
	2	8.98	40.06	5.99	40.91	0.91	0.10	0.91	2.34
	3	11.69	27.52	7.54	24.34	2.52	0.10	1.82	7.45
	4	11.69	27.57	9.03	23.12	2.19	0.10	1.52	7.33
	5	10.38	37.26	9.54	29.03	0.92	0.10	0.58	3.69
100	1	26.13	36.14	4.32	51.90	2.74	0.12	1.81	113.57
	2	17.89	42.94	5.26	42.74	1.65	0.13	1.25	86.16
	3	24.24	31.53	5.85	29.62	2.42	0.11	1.84	143.59
	4	23.56	29.51	5.20	29.38	2.67	0.12	1.97	168.93
	5	20.47	37.93	7.03	35.12	3.24	0.12	2.50	119.03
200	1	50.62	34.68	3.30	41.36	2.38	0.14	1.92	559.00
	2	33.91	40.81	3.46	40.96	1.81	0.16	1.68	573.85
	3	46.67	29.65	4.59	27.37	2.84	0.12	2.16	726.22
	4	46.11	28.66	4.56	25.84	2.51	0.12	2.06	706.51
	5	40.24	38.39	4.87	35.21	2.19	0.13	1.92	500.50
Avg		25.73	34.59	5.80	34.29	2.24	0.12	1.67	248.34

Table 2
Results of the upper bounds.

n	cl	L_{cg}	Greedy upper bounds					VNS		
			$\%g(U_{xf})$	$\%g(U_{mg})$	$\%g(U_{kp})$	$\%g(U_{min})$	*sec*	$\%g(U_{vns})$	*sec*	*md*
50	1	13.18	2.09	15.64	7.79	2.09	0.01	1.11	35.14	1
	2	8.89	1.59	15.62	9.47	1.59	0.01	0.91	21.47	1
	3	11.47	2.80	17.95	8.18	2.80	0.01	1.82	53.99	1
	4	11.51	2.85	17.84	6.45	2.70	0.01	1.52	43.17	1
	5	10.31	1.55	19.04	8.16	1.55	0.01	0.58	16.33	1
100	1	25.64	2.94	17.43	5.38	2.94	0.04	1.81	123.19	1
	2	17.67	1.80	20.46	4.93	1.80	0.05	1.25	59.20	1
	3	23.80	3.55	20.51	5.92	3.48	0.04	1.84	120.21	1
	4	23.09	3.91	20.20	4.92	3.67	0.04	1.97	130.82	1
	5	19.96	3.33	20.49	6.48	3.33	0.05	2.50	125.69	1
200	1	49.60	2.90	18.33	2.80	2.48	0.27	1.92	211.04	2
	2	33.31	2.07	20.97	2.99	1.91	0.30	1.68	154.08	1
	3	45.64	3.23	21.02	3.18	2.50	0.27	2.16	213.65	2
	4	45.16	3.24	20.32	3.54	2.71	0.27	2.06	225.83	2
	5	39.44	2.21	21.12	3.84	2.17	0.28	1.92	184.22	2
Avg		25.24	2.67	19.13	5.60	2.51	0.11	1.67	114.54	1.27

The lower bound values produced by L_0 are very bad, and lead to an average gap larger than 34%. This was expected as its performance depends from the maximum fragility value in the instance. Better values are obtained by L_1, that improves L_0 by about seven bins on average per instance and decrease the average gap to 5.8%. The performance of L_{cp} is comparable to that of L_0. The performance of L_k is not reported explicitly in the table as it is much weaker than that of L_{cp}. Lower bound L_2 consistently outperforms all previous lower bounds (it dominates L_0 and L_1, and in no instance the value of L_{cp} is greater than that of L_2). The average gap obtained by L_2 is just 2.24%. The CPU time required to run these algorithms never exceeds 0.2 s. The column generation approach manages to decrease the average percentage gap to 1.67%. This is however achieved with a large increase in the computational effort: just 5 s on average when $n = 50$, but about 10 min when $n = 200$.

The results of the greedy upper bounds of Section 4.1 and the VNS of Section 4.2 are summarized in Table 2. Once again each line gives average values over 45 instances. Column L_{cg} reports the average lower bound value produced by the column generation algorithm. The percentage gap of an upper bound, say U_x, is computed as $\%g(U_x) = 100(U_x - L_{cg})/U_x$. We use $U_{min} = \min\{U_{xf}; U_{mg}; U_{cb}; U_{kp}\}$ to denote the best upper bound value found by the greedy algorithms. Columns *sec* report the CPU seconds elapsed by all the greedy upper bounds and by the VNS. We do not report explicitly the results of the clique-based heuristic (U_{cb}), because it never improved the X-Fit heuristic (U_{xf}). Column *md* gives the maximum difference, in terms of number of bins, between U_{vns} and L_{cg}.

Among the greedy procedures, the X-Fit heuristic is the one achieving the best results. The merging heuristic (U_{mg}) obtains a gap that is roughly seven times worse than that produced by U_{xf}. A better behavior is shown by the knapsack-based heuristic (U_{kp}), which is weaker than U_{xf} on average, but outperforms it in terms of solution value on 43 instances. The use of these algorithms leads to an average gap of 2.51%. The execution of the complete set of greedy procedures is very fast, as it requires about 0.1 CPU seconds on average.

Table 3
Comparison with model (2)–(7).

n	cl	Lower and upper bounds					Compact model					
		L_{cg}	U_{vns}	%g	sec	opt	L_{root}	L	U	%g	sec	opt
50	1	13.18	13.31	1.11	42.22	39	13.11	13.24	13.24	0.00	57.89	45
	2	8.89	8.98	0.91	23.93	41	8.84	8.93	8.93	0.00	4.45	45
	3	11.47	11.69	1.82	61.55	35	11.38	11.60	11.62	0.17	125.74	44
	4	11.51	11.69	1.52	50.60	37	11.42	11.62	11.64	0.19	116.73	44
	5	10.31	10.38	0.58	20.13	42	10.24	10.33	10.36	0.19	91.88	44
100	1	25.64	26.13	1.81	236.92	23	25.49	25.62	26.09	1.85	1877.25	24
	2	17.67	17.89	1.25	145.54	35	17.49	17.60	18.00	2.18	1746.11	27
	3	23.80	24.24	1.84	263.95	25	23.67	23.78	24.38	2.43	2599.56	18
	4	23.09	23.56	1.97	299.91	24	22.91	23.07	23.82	3.22	2842.85	12
	5	19.96	20.47	2.50	244.88	22	19.76	20.02	20.47	2.07	1987.25	25
200	1	49.60	50.62	1.92	770.46	9	49.24	49.29	51.02	3.44	3600.08	0
	2	33.31	33.91	1.68	728.39	18	33.07	33.18	34.56	3.93	3459.92	2
	3	45.64	46.67	2.16	940.26	6	44.71	44.82	46.89	4.42	3600.08	0
	4	45.16	46.11	2.06	932.73	4	44.38	44.44	46.44	4.33	3600.07	0
	5	39.44	40.24	1.92	685.12	11	38.98	39.11	40.98	4.53	3600.08	0
Avg		25.24	25.73	1.67	363.11	371	24.98	25.11	25.90	2.20	1954.00	330

The VNS shows a good behavior because it decreases the solution value found by the greedy heuristics 137 times, i.e., for 20% of the instances. This leads to a decrease of about 0.8 of the average gap. We can note that this improvement in the gap is consistent, as it arises for all the cases of number of items and class. As expected, the CPU effort increases, but remains under 2 CPU minutes on average. The maximum difference between the best lower and upper bound found is usually just one bin, and just when $n = 200$ it increases to 2 bins for four classes.

In Table 3 we finally compare our lower and upper bounds with Model (2)–(7) of Section 2.1, solved with Cplex 12 (running in sequential mode on a single processor). The total computing time of our algorithms never exceeded 3200 s, so we decided to give Cplex one CPU hour to have a fair comparison. In the table, columns *%g*, *sec* and *opt* report, respectively, the percentage gap, the CPU time elapsed and the total number of proven optimal solutions for both algorithms. For the second algorithm, we also give in column L_{root} the rounding up of the value of the continuous relaxation of Model (2)–(7), in column L the rounding up of the final lower bound produced by Cplex in the given time limit, and in column U the final upper bound by Cplex.

The results show that the compact model is effective in solving instances with 50 items: just 3 solutions out of 135 are not proven to be optimal, the %gap is very small and the average CPU time is small. We can conclude that for these small-size instances it is a good choice to have some kind of enumeration scheme. When the number of items increases, however, our lower and upper bounds are more effective in tackling the difficulty of the BPPFO. For instances with 100 items, the lower and upper bounds manage to prove the optimality of 129 solutions (out of 225) against 106 by the model, in an average time that is roughly ten times smaller. As a consequence, the percentage gap is only 1.88%, against the 2.35% by the model.

For instances with 200 items this behavior becomes even more evident. Just two solutions are proven to be optimal by the model, against the 48 by our algorithms. The average gap of the model raises to more than 4%, while the one of our algorithms remains under 2%. Looking at the last line of the table, we can notice that our algorithms provide better values for lower bound, upper bound, gap, CPU time and number of proven optimal solutions.

In Table 4 we finally compare different versions of our column generation algorithm, to evaluate the impact of the dual cuts. In the left part of the table we present the results obtained without dual cuts, in the middle part the ones obtained by including dual cuts of type I and then, in the right part, the ones obtained by using also dual cuts of type II (i.e., the configuration we finally adopted). For each configuration, we present the number of columns generated, the number of dual cuts generated (except for the first configuration) and the CPU time in seconds.

We can note that the use of dual cuts of type I is effective and reduces the number of columns to be generated from 747 to 513 on average. The number of cuts to be added is quite high, about 2800 on average. The CPU time is decreased by about 50 s. The use of dual cuts of type II is less effective than that of type I, but nevertheless useful. By generating about 19 000 cuts we can save further 73 columns, on average, and reduce the CPU time by 3 further seconds.

7. Conclusions

We addressed a particularly difficult combinatorial optimization problem arising in the telecommunication field, and solved it by means of a series of non-trivial lower and upper bounds, including a Variable Neighborhood Search and a column generation algorithm. Since this is the first work in which computational results are proposed for this problem, we devoted a large computational effort to the development of difficult benchmark instances. These instances are now available online at http://www.or.unimore.it/resources.htm.

The results we obtained show the effectiveness of the algorithms we developed, especially on large-size instances, for both what concerns lower and upper bounds. In particular, the so called fractional lower bound shows a good compromise

Table 4
Impact of the dual cuts on the behavior of the column generation algorithm.

n	cl	No dual cuts		Dual cuts of type I			Dual cuts of types I and II		
		Columns	*sec*	Columns	Cuts	*sec*	Columns	Cuts	*sec*
50	1	125.0	7.74	91.0	356.3	7.87	74.9	1576.4	6.96
	2	43.8	2.66	29.0	101.2	2.35	24.1	442.5	2.34
	3	103.6	8.03	74.8	210.1	7.39	65.7	410.9	7.45
	4	99.7	8.41	66.5	189.0	7.11	61.4	380.4	7.33
	5	53.6	4.15	35.6	112.8	3.55	33.8	230.2	3.69
100	1	504.3	136.86	348.7	2165.7	115.11	294.5	15 535.8	113.57
	2	325.8	109.71	208.2	1079.9	85.11	179.9	6245.6	86.16
	3	518.2	177.28	336.8	1362.3	142.45	317.0	3302.6	143.59
	4	531.8	203.88	332.6	1185.0	171.18	306.8	2307.8	168.93
	5	553.1	150.34	374.0	1601.9	122.35	329.3	6607.0	119.03
200	1	1615.3	607.81	1148.3	8635.6	520.98	940.9	97 352.0	559.00
	2	1641.1	673.91	1158.6	6966.7	564.70	957.3	67 350.8	573.85
	3	1725.7	884.35	1171.2	5855.1	755.12	1021.0	25 260.9	726.22
	4	1766.6	850.66	1211.5	6337.8	720.73	1050.2	32 281.7	706.51
	5	1598.0	666.08	1106.2	5923.7	545.39	936.9	30 340.1	500.50
Avg		747.0	299.46	512.9	2805.5	251.43	439.6	19 308.3	248.34

between CPU time and solution quality. The column generation algorithm improves the results of the fractional bound by using a much larger CPU time. This time can however be reduced by considering dual information computed heuristically. In this case, as in many others seen in the literature, the use of heuristic techniques improves the robustness of exact techniques and reduces the computational effort.

For what concerns future research, we believe it is of interest to study enumeration techniques that make good use of the lower and upper bounding techniques we proposed. We intend to focus our attention to the development of branch-and-bound and branch-and-price exact techniques.

References

[1] K.I. Aardal, S.P.M. van Hoesel, A.M.C.A. Koster, C. Mannino, A. Sassano, Models and solution techniques for frequency assignment problems, Annals of Operations Research 153 (2007) 79–129.
[2] N. Bansal, Z. Liu, A. Sankar, Bin-packing with fragile objects, in: R.A. Baeza-Yates, U. Montanari, N. Santoro (Eds.), IFIP TCS, in: IFIP Conference Proceedings, vol. 223, Kluwer Academic Publishers, 2002, pp. 38–46.
[3] N. Bansal, Z. Liu, A. Sankar, Bin-packing with fragile objects and frequency allocation in cellular networks, Wireless Networks 15 (2009) 821–830.
[4] E.B. Baum, Iterated descent: a better algorithm for local search in combinatorial optimization problems, Technical Report, Caltech, Pasadena, CA, 1986.
[5] M. Boschetti, A. Mingozzi, The two-dimensional finite bin packing problem. Part I: new lower bounds for the oriented case, 4OR 1 (2003) 27–42.
[6] A. Ceselli, G. Righini, An optimisation algorithm for the orderded open-end bin-packing problem, Operations Research 56 (2008) 423–436.
[7] W.T. Chan, F.Y.-L. Chin, D. Ye, G. Zhang, Y. Zhang, Online bin packing of fragile objects with application in cellular networks, Journal of Combinatorial Optimization 14 (2007) 427–435.
[8] F. Clautiaux, C. Alves, J. Valério de Carvalho, A survey of dual-feasible and superadditive functions, Annals of Operations Research 179 (2010) 317–342.
[9] E.G. Coffman, G. Galambos, S. Martello, D. Vigo, Bin packing approximation algorithms: combinatorial analysis, in: D.-Z. Du, P.M. Pardalos (Eds.), Handbook of Combinatorial Optimization, Kluwer Academic Publishers, 1999, pp. 151–208.
[10] A.E. Fernandes Muritiba, M. Iori, E. Malaguti, P. Toth, Algorithms for the bin packing problem with conflicts, INFORMS Journal on Computing 22 (2010) 401–415.
[11] M. Gendreau, G. Laporte, F. Semet, Heuristics and lower bounds for the bin packing problem with conflicts, Computers & Operations Research 31 (2004) 347–358.
[12] P.C. Gilmore, R.E. Gomory, A linear programming approach to the cutting stock problem, Operations Research 9 (1961) 849–859.
[13] P.C. Gilmore, R.E. Gomory, A linear programming approach to the cutting stock problem—part II, Operations Research 11 (1963) 863–888.
[14] F. Glover, M. Laguna, Tabu Search, Kluwer, Boston, 1997.
[15] P. Hansen, V. Maniezzo, S. Voss, Special issue on mathematical contributions to metaheuristics editorial, Journal of Heuristics 15 (2009) 197–199.
[16] P. Hansen, N. Mladenović, J.A. Moreno Pérez, Variable neighbourhood search: methods and applications, Annals of Operations Research 175 (2010) 367–407.
[17] D.S. Johnson, Near-optimal bin packing algorithms, Ph.D. Thesis, MIT, Cambridge, MA, 1973.
[18] D.S. Johnson, Approximation algorithms for combinatorial problems, Journal of Computer and System Sciences 9 (1974) 256–278.
[19] A. Khanafer, F. Clautiaux, E.-G. Talbi, New lower bounds for bin packing problems with conflicts, European Journal of Operational Research 206 (2010) 281–288.
[20] V. Maniezzo, T. Stutzle, S. Voss, Matheuristics: Hybridizing Metaheuristics and Mathematical Programming, in: Annals of Information Systems, vol. 10, Springer, New York, 2009.
[21] S. Martello, D. Pisinger, P. Toth, Dynamic programming and strong bounds for the 0–1 knapsack problem, Management Science 45 (1999) 414–424.
[22] O. Martin, S.W. Otto, E.W. Felten, Large-step Markov chains for the traveling salesman problem, Complex Systems 5 (1991) 299–326.
[23] N. Mladenović, P. Hansen, Variable neighborhood search, Computers & Operations Research 24 (1997) 1097–1100.
[24] A. Scholl, R. Klein, C. Jürgens, BISON: a fast hybrid procedure for exactly solving the one-dimensional bin packing problem, Computers & Operations Research 24 (7) (1997) 627–645.
[25] G. Schrimpf, K. Schneider, H. Stamm-Wilbrandt, V. Dueck, Record breaking optimization results using the ruin and recreate principle, Journal of Computational Physics 159 (2000) 139–171.
[26] T. Stützle, Applying iterated local search to the permutation flow shop problem, Technical Report aida-98-04, FG Intellektik, TU Darmstadt, Germany, 1998.
[27] E.-G. Talbi, Metaheuristics From Design to Implementation, John Wiley & Sons, Hoboken, New Jersey, 2009.
[28] J. Valério de Carvalho, LP models for bin packing and cutting stock problems, European Journal of Operational Research 141 (2002) 253–273.
[29] J. Valério de Carvalho, Using extra dual cuts to accelerate column generation, INFORMS Journal on Computing 17 (2) (2005) 175–182.