European Journal of Operational Research 206 (2010) 281–288

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

New lower bounds for bin packing problems with conflicts

Ali Khanafer, François Clautiaux *, El-Ghazali Talbi

Université des Sciences et Technologies de Lille, LIFL CNRS UMR 8022, INRIA Lille - Nord Europe Bâtiment INRIA, Parc de la Haute Borne, 59655 Villeneuve d'Ascq, France

ARTICLE INFO

Article history:
Received 25 July 2008
Accepted 21 January 2010

Keywords:
Bin packing with conflicts
Knapsack problem
Lower bounds
Chordal graphs
Tree-decomposition
Dual-feasible functions

ABSTRACT

The *bin packing problem with conflicts* (BPC) consists of minimizing the number of bins used to pack a set of items, where some items cannot be packed together in the same bin due to compatibility restrictions. The concepts of *dual-feasible functions* (DFF) and *data-dependent dual-feasible functions* (DDFF) have been used in the literature to improve the resolution of several cutting and packing problems. In this paper, we propose a general framework for deriving new DDFF as well as a new concept of *generalized data-dependent dual-feasible functions* (GDDFF), a conflict generalization of DDFF. The GDDFF take into account the structure of the conflict graph using the techniques of graph triangulation and tree-decomposition. Then we show how these techniques can be used in order to improve the existing lower bounds.

© 2010 Published by Elsevier B.V.

1. Introduction

In this paper we consider the *bin packing problem, with conflicts* (BPC). The one-dimensional variant of this problem (BPC-1D) consists of an unlimited number of bins of size C and a set $I = \{1, 2, \ldots, n\}$ of n items, each item i of size $c_i \leqslant C$. We are also given an undirect graph of conflicts $G = (V, E)$, where each vertex v_i of V is related to an item i of I and $(v_i, v_j) \in E$ if items i and j are in conflict. In the rectangular two-dimensional variant (BPC-2D), each item i is a rectangle of width w_i and height h_i and the bin is a rectangle of width W and height H. An extensive survey on cutting and packing as well as a useful classification of these problems are given in [1–4].

The aim of BPC is to minimize the number of bins used to pack all items of I, provided that the contents of any bin consist of compatible items fitting within the bins capacity without overlapping.

The BPC is clearly a generalization of the following well-known $\mathcal{NP}$-Hard problems: the *unconstrained bin packing problem* (BP) when $E = \emptyset$ and the *vertex coloring problem* (VCP) when all the items are small enough to be packed in the same bin. The BPC arises in many areas such as school course time tables construction, database replicas storage, scheduling communication systems, load balancing, the assignment of processes or tasks to processors, etc.

The BPC-1D was introduced in 1997 by Jansen and Öhring [6] who proposed and analyzed several approximation algorithms. In 2004, Gendreau et al. [7] presented new lower and upper bounds for the BPC-1D, and introduced benchmark instances. The most recent work was provided by Fernandes-Muritiba et al. [8], who presented new lower bounds, upper bounds and an exact approach based on a set covering formulation solved through a branch-and-price algorithm. The lower bound computed through this exact method is of excellent quality, but the method may entail a large computing time, and the generalization to higher dimensions is not direct. For these reasons, we focus on polynomial or pseudo-polynomial techniques to derive lower bounds for the BPC in one or more dimensions.

We obtain our new lower bounds using generalizations of so-called *dual-feasible functions* (DFF). The idea of using DFF to compute lower bounds for BP and BPC was previously tried in [8–10]. For the BPC, the results obtained using these functions were not satisfactory. This is not surprising since these functions were designed for the unconstrained bin packing. Our approach consists in applying a preprocessing phase before using any DFF. In addition, we propose and take good use of a new concept that we call *generalized data-dependent dual-feasible functions* (GDDFF). This concept is a generalization of so-called *data-dependent dual-feasible functions* (DDFF), proposed by Carlier et al. [11] as an extension of DFF and defined dependently of the instance. Using GDDFF, we are able to take into account any type of conflicts that can occur between items. Computational experiments on data sets from the literature show that our techniques can bring some improvements for the one-dimensional case. For the two-dimen-

* Corresponding author.
E-mail addresses: Ali.Khanafer@inria.fr (A. Khanafer), Francois.Clautiaux@univ-lillel.fr (F. Clautiaux), El-ghazali.Talbi@lifl.fr (E.-G. Talbi).

0377-2217/$ - see front matter © 2010 Published by Elsevier B.V.
doi:10.1016/j.ejor.2010.01.037

sional case, our techniques improve the results for a large part of the benchmark.

2. Preliminaries and the literature review

A first way of obtaining lower bounds for the BPC is to relax it into BP [7,8]. This relaxation may be effective when the conflict graph is not dense. Otherwise, dedicated methods have to be used [7,8].

Next, we describe some state-of-art techniques and methods that can be used to compute lower bounds for the BP and BPC, focusing on the concept of DFF and DDFF.

The simplest bound for the BP and BPC is the continuous lower bound LB_0. The idea is to calculate the rounded up value of the total size of the items divided by the size of the bin.

2.1. Dual-feasible functions (DFF)

The lower bound LB_0 does not take into account the fact that many items cannot be packed together in a bin. One way of improving LB_0 is to modify the size of the items so that the new instance is feasible if the initial one is. This can be done using DFF introduced by Johnson [5].

Definition 1 [5]. f: [0,1] → [0,1] is a dual-feasible function (DFF) if for any finite set S of real numbers, we have $\sum_{x\in S} x \leqslant 1 \Rightarrow \sum_{x\in S} f(x) \leqslant 1$.

The idea is to apply these functions to the sizes of the items, and then to compute LB_0 on the new instance. Several papers describe lower bounds for the bin packing problem which use DFF (e.g. [9]). Fekete and Schepers [10] have shown that DFF can be applied to both dimensions of a BP-2D instance to obtain valid lower bounds.

Now, let us describe two DFF f_0 and f_1 that can be used to compute lower bounds for the BP and BPC (see [12] for a complete survey on dual-feasible functions for bin packing problems):

- The following family of functions f_0^k has been used for the first time by Fekete and Schepers [10]. A function f_0^k is defined from [0, C] to [0, C] for a given integer $1 \leqslant k \leqslant \frac{1}{2}C$ as follows: $f_0^k(x) = C$ if $x > C - k, f_0^k(x)$ if $C - k \geqslant x \geqslant k$ and $f_0^k(x) = 0$ if $x < k$.
- Function f_1^k [11] is based on a special rounding technique. It is an improvement on a function which has been indirectly used by [16] and defined from [0,C] to $\left[0, 2\left\lfloor \frac{C}{k} \right\rfloor\right]$ for a given integer $1 \leqslant k \leqslant \frac{1}{2}C$ as follows: $f_1^k(x) = 2\left(\left\lfloor \frac{C}{k} \right\rfloor - \left\lfloor \frac{C-x}{k} \right\rfloor\right)$ if $x > \frac{1}{2}C, f_1^k(x) = \left\lfloor \frac{C}{k} \right\rfloor$ if $x = \frac{1}{2}C$ and $f_1^k(x) = 2\left\lfloor \frac{x}{k} \right\rfloor$ if $x < \frac{1}{2}C$.

2.2. Data-dependent dual-feasible functions (DDFF)

DFF are defined independently of the instance. Carlier et al. [11] defined another class of functions which depends on the sizes of items in the instance.

Definition 2 [11]. Let $I = \{1, \ldots, n\}, c_1, c_2, \ldots, c_n n$ integer values and C an integer such that $c_i \leqslant C$ for $i = l, \ldots, n$. A data-dependent dual-feasible function (DDFF) g associated with C and $c_1, c_2, \ldots, c_n$ is a discrete mapping from [0, C] to $[0, C']$ such that $\forall I_1 \subseteq I, \sum_{i\in I_1} c_i \leqslant C \Rightarrow \sum_{i\in I_1} g(c_i) \leqslant g(C)$.

The following DDFF g_0 was proposed in Carlier et al. [11] for a given parameter $k, 1 \leqslant k \leqslant \frac{1}{2}C$ and a list of integer values c_1, $c_2, \ldots, c_n (I = 1, \ldots, n)$. The authors introduced the set $J = \{i \in I : k \leqslant c_i \leqslant \frac{1}{2}C\}$ and $M_C(X,J)$ the optimal value for the one-dimensional knapsack problem related to J and size X. This value is equal to the maximum number of items i which can be packed together in a container of size X. It can be solved in linear time if the items are sorted by increasing order of size. The formal definition of $g_0 : [0, C] \to [0, M_c(C,J)]$ follows:

$$c_i \mapsto \begin{cases} M_C(C,J) - M_C(C - c_i, J) & \text{if } c_i > \frac{1}{2}C \\ 1 & \text{if } k \leqslant c_i \leqslant \frac{1}{2}C \\ 0 & \text{otherwise} \end{cases}$$

2.3. Lower bounds from the literature

The literature on lower bounds for the BPC problem was initiated by Gendreau et al. [7]. They proposed a lower bound, denoted as constrained packing lower bound *(LBcp* in the following), which starts by computing a large clique Q on G using Johnson's heuristic [14], and then assigns each element of Q to a different bin. The items in $I \setminus Q$ are partially or entirely assigned to the $|Q|$ bins using a transportation problem, and a lower bound s_Q on the total weight of the items of $I \setminus Q$ that will not be packed in the $|Q|$ bins is evaluated. Formally, LB_{CP} is computed as follows $LB_{CP} = |Q| + s_Q$.

More recently, the literature has been extended by Fernandes-Muritiba et al. [8], who proposed several lower bounds. According to the remarks of the authors, two of them were of practical utility.

The first bound, denoted as LB_{CP}^{imp}, is an improvement on the bound LB_{CP}. Once Q is computed on G, a new graph G' may be obtained by adding to E all edges (i,j) such that $c_i + c_j > C$. Then Q' may be enlarged by selecting vertices in $I \setminus Q$ according to a non-increasing degree order. The clique Q' is a maximal clique of G' and contains Q.

For the second bound, the authors modeled the BPC as a *set covering problem* (SC), which have been proved to produce interesting results for both the BP and the VCP (see [8]). By relaxing the integrality constraints, they obtain a linear programming (LP) relaxation of the SC model. They denote by LB_{SC} the lower bound computed as $LB_{SC} = \lceil z^* \rceil$ where z^* is the value of the optimal solution of the LP relaxation.

Computational results reported in Fernandes-Muritiba et al. [8] show that LB_{SC} outperforms the other bounds, followed by LB_{CP}^{imp}, which is not surprising since LB_{SC} makes use of more advanced and time consuming techniques that those of LB_{CP} and LB_{CP}^{imp}.

3. New data-dependent dual-feasible functions

In this section, we propose a general framework for deriving new DDFF using the *knapsack problem* (KP).

Let $I = \{1, \ldots, n\}$ be a set of indices, C an integer value, $c_1, c_2, \ldots, c_n$ a list of integer values less than or equal to C, and J a subset of I. The following family of functions uses an arbitrary set of parameters $\alpha = \{\alpha_i \in \mathbb{N} : i \in I\}$. We denote by $KP(C,J,\alpha)$, the classical one-dimensional knapsack problem formulated as follows:

$$KP(C,J,\alpha) = \max\left\{ \sum_{j\in J} \alpha_j x_j / \sum_{j\in J} c_j x_j \leqslant C, x_j \in \{0,1\} \right\} \quad (1)$$

where J is the set of items, each item $j \in J$ has a weight c_j and a cost α_j, and the size of the knapsack is equal to C (bin size).

Proposition 1. *The following function g_1 is a DDFF defined for a given instance D. The formal definition of $g_1 : [0, C] \to [0, KP(C,J,\alpha)]$ follows*

$$c_i \mapsto \begin{cases} KP(C,J,\alpha) - KP(C - c_i, J, \alpha) & \text{if } c_i > \frac{1}{2}C \\ \alpha_i & \text{if } 1 \leqslant c_i \leqslant \frac{1}{2}C \end{cases}$$

When applying a DFF on D, removing an item may decrease the value of the lower bound obtained. When DDFF are used, this observation does not hold anymore since the value of other items may be increased using the knapsack problem.

The basic idea of applying g_1 on D is to modify the size of the items. Thus, the small items $i(c_i \leqslant \frac{1}{2}C)$ will have a new size equal to α_i, and the remaining items will be evaluated using the knapsack problem. Choosing the value of α_i for a given item i does not at all depend on its initial value. We have tested g_1 for several values of α_i, e.g. $c_i, \lfloor \frac{c_i}{k} \rfloor$ for a given $k(1 \leqslant k \leqslant C/2)$, random value, etc. Whatever the value we give to α_i, the function g_1 remains a valid DDFF, i.e. g_1 verifies Definition 2. In fact, the validity is assured by the knapsack problem when adjusting the size of large items $i(c_i > \frac{1}{2}C)$.

The knapsack problems involved are $\mathcal{NP}$-Hard in the general case. However they can be solved in pseudo-polynomial time using dynamic programming. When the size of the bin is large, it may entail a large computing time. In this case, the set of parameters a should be chosen in a way to re-enable the resolution of the knapsack problem in a polynomial time. This idea is investigated by Carlier et al. [11] by choosing $\alpha_i = 1, \forall i \in J$. The optimal value of the knapsack problem is then equal to the maximum number of items that can be put together in a knapsack of size C. It can be solved in linear time if the items are sorted by increasing order of size.

However, the general form g_1 is more efficient for computing lower bounds than g_0. The best results were obtained by using $\alpha_i = c_i, \forall i \in J$.

4. Generalized data-dependent dual-feasible functions

In this section, we present a generalization of the framework proposed in the previous section and used for deriving new DDFF. This generalization allows any new DDFF to take into account the conflicts that may occur between items. We will denote by *generalized data-dependent dual-feasible functions* (GDDFF) this new family of functions. This new concept may be applied to any conflict-based (not necessarily pairwise) variant of bin packing problems.

4.1. Formal definition

In the following, we consider a set of integers $I = \{0, 1, \ldots, n\}$ and a function $c : i \to c_i \in \mathbb{N}, \forall i \in I$ such that $c_i \leqslant c_0, \forall i \in I \setminus \{0\}$. When a bin packing problem is considered, $I \setminus \{0\}$ will represent the set of items and the singleton $\{0\}$ is the bin. The size of an item $i \in I \setminus \{0\}$ is given by c_i and c_0 is the size of the bin instead of C.

Definition 3 (*Let δ be a set of subsets of $I \setminus \{0\}$*). A generalized data-dependent dual-feasible function (GDDFF) for (I, c, δ) is a mapping h defined from I into $\mathbb{N}$ such that $\forall S \in \delta, \forall S_1 \subseteq S, \sum_{i \in S_1} c_i \leqslant c_0 \Rightarrow \sum_{i \in S_1} h(i) \leqslant h(0)$.

Compared to DDFF, not all possible subsets of $I \setminus \{0\}$ have to be taken into account. The relation in Definition 3 may not be verified for the subsets that do not belong to δ.

4.2. A first GDDFF: h_1

The first GDDFF is a generalization of function g_1 defined in Section 3. It relies on the fact that when a conflict graph G is considered, only stable sets of G can be solution of the knapsack problem of (1). Thus, when the size of a given item i has to be computed, a knapsack problem is considered for all maximal stable sets to which vertex v_i belongs.

4.2.1. Definition

As in Section 3, the following family of functions relies on an arbitrary set of integer parameters $\alpha_1 \ldots, \alpha_n$. In this function we take into account that not all possible combinations of items are allowed.

Proposition 2. *The following function h_1 is a GDDFF for the instance (I, J, c, δ) where J is a set of pairwise incompatible items and δ a set of valid (conflict free) subsets of $I \setminus \{0\}$. Formally $h_1 : I \to [0, \max_{K \in \delta}\{KP(c_0, K, \alpha)\}]$ follows*

$$i \mapsto \begin{cases} \max\limits_{k \in \delta}\{KP(c_0, K, \alpha)\} & \text{if } i = 0 \\ \max\limits_{k \in \delta}\{KP(c_0, K, \alpha)\} - \max\limits_{K \in \delta / i \in K}\{KP(c_0 - c_i, K \setminus \{i\}, \alpha)\} & \text{if } i \in J \\ \alpha_i & \text{otherwise} \end{cases}$$

Proof. Suppose h_1 is not a GDDFF. Then the relation in Definition 3 is not verified. Consequently, there exists a subset K of δ and $S_1 \subseteq K$ such that $\sum_{i \in S_1} c_i \leqslant c_0$ and $\sum_{i \in S_1} h_1(i) > \max_{K \in \delta}\{KP(c_0, K, \alpha)\}$ since $h_1(0) = \max_{K \in \delta}\{KP(c_0, K, \alpha)\}$. We consider different cases depending on whether or not there exists $j \in S_1$ such that $j \in J$. Items of size equal to the size of the bin are not considered:

- For all $j \in S_1, j \notin J$. In this case, $\sum_{i \in S_1} h_1(i) = \sum_{i \in S_1} \alpha_i \leqslant KP(c_0, K, \alpha) \leqslant \max_{K \in \delta}\{KP(c_0, K, \alpha)\}$. Thus the relation in Definition 3 is verified.
- There exists $j \in S_1$ such that $j \in J$. Then, based on Definition 3, we have the following relation:

$$h_1(j) + \sum_{i \in S_1 \setminus \{j\}} h_1(i) > \max_{K \in \delta}\{KP(c_0, K, \alpha)\} \tag{2}$$

We know that since $j \in J$, we have

$$h_1(j) = \max_{K \in \delta}\{KP(c_0, K, \alpha)\} - \max_{K \in \delta / j \in K}\{KP(c_0 - c_j, K \setminus \{j\}, \alpha)\}$$

Replacing $h_1(j)$ by its value in (2) leads to

$$\sum_{i \in S_1 \setminus \{j\}} \alpha_i > \max_{K \in \delta / j \in K}\{KP(c_0 - c_j, K \setminus \{j\}, \alpha)\}. \tag{3}$$

Since $\sum_{i \in S_1} c_i \leqslant c_0, \sum_{i \in S_1 \setminus \{j\}} \alpha_i \geqslant \max_{K \in \delta / j \in K}\{KP(c_0 - c_j, K \setminus, \alpha)\}$, which leads to a contradiction with (3).

Consequently, h_1 is a GDDFF for the input datum. □

For the present function, the number of knapsack problems to be solved may also be exponential. Therefore, we have to focus on special cases where the number of possible maximal subsets is not too large. In the general case, this happens when the number of possible subsets is limited (most of the patterns are forbidden). In the remainder, we describe our approach in the case of pairwise conflicts.

4.2.2. The special case of pairwise conflicts

In this particular case, obtaining all possible subsets of items is equivalent to enumerating all maximal stable sets in G, which are generally in exponential number in a graph. The maximal cliques in a graph (equivalent to the stable sets in the complementary graph) can be computed in linear time if the graph is *chordal* (every cycle of length >3 contains a chord, that is, an edge joining two non-consecutive vertices of the cycle). Tarjan and Yannakakis proved in [15] that any chordal graph G has at most n maximal cliques. In addition, they described a linear algorithm to recognize a chordal graph and to enumerate its maximal cliques. In our case, the compatibility graph $\overline{G}$ is rarely chordal. Since we are seeking a lower bound, we can relax the problem by adding edges to $\overline{G}$ so that the resulting graph is chordal. Finding the minimum set of edges (Minimum Fill-in) to add in order to obtain a chordal graph is a $\mathcal{NP}$-Hard problem. Practically, we use the algorithm called maximum cardinality search (MCS) [15] to triangulate a given graph (i.e. make it chordal).

4.3. A second GDDFF: h_2

The second GDDFF h_2 is dedicated to instances whose conflicts can be modeled by a graph of conflicts $G = (I \setminus \{0\}, E)$. Function h_2 is based on the concept of tree-decomposition, which captures the possible associations of items.

Note that when only pairwise conflicts are considered, the set δ of possible maximal subsets is composed of the stable sets in the conflict graph G. We use the notation $\delta(G)$ for the set of all maximal stable sets of G. For this function, we work on the graph of compatibility $\overline{G}$ (dual of the conflict graph G).

Let $T = (S, A)$ be a tree-decomposition of $\overline{G}$ computed using the MCS algorithm, where S is a set of subsets of $I \setminus \{0\}$ and A is a set of arcs that links the nodes of S in a tree.

The basic idea of h_2 is to assign a given DFF f_s to each node s of the tree-decomposition T. Let F be a list of valid DFF $f_1, \ldots, f_{|S|}$, one for each node of the tree-decomposition. For each vertex i in the graph we define S^i the set of nodes of the tree-decomposition containing i. The value that is assigned to i is $\min_{s \in S^i} f_s(c_i)$. In order to obtain a discrete function, we apply a scaling factor $\psi = \Pi_{s \in S} f_s(c_0)$. Clearly there is always a set of functions $f_1, \ldots, f_{|S|}$ that allows to dominate the application of a single DFF (e.g. $f_1 = f_2 = \ldots = f_{|S|}$).

Proposition 3. *The following family of functions hi is a family of GDDFF associated with the instance $(I, c, \delta(G))$. Formally $h_2 : I \to [0, \psi]$ follows*

$$i \mapsto \begin{cases} \min\limits_{s \in S^i} \left\{ \frac{f(c_i)}{f(c_0)} \psi \right\} & \text{if } i \neq 0 \\ \psi & \text{if } i = 0 \end{cases} \tag{4}$$

Proof. By definition, any clique of the compatibility graph $\overline{G}$ is included in a node of the tree-decomposition T of $\overline{G}$. Now consider a node s_i in T such that $\sum_{j \in s_i} c_j \leqslant c_0$. Applying h_2 on item $j \in s_i$ leads to the inequality $h_2(j) \leqslant f(c_j)\psi / f(c_0)$ for all $f \in F^j$. By adding these inequalities for all items $j \in s_i$, we obtain $\sum_{j \in s_i} h_2(j) \leqslant \sum_{j \in s_i} f(c_j)\psi / f(c_0)$. Since f is a valid DFF, $\sum_{j \in s_i} f(c_j)/f(c_0)$, and thus $\sum_{j \in s_i} f(c_j)\psi / f(c_0) \leqslant \psi$, which is the result sought. □

Another issue is to choose a suitable set of functions to be applied to the nodes of the tree-decomposition. We use the following heuristic. For each node s, we compute the value of the bound associated with each function of our initial set F and we record the function that leads to the best value. This strategy may not be optimal but it leads to fast bounds.

5. New lower bounds

In this section, we present our new lower bounds for the bin packing problem with conflicts in its two variants (BPC-1D and BPC-2D).

We first describe two reduction procedures that we apply on a BPC instance before applying any of the new lower bounds. These procedures can be used together and may reduce the size of the original instance. Then we propose a method for strengthening the transportation model used in [7,8]. Finally, we describe how DFF and our new GDDFF can be embedded into lower bounding procedures.

5.1. Reduction procedures

In the following, we describe simple preprocessing methods, which can be applied to a BPC instance to reduce its size $(n \to n')$. For the sake of simplicity, these procedures will be formalized for the two-dimensional case.

The first reduction procedure, proposed by Boschetti and Mingozzi [16] for the BP-2D does not remove any item, but increases the size of some items by reclaiming lost areas in the bins. For this purpose, each item $i \in I$ is considered in turn, and the quantity of lost space above i in any solution is computed. This value can be evaluated by solving a knapsack problem as defined in (1), and safely added to the height of i. The same procedure can be applied to the width. Each knapsack problem can be solved in linear time if the size of the bin is a small constant, and leading to a time complexity of $O(n^2)$.

The second one is a generalization of a reduction procedure proposed by Carlier et al. [11] for the BP-2D. It consists in considering a set of m large items (items that cannot be packed together in a bin) and checking if these items and all corresponding small items can be packed into m bins. When conflicts are considered, this procedure can be improved by considering a clique of the conflict graph instead of the set of large items. The complexity of this reduction procedure depends on the algorithms used to compute the large clique and to solve the packing problem. In our case, a large clique is computed by the mean of Johnson's heuristic [14] of time complexity $O(n^2)$ and the packing problem is solved with a time complexity of $O(n^3)$.

5.2. Strengthening the transportation model of [7,8]

The transportation problem (TP), proposed by Gendreau et al. [7] and used by Fernandes-Muritiba et al. [8], is also used by the new lower bound as a subroutine. In order to obtain a better bound, we enforce the original flow model to have less conflicting items in the same bin after application of the flow algorithm.

The flow model of [7,8] does not take into account the fact that two items i and j of $I \setminus Q$ may be in conflict with each other. Let k be an element of Q, then two arcs (i, k) of capacity c_i and (j, k) of capacity c_j will be created. Therefore, several parts of the two conflicting items i and j may be assigned to the same bin, although this violates the compatibility constraint. In order to escape from this situation, we create a dummy vertex that we call ijk. The arcs (i, k) and (j,k) are then removed, and three new arcs are created: (i,ijk) of capacity c_i. (j, ijk) of capacity c_j, and an arc (ijk,k) of capacity $\max\{c_i, c_j\}$. Formally, the compatibility constraint can be written as follows for two conflicting items i and $j : x_{ik} + x_{jk} \leqslant 1$. Since this constraint is not handled directly by the flow algorithm, we relax it to the following inequality: $c_i x_{ik} + c_j x_{jk} \leqslant \max\{c_i, c_j\}$.

Fig. 1 shows an example for an instance $I = \{i_1, i_2, i_3, i_4, i_5\}$ and $G = (I, E)$ where $E = \{(i_1, i_2), (i_2, i_3), (i_1, i_5), (i_3, i_4)\}$. Let $Q = \{i_4, i_5\}$ be a clique over G. The case "a" shows the flow model of [7,8] and case "b" shows our one.

In Fig. 1(a), it may be noted that there is no restrictions on the items representing the "origin" of the TP. These restrictions are used to control the flow outgoing from two conflicting items to the same bin (e.g. i_1 and i_2). In Fig. 1(b) we can see how to model these restrictions using dummy vertices. As for example, a dummy vertex (red one) is inserted between the items i_1 and i_2 and the bin containing i_4 since i_1 and i_2 are in conflict, and the outgoing arc is valuated by the value of $\max\{c_{i_1}, c_{i_2}\}$.

The same transformation is performed for each triplet (i,j,k). The set of dummy vertices to be created may be large. In order to reduce this set, we propose to calculate for each item k in Q the set of maximal cliques over the set of items in $I \setminus Q$ compatible with k. Then, for each clique of items we create a dummy vertex as described above.

When the network is large, a subset of these vertices is generated, and then dynamically added in an iterative process. The new network created leads to results that dominate those obtained using the network of [7,8].

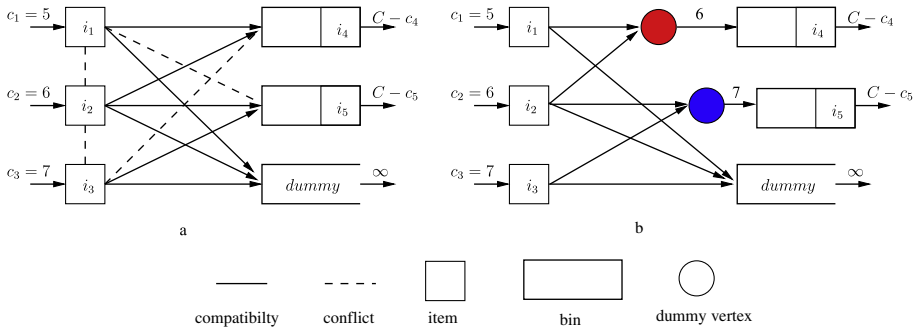

Fig. 1. Case "a" shows the flow model of [7,8] and case "b" shows our flow model.

The max flow algorithm takes $O(\hat{n}^3)$ time, where $\hat{n}$ is the number of vertices in the new network. This number is in $O(n^2)$, which would lead to a theoretical complexity of $O(n^6)$. Practically, the number of vertices is far smaller than n^2. However, for huge instances, it may be useful to limit the number of dummy vertices in such a way that $\hat{n} \in O(n)$.

5.3. A lower bound based on DFF

Fernandes-Muritiba et al. [8] proposed LB_{DFF} as a lower bound based on DFF. Their computational experiments seem to indicate that LB_{DFF} is worthless for BPC-1D when they are applied directly to the instance.

In order to improve LB_{DFF}, we propose LB^{imp}_{DFF} which mixes a bound based on the application of a DFF with the bound obtained by computing a clique in the graph.

In the following, $\mathcal{F} = \{f^k_0, f^k_1 : 1 \leqslant k \leqslant \frac{1}{2}C\}$ is the finite set of the DFF described in Section 2. Any other DFF can be added to $\mathcal{F}$ and may help improve the value of the lower bounds.

We use a given DFF $f \in \mathcal{F}$ to decompose the set of items into two sets. Each item i is assigned to a set depending on the value of $f(c_i)$. Let $I_l = \{i \in I : f(c_i) = f(C)\}$ be the set of large items, and $I_s = \{i \in I : 0 \leqslant f(c_i) < f(C)\}$ the set of small items. Let us denote by Q_s a big clique over I_s. The two-dimensional variant of LB^{imp}_{DFF} is computed as follows

$$LB^{imp}_{DFF} = |I_l| + \max\left\{\left\lceil \frac{\sum_{i \in I_s} f(w_i) f(h_i)}{f(W) f(H)} \right\rceil, |Q_s|\right\}$$

For the two-dimensional case, the time complexity of this bound is $O(n^3)$. Note that LB^{imp}_{DFF} is applied on the instance modified by the mean of the preprocessing procedures described in Section 5.1.

5.4. Lower bounds based on GDDFF

We denote by LB^{IMP}_{CP} the new lower bound that we propose as an improvement on LB^{imp}_{CP}, in which we start by applying a preprocessing phase, then we apply the GDDFF h_1 defined in Section 4 in order to modify the size of the items, and finally we solve the strengthened TP just described above. In our case, the TP is applied once whereas [7,8] uses a *step value* and solve a TP instance for each step value. Applying function h_1 compensates a part of the lost geometric information due to the relaxation of a BPC-2D instance into a BPC-1D one. The time complexity of this bound is equal to the time complexity of the algorithm used to solve the transportation problem.

Computing lower bounds using the GDDFF h_2 is straightforward. Once h_2 is applied to the initial instance, the continuous lower bound LB_0 is then applied on the modified instance. Similarly to LB_{DFF} and LB^{imp}_{CP}, The time complexity is equal to $O(n^3)$.

6. Computational experiments

In this section, we report on our computational experiments concerning the lower bounds described in this paper. Our algorithms, as well as those of [8], were coded in C++ and run on a Pentium IV CPU 3 GHz with 1 GB RAM.

The BPC-1D instances were proposed by Fernandes-Muritiba et al. [8] and publicly available for download at http://www.or.deis.unibo.at (see [8] for complete description).

The BPC-2D instances were constructed using the BP-2D instances generated by Berkey and Wang [13] and Martello and Vigo [18]. The benchmark consists of ten different classes of instances. For each class, five values of $n = \{20, 40, 60, 80, 100\}$ are considered. For each class and value of n, 10 instances have been considered and a conflict graph G was generated for each instance (see [7,8]). We used 10 values for the density of G, $d = \{0\%, 10\%, \ldots, 90\%\}$, yielding 5000 instances in total.

In the following, due to the large size of the test-bed, we focus on average results. Tables 1 and 2 consists of four main columns. The first column gives, for each line in the first subtable, the class (cl) to which the instances belong and the conflict graph density ($\%d$) in the second subtable. The second column UB gives the values of the upper bound. The third column called KCT represents our lower bounds and the fourth one called FIMT those of [8]. To better asses the performance of each lower bound LB, we evaluated the percentage gap as $100(UB - LB)/UB$. In the colmun KCT, we can notice that our bounds were run twice, one time with preprocessing and the other time without preprocessing. To measure the effectiveness of the reduction procedures, the column $\%n \rightarrow n'$ represents the percentage reduction of the original instance size evaluated as $100(n - n')/n$.

6.1. Numerical results for BPC-ID

In Table 1 we present the results obtained by the following lower bounds: $LB_{DFF}, LB^{imp}_{DFF}, LB^{imp}_{CP}, LB^{IMP}_{CP}$ and LB_{SC}. The upper bound UB is the value reported by [8]. Each line in the first subtable shows average values over 100 instances whereas each line in the second subtable shows average values over 80 instances.

The lower bound LB^{imp}_{DFF} is equal to LB_{DFF} when it is applied without a preprocessing phase. Therefore, these results are not reported in Table 1. The computing time of LB_{DFF} and LB^{imp}_{DFF} is small enough so that it was not reported as well as the number of optimal solutions reached by LB_{DFF} we can see in Table 1, LB^{imp}_{DFF} may lead to better results than LB_{DFF} when applied after our preprocessing phase. In the upper subtable, the average percentage gap of LB_{DFF} is always larger than 20% while it does not exceed 2% in case of LB^{imp}_{DFF}. The average percentage gap of LB^{imp}_{DFF} does not exceed 2% in the two subtables, while LB_{DFF} is always larger than 20% and can exceed 50% in the upper and lower subtables, respectively.

Table 1
Results for the instances from Fernandes-Muritiba *et al.* [8]. The column *cl* represents the class, *%d* gives the density of the conflict graph, and *U* gives the best known upper bound produced by [8]. The column KCT represents our lower bounds and FIMT those of [8]. For each lower bound, *%gap* represents the average percentage gap, *sec.* is the computing time in seconds and *#opt* is the number of instances for which $LB = UB$. The column $\% \ n \to n'$ represents the average percentage reduction of the original instance size. For each subtable, the line *Avg* shows the overall average results and *Ttl* shows the total value.

		KCT								FIMT						
		Reduction procedures		LB_{DFF}^{imp}		LB_{CP}^{IMP}				LB_{DFF}	LB_{CP}^{imp}			LB_{SC}		
	UB	$\%n \to n'$	sec.	%gap	#opt	%gap	#opt	With preprocessing sec.	Without preprocessing sec.	%gap	%gap	sec.	#opt	%gap	sec.	#opt
cl																
1	68.17	24.56	0.12	0.61	73	0.12	90	0.42	0.37	22.05	0.24	0	87	0.01	22.8	99
2	139.49	21.34	0.07	0.35	74	0.1	85	0.57	0.90	20.33	0.18	0.1	78	0	57.4	100
3	277.82	21.27	0.45	0.29	65	0.11	82	7.39	7.01	20.3	0.17	0.4	74	0.07	151.2	95
4	555.03	20.64	0.80	0.19	65	0.07	81	7.62	8.56	20.54	0.09	2.3	78	0.04	275.1	98
5	32.06	28.58	0.00	1.57	65	0.56	87	0.74	0.19	28.37	0.98	0	78	0.05	38.3	99
6	63.54	23.58	0.01	0.65	76	0.27	87	0.72	0.30	27.49	0.27	0	87	0.12	44.8	95
7	130.59	20.96	0.06	0.34	64	0.11	87	0.78	1.30	26.79	0.11	0.1	87	0.05	68.5	96
8	264.87	20.82	0.39	0.16	70	0.05	89	6.33	6.02	27.35	0.05	0.5	89	0.03	232.6	96
Avg	191.45	22.71	0.24	0.52		0.17		3.07	3.56	24.15	0.26	0.43		0.05	111.34	
Ttl					552		688						658			778
%d																
0	133.01	0.00	0.53	0.00	80	0.00	80	1.94	2.10	0.00	0.00	0.3	80	0.00	40.7	80
10	133.05	0.00	0.55	0.13	77	0.13	77	1.7	1.91	0.13	0.13	0.2	77	0.01	127.6	79
20	133.16	0.00	0.54	0.61	68	0.37	72	1.67	2.19	0.61	0.61	0.4	68	0.14	131.4	76
30	133.54	0.00	0.51	0.81	58	0.45	64	1.82	2.32	1.41	0.69	0.6	60	0.16	289.3	71
40	144.43	1.25	0.56	1.39	34	0.27	61	5.77	7.10	10.66	0.34	0.7	58	0.13	262.7	73
50	177.36	4.23	0.56	0.78	40	0.11	69	6.97	7.24	27.38	0.18	0.7	66	0.00	41.1	80
60	213.16	25.56	0.35	0.52	41	0.12	64	3.06	4.02	39.63	0.22	0.6	58	0.00	53.0	80
70	247.38	45.01	0.18	0.53	46	0.14	65	3.99	4.37	47.97	0.20	0.5	59	0.00	60.3	80
80	282.25	66.11	0.09	0.26	52	0.09	66	2.60	2.71	54.38	0.17	0.3	63	0.01	73.9	79
90	317.13	85.02	0.03	0.16	56	0.06	70	1.23	1.64	59.36	0.09	0.1	69	0.00	33.4	80
Avg	191.45	22.71	0.24	0.52		0.17		3.07	3.56	24.15	0.26	0.43		0.05	111.34	
Ttl					552		688						658			778

Table 2
Results for the instances derived from Berkey and Wang [13] and Martello and Vigo [18]. The column *cl* represents the class, *%d* gives the density of the conflict graph, and *U* gives the upper bound produced by the heuristic *Bottom-Left-Conflict*. The column KCT represents our lower bounds and FIMT those of [8]. For each lower bound, *%gap* represents the average percentage gap, *sec.* is the computing time in seconds and *#opt* is the number of instances for which $LB = UB$. The column $\% \ n \to n'$ represents the average percentage reduction of the original instance size. The line *Avg* shows the average results for the whole classes and *Ttl* shows the total value for the whole classes.

		KCT													FIMT		
		With preprocessing							Without preprocessing								
		Reduction procedures		LB_{DFF}^{imp}		LB_{CP}^{IMP}			LB_{DFF}		LB_{CP}^{IMP}				LB_{CP}^{imp}		
	UB	$\%n \to n'$	sec.	%gap	#opt	%gap	sec.	#opt	%gap	#opt	%gap	sec.	#opt		%gap	sec.	#opt
cl																	
1	36.65	51.3	0.00	5.47	163	5.19	0.12	170	5.56	154	5.21	0.35	169		7.49	0.11	110
2	26.65	84.8	0.00	1.33	455	1.33	0.10	455	1.33	455	1.33	0.40	455		1.33	0.11	455
3	32.9	53.63	0.01	5.69	177	5.61	0.21	180	5.84	172	5.61	0.41	180		7.46	0.21	139
4	25.2	83.4	0.03	1.68	435	1.68	0.36	435	1.68	435	1.68	0.64	435		1.68	0.21	435
5	33.83	50.77	0.04	6.05	174	5.9	1.03	178	6.27	167	5.91	1.84	178		10.02	0.21	118
6	28.89	86.6	0.03	1.46	458	1.46	1.97	458	1.46	458	1.46	2.70	458		1.46	0.22	458
7	32.32	43.9	0.00	6.03	183	5.68	1.10	188	6.16	181	5.69	1.54	188		9.91	0.21	155
8	33.1	42.62	0.03	6.19	159	6.04	1.28	161	6.39	153	6.05	1.96	161		10.12	0.22	102
9	48.16	94.65	0.00	0.76	344	0.72	0.43	352	0.85	333	0.72	0.88	352		14.85	0.18	78
10	27.8	52.65	0.02	7	191	6.9	1.13	193	7.08	186	6.91	1.79	193		6.95	0.22	165
Avg	32.55	64.43	0.01	4.17		4.05	0.77		4.26		4.05	1.25			7.13	0.19	
Ttl					2739			2770		2694			2769				2215
%d																	
0	15.98	21.1	0.04	6.48	218	6.42	0.93	221	6.63	213	6.42	1.48	221		12.76	0.18	160
10	16.95	33.29	0.04	7.17	180	7.13	0.81	183	7.28	176	7.13	1.21	183		13.38	0.19	125
20	19.63	40.29	0.03	8.65	197	8.58	0.89	198	8.73	193	8.58	1.15	198		13.77	0.2	148
30	23.57	50.79	0.01	7.92	231	7.63	1.03	234	7.97	229	7.63	1.95	234		11.06	0.2	196
40	28.16	58.85	0.02	4.45	247	4.24	1.18	252	4.67	246	4.29	1.67	251		7.07	0.2	213
50	33.22	70.21	0.02	3.5	231	3.16	0.8	237	3.64	227	3.18	1.2	237		5.77	0.2	195
60	39.32	84.18	0.00	1.65	292	1.59	0.76	294	1.73	287	1.61	1.21	294		3.23	0.19	241
70	44.71	90.08	0.00	1.04	329	1	0.42	331	1.12	324	1	0.81	331		2.25	0.18	267
80	49.55	97.12	0.00	0.53	383	0.51	0.56	386	0.58	378	0.51	1.10	386		1.26	0.18	316
90	54.42	98.41	0.00	0.26	431	0.24	0.35	434	0.31	421	0.24	0.73	434		0.73	0.17	354
Avg	32.55	64.43	0.01	4.17		4.05	0.77		4.26		4.05	1.25			7.13	0.19	
Ttl					2739			2770		2694			2769				2215

Applying LB_{CP}^{IMP} with or without a preprocessing phase gave the same results in terms of percentage gap and optimality. The only difference is the computing time, which is smaller when applying a preprocessing phase. However, in both cases, LB_{CP}^{IMP} was capable of providing a lower bound value better than the one found by LB_{CP}^{imp} for 39 instances. The average percentage gap of LB_{CP}^{IMP} is equal to 0.17 and 0.26 in case of LB_{CP}^{imp}, which means that we gain an average percentage gap of 0.09, but this gain may reach 4.76 for some instances. Using LB_{CP}^{IMP}, the number of optimal solutions was increased by 30 with respect to LB_{CP}^{imp}. This is however obtained with a larger CPU time (3.07 s in average and 91.04 s for the most time consuming instance [*BPPC_4_ 7_8.txt* : $LB_{CP}^{IMP} = 712$ in 91.04 s, $LB_{CP}^{IMP} = 704$ in 4.95 s and $LB_{SC} = 712$ in 154 s]) against the very small CPU time required by LB_{CP}^{imp}.

Regardless the computing time that may be large, LB_{SC} remains the best lower bound in all the cases. This is not surprising since LB_{SC} makes use of more advanced and time consuming techniques than those of LB_{CP}^{imp} or LB_{CPC}^{IMP}. The computing time entailed by LB_{CP}^{IMP} is far smaller than the time needed by LB_{SC} and remains reasonable when we deal with dense graphs. In addition, it is worth pointing out that in 32 cases out of 800, LB_{CP}^{imp} reached LB_{SC} with a smaller average computing time equal to 16.29 s instead of 142.16 s.

The reduction procedures seem to be efficient for densities greater than 60%. For density 90%, most of the items are removed, which is not surprising, since in an optimal solution, each of the items tends to be packed alone in a bin.

For the one-dimensional case, if one needs a fast lower bound, our preprocessing and lower bounding methods should be used. If a larger amount of time is allowed, LB_{SC} should be used.

6.2. Numerical results for BPC-2D

Our results for the two-dimensional case are obtained by applying our preprocessing methods and GDDFF on a two-dimensional instance, and then relaxing it into a one-dimensional instance.

Since no previous results were available for the BPC-2D, we compared our lower bounds with LB_{CP}^{imp} proposed by Fernandes-Muritiba et al. [8] for the BPC-1D. Similarly to our lower bounds, the initial two-dimensional instance is relaxed into a one-dimensional instance and then LB_{CP}^{imp} is applied. The comparison with LB_{SC} is not straightforward since it embeds a population heuristic based tabu search. Consequently it does not appear in Table 2. The objective of this numerical analysis is to prove the effectiveness of our methods and to show the improvements in terms of lower bounding values that we can brought with respect to LB_{CP}^{imp}.

In Table 2 we report the results obtained by the following lower bounds: $LB_{DFF}, LB_{DFF}^{imp}, LB_{CP}^{imp}$ and LB_{CP}^{IMP}. Each line in the two subtables shows average values over 500 instances. The upper bound value *UB* is computed by the mean of the heuristic that we called *Bottom-Left-Conflict (BLC)*, which is a simple adaptation of the classical Bottom-Left heuristic developed by Coffman et al. [17].

In the first subtable we can notice that $LB_{DFF}^{imp}, LB_{CP}^{imp}$ and LB_{CP}^{IMP} are equal over all graph densities for classes 2, 4 and 6. In case of class 1, LB_{CP}^{IMP} improves LB_{CP}^{imp} for 43.6% of the 500 instances, 28.6% for class 3, 54.6% for class 5, 44% for class 7, 41.4% for class 8, 82.2% for class 9 and 13.2% for class 10. For the class 9, results of LB_{CP}^{IMP} are clearly better than those of LB_{CP}^{imp} and are near optimal with an average difference of 0.66.

The results show also that our new lower bound LB_{CP}^{IMP} provides better results than LB_{CP}^{imp} even when applied without a preprocessing phase, and the same holds for LB_{DEF}^{imp}. Applying LB_{CP}^{IMP} on a preprocessed instance is faster than applying it on a non-processed instance. Note that the improvements brought by LB_{CP}^{IMP} with respect to LB_{CP}^{imp} are not due to the preprocessing procedures. On the contrary, LB_{DFF}^{imp} brings some improvements compared to LB_{DFF}.

Being weaker than LB_{CP}^{imp} for some instances (1.4% for class 1, 2.6% for class 2, 1% for class 7, 1.6% for class 8, 6% for class 10) can be justified by the fact that the triangulation of $\overline{G}$ may be of bad quality. In order to obtain the best bound available, a solution is to apply both lower bounds.

7. Conclusion

In this paper, we have studied several reduction procedures and techniques which can be used to improve the quality of the existing polynomial lower bounds for the bin packing problem with conflicts.

Investigating the concept of dual-feasible functions (DFF) and data-dependent dual-feasible functions (DDFF), we were able to propose a general framework for deriving new DDFF as well as the new concept of generalized data-dependent dual-feasible functions (GDDFF), a conflict generalization of this framework. The new concept of GDDFF enables any new DDFF to take into account the structure of the conflict graph. In addition, applying a GDDFF to the initial instance before solving the transportation problem allows the new lower bound to capture additional two-dimensional information.

The numerical results confirm the improvement brought by the new proposed techniques on the lower bound LB_{CP}^{imp} of Fernandes-Muritiba et al. [8]. For the one-dimensional case, the improvement seems to be slight, but is obtained in a reasonable computing time. However, better time consuming lower bounds can be obtained by using the LP relaxation of the set covering formulation [8]. For the two-dimensional case, our bounds improve the lower bound value for 30.76% of the benchmark. The improvement obtained can be large for some instances (up to 77.78%).

We are actually working on a bi-objective exact approach [21,22] in which we investigate a conflict generalization of the *one-dimensional knapsack problem* as well as the *rectangular knapsack problem* (see [19,20]).

Acknowledgements

We thank Region Nord-Pas de Calais for giving support to this project. The computational experiments have been executed at the French National Institute for Research in Computer Science and Control (INRIA Lille).

We also would like to thank the referees for their useful remarks.

References

[1] K.A. Dowsland, W.B. Dowsland, Packing problems, European Journal of Operational Research 56 (1) (1992) 2–14.
[2] A. Lodi, S. Martello, M. Monaci, Two-dimensional packing problems: a survey, European Journal of Operational Research 141 (2) (2002) 241–252.
[3] D. Psisinger, Heuristics for the container loading problem, European Journal of Operational Research 141 (2) (2002) 382–392.
[4] G. Wascher, H. Haussner, H. Schumann, H. Nagamochi, An improved typology of cutting and packing problems, European Journal of Operational Research 183 (3) (2007) 1109–1130.
[5] D.S. Johnson, Near Optimal Bin Packing Algorithms, Ph.D. Dissertation, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1973.
[6] K. Jansen, S. Ohring, Approximation algorithms for time constrained scheduling, Information and Computation 132 (2) (1997) 85–108.
[7] M. Gendreau, G. Laporte, F. Semet, Heuristics and lower bounds for the bin packing problem with conflicts, Computers and Operations Research 31 (3) (2004) 347–358.
[8] A.E. Fernandes-Muritiba, M. Iori, E. Malaguti, P. Toth, Algorithms for the bin packing problem with conflicts, Informns Journal on Computing, October 2, 2009 (Published online in Articles in Advance). doi: 10.1287/ijoc.l090.0355.
[9] S. Fekete, J. Schepers, New classes of fast lower bounds for bin packing problems, Mathematical Programming 91 (2001) 11–31.
[10] S. Fekete, J. Schepers, A general framework for bounds for higher-dimensional orthogonal packing problems, Mathematical Methods of Operations Research 60 (2004) 311–329.

[11] J. Carlier, F. Clautiaux, A. Moukrim, New reduction procedures and lower bounds for the two-dimensional bin packing problem with fixed orientation, Computers and Operations Research 34 (2007) 2223–2250.
[12] F. Clautiaux, C. Alves, J.V. de Carvalho, A survey of dual-feasible functions for bin-packing problems, Annals of Operations Research (2008).
[13] J.O. Berkey, P.Y. Wang, Two-dimensional finite bin packing algorithms, Journal of the Operational Research Society 38 (1987) 423–429.
[14] D.S. Johnson, Approximation algorithms for combinatorial problems, Journal of Computer and System Science 9 (1974) 256–278.
[15] R.E. Tarjan, M. Yannakakis, Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs, SIAM Journal on Computing 13 (3) (1984) 566–579.
[16] M. Boschetti, A. Mingozzi, The two-dimensional finite bin packing problem, part I: new lower bounds for the oriented case, Quarterly Journal of Operations Research 1 (2003) 27–42.
[17] E.G. Coffman, M.R. Garey, D.S. Johnson, Approximation algorithms for bin-packing – an updated survey, Algorithm Design for Computer System Design 0 (1984) 49–106.
[18] S. Martello, D. Vigo, Exact solution of the two-dimensional finite bin packing problem, Management Science 44 (3) (1998) 388–399.
[19] G.F. Cintra, F.K. Miyazawa, Y. Wakabayashi, E.C. Xavier, Algorithms for two-dimensional cutting stock and strip packing problems using dynamic programming and column generation, European Journal of Operational Research 191 (l) (2008) 61–85.
[20] S. Michel, N. Perrot, F. Vanderbeck, Knapsack problems with setups, European Journal of Operational Research 196 (3) (2009) 909–918.
[21] F. Clautiaux, J. Carlier, A. Moukrim, A new exact method for the two-dimensional orthogonal packing problem, European Journal of Operational Research 183 (3) (2007) 1196–1211.
[22] M. Kenmochi, T. Imamichi, K. Nonobe, M. Yagiura, H. Nagamochi, Exact algorithms for the two-dimensional strip packing problem with and without rotations, European Journal of Operational Research 198 (1) (2009) 73–83.