Computers & Industrial Engineering 66 (2013) 158–170

Contents lists available at SciVerse ScienceDirect

Computers & Industrial Engineering

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

Iterative approaches for solving a multi-objective 2-dimensional vector packing problem

Nadia Dahmani[a], François Clautiaux[b,*], Saoussen Krichen[a], El-Ghazali Talbi[b]

[a] Institut Supérieur de Gestion de Tunis, LARODEC, 41 Avenue de la Liberté, Cité Bouchoucha, 2000 Le Bardo, Tunisie
[b] Université de Lille 1, 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 21 May 2012
Received in revised form 2 March 2013
Accepted 26 May 2013
Available online 4 June 2013

Keywords:
2-Dimensional vector packing problem
Multi-objective optimization
Lower bounds
Heuristics
Metaheuristics

ABSTRACT

In this paper, we address a bi-objective 2-dimensional vector packing problem (Mo2-DBPP) that calls for packing a set of items, each having two sizes in two independent dimensions, say, a weight and a height, into the minimum number of bins. The weight corresponds to a "hard" constraint that cannot be violated while the height is a "soft" constraint. The objective is to find a trade-off between the number of bins and the maximum height of a bin. This problem has various real-world applications (computer science, production planning and logistics). Based on the special structure of its Pareto front, we propose two iterative resolution approaches for solving the Mo2-DBPP. In each approach, we use several lower bounds, heuristics and metaheuristics. Computational experiments are performed on benchmarks inspired from the literature to compare the effectiveness of the two approaches.

1. Introduction

In this paper, we address a new bi-objective version of the 2-dimensional vector packing problem (2-DVPP). A 2-DVPP instance consists of a set $\{1,\ldots,N\}$ of items i with two sizes c_i and h_i, and an unlimited number of bins with two sizes C and H. The objective is to pack the items into a minimum number of bins without violating the capacity constraint of each dimension. The problem is a generalization of the classical one-dimensional bin packing problem (BPP), and thus is $\mathcal{NP}$-hard in the strong sense (see Garey & Johnson, 1979).

Different greedy heuristics and exact methods have been proposed to solve d-DVPPs or more precisely the 2-DVPP (Alves, de Carvalho, Clautiaux, & Rietz, 2013; Caprara & Toth, 2001; Garey, Graham, Johnson, & Andrew, 1976; Spieksma, 1994). The d-DVPP arises in a large variety of real-world applications in computer science (assignment of jobs to processors, or virtual machine placement Lee et al., 2011), production planning and logistics (packing problems), or in steel industry (Chang, Hwang, & Park, 2005).

The literature on 2-DVPP problems focuses on the minimization of wasted space. However, in real-world applications, there are often conflicting criteria to be satisfied. In the computer processor selection with job assignment context for example, a finite number of real-time computer jobs (items) have to be assigned to a group of processors (bins). Each job has its own resource demands for CPU time and memory. Each processor has its time processing and memory resource constraints. The jobs must all run simultaneously and, for a fast time response, all must be memory-resident at all times. A decision maker may be interested in solutions with a good trade-off between the number of processors and the completion time.

In this paper, we consider a bi-objective version of the 2-dimensional vector packing problem, denoted as *multi-objective two dimensional vector packing problem* (Mo2-DBPP). We relax the height constraint, which becomes an objective to minimize. The objective is to find a trade-off between the number of used bins and the maximum height of a bin.

Only few works, published in the last 10 years, were dedicated to multi-objective bin packing problems. In addition to the the minimization of the number of used bins, other objectives were addressed. For instance, Liu, Tan, Huang, Goh, and Ho (2008) pointed out the issue of the balance of the bins, and tried to minimize the average deviation of center of gravity from an ideal position. The authors addressed the multi-objective problem using an evolutionary particle swarm optimization approach. Geiger (2007) focused on minimizing the heterogeneousness of the elements in each bin. An extension of the Best-Fit approximation algorithm was presented to solve the problem. In Sathe, Schenk, and Burkhart (2009), a cost function designed for the special real case study in the automobile sheet metal forming processes has

* Corresponding author. Tel.: +33 359632224; fax: +33 359632221.
E-mail address: francois.clautiaux@univ-lille1.fr (F. Clautiaux).

to be minimized. The problem was solved by using a combination of a multi-objective evolutionary algorithm with a clustering algorithm. Khanafer, Clautiaux, Hanafi, and Talbi (2012), addressed a bi-objective version of the bin packing with conflicts where the second objective was the minimization of the number of violated conflicts when the number of bins is fixed. Several heuristics, and an exact method were proposed to solve the problem. The previous methods proposed to solve multi-objective packing problems are dedicated to a specific cost function. They cannot be applied directly to our two-dimensional problem, since the subproblems to solve have a totally different structure.

In the Mo2-DBPP, the number of optimal points in the objective space is bounded by the number of items. This property led us to design two different resolution approaches that generate potentially efficient solutions by iteratively solving a mono-objective problem. The first method consists in iterating over the possible values of maximum height of a bin. This method gives rise to a pseudo-polynomial number of 2-DVPPs to solve iteratively. The second method consists in optimizing the maximum height of a bin while iteratively fixing the number of used bins. In this method, a fewer number of height minimization problems (MHPP) have to be solved.

The purpose of this paper is to compare the efficiency and the effectiveness of these two iterative approaches. For each one, we present integer linear programming models, several lower bounds and heuristic algorithms based on adaptations of vector packing and scheduling algorithms. We also devised three different metaheuristics: one population based algorithm and two local search methods.

To show the effectiveness of our algorithmic approaches, an experimental investigation is performed on various benchmarks inspired from the literature (Caprara & Toth, 2001). We compare the different algorithms dedicated to each approach, and then compare the two approaches in terms of computing time and solution quality.

The remainder of the paper is organized as follows: Section 2 provides a mathematical formulation of Mo2-DBPP and a general description of the two resolution approaches. Sections 3 and 4 deal with the iterative vector-packing and min-height based approaches. For each approach, we introduce heuristics and lower bounds for the related problems. In Section 5, a description of different metaheuristic algorithms is presented. Finally, our computational experiments are reported in Section 6.

2. Problem formulation and resolution approaches

In this section, we present the mathematical formulation of Mo2-DBPP. We then give a general description of our two iterative approaches.

2.1. Definition and model for the Mo2-DBPP

The Mo2-DBPP can be described as follows. Let $\{1,\ldots,N\}$ be a set of items. Each item i has a weight c_i and a height h_i. Let also $\{1,\ldots,\overline{M}\}$ be a set of bins, where $\overline{M}$ is an upper bound on the number of bins that can be used ($\overline{M} \leqslant N$). Each bin j has a weight capacity C. The two conflicting objectives that have to be simultaneously minimized are the number of used bins and the maximum height of a bin.

We now give a formal description of the problem, using binary variables x_{ij} that take the value of 1 if item i has been placed in bin j, 0 otherwise, and binary variables y_j that take the value of 1 if bin j is used, 0 otherwise. Integer variable H expresses the maximum height loaded into one bin. A mathematical formulation of the Mo2-DBPP can be stated as follows:

$$\min \left\langle \sum_{j=1}^{\overline{M}} y_j, H \right\rangle \tag{1}$$

$$\text{s.t.} \quad \sum_{j=1}^{\overline{M}} x_{ij} = 1, \quad i = 1,\ldots,N \tag{2}$$

$$\sum_{i=1}^{N} c_i x_{ij} \leqslant C y_j, \quad j = 1,\ldots,\overline{M} \tag{3}$$

$$\sum_{i=1}^{N} h_i x_{ij} - H \leqslant 0, \quad j = 1,\ldots,\overline{M} \tag{4}$$

$$x_{ij} \in \{0,1\}, \quad i = 1,\ldots,N, \quad j = 1,\ldots,\overline{M} \tag{5}$$

$$y_j \in \{0,1\}, \quad j = 1,\ldots,\overline{M} \tag{6}$$

$$H \in \mathbb{N} \tag{7}$$

The two objective functions minimize concurrently the number of used bins and the maximum height H of a bin. The partition constraints (2) ensure that each item i is assigned to exactly one bin j. Inequalities (3) express the maximum weight capacity for each bin. Inequalities (4) mean that each item size combination into a single bin j must not exceed the maximum height H.

Since we are handling a multi-objective problem, not only one optimal solution s exists but a set S of efficient solutions that minimize both objective functions. For a given solution s, let $f^1(s)$ be the number of bins in this solution and $f^2(s)$ the maximum height of a bin. A solution $s \in S$ is evaluated with respect to a vector $(f^1(s), f^2(s))$, and a decision vector s dominates a vector s' if $f^k(s) \leqslant f^k(s')$ $\forall k \in \{1,2\}$ and $\exists l | f^l(s) < f^l(s')$ for $l \neq k$. Hence, solving the problem aims to identify all the efficient outcomes or the Pareto optimal set $\mathcal{Z}^*$. A maximal set of non-dominated solutions will be denoted as Pareto approximation set $\mathcal{X}^*$.

2.2. Iterative resolution approaches

The number m of bins used is bounded by the number of items (trivially $m \leqslant N$). This means that the size of the Pareto front is at

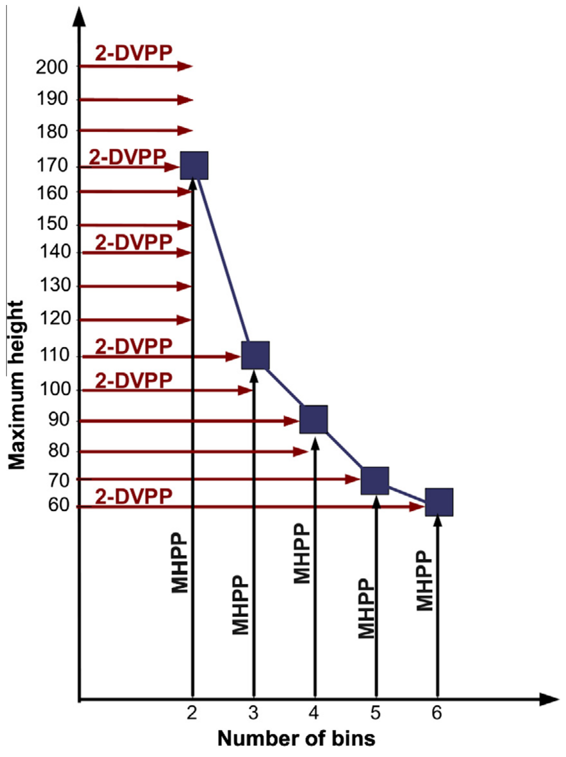

Fig. 1. Illustration of the two resolution approaches for Example 1.

most linear in the size of the data. This justifies the application of an iterative approach.

We used two different resolution approaches. The first one consists in iterating over the possible values of maximum height H. At each iteration, a 2-DVPP is solved. A second alternative is to optimize the maximum height H while iteratively fixing the number of used bins m. The problem to be solved is denoted as the *Min-height packing problem* (MHPP). These methods are similar to the classical ϵ-Constraint method (Haimes, Lasdon, & Wismer, 1971), but we add an equality constraint on the first objective instead of an inequality.

The main objective of this paper is to compare these two iterative approaches for generating the Pareto approximation set $\mathcal{X}^*$.

Example 1. Consider the following instance with $N = 8$, $C = 100$. The vector sizes of each item $i \in \{1,\ldots,N\}$: $(c_i, h_i) = \{(20,40); (60,30); (20,30); (40,60); (10,50); (30,40); (10,10); (10,60)\}$. The corresponding Pareto front depicted in Fig. 1 has been obtained by applying the proposed resolution approaches. In this example, we found a non-dominated solution for each possible number of bins between the extrema, which is not the case in general.

3. An iterative vector-packing based approach

3.1. Outline of the method

The resolution approach starts by computing an upper bound U and a lower bound L for the height H, and then iterates over all possible values.

The lower bound L is directly given by the item of maximum height. A naive upper bound would sum the heights of all items. This bound can be improved by considering the fact that not all items can fit together in the bin because of the weight constraint. Our upper bound U is thus obtained by solving a knapsack problem, where the size of the items is the weight and their profit is their height.

Not all values of H are possible. Some do not correspond with the sum of heights of any subset of items. Therefore, to ensure that a value H is useful, we solve a *Subset-Sum Problem*: Given a set of N positive integers and a value C, the objective of this problem is to find the subset of integers whose sum is the closest to C without exceeding C. If this problem has no solution for value H, then this value is skipped.

Instead of scanning linearly all possible values of H, we apply a divide-and-conquer strategy, which avoids the computation of many non-useful subproblems. At each iteration, a threshold value $H = \lfloor \frac{L+U}{2} \rfloor$ is considered. If 2-DVPP(H) = 2-DVPP(U) then set $U = H$ and iterate with the new values found. If instead 2-DVPP(H) $\neq$ 2-DVPP(U) the method is run in both intervals $[L, H]$ and $[H, U]$. A process is stopped when $L = U$.

The problem to solve at each step is a two-dimensional vector-packing problem. It can be stated as follows, using the variables of model (1)–(7):

$$\min \sum_{j=1}^{\overline{M}} y_j \tag{8}$$

$$\text{s.t.} \sum_{j=1}^{\overline{M}} x_{ij} = 1, \quad i = 1,\ldots,N \tag{9}$$

$$\sum_{i=1}^{N} c_i x_{ij} \leqslant C y_j, \quad j = 1,\ldots,\overline{M} \tag{10}$$

$$\sum_{i=1}^{N} h_i x_{ij} \leqslant H y_j, \quad j = 1,\ldots,\overline{M} \tag{11}$$

$$x_{ij} \in \{0,1\}, \quad i = 1\ldots,N, \quad j = 1,\ldots,\overline{M} \tag{12}$$

$$y_j \in \{0,1\}, \quad j = 1,\ldots,\overline{M} \tag{13}$$

3.2. Lower bounds

In this section, we present various lower bounds for the 2-DVPP from the literature, based on combinatorial considerations or linear programming.

The first lower bound was proposed by Spieksma (1994) and defined as the maximum between the continuous lower bound values of the two one-dimensional bin packing problems obtained by relaxing the first and the second dimension.

$$LB_1 = \max\left\{\left\lceil \frac{\sum_{i=1,\ldots,N} c_i}{C} \right\rceil, \left\lceil \frac{\sum_{i=1,\ldots,N} h_i}{H} \right\rceil\right\}$$

Caprara and Toth (2001) describe improved lower bounding procedures based on extensions of BPP lower bounds and an integer programming formulation whose linear programming relaxation can be solved by column generation. Recently, Alves et al. (2013) used the concept of *dual-feasible functions* (see Clautiaux, Alves, & Valério de Carvalho, 2010) to compute lower bounds for k-dimensional vector packing problems.

3.3. Heuristics

Many heuristics have been designed for solving the vector packing problem. The most popular and effective ones are First Fit Decreasing (FFD) based heuristics (see Caprara & Toth, 2001; Garey et al., 1976; Spieksma, 1994). Recently, Lee et al. (2011) presented new heuristics based on a generalization of FFD for solving a virtual machine consolidation problem. These heuristics take into account both item sizes and how well they fit the residual capacities of the opened bin. The authors showed that they outperform FFD-based heuristics in most cases.

In the 2-DVPP, there is not an obvious choice of how to sort the resources (items or bins) according to their dimensions. Hence, different ways of assigning the weights to the two dimensional vectors are possible. Therefore a scaling vector $w = \{w_1, w_2\}$ is used to normalize the sizes across dimensions and to weight the resources according to their importance. A natural choice of w is the average demand for each dimension.

$$\begin{cases} w_1 = \frac{1}{N}\sum_{i=1}^{N} c_i \\ w_2 = \frac{1}{N}\sum_{i=1}^{N} h_i \end{cases} \tag{14}$$

The *DotProduct* heuristic defines the *largest* item as the item that maximizes the dot product between the vector of remaining capacities $r = \{r_1, r_2\}$ of the current opened bin j and the vector sizes of item i. It selects the item i that maximizes the quantity $w_1 c_i r_1 + w_2 h_i r_2$ without violating the capacity constraints. The best results were obtained with a weight assignment of $\exp(0.01 \times w)$, where w stands for the size of the bin in the current dimension.

4. An iterative min-height based approach

4.1. Outline of the method

The exploration of the search space starts by computing lower and upper bounds $\tilde{m}$ and $\bar{m}$ on the number of bins. The used lower bound is LB_1 (see above) (Eilon & Christofides, 1971). The upper bound is generated using the *DotProduct* heuristic. For each valid number of bins $m \in [\tilde{m},\ldots,\bar{m}]$, a MHPP instance is optimized.

By associating items to jobs and bins to machines, we note that the problem corresponds to a scheduling problem denoted by $P\|C_{\max}$ with additional capacity constraints. In the latter problem, n jobs have to be assigned to m identical parallel machines. Each having a processing time p_i ($i = 1,\dots,n$) and the main objective is to minimize the maximum completion time of jobs (makespan). Since $P\|C_{\max}$ is known to be strongly $\mathcal{NP}$-hard, the same holds for MHPP. Considering a cardinality constraint k on the maximum number of items that have to be packed into one bin leads to a special case of MHPP denoted as $P\#|\leqslant k|C_{\max}$.

Using the variables of model (1)–(7), we now give the formal definition of MHPP.

$$\min \quad H \tag{15}$$

$$\text{s.t.} \quad \sum_{j=1}^{m} x_{ij} = 1, \quad i = 1,\dots,N \tag{16}$$

$$\sum_{i=1}^{N} c_i x_{ij} \leqslant C, \quad j = 1,\dots,m \tag{17}$$

$$\sum_{i=1}^{N} h_i x_{ij} - H \leqslant 0, \quad j = 1,\dots,m \tag{18}$$

$$x_{ij} \in \{0,1\}, \quad i = 1,\dots,N, \quad j = 1,\dots,m \tag{19}$$

$$H \in \mathbb{N} \tag{20}$$

4.2. Lower bounds

In this section, we present different lower bounds adapted from $P\|C_{\max}$, $P\#|\leqslant k|C_{\max}$. The following bounds were proposed in Dell'Amico and Martello (2001, 1995), Dell'amico, Iori, and Martello (2004), Dell'Amico, Iori, Martello, and Monaci (2006). The first lower bound L_1 is the maximum among the solution value of the continuous relaxation of the problem, the maximum height of an item and the minimum possible height needed to pack the $m + 1$ largest items.

$$L_1 = \max\left(\left\lceil \frac{1}{m}\sum_{i=1}^{n} h_i \right\rceil, \ \max_i\{h_i\}, \ h_m + h_{m+1}\right)$$

For our specific problem, we can slightly improve the lower bound of Dell'amico et al. (2004). A new lower bound can be derived from L_1 by eliminating the $n - m - 1$ smallest items and considering the weight capacity constraints. The optimal solution value of the relaxed problem is clearly no less than $h_{m+1} + \min_{i\in\{1,\dots,m\}:c_i+c_{m+1}\leqslant C} h_i$. Thus, an improved bound is

$$\widetilde{L}_1 = \max\left(L_1, h_{m+1} + \min_{i\in\{1,\dots,m\}:c_i+c_{m+1}\leqslant C} h_i\right)$$

When $n > m(k - 1)$, bound $\widetilde{L}_2$ was obtained by observing that at least one bin must contain k items among the first largest $m(k - 1) + 1$ ones. Thus, by considering the k smallest items, we have

$$L_2 = \max\left(\widetilde{L}_1, \sum_{i=(m-1)(k-1)+1}^{m(k-1)+1} h_i\right)$$

In the special case arising when $n = mk$, the bound L_2 can be improved by considering the total height of the bin that contain the largest item.

$$\widetilde{L}_2 = \max\left(L_2, h_1 + \sum_{i=n-k+2}^{n} h_i\right)$$

The overall lower bound is then $L_C = \max(L_1, \widetilde{L}_1, L_2, \widetilde{L}_2)$.

4.3. Heuristics

In this section, we describe several heuristic algorithms for solving MHPP. Most of them are adapted versions of heuristics proposed for the $P\|C_{\max}$ so as to handle the weight capacity constraint (17). Recalling that we are approximating a Pareto optimal set, the proposed heuristics have to be run for each valid number of bins.

4.3.1. The least loaded heuristic

The *least loaded* heuristic (LL) attempts to balance the load between the bins. Given a list of empty bins $\lambda = (1,\dots,m)$ and an ordered list of items σ, LL takes each item i following σ and tries to pack i into one of the current m bins by choosing the currently least loaded one (i.e. the bin of largest residual capacity). If LL fails to pack a selected item into the current m bins, a new bin is opened. The process is stopped when there are no more items to pack. The heuristic can be seen as the well known approximating algorithm *List Scheduling* (Graham, 1966) for the $P\|C_{\max}$ problem. One of the most effective sorting rules is the *Longest Processing Time* (LPT) that orders the jobs (items) in a non-increasing order of processing time (height/weight). In order to adapt LL to our two-dimensional problem, we examined various sorting rules: decreasing c_i (favoring the construction of feasible solutions), decreasing h_i (favoring good quality solutions) and decreasing $\max\{c_i, h_i\}$. After some preliminary experiments, the most effective sorting rule is decreasing h_i. This can be explained by the fact that for many instances, finding a feasible solution in term of c_i is not hard. In the iterative process, we choose the valid bin with the minimum height among the m current bins.

4.3.2. The min-bin heuristic

The *min-bin* (MB) heuristic was designed to be applied in our multi-objective framework. It starts from an initial solution with $m + 1$ bins and tries to build a new solution with m bins based on the latter one. The heuristic empties the bin j with the minimum total height and tries to repack its items into the remaining m bins. These items are sorted in a decreasing order of h_i and iteratively assigned to the first feasible bins. If the heuristic succeeds to pack all the items then bin j is discarded from the list. The process is stopped when it is unable to repack an unpacked item.

4.3.3. The multi-fit heuristic

The *multi-fit* (MF) heuristic has been proposed by Coffman, Garey, and Johnson (1978) for the $P\|C_{\max}$ problem. The heuristic finds through a binary search the smallest value $\tilde{c}$ for an associate BPP instance that uses no more than m bins of capacity $\tilde{c}$. In our case, MF takes as input a list of items $\sigma = (1,\dots,N)$ sorted in a decreasing order of h_i, and a number m of bins. The heuristic starts by computing lower and upper bounds L and U on the maximum height of a bin, according to Coffman et al. (1978). It then proceeds through a binary search to find the minimum height H that uses no more than m bins and that respects the weight constraints. At each iteration, we apply an adapted version of first fit decreasing that takes into account capacity constraints (17) and (18).

5. Improving both approaches using metaheuristics

In this section, we introduce several metaheuristics developed for both resolution approaches. We use exactly the same models and encodings for both problems. Only the cost functions and the initial solutions are modified.

One of the main issues when designing metaheuristics for solving combinatorial optimization is to find a trade-off between intensification and diversification (see for example Talbi, 2009). Whereas local search methods are known to be efficient as intensifying methods, evolutionary algorithms (EA) are powerful to explore the decision space thanks to their variation operators. These metaheuristics attested their efficiency for several challenging packing problems (see for example Falkenauer, 1996; Khanafer et al., 2012; Khanafer, Clautiaux, & Talbi, 2012). We examine the performance of these metaheuristics on solving our problem with both of our iterative approaches. We first describe an iterated tabu search (ITS) algorithm. The reader is referred to Glover and Laguna (1999) for a detailed description of tabu search algorithms. Second, a basic EA is presented in this section.

The choice of a solution representation is of considerable importance for the search operators and in the evaluation process. Several simple representations may be carried out for the same optimization problem such as permutation, float, discrete or binary vector representation. In this paper, we investigate two different solution encoding schemes, namely direct and indirect encoding schemes. In the sequel, we list the algorithms' features related to fitness function, solution representation and evaluation.

5.1. Fitness functions

For the Mo2-DBPP, different fitness criteria are worth considering. For a given solution s, let $m(s)$ be the number of bins used, and $H(s)$ the maximum height of a bin.

For the vector-packing problem where H is the height currently fixed, our metaheuristics use the following lexicographic fitness function:

$$F_{vp}(s) = \langle \max\{H(s) - H, 0\}, m(s)\rangle$$

This function first aims at finding a feasible solution. Then a solution with the smallest number of bins is sought.

In the min-height approach, many solutions may be equivalent in term of $H(s)$. Therefore, we introduce another term $h^2(s)$, the sum of the square of the height of each bin. This latter criterion is used to refine $H(s)$ and to favor solutions with few highly loaded bins. If the number of currently fixed bins is m, the lexicographic objective function is the following:

$$F_{mh}(s) = \langle \max\{m(s) - m, 0\}, H(s), h^2(s)\rangle$$

This function will always favor a feasible solution, then optimize the actual objective of height minimization. The third term is used to break the ties between solutions of same height.

5.2. An iterated tabu search with a direct encoding

A solution is encoded as a discrete vector of size N. Each vector's position a_i $(i = 1,\ldots,n)$ represents the index $j = 1,\ldots,m$ of the bin in which i is packed. Fig. 2 illustrates our solution encoding.

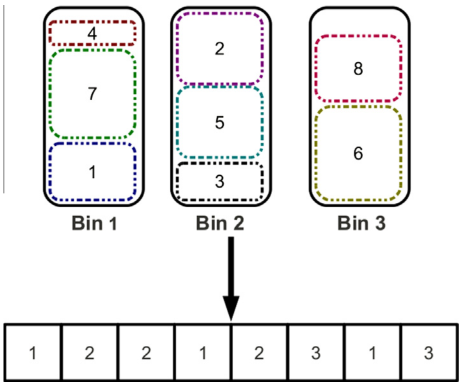

Fig. 2. An example of the direct encoding scheme.

5.2.1. Initial solution

In the vector-packing based approach, for each fixed maximum height H, the initial solution is generated by performing the *Dot-Product* heuristic.

In the min-height based approach, for each fixed number of bins m, the initial solution is generated as follows. If there is a previous generated solution with $m + 1$ bins, the min-bin heuristic (see Section 4.3.2) is performed in order to generate a new solution with m bins. If not, the LL heuristic (see Section 4.3.1) is applied. If the initial solution is not feasible, the objective function described above is used to drive the search toward feasible solutions.

5.2.2. Neighborhood exploration and evaluation

Neighbors for a current solution can be obtained by swapping items between all possible pairs of bins. We only consider swaps that preserve the capacity constraints. The swapping process starts by comparing the items in the first bin with the items in the second bin, and so on, sequentially down to the last bin in the initial solution and is repeated for each possible pair of bins. Each time the move is applied, the neighbor is evaluated through a lazy evaluation technique. The best solution in the neighborhood replaces the current solution even if it is not improving.

5.2.3. Perturbation method

When the search has not found an improving solution after a given number of iterations, a perturbation method is used. It consists in swapping $\frac{N}{4}$ times a pair of randomly chosen items. Each time the perturbation method is called, only feasible moves are allowed (i.e. swap item i from bin j with item i' from bin j' while satisfying the capacity constraints of each bin).

5.2.4. Tabu list and aspiration criterion

To prevent the search from revisiting previously visited solutions, a tabu list is used. In our implementation, the tabu list is a vector containing the attributes of tabu neighbors. Because of the loss of information concerning the search memory, the tabu list may prohibit attractive neighbors that have not yet been generated. Hence, it is necessary to use an aspiration criterion to accept some forbidden neighbors. Our aspiration criterion consists in choosing a tabu neighbor if it is the best solution found overall.

5.3. Metaheuristics with an indirect encoding

The major asset of the indirect encoding is to effectively avoid some symmetries and redundancy in the search space. This representation scheme is applied for both ITS and EA algorithms. A packing solution is represented as a permutation $\sigma = (1,\ldots,N)$ of N items. A straightforward decoder considers the permutation σ as a priority list and sequentially assigns the incoming items, in the vector-packing approach, using an adapted version of the First Fit algorithm. In the min-height approach, the LL heuristic is performed to assign the items to the m current bins. Fig. 3 presents the main steps of a solution encoding/decoding process for Example 1.

5.3.1. Initial population and variation operators for the EA

The initial population of 100 individuals is generated by combining random solutions with a solution constructed using the *Dot-Product* heuristic in the 2-DVPP approach and the LL heuristic in the min-height based approach. After the decoding process, a fitness value is assigned to every individual in the population (see Section 5.1). At each step of the algorithm, individuals are selected using the tournament selection method and reproduced using the variation operators (i.e. crossover and mutation). Finally, a generational replacement is applied to the initial population.

We use the *Two-points crossover operator* (Ishibuchi & Murata, 1998): a pair of crossing points is randomly selected, and the generated offspring preserves the items outside the selected two points from the first parent chromosome. The remaining items are inserted from the second parents respecting the order of their appearance. We also adapt two mutation operators: shift mutation operator and swap mutation operator. In the shift mutation operator, a pair of randomly chosen components are shifted from the permutation part of the chromosome, whereas in the swap mutation operator, a pair of randomly chosen components are swapped from the permutation part of the chromosome. From preliminary experiments, we noted that a good performance is achieved through randomly using the above described mutation operators, which are invoked with probabilities 0.5 and 0.5 respectively.

5.3.2. Initial solution and neighborhood exploration of the iterated tabu search

The algorithm starts by a solution constructed using the *DotProduct* heuristic in the vector-packing based approach and the LL heuristic in the min-height based approach. If LL fails to generate a solution with m bins, a random permutation is generated. The neighborhood exploration is based on pairwise exchanging operations (i.e. Two-exchange neighborhood). It consists in selecting randomly two items and swapping their position in the packing solution. The perturbation method consists in swapping $\frac{N}{4}$ times a pair of randomly chosen items. The tabu list and acceptance criteria are the same as those designed for the direct encoding case.

5.4. An hybrid metaheuristic for the min-height problem

We here describe a new hybrid metaheuristic *hybMeta* for solving the MHPP. It combines heuristics and exact methods for both MHPP and 2-DVPP problems.

Let m be the current fixed number of bins. We first compute lower and upper bounds L and U for the possible bin height using L_C and MF heuristic respectively. A binary search is then performed to find the smaller value of H between U and L that will lead to a solution with at most m bins.

This scheme leads to the resolution of several 2-DVPP problems. We first use the *DotProduct* heuristic to find an initial solution. Then a perturbation method is iteratively applied until an improved packing solution is found or the maximum computing time is reached.

The perturbation works as follows. We first choose a subset of bins to discard, and try to repack their contents into the remaining bins and new created bins. The selection of the bin subset consists in choosing a given number q of bins with total height equal to H and a random subset of "Least loaded" bins. We determine the least loaded bins with the rule described in Eq. (14). For repacking the items, we first use *DotProduct*. If it fails to reduce the number of bins, we use a branch-and-price based on Caprara and Toth (2001).

6. Experiments

We empirically investigate the performances of the vector-packing and min-height approaches in order to determine which approach is more relevant to solve Mo2-DBPP.

6.1. Benchmarks

We used the benchmarks of Caprara and Toth (2001) for 2-DVPP. Table 1 summarizes the main features of each class in the data set (*u.d.* stands for uniform distribution).

In the first six classes, the value N, and each value of c_i and h_i are random independent values generated from a uniform distribution in $[\alpha, \beta]$. Classes 1–3 were proposed in Spieksma, 1994. In these classes, each bin contains on average about 4, 2 and 2 items respectively. In order to consider instances where more items per bin are

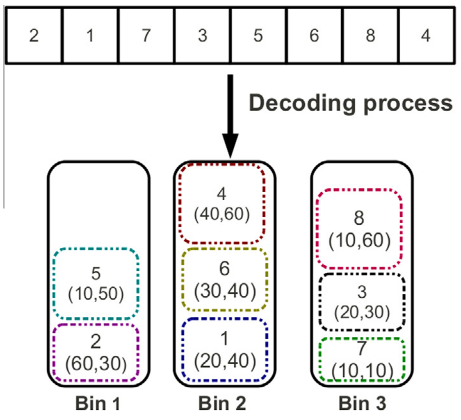

Fig. 3. The encoding/decoding process using the indirect encoding for Example 1.

Table 1
Benchmark description.

Class	C	c_i	h_i
1	1000	*u.d.*[100,400]	*u.d.*[100,400]
2	1000	*u.d.*[100,1000]	*u.d.*[100,1000]
3	1000	*u.d.*[200,800]	*u.d.*[200,800]
4	1000	*u.d.*[50,200]	*u.d.*[50,200]
5	1000	*u.d.*[25,100]	*u.d.*[25,100]
6	150	*u.d.*[20,100]	*u.d.*[20,100]
7	150	*u.d.*[20,100]	*u.d.*$[c_j - 10, c_j + 10]$
8	150	*u.d.*[20,100]	*u.d.*$[110 - c_j, 130 - c_j]$
9	$\left\lceil \frac{\sum_{i=1,\ldots,N} c_i}{L} \right\rceil$	*u.d.*[100,400]	*u.d.*[100,400]
10	100	See text	See text

packed, Caprara and Toth suggested two new classes 4 and 5 where each bin contains on average about 8 and 16 items respectively. By analogy to the most difficult instances of BPP mentioned in the literature (see Falkenauer, 1996), the authors proposed class 6, where bin capacities are equal to 150, and weights are uniformly distributed in [20,100]. In Class 7, the height and the weight are correlated for each item, whereas these two value are anti-correlated in Class 8. Instances in Classes 9 and 10 are artificially designed to be hard to solve. In Class 9, item sizes are generated as in Class 1. The value of L is computed as $\max\left\{\left\lceil \frac{\sum_{i=1,\ldots,N} c_i}{1000} \right\rceil ; \left\lceil \frac{\sum_{i=1,\ldots,N} h_i}{1000} \right\rceil\right\}$ and the bin weight capacities are defined as $C = \left\lceil \frac{\sum_{i=1,\ldots,N} c_i}{L} \right\rceil$. Instances in Class 10 are generated as the triplets (Falkenauer, 1996) for BPP. For our experimentation, about 40 instances are used by considering the problem sizes related to the values $N \in \{25, 50, 100, 200\}$ (for Class 10, $N \in \{24, 51, 99, 201\}$) and solving one instance for each value of N.

6.2. Computational results

The overall presented algorithms are implemented in C++. The CPLEX solver 12.0 is used to solve the (integer) linear models. Computational runs were performed under Linux operating system on an Intel Core 2 Duo 6600 (2.40 GHz) machine, with 2 GB RAM. All the metaheuristics presented in this paper have been implemented using the ParadisEO framework.[1]

The stopping criterion of all meta-heuristics is the CPU time. The time limit t is fixed depending on the problem size: for each

[1] ParadisEO is available at: http://paradiseo.gforge.inria.fr.

Table 2
Comparison of lower bounding procedures.

Class	n	L_{cont}			L_{int}			L_C			L_{CG}		
		%gap	r_{opt}	t	%gap	r_{opt}	t	%gap	r_{opt}	t	%gap	r_{opt}	t
1	25	4.8	0	0.02	0.93	0.27	59.1	2.71	0.2	0.11	0.06	0.73	0.11
	50	4.06	0	0.01	2.92	0.05	110.34	3.2	0.5	0.14	0.3	0.68	0.29
	100	4.63	0	0.05	3.6	0.05	170.63	2.73	0.05	0.38	0.91	0.47	0.68
	200	5.56	0	0.21	4.96	0	357.12	3.91	0.01	0.86	1.48	0.53	2
Average		4.76	0	0.07	3.1	0.09	174.3	3.14	0.19	0.37	0.69	0.6	0.77
2	25	23.13	0	0.02	0	1	0.27	8.69	0.33	0.07	0	1	0.04
	50	30.03	0	0.02	0	1	3.42	0	1	0.08	0	1	0
	100	13.68	0	0.07	8.92	0	176.59	5.24	0	0.09	0.03	0.78	0.43
	200	19.54	0	0.38	7.4	0	359.8	12.21	0	0.1	0	1	2.1
Average		21.6	0	0.12	4.08	0.5	135.02	6.54	0.33	0.09	0.01	0.95	0.64
3	25	11.1	0	0.02	3.58	0.2	57.59	6.04	0.2	0.04	0	1	0.08
	50	17.59	0	0.02	6.75	0.2	112.94	6.75	0.2	0.06	0	1	0.14
	100	13.92	0	0.07	9.62	0.06	178.5	7.54	0.05	0.24	0.02	0.84	0.41
	200	12.98	0	0.42	11.11	0	379.3	7.84	0	0.28	0.13	0.75	0.91
Average		13.9	0	0.13	7.77	0.12	118.05	7.04	0.11	0.16	0.04	0.9	0.39
4	25	3.87	0	0.01	0.99	0.21	56.28	2.23	0.13	0.14	0.07	0.79	0.12
	50	2.94	0	0.01	1.93	0.04	119.12	2.37	0.04	0.18	0.23	0.44	0.35
	100	3.51	0	0.05	2.68	0.04	177.42	2.15	0.04	0.45	0.57	0.35	0.56
	200	3.83	0	0.19	3.38	0	378.09	2.64	0	1	0.93	0.38	4.95
Average		3.54	0	0.07	2.25	0.07	182.73	2.35	0.05	0.44	0.45	0.49	1.5
5	25	3.62	0	0.01	0.88	0.5	35.26	2.26	0.13	0.11	0.22	0.67	0.11
	50	2.62	0	0.01	1.7	0.07	118.53	2.26	0.04	0.37	0.3	0.5	0.32
	100	3.34	0	0.04	2.46	0.02	179.4	2.21	0.02	0.67	0.36	0.45	0.58
	200	3.36	0	0.17	2.56	0	356.3	1.37	0.08	1.42	0.69	0.3	13.41
Average		3.24	0	0.06	1.9	0.15	172.37	2.03	0.07	0.64	0.39	0.48	3.61
6	25	6.82	0	0.02	1.63	0.29	56.71	3.78	0.14	0.08	0.12	0.83	0.09
	50	7.52	0	0.02	3.77	0.27	119.04	4.39	0.27	0.16	0.54	0.45	0.25
	100	6.6	0	0.05	4.25	0.08	169.59	4.22	0.07	0.24	0.12	0.83	0.67
	200	6.06	0	0.3	4.91	0	378.16	4.23	0	0.49	0.6	0.55	3.34
Average		6.75	0	0.1	3.64	0.16	180.88	4.16	0.12	0.24	0.35	0.67	1.09
7	25	5.59	0	0.02	1.46	0.5	59.61	2.98	0.2	0.05	0	1	0.08
	50	8.56	0	0.02	4.21	0.33	118.66	2.07	0.27	0.1	0	1	0.19
	100	4.25	0	0.05	2.59	0.1	173.34	2.67	0.09	0.14	0	1	0.7
	200	3.36	0	0.33	2.71	0	356.98	3.2	0.03	0.35	0	1	3.15
Average		5.44	0	0.11	2.74	0.23	177.15	2.73	0.15	0.16	0	1	1.03
8	25	15.36	0	0.02	0	1	51.63	11.66	0.11	0.08	0	1	0.06
	50	18.55	0	0.08	5.2	0.3	116.02	5.73	0.3	0.1	0	1	0.12
	100	17.94	0	0.19	2.41	2.26	169.38	13.89	0.32	0.29	0.86	0.6	0.55
	200	22.54	0	0.98	2.75	0.34	349.21	17.23	0.26	0.53	0.1	0.96	2.42
Average		18.6	0	0.32	2.59	0.98	171.56	12.13	0.25	0.25	0.24	0.89	0.79
9	25	5.21	0	0.02	0.98	0.36	45.61	2.92	0.18	0.1	0.01	0.82	0.1
	50	4.14	0	0.06	2.96	0.06	118.38	3.22	0.06	0.16	0.45	0.61	0.32
	100	6.2	0	0.17	5.31	0.02	149.51	4.3	0.02	0.28	0.95	0.63	0.67
	200	7.65	0	0.63	5.67	0.06	339.26	6	0	0.8	2.16	0.41	3
Average		5.8	0	0.22	3.73	0.13	163.19	4.11	0.07	0.34	0.89	0.62	1.02
10	24	9.93	0.08	0.02	1.16	0.73	24.69	2.23	0.38	0.09	0	1	0.09
	51	7.94	0.04	0.08	6.89	0.04	115.74	3.51	0.22	0.19	0.09	0.96	0.36
	99	10.89	0.02	0.18	6.51	0.3	139.92	2.38	0.29	0.5	0.49	0.84	0.54
	201	12.21	0	0.77	8.83	0.02	359.01	2.08	0.33	0.75	0.49	0.84	1.41
Average		10.24	0.04	0.26	5.85	0.27	159.84	2.55	0.31	0.38	0.27	0.91	0. 6
Overall average		9.39	0	0.15	3.77	0.27	163.51	4.68	0.17	0.31	0.33	0.75	1.14

value $N \in \{25, 50, 100, 200\}$, t is equal to $\{60, 120, 180, 360\}$ seconds respectively.

The tables below provide the following informations: let v_L be the value produced by a lower bounding procedure L, and v_H the solution value generated by a heuristic algorithm A. Let v_L^* and v_H^* denote the best lower bound and heuristic solution value obtained, respectively. For each test instance and for each algorithm, the tables report the following information.

- %gap = the average percentage gap for all points in the Pareto approximated set. The %gap is determined by $100(v_H - v_L^*)/v_L^*$ for the upper bounds and $100(v_H^* - v_L)/v_L^*$ for the lower bounds
- r_{opt} = the ratio between the number of proved optimal solutions and the number the generated approximated solutions.
- t = the average computing time needed to obtain the best solution.

We denote by (–) the instances where no solutions were found by the algorithm.

6.3. Lower bounds

In Table 2, we present the behavior of four lower bounding procedures. Columns L_{cont} and L_{int} respectively deal with the results of the continuous relaxation and the integer linear model (15)–(20) using the same CPU time as the metaheuristics. In columns L_C, we report the results for the bounds presented in Section 4.2.

Table 3
Overall performance of metaheuristics based on the vector-packing approach.

Class	n	*EA_BinPack*				*ITS_BinPack*				*ITS*			
		%gap			r_{opt}	*%gap*			r_{opt}	*%gap*			r_{opt}
		avg	*min*	*max*		*avg*	*min*	*max*		*avg*	*min*	*max*	
	25	5.89	5.15	6.07	0	5.85	4.95	6.07	0	5.76	5.76	5.76	0
1	50	5.1	5.1	5.89	0.05	3.67	2.8	4.3	0	8.13	8.13	8.13	0
	100	3.23	2.67	3.88	0.12	2.26	1.94	2.47	0.02	5.37	5.37	5.37	0
	200	2.3	1.76	2.87	0.19	2.21	1.94	2.48	0.15	3.91	3.91	3.91	0
Average		4.13	3.67	4.68	0.09	3.5	2.91	3.83	0.04	5.79	5.79	5.79	0
	25	15.19	15.19	15.19	0	15.19	15.19	15.19	0	9.83	9.83	9.83	0
2	50	–	–	–	–	–	–	–	–	–	–	–	–
	100	4.57	0.76	16.21	0.33	5.04	2.11	16.21	0	22.28	22.28	22.28	0
	200	11.73	1.2	26.38	0.3	21.3	11.58	38.67	0	6.56	6.56	6.56	0
Average		10.5	5.72	19.26	0.21	13.84	9.63	23.36	0	12.89	12.89	12.89	0
	25	3.6	3.6	3.6	0.2	3.6	3.6	3.6	0.2	4.19	4.19	4.19	0
3	50	4.88	4.88	4.88	0	4.88	4.88	4.88	0	5.07	5.07	5.07	0
	100	1.54	0.49	2.28	0.22	1.4	0.63	2.41	0.47	0.73	0.48	0.88	0.18
	200	2.49	1.92	3.11	0.13	3.74	2.53	5.5	0	1.77	1.77	1.77	0
Average		3.13	2.72	3.47	0.14	3.41	2.91	4.1	0.17	2.94	2.88	2.98	0.05
	25	6.69	6.2	6.81	0.14	6.81	6.81	6.81	0.07	6.93	6.93	6.93	0
4	50	4.64	4.37	4.77	0	4.12	3.68	4.23	0	7.42	7.42	7.42	0
	100	3.23	2.9	3.49	0.11	2.36	2.09	2.59	0.16	5.47	5.47	5.47	0
	200	2.26	2.04	2.42	0.24	1.58	1.42	1.71	0.28	4.48	4.48	4.48	0
Average		4.21	3.88	4.37	0.12	3.72	3.5	3.84	0.13	6.08	6.08	6.08	0
	25	7.44	6.84	8.34	0.07	6.22	6.07	6.84	0.06	10.97	10.97	10.97	0
5	50	3.99	3.82	4.23	0.07	3.42	3.42	3.42	0.14	6.95	6.95	6.95	0
	100	3.1	2.96	3.23	0.25	2.21	1.93	2.45	0.02	8.81	8.81	8.81	0.02
	200	2.17	2.02	2.31	0.04	1.61	1.49	1.73	0.25	4.8	4.8	4.8	0.01
Average		4.18	3.91	4.53	0.11	3.37	3.23	3.61	0.12	7.88	7.88	7.88	0.01
	25	14.03	13.72	14.66	0	3.35	2.87	3.91	0	7.78	7.78	7.78	0.14
6	50	3.91	2.42	5.11	0.11	3.54	3.18	3.99	0.11	5.69	5.66	5.7	0.11
	100	1.98	1.02	3.23	0.45	6.78	4.81	7.94	0.43	8.51	8.51	8.51	0.04
	200	2.69	1.82	3.86	0.51	7.92	6.18	9.27	0.24	10.21	10.21	10.21	0.03
Average		5.65	4.75	6.72	0.27	5.4	4.26	6.28	0.2	8.05	8.04	8.05	0.08
	25	5.02	4.51	5.78	0.17	5.05	5.05	5.05	0.14	5.78	5.78	5.78	0
7	50	1.66	1.31	2.47	0.33	1.51	1.51	1.51	0.33	1.51	1.51	1.51	0.33
	100	1.45	0.91	1.86	0.52	1.07	1.07	1.07	0.52	1.62	1.62	1.62	0.23
	200	2.5	2.3	2.8	0.19	0.77	0.77	0.77	0.44	1.11	1.11	1.11	0.3
Average		2.66	2.26	3.23	0.3	2.1	2.1	2.1	0.36	2.51	2.51	2.51	0.22
	25	6.86	6.86	6.86	0.27	6.86	6.86	6.86	0.27	9.98	9.98	9.98	0
8	50	6.77	6.77	6.77	0.25	6.49	5.39	6.77	0.25	11.17	11.17	11.17	0
	100	2.68	2.6	2.91	0.63	3.99	3.03	6.06	0.47	11.2	11.2	11.2	0.05
	200	2.27	2.01	2.64	0.56	4.17	2.69	5.69	0.48	13.91	13.91	13.91	0.03
Average		4.65	4.56	4.8	0.43	5.38	4.49	6.35	0.37	11.57	11.57	11.57	0.02
	25	4.89	3.83	5.52	0	4.92	3.38	5.52	0	5.09	5.09	5.09	0
9	50	4.15	3.59	4.34	0.06	3.44	2.16	3.88	0	7.79	7.79	7.79	0
	100	3.27	2.48	3.91	0.07	2.42	2.09	2.87	0.02	5.65	5.65	5.65	0
	200	3.03	2.47	3.58	0.19	3.19	2.62	3.62	0.15	3.31	3.31	3.31	0
Average		3.84	3.09	4.34	0.08	3.49	2.56	3.97	0.04	5.46	5.46	5.46	0
	24	3.4	3.4	3.4	0.17	3.51	3.2	3.82	0.15	4.37	4.37	4.37	0.17
10	51	2.07	1.58	2.33	0.58	1.52	1.34	1.77	0.58	4.11	4.11	4.11	0.48
	99	2.32	1.78	2.66	0.56	2.19	2.03	2.24	0.67	2.63	2.63	2.63	0.56
	201	2.48	2.09	2.8	0.5	3	2.62	3.22	0.5	1.48	1.48	1.48	0.4
Average		2.57	2.21	2.8	0.45	2.56	2.3	2.76	0.48	3.15	3.15	3.15	0.4
Overall average		4.47	3.66	5.6	0.22	4.53	3.7	5.73	0.19	6.57	6.57	6.58	0.07

The L_{CG} columns contain the results related to the vector packing based lower bound (Caprara & Toth, 2001).

The model (15)–(20) leads to weak results. It can be explained by the bad quality of its linear relaxation (9.39% gap in average, up to 30.03% for Class 2 with $n = 50$) and its large number of variables. The integer model is able to solve to optimality 27% of the instances, but fails to solve a large number of instances of size 25. The maximal remaining average gap for L_{int} reaches 11.11% for Class 3 with $n = 200$.

The fast lower bound L_C is slightly worse than L_{int} in term of overall average gap but is able to find better average gaps for subsets of all classes, except for class 8, where the gap of L_C is the worst (up to 17.23% with $n = 200$). This bound is always better than the linear relaxation L_{cont} for comparable computing times. The time required for L_C was at most one second in all cases, which is far less than the time given to the integer model.

The column generation L_{CG} provides the best lower bound (0.33% gap in average). It solved to optimality all instances in Class 7 and produced near optimal results for instances in Class 2 (0.01% of average gap and 95% of proved optimal solutions). The average gap for L_{CG} is never larger than 2.16% (Class 9, $n = 200$). The computing time of this bound remains reasonable (1, 14 s in average) with larger computing times for Classes 4 and 5 (up to 13, 41 s), where many items can be packed into a bin (and thus the pricing subproblems are more time-consuming).

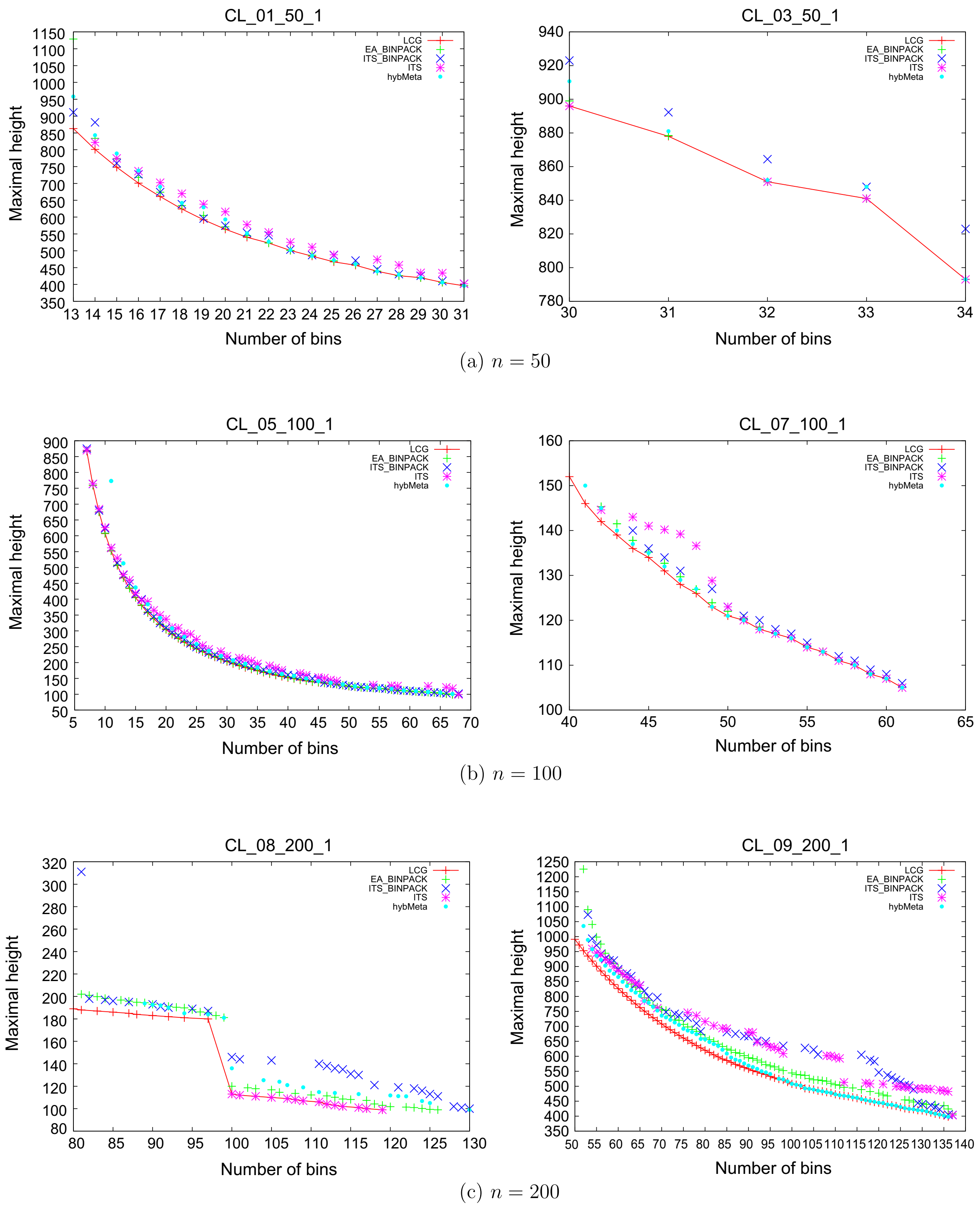

Fig. 4. Pareto approximation sets comparison for a randomly chosen instances with $n \in \{50, 100, 200\}$.

It transpires from our experiments that the best combination of bounds is to use L_C, which is always fast, before using L_{CG}.

6.4. Upper bounds

In this section, we compare the average overall performance of the proposed algorithms for both 2VPP and MHPP based approaches. For all the presented metaheuristics, we performed five independent runs. For each instance and each algorithm, we report the average, minimum and maximum percentage gap (*%gap*) in columns *avg*, *min* and *max*. Concerning the parameter settings, for the ITS algorithms, each time the construction phase is called, a tabu search algorithm is performed and stopped when the best solution cannot be improved within a given number of iterations equal to $\frac{n}{2}$. For the GA, the initial population size is fixed to 100 individuals and the crossover and mutation probabilities are set to 0.8 and 0.5 respectively.

Table 4
Overall performance of metaheuristics based on the min-height approach.

Class	*n*	*EA_BinPack*				*ITS_BinPack*				*ITS*				*hybMeta*			
		%gap			r_{opt}	%gap			r_{opt}	%gap			r_{opt}	%gap			r_{opt}
		avg	*min*	*max*		*avg*	*min*	*max*		*avg*	*min*	*max*		*avg*	*min*	*max*	
1	25	0.74	0.4	1.27	0.58	6.98	2.12	15.66	0	1.12	0.96	1.35	0.09	0.85	0.77	0.96	0.27
	50	2.68	2.26	3.18	0.42	2.59	0.67	6.17	0.05	5.79	4.81	6.6	0	2.71	2.68	2.75	0.21
	100	3.06	2.2	3.87	0.16	1.72	1.72	1.72	0.02	14.35	13.73	14.62	0.03	2.74	2.69	2.79	0.4
	200	5.19	3.81	6.69	0.01	18.17	18.17	18.17	0	7.1	7.07	7.12	0.07	3	2.34	2.7	0.35
Average		2.92	2.17	3.75	0.29	7.37	5.67	10.43	0.02	7.09	6.64	7.42	0.05	2.33	2.12	2.3	0.31
2	25	0	0	0	1	9.98	0.94	23.1	0	0	0	0	1	0.14	0	0.31	1
	50	0	0	0	1	19.23	7.39	35.02	0	0	0	0	1	0.28	0	0.81	1
	100	6.13	2.79	10.59	0	12.29	12.29	12.29	0	–	–	–	–	9.31	8.76	10.23	0
	200	24.26	16.87	32.84	0	33.19	33.19	33.19	0	–	–	–	–	9.53	7.31	13.35	0
Average		7.6	4.92	10.86	0.5	18.67	13.45	25.9	0	0	0	0	1	4.82	4.02	6.18	0.5
3	25	0	0	0	1	8.02	2.36	15.03	0	1.49	1.49	1.49	0.5	1.25	1.08	1.54	0.6
	50	0.08	0	0.36	1	2.16	0.54	4.69	0	0	0	0	1	0.59	0.32	0.75	0.2
	100	4.49	2.49	6.95	0.11	5.17	5.17	5.17	0	3.62	3.07	3.99	0	2.39	1.73	3.23	0.26
	200	4.68	3.06	6.95	0.05	11.18	11.18	11.18	0	8.6	5.92	10.88	0	2.07	1.3	3.07	0.05
Average		2.31	1.39	3.57	0.54	6.63	4.81	9.02	0	3.43	2.62	4.09	0.38	1.58	1.11	2.15	0.28
	25	0.41	0.21	0.71	0.53	6.34	1.55	13.46	0	0.98	0.84	1.11	0.33	0.48	0.4	0.65	0.58
4	50	0.74	0.44	1.11	0.4	1.99	0.55	4.18	0	5.26	4.43	6.08	0	1.51	1.51	1.51	0.16
	100	1.18	0.82	1.65	0.29	1.64	1.64	1.64	0	10.78	6.94	12.87	0.16	4.74	4.71	4.76	0.02
	200	1.78	1.35	2.24	0.04	14.45	14.45	14.45	0	6.44	6.41	6.47	0.07	3.6	3.6	3.6	0.01
Average		1.03	0.71	1.43	0.32	6.11	4.55	8.43	0	5.87	4.66	6.63	0.14	2.58	2.56	2.63	0.19
	25	0.32	0.2	0.57	0.69	5.71	1.26	12.48	0.13	1.13	1.03	1.27	0.38	0.71	0.69	0.8	0.42
5	50	0.67	0.43	0.99	0.39	1.86	0.85	3.86	0.04	4.92	4.13	5.63	0.07	1.1	1.1	1.1	0.25
	100	0.82	0.37	1.47	0.56	3.04	2.08	4.6	0	11.4	10.93	11.68	0.11	5.71	5.71	5.71	0.05
	200	1.58	1.17	2.05	0.09	12.39	12.39	12.39	0	5.91	5.91	5.92	0.22	1.34	1.34	1.34	0.15
Average		0.85	0.54	1.27	0.43	5.75	4.15	8.33	0.04	5.84	5.5	6.13	0.2	2.22	2.21	2.24	0.22
	25	0.25	0.1	0.81	0.86	9.2	2.2	17.2	0.14	0.97	0.97	0.97	0.33	2.61	2.19	3.28	0.63
6	50	2.06	1.09	3.4	0.45	2.92	1.31	5.12	0.09	4.18	3.18	5.08	0.18	7.57	6.38	8.58	0.2
	100	2.06	1.07	3.33	0.56	4.59	4.59	4.59	0	5.13	4.96	5.28	0.11	2.56	2.55	2.59	0.05
	200	6.24	4.64	7.83	0.06	24.7	24.7	24.7	0	2.87	2.79	2.91	0.13	2.77	2.77	2.77	0.06
Average		2.65	1.73	3.84	0.48	10.35	8.2	12.9	0.06	3.29	2.98	3.56	0.19	3.88	3.47	4.31	0.24
	25	0.03	0	0.13	1	6.41	1.17	14.51	0.33	0.17	0.17	0.17	0.83	1.6	1.3	1.91	0.71
7	50	0.44	0.43	0.48	0.8	1.96	1.13	4.34	0	1.16	0.69	1.72	0.53	0.45	0.44	0.48	0.8
	100	0.6	0.29	0.96	0.67	1.6	1.6	1.6	0	2.14	1.86	2.31	0.6	0.43	0.43	0.43	0.64
	200	3.14	2.58	3.84	0.03	13.21	13.21	13.21	0	5.41	5.11	5.59	0.29	1.78	1.78	1.78	0.42
Average		1.05	0.83	1.35	0.63	5.8	4.28	8.42	0.08	2.22	1.96	2.45	0.56	1.07	0.99	1.15	0.64
	25	0.02	0	0.06	1	6.64	2.07	13.67	0.22	0	0	0	1	1.77	0.92	3.22	0.78
8	50	0.36	0.19	0.66	0.79	4.56	1.51	9.66	0	0.35	0	0.61	1	3.43	2.74	4.54	0.17
	100	1.04	0.63	1.62	0.68	14.68	13.52	15.14	0	4.23	4.23	4.23	0.63	8.76	6.88	9.66	0.14
	200	3.97	2.95	5	0.26	16.16	16.16	16.16	0	3.8	3.8	3.8	0.63	9.79	9.13	10.55	0.1
Average		1.35	0.94	1.84	0.68	10.51	8.32	13.66	0.06	2.1	2.01	2.16	0.82	5.94	4.92	6.99	0.3
	25	0.48	0.25	0.92	0.64	5.7	1.18	11.52	0.18	1.18	1.01	1.34	0.18	0.97	0.89	1.07	0.45
9	50	0.74	0.38	1.14	0.56	5.72	2.85	10.89	0	5.59	4.8	6.38	0	2.64	2.56	2.7	0.22
	100	2.75	2.27	3.34	0.33	17.27	15.19	19.46	0	14.01	13.32	14.3	0.03	2.82	2.73	2.9	0.4
	200	7.59	6.34	8.99	0	17.95	17.95	17.95	0	16.22	16.22	16.22	0	2.27	2.25	2.29	0.34
Average		2.89	2.31	3.6	0.38	11.57	9.2	14.87	0.05	9.25	8.84	9.56	0.05	2.18	2.11	2.24	0.35
	24	0.09	0	0.53	0.83	8.97	1.89	17.36	0.17	0	0	0	1	0.37	0.11	0.55	0.92
10	51	1.06	0.68	1.57	0.73	5.58	3.04	10.34	0.04	3.01	2.17	3.82	0.26	1.65	1.54	1.8	0.72
	99	2.17	1.52	3	0.35	12.47	10.51	13.87	0	8.65	8.14	9.04	0.04	2.63	2.52	2.98	0.48
	201	4.97	3.92	6.24	0.05	13.6	13.6	13.6	0	10.48	10.37	10.57	0.04	2.15	2.15	2.15	0.43
Average		2.07	1.53	2.84	0.49	10.17	7.28	13.81	0.05	5.54	5.17	5.86	0.34	1.7	1.58	1.87	0.64
Overall average		2.47	1.71	3.43	0.47	9.3	7	12.58	0.04	4.7	4.25	5.04	0.34	2.83	2.51	3.2	0.37

6.4.1. Results for the vector-packing based approach

In Table 3, we report the results obtained by a population based and a single solution based metaheuristics using an indirect encoding (*EA_BinPack* and *ITS_BinPack*) and an iterated tabu search using a direct encoding *ITS*.

The two methods using the indirect encoding are better than the ITS with direct encoding (respectively 4.47% and 4.53% gap against 6.57% gap for ITS). This difference is reduced when one considers the worst run, but becomes larger when one considers the best run. This hints that the indirect encoding is more adapted in average for this vector-packing based approach.

However, note that for Class 2 with $n = 200$, *ITS* is clearly better than the two other methods. Furthermore, we observed that the *ITS* is also better for small size instances of Class 1 and 3. The weakness of our direct encoding ITS is that it tends to converge toward the same local optimum at each run.

Between the two indirect encoding based methods, the evolutionary algorithm is slightly better than the iterated tabu search for both average gap and number of optimal solutions. However this difference is small. The *ITS_BinPack* algorithm can outperform the *EA_BinPack* for some instances and vice versa.

6.4.2. Results for the min-height based approach

The results obtained by the MHPP based approaches are comparable with those obtained by the 2DVPP approaches, with some notable differences.

Table 5
Comparison of min-height and vector-packing based heuristics.

Class	n	*MB*			*LL*			*DotProduct*		
		%gap	opt	t	%gap	opt	t	%gap	opt	t
1	25	28.99	0	0.23	5.54	0.08	0.03	3.01	0.08	0.04
	50	36.76	0	0.44	6.5	0	0.036	4.3	0	0.16
	100	40.06	0.03	1.2	5.83	0.22	0.12	4.12	0.02	0.99
	200	40.31	0	3.2	6.17	0.07	0.04	3.45	0	4.23
Average		36.53	0.01	1.27	6.01	0.09	0.06	3.72	0.03	1.36
2	25	15.43	0.33	0.09	0.47	0.5	0.01	0	1	0.02
	50	69.03	0	0.16	–	–	0.02	–	–	0.02
	100	–	–	0.1	–	–	0.02	0.72	0	0.34
	200	–	–	0.3	–	–	0.04	8.78	0	2.39
Average		21.12	0.08	0.16	0.12	0.13	0.02	2.38	0.25	0.69
3	25	28.51	0	0.11	8.13	0	0.01	0.98	0.8	0.02
	50	24.54	0	0.26	8.63	0	0.02	0.55	0.2	0.07
	100	26.02	0	0.53	7.8	0	0.07	2.37	0.17	0.6
	200	32.32	0	1.32	15.3	0	0.17	2.65	0.03	2.58
Average		28.46	0	0.3	9.97	0	0.07	1.64	0.3	0.82
4	25	24.72	0.07	0.25	4.87	0.07	0.02	2.8	0.07	0.05
	50	26.13	0	0.69	5.87	0	0.048	4.14	0	0.16
	100	31.64	0.02	1.66	5.32	0	0.18	4.11	0.02	0.92
	200	34.35	0	3.93	5.56	0.07	0.36	3.77	0	4.82
Average		29.21	0.02	0.47	5.41	0.04	0.15	3.71	0.02	1.49
5	25	22.96	0.13	0.26	4.41	0.06	0.03	3.52	0.06	0.04
	50	26.53	0	0.68	5.19	0	0.06	3.84	0	0.16
	100	29.82	0.02	1.75	4.85	0.02	0.12	4.11	0.02	0.83
	200	33.03	0	4.48	5.77	0	0.47	4.61	0	4.54
Average		28.09	0.04	0.47	5.06	0.02	0.17	4.02	0.02	1.39
6	25	36.23	0.14	0.13	3.1	0.17	0.01	2.01	0.29	0.01
	50	40.45	0.11	0.29	5.88	0.22	0.03	3.26	0.11	0.04
	100	50.73	0.05	0.69	2.46	0.18	0.09	6.65	0.04	0.18
	200	53.08	0	1.76	5.17	0	0.2	2.89	0	0.88
Average		45.12	0.08	0.37	4.15	0.14	0.08	3.7	0.11	0.28
7	25	19.1	0.17	0.12	1.64	0.33	0.02	0.26	0.67	0.01
	50	13.19	0.47	0.37	1.29	0.79	0.04	0.54	0.73	0.04
	100	7.32	0.55	0.54	4.06	0.63	0.06	0.64	0.36	0.14
	200	6.54	0.29	1.13	6.22	0.29	0.09	1.02	0.3	0.54
Average		11.54	0.37	0.34	3.3	0.51	0.05	0.62	0.52	0.18
8	25	27.86	0.13	0.14	0	1	0.008	4.08	0.18	0.02
	50	33.56	0.1	0.39	0	1	0.03	9.28	0.09	0.1
	100	40.23	0	0.78	0	1	0.1	4.21	0.05	0.5
	200	43.16	0.03	2.13	0.55	0.71	0.23	27.71	0.03	2.29
Average		36.2	0.07	0.44	0.14	0.93	0.09	11.32	0.09	0.73
9	25	29.44	0.09	0.23	5.02	0.09	0.03	2.43	0.09	0.04
	50	41.36	0	0.47	6.47	0	0.04	4.78	0	0.15
	100	39.58	0.02	1.23	5.83	0.23	0.14	4.08	0.02	0.92
	200	40.24	0	3.03	6.18	0.07	0.64	3.68	0	5.2
Average		37.66	0.03	0.35	5.88	0.1	0.21	4.18	0.03	1.58
10	24	9.18	0.09	0.04	1.16	0.82	0.03	1.73	0.5	0.03
	51	27.75	0.04	0.11	2.79	0.83	0.07	1.98	0.61	0.1
	99	27.97	0.02	0.23	3.75	0.54	0.13	2.03	0.67	0.32
	201	31.74	0.01	0.62	3.85	0.4	0.24	2.62	0.39	1.77
Average		24.16	0.04	0.25	2.89	0.65	0.12	2.09	0.54	0.56
Overall average		30.44	0.08	0.46	4.45	0.22	0.1	3.92	0.15	0.95

Once again, the best method in average is *EA_BinPack*. However, it transpires that the *ITS_BinPack* is the worst in average, with poor performances for all classes.

The difference between *EA_BinPack* and *ITS* is smaller in this case. This means that applying the direct encoding scheme can be a good strategy in some cases, although *EA_BinPack* remains better in average for all classes, with a greater difference for large size instances.

Class 2 remains difficult, even with the min-height approach. The maximal average gap achieved for both *EA_BinPack* and *ITS_BinPack* (32.84% and 33.19% respectively) are obtained for Class 2 with $n = 200$. In this case, *hybMeta* performs better (13.35% avg. gap).

The minimum average gap of *hybMeta* was low for almost all instances especially for small value of n. This is because of the combination of heuristic and exact methods in the *hybMeta* algorithm. When the value of n is small, the exact method can solve an instance constructed with a large subset of the items. This method is stable: the worst gap is never far from the best.

6.5. Pareto approximation sets comparison

We plot in Fig. 4 the Pareto approximation sets generated from the different metaheuristics (*EA_BinPack*, *ITS_BinPack*, *ITS* and *hybMeta*) for a randomly chosen instances with $n \in \{50, 100, 200\}$. We

Table 6
Studying the impact of the variation of the resources sizes on the computational time needed for min-height and vector-packing based heuristics.

Class	n	*min-bin*			*LL*			*DotProduct*		
		10	100	1000	10	100	1000	10	100	1000
1	50	0.04	0.056	0.056	0.048	0.032	0.036	0.79	1.92	10.04
	100	0.12	0.1	0.12	0.1	0.108	0.108	5.34	13.40	43.01
	200	0.196	0.196	0.164	0.22	0.188	0.268	30.778	71.38	211.189
3	50	0.016	0.032	0.048	0.012	0.028	0.024	0.928	1.484	9.549
	100	0.044	0.032	0.064	0.072	0.048	0.044	5.80	10.885	30.218
	200	0.152	0.108	0.124	0.084	0.104	0.088	44.235	132.868	170.855
5	50	0.06	0.064	0.072	0.06	0.072	0.072	0.82	1.916	8.157
	100	0.116	0.116	0.176	0.164	0.152	0.12	5.484	12.329	30.478
	200	0.264	0.24	0.22	0.28	0.28	0.26	24.802	52.107	146.433
7	50	0.02	0.012	0.036	0.024	0.048	0.048	0.268	0.44	1.172
	100	0.028	0.06	0.064	0.04	0.06	0.056	0.86	1.52	3.104
	200	0.068	0.068	0.092	0.072	0.076	0.068	4.22	8.737	11.941

compare these approximated Pareto fronts against the column generation lower bound L_{CG}, which is our best lower bound.

The selected test instances are not equally difficult to solve, which affects the number of the generated potentially efficient solutions of each algorithm. This number varies from 5 to 19 (for instances of 50 items), 42 to 62 (for instances of 100 items) and 75 to 81 (for instances of 200 items).

The examples in Fig. 4 illustrate the numerical results. The algorithms with indirect encoding (*EA_BinPack* and *ITS_BinPack*) were generally equivalent by producing near optimal solutions and outperform the *ITS* in most cases. However, applying the algorithm with direct encoding (*ITS*) can be helpful in some cases especially when the number of bins is small. The hybrid metaheuristic *hybMeta* also performs well for hard instances where the exact method generates a solution with a large subset of the items.

6.6. Comparison of the two iterative approaches

From Tables 3 and 4, it transpires that the best average gap are obtained by the min-height approaches. Since the number and the structure of the subproblems to solve is not the same, we now provide a comparison between the two approaches, using heuristics only.

In Table 5, we compare two MHPP heuristics (*MB* and *LL*), and the 2VPP heuristic (*DotProduct*) in terms of solutions and computing time.

Heuristic *MB* performs uniformly bad for all instances (with the notable exception of class 2, $n = 50$ where it finds a feasible solution whereas LL does not). Since its computing time is larger than the computing time of LL, we will focus on this latter heuristic for the following comparisons.

The *DotProduct* heuristic outperforms *LL* on average *%gap*. However, *LL* is better than *DotProduct* for the average number of proved optimal solutions r_{opt}. Moreover, *DotProduct* is often slower than the min-height based heuristics because of the large number of problems to solve.

Since the vector packing approach needs to be run a pseudo-polynomial number of times, we study in Table 6 the variation of the average computational time when the size of the bin grows. For each instance, we successively multiply the resources sizes by 10, 100 and 1000. The results generated by the *DotProduct* algorithm show that solving the problem using the vector-packing based approach consumes much more time than heuristics from the min-height based approach (*MB* and *LL*). This is due to the pseudo-polynomial number of 2-DVPPs to be solved. Therefore, for large sizes of bin, this approach should not be used.

7. Conclusion

Throughout this paper, we studied a bi-objective version of the 2-DVPP. To tackle the problem, we presented two resolution approaches that iteratively solve mono-objective problems. Our computational experiments show that the min-height packing approach is more relevant to solve the bi-objective problem when the size of the bin becomes large, since the computational effort needed by the vector-packing approach becomes too large in this case. However, for medium sizes of bin, this latter approach leads to good results.

References

Alves, C., de Carvalho, J. V., Clautiaux, F., & Rietz, J. (2013). Multidimensional dual-feasible functions and fast lower bounds for the vector packing problem. *European Journal of Operational Research* (submitted for publication).

Caprara, A., & Toth, P. (2001). Lower bounds and algorithms for the 2-dimensional vector packing problem. *Discrete Applied Mathematics, 111*(3), 231–262.

Chang, S. Y., Hwang, H.-C., & Park, S. (2005). A two-dimensional vector packing model for the efficient use of coil cassettes. *Computers and Operations Research, 32*, 2051–2058.

Clautiaux, F., Alves, C., & Valério de Carvalho, J. (2010). A survey of dual-feasible and superadditive functions. *Annals of Operations Research, 179*, 317–342.

Coffman, J. E. G., Garey, M. R., & Johnson, D. S. (1978). An application of bin-packing to multiprocessor scheduling. *INFORMS Journal on Computing, 7*(1), 1–17.

Dell'amico, M., Iori, M., & Martello, S. (2004). Heuristic algorithms and scatter search for the cardinality constrained p||Cmax problem. *Journal of Heuristics, 10*, 169–204.

Dell'Amico, M., Iori, M., Martello, S., & Monaci, M. (2006). Lower bounds and heuristic algorithms for the ki-partitioning problem. *European Journal of Operational Research, 171*(3), 725–742.

Dell'Amico, M., & Martello, S. (1995). Optimal scheduling of tasks on identical parallel processors. *INFORMS Journal on Computing, 7*(2), 191–200.

Dell'Amico, M., & Martello, S. (2001). Bounds for the cardinality constrained p||Cmax problem. *Journal of Scheduling, 4*, 123–138.

Eilon, S., & Christofides, N. (1971). The loading problem. *Management Science, 17*, 259–267.

Falkenauer, E. (1996). A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics, 2*, 5–30.

Garey, M. R., Graham, R. L., Johnson, D. S., & Andrew (1976). Resource constrained scheduling as generalized bin packing. *Journal of Combinatorial Theory, 21*, 257–298.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-completeness (Series of books in the mathematical sciences)*. W.H. Freeman.

Geiger, M. J. (2007). Bin packing under multiple objectives – A heuristic approximation approach. In *The fourth international conference on evolutionary multi-criterion optimization: Late breaking papers, Matsushima, Japan* (pp. 53–56).

Glover, F., & Laguna, M. (1999). *TABU search*. Kluwer.

Graham, R. L. (1966). Bounds for certain multiprocessing anomalies. *Bell System Technical Journal, 45*, 1563–1581.

Haimes, Y., Lasdon, L., & Wismer, D. (1971). On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE Transactions on Systems, Man, and Cybernetics, 1*(3), 296–297.

Ishibuchi, H., & Murata, T. (1998). Multi-objective genetic local search algorithm and its application to flowshop scheduling. *IEEE Transactions on Systems, Man and Cybernetics, 28*(3), 392–403.

Khanafer, A., Clautiaux, F., Hanafi, S., & Talbi, E.-G. (2012). The min-conflict packing problem. *Computers and Operations Research, 39*(9), 2122–2132.

Khanafer, A., Clautiaux, F., & Talbi, E.-G. (2012). Tree-decomposition based heuristics for the two-dimensional bin packing problem with conflicts. *Computers and Operations Research, 39*(1), 54–63.

Lee, S., Panigrahy, R., Prabhakaran, V., Ramasubramanian, V., Talwar, K., Uyeda, L., et al. (2011). *Validating heuristics for virtual machines consolidation*. TechReport MSR-TR-2011-9, Microsoft Research.

Liu, D., Tan, K., Huang, S., Goh, C., & Ho, W. (2008). On solving multiobjective bin packing problems using evolutionary particle swarm optimization. *European Journal of Operational Research, 190*(2), 357–382.

Sathe, M., Schenk, O., & Burkhart, H. (2009). Solving bi-objective many-constraint bin packing problems in automobile sheet metal forming processes. In *EMO '09: Proceedings of the 5th international conference on evolutionary multi-criterion optimization* (pp. 246–260).

Spieksma, F. C. R. (1994). A branch-and-bound algorithm for the two-dimensional vector packing problem. *Computers and Operations Research, 21*(1), 19–25.

Talbi, E.-G. (2009). *Metaheuristics: From design to implementation*. John Wiley & Sons.