Simple but effective heuristics for the 2-Constraint Bin Packing Problem

R. Aringhieri, D. Duma, A. Grosso and P. Hosteins

Simple but effective heuristics for the 2-constraint bin packing problem

Journal of Heuristics, 2017. Available online 15 March 2017.

DOI: 10.1007/s10732-017-9326-0

Journal of Heuristics manuscript No.
(will be inserted by the editor)

Simple but effective heuristics for the 2-Constraint Bin Packing Problem

Roberto Aringhieri · Davide Duma · Andrea Grosso · Pierre Hosteins

Received: date / Accepted: date

Abstract The 2-constraint bin packing problem consists in packing a given number of items, each one characterised by two different but not related dimensions, into the minimum number of bins in such a way to do not exceed the capacity of the bins in either dimension. The development of the heuristics for this problem is challenged by the need of a proper definition of the criterion for evaluating the feasibility of the two capacity constraints on the two different dimensions. In this paper, we propose a computational evaluation of several criteria, and two simple but effective algorithms – a greedy and neighbourhood search algorithms – for solving the 2-constraint bin packing problem. An extensive computational analysis supports our main claim.

Keywords 2-constraint bin packing problem · criteria · heuristics

Roberto Aringhieri
Dipartimento di Informatica, Università degli Studi di Torino, Turin, Italy
Tel.: +39-011-6706755
Fax: +39-011-751603
E-mail: roberto.aringhieri@unito.it

Davide Duma
Dipartimento di Informatica, Università degli Studi di Torino, Turin, Italy
Tel.: +39-011-6706744
Fax: +39-011-751603
E-mail: davide.duma@unito.it

Andrea Grosso
Dipartimento di Informatica, Università degli Studi di Torino, Turin, Italy
Tel.: +39-011-6706824
Fax: +39-011-751603
E-mail: andrea.grosso@unito.it

Pierre Hosteins
Dipartimento di Informatica, Università degli Studi di Torino, Turin, Italy
Tel.: +39-011-6706744
Fax: +39-011-751603
E-mail: hosteins@di.unito.it

1 Introduction

The 2-Constraint Bin Packing Problem (2CBP) is a generalisation of the classic bin packing problem in which each item $i \in I$ is characterised by two attributes or dimensions, usually called weight w_i and length ℓ_i such that $w_i \geq 0$, $\ell_i \geq 0$, and $w_i + \ell_i > 0$. The $n = |I|$ items have to be packed into the minimum number of bins in such a way that the sum of the w_i and the sum of the ℓ_i do not exceed the maximum weight W and the maximum length L of each bin, respectively. A straightforward integer linear programming formulation is the following:

$$\min z = \sum_{j \in J} y_j \tag{1a}$$

$$\sum_{j \in J} x_{ij} = 1, \qquad i \in I, \tag{1b}$$

$$\sum_{i \in I} w_i x_{ij} \leq W y_j, \qquad j \in J, \tag{1c}$$

$$\sum_{i \in I} \ell_i x_{ij} \leq L y_j, \qquad j \in J, \tag{1d}$$

$$x_{ij} \in \{0,1\},\ y_j \in \{0,1\} \qquad i \in I, j \in J, \tag{1e}$$

where: J is the set of possible bins indexed by j; $y_j = 1$ if and only if the bin j is used; $x_{ij} = 1$ if and only if the item i is packed into the bin j. Constraints (1b) guarantee that all the items are assigned to a bin while constraints (1c) and (1d) ensure that such an assignment does not exceed the bin capacities, when the bin is used. Finally, the objective function (1a) seeks to minimise the number of bins used.

Apart from the applications in loading and scheduling contexts considered by Spieksma (1994), the 2CBP has a new important application in the emerging field of the management of cloud computing platforms. Guazzone et al (2013) used 2CBP to evaluate the cost of a cloud federation for the reduction of the energy bill. Such a cost is then the pay-off of a cooperative game which models the capability of the different cloud providers to self-organise into Nash-stable federations. To find the solution of the cooperative game, the authors proposed a distributed algorithm which finds the equilibrium by means of iterated executions. The 2CBP arises also as task placement problem for packing processes onto a cluster of servers (Wilcox et al 2011) or for data placement problem in the context of multimedia storage systems (Shachnai and Tamir 2012).

The 2CBP is a particular case of the m-dimensional vector bin packing problem proposed by Garey et al (1976), and is NP-hard in the strong sense (see, e.g., Monaci and Toth (2006)). Spieksma (1994) proposed lower bounds, constructive heuristics and a branch-and-bound algorithm for the problem while Garey et al (1976) and Kellerer and Kotov (2003) presented some approximation algorithms with guaranteed performance ratio. An extensive analysis

of the 2CBP has been reported in Caprara and Toth (2001): the authors proposed and compared several lower bounds, greedy procedures and constructive heuristics, and exact algorithms based on branch-and-bound and branch-and-price. Exploiting a set-covering formulation, Monaci and Toth (2006) proposed a general heuristic algorithm for solving the 2CBP in which both the column-generation and the column-optimisation phases are heuristically performed. Recently, in their work on a single machine scheduling problem with two-dimensional vector packing constraints, Billaut et al (2015) extended their proposed algorithm also to solve 2CBP. The algorithm is a two-step approach embedding a Recovering Beam Search (Della Croce et al 2004) algorithm, which generates a good-quality initial solution in a short amount of time, and a more time consuming matheuristic algorithm.

The main contributions of our paper to the 2CBP are the following: an extended analysis of different criteria for driving both constructive and improvement algorithms, and two simple and effective algorithms, that is a greedy and a neighbourhood search heuristic.

The paper is organised as follows. The extended analysis of different criteria is reported in Section 2 in which the greedy solution framework is also presented. The neighbourhood search heuristic is discussed in Section 3. An extensive computational analysis is reported in Section 4. Conclusions are discussed in Section 5. In the remainder of the paper, the reader can refer to the books of Martello and Toth (1990) and Kellerer et al (2004) for the traditional bin packing algorithms, which are cited in the paper.

2 Selection Criteria and Greedy Solutions

When devising heuristics for the 2CBP, the first question is which selection criterion should be employed. For instance, how should the item to be added to or deleted from the current solution be selected in a greedy or constructive algorithm? An analysis of the literature reveals that a comparison among different criteria is not reported, to the best of our knowledge.

Garey et al (1976) proposed an adaptation of the First Fit Decreasing (FFD) heuristic where the items are considered in decreasing order of $\max\{w_i, \ell_i\}$, and the item with highest weight is packed in the first bin with enough residual capacity. The heuristic, called 2FFD in Caprara and Toth (2001), requires $O(n \log n)$ time. Spieksma (1994) proposed the heuristics 2FFD_μ, which considers the items in decreasing order of $\mu w_i + \ell_i$ values, where $\mu \geq 0$: each iteration consists in an execution of the 2FFD to compute a solution which is then examined to determine which bins are "well-filled" and to update μ; the items belonging to the well-filled bins will be not considered in the following iterations. The algorithm requires $O(n^2 \log n)$ time. Caprara and Toth (2001) introduced the *surrogate cost* s_i equal to

$$s_i = \lambda w_i + (1 - \lambda)\ell_i\,, \quad i \in I\,, \tag{2}$$

associated to each item $i \in I$. Exploiting such a definition, the authors considered the heuristic 2FFD$_\lambda$ as an adaptation of the heuristic of Spieksma (1994). In addition, they extended the classical Best Fit Decreasing (BFD) heuristic – which distinguishes itself from the FFD by inserting the item with highest weight into a bin in such a way that the residual capacity of the bin after the insertion is minimal, possibly 0 – to the 2BFD, in the form of 2BFD$_\mu$ and 2BFD$_\lambda$.

In the following, we will consider a *criterion* as a combination of a way to insert an item into a bin, and a function that returns a surrogate weight of the item. Hereafter, we will consider normalised weights, that is $w'_i = \frac{w_i}{W}$ and $\ell'_i = \frac{\ell_i}{L}$. Given an item $i \in I$, let us consider the following *surrogate weight functions*, that is $v_i^{\max} = \max\{w'_i, \ell'_i\}$, $v_i^{\min} = \min\{w'_i, \ell'_i\}$, $v_i^{sub} = |w'_i - \ell'_i|$, and $v_i^{avg} = \frac{1}{2}w'_i + \frac{1}{2}\ell'_i$. Note that $v_i^{avg} = s_i$ when $\lambda = \frac{1}{2}$. Let us also consider the following traditional bin packing insertion rules, that is a First Fit (FF) and a Best Fit (BF) policies.

Now let us consider the classic Bin Packing Greedy (BPG) framework, that first selects the *best* item among those not already inserted, and then try to insert it into one of the open bin(s) or into a new bin when the item cannot be packed into the already open bins. Over the BPG framework, we can obtain 7 greedy algorithms combining an insertion rule and a surrogate weight function properly. Table 1 summarises the proposed algorithms.

The 2FFD$_{\min}$ and 2FFD$_{\max}$ combine the FF rule with $v_i^{\min}$ and $v_i^{\max}$, respectively. Combining the BF rule with $v_i^{\max}$, v_i^{avg} and v_i^{sub}, we obtain respectively the 2BFD$_{\max}$, 2BFD$_{avg}$ and the 2BFD$_{sub}$ algorithms. Note that 2FFD$_{\max}$ corresponds to the greedy proposed by Garey et al (1976).

The last two algorithms are characterised by an alternative use of different surrogate weight functions. The rationale here is to mitigate the effect of the insertion of *heavy* items with the insertion of *lighter* ones. The 2FFD$_{\max\min}$ combines the FF rule with $v_i^{\max}$ and $v_i^{\min}$, alternatively. Finally, the 2BF/FFD$_{avg\min}$ alternates both the surrogate weight function and the insertion rule: after one iteration, which combines the BF rule with v_i^{avg}, follows an iteration in which the FF rule is combined with $v_i^{\min}$.

Table 1 Summary of the 7 greedy algorithms proposed.

algorithm	primary		alternate	
	insertion rule	surrogate weight	insertion rule	surrogate weight
2FFD$_{\min}$	FF	$v_i^{\min}$	–	–
2FFD$_{\max}$	FF	$v_i^{\max}$	–	–
2BFD$_{\max}$	BF	$v_i^{\max}$	–	–
2BFD$_{avg}$	BF	v_i^{avg}	–	–
2BFD$_{sub}$	BF	v_i^{sub}	–	–
2FFD$_{\max\min}$	FF	$v_i^{\max}$	FF	$v_i^{\min}$
2BF/FFD$_{avg\min}$	BF	v_i^{avg}	FF	$v_i^{\min}$

The quality of the solutions provided by applying the above greedy algorithms (*greedies* hereafter) can depend on the number of open bins where it is possible to insert the current item. To overcome this limitation for greedies based on the BF insertion rule, we exploit the information provided by a lower bounding procedure LB on the minimum number of bins required, which returns the value z_{LB}.

We thus define the 2BFD-LB framework as an adaptation of the 2BFD in which the algorithm starts with z_{LB} bins open instead of only 1. The 2BFD-DYNLB algorithm extends the 2BFD-LB as follows: at the end of each iteration, the value z_{LB} is recomputed after the insertion of a new item. If the new lower bound z_{LB} is greater than the current number of open bins z, the algorithm will open $z_{LB} - z$ new bins. Algorithm 1 depicts the pseudo-code of the 2BFD-DYNLB greedy. The pseudo-code of the 2BFD-LB greedy can be obtained from algorithm 1 dropping the lines 12–13. Hereafter, we denote with B_j the set of items belonging to the bin j.

Algorithm 1: 2BFD-DYNLB

```
   Data: I, s_w surrogate weight function.
 1 S = I;
 2 z_LB = computeBound(S);
 3 z = z_LB; B_{1,...,z} = ∅ ;                          /* open new z bins */
 4 while S ≠ ∅ do
 5     i = selectItemFrom(S, s_w);
 6     j = findBinForItem(i, B_{1,...,z}) ;    /* returns 0 when i cannot be packed */
 7     if j > 0 then
 8         addItem(i, B_j)                                  /* add i to B_j */
 9     else
10         z = z + 1; B_z = ∅; addItem(i, B_z)          /* add i to new bin */
11     S = S \ {i};
12     z_LB = updateBound(S);
13     if z_LB > z then B_{z+1,...,z_LB} = ∅; z = z_LB      /* add new bins */;
   Result: z, B_{1,...,z}.
```

Several bounds have been proposed in literature (see, e.g., Alves et al (2014)). In line with the simplicity of the methods used in the development of our algorithms, we would like to adopt a simple lower bound procedure. Following the quantitative analysis reported by Caprara and Toth (2001) regarding the quality of the lower bound and the running time required, we adopted the simple L_C lower bound, introduced by Spieksma (1994), defined as

$$L_C = \max \left\{ \left\lceil \sum_{i \in I} w'_i \right\rceil , \left\lceil \sum_{i \in I} \ell'_i \right\rceil \right\}. \tag{3}$$

The updated number of bins (see line 12) is given by the number of already open bins plus the value of the lower bound computed as follows: computes L_C considering only the set of not inserted items and subtracting the corre-

sponding residual normalised capacity of the already opened bins to each term of (3).

3 Neighbourhood Search

To the best of our knowledge, 2REF by Caprara and Toth (2001) is the only neighbourhood search proposed to solve the 2CBP. The authors used 2REF to improve the solution computed by their greedies, reported in Section 2. The exchange procedure works as follows: the item i with maximum s_i is selected from the bin of minimum overall surrogate cost; such an item is moved to another bin with minimum surrogate cost such that the item and all items already belonging to the bin can be packed according to a BF policy.

In this Section, we provide a simple and effective neighbourhood search for solving the 2CBP, which is characterised by a clever way to exploit the unfeasibility of a solution, already tested in Aringhieri and Dell'Amico (2005a,b) and Aringhieri et al (2016).

Two classical neighbourhoods have been employed by our algorithm. The *shift* neighbourhood N_1 moves one item from a bin to a different one while the *swap* neighbourhood N_2 exchanges two items belonging to two different bins. The objective of such neighbourhoods is to maximise the residual capacity of the emptiest bin according to the weights v_i^{avg} introduced in Section 2. The shift and the swap require $O(n)$ and $O(n^2)$ time for each neighbourhood exploration.

In order to avoid looping over already visited solutions, our algorithm employs two tabu lists having fixed length δ_1 and δ_2 respectively, implemented using tabu tags Gendreau et al (1994). The first tabu list forbids an item to be part of a move for the next δ_1 iterations while the second tabu list forbids an item to be packed back to the starting bin for the next δ_2 iterations. Clearly, it is required that $\delta_1 < \delta_2$. The rationale here is to block the item $i \in I$ in order to allow the search to adjust the solution after moving it; then, we prevent the item i from being packed back to its starting bin to allow the algorithm to compose an exchange in two steps, when the exchange is not allowed to be done directly. The effectiveness of this setting has already been discussed, for instance, in Aringhieri (2009) and Aringhieri et al (2015).

In order to minimise the number of bins, our algorithm first detects the *emptiest* bin, that is the bin whose surrogate weight is minimum, and then it simply moves the items belonging to that bin into the remaining bins by forcing the application of N_1. Clearly, the resulting solution can be unfeasible since one or more items can exceed the bin capacity. Feasibility can be regained applying N_2 in such a way to find a move that minimises the unfeasibility of the solution measured as follows

$$V = \sum_{j \in J} \max \left\{ 0, -W + \sum_{i \in B_j} w_i \right\} + \max \left\{ 0, -L + \sum_{i \in B_j} \ell_i \right\}. \qquad (4)$$

We remark that each move having $V = 0$ individuates a swap that restores the feasibility of the incumbent solution.

Each step of our algorithm is composed of two phases. In the first phase, the algorithm finds the best feasible move (best improvement exploration of N_1 and N_2), not necessarily better than the incumbent solution, that maximises the residual capacity of the emptiest bin. After M_1 not improving moves, the second phase starts by (i) emptying the emptiest bin forcing the application of N_1, then (ii) trying to restore the feasibility by applying N_2 in such a way to find a move that minimises V. The second phase will end as soon as $V = 0$ or after M_2 non improving moves. If it restores feasibility, the algorithm comes back to the first phase, otherwise it applies a restart. We remark that the restart works as a diversification strategy while the second phase acts as an intensification strategy. The algorithm will finish after R restarts. Algorithm 2 depicts the pseudo-code of our algorithm NSwR-2CBP. The symbol Σ represents a solution to the problem, i.e. a set of bins $\Sigma = \{B_1, \ldots, B_z\}$, while $z(\Sigma)$ is the number of bins in solution Σ.

Algorithm 2: NSwR-2CBP

Data: I, s_w surrogate weight function, R, M_1, M_2.
$r = 0$; solutionIsImproving = true;
while $r < R$ **do**
  Σ = computeStartingSolution(I, s_w); ; // $\Sigma = \{B_1, \ldots, B_z\}$
  repeat
    /* phase 1: maximise the residual capacity of the emptiest bin */
    $m_1 = 0$; solutionIsImproving = false;
    while $m_1 < M_1$ **do**
      Σ' = getBestSolFrom(N_1, N_2, s_w);
      if *isBetter*(Σ') **then** $\Sigma = \Sigma'$; $m_1 = 0$;
      else $m_1 = m_1 + 1$;
    /* phase 2: reducing the number of bins handling unfeasibility */
    Σ' = emptyTheEmptiestBin(Σ, s_w);
    V = computeUnfeasibility(Σ');
    if $V < 0$ **then**
      $m_2 = 0$;
      while $m_2 < M_2$ ***and*** $V > 0$ **do**
        Σ'' = getBestSolFrom(N_2, s_w);
        V = computeUnfeasibility(Σ'');
        if *isBetter*(Σ'') **then** $m_2 = 0$;
        else $m_2 = m_2 + 1$;
      if $V = 0$ **then** solutionIsImproving = true; $\Sigma = \Sigma''$;
    else solutionIsImproving = true; $\Sigma = \Sigma'$;
  until *solutionIsImproving*;
  /* Store the solution obtained if it improves the current best */
  storeBestSolution(Σ, r);
  $r = r + 1$;
Result: Σ =extractBestSolution().

At each restart, the starting solution is computed with 2BFD-DYNLB, which adopts the v_i^{avg} weights. In order to generate different starting solutions, we propose a simple randomisation of the item insertion policy. Instead of inserting the current item in the bin that minimises its residual capacity after the item insertion, the algorithm randomly selects one of the possible bins whose residual capacity after the insertion differs from the best one by a value of at most Δ.

4 Computational Analysis

In this Section, we report our computational tests on the algorithms proposed in Sections 2 and 3. Our codes have been implemented in `C++` and ran on a Linux 64 bit laptop with a 2.4 GHz Intel Core i3 processor and 4 GB of main memory.

We performed our computational analysis on the instances whose characteristics are reported in Table 2, which replicates those reported in Caprara and Toth (2001). Such instances have been introduced by Spieksma (1994) (instances belonging to the classes 1–3) and Caprara and Toth (2001) (instances belonging to the classes 4–10). Instances are available online at `http://or.dei.unibo.it/library`. For a complete description of the instance characteristics, we refer to that reported in Caprara and Toth (2001).

Table 2 Test instances: $u.d.[a,b]$ means uniformly distributed in the interval $[a,b]$.

class	W	L	w_i	ℓ_i	n
1	1000	1000	$u.d.[100,400]$	$u.d.[100,400]$	$\{25,50,100,200\}$
2	1000	1000	$u.d.[1,1000]$	$u.d.[1,1000]$	$\{25,50,100,200\}$
3	1000	1000	$u.d.[200,800]$	$u.d.[200,800]$	$\{25,50,100,200\}$
4	1000	1000	$u.d.[50,200]$	$u.d.[50,200]$	$\{25,50,100,200\}$
5	1000	1000	$u.d.[25,100]$	$u.d.[25,100]$	$\{25,50,100,200\}$
6	150	150	$u.d.[20,100]$	$u.d.[20,100]$	$\{25,50,100,200\}$
7	150	150	$u.d.[20,100]$	$u.d.[wj-10,wj+10]$	$\{25,50,100,200\}$
8	150	150	$u.d.[20,100]$	$u.d.[110-wj,130-wj]$	$\{25,50,100,200\}$
9	see text	see text	$u.d.[100,400]$	$u.d.[100,400]$	$\{25,50,100,200\}$
10	100	100	see text	see text	$\{24,51,99,201\}$

Instances belonging to the class 9 are similar to those in class 1: actually, items are generated following the same distribution, while the capacities are computed as

$$W = \frac{\sum_{i \in I} w_i}{M} \text{ and } \frac{\sum_{i \in I} \ell_i}{M}$$

where M is computed as

$$M = \max \left\{ \frac{\sum_{i \in I} w_i}{1000}, \frac{\sum_{i \in I} \ell_i}{1000} \right\}.$$

Instances belonging to the class 10 are generated in such a way that each bin can be filled up with just one triplet of items: considering the generic triplet composed of three items 1, 2 and 3, the first two items are generated in such a way that the weights w_1 and ℓ_1,and w_2 and ℓ_2 are randomly selected in $[25, 50]$, and then the third item is generated setting $w_3 = W - w_1 - w_2$ and $\ell_3 = L - \ell_1 - \ell_2$. Consequently, the number of items n should be a multiple of 3 as reported in the last column of the Table 2. We remark that the instances belonging to the class 10 are introduced by Caprara and Toth (2001) to test the goodness of their lower bounds.

Table 3 Comparing selection criteria (running time negligible).

	2FFD			2BFD			2BF/FFD
class	max	min	max min	max	*avg*	*sub*	*avg* min
1	2	4	5	12	19	8	7
2	0	0	0	16	15	7	3
3	0	0	0	11	26	1	3
4	11	12	12	10	20	14	
5	31	31	31	20	28	20	27
6	0	0	0	7	12	7	0
7	0	0	0	28	38	28	2
8	1	0	0	40	39	31	9
9	0	1	2	5	17	4	4
10	0	0	2	5	2	0	0
total	**45**	**48**	**52**	**151**	**219**	**120**	**68**

	2BFD-LB			2BFD-DYNLB		
class	max	*avg*	*sub*	max	*avg*	*sub*
1	19	31	30	24	35	34
2	11	33	13	15	34	13
3	12	36	14	14	33	14
4	29	36	36	30	37	37
5	31	38	37	31	38	36
6	7	25	9	7	36	9
7	9	40	4	13	39	5
8	34	32	40	40	40	40
9	11	31	24	20	33	23
10	1	34	4	0	29	3
total	**164**	**336**	**211**	**194**	**354**	**214**

Table 3 reports the results of the comparisons among the different criteria discussed in Section 2, and the resulting 13 greedy algorithms. Each column reports the number of *best* solutions computed by the corresponding algorithm on the 10 different classes of instances, where the best value for each instance is the best value computed by at least one of the 13 algorithms reported. The reported results show the superiority of the *avg* criteria combined with the BF insertion rule: actually, considering the 7 algorithms summarised

in Table 1, 2BFD$_{avg}$ computes 219 bests, 68 more than the second ranked 2BFD$_{\max}$. Such a behaviour is confirmed also in the case of the 2BFD-LB$_{avg}$ and 2BFD-DYNLB$_{avg}$ algorithms. Table 3 demonstrates also the capability of the two algorithmic frameworks – 2BFD-LB and 2BFD-DYNLB – to improve the quality of the greedy solutions: 2BFD-DYNLB$_{avg}$ is able to compute 354 best solutions out of 400, and to improve the number of the best solutions computed by 2BFD$_{avg}$ and 2BFD-LB$_{avg}$ of 117 and 18, respectively.

To evaluate our algorithms and to compare them with those in the literature, we use the number of optimal solutions computed, as already done in the 2CBP literature. With regards to our algorithms, optimality is checked comparing the solution value with the lower bound reported in Monaci and Toth (2006) (only for instances belonging to classes 1, 6, 7, 9, 10), and available online at `http://or.dei.unibo.it/library`, or with the optimal solutions computed by Cplex within a time limit of 1 hour, when available. For the competitor algorithms, we consider the number of optimal solutions reported in the corresponding paper.

Table 4 Comparing greedies (running time negligible).

class	2BFD-DYNLB *avg*	2BFD-DYNLB *all*	2FFD$_{\mu}$	GREEDY	GREEDY + 2REF
1	14	14	6	10	12
2	32	38	39	39	40
3	30	37	40	40	40
4	20	23	19	22	25
5	37	39	32	36	36
6	16	20	10	13	17
7	22	23	19	19	17
8	40	40	40	40	40
9	11	13	10	17	21
10	0	0	0	0	0
total	**222**	**247**	**215**	**236**	**248**

Table 4 reports the results of the comparison among 2BFD-DYNLB$_{avg}$ and other algorithms illustrated in Section 2 whose results are extracted from those reported in Caprara and Toth (2001): the greedy based constructive heuristic 2FFD$_{\mu}$ is that proposed by Spieksma (1994); *GREEDY* provides the best results among six greedy algorithms while *GREEDY+2REF* improves such solutions by applying the 2REF local search. Finally, column *all* reports the best results for all the three 2BFD-DYNLB algorithms. The results in Table 4 show that 2BFD-DYNLB$_{avg}$ is slightly better than 2FFD$_{\mu}$, especially on the instances in class 1. Further, we remark that GREEDY has comparable results with all the 2BFD-DYNLB only by adding the local search 2REF. The results prove that our simple 2BFD-DYNLB$_{avg}$ is competitive with more sophisticated constructive approaches.

Finally, we compare the results of our NSwR-2CBP algorithm with the Set-Covering based Heuristic (SCH) proposed by Monaci and Toth (2006), and the Recovering Beam Search with MatHeuristic (RBS-MH) by Billaut et al (2015). To the best of our knowledge, these algorithms are the best available in the literature. The parameter setting of NSwR-2CBP have been obtained through a preliminary computational analysis performed on a limited number of instances (classes 1 and 9) in order to maximise the number of optimal solutions computed. The values of the parameters R, M_1, M_2, δ_1 and δ_2 belong to the sets $\{1000, 1500, 2000, 2500\}$, $\{20, 30, 40\}$, $\{20, 30, 40\}$, $\{7, 9, 11\}$ and $\{15, 17, 19, 21\}$, respectively. We remark that the overall results showed little differences when varying the M_1, M_2, δ_1 and δ_2 parameters. On the contrary, the number of restart R can influence the final results. Finally, we selected the following setting: $R = 2000$, $M_1 = M_2 = 30$, $\delta_1 = 7$ and $\delta_2 = 19$.

Table 5 Comparing competitors.

		NSwR-2CBP		SCH (30s)		SCH (100s)		RBS-MH	
class	n	# opt	time	# opt	time	# opt	time	# opt	time
	50	10	–	10	–	10	–	10	–
1	100	7	8.66	5	10.10	5	32.05	7	45.61
	200	5	46.58	1	29.63	3	93.69	8	608.87
	total	**32**		**26**		**28**		**35**	
	50	9	–	9	–	9	–	9	–
6	100	5	10.28	5	15.63	5	50.73	5	153.13
	200	1	63.18	0	26.01	2	61.49	0	743.79
	total	**25**		**24**		**26**		**24**	
	50	9	–	9	–	9	–	8	–
7	100	3	10.55	3	11.67	3	37.07	3	0.32
	200	7	64.80	7	19.98	7	59.93	6	428.99
	total	**29**		**29**		**29**		**27**	
	50	9	–	9	–	9	–	9	–
9	100	0	8.49	0	19.40	0	52.45	0	1.59
	200	0	46.27	0	29.05	0	73.17	0	23.28
	total	**19**		**19**		**19**		**19**	
	51	9	–	10	–	10	–	0	–
10	99	0	9.36	8	15.26	9	50.38	0	7.97
	201	0	60.61	0	28.58	2	97.49	0	97.92
	total	**19**		**28**		**31**		**10**	
total		**124**		**126**		**133**		**115**	
total (no 10)		*105*		*98*		*102*		*105*	

Each column of Table 5 reports the number of optimal solutions computed (*# opt*) and the average running times (*time*). We report the running time only for the largest instances since they are the most significant. We report two columns for SCH since the heuristic proposed by Monaci and Toth (2006) showed quite different results changing the algorithm time limit from 30 seconds to 100. Finally, we report the results only for the instances belonging to the classes 1, 6, 7, 9 and 10 since they result being the most difficult in the

previous studies, and therefore are the only available. Note that the results for the smallest instances, that is those with $n = 25$ (24 for class 10) are omitted since they are always equal to the optimal values and, by consequence, # opt is always equal to 10.

The results summarised in the second last row of Table 5 showed that our simple algorithm is competitive with its more sophisticated competitors: NSwR-2CBP computes a larger number of optimal solutions than RBS-MH, almost the same amount as those computed by SCH within a time limit of 30 seconds, and less than SCH within a time limit of 100 seconds. In terms of number of optimal solutions, the main difference between NSwR-2CBP and SCH within a time limit of 100 seconds are in the results obtained for class 10: actually SCH computes 12 optimal solutions more than NSwR-2CBP. We recall that such instances are generated in such a way that it is easier to compute the lower bound (equal to $\frac{n}{3}$) and, by construction, only one optimal solution exists. By consequence, such instances are really hard for neighbourhood based algorithm, as also proved by the standard RBS and RBS-MH proposed by Billaut et al (2015). As a matter of fact, if we do not consider the instances in class 10 (last row of Table 5), we observe that NSwR-2CBP and RBS-MH are the algorithms which compute the largest number of optimal solutions.

Running times are normalised with respect to the slowest CPU, which is the Digital Alpha 533 MHz used by Caprara and Toth (2001), while for Billaut et al (2015) is 1.7GhZ. This means that the real running time of NSwR-2CBP and RBS-MH has been multiplied respectively for 4.503 and 3.189 in order to make them comparable with those of SCH. Running times for RBS-MH are the time-to-target time, while the authors reported that "*the CPU time required by RBS-MH is on the average well below 400 seconds*", which means less than 1300 normalised seconds. NSwR-2CBP seems competitive also in terms of running time with respect to its competitors. Note that the reported results of NSwR-2CBP are obtained setting 2000 restarts in order to improve the results for class 10, while the results for instances belonging to the other classes can be reached with a smaller number of restarts (about 60%, i.e. 1200 restarts).

5 Conclusions

The development of the heuristics for the 2CBP problem is challenged by the need of a proper definition of the criterion for evaluating the feasibility of the two capacity constraints on the two different dimensions.

We proposed an extended analysis of different criteria for driving both constructive and improvement algorithms. Such an analysis suggested the use of the surrogate weight based on the average of the weights w and ℓ. Such a criterion has been then used to develop two simple and effective algorithms for solving the 2CBP problem.

Given a lower bound procedure returning the value z_{LB}, the first algorithm 2BFD-DYNLB is an extension of the classical BFD greedy for the bin packing

problem, in which the algorithm starts with z_{LB} bins open instead of only 1. Further, at the end of each iteration, the value z_{LB} is recomputed and, if the new lower bound z_{LB} is greater than the current number of open bins z, the algorithm will open $z_{LB} - z$ new bins.

The second algorithm NSwR-2CBP is a simple multi start neighbourhood search whose main characteristics is to exploit the unfeasibility of a solution in a clever way. The algorithm tries to release a bin in order to minimise its surrogate weight by swapping pairs of items between bins, and then it spreads its items on the other bins leading to a possible unfeasible solution. In that case, the above swapping is exploited to regain the feasibility.

The computational analysis on the literature benchmark sets (for a total number of 400 instances) proves the efficiency and the effectiveness of our simple algorithms when compared with the corresponding best algorithms available in literature.

The 2CBP shares its basic features with the problems addressed in Aringhieri and Dell'Amico (2005a,b) and Aringhieri et al (2016), that is a quite flat objective function opposed to some capacity constraints. For these problems, we remark that the idea of exploiting the unfeasibility forcing the reduction of the objective function by spreading the elements of a solution represents the key component of the corresponding solution algorithms. Therefore, it seems promising to adopt such an approach as a general solution approach for this class of problems. The validation of such a claim can represent an interesting future research direction.

Acknowledgements The authors would thank the students Gianluca Bortignon and Federico Iannicelli for running part of the computational tests.

References

Alves C, de Carvalho JV, Clautiaux F, Rietz J (2014) Multidimensional dual-feasible functions and fast lower bounds for the vector packing problem. European Journal of Operational Research 233(1):43 – 63

Aringhieri R (2009) Composing medical crews with equity and efficiency. Central European Journal of Operations Research 17(3):343–357

Aringhieri R, Dell'Amico M (2005a) Comparing metaheuristic algorithms for sonet network design problems. Journal of Heuristics 11(1):35–57

Aringhieri R, Dell'Amico M (2005b) Solution of the SONET Ring Assignment Problem with capacity constraints. In: Rego C, Alidaee B (eds) Metaheuristic Optimization via Memory and Evolution: Tabu Search and Scatter Search, Kluwer Academic Publisher, pp 93–116

Aringhieri R, Landa P, Soriano P, Tànfani E, Testi A (2015) A two level Metaheuristic for the Operating Room Scheduling and Assignment Problem. Computers & Operations Research 54:21–34

Aringhieri R, Bruglieri M, Malucelli F, Nonato M (2016) A special vrp arising in the optimization of waste disposal: a real case. Transportation Science URL `http://hdl.handle.net/2318/1596992`, accepted paper. To appear

Billaut JC, Della Croce F, Grosso A (2015) A single machine scheduling problem with two-dimensional vector packing constraints. European Journal of Operational Research 243(1):75–81

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

Della Croce F, Ghirardi M, Tadei R (2004) Recovering beam search: enhancing the beam search approach for combinatorial optimization problems. Journal of Heuristics 10:1089

Garey M, Graham R, Johnson D, Yao A (1976) Resource constrained scheduling as generalized bin packing. Journal of Combinatorial Theory, Series A 21(3):257 – 298

Gendreau M, Hertz A, Laporte G (1994) A tabu search heuristic for the vehicle routing problem. Management Science 40(10):1276–1290

Guazzone M, Anglano C, Aringhieri R, Sereno M (2013) Distributed coalition formation in energy-aware cloud federations: A game-theoretic approach (extended version). CoRR abs/1309.2444, URL `http://arxiv.org/abs/1309.2444`

Kellerer H, Kotov V (2003) An approximation algorithm with absolute worst-case performance ratio 2 for two-dimensional vector packing. Operations Research Letters 31(1):35–41

Kellerer H, Pferschy U, Pisinger D (2004) Knapsack Problems. Springer

Martello S, Toth P (1990) Knapsack Problems: Algorithms and Computer Implementations. Wiley-Intersci. Ser. Discrete Math. Optim., John Wiley and Sons

Monaci M, Toth P (2006) A set-covering-based heuristic approach for bin-packing problems. INFORMS Journal on Computing 18(1):71–85

Shachnai H, Tamir T (2012) Approximation schemes for generalized two-dimensional vector packing with application to data placement. Journal of Discrete Algorithms 10(1):35–48

Spieksma FC (1994) Branch-and-bound algorithm for the two-dimensional vector packing problem. Computers and Operations Research 21(1):19–25

Wilcox D, McNabb A, Seppi K (2011) Solving virtual machine packing with a reordering grouping genetic algorithm. In: 2011 IEEE Congress of Evolutionary Computation, CEC 2011, pp 362–369