Computers & Operations Research 173 (2025) 106860

Contents lists available at ScienceDirect

Computers and Operations Research

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

Classification and evaluation of the algorithms for vector bin packing

Clément Mommessin [a,b], Thomas Erlebach [c], Natalia V. Shakhlevich [b,*]

[a] IMT Atlantique, Nantes Université, École Centrale Nantes, CNRS, INRIA, LS2N, UMR 6004, F-44000 Nantes, France
[b] School of Computer Science, University of Leeds, UK
[c] Department of Computer Science, University of Durham, UK

ARTICLE INFO

Dataset link: https://github.com/Vectorpack/Vectorpack_cpp, https://github.com/Vectorpack/experiments_vector_paper

Keywords:
Vector bin packing
Bin packing
Heuristics

ABSTRACT

Heuristics for Vector Bin Packing (VBP) play an important role in modern distributed computing systems and other applications aimed at optimizing the usage of multidimensional resources. In this paper we perform a systematic classification of heuristics for VBP, with the focus on construction heuristics. We bring together existing VBP algorithms and their tuning parameters, and propose new algorithms and new tuning parameters. For a less studied class of multi-bin algorithms, we explore their properties analytically, considering monotonic and anomalous behavior and approximation guarantees. For empirical evaluation, all algorithms are implemented as the *Vectorpack* library and assessed through extensive experiments. Our findings may serve as the basis for the development of more complex, hybrid algorithms, hyperheuristics and machine learning algorithms. The *Vectorpack* library can also be adjusted for addressing enhanced VBP problems with additional features, which arise in applications, especially those typical for modern distributed computing systems.

1. Introduction

In the *bin packing problem* (BP), there are n items $\mathcal{I} = \{1, 2, \dots, n\}$ with given sizes s_i for items $i \in \mathcal{I}$, and a set of identical bins, each bin with capacity c. The items need to be allocated to the minimum number of bins without exceeding the bin capacity c. We denote by $\mathcal{B}$ the set of activated bins at some stage of an algorithm and in the final solution.

In the *vector bin packing problem* (VBP), sometimes called *multi-capacity bin packing problem*, each of the n items is characterized by a d-tuple of sizes s_{ih}, $i \in \mathcal{I}$, $1 \le h \le d$. The bins are identical, with a given size c_h for each dimension h, $1 \le h \le d$.

Allocating a subset of items $\mathcal{I}' \subseteq \mathcal{I}$ to one bin is feasible if

$$\sum_{i \in \mathcal{I}'} s_{ih} \le c_h \quad \text{for each } h = 1, \dots, d.$$

If a bin B_k is packed with a subset of items $\mathcal{I}_k \subseteq \mathcal{I}$, then its residual capacity is characterized by d values

$$r_{kh} = c_h - \sum_{i \in \mathcal{I}_k} s_{ih}, \quad 1 \le h \le d.$$

In what follows, we use the notations $\mathbf{s}_i = (s_{i1}, s_{i2}, \dots, s_{id})$ and $\mathbf{r}_k = (r_{k1}, r_{k2}, \dots, r_{kd})$ in bold font to denote the d -component vectors of item sizes and bin residual capacities. During the course of an algorithm, we denote by $\mathcal{I}^* \subseteq \mathcal{I}$ the subset of *unallocated* items.

Allocating item i in addition to those already in the bin is *feasible* if

$$s_{ih} \le r_{kh} \quad \text{for each } h = 1, \dots, d. \tag{1}$$

Without loss of generality, we assume that the bin sizes are normalized, so that

$$c_h = 1, \quad 1 \le h \le d. \tag{2}$$

If an instance is given with $c_h \neq 1$, then normalization is performed by setting $c_h = 1$ and replacing original values s_{ih} by s_{ih}/c_h for all items $i \in \mathcal{I}$ and all dimensions h, $1 \le h \le d$.

With normalized bin capacities, the simplest and most popular lower bound for the VBP problem decomposes the d-dimensional problem into d one-dimensional subproblems. For each subproblem corresponding to dimension h, the minimum number of bins is at least $\lceil \sum_{i \in \mathcal{I}} s_{ih} \rceil$. Therefore, the minimum number of bins for the main d-dimensional problem is at least

$$LB = \max_{1 \le h \le d} \left\{ \left\lceil \sum_{i \in \mathcal{I}} s_{ih} \right\rceil \right\}. \tag{3}$$

For examples of advanced methods known for lower bound calculation, see Spieksma (1994), Caprara and Toth (2001), Alves et al. (2014, 2016), Brandão and Pedroso (2016) and Gurski and Rehs (2020). Note

* Correspondence to: School of Computer Science, University of Leeds, LS2 9JT, UK.
E-mail address: N.Shakhlevich@leeds.ac.uk (N.V. Shakhlevich).

https://doi.org/10.1016/j.cor.2024.106860
Received 5 May 2023; Received in revised form 25 September 2024; Accepted 30 September 2024
Available online 5 October 2024

that comparing the accuracy of lower bounds is beyond the scope of our paper.

In the decision version of VBP, denoted by VBP(m), the number of bins m is given and fixed, and the objective is to find a feasible allocation of n items into at most m bins, if one exists. Note that the decision version of VBP coincides with the decision versions of the following problems: *vector scheduling* (Chekuri and Khanna, 2004), *d-constraint multiple knapsack problem* (Ahuja and Cunha, 2005), *multiple multidimensional knapsack problem* (Cacchiani et al., 2022), *multidimensional bin packing to maximize the number of items packed* (Coffman et al., 2013).

1.1. Literature review

In this paper we consider construction heuristics for VBP in the setting with arbitrary dimension d: their general principles, implementation details, properties, and performance, evaluated via experiments on several types of datasets. It complements the mainstream research, which is mostly dominated by the approximability study and by the design of enumerative algorithms and metaheuristics.

Summarizing approximability results we note that there are provable limitations on the design of fast algorithms with guaranteed accuracy smaller than d; see the surveys by Christensen et al. (2017), Csirik and Dósa (2018), Coffman et al. (2013) and Epstein and van Stee (2018). Considering exact methods, we refer to Delorme et al. (2016) and Baldacci et al. (2023) for the comparative analysis of state-of-the-art algorithms for the one-dimensional case and to Pessoa et al. (2021) for both one- and multi-dimensional cases. Although major progress has been recently achieved in the design of powerful solvers, computation time is still quite high for difficult one-dimensional instances with $n \leq 1000$ and for multi-dimensional instances with $d \leq 20$ and $n \leq 200$; the computation time in experiments is usually limited to 1 hour, which is often unacceptable for real-world applications. Most of the time, successful approaches, which compete very well in experiments and outperform other approaches in terms of computation time and accuracy, are restricted to handling 2-dimensional problems only; see, e.g., Wei et al. (2020).

Considering metaheuristics, their comparison is performed mostly for the one-dimensional case; see the survey by Munien et al. (2020). The comparison of several approaches for the multi-dimensional case can be found in the recent paper by Nagel et al. (2023) . Metaheuristics are capable of finding solutions of good quality, outperforming in some cases best-known heuristics, but this is achieved at the cost of substantial computation time, up to 1 hour for some methods.

To conclude our overview we cite Epstein and van Stee (2018) who observed that “after many years of study, we still do not understand the multidimensional case as well as the one-dimensional case”. While this observation is stated in relation to approximation algorithms, it is generally true in relation to fast heuristics for VBP, the subject of our study. Note that in the most recent handbook by Gonzalez (2018), which includes six surveys on packing problems, only two surveys discuss practical packing algorithms (although geometric bin packing is not very relevant for our study), while the remaining four surveys focus on approximability results.

The number of publications which evaluate fast heuristics for VBP is relatively small and quite often focuses on the two-dimensional case. The report by Panigrahy et al. (2011) has been the main reference for researchers for more than 10 years. The need for an up-to-date methodological paper on VBP heuristics has become particularly acute due to the growing research in the context of distributed computing, where fast VBP algorithms are adapted for solving enhanced problems with additional features typical for Grids and Clouds.

1.2. Contributions and paper outline

The main goal of our work is to systemize the collection of known VBP heuristics, to enrich it with new approaches, and to compile a comprehensive list of algorithms’ parameters. For old and new methods, we study combinations of parameters not explored in prior research. The parameters are special size measures for items and bins, as well as scores for item-bin allocations, together with weights prioritizing the dimensions.

A high-level summary of the VBP heuristics and their complexity estimates is presented in Fig. 1. The first approach, *item-centric*, stems from the classical algorithms originally proposed in the context of one-dimensional BP. Item-centric algorithms consider items one-by-one and find the “best” bin for a current item.

Bin-centric algorithms consider the bins one-by-one and find the “best” items to be allocated to a current bin. In either case, a new bin is activated only if no further allocation can be done with the current pool of activated bins — an important feature aimed at minimizing the number of bins in use.

The two traditional classes of algorithms, item-centric and bin-centric, activate a new bin only if an item cannot be allocated using activated bins. An alternative approach is *multi-bin activation* (MB). It solves a series of decision problems VBP(m), defined for different target numbers of bins m. The output is the smallest possible m for which a feasible solution is found. For solving an individual problem VBP(m), we propose special adaptations of item-centric algorithms and a new family of *pairing* algorithms. The latter consider all feasible item-bin pairs when making allocation decisions.

The whole algorithmic toolkit for VBP is evaluated via computational experiments, providing guidelines for practitioners, with recommendations of the most appropriate algorithm choice depending on the features of datasets.

Classification and empirical analysis of the VBP algorithms may serve as the starting point for the development of advanced algorithms: hybrid algorithms, hyperheuristics and machine learning algorithms for VBP. It can also be used for addressing enhanced problems with additional features, which arise in applications, especially those typical for modern distributed computing systems. Examples of such problems can be found in the papers by Garefalakis et al. (2018), Cai et al. (2022) or Mommessin et al. (2023), which extend the VBP problem with additional ‘affinity’ features, limited co-location of certain items to one bin, or in the papers by Gabay and Zaourar (2016) and Jangiti et al. (2019a), with non-identical bins.

The paper is organized as follows. We start with a summary of existing and new algorithms for the item-centric and bin-centric approaches, presented in Sections 2 and 3. In Section 4, we introduce expressions for weights used in heuristics for prioritizing the dimensions. The new multi-bin approach is elaborated in Section 5. Overall, Sections 2–5 describe dozens of algorithms and their variations, capable of producing different solutions for the same instance of the multidimensional VBP.

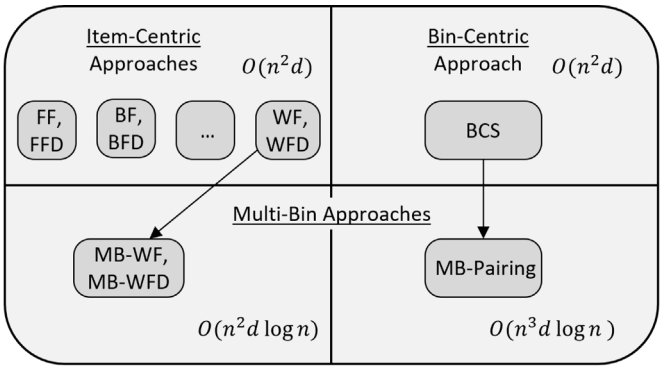
The new family of multi-bin activation algorithms is analyzed theoretically in Sections 6–7, where we study the issues related to their monotonicity and anomalous behavior and discuss their accuracy guarantees.

The whole range of the solution approaches is analyzed empirically using the existing and new classes of benchmarks; see Section 8. Conclusions are presented in Section 9.

2. Item-centric approach

Algorithms for the BP problem have been under study since the early 1970s. Being a 1-dimensional version of VBP, in the BP problem the items are prioritized according to their sizes and partially loaded bins are prioritized according to their residual capacities.

A natural and widely explored approach is to replace an instance of the d-dimensional VBP by an instance of the 1-dimensional BP and



Abbreviations

FF, FFD – First Fit, First Fit Decreasing
BF, BFD – Best Fit, Best Fit Decreasing
WF, WFD – Worst Fit, Worst Fit Decreasing
MB-WF – Multi Bin approach combined with WF
MB-WFD – Multi Bin approach combined with WFD
BCS – Bin Centric Approach with Scores computed for item-bin pairs
MB-Pairing – Multi-Bin Approach with scores computed for item-bin pairs

Fig. 1. Heuristic types for VBP.

Table 1
Ordering rules for item-centric algorithms.

The list of activated bins $\mathcal{B}$ in	The list of items $\mathcal{I}$ in	
	An arbitrary fixed order	Decreasing order of $v(i)$
An arbitrary fixed order	First Fit (*FF*)	First Fit Decreasing (*FFD*)
increasing order of $v(B_k)$	Best Fit (*BF*)	Best Fit Decreasing (*BFD*)
Decreasing order of $v(B_k)$	Worst Fit (*WF*)	Worst Fit Decreasing (*WFD*)

to apply the algorithms known in the BP literature. This is done by aggregating the d-tuples for items' sizes and for bins' residual capacities into single scalar measures: a *combined size* $v(i)$ of item $i \in \mathcal{I}$, and a *combined residual capacity* $v(B_k)$ for an activated bin $B_k \in \mathcal{B}$. There are multiple ways for computing $v(i)$ and $v(B_k)$, as we discuss at the end of this section.

Any *item-centric* algorithm operates on an ordered list of items $\mathcal{I}$ and an ordered list of activated bins $\mathcal{B}$, where $\mathcal{B}$ initially consists of a single bin. The items are considered one by one in accordance with their ordering in $\mathcal{I}$. A current item is allocated to the first feasible bin on the ordered list $\mathcal{B}$, where feasibility is checked via (1), or a new bin is activated and added at the end of list $\mathcal{B}$, if no feasible allocation can be found. Since every allocation changes the residual capacity of the bin used, the ordering of list $\mathcal{B}$ requires updating after each packing of an item.

Typical priority rules for ordering $\mathcal{I}$ and $\mathcal{B}$ were originally formulated for one dimensional BP and later on adopted for the multidimensional case of VBP, with scalars $v(i)$ and $v(B_k)$ used as the size measures for items and bins. The summary of the item-centric algorithms is presented in Table 1. Note that each algorithm from the table gives rise to multiple versions, depending on the way the size measures $v(i)$ and $v(B_k)$ are computed.

In what follows we introduce the expressions for computing the size measures. Early examples of such measures can be found in the papers by Kou and Markowsky (1977) and Maruyama et al. (1977) and in the literature on the *multidimensional knapsack problem* (e.g., Kellerer et al. (2004)). The most recent summary is provided by Panigrahy et al. (2011). The extended list of measures, which includes the latest findings from publications in Distributed Computing, is presented in Table 2.

The first three size measures are the ℓ_∞, ℓ_1 and ℓ_2 norms of the corresponding d-tuples. The fourth size measure only differs from the third for the computation of the combined bin residual capacity, which is based on the bin load vector, with components $(1 - r_{kh})$, instead of the bin residual capacity vector, with components r_{kh}. If an algorithm calls for creating an ordered list of items and ordered list of bins, then the size measures $v(i)$ and $v(B_k)$ of the same type are usually used. The partner measures are listed in one line of Table 2. For simplicity, the square root is dropped from the expressions for the ℓ_2-size, as it has no effect on the ordering of items and bins.

We distinguish between the forward and reverse size measures of bins. A *forward* size measure is non-decreasing: if $\mathbf{r}_k \leq \mathbf{r}_j$ for the vectors of residual capacities of two bins B_k and B_j, then $v(B_k) \leq v(B_j)$. The first three bin size measures in Table 2 satisfy this property. A *reverse* size measure is non-increasing. An example of such a measure is in the last row of Table 2.

Note that the BF algorithm with the ℓ_2-size of bin residual capacity does not necessarily produce the same solution as the WF algorithm with the ℓ_2-size of bin load. For example, consider an instance where the first four items are of sizes (0.2, 0.9), (0.1, 0.8), (0.5, 0.6) and (0.6, 0.4). For that instance any item-centric algorithm activates four bins, with one item per bin. If we use BF with the ℓ_2-size of residual capacity, then $v(B_1) = 0.65$, $v(B_2) = 0.85$, $v(B_3) = 0.41$, $v(B_4) = 0.52$, and for allocating the next, fifth item, the bins are considered in the order B_3, B_4, B_1, B_2. On the other hand, if we use WF with the ℓ_2-size of bin load, then $v(B_1) = 0.85$, $v(B_2) = 0.65$, $v(B_3) = 0.61$, $v(B_4) = 0.52$, and for allocating the next item the bins are considered in the order B_1, B_2, B_3, B_4.

All norms can be considered as unweighted, assuming $w_h = 1$ for all $1 \leq h \leq d$, or with special non-negative weights defined for each dimension h. We give the formulae for w_h in Section 4.

We keep in Table 2 the most promising measures, as found in experiments. The ℓ_∞ and ℓ_1-sizes are most popular and can be found in multiple sources; see, e.g., Caprara and Toth (2001) and Spieksma (1994). The ℓ_2-sizes were studied by Maruyama et al. (1977) (the first version in Table 2) and Shi et al. (2013) (the version for bin load).

For completeness, we describe below the alternative formulae which appear to be less successful in experiments:

Table 2
Combined size measures of items and bins for item-centric algorithms.

Measure type	Combined size of item $i \in \mathcal{I}^*$	Combined residual capacity of bin $B_k \in \mathcal{B}$	
ℓ_∞-size	$v(i) = \max_{1 \le h \le d} \{w_h s_{ih}\}$	$v(B_k) = \max_{1 \le h \le d} \{w_h r_{kh}\}$	
ℓ_1-size	$v(i) = \sum_{h=1}^{d} w_h s_{ih}$	$v(B_k) = \sum_{h=1}^{d} w_h r_{kh}$	Forward measures
ℓ_2-size	$v(i) = \sum_{h=1}^{d} w_h (s_{ih})^2$	$v(B_k) = \sum_{h=1}^{d} w_h (r_{kh})^2$	
ℓ_2-size of bin load	$v(i) = \sum_{h=1}^{d} w_h (s_{ih})^2$	$v(B_k) = \sum_{h=1}^{d} w_h (1 - r_{kh})^2$	Reverse measure

- The *Volume* of items $v(i) = \Pi_{h=1}^{d} w_h s_{ih}$ and the *unused volume* of bins $v(B_k) = \Pi_{h=1}^{d} w_h r_{kh}$, in their unweighted and weighted forms, were explored by Maruyama et al. (1977) and later by Panigrahy et al. (2011). Note that if an item has size 0 in at least one dimension, then the measure $v(i)$ is 0, which leads to information loss while comparing items.
- The *Aggregated Rank* was proposed by Jangiti et al. (2019b). It is defined as the sum of rank-values of an item in every dimension, where for a given dimension, an item i is given the rank u if it has the uth smallest size in this dimension among all items. The rank-values of bins are defined similarly, with respect to their residual capacities in every dimension.
- There is no need for computing size measures if items and bins are ordered *lexicographically*. Such an ordering may be useful in the presence of bottleneck dimensions. The item-centric algorithms of this type were explored by Kou and Markowsky (1977) and Stillwell et al. (2010).

A more complicated version of the item-centric approach is proposed by Spieksma (1994) for the 2-dimensional case; see also Caprara and Toth (2001). Having found a first solution, its part corresponding to "well-filled" bins is kept, and an attempt is made to obtain a better solution for the items from the remaining bins. The new, smaller problem is solved by the same item-centric algorithm, but with modified weights w_h in the expressions for the item and bin size-measures. Note that the ℓ_1-size measure is used by Spieksma (1994) and Caprara and Toth (2001), while in fact the iterative heuristic can use any weighted size measure from Table 2 in combination with any item-centric algorithm. The generalization of that approach to the d-dimensional case with $d > 2$ is not trivial and beyond the scope of our paper.

3. Bin-centric approach

Bin-centric algorithms pack bins one at a time. The unpacked items feasible for the current bin are prioritized in accordance with a specified policy, and they are assigned to the current bin until no further items can be packed. After that a new bin is activated and the process continues until all items are packed.

In the two simplest versions of the bin-centric approach, the items are prioritized either according to their numbering or according to their size $v(i)$, with the 'largest item first' rule. In the former case, the bin-centric algorithm produces the same solution as the item-centric algorithm FF. In the latter case, the solution is the same as the one found by FFD.

Instead of considering item sizes in isolation, it can be preferable to take into account the item sizes together with the free space in the current bin, and to estimate the appropriateness of a candidate item to a bin. Table 3 presents various *item-bin scores* ξ_{ik}, which measure how well an unallocated item $i \in \mathcal{I}^*$ fits into a current bin B_k. The scores are computed only for items which are feasible for bin B_k in terms of condition (1). The weights w_h for each dimension are defined in Section 4. The expressions for the scores are designed so that the items which use the residual bin capacity to the highest extent receive the highest scores. At each packing step, the bin-centric algorithm gives preference to the item which delivers the largest score for the current bin.

For an item i and a bin B_k, the closeness between the vector of the item sizes $\mathbf{s}_i$ and the vector of the bin residual capacities $\mathbf{r}_k$ can be measured via the *Dot-Product* $\mathbf{s}_i \cdot \mathbf{r}_k$ of the two vectors. The three dot-product scores in Table 3 differ in scaling coefficients. The first dot-product score is a natural generalization of the dot-product formula which uses different weights w_h for dimensions. It was proposed and evaluated in experiments by Panigrahy et al. (2011). The other two dot-product scores were proposed by Gabay and Zaourar (2016) to measure the angle between the two vectors (the lower the angle, the better), and the projection of the item vector $\mathbf{s}_i$ to the free space vector $\mathbf{r}_k$ (the larger the projection, the better). Note that the difference in Dot-Product 1 and Dot-Product 3 scores has no effect in a bin-centric algorithm since the scaling weight $1/\left(\|\mathbf{r}_k\|_2\right)^2$ has a constant value for the current bin. We keep the expression for Dot-Product 3 in the table since we reuse the ξ-expressions in the MB-Pairing algorithm (Section 5), where the scores are computed for all item-bin pairs.

The *Normalized Dot-Product* score can be considered as a slightly modified version of the *Fitness* score of Cai et al. (2022). In the expression, the item sizes s_{ih} are normalized by D_h, the total demand of all items in dimension h, and the residual capacities r_{kh} of the bins are normalized by R_h, the total residual capacity of all bins in dimension h. While D_h is treated as a constant computed once at the beginning of the algorithm, the values of R_h are recalculated dynamically, each time an item is allocated to a bin and its residual capacity changes.

The *ℓ_2-Norm of Slacks*, also introduced by Panigrahy et al. (2011), measures the norm of residual capacity that would be left in bin B_k if item i was allocated to it. The best allocation corresponds to the smallest slack value. Since the bin-centric algorithm gives preference to the highest score, the ξ-expression for the slack is set as negative.

We introduce in this work two new item-bin scores, denoted by *Tight Fill*, which measure how well an item would use free space in a bin, if allocated. The ratios s_{ih}/r_{kh} characterize the fitness of item i for allocation to bin B_k with respect to dimension h. The first *Tight Fill* expression is the weighted sum of the fitness values, while the second expression focuses on the dimension with the smallest fitness value.

4. Prioritizing dimensions

The expressions for size measures $v(i)$ and $v(B_k)$ introduced in Section 2 and item-bin scores ξ_{ik} in Section 3 are presented in the most general form, possibly with different weights w_h for dimensions $h = 1, \ldots, d$. The expressions for w_h, collected from multiple sources, are summarized in Table 4. We do not report alternative definitions of w_h that were found unsuccessful in the existing body of research. In particular, the usage of the *degree of dominancy* weight proposed

Table 3
Bin-item scores for bin-centric algorithms.

Score type	Score formula with item $i \in \mathcal{I}^*$, bin $B_k \in \mathcal{B}$
Dot-Product 1	$\xi_{ik} = \sum_{h=1}^{d} w_h s_{ih} r_{kh}$
Dot-Product 2	$\xi_{ik} = \frac{1}{\|\mathbf{s}_i\|_2 \cdot \|\mathbf{r}_k\|_2} \sum_{h=1}^{d} w_h s_{ih} r_{kh}$
Dot-Product 3	$\xi_{ik} = \frac{1}{(\|\mathbf{r}_k\|_2)^2} \sum_{h=1}^{d} w_h s_{ih} r_{kh}$
Normalized Dot-Product	$\xi_{ik} = \sum_{h=1}^{d} w_h \frac{s_{ih}}{D_h} \cdot \frac{r_{kh}}{R_h} \quad \left(D_h = \sum_{i \in \mathcal{I}} s_{ih},\ R_h = \sum_{B_k \in \mathcal{B}} r_{kh} \right)$[a]
ℓ_2-Norm of Slacks	$\xi_{ik} = -\sum_{h=1}^{d} w_h \left(r_{kh} - s_{ih}\right)^2$
Tight fill with sum	$\xi_{ik} = \sum_{h=1}^{d} w_h \frac{s_{ih}}{r_{kh}}$
Tight fill with min	$\xi_{ik} = \min_{1 \le h \le d} w_h \frac{s_{ih}}{r_{kh}}$

[a] The values D_h, $1 \le h \le d$, are computed only once, considering the whole set of items $\mathcal{I}$, while the values R_h are recalculated dynamically each time the load of one of the activated bins from $\mathcal{B}$ changes.

Table 4
Weights w_h for dimension h, $1 \le h \le d$. For Exponential, a small constant ε is selected for scaling.

Weights	Static item-based	Dynamic item-based	Dynamic bin-based
Unit	$w_h = 1$	N/A	N/A
Average	$w_h = d_h$	$w_h = d_h^*$	$w_h = r_h$
Exponential	$w_h = e^{\varepsilon d_h}$	$w_h = e^{\varepsilon d_h^*}$	$w_h = e^{\varepsilon r_h}$
Reciprocal average	$w_h = 1/d_h$	$w_h = 1/d_h^*$	$w_h = 1/r_h$
Utilization ratio	N/A	$w_h = D_h^*/R_h$	

by Maruyama et al. (1977) was not found superior to other rules in the extensive experiments by Stillwell et al. (2010).

The expressions use the following characteristics of items and activated bins:

- $d_h = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} s_{ih}$, the average size (demand) of all items $\mathcal{I}$ in dimension h,
- $D_h^* = \sum_{i \in \mathcal{I}^*} s_{ih}$ and $d_h^* = \frac{1}{|\mathcal{I}^*|} \sum_{i \in \mathcal{I}^*} s_{ih}$, the total and the average size (demand) of all unallocated items $\mathcal{I}^* \subseteq \mathcal{I}$ in dimension h,
- $R_h = \sum_{B_k \in \mathcal{B}} r_{kh}$ and $r_h = \frac{1}{|\mathcal{B}|} \sum_{B_k \in \mathcal{B}} r_{kh}$, the total and the average residual capacity of all activated bins $\mathcal{B}$ in dimension h.

The *Unit* weights are useful if all dimensions h, $1 \le h \le d$, are of equal importance. They were popular in early papers on VBP; see, e.g., Kou and Markowsky (1977) and Maruyama et al. (1977).

The *Average* weights, initially proposed by Caprara and Toth (2001) for the 2-dimensional case, and *Exponential* weights, proposed by Panigrahy et al. (2011), are computed based on the average demand d_h of all items in dimension h or on average residual capacities of the activated bins in dimension h.

The *Reciprocal Average* and *Utilization Ratio* weights were originally introduced by Gabay and Zaourar (2016) for the purpose of solving VBP(m). The effect of using such weights is similar to normalization of item sizes in each dimension. Note that a slightly modified version of the *Reciprocal Average* weight was used by Cai et al. (2022) for computing measures for items and bins similar to our ℓ_1-size as defined in Section 2.

Each weight formula is defined in three variations, depending on whether the weights are computed once, or updated dynamically and based on item or bin characteristics. Static item-based weights are computed once, using the initial characteristics d_h of the set of items $\mathcal{I}$. Dynamic item-based weighs are computed dynamically, every time a decision is made to allocate an item to a bin. Therefore, their formulae depend on the subset of unallocated items $\mathcal{I}^*$ and (possibly) the subset of activated bins $\mathcal{B}$. Dynamic bin-based weights take into account activated bins only and they also require regular updates.

In the case of bin-centric algorithms, applying one of the expressions from Table 4 is straightforward: once the expression is selected, it is used consistently in all computations of item-bin scores ξ_{ik}. In the case of item-centric algorithms, different approaches can be adopted. One option is to select a single expression from Table 4 for computing the weights $(w_1, w_2, \ldots, w_d)$ and to use the common d-tuple of weights for finding item sizes $v(i)$ and bin sizes $v(B_k)$. Another option is to use one w-expression of type 'static item-based' or 'dynamic item-based' for finding item sizes $v(i)$ and another w-expression of type 'dynamic bin-based' for finding bin sizes $v(B_k)$; see Section 8 for further details.

Overall the weight formulae presented in Table 4 are rather general and can be easily tuned or replaced by other formulae, with $w_h \ge 0$. For example, in the presence of a bottleneck dimension h' it is reasonable to keep $w_{h'} = 1$ and to eliminate the remaining dimensions from consideration by setting $w_h = 0$ for $h \ne h'$. Note also that when the sum of item sizes is equal in every dimension, then all static item-based weights are similar to the Unit weights.

In the course of an algorithm, it may happen that the total size of remaining items in a dimension h equals 0 or, possibly, the total residual capacity of the bins in the dimension equals 0. In such a case, dimension h is deactivated by setting $w_h = 0$. This way we avoid numerical issues (division by 0) when computing the Reciprocal Average and Utilization Ratio weights, as well as the item-bin scores of certain type, namely the Normalized Dot-Product and Tight Fill.

5. Multi-bin activation approach

In this section we propose an approach that overcomes the myopic nature of item-centric and bin-centric heuristics: instead of activating one bin at a time only when it is necessary, it activates multiple bins

at the start and makes decisions on the items' allocation considering the whole pool of bins. Since the minimum number of bins is not known in advance, the multi-bin (MB) algorithm solves a series of problems VBP(m), with a trial parameter m for the number of bins in use. Depending on the outcome of solving VBP(m), the MB approach stops and outputs the current feasible solution, or attempts to find a solution to VBP(m') with a different input m'.

Compared to item-centric and bin-centric approaches, MB approach has more freedom in decision making. This comes at the cost of increased running time since VBP(m) has to be solved multiple times with different trial values of m; see Fig. 1 for algorithm types and complexity estimates.

The two key ingredients of an MB algorithm are the *bin activation strategy* to select the trial values m, and an *algorithm for solving VBP*(m).

We use notation $\mathcal{A}(m)$ for a single call of an algorithm $\mathcal{A}$ that solves the VBP(m) problem, and MB-$\mathcal{A}$ for multiple calls of $\mathcal{A}(m)$ with different values of m, organized via a specified bin activation strategy.

5.1. Bin activation strategies

The simplest approach is *incremental*. It starts with a lower bound $\underline{m}$ on the number of bins and solves a series of problems VBP($\underline{m}$), VBP($\underline{m}+1$), VBP($\underline{m}+2$), ..., terminating the first time when VBP($\underline{m}+k$) turns out to be solvable. The search can be sped up by using a *batched* bin activation approach, incrementing m in larger steps.

Another approach is to use *binary search* over a series of trial m - values, with $m \in \{\underline{m}, \dots, \overline{m}\}$, where $\underline{m}$ is the lower bound, as before, and $\overline{m}$ is an upper bound found by one of the item-centric or bin-centric approaches.

The usage of the batched or binary search strategy is justifiable if an algorithm for solving VBP(m) is *monotonic*; see Section 6 for definitions and monotonicity results.

5.2. Algorithms for solving VBP(m)

Any of the item-centric or bin-centric algorithms discussed in Sections 2–3 can be adapted for *VBP*(m) by introducing an initial step: activating m bins $\mathcal{B} = \{B_1, \dots, B_m\}$ at the start.

An adaptation of an item-centric algorithm for VBP(m) considers a current item i and selects one of the bins from the *whole* set $\mathcal{B}$ according to the rules from Table 1. For the algorithm notation we use the same name as in Section 2 , adding parameter m in parentheses to highlight the main feature of the multi-bin approach (e.g., WFD(m)).

An adaptation of a bin-centric algorithm for VBP(m) computes item-bin scores ξ_{ik} for all unallocated items and the bins of the set $\mathcal{B}$ that can fit them, and selects a highest score pair for the next allocation. Recall that the scores are defined in Table 3 . We call the resulting algorithms Pairing(m). Due to the large number of pairs to be explored, Pairing(m) is more demanding in terms of computation time compared to the item-centric adaptation; see Fig. 1 for algorithm types and complexity estimates.

Some versions of the adapted algorithms can be excluded from consideration since they produce exactly the same solution as their counterparts with bins activated one-by-one. Those which perform differently are listed in the next statement.

Statement 1. *The following algorithms with m preactivated bins produce (possibly) different solutions compared to those they produce in the traditional scenario, with bins activated one-by-one:*

- *WF(m) and WFD(m) with a forward size measure (ℓ_∞, ℓ_1 and ℓ_2-sizes of residual capacity);*
  BF(m) and BFD(m) with a reverse size measure (ℓ_2-size of bin load);
- *Pairing(m) with ξ-scores computed as Dot-Product 1-3 and Normalized Dot-Product, assuming the weights w_h are non-negative, $1 \le h \le d$.*

Proof. Consider the first part of the statement, namely the item-centric algorithms WF(m) and WFD(m) with a forward size measure. They prioritize the selection of the bin with the largest residual capacity, measured as $v(B_k)$, to pack an item. This way, they attempt to achieve a balanced load over all m bins at every step, performing differently if compared to WF and WFD applied in a scenario with bins activated one-by-one. The same is true for BF(m) and BFD(m) with a reverse bin size measure.

The alternative item-centric algorithms, FF(m), FFD(m) (with any bin size measure), WF(m) and WFD(m) with a reverse bin size measure, and BF(m) and BFD(m) with a forward bin size measure, place a current item into an empty bin B_k only if that item does not fit into any of the partially loaded bins $B_1, \dots, B_{k-1}$. The resulting solution is the same as the one found by one of the item-centric algorithms applied in a scenario with bins activated one-by-one.

To prove the second part of the statement, we first show that the following algorithms should be excluded from consideration:

- Pairing(m) with ℓ_2-Norm of Slacks score,
- Pairing(m) with Tight Fill with Sum score,
- Pairing(m) with Tight Fill with Min score.

Consider Pairing(m) with the *ℓ_2-Norm of Slacks* score. Let i be an item of size $(s_{i1}, s_{i2}, \dots, s_{id})$, B_k be a non-empty bin with residual capacity $(r_{k1}, r_{k2}, \dots, r_{kd})$ and B_q be an empty bin with residual capacity $(1, 1, \dots, 1)$. We assume that item i fits in bin B_k, i.e.,

$$s_{ih} \le r_{kh} \le 1 \text{ for all } h = 1, \dots, d.$$

Then the scores for allocating i to B_k and B_q are

$$\xi_{ik} = -\sum_{h=1}^{d} w_h (r_{kh} - s_{ih})^2 \text{ and } \xi_{iq} = -\sum_{h=1}^{d} w_h (1 - s_{ih})^2, \tag{4}$$

so that

$$\xi_{ik} \ge \xi_{iq},$$

provided $w_h \ge 0$, $1 \le h \le d$. Therefore, Pairing(m) uses an empty bin only if no item fits any of the partially loaded bins.

The proofs for the scores *Tight Fill with Sum* and *Tight Fill with Min* are similar. We just need to replace expressions (4) by

$$\xi_{ik} = \sum_{h=1}^{d} w_h \frac{s_{ih}}{r_{kh}} \text{ and } \xi_{iq} = \sum_{h=1}^{d} w_h s_{ih},$$

or by

$$\xi_{ik} = \min_{1 \le h \le d} w_h \frac{s_{ih}}{r_{kh}} \text{ and } \xi_{iq} = \min_{1 \le h \le d} w_h s_{ih}.$$

To illustrate that Pairing(m) algorithms with the scores listed in the second part of the statement can produce different solutions in the multi-bin scenario and in the scenario with bins activated one-by-one, consider the following example with two dimensions and unit weights. There are $m = 2$ bins and $n = 2$ items with sizes $(0.3,\ 0.01)$ and $(0.2,\ 0.01)$. Both items fit into a single bin. However, in the cases of Dot-Product 1, Dot-Product 3 and Normalized Dot-Product, Pairing(2) places the first item into one bin and the second item into another bin. In the case of Dot-Product 2, Pairing(2) selects initially the second item and places it into one bin and then it places the first item into another bin. ■

Note that algorithms WF and WFD are not quite popular in the bin packing literature. A rare exception is the application of WFD(m) as part of the MB approach for the version of the one-dimensional BP problem with an additional constraint on the number of items per bin. Proposed and analyzed by Krause et al. (1975), the algorithm was later cited under the name *Iterated Lowest Fit Decreasing*; see Coffman et al. (1984). In this work, we produce complementary results on non-monotonicity of WF(m) and WFD(m); see Section 6.

The domain where WF(m) and WFD(m) are popular and recognized as successful approaches is scheduling, namely, the problem of allocating n jobs to m parallel identical machines in order to minimize the makespan. The one-dimensional versions of WF(m) and WFD(m) can be seen as the *List Scheduling* algorithm and the *Longest Processing Time* (LPT) algorithm, both analyzed by Graham (1969). Note that LPT is one of the most popular algorithms used by practitioners.

Finally, although FFD(m) is not useful as part of the MB approach for solving VBP according to Statement 1, it plays the key role in solving the parallel machine scheduling problem as part of the famous *Multifit* algorithm proposed by Coffman et al. (1978) for solving the makespan minimization problem on m parallel machines. *Multifit* uses binary search, calling algorithm FFD(m, C) for a series of trial values of the makespan C, where C corresponds to the bin capacity in the BP interpretation. Algorithm FFD(m, C) is non-monotonic with respect to C, as shown by Coffman et al. (1978): it can find a feasible solution for some C but fails for a larger value of C. Contrarily, in the BP context, FFD(m, C) is monotonic with respect to m: if the algorithm succeeds to solve VBP(m) for some m, then it uses the same number of bins even if more than m bins have been activated initially, providing a solution for VBP($m+1$).

6. Monotonic and anomalous behavior of the algorithms for BP(m) and VBP(m)

Problems BP(m) and VBP(m) are defined with parameter m for a given number of preactivated bins. As established in Section 5.2, only some algorithms benefit from preactivating multiple bins, namely, WF(m), WFD(m), BF(m), BFD(m) and Pairing(m), with appropriately chosen size measures and scores; see Statement 1. In this section, we study whether the listed algorithms have monotonic behavior or exhibit anomalies when the number of activated bins m changes. We define the terms monotonicity and anomaly in Definition 2.

Note that the known results in the literature on non-monotonic behavior of the bin-packing heuristics are related to the item-centric applications in the one-dimensional case. In particular, each of the algorithms BF, BFD, FF, FFD, WF and WFD may use fewer bins when applied to a dominating instance compared to the application of the same algorithm to an instance which is smaller in the sense of the number of items and/or their sizes; see, e.g., Murgolo (1988) for a summary of the results.

For the multi-bin approaches we consider, there are m preactivated bins, a given set of items to pack and algorithm $\mathcal{A}(m)$ for solving VBP(m). We declare that $\mathcal{A}(m)$ *succeeds* if it finds a feasible allocation of the items to m bins and *fails* if it cannot find a feasible allocation.

Definition 2. Algorithm $\mathcal{A}(m)$ is monotonic if for any instance and any positive integers m' and k,

(i) whenever $\mathcal{A}(m')$ succeeds, $\mathcal{A}(m'+k)$ succeeds as well,

(ii) whenever $\mathcal{A}(m')$ fails, $\mathcal{A}(m'-k)$ fails as well (here $m' > k$).

Algorithm $\mathcal{A}(m)$ is anomalous if there exists an instance such that $\mathcal{A}(m')$ succeeds for some $m' > 0$, but $\mathcal{A}(m'+k)$ fails.

Let m^* be the smallest number of bins m, for which an algorithm employed for solving VBP(m) finds a feasible solution. If the algorithm is monotonic, then the *binary search* bin activation strategy is guaranteed to find m^*, and the *batched* strategy is guaranteed to find m^* within a given accuracy bound. If an algorithm is anomalous or its monotonicity is not established, then the *incremental* bin activation strategy should be used for finding m^*.

It appears that the behavior of multi-bin activation algorithms depends essentially on the dimension of an instance. Our two main results are formulated as Theorems 3 and 4, and they are proved in Appendices A and B, respectively. In these theorems, we consider only the algorithms listed in Statement 1, excluding from consideration those which do not benefit from multiple bin preactivation.

Theorem 3. *In the one-dimensional case, the following algorithms are monotonic:*

- *WF(m) and WFD(m) with ℓ_∞, ℓ_1 and ℓ_2 size measures of residual capacity;*
  BF(m) and BFD(m) with the ℓ_2 size measure of bin load,
- *Pairing(m) with ξ-scores computed as Dot-Product 1-3 and Normalized Dot-Product, assuming the weights w_h are non-negative, $1 \le h \le d$.*

Theorem 4 establishes that the majority of the listed algorithms are non-monotonic in the multidimensional case, in contrast to their monotonic behavior in the one-dimensional case. We assume that the size measures are static. We believe that the presented results also hold for dynamic weights, but we do not have formal proofs for that.

Theorem 4. *Algorithms WF(m), WFD(m) with the (static) ℓ_1, ℓ_2 and ℓ_∞ size measures, and BF(m), BFD(m) with the (static) ℓ_2 size measure of bin load are non-monotonic if $d \ge 2$.*

Algorithm Pairing(m) with the scores computed as Dot-Product 1 (with static weights) is non-monotonic. The same is true for the static version of the Normalized Dot-Product scores.

We leave the question about (non)monotonicity of Pairing(m) with the scores computed as Dot-Product 2 or 3 as open. Note that in our computational experiments performed on a broad variety of instances we have not observed non-monotonic behavior of Pairing(m) with Dot-Product 2 and Dot-Product 3 scores, while there have been rare cases of non-monotonic behavior of Pairing(m) with Dot-Product 1 score. The details of our experiments are discussed in Section 8.

7. Theoretical analysis of the multi-bin activation algorithm

In this section we estimate the accuracy of the multi-bin activation algorithm, if it uses WF(m) or WFD(m). We are interested in finding out whether there exists a factor $\rho(d)$ such that algorithms WF(m) and WFD(m) are guaranteed to successfully pack all items when $m \ge \rho(d) \cdot OPT$, where OPT is the minimum number of bins for a given instance, and $\rho(d)$ may depend only on the dimension, but not on the number n of items. We show that for WF(m) no factor $\rho(d)$ exists, while for WFD(m)

$$d/2 \le \rho(d) \le 2d, \tag{5}$$

assuming a forward bin capacity measure is used and all dimensions have weights $w_h = 1$, $h = 1, \dots, d$.

Theorem 5. *Assume that WF(m) uses a forward bin capacity measure. For $d \ge 1$, there is no factor $\rho(d)$ (which may depend only on the dimension, but not on the number n of items) such that WF(m) is guaranteed to successfully pack all items when $m \ge \rho(d) \cdot OPT$.*

Proof. Assume for a contradiction that there is such a factor $\rho(d)$. Consider any $K > 1$ and the following list of items: $(\rho(d) \cdot K)$ *tiny* items of size $s_{ih} = 1/(\rho(d) \cdot K)$ in each dimension $h = 1, \dots, d$, followed by $(K-1)$ *large* items of size $s_{ih} = 1$ in each dimension. Observe that $OPT = K$ as all tiny items fit into one bin and there are $K-1$ large items.

Run WF(m) for $m = \rho(d) \cdot OPT = \rho(d) \cdot K$. The algorithm packs one tiny item into each of the m bins and then it fails to pack any of the large items. This is a contradiction to the assumption that WF(m) successfully packs all items into $m = \rho(d) \cdot OPT$ bins. ■

Consider now WFD(m). For norms ℓ_1 and ℓ_2 the algorithm always gives preference to an empty bin, if one exists. For norm ℓ_∞ an algorithm may wrongly select a non–empty bin B_k if its residual capacity computed as ℓ_∞-norm is 1. This happens if the total size of all allocated items in one dimension is 0. Therefore, we assume the tie-breaking rule for the ℓ_∞ size measure which always selects an empty bin if one exists.

Theorem 6. *Assume that WFD(m) uses a forward bin capacity measure and all dimensions have unit weights ($w_h = 1$, $h = 1, \dots, n$). If $m \geq 2d \cdot OPT$, then WFD(m) successfully packs all items.*

Proof. Let $v(i)$ denote the combined size of item i computed as ℓ_1-, ℓ_∞- or ℓ_2-norm; see Table 2. For a bin B_k that already contains a set of items $\mathcal{I}_k$, define the total load of the bin as

$$\lambda_1(B_k) = \sum_{h=1}^{d} \sum_{i \in \mathcal{I}_k} s_{ih} \qquad \text{for } \ell_1 - \text{norm},$$

$$\lambda_\infty(B_k) = \max_{1 \leq h \leq d} \left\{ \sum_{i \in \mathcal{I}_k} s_{ih} \right\} \qquad \text{for } \ell_\infty - \text{norm},$$

$$\lambda_2(B_k) = \sum_{h=1}^{d} \left(\sum_{i \in \mathcal{I}_k} s_{ih} \right)^2 \qquad \text{for } \ell_2 - \text{norm}.$$

Note that

$$\lambda_1(B_k) \geq \lambda_\infty(B_k), \tag{6}$$

$$\lambda_1(B_k) \geq \left(\lambda_2(B_k)\right)^{1/2}. \tag{7}$$

In the proof we use the following estimate on the optimal number of bins OPT:

$$OPT \geq \frac{1}{d} \sum_{h=1}^{d} \sum_{i \in \mathcal{I}} s_{ih}, \tag{8}$$

which follows from (3).

Assume for a contradiction that there is an input for which WFD(m) cannot successfully pack all items into m bins for some $m \geq 2d \cdot OPT$. Let i be the first item that WFD(m) cannot pack. We distinguish between two cases, depending on the size of item i.

Case 1: item i is 'large', which is defined as $v(i) \geq \frac{1}{2}$ for norms ℓ_1 and ℓ_∞, and $v(i) \geq \frac{1}{4}$ for norm ℓ_2. Then each of the m bins already contains at least one item j with $v(j) \geq v(i)$ (because j was packed before i).

Norm ℓ_1. For any bin B_k, $1 \leq k \leq m$,

$$\lambda_1(B_k) \geq \frac{1}{2}. \tag{9}$$

Therefore, the combined ℓ_1-size of all items is no less than the combined size of all packed items over all m bins plus the ℓ_1-size of item i, so that

$$\sum_{h=1}^{d} \sum_{i \in \mathcal{I}} s_{ih} \geq m \cdot \frac{1}{2} + v(i) > m \cdot \frac{1}{2}.$$

By (8), we get $OPT > m/(2d)$, a contradiction to $m \geq 2d \cdot OPT$.

Norm ℓ_∞. For any bin B_k, $1 \leq k \leq m$,

$$\lambda_\infty(B_k) \geq \frac{1}{2},$$

which by (6) implies (9). The remaining part of the proof is the same as that for norm ℓ_1.

Norm ℓ_2. For any bin B_k, $1 \leq k \leq m$,

$$\lambda_2(B_k) \geq \frac{1}{4},$$

which by (7) implies (9). Again, the remaining part of the proof is the same as that for norm ℓ_1.

Case 2: item i is 'small', which is defined as $v(i) < \frac{1}{2}$ for norms ℓ_1 and ℓ_∞, and $v(i) < \frac{1}{4}$ for norm ℓ_∞. This means that, for any norm, $s_{ih} < \frac{1}{2}$ for all h, $1 \leq h \leq d$. As i does not fit into any of the m bins, each of the m bins must have a load greater than $\frac{1}{2}$ in at least one dimension, and hence $\sum_{h=1}^{d} \sum_{i \in \mathcal{I}_k} s_{ih} \geq \frac{1}{2}$, or, equivalently, (9) holds for each bin B_k. The remaining part of the proof is the same as that for Case 1. ■

An algorithm for VBP has *approximation ratio* ρ if it uses at most $\rho \cdot OPT$ bins on any VBP instance, where OPT denotes the optimal number of bins for that instance. It has *asymptotic approximation ratio* ρ if it uses at most $\rho \cdot OPT + c$ bins for some constant c that is independent of the instance.

Corollary 7. *The multi-bin algorithm that uses WFD(m) with binary search over m has approximation ratio of at most 2d. This holds for ℓ_1, ℓ_2 and ℓ_∞ size measures.*

Proof. As WFD(m) will pack the items successfully for all $m \geq 2d \cdot OPT$, the binary search is guaranteed to terminate with a value of m that is at most $2d \cdot OPT$. ■

Example 1. We show that $\frac{d}{2} \cdot OPT$ bins are not always sufficient for WFD(m) to be able to pack all items. We give an example for $d = 2$, but generalizing it to larger values of d is straightforward. The example works for ℓ_1, ℓ_2 and ℓ_∞ size measures. Consider the following set of items:

- $(1, 0)$ (ℓ_1-size 1, ℓ_∞-size 1, ℓ_2-size 1)
- $(0, 1)$ (ℓ_1-size 1, ℓ_∞-size 1, ℓ_2-size 1)
- $(\varepsilon, \varepsilon)$ (ℓ_1-size 2ε, ℓ_∞-size ε, ℓ_2-size $2\varepsilon^2$)

These items can be packed into 2 bins: the first two items can be packed together into bin 1, and the last item into bin 2. Hence, $OPT = 2$.

WFD(2) starts by packing the first two items into two separate bins and then it fails to pack the last item. Therefore $\frac{d}{2} \cdot OPT = 2$ bins are not sufficient.

The upper bound of $2d \cdot OPT$ bins and the lower bound of $\frac{d}{2} \cdot OPT$ bins are a factor of 4 apart. We leave as an open question whether both the lower and the upper bounds can be improved.

8. Computational experiments

In the previous sections we described the main three classes of heuristics for VBP: item-centric, bin-centric and multi-bin activation. In this section we analyze their performance empirically.

Note that analytical results are available mostly for the one-dimensional case: for item-centric and bin-centric approaches asymptotic approximation ratios are 11/9 for BFD and FFD (Coffman et al., 2013), and approximation ratios are 1.7 for BF and FF (Dosa and Sgall, 2013, 2014). For the d-dimensional case, asymptotic approximation ratios are known only for FF and FFD, namely, $d + 0.7$ and $d + 1/3$ (see Garey et al., 1976), and also for several special algorithms, which are not closely related to item-centric and bin-centric approaches (see the survey paper by Christensen et al. (2017)). Recall that in Section 7 it is shown that the multi-bin activation approach has an approximation ratio of at most $2d$.

The VBP algorithms discussed in this paper are implemented as a C++ library called *Vectorpack*. It contains all versions of the item-centric, bin-centric and multi-bin approaches. Taking into account different size measures of items and bins, scores for item-bin pairs and expressions for dimension weights, there are, in total, 351 algorithms and their variations in *Vectorpack*: 209 item-centric, 66 bin-centric and 76 multi-bin activation algorithms. The library is publicly available on GitHub[1] under the LGPL-3.0 license and it is open for further extensions. All experimental data, including benchmark generation scripts, the summaries of the results and illustrative figures, are available in a companion repository on GitHub.[2] The total number of instances tested is 2940.

[1] https://github.com/Vectorpack/Vectorpack_cpp.

[2] https://github.com/Vectorpack/experiments_vector_paper.

8.1. Benchmark instances

In our computational experiments we use 3 types of benchmark instances, which we denote as *Panigrahy*, *Triplet* and *New*. Recall that the algorithms in the previous sections are presented for the bins of unit capacities in all dimensions and item sizes normalized in $[0, 1]$. The instances used in experiments follow the setups from the original papers where they were introduced. While we observe those setups, the data are then normalized in order to apply the algorithms with size-measures and item-bin scores specified in Tables 2–3.

The benchmark set *Panigrahy* consists of 9 classes of instances, proposed by Panigrahy et al. (2011) as an extension of well-known classes of one-dimensional instances introduced by Caprara and Toth (2001). Following the same setup, we generated 10 instances for each class and each combination of the parameters d and n, with the number of dimensions $d \in \{3, 5, 10\}$ and the number of items $n \in \{20, 40, 60, 120, 250, 500\}$. In total, there are $10 \times 9 \times 3 \times 6 = 1620$ instances of the *Panigrahy* type.

The benchmark set *Triplet* includes instances with 'triplets' of items, where each triplet perfectly fills up a bin. We perform experiments with 2 classes of Triplet instances: instances of type *Triplet* Class F are generated following the original procedure described by Falkenauer (1996), while instances of type *Triplet* Class CT are generated following the procedure described by Caprara and Toth (2001) for their Class 10. We generated 10 instances for each class and each combination of $d \in \{3, 5, 10\}$ and $n \in \{60, 120, 249, 501\}$. In total, there are $10 \times 2 \times 3 \times 4 = 240$ instances of the *Triplet* type.

The benchmark set *New* consists of 6 newly introduced classes of instances: Class 1 is characterized by item sizes which are less uniform and more heterogeneous compared to the *Panigrahy* and *Triplet* benchmarks; Classes 2 and 3 have small-size items, and Classes 4, 5 and 6 consist of items of mixed sizes. In all instances of the *New* set, the bin capacity is set to 100 for each dimension.

All instances in *New* Class 1 contain d groups of items. The items in group h have large sizes in dimension h ($1 \le h \le d$) and small sizes in all other dimensions. There are n/d items in each group, except for the last one, which may have additionally $(n \mod d)$ items, if n is not divisible by d. The item sizes are randomly picked following a uniform distribution in the range $[50, 100]$ for a large size, and in the range $[0, 25]$ for a small size.

The instances of *New* Classes 2 and 3 contain items that are small in every dimension. Item sizes of Class 2 are randomly selected following the normal distribution with the mean $\mu = 20$ and standard deviation $\sigma = 10$. Item sizes of Class 3 follow the same distribution with $\mu = 30$ and $\sigma = 15$.

The instances of *New* Classes 4, 5 and 6 contain items of two types: items that are large in every dimension and items that are small in every dimension. The number of large items in each instance is randomly chosen from $\{0, 1, \ldots, n\}$ following the normal distribution with parameters $\mu = n/10$ and $\sigma = \frac{1}{2}\sqrt{n}$. The sizes of large and small items in each dimension are randomly selected following the normal distribution with different parameters, specific for each class: for Class 4, $\mu_{\text{large}} = 70$, $\mu_{\text{small}} = 30$, $\sigma = 15$; for Class 5, $\mu_{\text{large}} = 80$, $\mu_{\text{small}} = 20$, $\sigma = 10$; for Class 6, $\mu_{\text{large}} = 70$, $\mu_{\text{small}} = 15$, $\sigma = 10$.

We generated 10 instances for each class of the *New* type and each combination of $d \in \{3, 5, 10\}$ and $n \in \{20, 40, 60, 120, 250, 500\}$, so that, in total, there are $10 \times 6 \times 3 \times 6 = 1080$ instances.

8.2. Evaluation methodology and metrics

We run each of the 351 versions of the VBP algorithms on every instance of the three benchmarks and, for each combination of algorithm and instance, we store the number of bins of the solution found and the time taken to find the solution. The experiments are performed on a single core of a machine equipped with one Intel Xeon Gold 6138 CPU having 8 cores, with 16 GB of memory per core.

To evaluate the performance of the algorithms, we use a primary metric ϱ defined as the average percentage error of the solution found relative to the optimal solution, when known, or to the lower bound. A summary of available optimal solutions is presented in Appendix C. Lower bounds are found as the maximum of the trivial lower bound computed by (3) and a clique-based lower bound by Gurski and Rehs (2020). For the latter, we use the heuristic max-clique algorithm by Johnson (1974) applied to the graph defined by Gurski and Rehs (2020).

As a secondary metric, we compare the computation times of the algorithms.

8.3. Algorithm naming

The names of item-centric algorithms combine the heuristic name from Table 1: FF, FFD, BF, BFD, WF, WFD, and the metric name of the size measure from Table 2: Linf, L1, L2 and L2Load. Recall that the same type of metric is used for both, item sizes $v(i)$ and residual capacities of the bins $v(B_k)$, if using algorithms BF, BFD, WF, WFD.

The names of bin-centric algorithms have prefix BC followed by the score type from Table 3: DP1, DP2 and DP3 for Dot-Product 1, 2 and 3, NormDP for Normalized Dot-Product, L2Slacks for ℓ_2 Norm of Slacks, TFSum for Tight Fill with Sum, and TFMin for Tight Fill with Min.

The names of the multi-bin activation algorithms start with MB-WFD, MB-BFD or MB-Pairing, followed by the algorithm specifics: the size measure (for MB-WFD and MB-BFD) or the score formula (for MB-Pairing).

Each algorithm name has an ending indicating the weights for dimensions, as defined in Table 4: Unit, Avg, Expo, RecipAvg and Util-Ratio. By default, the static item-based weight formulae from Table 4 (first column) are used. Additional suffixes '-Dyn' and '-Bin' indicate dynamic item-based (second column) and bin-based versions (third column) of the weight formulae. Note that the bin-based weights are used only in conjunction with the BC and MB-Pairing algorithms.

In addition, special notations are added to algorithms BFD and WFD to specify which types of weights for dimensions are used. Type 'T1' indicates that the same item-based weights are used for ordering items and bins (first or second column of Table 4). Type 'T2' indicates that item-based weights are used for items (first or second column) and bin-based weights are used for bins (third column), with both expressions taken from the same row of Table 4. Type 'T3' indicates a scenario similar to 'T2', but with the two expressions taken from different rows of Table 4. For example, algorithm 'WFD-T3-Linf-Avg-Dyn-RecipAvg' denotes the version of WFD of type T3, in which item sizes $v(i)$ are computed as the ℓ_∞ size measure with Average dimension weights $w_h = d^*_h$ updated dynamically, while residual bin capacities $v(B_k)$ are computed as the ℓ_∞ size measure with Reciprocal Average weights $w_h = 1/r_h$. Algorithms MB-WFD and MB-BFD are based on their item-centric counterparts but only for the type 'T1' weights, and thus we omit the 'T1' in their names.

8.4. Analysis of computational results

8.4.1. Preliminary notes

In our experiments, we observed that the item-centric algorithms BF and WF were often outperformed by a simpler and faster algorithm FF, regardless of the measure or weights formulae used. However, in a small proportion of instances (less than 100 instances out of the 2940), some BF and WF versions found better solutions not only in comparison with FF, but even in comparison with the usually best performing algorithms BC-DP1 and MB-Pairing-DP3. Since superior performance of BF and WF is rather rare, we do not show their results in the summary diagrams presented in this section.

We eliminate from our analysis the multi-bin activation algorithms which use the *incremental* or *batched* bin activation strategies: the incremental method leads to extensive computation times, while the

batched method is hard for tuning when choosing the increments in m-values. In the summary diagrams we only keep multi-bin algorithms in combination with the *binary search* strategy. In spite of the potential anomalies of WFD(m), BFD(m) and Pairing(m) discussed in Section 6, the actual performance of these algorithms combined with binary search is often fairly good.

Generally speaking, for each family of algorithms, there is no algorithm consistently outperforming the others. Moreover, the best performing algorithms vary for different classes of the three types of benchmarks. Looking for consistency in algorithms' behavior, we selected 4 representative classes. Instances of *Panigrahy* Class 5 typically have solutions with 14 items per bin on average, with item sizes between 25 and 100 and a bin capacity of 1000 in every dimension. Instances of *Panigrahy* Class 8 are characterized by negative correlation of item sizes in every two consecutive dimensions, and solutions contain 2 items per bin on average. Instances of *New* Class 5 have about 10% of large items and 90% of small items, and solutions contain 3 items per bin on average. Finally, we selected *Triplets* Class F instances constructed by Falkenauer's generation procedure: for any instance, an optimum solution is characterized by triplets of items which perfectly fit a bin without any gaps; see Section 8.1. Note that an optimal bin number is known for any *Triplet* instance (it is equal to the lower bound (3)). Contrarily, for the instances of *Panigrahy* Class 5 and *New* Class 5 optimal solutions are known only for small size instances; see Appendix B.

In the following, we present the results for the 4 selected classes of instances. For completeness, we keep the summary figures for other classes of instances in the companion GitHub repository.[3]

8.4.2. Comparing heuristics with unit weights

When all dimensions have unit weights, we are left with the 26 algorithms, which we group into 'families' as follows:

Item-Centric

(a) FF (used as a reference to what is achievable by the fastest and simplest method);

(b) FFD, with 3 expressions for $v(i)$ from Table 2;

(c) BFD, with 4 pairs of expressions for $v(i)$ and $v(B_k)$ from Table 2;

(d) WFD, also with 4 pairs of expressions for $v(i)$ and $v(B_k)$;

Bin-Centric

(e) 6 versions, with scores ξ_{ik} defined in Table 3, with the exception of DP3, which in the case of unit weights reduces to DP1;

Multi-Bin Activation

(f) MB-Pairing, with 4 scores ξ_{ik}, namely DP1, DP2, DP3 and NormDP (the remaining 3 scores from Table 3 are excluded from consideration by Statement 1);

(g) MB-WFD, with 3 pairs of expressions for $v(i)$ and $v(B_k)$ from Table 2, namely Linf, L1 and L2 (the pair of expressions from the last row of Table 2 is excluded from consideration by Statement 1), and also MB-BFD, with L2Load size measure for $v(i)$ and $v(B_k)$ (with remaining pairs of expressions excluded by the same statement).

Figs. 2–3 show the average percentage error ϱ of the algorithms as a function of the number of items n. For readability of the figures, each family of algorithms is represented by the best performing algorithm. Interestingly, best performing algorithms of types FFD, BFD and WFD produce very similar results and they are often represented by the same plot in the figures.

For instances of *Panigrahy* Class 5 (Fig. 2(a)), the best multi-bin activation algorithm (MB-Pairing-NormDP-Unit) and the best bin-centric algorithm (BC-L2Slacks-Unit) outperform other algorithms. Their average ϱ-values, taken over all instances with different d and n, are $\varrho = 2.8$ and $\varrho = 2.9$, respectively. The next successful algorithm is from the remaining multi-bin activation family (MB-WFD-L2-Unit), with $\varrho = 5.5$. The best item-centric algorithms conclude the list, with FFD-L2-Unit, BFD-T1-L2Load-Unit and WFD-T1-L2Load-Unit, all having $\varrho = 5.9$. Interestingly, the simplest item-centric algorithm FF performs quite similarly compared to more sophisticated item-centric algorithms.

For instances of *Panigrahy* Class 8 (Fig. 2(b)), the best multi-bin activation algorithm (MB-WFD-Linf-Unit) achieves a slightly better accuracy compared to best item-centric algorithm, with $\varrho = 1.5$. The remaining item-centric algorithms consistently demonstrate slightly larger ϱ, but still they outperform the multi-bin activation algorithms of type MB-Pairing (with $\varrho = 3.2$ for MB-Pairing-DP3-Unit) and the family of bin-centric algorithms (with $\varrho = 3.6$ for BC-DP1-Unit). FF consistently demonstrates a worse performance.

For instances of *New* Class 5 (Fig. 3(a)), the best algorithms in each family perform similarly, with a slight advantage for the bin-centric and MB-Pairing families. Again, FF consistently demonstrates a worse performance. Contrary to the two previous classes, the ϱ -values for *New* Class 5 significantly depend on the number of dimensions d. Specifically, considering best performing algorithms in each family, $\varrho \in [6.5, 7.4]$ when $d = 3$, $\varrho \in [10.8, 11.6]$ when $d = 5$, and $\varrho \in [17, 17.6]$ when $d = 10$.

Finally, for instances of *Triplet* Class F (Fig. 3(b)), almost all algorithms perform similarly with $\varrho = 17.8$, except for one outlier — MB-Pairing-DP1-Unit. It achieves a lower average error $\varrho = 11.7$ when $d = 3$ and a larger average error $\varrho = 22.5$ when $d = 10$.

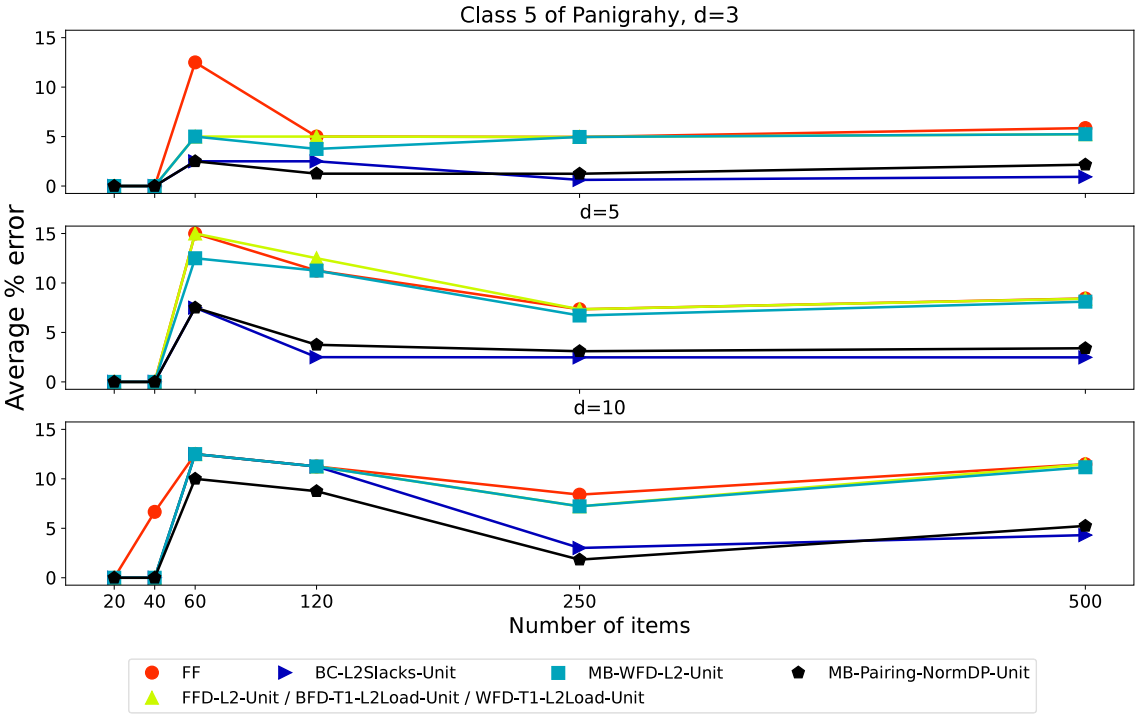
Regarding algorithms' running times, the item-centric algorithms are the fastest, followed by the bin-centric algorithms: FF and FFD families of algorithms are able to solve the largest instances in less than 7 ms, BFD and WFD families need at most 15 ms, while the bin-centric algorithms require up to 56 ms. For the multi-bin activation algorithms, the family of MB-WFD and MB-BFD solves the largest instances in less than 105 ms. The running times of the MB-Pairing algorithms are much larger: 12,000 ms on average and up to 32,000 ms for the largest instances.

8.4.3. The impact of replacing unit weights by average weights

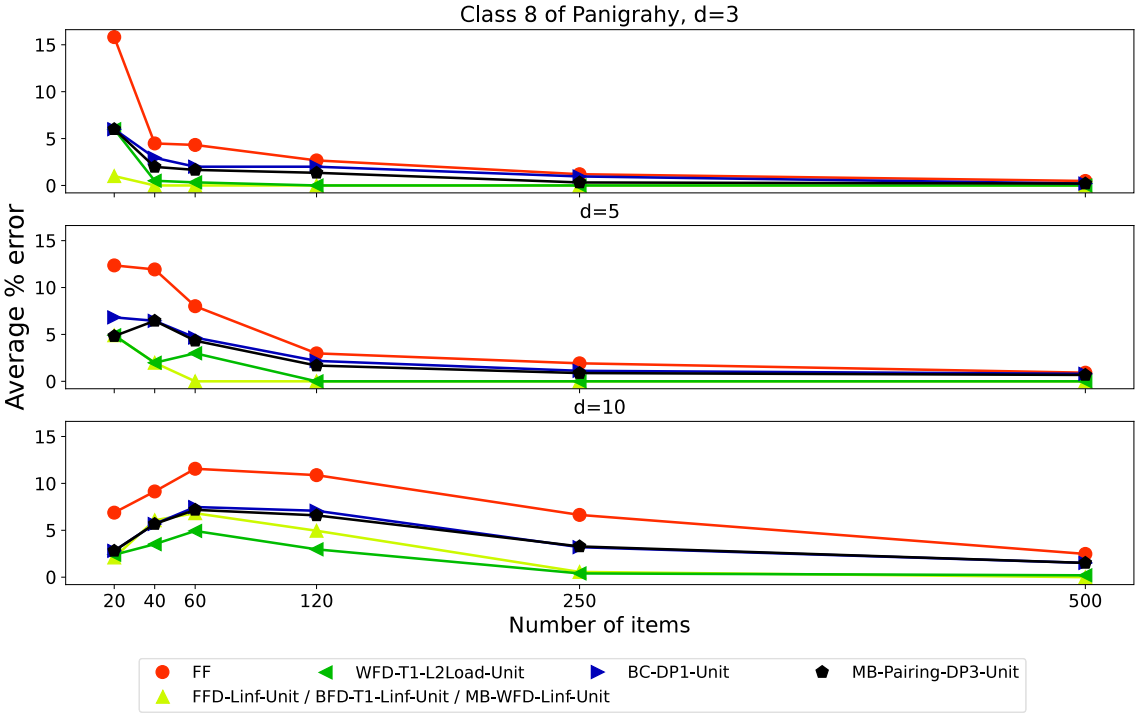
Introduction of Average weights generally improves the performance of the algorithms, and using the dynamic version of Average weights leads to further improvements in the majority of cases. We illustrate this by considering the algorithms identified as the best ones in the previous section for the instances from *Panigrahy* Class 5 restricted to $n = 500$ and $d = 10$. The level of improvements is demonstrated in Fig. 4.

Item-centric algorithms FFD-L2 and WFD-T1-L2Load, which have relatively large $\varrho = 11.5$ in the case of Unit weights, achieve an improvement, with $\varrho = 10.9$ and $\varrho = 11.2$ for the static version of Average weights, and even a further improvement, with $\varrho = 9.3$, for the dynamic version of Average weights. Algorithm BC-L2Slacks, the best-performing bin-centric algorithm for the benchmarks under consideration, improves its ϱ-value from 4.3, for Unit weights, to 4, for both static and dynamic Average weights.

[3] https://github.com/Vectorpack/experiments_vector_paper.



(a) *Panigrahy* Class 5. For $d = 10$, the plot for MB-WFD-L2-Unit mostly coincides with the plots for the group FFD-L2-Unit / BFD-T1-L2Load-Unit / WFD-T1-L2Load-Unit

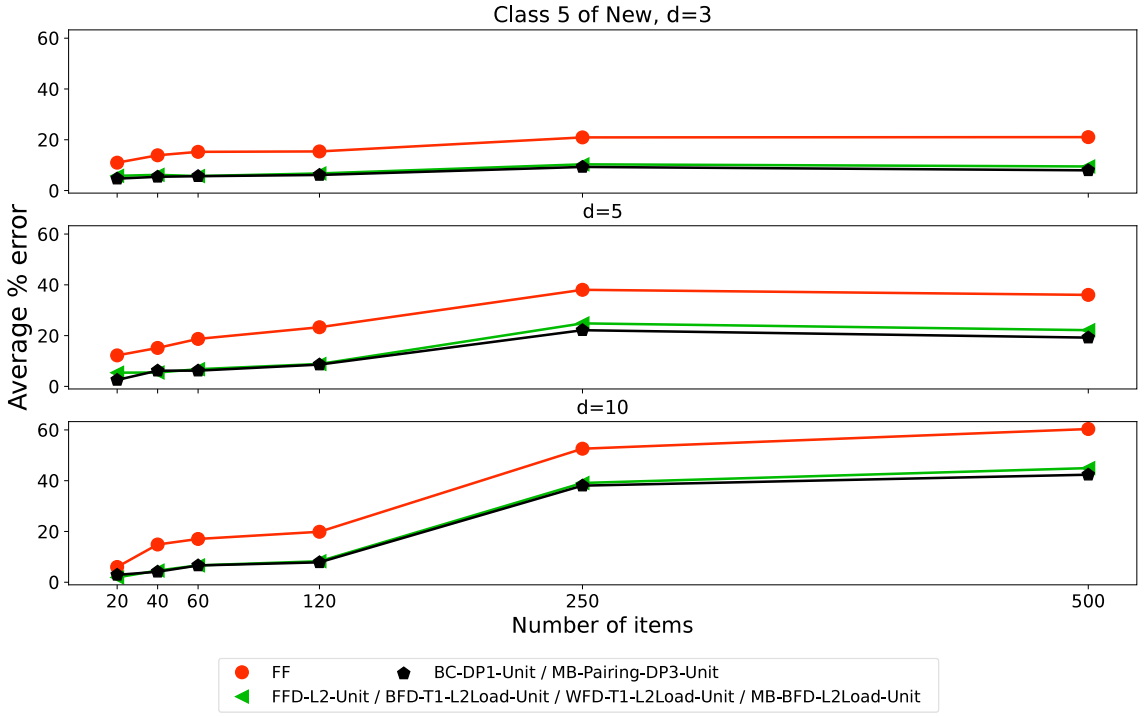


(b) *Panigrahy* Class 8

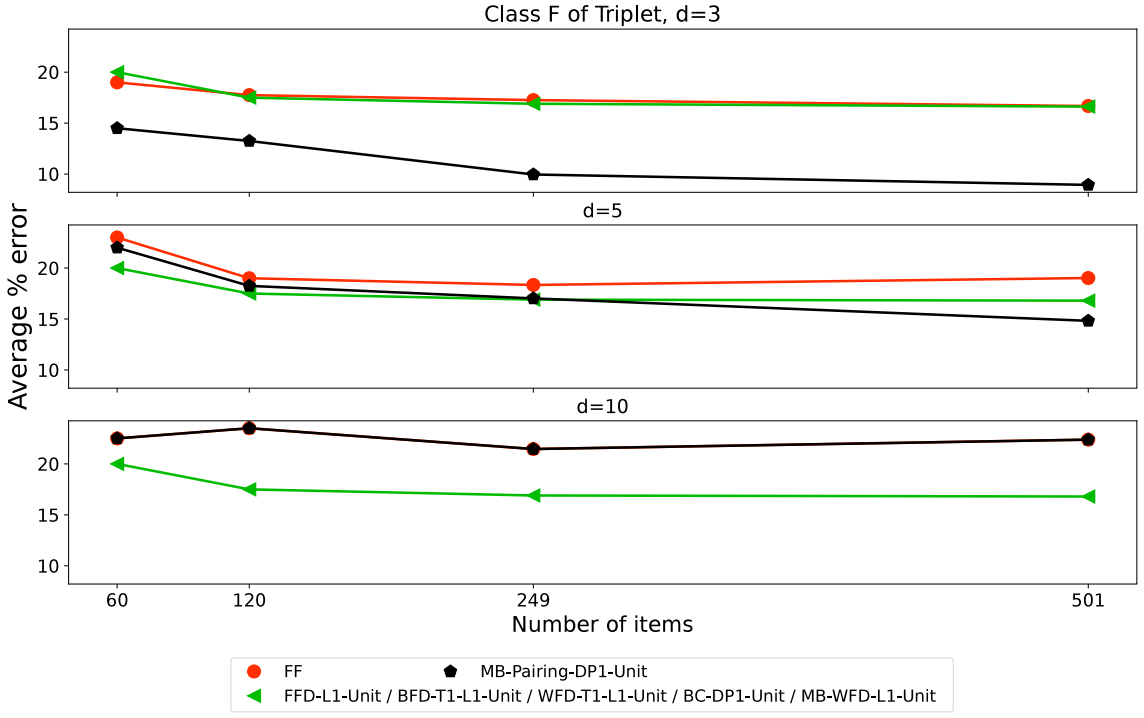
Fig. 2. Performance summary of the best-performing algorithms with Unit weights in each family.

There are rare cases when the introduction of Average weights does not lead to better results, or the improvement is insignificant. For example, there is generally no change in ϱ-values for MB-Pairing-NormDP and a minor change for MB-WFD-L2: $\varrho = 11.2$ for Unit weights and $\varrho = 10.9$ for the static version of Average weights. Interestingly, dynamic version of Average weights MB-WFD-L2-Avg-Dyn is less beneficial than its static version MB-WFD-L2-Avg.

Regarding algorithms' running times, introduction of static Average weights instead of Unit weights does not slow down the algorithms significantly. The situation with dynamic Average weights is different.



(a) *New* Class 5



(b) *Triplet* Class F. For $d = 10$, the plot for MB-Pairing-DP1-Unit mostly coincides with the plot for FF.

Fig. 3. Performance summary of the best-performing algorithms with Unit weights in each family (cont.).

For item-centric algorithms, their running times increase by up to 100 times. Still the largest instances are solvable in less than 100 ms. A similar behavior is observed for MB-WFD and MB-BFD algorithms, with a maximum running time of 382 ms for the largest instances.

On the positive side, using dynamic weights does not essentially affect the running times of the bin-centric algorithms and MB-Pairing. This is explained by the fact that the cost of re-computing the weights in each packing step is dominated by the cost of computing the scores for selecting the next item-bin pair. On average, the best bin-centric algorithms solve the largest instances in under 20 ms, with a maximum of 40 ms, while the best MB-Pairing algorithms solve the largest instances in less than 12,000 ms, with a maximum of 33,000 ms.

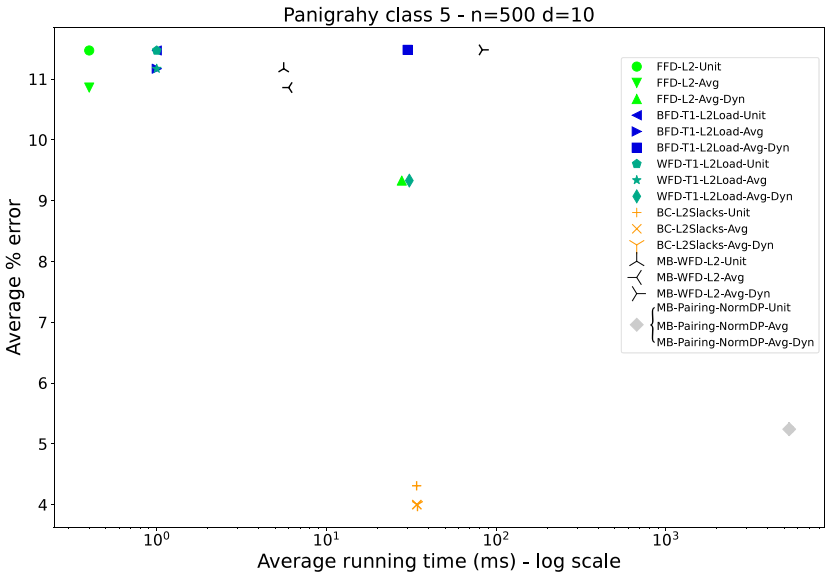


Fig. 4. Impact of Average weights in the experiments for the *Panigrahy* Class 5, with $n = 500$ and $d = 10$.

8.4.4. Comparing heuristics with all types of weights

We now discuss the experiments performed on all 351 variations of algorithms with different types of weights. Recall that for Unit weights we have 26 variations of algorithms.

Similar to the case of Unit weights, there is no algorithm consistently outperforming the others, and the best-performing algorithms vary for different types of instances. The results for the selected four classes of instances are summarized in Figs. 5–6. As before, we select the best performing algorithm in each family of algorithms, where algorithms' families are defined in Section 8.4.2. In this section, each family contains all versions of the relevant algorithms with static item-based weights, dynamic item-based weights and dynamic bin-based weights; see Table 4 for weight formulae. Recall that the complete summary of the results is available in the companion GitHub repository.

For instances of *Panigrahy* Class 5, *Panigrahy* Class 8 and *New* Class 5, the majority of best-performing algorithms use the dynamic versions of Average or Reciprocal Average weights. For instances of *Triplet* Class F, the majority of best-performing algorithms use the Utilization Ratio weights.

We now compare the algorithms within each family. For item-centric algorithms BFD and WFD, the type T2 and type T3 are generally no better than their T1 counterparts; see Section 8.3 for the definitions of T1-T3. The choice of a specific size measure does not significantly affect the performance of BFD, but in the case of WFD the reverse ℓ_2 measure of bin load is beneficial. Note that the latter version of WFD is similar to BFD with the forward ℓ_2 measure, although they occasionally may produce different results.

In the family of bin-centric algorithms, BC-DP1 with any type of dynamic weights is superior, although the dynamic Average weights stand out, followed by L2Slacks.

In the family of MB-Pairing algorithms, although the DP1 or NormDP scores with RecipAvg weights are the best in some cases (e.g., Figs. 5(a) and 6(b)), the DP3 score with static Average weights is superior on average.

Considering the bin-based weights combined with the bin-centric and MB-Pairing algorithms, we observe that they do not generally provide an improvement over the item-based dynamic weights. Recall that bin-based weights are only applicable to bin-centric and MB-Pairing algorithms.

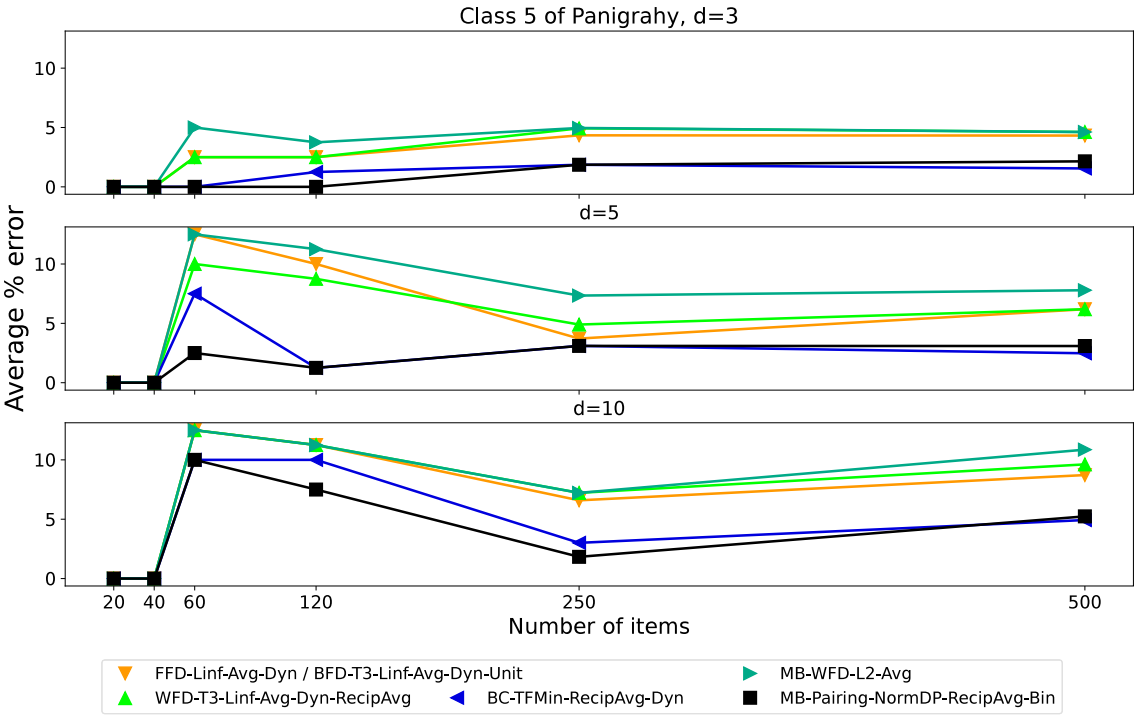
In addition to the previously defined algorithms, we also evaluated the virtual performance of two "meta" algorithms: *meta-centric* executes all versions of item-centric and bin-centric algorithms and keeps the best solution found, and *meta-all* executes all versions of all algorithms, item-centric, bin-centric and multi-bin activation, and keeps the best solution found. The running time of a meta algorithm is defined as the sum of the running times of individual algorithms.

For instances of *Panigrahy* Class 5 (Fig. 5(a)), *meta-centric* has $\varrho = 1.7$ with a running time of at most 6000 ms, and *meta-all* has $\varrho = 1.2$ with a running time of 83,000 ms on average and 130,000 ms at maximum. Since different individual algorithms deliver superior solutions on different instances in the class, each individual algorithm cannot achieve the accuracy of *meta-all* on all instances. In particular, the overall best-performing individual algorithm is MB-Pairing-NormDP-RecipAvg-Bin with $\varrho = 2.1$ and a running time under 6000 ms for the largest instances. Note that the best item-centric algorithm achieves $\varrho = 4.7$ and solves even the largest instances in less than 30 ms, while the best bin-centric algorithm achieves $\varrho = 2.6$ with a maximum running time of 46 ms.

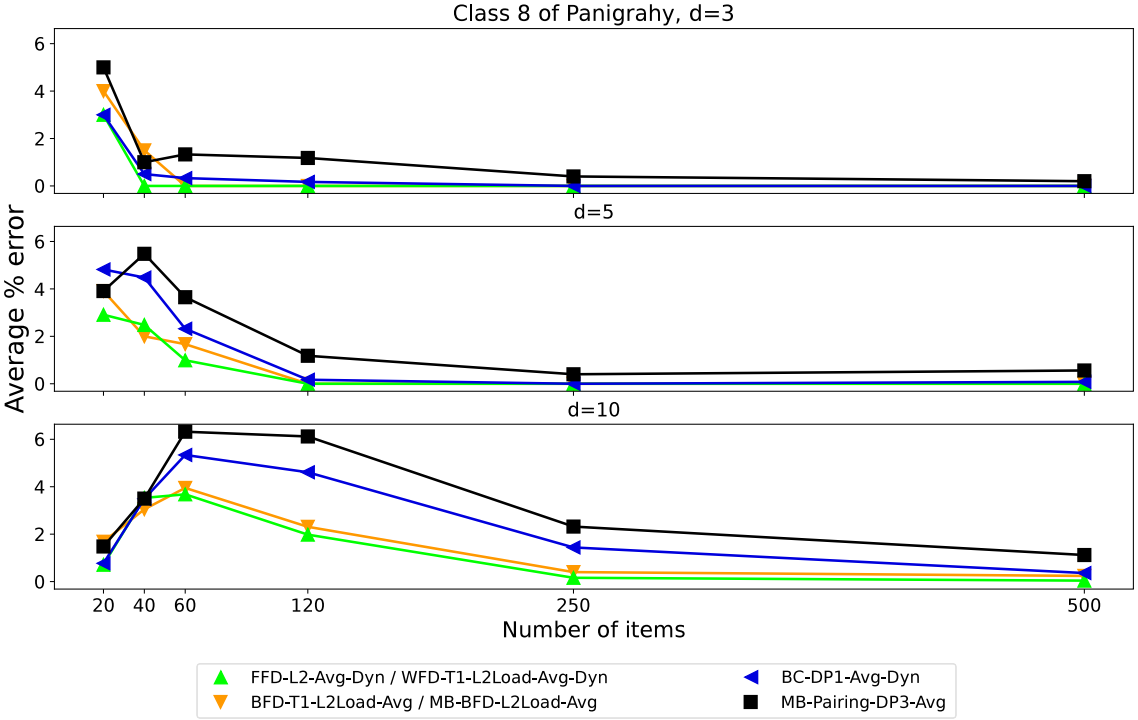
For instances of *Panigrahy* Class 8 (Fig. 5(b)), *meta-centric* has $\varrho = 0.2$ with a running time of 8 s on average, and *meta-all* does not get a further improvement in terms of ϱ, but incurs much more computation time, around 9 min. For comparison, the overall best-performing individual algorithm is FFD-L2-Avg-Dyn with $\varrho = 1$ and a running time of 27 ms.

For instances of *New* Class 5 (Fig. 6(a)), *meta-centric* has $\varrho = 9.2$ with a running time of 9 s on average, and *meta-all* only marginally improves this result with a running time of 10 min on average for the largest instances, and a maximum of 15 min. For comparison, the overall best-performing individual algorithm is BC-DP1-Avg-Dyn with $\varrho = 11.1$ and a running time of 26 ms.

For instances of *Triplet* Class F (Fig. 6(b)), *meta-centric* has $\varrho = 16.7$ with a running time of 9 s on average, and *meta-all* has $\varrho = 14.5$ with a running time of about 9 min for the largest instances, and up to 12 min at maximum. Interestingly the best-performing individual algorithm is MB-Pairing-DP1-RecipAvg-Dyn, not included in *meta-centric*. It outperforms the *meta-centric* algorithm, achieving $\varrho = 16.5$ as the average value over all experiments, but at the cost of a larger running time, 15 s. For comparison, the other individual algorithms mostly achieve a similar ϱ of 17.6, with a maximum running time under 60 ms. It is also worth noticing that the performance of the MB-Pairing algorithm is not stable when the number of dimensions changes. The algorithm significantly outperforms item-centric and bin-centric algorithms when



(a) *Panigrahy* Class 5

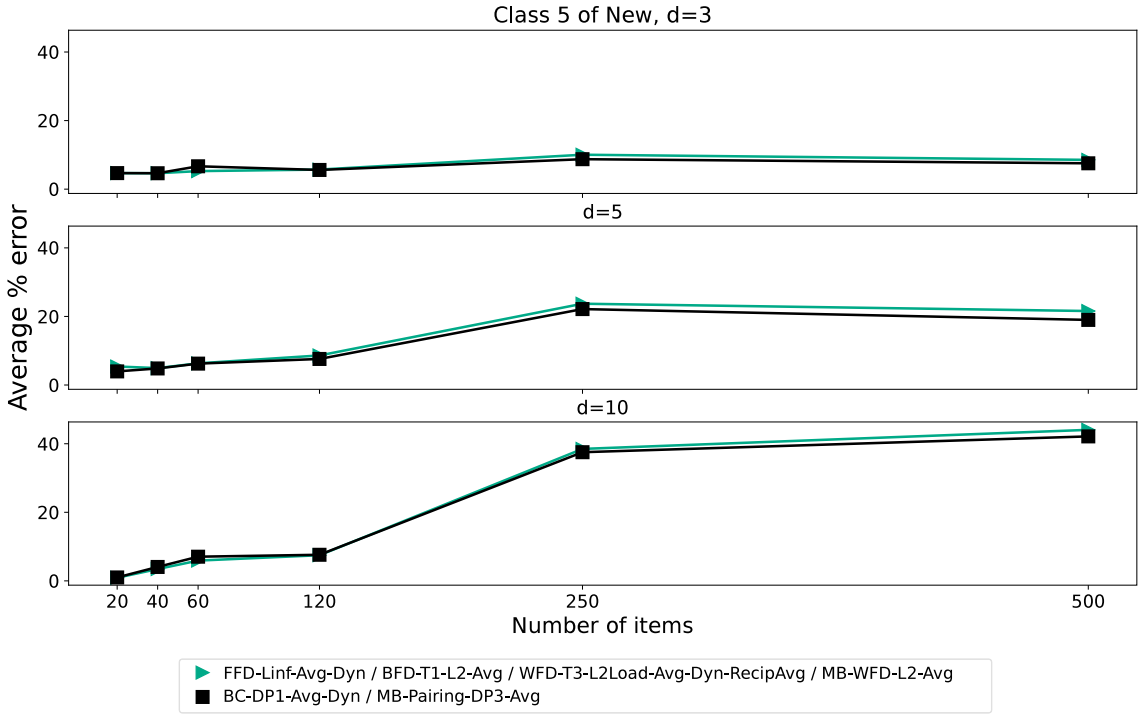


(b) *Panigrahy* Class 8

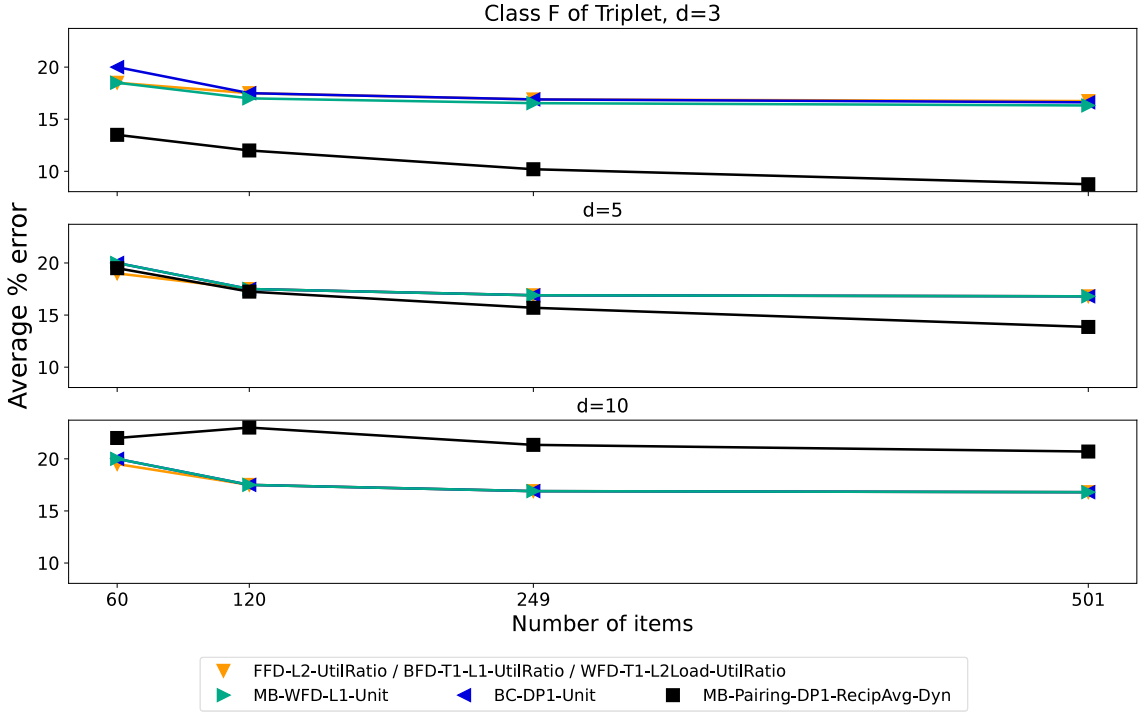
Fig. 5. Performance summary of the best-performing algorithm in each family considering all weights.

$d = 3$, has a similar performance when $d = 5$, but it is outperformed by the others when $d = 10$ (with the exception of the simplest algorithm, FF).

As a summary, we present Fig. 7 to illustrate the trade-off between ϱ-values and running times for best-performing algorithms representing each family. The ϱ-values are averaged over all instances of *Panigrahy* Class 5. We highlight three algorithms: FFD-L1-RecipAvg is very fast but moderately accurate ($\varrho = 10.6$, the running time is 0.5 ms), BC-L2Slacks-Avg-Bin is accurate but with intermediate running time ($\varrho = 4.3$, the running time is 35 ms), and MB-Pairing-DP3-Avg-Dyn is among the most accurate algorithms but with a larger running time ($\varrho = 4$, the running time is 1300 ms).



(a) *New* Class 5



(b) *Triplet* Class F. The plot for BC-DP1-Unit mostly coincides with the plot for the group FFD-L2-UtilRatio / BFD-T1-L1-UtilRatio / WFD-T1-L2Load-UtilRatio

Fig. 6. Performance summary of the best-performing algorithm in each family considering all weights (cont.).

8.4.5. Impact of n and d on algorithms' accuracy

The experiments discussed in the previous sections focus on four classes of instances: two of the *Panigrahy* type, one of the *New* type and one of the *Triplet* type. In this section we summarize our findings for all classes of instances of all types, *Panigrahy*, *New* and *Triplet*.

In the majority of classes, ϱ-values increase as the number of dimensions d increases. There are 4 exceptions: the instances of *Panigrahy* Class 2 and Class 3, which are almost always solved to optimality, the instances of *Panigrahy* Class 6, which were noted as the most difficult instances for the one-dimensional BP, and the instances of

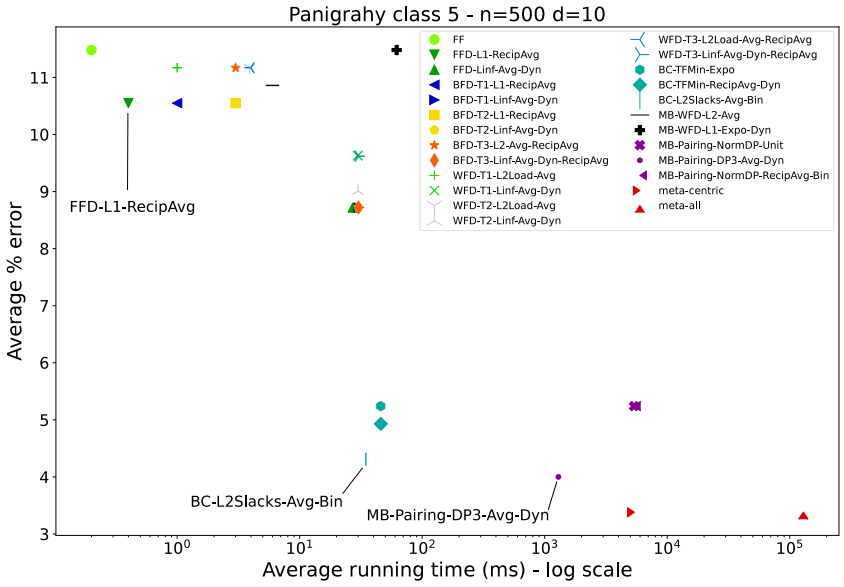


Fig. 7. Performance summary of algorithms for *Panigrahy* Class 5, with $n = 500$ and $d = 10$. The marks for BFD-T1-Linf-Avg-Dyn and BFD-T2-Linf-Avg-Dyn mostly coincide with the mark for BFD-T3-Linf-Avg-Dyn-RecipAvg.

Panigrahy Class 7, where item sizes have positive correlation in every two consecutive dimensions. It is worth noticing that, for these 4 classes, solutions contain on average between 2 and 2.5 items per bin.

For almost all classes of the *Panigrahy* type, ϱ-values are close to 0 when $n = 20$ and they increase with the number of items n. Exceptional instances are those of *Panigrahy* Class 8, for which the opposite is observed (Fig. 5(b)), and *Panigrahy* Class 5, for which there seems to be a plateau for large values of n (Fig. 7).

For the instances of the *New* benchmark, the ϱ-values increase as n increases. That increase is generally more significant in larger dimensions (see Figs. 3(a) and 6(a)). On the contrary, for the instances of the *Triplet* benchmark the ϱ-values tend to decrease as n increases.

Finally, quite naturally, the running times of algorithms increase when n and d increase.

8.4.6. Finding optimal solutions

Among the algorithms with Unit weights, guaranteed optimal solutions were found for 893 instances out of 1620 of the *Panigrahy* type, with 601 of them matching the lower bounds. Note that the actual number of optimal solutions found might be higher, since comparison is done with the lower bound if an optimal solution is not known. For the instances of the *New* type, at least 200 optimal solutions were found out of 1080 instances, with 87 of them matching the lower bounds. For 240 instances of the *Triplet* type, the algorithms did not find optimal solutions.

The algorithms with non-unit weights find a larger number of optimal solutions. Considering all algorithms with all weights, the number of guaranteed optimal solutions found increases to 992 for the instances of the *Panigrahy* type, with 643 of them matching the lower bound, 260 optimal solutions for the instances of the *New* type, with 116 of them matching the lower bound. Still no optimal solutions were found for the instances of the *Triplet* type. Interestingly, the majority of optimal solutions are found by item-centric and bin-centric algorithms, namely, 985 and 257 optimal solutions for the instances of the *Panigrahy* and *New* type, respectively.

8.4.7. Summary of algorithms' evaluation

The experiments performed on instances of different types confirm that among the 351 algorithm variations tested, there is no one which is an ultimate winner. Recommendations can be given for instances with common features, like those grouped in special classes of benchmarks. Depending on an acceptable time limit, practitioners can conduct experiments with the whole collection of algorithms to pinpoint those which are best suited for a given use-case. The results obtained with the two meta-algorithms show that executing all algorithms, returning the best solution found, is consistently successful. The downside of this method is the running time that may become prohibitively large, but on the positive side the method helps in identifying most promising approaches for a class of instances under consideration.

Alternatively, one may consider the following list of the most promising algorithms, selected through our experiments on a variety of benchmarks.

- The bin-centric algorithm with Dot-Product 1 or ℓ_2-Norm of Slacks scores does stand out, especially if combined with static Average weights or dynamic Average weights.
- In the group of multi-bin activation algorithms, MB-Pairing with Dot-Product 3 score and static Average weights demonstrates particularly good performance, comparable to or even better than the best performing bin-centric algorithms.
- In the group of item-centric algorithms, BFD- and WFD-type algorithms are not significantly superior if compared to the FFD-type algorithms.

On a different note, optimal solvers, such as VPSolver by Brandão and Pedroso (2016), are powerful but limited to small- or medium-size instances, when the number of items per bin is not too large; see Appendix B. Larger instances can be solved in reasonable time mostly in the case of $d = 2$; see Spieksma (1994), Caprara and Toth (2001) and Wei et al. (2020). The usage of heuristics is fully justified for large size instances in the multidimensional case.

9. Conclusions

Our work bridges the gap between the body of research on theoretical analysis of Bin Packing and Vector Bin Packing algorithms, and applied work on actual performance of heuristics when addressing real-world applications. The need for applied research on VBP has become particularly apparent with the growing demand of modern distributed systems for effective and efficient algorithms which optimize resource usage, see, e.g., Kumaraswamy and Nair (2019) and Mann (2015). Thus, the main motivation of our study is to analyze empirically traditional and new heuristics for VBP, providing a foundation for future study of enhanced VBP models arising in distributed computing and other areas.

The main contribution of our paper is twofold:

(1) a systematic classification of heuristics for the Vector Bin Packing problem and algorithms' tuning parameters of three types: size measures for items and bins, scores for item-bin pairs, and weights to differentiate dimensions;

(2) evaluation of the algorithms' performance via extensive computational experiments, identifying the most successful algorithms for datasets of different types, global top performers and less promising approaches.

In our study, we use elementary construction heuristics, item-centric and bin-centric, as the basis for developing more complicated multi-bin activation heuristics. Some of the newly introduced algorithms, e.g., MB-Pairing (with the tuning parameters Dot Product 3 for item-bin scores and dynamic Average weights for prioritizing dimensions) or MB-WFD (with the ℓ_1-size measure for items and bins' residual capacities), can achieve superior results compared to the traditional item-centric and bin-centric approaches. Note that algorithms' tuning parameters are collected and put together from multiple sources and enhanced with newly introduced parameters. Further evidence of the power of the proposed multi-bin activation approach is provided in our recent work on optimizing resource provisioning in shared clusters (Mommessin et al., 2023).

We foresee the following directions for future work. First, the proposed multi-bin activation algorithms, presented in the context of the decision problem VBP(m), might be useful for solving the Multiple Multidimensional Knapsack Problem, the decision version of which is the same as VBP(m). Second, the toolkit for the classical VBP problem can be adjusted and extended for solving the versions of VBP with additional features: packing items with restrictions (e.g., conflicts, see Gendreau et al. (2004)), unequal bin capacities, cost constraints (monetary or energy), and various further features arising, for example, in the context of Distributed Computing. Finally, having classified the VBP algorithms and implemented them as the C++ *Vectorpack* library, we consider them as a starting point for the design of more complicated, hybrid algorithms, as well as self-tunable hyperheuristics and Machine Learning algorithms.

CRediT authorship contribution statement

Clément Mommessin: Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Formal analysis, Data curation, Conceptualization. **Thomas Erlebach:** Writing – review & editing, Writing – original draft, Methodology, Formal analysis, Conceptualization. **Natalia V. Shakhlevich:** Writing – review & editing, Writing – original draft, Project administration, Methodology, Funding acquisition, Formal analysis, Conceptualization.

Data availability

The developed algorithms and all experimental data are available at https://github.com/Vectorpack/Vectorpack_cpp, https://github.com/Vectorpack/experiments_vector_paper.

Acknowledgments

This work is supported by the project EP/T01461X/1 "Algorithmic Support for Massive Scale Distributed Systems" funded by the UK Engineering and Physical Sciences Research Council (EPSRC). We are grateful to the reviewers for their valuable comments and recommendations.

Appendix A. The proof of Theorem 3

The first result of Section 6 is Theorem 3 that establishes monotonicity of WF(m) and WFD(m) for ℓ_∞, ℓ_1 and ℓ_2 size measures, monotonicity of BF(m) and BFD(m) for the ℓ_2 size measure of bin load, as well as monotonicity of Pairing(m).

We start with the proof for **WF(m)**. It holds for **WFD(m)** as well since ordering items in a specific way does not affect the proof.

Compare the operation of WF(m) on an instance with n items and the operation of WF(m+1) on the same instance. We call the two scenarios α and β, respectively. The comparison is performed at various timesteps: at timestep 0 no item is packed yet; at timestep j, the first j items have been packed.

Let $r_k^\alpha(j)$ and $r_k^\beta(j)$ denote the residual capacity of the kth bin after timestep j in scenario α and β, respectively. To simplify the analysis, we repeatedly renumber the bins in nondecreasing order of residual capacities every time an item is allocated, so that for each timestep j,

$$r_1^\alpha(j) \le r_2^\alpha(j) \le \cdots \le r_m^\alpha(j),$$
$$r_1^\beta(j) \le r_2^\beta(j) \le \cdots \le r_m^\beta(j) \le r_{m+1}^\beta(j).$$

Without loss of generality we consider a tie breaking rule that always places a current item into the last bin. At each timestep, the current item is allocated to the last bin and that bin "bubbles" forward in the list to observe the residual capacity order.

The monotonicity of WF(m) follows from the property:

$$r_k^\alpha(j) \le r_k^\beta(j), \quad \text{for each timestep } j = 0, 1, 2, \dots, n \text{ and any } k,\ 1 \le k \le m. \tag{10}$$

Here we assume that item j can be feasibly packed by WF in both scenarios α and β.

Clearly, property (10) holds for any $j \le m$, since in either scenario, α or β, each of the j items is placed into a separate bin. Consider timestep $j > m$ and suppose (10) holds for all previous timesteps $0, 1, 2, \dots, j-1$. We prove by induction that (10) holds for j.

Let k' (k'') be the index of the bin containing item j after that item is allocated under scenario α (scenario β) and bins have been renumbered. Then the following useful property holds:

$$r_{k'}^\alpha(j) \le r_{k''}^\beta(j) \text{ for } j > m. \tag{11}$$

Indeed

$$r_{k'}^\alpha(j) = r_m^\alpha(j-1) - s_j,$$
$$r_{k''}^\beta(j) = r_{m+1}^\beta(j-1) - s_j,$$

and additionally

$$r_m^\alpha(j-1) \overset{\text{induction hypothesis}}{\le} r_m^\beta(j-1) \overset{\text{bin numbering}}{\le} r_{m+1}^\beta(j-1),$$

where the first inequality holds by the induction hypothesis and the second inequality holds due to the bin numbering at timestep $j-1$.

We now turn to proving inequality (10) for $j > m$, assuming it is satisfied for $0, 1, 2, \dots, j-1$.

If $k < \min\{k', k''\}$, then

$$r_k^\alpha(j) = r_k^\alpha(j-1) \overset{\text{induction hypothesis}}{\le} r_k^\beta(j-1) = r_k^\beta(j).$$

Table 5
Proof of Theorem 4: item sizes for WF(m) and WFD(m).

Item	a	b	c	d	e	f
s_{i1}	0.485	0.484	0.505	0.495	0.02	0.01
s_{i2}	0.99	0.985	0	0	0.01	0.014
$v(i)$ for ℓ_∞	0.99	0.985	0.505	0.495	0.02	0.014
$v(i)$ for ℓ_1	1.475	1.469	0.505	0.495	0.03	0.024
$v(i)$ for ℓ_2	1.215325	1.204481	0.255025	0.245025	0.00050	0.000296

Similarly, if $k > \max\{k', k''\}$, then

$$r^{\alpha}_{k}(j) = r^{\alpha}_{k-1}(j-1) \overset{\text{induction hypothesis}}{\leq} r^{\beta}_{k-1}(j-1) = r^{\beta}_{k}(j).$$

It remains to prove that inequality (10) holds for any k, $\min\{k', k''\} \leq k \leq \max\{k', k''\}$. We distinguish between the following two cases.
Case 1: if $k'' \leq k'$, then for any k, $k'' \leq k \leq k'$,

$$r^{\alpha}_{k}(j) \overset{\text{bin numbering}}{\leq} r^{\alpha}_{k'}(j) \overset{(11)}{\leq} r^{\beta}_{k''}(j) \overset{\text{bin numbering}}{\leq} r^{\beta}_{k}(j).$$

Case 2: if $k' < k''$, then for any k, $k' \leq k < k''$,

$$r^{\alpha}_{k}(j) \overset{\text{bin numbering}}{\leq} r^{\alpha}_{k+1}(j) = r^{\alpha}_{k}(j-1) \overset{\text{induction hypothesis}}{\leq} r^{\beta}_{k}(j-1) = r^{\beta}_{k}(j),$$

and for $k = k''$

$$r^{\alpha}_{k''}(j) = r^{\alpha}_{k''-1}(j-1) \overset{\text{induction hypothesis}}{\leq} r^{\beta}_{k''-1}(j-1) = r^{\beta}_{k''-1}(j) \overset{\text{bin numbering}}{\leq} r^{\beta}_{k''}(j).$$

The proof for **BF(m)** and **BFD(m)** with the ℓ_2 size measure of bin load is similar to the proof presented above.

Consider now **Pairing(m)** with the scores listed in Theorem 3.

- Pairing(m) with Dot-Product 1 becomes MB-WFD. Indeed, the scores are $\xi_{ik} = s_i r_k$, and the maximum value is achieved for the bin B_k with the largest r_k and for the item i with the largest size s_i, $s_i \leq r_k$. The same holds for Pairing(m) with Normalized Dot Product.
- Pairing(m) with Dot-Product 2 computes the scores $\xi_{ik} = 1$ for all item-bin combinations, so that with an appropriate tie-breaking rule it performs like FF.
- Pairing(m) with Dot-Product 3 becomes BFD. Indeed, the scores $\xi_{ik} = \frac{s_i}{r_k}$ are computed for all item-bin pairs with $s_i \leq r_k$, the pair (i^*, B_{k^*}) with the highest score is selected and item i^* is packed into bin B_{k^*}. If i^* has the largest size among all unallocated items, then allocation (i^*, B_{k^*}) is the same as the one which would be adopted by BFD. If i^* is not the largest item, then BFD would handle larger items first, which would not affect B_{k^*} (larger items do not fit B_{k^*}) and at some stage it would allocate i^* to B_{k^*} in the same fashion as Pairing(m).

Appendix B. The proof of Theorem 4

The second result of Section 6 is Theorem 4 that establishes non-monotonicity of multi-bin and bin-centric algorithms in the multi-dimensional case. We consider multi-bin algorithms in part B1 and bin-centric algorithms in part B2.

B.1. Non-monotonicity of WF(m) and WFD(m)

The proof is based on the instance given by Table 5 with the set of $n = 6$ two-dimensional items $\mathcal{I} = \{a, b, \dots, f\}$. We show that for each size measure, all items can be packed by the algorithm into $m = 2$ bins, but the same algorithm fails to find a feasible packing when $m = 3$. According to the definition of algorithm WF(m), it considers the items from $\mathcal{I}$ in the order they appear on the list. Since $v(a) \geq v(b) \geq \cdots \geq v(f)$ for each type of the forward size measure, the same item order is observed by algorithm WFD(m); see the three bottom rows of Table 5.

In the case of ℓ_∞, algorithms WF(2) and WFD(2) find a feasible solution $\{a, d, e\}$, $\{b, c, f\}$. Algorithms WF(3) and WFD(3) find a partial solution $\{a\}$, $\{b, e\}$, $\{c, d\}$, failing to allocate the remaining item f.

The decision making for WF(m) and WFD(m) with the ℓ_∞-size measure is illustrated in Table 6, where ✓ indicates the allocation of a current item to a bin and × indicates impossibility of allocating an item to a bin. The numbers in the table specify the residual capacities $v(B_k)$ of the bins computed via the ℓ_∞-norm after an allocation decision is made.

Now we show that for the ℓ_1-size measure, algorithms WF(2) and WFD(2) find a feasible solution $\{a, d, e\}$, $\{b, c, f\}$, while algorithms WF(3) and WFD(3) fail to allocate all items into 3 bins. In Table 7, ✓ indicates the allocation of a current item to a bin and × indicates impossibility of allocating an item to a bin. The numbers specify the residual capacities $v(B_k)$ of the bins computed via the ℓ_1-norm after an allocation decision is made.

Finally, we repeat the calculations for the ℓ_2-size measure. The results are presented in Table 8.

A similar reasoning can be applied to prove non-monotonicity of BF(m) and BFD(m) with the ℓ_2-size measure of bin load.

Note that the item sizes given in Table 5 satisfy: $\sum_{i\in\mathcal{I}} s_{i1} = \sum_{i\in\mathcal{I}} s_{i2} = 1.999$. Thus, with the common constant $d_h = \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} s_{ih}$ for $h = 1, 2$, the theorem holds for any weight-function presented in Table 4.

To conclude we observe that the results hold for any number of dimensions $d \geq 3$ by setting $s_{ih} = \frac{1}{2}(s_{i1} + s_{i2})$ for all items $i \in \mathcal{I}$ in dimensions $h = 3, \dots, d$. Note that for the modified data we have $v(a) > v(b) > v(c) > v(d) > v(e) > v(f)$, so that the item ordering used by WFD(m) remains the same as in the instance elaborated above, and algorithms WF(m) and WFD(m) operate in the same way as in the two-dimensional case. Note also that for the d-dimensional instance, $\sum_{i\in\mathcal{I}} s_{ih}$ is the same constant in each dimension $h = 1, \dots, d$, so that the instance works for any weight-function.

B.2. Non-monotonicity of Pairing(m)

Consider the instance of the two-dimensional problem with $n = 6$ items with sizes given in Table 9.

Suppose the scores are computed as Dot-Product 1. The operation of Pairing(2) and Pairing(3) is illustrated in Tables 10 and 11. The initial values of the residual capacities of the bins and initial scores ξ_{ik} are specified in the top part of these tables. In both cases the first decision is the same: allocating item a to bin B_1, according to the largest score, with ties broken in favor of the smallest indexed bin; the corresponding score is enframed. Next steps perform regular updates of bin residual capacities and scores. Pairing(2) produces a feasible solution $\{a, d, e\}$, $\{b, c, f\}$, while Pairing(3) produces a partial solution $\{a\}$, $\{b, e\}$, $\{c, d\}$, but fails to allocate item f.

We conclude the proof by observing that in the formulated instance $D_h = \sum_{i\in\mathcal{I}} s_{ih} = 2$ and $d_h = \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} s_{ih} = 2/7$ for $h = 1, 2$. This implies that $w_1 = w_2$ whichever w_h-formula from column (a) of Table 4 is used.

Considering now the Normalized Dot-Product score, we observe that the scores for the formulated instance are the same as for Dot-Product 1 subject to the multiplier $\frac{1}{D_h} = 1/2$, common for all item-bin pairs.

Finally, non-monotonicity holds for the multidimensional case with any $d \geq 2$ by setting $s_{ih} = 0$ for all items $i \in \mathcal{I}$ in all dimensions h strictly greater than 2.

Table 6
Operation of WF(m) and WFD(m) under ℓ_∞-size measure.

Operation of WF(2) and WFD(2) (ℓ_∞ size measure)					Operation of WF(3) and WFD(3) (ℓ_∞ size measure)						
Item	B_1	$v(B_1)$	B_2	$v(B_2)$	Item	B_1	$v(B_1)$	B_2	$v(B_2)$	B_3	$v(B_3)$
a	✓	0.515		1	a	✓	0.515		1		1
b		0.515	✓	0.516	b		0.515	✓	0.516		1
c		0.515	✓	0.015	c		0.515		0.516	✓	1
d	✓	0.020		0.015	d		0.515		0.516	✓	1
e	✓	0		0.015	e		0.515	✓	0.496	×	1
f		0	✓	0.001	f	×	0.515	×	0.496	×	1

Table 7
Operation of WF(m) and WFD(m) under ℓ_1-size measure.

Operation of WF(2) and WFD(2) (ℓ_1 size measure)					Operation of WF(3) and WFD(3) (ℓ_1 size measure)						
Item	B_1	$v(B_1)$	B_2	$v(B_2)$	Item	B_1	$v(B_1)$	B_2	$v(B_2)$	B_3	$v(B_3)$
a	✓	0.525		2	a	✓	0.525		2		2
b		0.525	✓	0.531	b		0.525	✓	0.531		2
c		0.525	✓	0.026	c		0.525		0.531	✓	1.495
d	✓	0.03		0.026	d		0.525		0.531	✓	1
e	✓	0		0.026	e		0.525	✓	0.501	×	1
f		0	✓	0.002	f	×	0.525	×	0.501	×	1

Table 8
Operation of WF(m) and WFD(m) under ℓ_2-size measure.

Operation of WF(2) and WFD(2) (ℓ_2 size measure)					Operation of WF(3) and WFD(3) (ℓ_2 size measure)						
Item	B_1	$v(B_1)$	B_2	$v(B_2)$	Item	B_1	$v(B_1)$	B_2	$v(B_2)$	B_3	$v(B_3)$
a	✓	0.265325		2	a	✓	0.265325		2		2
b		0.265325	✓	0.266481	b		0.265325	✓	0.266481		2
c		0.265325	✓	0.000346	c		0.265325		0.266481	✓	1.245025
d	✓	0.0005		0.000346	d		0.265325		0.266481	✓	1
e	✓	0		0.000346	e		0.265325	✓	0.246041	×	1
f		0	✓	0.000002	f	×	0.265325	×	0.246041	×	1

Table 9
Proof of Theorem 4: item sizes for Pairing(m).

Item	a	b	c	d	e	f
s_{i1}	0.485	0.484	0.505	0.495	0.02	0.011
s_{i2}	0.981	0.974	0.006	0.011	0.008	0.02

Appendix C. Optimal solutions for evaluating accuracy of heuristics

Optimal solutions for the *Triplet* benchmarks are known, as the instances are generated to get full occupancy of the bins.

For the remaining two types of benchmarks, *Panigrahy* and *New*, we attempt finding optimal solutions using the state-of-the-art solver for VBP by Brandão and Pedroso (2016), available as open-source software VPSolver; see Brandão (2016). VPSolver is used in our experiments in conjunction with the Gurobi solver 8.1.1. The Gurobi parameters are the same as in the experiments by Brandão and Pedroso (2016), with the exception of MIPFocus = 1 and Thread = 8. The experiments are performed on a single machine equipped with one Intel Xeon Gold 6138 CPU having 8 cores, with 16 GB of memory per core. We set a time limit of 4 hours for the Gurobi solver.

The instances of the *Panigrahy* type are of mixed complexity for VPSolver. Instances of Classes 1, 4 and 5 are the hardest to solve, as the average number of items per bin in a solution is larger than for other instances (about 4, 8 and 16 items per bin, respectively). Optimal solutions have been found for Class 1 instances with up to 120 items and only for a few instances of Classes 4 and 5, with 20 or 40 items. It is worth noticing that instances with hundreds of items often incur substantial computation time for creating the arc-flow model of VPSolver (e.g., more than 8 hours for a Class 1 instance with 500 items) and this stage may fail due to memory overflow (e.g., 128 GB was insufficient for one Class 4 instance with 120 items). For the remaining classes of the *Panigrahy* type, all instances could be solved in less than 5 min, and in some cases in a few seconds, except for the instances with 500 items of Classes 6 and 7, which require up to 3 hours of computation time.

The instances of the *New* type are generally hard to solve optimally. For Classes 1, 2 and 5, no instance with 250 or 500 items could be solved under 4 hours. For Classes 3 and 4, no instance with 500 items could be solved under 4 hours. For Class 6, only instances with up to 60 items could be solved under 4 hours.

Finally, we observe that small dimension instances often incur more computation time than higher dimension instances, if VPSolver is used. It is likely that there are more feasible combinations of items for packing when the dimension is smaller, and therefore the associated arc-flow model has more variables and constraints.

Appendix D. Supplementary data

The heuristics for VBP discussed in this paper are implemented as the C++ library *Vectorpack* available at https://github.com/Vectorpack/Vectorpack_cpp. The library is open for further extensions.

The supplementary materials, including all data, scripts, the summaries of the results and illustrative diagrams, are available in a companion repository: https://github.com/Vectorpack/experiments_vector_paper.

Table 10
Operation of Pairing(m) when $m = 2$.

	r_{k1}	r_{k2}		a	b	c	d	e	f
Initial values of r_{kh} and ξ_{ik} ($i \in \mathcal{I}$, $k = 1,2$, $h = 1,2$)									
B_1 :	1	1	ξ_{i1} :	1.466	1.458	0.511	0.506	0.028	0.031
B_2 :	1	1	ξ_{i2} :	1.466	1.458	0.511	0.506	0.028	0.031
The values of r_{kh} and ξ_{ik} after allocating a to B_1									
B_1 :	0.515	0.019	ξ_{i1} :	–	0.268	0.260	0.255	0.010	0.006
B_2 :	1	1	ξ_{i2} :	–	1.458	0.511	0.506	0.028	0.031
The values of r_{kh} and ξ_{ik} after allocating b to B_2									
B_1 :	0.515	0.019	ξ_{i1} :	–	–	0.260	0.255	0.010	0.006
B_2 :	0.516	0.026	ξ_{i2} :	–	–	0.261	0.256	0.011	0.006
The values of r_{kh} and ξ_{ik} after allocating c to B_2									
B_1 :	0.515	0.019	ξ_{i1} :	–	–	–	0.2551	0.0105	0.0060
B_2 :	0.011	0.020	ξ_{i2} :	–	–	–	0.0057	0.0004	0.0005
The values of r_{kh} and ξ_{ik} after allocating d to B_1									
B_1 :	0.020	0.008	ξ_{i1} :	–	–	–	–	0.00046	0.00038
B_2 :	0.011	0.020	ξ_{i2} :	–	–	–	–	0.00038	0.00052
The values of r_{kh} and ξ_{ik} after allocating f to B_2									
B_1 :	0.20	0.008	ξ_{i1} :	–	–	–	–	0.00046	-
B_2 :	0	0	ξ_{i2} :	–	–	–	–	0	-

Table 11
Operation of Pairing(m) when $m = 3$.

	r_{k1}	r_{k2}		a	b	c	d	e	f
Initial values of r_{kh} and ξ_{ik} ($i \in \mathcal{I}$, $k = 1,2$, $h = 1,2$)									
B_1 :	1	1	ξ_{i1} :	1.466	1.458	0.511	0.506	0.028	0.031
B_2 :			ξ_{i2} :	1.466	1.458	0.511	0.506	0.028	0.031
B_3 :	1	1	ξ_{i3} :	1.466	1.458	0.511	0.506	0.028	0.031
The values of r_{kh} and ξ_{ik} after allocating a to B_1									
B_1 :	0.515	0.019	ξ_{i1} :	–	0.268	0.260	0.255	0.010	0.006
B_2 :	1	1	ξ_{i2} :	–	1.458	0.511	0.506	0.028	0.031
B_3 :	1	1	ξ_{i3} :	–	1.458	0.511	0.506	0.028	0.031
The values of r_{kh} and ξ_{ik} after allocating b to B_2									
B_1 :	0.515	0.019	ξ_{i1} :	–	–	0.260	0.255	0.010	0.006
B_2 :	0.516	0.026	ξ_{i2} :	–	–	0.261	0.256	0.011	0.006
B_3 :	1	1	ξ_{i3} :	–	–	0.511	0.506	0.018	0.031
The values of r_{kh} and ξ_{ik} after allocating c to B_3									
B_1 :	0.515	0.019	ξ_{i1} :	–	–	–	0.2551	0.0105	0.0060
B_2 :	0.516	0.026	ξ_{i2} :	–	–	–	0.2557	0.0105	0.0062
B_3 :	0.495	0.994	ξ_{i3} :	–	–	–	0.2560	0.0178	0.0253
The values of r_{kh} and ξ_{ik} after allocating d to B_3									
B_1 :	0.515	0.019	ξ_{i1} :	–	–	–	–	0.01045	0.00604
B_2 :	0.516	0.026	ξ_{i2} :	–	–	–	–	0.01053	0.00620
B_3 :	0	0.983	ξ_{i3} :	–	–	–	–	0.00786	×
The values of r_{kh} and ξ_{ik} after allocating e to B_2									
B_1 :	0.515	0.019	ξ_{i1} :	–	–	–	–	–	×
B_2 :	0.496	0.018	ξ_{i2} :	–	–	–	–	–	×
B_3 :	0	0.983	ξ_{i3} :	–	–	–	–	–	×

References

Ahuja, R.K., Cunha, C.B., 2005. Very large-scale neighborhood search for the k-constraint multiple knapsack problem. J. Heuristics 11, 465–481. http://dx.doi.org/10.1007/s10732-005-2634-9.

Alves, C., Clautiaux, F., de Carvalho, J.V., Rietz, J., 2016. Dual-Feasible Functions for Integer Programming and Combinatorial Optimization: Basics, Extensions and Applications. Springer International Publishing, Cham, http://dx.doi.org/10.1007/978-3-319-27604-5.

Alves, C., de Carvalho, J.V., Clautiaux, F., Rietz, J., 2014. Multidimensional dual-feasible functions and fast lower bounds for the vector packing problem. European J. Oper. Res. 233, 43–63. http://dx.doi.org/10.1016/j.ejor.2013.08.011.

Baldacci, R., Coniglio, S., Cordeau, J.F., Furini, F., 2023. A numerically exact algorithm for the bin-packing problem. INFORMS J. Comput. 36, 141–162. http://dx.doi.org/10.1287/ijoc.2022.0257.

Brandão, F., 2016. Arc-flow Vector Packing Solver (VPSolver). https://vpsolver.fdabrandao.pt/.

Brandão, F., Pedroso, J.P., 2016. Bin packing and related problems: General arc-flow formulation with graph compression. Comput. Oper. Res. 69, 56–67. http://dx.doi.org/10.1016/j.cor.2015.11.009.

Cacchiani, V., Iori, M., Locatelli, A., Martello, S., 2022. Knapsack problems – An overview of recent advances. Part II: Multiple, multidimensional, and quadratic knapsack problems. Comput. Oper. Res. 105693. http://dx.doi.org/10.1016/j.cor.2021.105692.

Cai, B., Guo, Q., Yu, J., 2022. LraSched: Admitting more long-running applications via auto-estimating container size and affinity. Comput. J. 65, 2377–2391. http://dx.doi.org/10.1093/comjnl/bxab072.

Caprara, A., Toth, P., 2001. Lower bounds and algorithms for the 2-dimensional vector packing problem. Discrete Appl. Math. 111, 231–262. http://dx.doi.org/10.1016/S0166-218X(00)00267-5.

Chekuri, C., Khanna, S., 2004. On multidimensional packing problems. SIAM J. Comput. 33, 837–851. http://dx.doi.org/10.1137/S0097539799356265.

Christensen, H.I., Khan, A., Pokutta, S., Tetali, P., 2017. Approximation and online algorithms for multidimensional bin packing: A survey. Comput. Sci. Rev. 24, 63–79. http://dx.doi.org/10.1016/j.cosrev.2016.12.001.

Coffman, E.G., Csirik, J., Galambos, G., Martello, S., Vigo, D., 2013. Bin packing approximation algorithms: Survey and classification. In: Pardalos, P., Du, D.Z.,

Graham, R. (Eds.), Handbook of Combinatorial Optimization. Springer, New York, pp. 455–531. http://dx.doi.org/10.1007/978-1-4419-7997-1_35.

Coffman, E.G., Garey, M.R., Johnson, D.S., 1978. An application of bin-packing to multiprocessor scheduling. SIAM J. Comput. 7, 1–17. http://dx.doi.org/10.1137/0207001.

Coffman, E.G., Garey, M.R., Johnson, D.S., 1984. Approximation algorithms for bin-packing — An updated survey. In: Ausiello, G., Lucertini, M., Serafini, P. (Eds.), Algorithm Design for Computer System Design, International Centre for Mechanical Sciences. Vol. 284, Springer, Vienna, pp. 49–106. http://dx.doi.org/10.1007/978-3-7091-4338-4_3.

Csirik, J., Dósa, G., 2018. Performance guarantees for one-dimensional bin packing. In: Gonzalez, T. (Ed.), Handbook of Approximation Algorithms and Metaheuristics. Chapman and Hall/CRC, New York, pp. 491–517. http://dx.doi.org/10.1201/9781351236423.

Delorme, M., Iori, M., Martello, S., 2016. Bin packing and cutting stock problems: Mathematical models and exact algorithms. European J. Oper. Res. 255, 1–20. http://dx.doi.org/10.1016/j.ejor.2016.04.030.

Dosa, G., Sgall, J., 2013. First fit bin packing: A tight analysis. In: 30th International Symposium on Theoretical Aspects of Computer Science. STACS 2013, pp. 538–549. http://dx.doi.org/10.4230/LIPIcs.STACS.2013.538.

Dosa, G., Sgall, J., 2014. Optimal analysis of best fit bin packing. In: Lect. Notes Comput. Sc., vol. 8572, pp. 429–441. http://dx.doi.org/10.1007/978-3-662-43948-7_36.

Epstein, L., van Stee, R., 2018. Multidimensional packing problems. In: Gonzalez, T. (Ed.), Handbook of Approximation Algorithms and Metaheuristics. Chapman and Hall/CRC, New York, pp. 553–570. http://dx.doi.org/10.1201/9781351236423.

Falkenauer, E., 1996. A hybrid grouping genetic algorithm for bin packing. J. Heuristics 2, 5–30. http://dx.doi.org/10.1007/BF00226291.

Gabay, M., Zaourar, S., 2016. Vector bin packing with heterogeneous bins: Application to the machine reassignment problem. Ann. Oper. Res. 242, 161–194. http://dx.doi.org/10.1007/s10479-015-1973-7.

Garefalakis, P., Karanasos, K., Pietzuch, P., Suresh, A., Rao, S., 2018. Medea: Sheduling of long running applications in shared production clusters. In: Proceedings of the Thirteenth EuroSys Conference. pp. 1–13. http://dx.doi.org/10.1145/3190508.3190549.

Garey, M.R., Graham, R.L., Johnson, D.S., Yao, A.C.C., 1976. Resource constrained scheduling as generalized bin packing. J. Combin. Theory Ser. A 21, 257–298. http://dx.doi.org/10.1016/0097-3165(76)90001-7.

Gendreau, M., Laporte, G., Semet, F., 2004. Heuristics and lower bounds for the bin packing problem with conflicts. Comput. Oper. Res. 31, 347–358. http://dx.doi.org/10.1016/S0305-0548(02)00195-8.

Gonzalez, T., 2018. Handbook of Approximation Algorithms and Metaheuristics. Chapman and Hall/CRC, New York, http://dx.doi.org/10.1201/9781351236423.

Graham, R.L., 1969. Bounds on multiprocessing timing anomalies. SIAM J. Appl. Math. 17, 263–269. http://dx.doi.org/10.1137/0117039.

Gurski, F., Rehs, C., 2020. Counting and enumerating independent sets with applications to combinatorial optimization problems. Math. Methods Oper. Res. 91, 439–463. http://dx.doi.org/10.1007/s00186-019-00696-4.

Jangiti, S., Jayaraman, R., Ramprasad, H., Shankar Sriram, V.S., 2019a. Resource ratio based virtual machine placement in heterogeneous cloud data centres. Sādhanā 44, 236. http://dx.doi.org/10.1007/s12046-019-1215-9.

Jangiti, S., Sri Ram, E., Shankar Sriram, V.S., 2019b. Aggregated rank in first-fit-decreasing for green cloud computing. In: Mallick, P., Balas, V., Bhoi, A., Zobaa, A. (Eds.), Cognitive Informatics and Soft Computing. vol. 768, Springer, Singapore, pp. 545–555. http://dx.doi.org/10.1007/978-981-13-0617-4_53.

Johnson, D.S., 1974. Approximation algorithms for combinatorial problems. J. Comput. System Sci. 9, 256–278. http://dx.doi.org/10.1016/S0022-0000(74)80044-9.

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

Kou, L.T., Markowsky, G., 1977. Multidimensional bin packing algorithms. IBM J. Res. Dev. 21, 443–448. http://dx.doi.org/10.1147/rd.215.0443.

Krause, K.L., Shen, V.Y., Schwetman, H.D., 1975. Analysis of several task-scheduling algorithms for a model of multiprogramming computer systems. J. ACM 22, 522–550. http://dx.doi.org/10.1145/321906.321917.

Kumaraswamy, S., Nair, M.K., 2019. Bin packing algorithms for virtual machine placement in cloud computing: a review. Int. J. Electr. Comput. Eng. 9, 512–524. http://dx.doi.org/10.11591/ijece.v9i1.pp512-524.

Mann, Z.Á., 2015. Allocation of virtual machines in cloud data centers—a survey of problem models and optimization algorithms. ACM Comput. Surv. 48, 1–34. http://dx.doi.org/10.1145/2797211.

Maruyama, K., Chang, S.K., Tang, D.T., 1977. A general packing algorithm for multidimensional resource requirements. Int. J. Comput. Inf. Sci. 6, 131–149. http://dx.doi.org/10.1007/BF00999302.

Mommessin, C., Yang, R., Shakhlevich, N.V., Sun, X., Kumar, S., Xiao, J., Xu, J., 2023. Affinity-aware resource provisioning for long-running applications in shared clusters. J. Parallel Distrib. Comput. 177, 1–16. http://dx.doi.org/10.1016/j.jpdc.2023.02.011.

Munien, C., Mahabeer, S., Dzitiro, E., Singh, S., Zungu, S., Ezugwu, A.E.-S., 2020. Metaheuristic approaches for one-dimensional bin packing problem: A comparative performance study. IEEE Access 8, 227438-227465. http://dx.doi.org/10.1109/ACCESS.2020.3046185.

Murgolo, F.D., 1988. Anomalous behavior in bin packing algorithms. Discrete Appl. Math. 21, 229–243. http://dx.doi.org/10.1016/0166-218X(88)90069-8.

Nagel, L., Popov, N., Süß, T., Wang, Z., 2023. Analysis of heuristics for vector scheduling and vector bin packing. In: Lect. Notes Comput. Sc., vol. 14286, pp. 583–598. http://dx.doi.org/10.1007/978-3-031-44505-7_39.

Panigrahy, R., Talwar, K., Uyeda, L., Wieder, U., 2011. Heuristics for Vector Bin Packing. Microsoft Research, https://www.microsoft.com/en-us/research/publication/heuristics-for-vector-bin-packing/.

Pessoa, A., Sadykov, R., Uchoa, E., 2021. Solving bin packing problems using VRPSolver models. Oper. Res. Forum 2, 20. http://dx.doi.org/10.1007/s43069-020-00047-8.

Shi, L., Furlong, J., Wang, R., 2013. Empirical evaluation of vector bin packing algorithms for energy efficient data centers. In: 2013 IEEE Symp. Comp. Commu. pp. 000009–000015. http://dx.doi.org/10.1109/ISCC.2013.6754915.

Spieksma, F.C.R., 1994. A branch-and-bound algorithm for the two-dimensional vector packing problem. Comput. Oper. Res. 21, 19–25. http://dx.doi.org/10.1016/0305-0548(94)90059-0.

Stillwell, M., Schanzenbach, D., Vivien, F., Casanova, H., 2010. Resource allocation algorithms for virtualized service hosting platforms. J. Parallel Distrib. Comput. 70, 962–974. http://dx.doi.org/10.1016/j.jpdc.2010.05.006.

Wei, L., Lai, M., Lim, A., Hu, Q., 2020. A branch-and-price algorithm for the two-dimensional vector packing problem. European J. Oper. Res. 281, 25–35. http://dx.doi.org/10.1016/j.ejor.2019.08.024.