Ann Oper Res
DOI 10.1007/s10479-015-1973-7

Vector bin packing with heterogeneous bins: application to the machine reassignment problem

Michaël Gabay[1,2] · **Sofia Zaourar**[3]

Abstract In this paper, we introduce a generalization of the vector bin packing problem, where the bins have variable sizes. This generalization can be used to model virtual machine placement problems and in particular to build feasible solutions for the machine reassignment problem. We propose several families of greedy heuristics for this problem and show that they are flexible and can be adapted to handle additional constraints. We present structural properties of the machine reassignment problem, that allow us to decompose it into smaller subproblems and adapt our heuristics to them. We evaluate our heuristics on academic benchmarks of the vector bin packing problem, a randomly generated vector bin packing problem with heterogeneous bins benchmark as well as Google's realistic instances of the machine reassignment problem.

Keywords Vector bin packing with heterogeneous bins · Machine reassignment problem · Heuristics · Virtual machine placement · Vector bin packing

1 Introduction

In service hosting and virtualized hosting, services or virtual machines must be assigned to clustered servers. Each server has to provide enough resources, such as CPU, RAM or disk, in order to have all of its processes running. The machine reassignment problem, proposed by Google for the ROADEF/EURO challenge 2012[1], is such an assignment problem, with

[1] http://challenge.roadef.org/2012/en/.

✉ Michaël Gabay
michael@gabay.email

Sofia Zaourar
sofia.zaourar@inria.fr

[1] Univ. Grenoble Alpes, G-SCOP, 38000 Grenoble, France

[2] CNRS, G-SCOP, 38000 Grenoble, France

[3] UJF, Inria Grenoble, 655 Avenue de l'Europe, Montbonnot, 38334 Saint Ismier Cedex, France

Published online: 07 September 2015

additional constraints and a cost function to minimize. In this paper, we propose a modeling framework for this kind of packing problems and a greedy heuristic framework to find feasible assignments. Several classical bin packing heuristics are adapted and new variants are proposed. A worst-case complexity analysis of these algorithms is also provided. Moreover, we present some structural properties of the machine reassignment problem and use them to decompose the problem into smaller and easier subproblems, that can be tackled with our heuristics. Finally, we provide experimental results on vector bin packing benchmarks, on a new benchmark for the vector bin packing problem with heterogeneous bins and on realistic instances for the machine reassignment problem.

1.1 Bin packing problems

In the classical bin packing (BP) problem, we are given a set $\mathcal{I} = \{I_1, \ldots, I_n\}$ of n items, a capacity $C \in \mathbb{N}$ and a size function $s : \mathcal{I} \to \mathbb{N}$. The goal is to find a feasible assignment minimizing the number of bins used. A feasible assignment of the items into N bins is a partition $P_1, \ldots, P_N$ of the items, such that for each P_k, the sum of the sizes of the items in P_k does not exceed the capacity C. In the decision version of this problem, the number of bins N is part of the input and the objective is to decide whether all the items can be packed using at most N bins. This problem is known to be strongly NP-hard (Garey and Johnson 1979).

Garey et al. (1976) introduced a generalization of this problem, called vector bin packing (VBP) or d-dimensional vector packing (d-DVP). In this problem, the weights of the items are described by a d-dimensional vector: $(s_i^1, \ldots, s_i^d)$ and bins have a capacity C in all dimensions. A feasible assignment of the items into N bins is a partition $P_1, \ldots, P_N$ of the items such that for each P_k, on each dimension, the sum of the sizes of the items in P_k does not exceed the capacity:

$$\sum_{i \in P_k} s_i^j \leq C, \quad \forall k = 1, \ldots, N, \quad \forall j = 1, \ldots, d.$$

Vector bin packing is often used to model virtual machine placements (Lee et al. 2011; Panigrahy et al. 2011; Stillwell et al. 2010). In such cases, all machines are supposed to have the same capacities. This could be the case when a new computer cluster is built. However, as it grows and servers are renewed, new machines are introduced and the cluster becomes heterogeneous.

In this work, we are interested in a further generalization of this problem, where each bin has its own vector of capacities $(c_k^1, \ldots, c_k^d)$ and the goal is to find a feasible packing of the items. We call this problem the Vector bin packing with heterogeneous bins (VBPHB) problem; to the best of our knowledge, this problem has not been specifically addressed in the literature so far. VBPHB can be used to model previously mentioned virtual machine placement problems in a realistic heterogeneous environment.

1.2 Machine reassignment problem

The machine reassignment problem was proposed by Google for the 2012 challenge organized jointly by ROADEF and EURO. The subject of this challenge was based on problems occurring in Google's data centers and realistic instances were provided. In the machine reassignment problem, a set of processes needs to be (re)assigned to a set of machines. There are m resources and each machine (resp. process) has its own capacity (resp. requirement) for each resource. There are also additional constraints presented in Sect. 4. The aim is to find a feasible assignment minimizing a weighted cost.

In the challenge, an initial feasible solution was provided. Therefore, local search-based heuristics were a natural (and successful) approach. Local search aims at improving iteratively a given solution by applying small modifications (called moves) and is well-suited to quickly improve solutions of very large-scale problems, see Aarts and Lenstra (1997) for a detailed survey on local search heuristics. When using local search, the explored solutions are limited by the initial solution and the set of accepted moves. This exploration space can be enlarged by running several local searches with different parameters and starting with a diversified set of initial solutions. Feo and Resende (1989), Feo and Resende (1995) designed the greedy randomized adaptive search procedure (GRASP) which is an iterative process where one successively creates a new feasible solution, then optimizes it using a local search algorithm. When applying a GRASP heuristic to the machine reassignment problem, VBPHB arises as a subproblem for generating new initial solutions. Furthermore, for other real-life machine reassignment's type of problems, a feasible solution may not be part of the instances. We explain how we can handle additional constraints and find new feasible assignments by solving VBPHB problems in Sect. 4.

Competitors in ROADEF challenge have exposed their approaches and results in Gavranović et al. (2012), Mehta et al. (2012), Portal (2013), Lopes et al. (2014).

1.3 Outline

This paper is organized as follows. In Sect. 2, we define VBPHB and present related work on vector bin packing. In Sect. 3, we present our heuristics for this problem. In Sect. 4, we discuss structural properties of the machine reassignment problem that allow us to adapt our heuristics to this problem. Experimental results are reported in Sects. 3.5 and 4.5.

2 Vector bin packing problem with heterogeneous bins

An instance of the vector bin packing problem with heterogeneous bins is defined by C an $\mathbb{N}^{N\times d}$ capacity matrix and S an $\mathbb{N}^{n\times d}$ size matrix, where c_k^j (resp. s_i^j) denotes the capacity of bin k (resp. the size of item i) in dimension j. We define the following index sets: $\mathcal{B} = \{1,\ldots,N\}$ for the bins, $\mathcal{I} = \{1,\ldots,n\}$ for the items, and $\mathcal{D} = \{1,\ldots,d\}$ for the dimensions. The problem is to find a feasible assignment $x \in \mathbb{N}^{n\times N}$ of the items into the bins such that:

$$\sum_{i\in\mathcal{I}} s_i^j x_{i,k} \leq c_k^j \qquad \forall j \in \mathcal{D}, \quad \forall k \in \mathcal{B} \tag{1}$$

$$\sum_{k\in\mathcal{B}} x_{i,k} = 1 \qquad \forall i \in \mathcal{I} \tag{2}$$

$$x_{i,k} \in \{0,1\} \qquad \forall i \in \mathcal{I}, \quad \forall k \in \mathcal{B}. \tag{3}$$

Inequality (1) models the capacity constraints while constraints (2) and (3) ensure that each item is assigned to a bin.

Observe that any instance of the vector bin packing problem with heterogeneous bins can be transformed into an instance of the vector bin packing problem. Indeed, if we denote by c the maximum capacity ($c = \max_{j,k} c_k^j$), we obtain an equivalent vector bin packing problem by adding one dimension, defining all bin capacities to be equal to c and adding $|\mathcal{B}|$ artificial items with requirements $(c - c_k^1, \ldots, c - c_k^d, c)$. All other items requirements are set to 0 in the dimension $d + 1$. Yet, we introduce the vector bin packing problem with heterogeneous

bins because we want to be able to explicitly exploit the variable bin sizes to find solutions. Moreover, vector bin packing with heterogeneous bins is a natural framework to model heterogeneous data centers for instance but it also naturally accounts for rare resources. For instance, few machines may be equipped with GPUs so using them for processes which do not require GPUs can easily lead to infeasible solutions. In vector bin packing problem, all bins are identical and if we introduce artificial items, we conceal the fact that there are rare resources which should not be wasted.

2.1 Related work

Since VBPHB is a generalization of bin packing, this problem is strongly NP-hard. Moreover, Chekuri and Khanna (1999) proved that the 2-dimensional vector bin packing is *APX*-hard and showed $d^{1/2-\epsilon}$ hardness of approximation. Woeginger (1997) proved that there is no asymptotic PTAS (unless P=NP). Hence, as a generalization of d-DVP, the optimization version of VBPHB (where the different bins are types on bins, each one with a cost) is *APX*-hard and cannot have an asymptotic PTAS. Maruyama et al. (1977) generalized classical bin packing heuristics into a general framework for the vector bin packing.

There are many theoretical results for the vector bin packing problem: Kou and Markowsky (1977) studied lower and upper bounds and showed that the worst case performance ratio for the generalization of some classical bin packing algorithms is larger than d, where d is the dimension. Yao (1980) proved that any $o(n \log n)$ time algorithm has a worst case performance ratio bigger than d. Bansal et al. (2006) proposed a randomized $(\log d + 1 + \epsilon)$-approximation. Their algorithm is polynomial for fixed d. Spieksma (1994) proposed two lower bounds for 2-DVP and a branch-and-bound algorithm using these bounds. Caprara and Toth (2001) analyzed several lower bound for 2-DVP and showed that the lower bound obtained by the linear programming relaxation of the (huge) integer programming formulation they propose, dominates all these bounds. Chang et al. (2005) used 2-DVP to model a packing problem where steel products have to be packed into special containers and they proposed a heuristic. Caprara et al. (2003) showed that there is a PTAS for d-DVP if all items sizes are totally ordered. Shachnai and Tamir (2003) studied data placement problem as an application of VBP and proposed a PTAS for a subcase of VBP. Karp et al. (1984) studied VBP where all items sizes are drawn independently from the uniform distribution over [0,1]. They proved that the expected wasted space by the optimal solution is $\Theta(n^{\frac{d-1}{d}})$ and proposed an algorithm that tries to pack two items in each bin and has the same expected wasted space.

Stillwell et al. (2010) implemented and compared several heuristics for VBP with additional real-world constraints in the case of virtualized hosting platforms. They found out that the algorithm which is performing the best is the choose pack heuristic from Leinberger et al. (1999) with items sorted by decreasing order of the sum of their requirements.

Brandao and Pedroso (2013) generalized the arc-flow formulation from Carvalho (1999) for bin packing and cutting stock to the vector bin packing problem and proposed improvements for this approach. They experimented their approach on academic benchmarks and closed many open instances.

Other works have considered the variable sized bin packing problem in which there are several types of bins and the aim is to minimize the sum of bin costs. Han et al. (1994) studied the 2-dimensional vector bin packing problem in which items have specific requirements for each bin and proposed exact and heuristic approaches along with a process to improve lower bounds.

In the classical first fit decreasing (FFD) heuristic, one has to select the *largest* item and then pack it into a bin. Hence, if one generalizes this heuristic to the multidimensional case, it has to be determined how to measure and compare items. Panigrahy et al. (2011) presented a generalization of the classical first fit decreasing (FFD) heuristic to VBP and experimented several measures. A promising measure is the *DotProduct* which defines the *largest* item as the item that maximizes some weighted dot product between the vector of remaining capacities and the vector of requirements for the item.

3 Heuristic framework

We generalize the classical first fit decreasing (FFD) and Best Fit Decreasing (BFD) heuristics to VBPHB. Algorithm 1 is the classical BFD algorithm. Panigrahy et al. (2011) proposed a different approach of this algorithm which focuses on the bins, as detailed in Algorithm 2. In order to use these algorithms in multidimensional packing problems, one needs to define an ordering on bins and items.

This ordering can be defined using a measure: a size function which returns a scalar for each bin and item. In the following sections we propose several measures; in particular we consider *dynamic* measures that depends on the *remaining capacities* of the bins and decisions made. In that case, both the orderings of items and bins may change in the course of the algorithm. Observe that if the order is unchanged then item centric (Algorithm 1) and bin centric (Algorithm 2) heuristics give the same results (either both are infeasible or both are feasible and return the same solution).

Remark that any greedy algorithm for this problem can be reduced to a best fit item centric heuristic by computing the next decision of the algorithm in the measure and returning size 2 for chosen item, size 0 for chosen bin and size 1 for other bins and items.

Algorithm 1: BFD Item Centric

```
while There are unpacked items do
    Compute sizes
    Pack the biggest item into the smallest feasible bin
    if the item cannot be packed then
        return Failure
return Success
```

Algorithm 2: BFD Bin Centric

```
while The list of bins is not empty do
    Compute sizes
    Select b the smallest bin
    while An unpacked item fits into b do
        Compute sizes
        Pack the biggest feasible item into b
    Remove b from the list of bins
if An item has not been packed then
    return Failure
return Success
```

3.1 Measures

In order to sort items and bins, we define a measure. Let $i \in \mathcal{I}$, $k \in \mathcal{B}$, $j \in \mathcal{D}$. We define $\mathcal{I}_r$ as the set of unpacked items and $\mathcal{B}_r$ as the set of remaining bins (unless we are using a bin centric approach, $\mathcal{B}_r = \mathcal{B}$). We denote by r_k^j the remaining capacity of bin k in dimension j, by $C(j)$ the total remaining capacity in dimension j and by $R(j)$ the total requirement in dimension j: $C(j) = \sum_{k \in \mathcal{B}_r} r_k^j$ and $R(j) = \sum_{i \in \mathcal{I}_r} s_i^j$. A natural idea to define a scalar size from a vector size is to take a weighted sum of the vector components. We define the following sizes:

$$S_{\mathcal{B}}(k) = \sum_{j \in \mathcal{D}} \alpha_j r_k^j \qquad \forall k \in \mathcal{B},$$

$$S_{\mathcal{I}}(i) = \sum_{j \in \mathcal{D}} \beta_j s_i^j \qquad \forall i \in \mathcal{I},$$

where α and β are two scaling vectors. We propose three different scaling coefficients: $\frac{1}{C(j)}$, $\frac{1}{R(j)}$ and $\frac{R(j)}{C(j)}$. The first ratio normalizes based on bins capacities; the second normalizes based on items requirements. The third coefficient takes into account both capacities and requirements. For a feasible instance, the ratio $\frac{R(j)}{C(j)}$ is in $(0, 1]$ for all resource j; the closer it is to one, the more critical is the resource. In that case, we give more importance, i.e. more weight, to the corresponding component of the items (resp. machines) requirements (resp. capacities).

Another possibility is to define the size of an item as its maximal normalized requirement over the resources. We obtain the *priority* measure:

$$S_{\text{prio}}(i) = \max_{j \in \mathcal{D}} \frac{s_i^j}{C(j)} \qquad \forall i \in \mathcal{I}.$$

The previous sizes can be computed once and for all, before the first run of the algorithm; we call the resulting measures and heuristics *static*. If the sizes are updated at each iteration of the algorithm, we call the measures and heuristics *dynamic*.

For static measures, the ordering is fixed and Algorithms 1 and 2 become first fit heuristics and return the same results. In this work, we propose both static and dynamic heuristics. In the remaining of the paper, we choose $\alpha = \beta$; as a consequence, the measure S has the following property:

Property 1 If $\alpha = \beta$ and $S_{\mathcal{B}}(k) < S_{\mathcal{I}}(i)$ then the item i does not fit into the bin k.

Proof If $S_{\mathcal{B}}(k) < S_{\mathcal{I}}(i)$, then $\sum_{j \in \mathcal{D}} \alpha_r (r_k^j - s_i^j) < 0$. Since both r and s are positive, $r_k^j < s_i^j$ for some j. □

Using this property, one only needs to check whether an item fits in a bin if $S_{\mathcal{B}}(k) \geq S_{\mathcal{I}}(i)$.

3.2 Bin balancing

In Sect. 3.1, we presented measures which yield different heuristics when combined with Algorithms 1 and 2. Since bin capacities are different though, it is hard to predict which resource, bin or item will be the bottleneck. Moreover, we can take advantage of the fact that we are only interested in finding feasible assignments. Packing as many items as possible in a bin implicitly aims at minimizing the number of bins. Instead, our idea is to balance the load.

The permutation pack and choose pack heuristics of Leinberger et al. (1999) use such an approach to pack items. We propose another approach: using the item centric heuristic, once a bin is assigned a new item, it is moved to the end of the list of bins that will be considered to pack the next item. This approach is detailed in Algorithm 3. Line 3, l_B is updated by one of the two following ways:

- *Single bin balancing:* Used bin is moved to the end of the list
- *Bin balancing:* All bins tried (including the successful bin) are moved to the end of the list in the same order: let l be the new list. We have:
  $l(1) = l_B(j+1), \ldots, l(N-j) = l_B(N), l(N-j+1) = l_B(1), \ldots, l(N) = l_B(j)$
  (in practice, this is simply achieved by using a modulo)

Algorithm 3: Bin Balancing Heuristics

```
1  Sort l_B (bins list) and l_I (items list)
2  while There are unpacked items do
3      Let I be the biggest unpacked item
4      for j = 1 to N do
5          if item I can be packed into l_B[j] then
6              Pack I into l_B[j]
7              Update l_B
8              break
9      if I has not been packed then
10         return Failure
11 return Success
```

The main idea of this algorithm is that once an item is assigned to a bin, we try to assign the following items to other bins, in order to prevent critical bins from being filled too early.

3.3 Dot product

We generalize the *dot product* heuristic of Panigrahy et al. (2011). In this heuristic we assign in priority the item i to the bin k such that the weighted dot product between their requirement and capacity is maximal:

$$(i,k) = \underset{i\in\mathcal{I},k\in\mathcal{B}}{\arg\max} \sum_{j\in\mathcal{D}} \alpha_{i,k}\, s_i^j\, r_k^j.$$

The idea of this heuristic is to pack in priority items on bins with similar "shape". We propose three variants of this heuristic, corresponding to weights $\alpha_{i,k} = 1, \alpha_{i,k} = (\|s_i\|_2\, \|r_k\|_2)^{-1}$ or $\alpha_{i,k} = \|r_k\|_2^{-2}$.

With the first coefficients, we maximize the similarity and the size used. With the second, we normalize both sizes and capacities, so we minimize the angle between the two vectors. Eventually, if we re-scale by $\frac{1}{\|r_k\|_2^2}$, then we focus on maximizing the scalar projection of the item, i.e. maximizing similarity.

In the first iteration, we compute dot products for all feasible pairs, then store these values. On the following iterations, only the dot products involving the bin where an item has just been packed need to be updated. The worst case time and space complexity for initializing sizes is $\mathcal{O}(dnN\log(nN))$. The complexity of computing costs afterward is at most $\mathcal{O}(dn)$ and the list can be maintained in $\mathcal{O}(n\log(nN))$.

This heuristic maximizes the *similarity* of a bin and an item (the scalar projection of the item sizes onto the bin remaining capacities). Moreover, we need to be able to compare these

dot products for all pairs of bins and items. On one hand, if we do not scale the vectors, then we maximize both the similarity and the size used. On the other hand, if we normalize both sizes and capacities, we minimize the angle between the two vectors. Eventually, if we re-scale by $\frac{1}{\|r_k\|_2^2}$, then we focus on maximizing the scalar projection of the item and maximize similarity.

3.4 Complexity

We denote $p = \max(n, N)$. In the worst case scenario, both the item centric and the bin centric algorithms behave as shown in Algorithm 4. Hence, the overall time complexity is $\mathcal{O}(dp^2 + p^2 \log p)$. The space complexity is $\mathcal{O}(p^2 + dp)$ for the dot product and $\mathcal{O}(dp)$ for other measures.

Algorithm 4: Worst-case heuristics behavior

```
   Initialize sizes                          // O(dp^2) (DotProduct)
   for i = 1 to p do                         // p×
2      Compute sizes                         // O(dp) (for given measures)
       Sort lists                            // O(p log p)
       Pick an item                          // O(1)
1      Pack it                               // O(dp)
```

The overall complexity is $\mathcal{O}(dp^2)$.

3.5 Experiments

We experimented all described heuristics on academic bin packing and vector bin packing benchmarks as well as a new VBPHB benchmark that we generated. Since this problem is new in the literature, there were no benchmark available. However, since it is a generalization of the vector bin packing problem, we led experiments using academic benchmarks for these problems. In this section, we present the heuristics and experiments on classical benchmark for bin and vector packing as well as experiments on our new benchmark for vector bin packing with heterogeneous bins. We first present aggregated results of our heuristics, then the new benchmark with detailed results for each heuristic.

3.5.1 Heuristics

We use the heuristics presented in previous sections with following measures: *none* (static, items and bins are kept as provided in the input), static shuffle, static $1/C$, static $1/R$, static R/C, dynamic shuffle, dynamic $1/C$, dynamic $1/R$ and dynamic R/C. We use these measures with the item centric, bin centric, bin balancing and single bin balancing heuristics. This gives 31 heuristics since item centric and bin centric heuristics are the same for static measures. We also use the 3 variants of the dot product heuristic, for a total of 34 heuristics.

Heuristics are implemented in Python and the focus was made on simplicity rather than efficiency. For this reason, we do not report running times. However, even with these simple implementations, all heuristics (except dot-product) run in less than 0.1 s on every instance.

Since the vector bin packing problem with heterogeneous bins is defined as a decision problem, we implemented heuristics dedicated to this approach. So the number of bins and the bins capacities are fixed when a heuristic is called.

However, academic benchmarks for vector bin packing and bin packing problems are focused on optimization, so we implemented a very simple optimization procedure. Observe that all heuristics described here are simple and efficient so a natural way towards solving a problem efficiently is to combine them all in a best-of-many algorithm: call all heuristics on the problem and keep the best result.

We implemented a binary search procedure on the number of bins. For a fixed number of bins, heuristics are called until one succeeds or they all fail. Each heuristic (including random) is only called 0 or 1 time for a given number of bins. Obviously, one can implement these heuristics without using binary search and with a much better performance. Especially for item centric heuristics whose number of bins can actually be computed in a single pass.

For vector bin packing instances, let C_j be the capacity of all bins in dimension j. In each dimension of a vector bin packing problem, we have a bin packing problem whose minimum number of bins is smaller than or equal to the minimum number of bins in the vector bin packing problem. If we formulate all these bin packing problems using the assignment based formulation and compute and round up the maximum of the linear programming relaxation over all dimensions, we obtain a lower bound on the vector bin packing problem. This lower bound is equal to:

$$l_\infty = \max_{j \in \mathcal{D}} \left\lceil \frac{\sum_{i \in \mathcal{I}} r_i^j}{C_j} \right\rceil \tag{4}$$

We computed this lower bound on all open instances of the vector bin packing benchmark and used it to compare our heuristics. This simple lower bound, combined with our heuristics allows us to solve and prove optimality (the lower bound l_∞ is equal to the heuristic solution) on 54 out of the 77 open instances from Brandao and Pedroso (2013). For all open instances in which this lower bound combined with our heuristics was not sufficient to close the gap, we also computed the lower bound using optimum integer solutions of the bin packing problems in each dimension; in all cases the two bounds were equal.

3.5.2 Vector bin packing benchmark

Brandao and Pedroso (2013) gathered instances on bin packing, cutting stock and vector bin packing from the literature. In our experiments, we use all the vector bin packing instances of their benchmark and some of their bin packing instances.

We briefly describe each set of instances that we considered. The HARD28 data set is a selection of 28 very difficult instances from Schoenfield (2002). The BPP FLK data set was proposed by Falkenauer (1996) and is composed of random uniform instances and triplets instances in which each bin is filled with three items in an optimal solution. The SCHOLL dataset, from Scholl et al. (1997), is composed of random instances with expected number of bins smaller than or equal to 3; expected number of items per bin equal to 3, 5, 7, 9; and difficult instances with 200 items. Caprara and Toth (2001) proposed the 2CBP data set of two-dimensional vector packing instances with several sizes and classes detailed in their paper. Brandao and Pedroso (2013) elaborated the 20CBP data set, a set of 40 20-dimensional instances obtained by concatenating instances of the same classes from Caprara and Toth (2001).

There are 1838 instances in the benchmark, including 77 open instances. All previously known optimal solutions can be obtained with the algorithm from Brandao and Pedroso (2013) with running times from less than a second up to hours.

Lower bound 4 is tight on 1054 out of the 1761 instances with known optimum solution and on at least 54 out of the 77 remaining instances. Combining this lower bound with the

Table 1 Results on academic benchmark

	Instances				Optimality		% Gap
Data set	$\#dim$	$\#inst$	$\#kopt$	$n_{\max}$	$\#opt$	$\#new$	$mean$
HARD28	1	28	28	200	5	–	1.20
BPP FLK	1	160	160	1000	7	–	4.30
SCHOLL	1	1210	1210	500	839	–	0.99
2CBP	2	400	330	201	249	52	2.27
20CBP	20	40	33	201	20	2	2.86

$\#dim$ is the number of dimensions, $\#inst$ is the number of instances in the benchmark, $\#kopt$ is the number of instances whose optimum was previously known, $n_{\max}$ is the maximum number of items in the benchmark, $\#opt$ is the number of optimum solutions obtained with our heuristics, $\#new$ is the number of new optimum found, $mean$ is the average gap in percent between our solutions and the optimum when it is known

heuristics allow to find optimal solutions and prove optimality on 530 instances over the benchmark.

Table 1 summarizes the results on this benchmark. We observe that the combination of heuristics performs very well on all instances. In particular, an optimal packing is obtained on 1120 instances. Furthermore, using the heuristics and lower bound 4, 52 out of the 70 open instances in 2CBP and 2 out of the 7 open instances in 20CBP are solved with proven optimality.

3.5.3 Vector bin packing with heterogeneous bins benchmark

We generated 5 classes of instances for the vector bin packing problem with heterogeneous bins. For each of these classes, we generated 100 feasible instances for each configuration with 10, 30 and 100 bins and 2, 5 and 10 dimensions. The whole test bed contains 4500 generated instances. We define the average usage $u_k \in [0; 1]$ of a bin k as:

$$u_k = \frac{1}{|\mathcal{D}|} \sum_{\substack{j \in \mathcal{D} \\ \text{s.t. } c_k^j \neq 0}} \frac{c_k^j - r_k^j}{c_k^j}$$

We now present the instance classes together with analogies to process assignments in data centers, in order to show the diverse situations that our instances cover.

Instances. In the first class of instances (*Random uniform*), bin capacities are chosen independently using a uniform distribution on [10;1000]. Then, items sizes are independently drawn from a uniform distribution on $[0; 0.8 \times r_k^j]$ until at least 80 % of the bin capacity is used.

These instances account for diversified machines and processes. They could represent data centers with heterogeneous machines and processes. This especially makes sense when few resources are considered (CPU, RAM, disk, bandwidth) however, as the number of resource grows, it is unlikely that such a distribution accounts for realistic instances.

The second class of instances (*Random uniform with rare resources*) is the same as the first one, except that after generating the capacities of a bin, the capacity in dimension d is set to 0 with probability 0.75. Last dimension is a rare resource.

As explained in Sect. 2, in a data center, rare resources model rare components in machines such as GPUs or physical random number generators for instance.

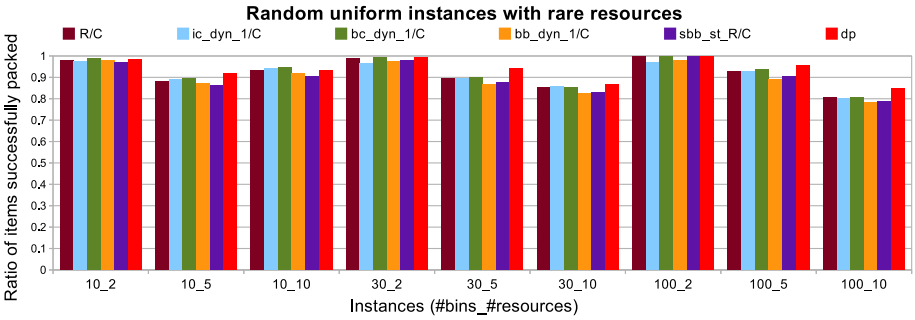

Fig. 1 Random uniform instances with rare resources, average ratios of items packed

In the third class of instances (*Correlated capacities*), for each bin, an integer $b \in [10; 1000]$ is uniformly generated. Then, each capacity $j \in \mathcal{D}$ is set to $0.9 \times b + X_j$ where X_j is an exponentially distributed random variable with rate parameter $1/(0.1 \times b)$ (standard deviation is equal to 10 % of b). Items sizes are generated as in the first and second classes.

This class accounts for a set of machines which are gradually renewed. The characteristics of the machines are improving at a constant rate.

Bins in the fourth class (*Correlated capacities and requirements*) are generated as in the third one. In this class, items are generated similarly to the bins, with $b' \in [1; 0.8 \times r_k^j]$ and until at least 80 % of the bin capacity is used or we failed 100 times to generate a feasible item.

This class is the same as the previous one except that requirements are now growing as the set of machines is upgraded.

In the fifth class of instances (*Similar items and bins*), bin capacities are chosen uniformly and independently on [10;1000]. For each item, size in dimension j is set to $X_j + c_k^j/5$ where X_j is an exponentially distributed random variable with rate parameter $1/(0.2 \times c_k^j/5)$ (standard deviation is equal to 20 % of $c_k^j/5$). Items are generated until at least 70 % of the bin capacity is used or we failed 100 times to generate a feasible item.

In these instances, items are similar to the bins they are contained within. This could occur if new machines are bought to host bunches of similar new processes.

In classes 1–4, on average 85 % of the bins capacities are used. In class 5, 79 % of bins capacities are used on average. Table 2 gives detailed description of average resource usage and number of items in the set of generated instances.

Results. We benchmarked all the 34 heuristics on these instances. In Tables 3, 4, 5, 6, 7, we provide detailed results with the number of feasible solutions obtained on each of the different classes for all different sizes. On instances of class 2, we also present the percent of items packed. Figure 2 shows the total number of success of all heuristics on the whole benchmark.

In Table 3, on random uniform instances, we observe that static best fit, dynamic item centric, dynamic bin centric and dot product heuristics are roughly achieving the same performance. Yet, the non weighted dot product heuristic slightly outperforms other heuristics. In particular, the only heuristic providing feasible solutions with 100 bins and 5 resources. With a rare resource, we see in Table 4 that bin centric heuristics give slightly better results than other heuristics. Since bin centric heuristics focus on the bins rather than the items, they can make better use of the bins, especially if they first consider bins with null rare resources

Table 2 Average bin usage in generated instances

		Class 1 (unif)			Class 2 (rare)			Class 3 (cor. cap.)			Class 4 (cor. cap/req)			Class 5 (similar)		
N	d	n	u (%)	$u_{\max}$ (%)	n	u (%)	$u_{\max}$ (%)	n	u (%)	$u_{\max}$ (%)	n	u (%)	$u_{\max}$ (%)	n	u (%)	$u_{\max}$ (%)
10	2	36	85.8	87.6	34	86.1	88.6	36	85.5	87.6	33	86.9	89.0	39	79.6	81.3
10	5	36	85.1	89.4	36	85.2	90.4	36	85.0	89.6	38	84.7	88.5	39	79.4	82.8
10	10	36	84.6	90.6	36	84.7	91.0	36	84.5	90.5	40	82.6	87.9	39	78.5	82.9
30	2	107	85.8	87.0	102	85.5	87.1	108	85.7	86.9	99	87.0	88.1	119	79.6	80.6
30	5	108	85.1	87.7	109	85.1	88.0	109	85.1	87.8	111	84.3	86.7	119	79.2	81.2
30	10	109	84.6	88.1	109	84.6	88.5	109	84.6	88.4	121	82.5	85.5	118	78.5	81.0
100	2	357	85.8	86.4	342	85.7	86.6	358	85.7	86.4	333	86.9	87.5	397	79.4	79.9
100	5	363	85.1	86.6	363	85.1	86.9	363	85.0	86.5	373	84.2	85.6	397	79.2	80.1
100	10	365	84.7	86.6	364	84.7	86.9	364	84.6	86.7	407	82.5	84.2	394	78.7	80.1
Average:		3.6	85.2	87.8	3.6	85.2	88.2	3.6	85.1	87.8	3.7	84.6	87.0	4.0	79.1	81.1

Each line corresponds to one of the instance sizes. N is the number of bins and d is the number of resources in all these instances. For each class of instances, we give n the average number of items, u the average usage over all resources and $u_{\max}$ the average maximum usage of a resource

Table 3 Results of all heuristics on instances of class 1 (uniform)

	Instances									Results	
	#bins	10	10	10	30	30	30	100	100	100	Total	%feasible
	#resources	2	5	10	2	5	10	2	5	10		
	Avg. #items	36	36	36	107	108	109	357	363	365		
Heuristic	Sorting	#feasible										
Static	None	6	0	0	10	0	0	48	0	0	64	7.1
	Shuffle	7	0	0	9	0	0	57	0	0	73	8.1
	$1/C$	87	1	21	**100**	0	0	**100**	0	0	309	34.3
	$1/R$	87	1	20	**100**	0	0	**100**	0	0	308	34.2
	R/C	88	1	22	**100**	0	0	**100**	0	0	311	34.6
Item centric (dynamic)	shuffle	0	0	0	0	0	0	0	0	0	0	0.0
	$1/C$	81	1	**33**	99	0	0	**100**	0	0	314	34.9
	$1/R$	77	2	29	99	0	0	**100**	0	0	307	34.1
	R/C	85	1	27	100	0	0	**100**	0	0	313	34.8
Bin centric (dynamic)	shuffle	2	0	0	0	0	0	0	0	0	2	0.2
	$1/C$	89	2	22	**100**	0	0	**100**	0	0	313	34.8
	$1/R$	87	2	20	**100**	0	0	**100**	0	0	309	34.3
	R/C	89	**3**	22	**100**	0	0	**100**	0	0	314	34.9
Bin balancing	none	1	0	0	0	0	0	0	0	0	1	0.1
	Static shuffle	3	0	0	0	0	0	0	0	0	3	0.3
	Dynamic shuffle	1	0	0	0	0	0	0	0	0	1	0.1
	Static $1/C$	40	0	15	13	0	0	9	0	0	77	8.6
	Dynamic $1/C$	43	2	20	24	0	0	15	0	0	104	11.6
	Static $1/R$	38	0	14	23	0	0	14	0	0	89	9.9

Table 3 continued

	Instances									Results	
	#bins	10	10	10	30	30	30	100	100	100	Total	%feasible
	#resources	2	5	10	2	5	10	2	5	10		
	Avg. #items	36	36	36	107	108	109	357	363	365		
Heuristic	Sorting	#feasible										
	Dynamic $1/R$	47	0	20	22	0	0	12	0	0	101	11.2
	Static R/C	48	0	14	21	0	0	12	0	0	95	10.6
	Dynamic R/C	43	0	13	13	0	0	14	0	0	83	9.2
Single bin balancing	None	0	0	0	0	0	0	0	0	0	0	0.0
	Static shuffle	2	0	0	1	0	0	0	0	0	3	0.3
	Dynamic shuffle	1	0	0	0	0	0	0	0	0	1	0.1
	Static $1/C$	71	0	15	97	0	0	**100**	0	0	283	31.4
	Dynamic $1/C$	68	0	17	93	0	0	**100**	0	0	278	30.9
	Static $1/R$	73	1	13	97	0	0	**100**	0	0	284	31.6
	Dynamic $1/R$	66	0	20	94	0	0	**100**	0	0	280	31.1
	Static R/C	79	0	15	94	0	0	**100**	0	0	288	32.0
	Dynamic R/C	80	0	10	96	0	0	**100**	0	0	286	31.8
Dot product	Non weighted	**91**	2	31	**100**	0	0	**100**	**38**	0	**362**	**40.2**
	$1/C^2$	88	2	31	**100**	0	0	**100**	14	0	335	37.2
	$1/(s \times C)$	17	0	11	11	0	0	43	0	0	82	9.1

Table 4 Results of all heuristics on instances of class 2 (uniform with rare resources)

	Instances									Results	
	#bins	10	10	10	30	30	30	100	100	100	Total	%feasible
	#resources	2	5	10	2	5	10	2	5	10		
	Avg. #items	34	36	36	102	109	109	342	363	364		
Heuristic	Sorting	#feasible										
Static	None	18	0	0	0	0	0	1	0	0	19	2.1
	Shuffle	20	0	0	1	0	0	0	0	0	21	2.3
	1/*C*	67	5	36	40	0	0	50	0	0	198	22.0
	1/*R*	68	6	32	40	0	0	47	0	0	193	21.4
	R/*C*	69	3	29	65	0	0	**98**	0	0	264	29.3
Item centric (dynamic)	Shuffle	9	0	0	0	0	0	0	0	0	9	1.0
	1/*C*	52	7	35	13	0	0	5	0	0	112	12.4
	1/*R*	50	3	32	6	0	0	0	0	0	91	10.1
	R/*C*	30	0	30	7	0	0	22	0	0	89	9.9
Bin centric (dynamic)	Shuffle	8	0	0	0	0	0	0	0	0	8	0.9
	1/*C*	76	6	**41**	**79**	0	0	97	0	0	**299**	**33.2**
	1/*R*	**80**	**9**	33	73	0	0	**98**	0	0	293	32.6
	R/*C*	68	3	31	70	0	0	97	0	0	269	29.9
Bin balancing	None	9	0	0	0	0	0	0	0	0	9	1.0
	Static shuffle	12	0	0	0	0	0	0	0	0	12	1.3
	Dynamic shuffle	7	0	0	0	0	0	0	0	0	7	0.8
	Static 1/*C*	45	1	24	12	0	0	0	0	0	82	9.1
	Dynamic 1/*C*	53	0	26	10	0	0	5	0	0	94	10.4
	Static 1/*R*	50	1	19	9	0	0	1	0	0	80	8.9

Table 4 continued

	Instances									Results	
	#bins	10	10	10	30	30	30	100	100	100	Total	%feasible
	#resources	2	5	10	2	5	10	2	5	10		
	Avg. #items	34	36	36	102	109	109	342	363	364		
Heuristic	Sorting	#feasible										
	Dynamic $1/R$	58	0	22	15	0	0	0	0	0	95	10.6
	Static R/C	42	1	18	0	0	0	0	0	0	61	6.8
	Dynamic R/C	35	0	20	2	0	0	0	0	0	57	6.3
Single bin balancing	None	16	0	0	1	0	0	1	0	0	18	2.0
	Static shuffle	14	0	0	0	0	0	0	0	0	14	1.6
	Dynamic shuffle	6	0	0	0	0	0	0	0	0	6	0.7
	Static $1/C$	57	0	28	31	0	0	42	0	0	158	17.6
	Dynamic $1/C$	59	2	33	12	0	0	11	0	0	117	13.0
	Static $1/R$	58	1	28	30	0	0	42	0	0	159	17.7
	Dynamic $1/R$	60	0	31	9	0	0	0	0	0	100	11.1
	Static R/C	51	1	19	35	0	0	88	0	0	194	21.6
	Dynamic R/C	48	1	18	35	0	0	82	0	0	184	20.4
Dot product	Non weighted	69	5	28	51	**1**	0	90	0	0	244	27.1
	$1/C^2$	37	1	28	25	0	0	71	0	0	162	18.0
	$1/(s \times C)$	57	0	9	20	0	0	10	0	0	96	10.7

Table 5 Results of all heuristics on instances of class 3 (correlated capacities)

	Instances									Results	
	#bins	10	10	10	30	30	30	100	100	100	Total	%feasible
	#resources	2	5	10	2	5	10	2	5	10		
	Avg. #items	36	36	36	108	109	109	358	363	364		
Heuristic	Sorting	#feasible										
Static	None	31	0	0	48	0	0	95	0	0	174	19.3
	Shuffle	27	0	0	52	0	0	91	0	0	170	18.9
	1/*C*	95	3	0	**100**	5	0	**100**	38	0	341	37.9
	1/*R*	95	2	0	**100**	2	0	**100**	40	0	339	37.7
	R/*C*	95	5	0	**100**	2	0	**100**	38	0	340	37.8
Item centric (dynamic)	Shuffle	4	0	0	1	0	0	0	0	0	5	0.6
	1/*C*	95	3	0	**100**	4	0	**100**	37	0	339	37.7
	1/*R*	94	3	0	**100**	3	0	**100**	28	0	328	36.4
	R/*C*	94	6	0	**100**	7	0	**100**	47	0	354	39.3
Bin centric (dynamic)	Shuffle	11	0	0	8	0	0	5	0	0	24	2.7
	1/*C*	93	2	0	**100**	3	0	**100**	24	0	322	35.8
	1/*R*	93	2	0	**100**	1	0	**100**	17	0	313	34.8
	R/*C*	97	4	0	**100**	2	0	**100**	46	0	349	38.8
Bin balancing	None	7	0	0	0	0	0	0	0	0	7	0.8
	Static shuffle	6	0	0	0	0	0	0	0	0	6	0.7
	Dynamic shuffle	4	0	0	0	0	0	0	0	0	4	0.4
	Static 1/*C*	66	0	0	83	0	0	98	0	0	247	27.4
	Dynamic 1/*C*	74	0	0	83	0	0	99	0	0	256	28.4
	Static 1/*R*	68	0	0	76	0	0	99	0	0	243	27.0

Table 5 continued

	Instances									Results	
	#bins	10	10	10	30	30	30	100	100	100	Total	%feasible
	#resources	2	5	10	2	5	10	2	5	10		
	Avg. #items	36	36	36	108	109	109	358	363	364		
Heuristic	Sorting	#feasible										
	Dynamic $1/R$	85	1	0	91	0	0	97	0	0	274	30.4
	Static R/C	71	0	0	85	0	0	99	0	0	255	28.3
	Dynamic R/C	70	0	0	80	0	0	98	0	0	248	27.6
Single bin balancing	None	16	0	0	15	0	0	35	0	0	66	7.3
	Static shuffle	16	0	0	10	0	0	34	0	0	60	6.7
	Dynamic shuffle	8	0	0	3	0	0	0	0	0	11	1.2
	Static $1/C$	86	0	0	99	0	0	**100**	1	0	286	31.8
	Dynamic $1/C$	88	1	0	**100**	0	0	**100**	1	0	290	32.2
	Static $1/R$	87	0	0	97	0	0	**100**	1	0	285	31.7
	Dynamic $1/R$	86	1	0	99	0	0	**100**	0	0	286	31.8
	Static R/C	89	0	0	99	0	0	**100**	0	0	288	32.0
	Dynamic R/C	87	0	0	98	0	0	**100**	0	0	285	31.7
Dot product	Non weighted	92	**8**	0	**100**	**26**	0	**100**	54	0	380	42.2
	$1/C^2$	**99**	4	0	**100**	21	0	**100**	**99**	0	**423**	**47.0**
	$1/(s \times C)$	30	0	0	48	0	0	80	2	0	160	17.8

Table 6 Results of all heuristics on instances of class 4 (correlated capacities and requirements)

	Instances									Results	
	#bins	10	10	10	30	30	30	**100**	**100**	**100**	Total	%feasible
	#resources	2	5	10	2	5	10	2	5	10		
	Avg. #items	33	38	40	99	111	121	333	373	407		
Heuristic	Sorting	#feasible										
Static	None	50	46	22	82	75	60	95	97	88	615	68.3
	Shuffle	50	45	30	86	69	56	98	94	85	613	68.1
	1/*C*	**100**	99	95	**100**	**100**	**100**	**100**	**100**	**100**	894	99.3
	1/*R*	**100**	99	95	**100**	**100**	**100**	**100**	**100**	**100**	894	99.3
	R/*C*	**100**	99	95	**100**	**100**	**100**	**100**	**100**	**100**	894	99.3
Item centric (dynamic)	Shuffle	17	5	3	4	2	0	0	0	0	31	3.4
	1/*C*	**100**	99	**96**	**100**	**100**	**100**	**100**	**100**	**100**	**895**	**99.4**
	1/*R*	**100**	99	**96**	**100**	**100**	**100**	**100**	**100**	**100**	**895**	**99.4**
	R/*C*	**100**	99	**96**	**100**	**100**	**100**	**100**	**100**	**100**	**895**	**99.4**
Bin centric (dynamic)	Shuffle	27	15	13	18	2	2	20	1	0	98	10.9
	1/*C*	**100**	99	95	**100**	**100**	**100**	**100**	**100**	**100**	894	99.3
	1/*R*	**100**	99	95	**100**	**100**	**100**	**100**	**100**	**100**	894	99.3
	R/*C*	**100**	99	95	**100**	**100**	**100**	**100**	**100**	**100**	894	99.3
Bin balancing	None	21	7	4	2	0	0	0	0	0	34	3.8
	Static shuffle	19	11	6	2	1	1	0	0	0	40	4.4
	Dynamic shuffle	10	7	4	0	0	0	0	0	0	21	2.3
	Static 1/*C*	**100**	**100**	93	**100**	**100**	**100**	**100**	**100**	**100**	893	99.2
	Dynamic 1/*C*	**100**	**100**	93	**100**	**100**	**100**	**100**	**100**	**100**	893	99.2
	Static 1/*R*	**100**	**100**	93	**100**	**100**	**100**	**100**	**100**	**100**	893	99.2

Table 6 continued

	Instances									Results	
	#bins	10	10	10	30	30	30	**100**	**100**	**100**	Total	%feasible
	#resources	2	5	10	2	5	10	2	5	10		
	Avg. #items	33	38	40	99	111	121	333	373	407		
Heuristic	Sorting	#feasible										
	Dynamic $1/R$	**100**	**100**	93	**100**	**100**	**100**	**100**	**100**	**100**	893	99.2
	Static R/C	**100**	**100**	93	**100**	**100**	**100**	**100**	**100**	**100**	893	99.2
	Dynamic R/C	**100**	**100**	93	**100**	**100**	**100**	**100**	**100**	**100**	893	99.2
Single bin balancing	None	36	16	14	56	33	20	66	47	31	319	35.4
	Static shuffle	34	18	11	54	23	13	77	57	26	313	34.8
	Dynamic shuffle	16	14	5	9	2	3	3	0	0	52	5.8
	Static $1/C$	**100**	99	95	**100**	**100**	**100**	**100**	**100**	**100**	894	99.3
	Dynamic $1/C$	**100**	99	95	**100**	**100**	**100**	**100**	**100**	**100**	894	99.3
	Static $1/R$	**100**	99	95	**100**	**100**	**100**	**100**	**100**	**100**	894	99.3
	Dynamic $1/R$	**100**	99	95	**100**	**100**	**100**	**100**	**100**	**100**	894	99.3
	Static R/C	**100**	99	95	**100**	**100**	**100**	**100**	**100**	**100**	894	99.3
	Dynamic R/C	**100**	99	95	**100**	**100**	**100**	**100**	**100**	**100**	894	99.3
Dot product	Non weighted	99	97	93	**100**	**100**	99	**100**	**100**	**100**	888	98.7
	$1/C^2$	**100**	99	**96**	**100**	**100**	**100**	**100**	**100**	**100**	**895**	**99.4**
	$1/(s \times C)$	21	7	3	11	2	1	1	1	0	47	5.2

Table 7 Results of all heuristics on instances of class 5 (similar items and bins)

	Instances									Results	
	#bins	10	10	10	30	30	30	100	100	100	Total	%feasible
	#resources	2	5	10	2	5	10	2	5	10		
	Avg. #items	39	39	39	119	119	118	397	397	394		
Heuristic	Sorting	#feasible										
Static	None	16	0	0	12	0	0	10	0	0	38	4.2
	Shuffle	10	0	0	1	0	0	13	0	0	24	2.7
	1/*C*	37	0	0	47	0	0	66	0	0	150	16.7
	1/*R*	35	0	0	49	0	0	75	0	0	159	17.7
	R/*C*	37	0	0	46	0	0	65	0	0	148	16.4
Item centric (dynamic)	Shuffle	3	0	0	0	0	0	0	0	0	3	0.3
	1/*C*	31	0	0	37	0	0	55	0	0	123	13.7
	1/*R*	31	0	0	24	0	0	27	0	0	82	9.1
	R/*C*	37	0	0	42	0	0	74	0	0	153	17.0
Bin centric (dynamic)	Shuffle	11	0	0	5	0	0	1	0	0	17	1.9
	1/*C*	38	0	0	45	0	0	50	0	0	133	14.8
	1/*R*	33	0	0	31	0	0	40	0	0	104	11.6
	R/*C*	41	0	0	55	0	0	78	0	0	174	19.3
Bin balancing	None	2	0	0	0	0	0	0	0	0	2	0.2
	Static shuffle	4	0	0	0	0	0	0	0	0	4	0.4
	Dynamic shuffle	5	0	0	0	0	0	0	0	0	5	0.6
	Static 1/*C*	19	0	0	1	0	0	0	0	0	20	2.2
	Dynamic 1/*C*	14	0	0	4	0	0	1	0	0	19	2.1
	Static 1/*R*	22	0	0	1	0	0	1	0	0	24	2.7

Table 7 continued

	Instances									Results	
	#bins	10	10	10	30	30	30	100	100	100	Total	%feasible
	#resources	2	5	10	2	5	10	2	5	10		
	Avg. #items	39	39	39	119	119	118	397	397	394		
Heuristic	Sorting	#feasible										
	Dynamic $1/R$	18	0	0	6	0	0	1	0	0	25	2.8
	Static R/C	24	0	0	5	0	0	1	0	0	30	3.3
	Dynamic R/C	18	0	0	4	0	0	0	0	0	22	2.4
Single bin balancing	None	11	0	0	2	0	0	0	0	0	13	1.4
	Static shuffle	13	0	0	5	0	0	1	0	0	19	2.1
	Dynamic shuffle	8	0	0	2	0	0	0	0	0	10	1.1
	Static $1/C$	33	0	0	44	0	0	67	0	0	144	16.0
	Dynamic $1/C$	30	0	0	36	0	0	59	0	0	125	13.9
	Static $1/R$	33	0	0	46	0	0	72	0	0	151	16.8
	Dynamic $1/R$	28	0	0	23	0	0	49	0	0	100	11.1
	Static R/C	33	0	0	38	0	0	71	0	0	142	15.8
	Dynamic R/C	32	0	0	43	0	0	68	0	0	143	15.9
Dot product	Non weighted	**96**	0	11	**100**	0	0	**100**	0	0	307	34.1
	$1/C^2$	85	0	4	**100**	0	0	**100**	0	0	289	32.1
	$1/(s \times C)$	95	**76**	**98**	99	**22**	**87**	**100**	**5**	**30**	**612**	**68.0**

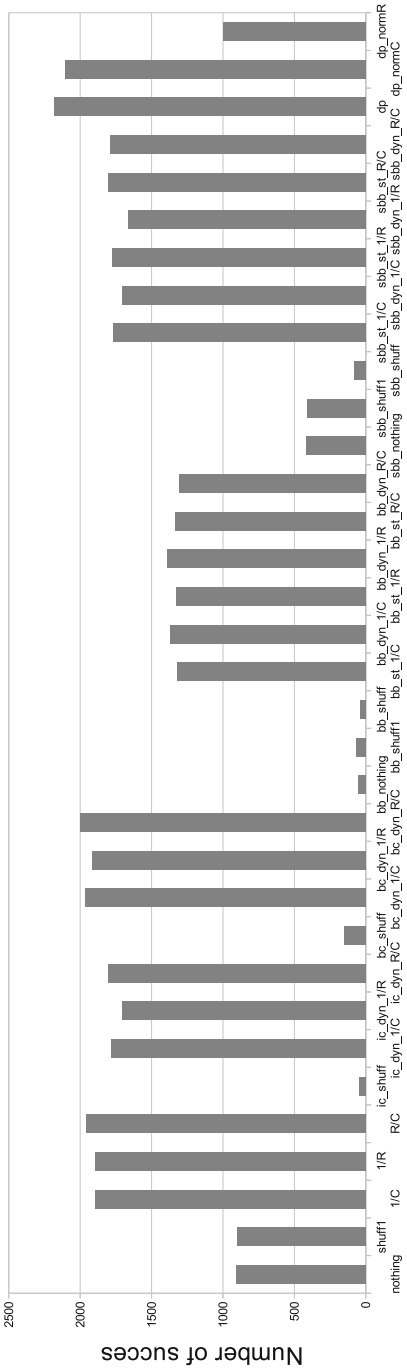

Fig. 2 Total number of feasible assignments obtained by each of the 34 heuristics, on the 4500 generated instances

before considering bins providing rare resources. This helps avoiding to get stuck later on in the algorithm if items with null requirements in the rare resource were packed in bins providing this resource.

Surprisingly, with 10 bins, instances with 10 resources are easier to solve than instances with 5 resources. The reason is that the higher the number of resources, the lower is the probability that an item fits into a different bin other than its initial bin. With very few bins, this actually guides the heuristic.

Figure 1 gives the ratio of items packed using the best heuristic of each class for instances of class 2. Observe that even though heuristics may not provide feasible solutions, 90 % of the items are packed on average. Additionally, the average percentage of items packed depends more on the class of instance than the heuristic used. Yet, there are significant differences regarding feasibility (e.g. between instances with 2 resources and 10 or 30 bins).

In Table 5, we observe that when bin capacities are correlated, heuristics perform very well on instances with few resources while their performances drastically decrease as the number of resources increases. The dot product normalized by bin capacities accounts for these correlations and achieves much better performance than other heuristics.

When both capacities and item sizes are correlated, the problem is almost the same as the single dimensional bin packing decision problem. Table 6, we report results for all heuristics on this case. We observe that all heuristics, except random heuristics and the third dot product, are performing very well and achieve an over 99 % success rate. For the third dot product, we remark that since items and bins are normalized, all items and bins are roughly the same to this heuristic, resulting in an almost random assignment which explains why these results are close to random assignment results. The hardest instances in this case are the ones with few bins and many resources because with more bins it is very likely to have more similar (and exchangeable) items.

On similar instances, Table 7, observe that the third dot product heuristic significantly outperforms all other heuristics. This performance is explained by the measure: notice that on the initial configuration (all items remaining and empty bins), the normalized dot product of an item with its initial bin will be close to 1 with high probability and the normalized dot product with other bins will be smaller than 1. Other heuristics are blind to this similarity criterion.

On this benchmark we observe that as the number of resources grows, the problem quickly becomes much harder. Moreover, dynamic item centric, dynamic bin centric and dot product heuristics outperform bin balancing heuristics in terms of number of feasible solutions found. Figure 2, we report the total number of success of all heuristics on the benchmark. Observe that the non-weighted dot product heuristic has the highest total number of success but does not significantly outperform other heuristics.

Since all these heuristics are very fast to compute, one can consider applying all of them to problem instances, as we did for the vector bin packing benchmark. By combining all heuristics, one can expect a slight improvement over the results, and more importantly, one can expect good results without having to carefully analyze the properties of the instance in order to chose the best suited heuristic.

4 Application to the machine reassignment problem

The machine reassignment problem is a typical example of problems encountered with data centers: several processes are assigned to different servers, in several data centers. The system

needs to be robust to energy or machine failures. Moreover, some processes depend on each other and hence have to run on machines which are *close to* each other. Occasionally they consider moving processes to different servers in order to increase system performance. In the machine reassignment problem, the performance of the system is measured by an aggregated cost and the aim is to minimize it.

The vector bin packing problem with heterogeneous bins is a subproblem of the machine reassignment problem: any feasible assignment for the machine reassignment problem is a feasible VBPHB assignment for the problem defined with items sizes being the processes requirements and bins capacities being the machines capacities. However, there are some additional constraints in the machine reassignment problem:

- *Conflict constraints*: processes are partitioned into services and two processes of the same service cannot be assigned to the same machine.
- *Transient usage constraints*: when a process is moved from one machine to another, some resources (such as disk space) remain used on the first machine. Thus, the process consumes its requirement of these transient resources on both its initial and final machines.
- *Spread constraints*: machines are partitioned into locations and each service s needs to have its processes spread over a minimum number of distinct locations, denoted $spreadMin(s)$.
- *Dependency constraints*: machines are partitioned into neighborhoods and if a service s^a depends on a service s^b, then any process from s^a has to run on some machine having in its neighborhood a machine running a process from s^b.

The goal of the machine reassignment problem is to find a feasible assignment minimizing a weighted cost.

In this paper, we focus on the problem of building feasible assignments of the processes to the machines. Such assignments could be used (in complement of the provided initial solution) as different starting points for approaches such as local search heuristics. In this section, we highlight some structural properties of the machine reassignment problem and show how our heuristics can be adapted to handle the additional constraints. In particular, this is an illustration of how one can tailor the proposed heuristics to real-life problems.

We use the notations of the ROADEF/EURO challenge subject. More precisely, we denote by $\mathcal{M}$ the set of machines, $\mathcal{N}$ the set of neighborhoods, $\mathcal{P}$ the set of processes, $\mathcal{R}$ the set of resources, $\mathcal{TR} \subseteq \mathcal{R}$ the set of transient resources and $\mathcal{S}$ the set of services. $\mathcal{N}$ is a partition of $\mathcal{M}$ and $\mathcal{S}$ is a partition of $\mathcal{P}$. The function $N : \mathcal{M} \to \mathcal{N}$ maps each machine to its neighborhood. The function $M : \mathcal{P} \to \mathcal{M}$ is the assignment: it maps each process to its machine. M_0 denotes the initial assignment. $R(p, r)$ is the requirement of resource $r \in \mathcal{R}$ for the process $p \in \mathcal{P}$. We denote by $C(m, r)$ the capacity of resource $r \in \mathcal{R}$ for the machine $m \in \mathcal{M}$. The two functions $R(r)$ and $C(r)$ are shorthands for $\sum_{p\in\mathcal{P}} R(p, r)$ and $\sum_{m\in\mathcal{M}} C(m, r)$, the overall requirement and capacity on resource r. We denote the initial amount of resource r consumed on machine m by

$$U_0(m, r) = \sum_{\substack{p\in\mathcal{P} \\ s.t.\ M_0(p)=m}} R(p, r)\,.$$

In this problem, an initial feasible solution is provided and is used in particular to define transient usage constraints. We will rely on this initial solution to derive properties and fix some processes assignments.

With constructive heuristics in mind, note that the constraints described above have different natures. Conflict and transient usage constraints can be seen as local: they can be checked

for each machine separately. So these constraints can be verified for any partial assignment. In contrast, spread and dependency constraints are more global: they link the assignments of different machines. These constraints are therefore not well defined for partial assignments. In the following subsections, we present some properties of the constraints of the machine reassignment problem and explain how they can be used to adapt our heuristics.

4.1 Transient usage constraints

Our heuristics can integrate transient usage constraints by modifying the bins capacities and the processes requirements. For each machine $m \in \mathcal{M}$, we set its capacity in a non-transient resource r_i to $C(m, r_i)$ while we set its capacity in a transient resource r_j to $C(m, r_j) - U_0(m, r_j)$. Then, processes requirements depend on machines: for all $r \in \mathcal{R} \backslash \mathcal{TR}$ and all machines, they are equal to $R(p, r)$, while for all $r \in \mathcal{TR}$ they are equal to 0 for the machine $M_0(p)$ and to $R(p, r)$ otherwise. In other words, when a process is assigned to its initial machine, the capacity constraints on transient resources are always satisfied.

These constraints can be taken into account when sizes are computed. We can decide, for instance, that processes with huge requirements on some transient resources will not be moved. Moreover, observe that if a process is moved from its initial machine, then for all of its transient resources, the space used is lost. Hence, we have the following property:

Property 2 For each process $p \in \mathcal{P}$, if there is a transient resource $r \in \mathcal{TR}$ such that $R(p, r) > C(r) - R(r)$, then in every feasible assignment, p has to be assigned to its initial machine.

Proof Let p be a process and r a transient resource such that $R(r) > C(r) - R(p, r)$. If process p is moved, since r is transient, a space $R(p, r)$ on machine m cannot be used by any process. Hence, the total available space for all processes in resource r is $C(r) - R(p, r)$, which is smaller than the total requirement. Therefore, any assignment M with $M(p) \neq M_0(p)$ is not feasible. □

Using Property 2, we can determine that some processes cannot be moved. In such cases, we can fix them to their initial machines. If we are interested in moving a set of processes P, then we obtain the following corollary:

Corollary 1 *Let $P \subseteq \mathcal{P}$ be a subset of processes. If there is a transient resource $r \in \mathcal{TR}$ such that $\sum_{p \in P} R(p, r) > C(r) - R(r)$, then in every feasible assignment, at least one process from P is assigned to its initial machine.*

In a greedy approach, Property 2 and Corollary 1 can be used with C and R, the *remaining* capacities and requirements. Moreover, thanks to these properties, one can fix items or conclude—before being unable to pack an item—that an intermediate solution (a partial assignment) is infeasible. Notice that Property 2 and its corollary can also be used for further optimization purposes.

4.2 Conflict constraints

In order to satisfy conflict constraints, when trying to assign a process p from a service s to a machine m, one just needs to check that there is no process from service s which is already assigned to m.

4.3 Spread constraints

A simple way to make sure that these constraints are satisfied is the following: for each service $s \in \mathcal{S}$, take a subset of processes $P \subseteq s$ such that $|P| = spreadMin(s)$, and assign all processes of P to distinct locations. To make sure that there is a feasible solution, we use the initial solution to choose a subset of processes which will be assigned to their initial machines.

4.4 Dependency constraints

Dependency constraints are difficult constraints to cope with, because they bound processes to each other and can be cyclic. We propose to take advantage of these constraints to decompose the problem into smaller subproblems where all dependency constraints are satisfied. More precisely, let $g \in \mathcal{N}$ be a neighborhood, $m_1, m_2 \in g$ and $p \in \mathcal{P}$. Note that if M is a feasible assignment with $M(p) = m_1$, then, setting $M(p) = m_2$ does not violate any dependency constraint. We can even generalize this property to all the processes from any neighborhood into any other neighborhood:

Property 3 Let M be a feasible assignment. Denote by P_n the set of processes assigned to neighborhood $n \in \mathcal{N}$: $P_n = \{p \in \mathcal{P} : M(p) \in n\}$. Any assignment M' such that $\forall n \in \mathcal{N}, \quad \forall p_1, p_2 \in P_n, \ N(M'(p_1)) = N(M'(p_2))$, satisfies all dependency constraints.

Proof Let $s^a, s^b \in \mathcal{S}$, s^a depends on s^b. Let $p \in s^a$. The assignment M is feasible, hence $\exists p' \in s^b$ such that $M(p') \in N(M(p))$. Moreover $p, p' \in P_{N(M(p))}$. Therefore $p' \in N(M'(p))$. □

Property 3 implies that if one takes the processes of a given neighborhood and reassign all of them to a same neighborhood, then the new assignment satisfies all dependency constraints.

We use Property 3 with $M = M_0$ to decompose the problem into as many subproblems as there are neighborhoods. Each subproblem consists in finding an assignment for the processes of a given neighborhood either into itself, or into another neighborhood. In this latter case, recall that all transient resources used by the processes are lost. Hence, we have to make sure that Corollary 1 does not immediately induce that there is no feasible assignment. Moreover, such reassignment also implies that every process will be moved, probably resulting in large move costs.

4.5 Experiments

In this section, we apply several variants of VBPHB heuristics to machine reassignment problems.

Test problems. We use the 30 instances (sets A, B and X) provided during the ROADEF/EURO challenge. They are realistic instances, randomly generated according to real-life Google statistics. The instances contain up to 5, 000 machines, 50, 000 processes and 12 resources. More details on the instances can be found on the challenge web page.[2]

Implemented heuristics. Combining the above ideas to handle the additional constraints, our algorithm proceeds as follows. First, some processes are assigned to their initial machines in order to satisfy the spread constraints. In our experiments, on average 26 % of the processes are assigned during this phase. Then, we decompose the problem into smaller independent

[2] http://challenge.roadef.org/2012/files/Roadef-results.pdf.

subproblems. We define a subproblem by selecting all processes initially assigned to a neighborhood and the aim is to find a feasible assignment of these processes into this neighborhood. This makes dependency constraints automatically satisfied by any feasible assignment of the subproblems. We apply our VBPHB heuristics to each neighborhood with conflict and transient usage constraints are checked on the fly. Finally, the subproblems assignments are combined to form the global solution.

We implemented the different types of VBPHB heuristics: item centric, bin centric and bin balancing. For each type, we used several measures, including the static $1/C$, $1/R$ and R/C measures, and the dynamic dot product and process priority measures. We also combined these measures with random orderings. In this case, we report the average results over 50 runs.

We implemented these heuristics in C++ using efficient data structures. Although there is still room for code optimization, we will see below that most heuristics are already very fast.

Results. In order to compare the different heuristics even on instances where they do not find feasible assignments, we report the percentage of assigned processes. A reported 100 % means that the heuristic found a feasible solution. Tables 8 and 9 present the results of each heuristic on each instance. The second and third rows of the tables describes the sorting used: "Rand" stands for random and "Prio" for priority. The processes (Proc) are the items and the machines (Mach) are the bins. The percentage of assigned items is in bold font when the assignment is feasible. If no heuristic finds a feasible assignment, the best percentage of assigned items is in italic font.

We observe that all heuristics find feasible solutions and assign a high percent of processes in average. We also note that for instances where feasible solutions are found, the different heuristics are complementary.

Regarding running times, bin balancing variants with static measures are the fastest: they take less than 1 s to solve all the instances. Bin centric static variants take a few seconds. As expected, the slowest variants are the ones using dynamic measures. In particular, the bin centric dot product heuristic does not scale well to large instances.

In terms of number of feasible solutions found, the best heuristics are the item centric heuristics with priorities on processes and machines ordered randomly or normalized by bins capacities. We observe that all heuristics assign almost all processes on almost all instances. Moreover, even heuristics with the lowest percentage of assigned items are useful. For example, the bin balancing heuristic with normalizations $1/R$ on processes and $1/C$ on machines is the only heuristic which finds a feasible assignment for instance `b_3`. Notice that the heuristics with the highest percentage of assigned processes in average (bin balancing "prio proc–rand mach" and "rand proc–$1/C$ mach") are also the ones with some of the smallest numbers of feasible instances. These two heuristics might however be very useful to provide almost feasible solutions if a repairing algorithm (such as a feasibility-focused local search heuristic) is available.

We remark that for instance `a_1_4`, since each neighborhood is reduced to one machine, our neighborhood decomposition makes all the heuristics find the initial solution.

Finally, observe that by combining the different heuristics, a feasible assignment is found on 16/30 instances. Out of the 16 feasible instances, 2 are from the instance set A and 7 from each of the sets B and X. We observe that our heuristics are more likely to find solutions when the capacity constraints are not so tight; with an average ratio $R(r)/C(r)$ below 85 %.

Table 8 Results of bin centric heuristics using different measures, on ROADEF/EURO challenge machine reassignment instances. For each variant, the percentage of processes successfully assigned (column "%") and the CPU time (in seconds) are reported.

	Item Centric Heuristics				**Bin Centric Heuristics**											
	Prio Proc Rand Mach		Prio Proc 1/*C* Mach		Rand Proc Rand Mach		1/*C* Proc 1/*C* Mach		1/*R* Proc 1/*C* Mach		1/*C* Proc Rand Mach		Rand Proc 1/*C* Mach		Dot Prod Proc 1/*C* Mach	
Instance	%	time	%	time	%	time	%	time	%	time	%	time	%	time	%	time
a_1_1	97	0.00	98	0.00	*99*	0.00	88	0.00	88	0.00	89	0.00	98	0.00	88	0.00
a_1_2	92.1	0.02	*92.9*	0.02	91.9	0.00	79.3	0.00	79.3	0.00	79.8	0.00	92.2	0.00	80.1	0.13
a_1_3	*97.5*	0.00	97.2	0.00	96.7	0.00	94	0.00	94.7	0.00	96	0.00	96.5	0.00	95.2	0.01
a_1_4	**100**	0.00	**100**	0.00	**100**	0.00	**100**	0.00	**100**	0.00	**100**	0.00	**100**	0.00	**100**	0.00
a_1_5	*99.9*	0.01	97.2	0.01	96.7	0.00	74.4	0.00	78	0.00	76	0.00	94.8	0.00	81.7	0.02
a_2_1	96.4	0.02	96.9	0.02	98.2	0.00	**100**	0.00	**100**	0.00	**100**	0.00	97.9	0.00	**100**	0.73
a_2_2	96.4	0.01	96.8	0.01	96.7	0.00	*98.3*	0.00	*98.3*	0.00	98	0.00	96.8	0.00	97.8	0.01
a_2_3	96.9	0.01	97.1	0.01	97.1	0.00	98.6	0.00	98.7	0.00	98.7	0.00	96.7	0.00	*99.1*	0.01
a_2_4	*97.5*	0.01	96.7	0.01	96.4	0.00	95	0.00	95.3	0.00	92.8	0.00	96.2	0.00	94.4	0.01
a_2_5	94.2	0.01	95.3	0.01	95.1	0.00	89.1	0.00	88.3	0.00	87.7	0.00	95.2	0.00	89.4	0.01
b_1	97.1	0.25	97.5	0.26	97.2	0.01	81.5	0.01	81.5	0.01	82.3	0.01	96.7	0.00	74.7	0.69
b_2	87.8	0.24	86	0.24	85.8	0.01	57.6	0.01	57.1	0.01	57.9	0.01	86.6	0.01	57.5	0.80
b_3	99.9	3.26	99.9	3.43	99.9	0.02	99.4	0.02	99.4	0.03	99.7	0.02	99.9	0.02	96.2	13.67
b_4	**100**	1.37	99.9	1.39	99.9	0.06	96.8	0.09	97.4	0.09	96.6	0.09	**100**	0.06	89.3	36.21
b_5	**100**	14.64	**100**	16.00	**100**	0.05	**100**	0.05	**100**	0.05	**100**	0.05	**100**	0.05	98.4	65.07
b_6	**100**	8.54	**100**	8.77	**100**	0.07	73.3	0.11	71.1	0.12	71.6	0.12	**100**	0.07	72.3	128.21
b_7	**100**	9.17	**100**	9.41	**100**	0.92	**100**	1.47	**100**	1.45	**100**	1.28	**100**	0.90	**100**	1886.06
b_8	99.8	15.38	**100**	17.73	99.9	0.05	**100**	0.07	**100**	0.07	**100**	0.06	99.9	0.05	99.9	61.17
b_9	97.8	6.63	97.7	6.87	98.6	0.31	80	0.50	79.5	0.51	83.1	0.50	98.5	0.31	86.8	397.66
b_10	**100**	7.33	**100**	7.54	**100**	0.92	**100**	1.56	**100**	1.55	**100**	1.32	**100**	0.91	**100**	1368.69

Table 8 continued

	Item Centric Heuristics				**Bin Centric Heuristics**											
	Prio Proc Rand Mach		Prio Proc 1/*C* Mach		Rand Proc Rand Mach		1/*C* Proc 1/*C* Mach		1/*R* Proc 1/*C* Mach		1/*C* Proc Rand Mach		Rand Proc 1/*C* Mach		Dot Prod Proc 1/*C* Mach	
Instance	%	time	%	time	%	time	%	time	%	time	%	time	%	time	%	time
x_1	97.2	0.25	97.1	0.25	96.8	0.01	80	0.01	80.8	0.01	81.1	0.01	96.7	0.00	79.5	0.63
x_2	86.1	0.24	85.7	0.24	85.9	0.01	58.6	0.01	58.4	0.01	58	0.01	86	0.01	56.5	0.84
x_3	99.9	3.24	99.9	3.43	99.9	0.02	99.1	0.02	98.8	0.03	99.5	0.02	99.9	0.02	90.7	14.79
x_4	**100**	1.15	**100**	1.18	**100**	0.06	96.2	0.09	95.2	0.09	98.9	0.08	**100**	0.05	89.2	29.94
x_5	**100**	14.53	**100**	15.91	**100**	0.05	**100**	0.05	**100**	0.05	**100**	0.05	**100**	0.05	96.3	66.74
x_6	**100**	8.27	**100**	8.48	**100**	0.07	70.6	0.11	69.1	0.12	70.8	0.12	**100**	0.07	71.9	124.91
x_7	**100**	9.63	**100**	9.87	**100**	0.95	**100**	1.52	**100**	1.52	**100**	1.33	**100**	0.95	**100**	2012.85
x_8	**100**	15.44	**100**	17.83	**100**	0.05	**100**	0.06	**100**	0.06	**100**	0.06	**100**	0.05	98.1	58.98
x_9	98.2	6.46	98.6	6.68	99	0.31	89.6	0.51	89.6	0.51	91.7	0.49	99.1	0.31	91.2	381.08
x_10	**100**	7.12	**100**	7.33	**100**	0.92	**100**	1.53	**100**	1.50	**100**	1.31	**100**	0.90	**100**	1310.05
Avg/sum	97.7	133.2	97.7	142.9	97.7	4.8	90	7.81	90	7.77	90.3	6.95	97.6	4.81	89.1	7960.00
#feasible	**12**		**12**		11		10		10		10		**12**		6	

Table 9 Results of bin balancing heuristics using different measures, on ROADEF/EURO challenge machine reassignment instances

	Bin balancing heuristics													
	1/*C* Proc 1/*C* Mach		1/*R* Proc 1/*C* Mach		1/*C* Proc Rand Mach		Rand Proc Rand Mach		Prio Proc 1/*C* Mach		Prio Proc Rand Mach		Rand Proc 1/*C* Mach	
Instance	%	Time	%	Time	%	Time	%	Time	%	Time	%	Time	%	Time
a_1_1	97	0.00	95	0.00	91	0.00	*99*	0.00	*99*	0.00	*99*	0.00	*99*	0.00
a_1_2	76.6	0.00	77.3	0.00	77.5	0.00	89.8	0.00	88.6	0.02	90.2	0.02	89.8	0.00
a_1_3	94.5	0.00	93.7	0.00	95	0.00	95.8	0.00	96.2	0.00	96.1	0.00	95.9	0.00
a_1_4	**100**	0.00	**100**	0.00	**100**	0.00	**100**	0.00	**100**	0.00	**100**	0.00	**100**	0.00
a_1_5	81.8	0.00	85.4	0.00	77.7	0.00	96.5	0.00	95.7	0.01	96.3	0.01	96.4	0.00
a_2_1	62.7	0.00	61.5	0.00	61.5	0.00	98.6	0.00	98.6	0.02	98.7	0.02	98.7	0.00
a_2_2	96	0.00	97.3	0.00	97.1	0.00	96	0.00	95.6	0.01	96.1	0.01	96	0.00
a_2_3	97.2	0.00	97.2	0.00	97.4	0.00	96.6	0.00	95.9	0.01	96.5	0.01	96.5	0.00
a_2_4	93.8	0.00	88.1	0.00	90.9	0.00	96.2	0.00	96.1	0.01	96.5	0.01	96.2	0.00
a_2_5	84.7	0.00	85.6	0.00	87.2	0.00	95.2	0.00	*95.9*	0.01	95.3	0.01	95.3	0.00
b_1	82.8	0.00	82.5	0.00	83.4	0.00	96.8	0.00	*97.6*	0.25	97.3	0.25	96.8	0.00
b_2	58.5	0.00	59.8	0.00	60.9	0.00	92.9	0.00	90	0.24	92	0.24	*93.1*	0.00
b_3	99.8	0.02	**100**	0.02	99.5	0.02	99.8	0.01	99.9	3.37	99.8	3.44	99.8	0.01
b_4	92.6	0.02	94.7	0.02	94	0.02	**100**	0.01	**100**	1.31	**100**	1.32	**100**	0.01
b_5	**100**	0.03	**100**	0.03	**100**	0.03	99.9	0.03	99.9	15.05	99.9	16.06	99.9	0.03
b_6	**100**	0.02	**100**	0.02	**100**	0.02	**100**	0.02	**100**	8.66	**100**	8.71	**100**	0.02
b_7	**100**	0.03	99.9	0.03	99.9	0.03	99.7	0.03	99.7	8.57	99.7	8.65	99.7	0.03
b_8	**100**	0.04	**100**	0.04	**100**	0.04	**100**	0.03	**100**	16.02	99.9	17.94	**100**	0.03
b_9	72.6	0.27	72.4	0.27	73.9	0.27	99.9	0.03	99.9	6.42	*99.9*	6.45	99.9	0.02
b_10	**100**	0.02	**100**	0.03	**100**	0.03	99.9	0.03	99.9	6.38	99.9	6.41	99.9	0.03

Table 9 continued

	Bin balancing heuristics													
	1/*C* Proc 1/*C* Mach		1/*R* Proc 1/*C* Mach		1/*C* Proc Rand Mach		Rand Proc Rand Mach		Prio Proc 1/*C* Mach		Prio Proc Rand Mach		Rand Proc 1/*C* Mach	
Instance	%	Time	%	Time	%	Time	%	Time	%	Time	%	Time	%	Time
x_1	82.1	0.00	84.2	0.00	82	0.00	96.6	0.00	95.8	0.25	96.6	0.25	96.6	0.00
x_2	62.8	0.00	60.8	0.00	59.8	0.00	92.4	0.00	*94.3*	0.24	92.8	0.24	92.9	0.00
x_3	99.7	0.02	**100**	0.02	99.4	0.02	99.8	0.01	99.8	3.34	99.8	3.43	99.8	0.01
x_4	95.2	0.02	92.3	0.02	95.9	0.02	**100**	0.01	**100**	1.10	**100**	1.10	**100**	0.01
x_5	**100**	0.03	**100**	0.03	**100**	0.03	99.9	0.03	99.9	14.95	99.9	15.98	99.9	0.03
x_6	**100**	0.02	**100**	0.02	**100**	0.02	**100**	0.02	**100**	8.39	**100**	8.43	**100**	0.02
x_7	99.7	0.04	99.8	0.04	99.6	0.04	99.6	0.04	99.6	9.03	99.6	9.08	99.6	0.03
x_8	**100**	0.04	**100**	0.04	**100**	0.04	**100**	0.03	99.9	16.06	99.9	18.08	**100**	0.03
x_9	73.1	0.25	74.7	0.24	77.6	0.23	99.9	0.03	99.9	6.26	*99.9*	6.29	*99.9*	0.02
x_10	**100**	0.02	**100**	0.02	**100**	0.03	99.9	0.03	99.9	6.22	99.9	6.25	99.9	0.03
Avg/sum	90.1	0.92	90.1	0.92	90	0.93	98	0.41	97.9	132.19	*98.1*	138.6	*98.1*	*0.38*
#feasible	10		11		9		7		6		5		7	

For each variant, the percentage of processes successfully assigned (column "%") and the CPU time (in seconds) are reported

5 Conclusion

In this paper, we have introduced the vector bin packing with heterogeneous bins problem, a generalization of the vector bin packing problem that can model several real-life problems. We have proposed families of heuristics for this problem, including adaptation of the well-known first fit and best fit bin packing heuristics, as well as some new variants taking advantage of the multidimensional resources and variable bin sizes. To be pertinent for real-life problems, our heuristics are fast, flexible and easy to implement. Thanks to their efficiency, we can combine all of them and apply them all on any instance, getting the best of each heuristic without having to analyze instances to pick the heuristic which is the most likely to be successful.

Although the heuristics were designed for the vector bin packing with heterogeneous bins problem, they can also be used to solve vector bin packing problems. By combining our heuristics with a simple lower bounding procedure, we were able to solve and prove optimality for 54 out of the 77 instances left open in the vector bin packing benchmark from Brandao and Pedroso (2013).

We have analyzed the machine reassignment problem and presented some of its properties in order to adapt our VBPHB heuristics to the problem of finding feasible assignments. We were able to generate new feasible solutions for Google machine reassignment instances, which are in particular interesting as various starting points for optimization algorithms.

In future works, one can experiment with more sophisticated measures, possibly based on a relaxation of the problem, or the *permutation pack* and *choose pack* heuristics of Leinberger et al. (1999). When considering greedy or constructive assignment-based approaches, one can also reason on partial solutions and infer that some items have to be packed in a subset of the remaining bins. We can use constraint programming to implement such an approach: propagate decisions taken by the heuristic, then take next decisions using updated domains.

Python implementations of the heuristics as well as our programs for the challenge have been published[3] under open source licenses.

References

Aarts, E., & Lenstra, J. K. (1997). *Local search in combinatorial optimization*. Princeton: Princeton University Press.

Bansal, N., Caprara, A., & Sviridenko, M. (2006). Improved approximation algorithms for multidimensional bin packing problems. In *Foundations of Computer Science*, IEEE (pp. 697–708).

Brandao, F., & Pedroso, J. P. (2013) Bin packing and related problems: General arc-flow formulation with graph compression. *arXiv preprint* (13106887).

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

Caprara, A., Kellerer, H., & Pferschy, U. (2003). Approximation schemes for ordered vector packing problems. *Naval Research Logistics*, *50*(1), 58–69.

Chang, S. Y., Hwang, H. C., & Park, S. (2005). A two-dimensional vector packing model for the efficient use of coil cassettes. *Computers & Operations Research*, *32*(8), 2051–2058.

Chekuri, C., & Khanna, S. (1999) On multi-dimensional packing problems. In *Symposium On Discrete Algorithms* (pp. 185–194).

de Carvalho, J. M. V. (1999). Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research*, *86*, 629–659.

Falkenauer, E. (1996). A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, *2*(1), 5–30.

Feo, T. A., & Resende, M. G. C. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, *8*(2), 67–71.

[3] https://github.com/TeamJ19ROADEF2012/.

Feo, T. A., & Resende, M. G. C. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, *6*(2), 109–133.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. New York: Freeman.

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

Gavranović, H., Buljubašić, M., & Demirović, E. (2012). Variable neighborhood search for Google machine reassignment problem. *Electronic Notes in Discrete Mathematics*, *39*, 209–216.

Han, B. T., Diehr, G., & Cook, J. S. (1994). Multiple-type, two-dimensional bin packing problems: Applications and algorithms. *Annals of Operations Research*, *50*(1), 239–261.

Karp, R. M., Luby, M., & Marchetti-Spaccamela, A. (1984). A probabilistic analysis of multidimensional bin packing problems. In *Symposium on Theory of Computing*, ACM (pp. 289–298).

Kou, L. T., & Markowsky, G. (1977). Multidimensional bin packing algorithms. *IBM Journal of Research and Development*, *21*(5), 443–448.

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

Leinberger, W., Karypis, G., & Kumar, V. (1999). Multi-capacity bin packing algorithms with applications to job scheduling under multiple constraints. In *International Conference on Parallel Processing*, IEEE (pp. 404–412).

Lopes, R., Morais, V. W., Noronha, T. F., & Souza, V. A. (2014). Heuristics and matheuristics for a real-life machine reassignment problem. *International Transactions in Operational Research*, *22*(1), 77–95.

Maruyama, K., Chang, S., & Tang, D. (1977). A general packing algorithm for multidimensional resource requirements. *International Journal of Computer & Information Sciences*, *6*(2), 131–149.

Mehta, D., O'Sullivan, B., & Simonis, H. (2012). Comparing solution methods for the machine reassignment problem. In M. Milano (Eds.), *Principles and practice of constraint programming, Proceedings of 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012* (pp. 782–797). Berlin: Springer.

Panigrahy, R., Talwar, K., Uyeda, L., & Wieder, U. (2011). Heuristics for vector bin packing. *Microsoft Research*. Technical report.

Portal, G. M. (2013). An algorithmic study of the machine reassignment problem. Master's thesis, Universidade Federal do Rio Grande do Sul.

Schoenfield, J. E. (2002). Fast, exact solution of open bin packing problems without linear programming. *US Army Space and Missile Defense Command*. Technical report.

Scholl, A., Klein, R., & Jürgens, C. (1997). Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research*, *24*(7), 627–645.

Shachnai, H., & Tamir, T. (2003). Approximation schemes for generalized 2-dimensional vector packing with application to data placement. In S. Arora, K. Jansen, J. D. P. Rolim, A. Sahai (Eds.), *Approximation, randomization and combinatorial optimization. Algorithms and techniques: 6th international workshop on approximation algorithms for combinatorial optimization problems, APPROX 2003 and 7th international workshop on randomization and approximation techniquesin computer science, RANDOM 2003, Princeton, NJ, USA, August 24-26, 2003. Proceedings* (pp. 165–177). Berlin: Springer.

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

Stillwell, M., Schanzenbach, D., Vivien, F., & Casanova, H. (2010). Resource allocation algorithms for virtualized service hosting platforms. *Journal of Parallel and Distributed Computing*, *70*(9), 962–974.

Woeginger, G. J. (1997). There is no asymptotic PTAS for two-dimensional vector packing. *Information Processing Letters*, *64*(6), 293–297.

Yao, A. C. C. (1980). New algorithms for bin packing. *Journal of the ACM*, *27*(2), 207–227.