J Supercomput (2017) 73:5516–5538
DOI 10.1007/s11227-017-2100-0

Optimal multi-dimensional vector bin packing using simulated evolution

Sadiq M. Sait[1,2] · **Kh. Shahzada Shahid**[2]

Published online: 30 June 2017

Abstract The use of the evolutionary heuristic simulated evolution for the optimization of the multi-dimensional vector bin packing problem, which is encountered in several industrial applications, is described. These applications range from production planning and steel fabrication to assignment of virtual machines (VMs) onto physical hosts at cloud-based data centers. The dimensions of VMs can include demands of CPU, memory, bandwidth, disk space etc. The generalized *goodness* functions that aid traversing the search space in an intelligent manner are designed to cater to the multidimensional nature of items (VMs). The efficiency of heuristics is tested by considering phase transition in the generation of difficult test cases. The quality of the heuristics is judged by determining how close the solution is to the estimated lower bound. A new implementation of a tighter lower bound is proposed. Experiments show that superior quality results are obtained by employing the proposed strategy.

Keywords Combinatorial optimization · Evolutionary metaheuristic · Simulated evolution · Virtual machine placement · NP-hard · Nondeterministic algorithms · Vector bin packing problem

✉ Sadiq M. Sait
sadiq@kfupm.edu.sa

Kh. Shahzada Shahid
khawajas@kfupm.edu.sa

1 Center for Communications and IT Research, Research Institute, King Fahd University of Petroleum & Minerals, Dhahran 31261, Kingdom of Saudi Arabia

2 Department of Computer Engineering, King Fahd University of Petroleum & Minerals, Dhahran 31261, Kingdom of Saudi Arabia

1 Introduction

In recent years, cloud-based data centers have emerged as a popular choice for hosting and delivering IT services. In this approach, a user can request as much or as little computing resources as one desires. Users' requests are mapped to virtual machines (VMs) with a specific amount of CPU, memory, bandwidth, and disk space demands. Each physical machine (PM) of a data center has a fixed capacity of these resources. Moreover, when multiple VMs are placed on a single PM, the host operating system or hypervisor also consumes some extra resources, e.g., for resource scheduling, or context switching [1]. This is mainly to avoid performance degradation of the PMs. An upper-bound on the maximum utilization of any resource of a PM is generally set with some threshold value. Optimizing the total number of PMs required is important for the cost-effective and energy-efficient operation of data centers [2–4].

The optimization of the VM assignment problem is a classic case of a multi-dimensional vector bin packing problem (MDVBPP), where several items are defined by an R-dimensional resource requirement vector. These items have to be packed into a minimum number of bins with fixed resource capacity in each of these R-dimensions. Figure 1 illustrates the packing of five 4-dimensional items into two bins. MDVBPP is an NP-hard problem. Even its one-dimensional version which reduces to 1D-BPP is NP-Hard [5].

A large amount of work on MDVBPP has been published. This is due to the large number of real-world applications where this problem is encountered. Examples of MDVBPP applications include but are not limited to the following fields: computer network design [6], computer sciences (assignment problems, e.g., virtual machine placement (VMP) in a data center [7], assignment of jobs to processors, file placement for a multi-device storage system [8]), layout design [9], robot selection and workstation assignment [10], production planning and logistics (packing problems) [11], steel industry [12], etc. The widespread use of MDVBPP is due to the fact that it efficiently

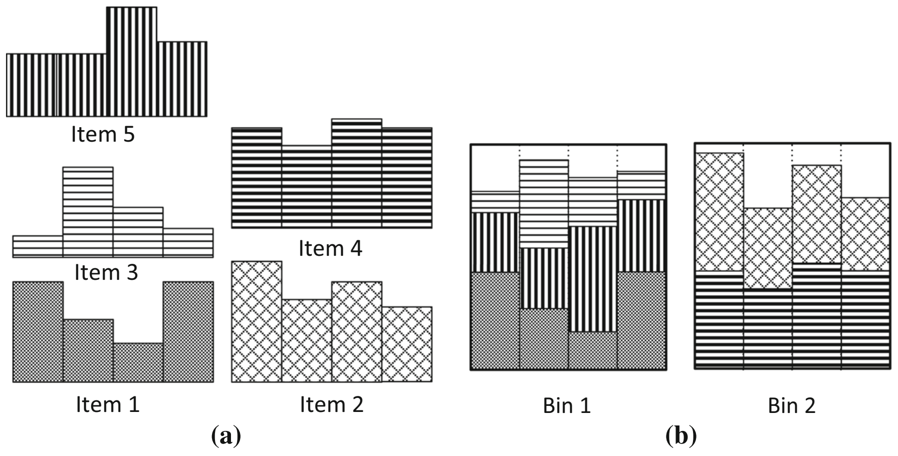


Fig. 1 Example of MDVBPP for $R = 4$. **a** Five items with 4-D resource requirement to be assigned. **b** Assignment of five items in two bins

models the scheduling or assignment requirements that are encountered in several disciplines. Lately, MDVBPP has gained popularity among the research community working on the virtual machine (VM) assignment problems of data centers.

Over the past few decades, several exact methods, lower bounds, approximation methods, and their worst-case analysis for the 2-D version of MDVBPP have been described in the literature [8,13–16]. Even for two dimensions, i.e., when $R = 2$, MDVBPP is APX-hard, which means that an asymptotic polynomial time approximation scheme does not exist unless $P = \mathrm{NP}$ [17]. For R-dimensional MDVBBP, Yao [18] showed that any algorithm that runs in $O(n \log n)$-time cannot give a solution with the number of bins less than R times the optimum value. Fernandez de la Vega and Lueker [19] proved that for MDVBPP, a linear time algorithm can find a solution with $R + \epsilon$ times the number of bins in the optimal solution, where ϵ is a positive constant. Chekuri and Khanna [20] described a polynomial time approximation algorithm that further improves this worst-case performance ratio to $(1 + \epsilon.R + O(\ln \epsilon^{-1}))$, which reduces to $O(\ln R)$-approximation when R is fixed.

Variants of First Fit Decreasing (FFD) are the most common deterministic methods applied to finding an approximate solution for MDVBPP [21]. FFD first sorts the items in decreasing order of their sizes and then places them into bins according to First Fit (FF) strategy. Here "size" could be any measure that effectively combines the size of the multi-dimensional item into a single scalar value, e.g., the size can be defined as a sum or multiple of all dimensions. The efficiency of an FFD-variant highly depends on the definition of the size. In the context of the VMP problem, Lei Shi et al. [21] presented and evaluated the performance of six different FFD-based algorithms for a 2-D version of MDVBPP. Stillwell et al. [22] presented a performance comparison of several FFD-variants for MDVBPP. They sampled demands across different dimensions generated for the VMP problem instances, independently and an identically distributed manner. However, the VMP problem instances encountered in the real-world scenarios may have complementary resource requirements across different dimensions. For example, a VM request with scientific computations may have high CPU requirements but low I/O requirements while VMs that are acting as web servers would behave in the opposite manner. FFD-variants that are not designed to take advantage of such complementary requests can give solutions that are far from the optimal solution [17]. Panigrahy et al. systematically studied the family of FFD heuristics and their limitations and suggested a new geometric based heuristic approach for MDVBPP [17].

Recently, the use of a genetic algorithm [2], particle swarm optimization [23], ant colony system algorithm [3,24], and cuckoo search optimization [7] have been attempted for optimizing the VM placement problem in the category of non-deterministic metaheuristics. However, these were implemented for the one or two-dimensional problems. There are only a very few applications of metaheuristics reported for $R \geq 2$. For $R \geq 2$, Stillwell et al. [22] showed that FFD-based heuristics perform better than a genetic algorithm.

In this paper, the application of a heuristic optimization based on simulated evolution (SimE) is described and its performance for MDVBPP is evaluated against another well-known iterative heuristic, simulated annealing (SA). The performance of the proposed optimization method was also compared with two popular FFD-based heuristics that out-performed others, referred to here as *Norm Based-FFD* ($\texttt{FFD}_{\texttt{NB}}$) and *Dot Product-Based FFD* ($\texttt{FFD}_{\texttt{DP}}$), both of which are implemented in Microsoft's Virtual Machine Manager [25,26]. A method to generate a data set for MDVBPP covering a variety of parameters that can potentially affect the difficulty of the problem is also proposed. In addition to this, a new method for the estimation of lower bound (LB) on the required number of bins is also proposed for the problem instances where the optimal solution contains a few items per bin. Simulation results demonstrating the effectiveness of the proposed approach are also presented.

The remainder of this paper is organized as follows: In Sect. 2 the problem is formally defined. Section 3 explains the proposed approach and other deterministic heuristics. In Sects. 4 and 5 the data set generation and lower bound methods, respectively, are briefly explained. Section 6 provides a comparison of the results with other heuristics. Finally, the conclusions of the study are provided in Sect. 7.

2 Problem formulation

In this section, the definition, notations, and the mathematical formulation of MDVBPP are presented.

2.1 Definition and notations

Suppose that there are n items to be assigned. Each item I_i, $i \in \{1, 2, 3, \ldots, n\}$ is defined as an R-dimensional requirement vector, $I_i = [I_i^1, I_i^2, I_i^r, \ldots, I_i^R]$ where I_i^r, represents the size of rth resource requirement. These items are to be assigned to a minimum number of possible, say m bins. Let C^r be the maximum capacity (or threshold value) in the rth dimension, associated with each bin B_j, $j \in \{1, 2, 3, \ldots, m\}$. The assignment solution is represented by an $m \times n$ matrix A, where

$$A_{ji} = \begin{cases} 1 & \text{if } \ I_i \text{ is assigned to } B_j \\ 0 & \text{otherwise} \end{cases}$$

Let $f(B_j)$ be a function such that $f(B_j) = 1$, if bin B_j is loaded with at least one item, and $f(B_j) = 0$ otherwise.

2.2 Optimization formulation

The problem of assigning all items to the least number of bins, and subject to the constraints, can be formulated as follows:

$$\text{minimize} \sum_{j=1}^{m} f(B_j) \tag{1}$$

$$\text{subject to} \sum_{i=1}^{n} A_{ji}.I_i^r \leq C^r, \quad \forall r \in \{1, 2, 3, \ldots, R\}, \text{ and } \forall j \in \{1, 2, 3, \ldots, m\} \tag{2}$$

```
Procedure
While there are items remaining to be placed Do
  Open a new bin.
   While some item fits in this bin Do
      Place "largest" remaining item that fits in the bin.
   EndWhile
EndWhile
```

Fig. 2 Bin-centric First Fit Decreasing (FFD) procedure [17]

$$\sum_{j=1}^{m} A_{ji} = 1 \quad \forall i \in \{1, 2, 3, \ldots, n\} \tag{3}$$

Constraint (2) imposes a capacity limit on each bin in each dimension, while constraint (3) ensures that each item will be assigned to only one bin.

3 Algorithms for MDVBPP

In this section, two FFD-based heuristics proposed by Panigrahy et al. [26] and the approach proposed in this study for MDVBPP are briefly explained.

3.1 Dot product-based FFD ($\texttt{FFD}_{\texttt{DP}}$)

This heuristic follows the bin-centric FFD approach, procedure of which is given in Fig. 2. Here the size of each item is determined by the weighted dot product between the vector of remaining capacities of the current open bin and the vector of demands for the item. The weighted dot product is calculated using expression 4 below:

$$\sum_{r} a_r . I_i^r . RemCap(B_j)_r \tag{4}$$

where $RemCap(B_j)_r$ is the remaining capacity of bin B_j in rth dimension, and a_r is the weight of rth dimension that is calculated in the following manner:

$$a_r = e^{\left(0.01 * \frac{1}{n} . \sum_{i=1}^{n} I_i^r\right)} \tag{5}$$

The item I_i that maximizes the dot product without violating the capacity constraint is considered to be placed first. For more details of this algorithm, readers are referred to the paper presented by Panigrahy et al. [17].

3.2 Norm-based FFD ($\texttt{FFD}_{\texttt{NB}}$)

This is another bin-centric heuristic that looks at the difference between the vectors I_i and the residual capacity $RemCap(B_j)$ under a certain norm, instead of the dot product. For example, for the l_2 norm, from all unassigned items, it places the item I_i that minimizes the quantity $\sum_r a_r.(I_i^r - RemCap(B_j)_r)^2$ provided the assignment does not violate the capacity constraints. The weights a_r are again chosen as in Equation (5).

3.3 Proposed approach

In this subsection, the SimE-based MDVBPP heuristic is described. A brief description of the basic SimE heuristic is provided first.

3.3.1 Simulated evolution

Simulated evolution is a general iterative heuristic proposed by Kling and Banerjee [27]. This scheme combines iterative improvement and constructive perturbation and saves itself from getting trapped in local minima by following a stochastic approach. In SimE, the search space is traversed by making intelligent moves, unlike in other nondeterministic algorithms such as SA, where random moves are made. The core of the algorithm is the goodness estimator. SimE assigns each moveable element a goodness value. The goodness value indicates how well a certain movable element is currently assigned. The higher the goodness value, the lower is the probability of the element being selected for reallocation.

The structure of the SimE algorithm is shown in Fig. 3. SimE assumes that there exists a solution Φ of a set I containing n movable elements (items). The algorithm starts from an initial assignment Φ_i, and then, following an evolution-based approach seeks to reach better assignments from one generation to the next by perturbing some ill-assigned elements (items) while retaining the near-optimal ones. The algorithm has one main loop consisting of three basic steps, *evaluation*, *selection*, and *allocation*. The three steps are executed in sequence until the average goodness of the solution reaches a maximum value, or no noticeable improvement in the solution quality is observed after a given number of iterations [28].

Goodness evaluation

The Evaluation step consists of evaluating the goodness (fitness) g_i of each item I_i assigned to bin B_j in current solution Φ'. The effective goodness measures can be thought of based on the domain knowledge of the optimization problem [29]. The goodness measure must be a single number expressible in the range [0,1]. For MDVBPP, the goodness measure is defined as follows:

$$g_i = \frac{\sum_{r=1}^{R} I_i^r}{\sum_{r=1}^{R} B_j^r}, \quad \forall r \in \{1, 2, 3, \ldots, R\} \tag{6}$$

```
ALGORITHM
Simulated_Evolution($I$, Stopping − criteria);
/* $\mathbf{\Phi_i}$: Initial Solution; */
/* $\mathbf{\Phi_p}$: Partial Solution; */
/* $\mathbf{\Phi'}$: New Solution; */
/* I: Set of all items, where $|I| = n$; */
/* $\mathbf{I_s}$: Set of items, selected for reallocation; */
/* $\mathbf{B_o}$: Open bins in $\Phi_p$; */
/* B: Selection bias; */
/* maxSelection: Upper limit of the selection set size; */
INITIALIZATION;
    $\Phi_i = initial\_placement(V)$;
    $\Phi' = \Phi_i$;
Repeat
EVALUATION:
    ForEach  $I_i \in I$ Do
        $g_i = Evaluate(I_i)$; /* Evaluate using goodnes estimator (Equation 6) */
    EndForEach;
SELECTION:
    $\Phi_p = \Phi'$;
    $counter = \mathbf{0}$;
    ForEach  $I_i \in I$ Do
        If $(Random() \leq (1 - g_i)) \wedge (counter \leq maxSelection)$ Then
            /* Random() is a function that randomly returns a number in between 0 and 1 */
            $I_s = I_s \cup \{I_i\}$;
            $\Phi_p = \Phi_p - \{I_i\}$;
            $counter = counter + 1$;
        EndIf;
    EndForEach;

ALLOCATION:
Sort the items in set $I_s$ based on their sizes (Equation (8));
Sort the open bins $B_o$ based on their occuppied sizes (Equation (9));
    ForEach  $I_i \in I_s$ Do
        Allocate( $I_i$ , $\Phi_p$); /* Allocate $I_i$ in $\Phi_p$, using First Fit Strategy */
    EndForEach;
        $\Phi' = \Phi_p$;
Until Stopping-criterion is satisfied;
Return (BestSolution);
End Simulated_Evolution.
```

Fig. 3 Simulated evolution algorithm for MDVBPP

Another appropriate goodness measure can also be defined as:

$$g_i = \frac{1}{R} \cdot \sum_{r=1}^{R} \frac{I_i^r}{B_j^r}, \quad \forall r \in \{1, 2, 3, \ldots, R\} \tag{7}$$

where I_i^r is size of item I_i in dimension r, and B_j^r is the available space of partially used bin B_j in dimension r after removing item I_i from bin B_j in the current solution Φ'. Equations (6 and 7) assume a minimization of resource wastage in bin B_j (maximization of goodness). The goodness of an item I_i will be 1 if it is assigned to such a partially used bin B_j that $I_i^r = B_j^r$, $\forall\, r \in \{1, 2, 3, \ldots, R\}$. This means that the current assignment of item I_i exactly packs the bin B_j and hence optimally utilizes the

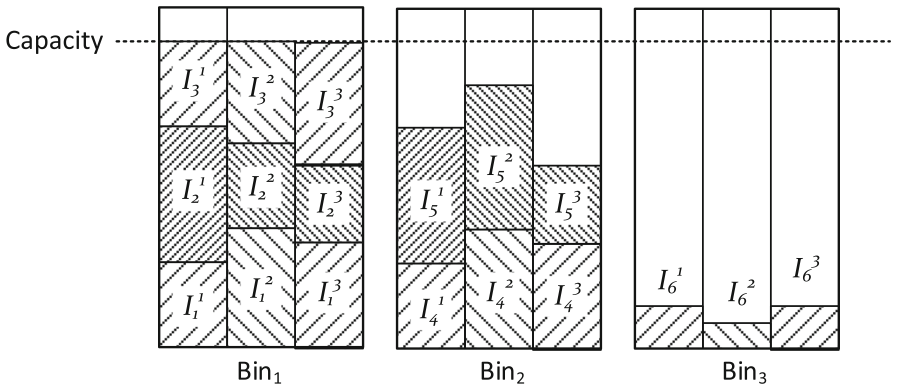


Fig. 4 Placement of 6 items in 3 bins

bin B_j. For example, the goodness values of items I_1, I_2 and I_3 (Fig. 4) are 1 as their combined placements optimally utilize bin B_j in each dimension. On the other hand, the goodness g_i will be near 0, when an item I_i, with a very small size, is placed in an empty bin B_j i.e., $I_i^r << B_j^r, \; \forall \, r \in \{1, 2, 3, \ldots, D\}$. Such an assignment will result in a maximum resource wastage. Item I_6, in Fig. 4, has approximately zero goodness value. Note that this goodness estimation is strongly related to the target objective of the given problem. The quality of a solution can also be estimated by summing up the goodness of all of its constituent elements (items).

Selection

In this step, the algorithm probabilistically selects elements for reallocation. Elements with low *goodness* values have higher probabilities of getting selected. Selection step partitions Φ' into two disjoint sets; a selection set I_s and a partial solution Φ_p containing the remaining elements of the solution Φ'. Each element in the solution is considered separately from all other elements. The decision whether to assign an element I_i to the set I_s is based solely on its goodness g_i. The selection operator has a nondeterministic nature, i.e., an individual with a high goodness (close to one) still has a non-zero probability of being assigned to the selection set I_s. This element of nondeterminism gives SimE the capability of escaping local minima. Each time an item I_i is considered for selection, a random number is generated. The inequality $Random() \leq (1 - \; g_i)$ is used for this purpose (referred to Fig. 3). Large selection sets may lead to a better solution, but will require a higher run time. On the other hand, small selection sets will speed up the algorithm, but with the risk of an early convergence to a sub-optimal solution (local minima) [28]. A parameter $maxSelection$ provides control over the selection process and restricts the maximum size of the selection set. In this work, a value of 40% of the total number of items was adopted. This keeps the time requirements of the SimE algorithm under control, especially during the allocation step, which is the most time-consuming step of the algorithm.

Allocation

Allocation is the SimE operator that has the most impact on the quality of the solution. Allocation takes as input the set I_s and the partial solution Φ_p and generates a completely new solution Φ' with the elements of set I_s mutated according to the allocation strategy. The goal of *Allocation* is to favor improvements over the previous generation, without being too greedy [28]. As the *goodness* of each individual element is also tightly coupled with the target objective, superior alterations are supposed to gradually improve the individual goodness as well. Hence, *Allocation* allows the search to progressively converge towards a configuration where each individual is optimally located.

The choice of a suitable allocation function is problem specific. Similar to the design of the goodness function, the choice of the allocation strategy also requires ingenuity on the part of the designer. In this work, a variant of the FFD heuristic as the allocation strategy was adopted. The items selected during the *selection* step are sorted in decreasing order of their sizes (R_{I_i}) computed using Eq. (8).

$$R_{I_i} = \sum_{r=1}^{R} (I_i^r)^2 \tag{8}$$

The bins in partial solution Φ_p are also sorted in the decreasing order of the linear sum of their occupied space in each dimension, (O_{B_j}) computed using Eq. (9):

$$O_{B_j} = \sum_{r=1}^{R} (1 - {B_j}^r) \tag{9}$$

Subsequently, First Fit algorithm is applied to generate the new solution Φ'. Initial placement Φ_i is also obtained by this same *allocation* strategy but with the difference that all the items are treated as *selected*.

3.4 Complexity analysis

The proposed SimE-based algorithm consists of four steps in a loop as illustrated in Fig. 3. The evaluation step computes *goodness* value of all n items using Eq. (6). This takes $O(n)$ time. The selection step probabilistically selects ill-assigned items and this also takes $O(n)$ time. In the sorting step prior to allocation, both the lists of selected items and open bins are sorted and this takes $O(n \log n)$ time. In the allocation step, First Fit (FF) algorithm sequentially checks if all selected items can be packed into one of the m currently open bins. FF then packs each selected item into a bin first found to be able to accommodate it. If an item cannot be packed into any current active bin, the $(m + 1)$th bin is opened to accommodate it. The complexity of this step is $O(n^2)$. The overall complexity of our algorithm is $O(k \cdot n^2)$, where k is the number of iterations. Experiments indicate that k remains fairly constant as n increases, e.g., k varies in the range of 65–75 when n is increased from 200 to 1000.

4 Data generation for MDVBPP

In practice, engineers invariably are more interested in typical instances of optimization rather than looking for the hardest possible instances. For this reason, suitably parameterized random ensembles of instances of problems are introduced. In some regions of the ensemble space, instances are typically easy to solve, while in other regions instances are found to be typically hard. This change in behavior resembles the phase transitions observed in physical systems [30]. The properties that determine the phase transition in our case are: (a) the correlation between different dimensions of item vectors; and (b) size of the item vectors. If there is a positive correlation in different dimensions of all the items then the problem reduces to the one-dimensional case that is relatively easier to solve than the multi-dimensional one [31]. However, for negatively correlated instances, FFD-based heuristics do not perform particularly well [17]. Instances with very small or very big item sizes are also easier to be solved to a near optimum value. The case where most of the items are of small sizes, the optimal solution contains a few number of bins and many different combinations result in the same number of bins. This case resembles the scenario where one is filling the bins with sand. On the other hand, items with big sizes will lead to solutions where one item is assigned to a bin. This leads to finding of optimal solution trivial.

In this section, the details of the proposed data set generation method for MDVBPP are discussed. The method introduces a variety of different parameters to cover the wide range of possible workloads.

4.1 Parameters

Each instance of MDVBPP can be characterized by a tuple $(C,\ n,\ R,\ Corr,\ v_1, v_2)$, where C represents the bin capacity in each dimension, n the number of items to be packed, R the number of dimensions of each item, $Corr$ is the correlation among the rth and $(r-1)$th dimensions, and, v_1 and v_2 define the interval $[v_1.C,\ v_2.C]$ for the range of the item size in each dimension.

4.2 Data set generation

Each tuple $(C,\ n,\ R,\ Corr,\ v_1, v_2)$ describes a specific class of instances of the MDVBPP. For fixed problem parameters $C,\ n,\ R,\ Corr,\ v_1$, and v_2, any test problem can be interpreted as the realization of an R-dimensional n random variable vectors I_i, i.e.,

$$I_i = [I_i^1, I_i^2, I_i^3, \ldots, I_i^R], \quad \forall i \in \{1, 2, 3, \ldots, n\} \tag{10}$$

These values are generated using the procedure shown in Fig. 5. In Fig. 5, $halfDiff$ is half of the range of interval $[v_1 \cdot C,\ v_2 \cdot C]$, and rand(1) is a function that returns a uniformly distributed random real number in the range [0, 1). The size of each item in the first dimension, i.e., I_i^1 is set to a random number uniformly distributed in the interval $[v_1 \cdot C,\ v_2 \cdot C]$ (refer to lines 9–11). Similarly, sizes in other dimensions, i.e.,

```
PROCEDURE
$DataSet\_Generator(C,\ R,\ n,\ Corr,\ v_1,\ v_2)$;
/* Corr: Correlation; */
/* n: number of items; */
/* R: number of dimensions; */
/* C: Total capacity in each dimension; */
/* I: Set of all items, where $|I| = n$; */
/* $v_1$: lower limit of item weights (as a fraction of total capacity); */
/* $v_2$: upper limit of item weights (as a fraction of total capacity); */
$INITIALIZATION$;
 1. If  $(Corr < 0)$ Then
 2.     $P_c = 0.0$;
 3. ElseIf $(Corr = 0)$ Then
 4.     $P_c = 0.5$;
 5. Else $(Corr > 0)$ Then
 6.     $P_c = 1.0$;
 7. EndIf;
 8. $halfDiff = C * [\frac{v_1+v_2}{2} - v_1]$;
 9. ForEach  $I_i \in I$ Do
10.     $I_i^1 = ((v_1 + (v_2 - v_1) * rand(1)) * C$; /* $I_i^1 \in U(v_1.C,\ v_2.C)$ */
11. EndForEach;
12. For $r = 2$ To $R$ Do
13.     $meanValue = mean(I_i^{r-1})$;
14.     ForEach  $I_i \in I$ Do
15.         $I_i^r = ((v_1 + (\frac{v_2-v_1}{2}) * rand(1)) * C$; /* $I_i^r \in U(v_1.C,\ \frac{v_2-v_1}{2}.C)$ */
16.         $rnd = rand(1)$;
17.         If $(rnd < Pc \wedge I_i^r \geq meanValue) \vee (rnd \geq Pc \wedge I_i^r < meanValue)$ Then
18.             $I_i^r = I_i^r + halfDiff$;
19.         EndIf;
20.         If  $I_i^r > C$ Then
21.             $I_i^r = C$;
22.         EndIf;
23.     EndForEach;
24. EndFor;
25. Return $(I)$;
26. End $DataSet\_Generator$.
```

Fig. 5 Procedure to generate data set for MDVBPP

I_i^r, $\forall r \in \{2, 3, \ldots, R\}$ are first generated through another uniform random variable in the range half of the given interval, i.e., $[v_1.C,\ \frac{v_1+v_2}{2} \cdot C]$ and then a probability P_c is used to decide whether I_i^r is to be increased by a value $halfDiff$ or not (refer to lines 15–8). This step is to introduce the required correlation between rth and $(r-1)$th dimensions. The probability P_c is selected according to the required correlation value (refer to lines 1–7).

4.3 Example

Let a class of problem instances characterized by $(C,\ n,\ R,\ Corr,\ v_1,\ v_2) = (1000,\ 20,\ 4,\ Corr,\ 0.001,\ 0.9)$, where $Corr$ can be negative, zero or positive. For negative correlation, a sample data set is given in Table 1.

Table 1 Sample data set with negative correlation between adjacent dimensions of items

Item no.	Item size	
1	[676, 128, 722, 279]	
2	[178, 538, 307, 539]	
3	[868, 189, 861, 81]	
4	[550, 204, 493, 121]	
5	[89, 797, 389, 747]	
6	[208, 836, 68, 500]	
7	[719, 242, 703, 466]	
8	[682, 137, 846, 183]	
9	[436, 477, 246, 519]	
10	[198, 572, 98, 551]	
11	[157, 730, 444, 821]	
12	[858, 173, 608, 246]	
13	[754, 328, 578, 243]	
14	[347, 878, 329, 645]	
15	[548, 73, 520, 461]	
16	[899, 188, 707, 99]	
17	[302, 845, 111, 857]	
18	[367, 636, 276, 516]	
19	[884, 366, 818, 239]	
20	[293, 893, 199, 689]	
Correlation	Dimension 1 and 2	−0.8041
	Dimension 2 and 3	−0.7887
	Dimension 3 and 4	−0.7343

5 Improved lower bound

To assess the performance of the proposed heuristic, the consolidation ratio(q/LB), which is defined as the ratio of the obtained cost, i.e., the number of bins q to the estimated LB, is compared across heuristics. Thus, to obtain a better estimate of the performance of the proposed heuristic, a new procedure for a tighter LB is proposed.

5.1 Procedure for the calculation of LB

In the case of 1D-BPP, a continuous LB is calculated with an assumption that items can be allocated in fractional quantities of their sizes. For MDVBPP, one natural way of estimating the LB is to calculate the maximum of the continuous LB values obtained by considering each of its R dimensions individually at a time. This can be done using Eq. 11. A similar LB was proposed by Spieksma [13] for $R = 2$.

$$\mathrm{LB}_c = \max\left(\left\lceil \sum_{i=1}^{n} \frac{I_i^1}{C^1} \right\rceil, \left\lceil \sum_{i=1}^{n} \frac{I_i^2}{C^2} \right\rceil, \ldots, \left\lceil \sum_{i=1}^{n} \frac{I_i^R}{C^R} \right\rceil\right) \tag{11}$$

```
Procedure
$LowerBound(C,\ I)$;
/* n: Number of items; */
/* I: Set of all items, where $|I| = n$; */
/* C: Total capacity in each dimension; */
$STEP\ 1:$
ForEach  $I_i \in I$ Do
    $/*$ Construct a set $S_i$ containing all the items that cannot be combined
     with $I_i$ due to capacity constraints in any dimension. $*/$
    $S_i = \{I \setminus I_i \mid \exists\ r \in \{1, 2, 3, ..., R\} : I_i^r + I_j^r > C\}$;
EndForEach;
$STEP\ 2:$
$/*$ Construct a set $S_{alone}$ containing all the items that cannot be combined
 with any other item. $*/$
$S_{alone} = \{I_i\ :\ |S_i| = n - 1\}$;
$STEP\ 3:$
$/*$ Subtract the items in set $S_{alone}$ from all other sets, i.e., set $I$ and each
 set $S_i$ to reduce the problem size for remaining steps. $*/$
$I = I \setminus S_{alone}$;
ForEach  $I_i \in I$ Do
    $S_i = S_i \setminus S_{alone}$;
EndForEach;
$STEP\ 4:$
ForEach  $I_i \in I$ Do
    $/*$ Construct a set $t_i$ such that none of its member item can share the same bin
     with any other item belonging to the same set $t_i$ due to capacity constraint
     violation in any dimension. $*/$
    $t_i = \{I_i\}$;
    $S_{temp} = S_i$;
    While $S_{temp} \neq \varnothing$
        pick item $I_j \in S_{temp} \mid j \leq k, \forall I_k \in S_{temp}$ and update $t_i$ and $S_{temp}$ as following
        $t_i = t_i \cup \{I_j\}$;
        $S_{temp} = S_{temp} \cap \{S_j\}$;
    EndWhile;
EndForEach;
$STEP\ 5:$
$S_t = largest\ t_i$;
$LB_2 = |S_{alone}|\ +\ |S_t|$;
Return $(LB_2)$;
End $LowerBound$.
```

Fig. 6 Procedure for determining the lower bound LB_2

This lower bound LB_c is trivially computed in $O(n)$ time and seems to be appropriate when items are relatively small in size w.r.t. the capacity of bins [13] (e.g., in our case for classes where $v_1 \in [0.001, 0.25]$ and $v_2 \in [0.1, 0.4]$). This is due to optimal solutions of such instances tend to have only a few bins and little empty spaces in them that lead to less amount of error in the LB estimation. On the other hand, this lower bound (LB_c) inherently performs poorly for the instances where optimal solution contains many empty spaces. For such instances, another lower bound procedure is proposed, called LB_2. The idea is to find the maximum number of items for which it is known that no two of these items can be assigned to the same bin. Evidently, this

Table 2 Sample data set

I_i	Item size	S_i	$\|S_i\|$
I_1	[321, 666, 878, 220]	$S_1 = \{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$	11
I_2	[324, 212, 525, 667]	$S_2 = \{1, 3, 4, 5, 6, 7, 8, 9, 11, 12\}$	10
I_3	[315, 232, 566, 358]	$S_3 = \{1, 2, 4, 5, 7, 9, 10, 11, 12\}$	9
I_4	[87, 67, 680, 258]	$S_4 = \{1, 2, 3, 5, 7, 9, 10, 11, 12\}$	9
I_5	[636, 233, 619, 759]	$S_5 = \{1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12\}$	11
I_6	[560, 482, 319, 568]	$S_6 = \{1, 2, 5, 7, 8, 9, 11, 12\}$	8
I_7	[7, 428, 847, 774]	$S_7 = \{1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12\}$	11
I_8	[562, 636, 43, 398]	$S_8 = \{1, 2, 5, 6, 7, 9, 11, 12\}$	8
I_9	[204, 781, 710, 739]	$S_9 = \{1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12\}$	11
I_{10}	[409, 294, 469, 197]	$S_{10} = \{1, 3, 4, 5, 7, 9, 12\}$	7
I_{11}	[350, 311, 221, 775]	$S_{11} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 12\}$	10
I_{12}	[464, 210, 598, 794]	$S_{12} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$	11

number serves as a lower bound for the optimal solution. The procedure to calculate this number is shown in Fig. 6. The core of the procedure is to divide a given set of items I into three subsets S_{alone}, S_t, and S_r. For illustration of the procedure, consider the data set given in Table 2.

Step1 generates sets S_i corresponding to each item I_i, where each set S_i contains all those items that cannot be combined with item I_i (Note that in this example, maximum capacity for each bin in each dimension is $C = 1000$). These sets are shown in the third column of Table 2. The complexity of this step is $O(n^2)$ as the feasibility of packing of each item with every other item is to be tested one by one. In *step2*, set S_{alone} is constructed by adding all those items that cannot be packed with any other item in set I. Such items are identified by the cardinality of their corresponding set S_i. This step is executed in $O(n)$ time. In our example S_{alone} is as follows:

$$S_{\text{alone}} = \{1,\ 5,\ 7,\ 9,\ 12\ \}$$

As these five items will not share a bin with any other item in set I, these items are subtracted from set I to focus on the remaining items only. This is done in *step 3* of the procedure (Fig. 6). This subtraction step significantly reduces the execution time of the rest of the steps. The reduced problem set for further calculation is shown in Table 3.

Then in *step 4*, all possible sets t_i corresponding to each item I_i in reduced problem set are calculated in such a way that items in each set t_i cannot share the same bin with other items in the same set. This step constructs each set t_i in multiple iterations. Iterative steps to calculate set t_2 are shown in Table 4. Clearly no two items in set $t_2 = \{2,\ 3,\ 4,\ 11\}$ can be combined with each other in one bin. Similarly remaining sets $t_3 = \{3, 2, 4, 11\}$, $t_4 = \{4, 2, 3, 11\}$, $t_6 = \{6, 2, 8, 11\}$, $t_8 = \{8, 2, 6, 11\}$, $t_{10} = \{10, 3, 4\}$, $t_{11} = \{11, 2, 3, 4\}$ can also be calculated. In the last step, largest of these sets t_i is declared as set S_t. In this example, $S_t = \{2,\ 3,\ 4,\ 11\}$. Rest of the items are

Table 3 Reduced problem set at an intermediate stage of lower bound procedure

I_i	Item size	S_i
I_2	[324, 212, 525, 667]	$S_2 = \{3, 4, 6, 8, 11\}$
I_3	[315, 232, 566, 358]	$S_3 = \{2, 4, 10, 11\}$
I_4	[87, 67, 680, 258]	$S_4 = \{2, 3, 10, 11\}$
I_6	[560, 482, 319, 568]	$S_6 = \{2, 8, 11\}$
I_8	[562, 636, 43, 398]	$S_8 = \{2, 6, 11\}$
I_{10}	[409, 294, 469, 197]	$S_{10} = \{3, 4\}$
I_{11}	[350, 311, 221, 775]	$S_{11} = \{2, 3, 4, 6, 8\}$

Table 4 Iterative steps to calculate set t_2

Iter.	While	t_2	S_{temp}
–	–	$t_2 = \{2\}$	$S_{temp} = S_2 = \{3, 4, 6, 8, 11\}$
1	True	$t_2 = \{2, 3\}$	$S_{temp} = S_{temp} \cap S_3 = \{3, 4, 6, 8, 11\} \cap \{2, 4, 10, 11\} = \{4,11\}$
2	True	$t_2 = \{2, 3, 4\}$	$S_{temp} = S_{temp} \cap S_4 = \{4,11\} \cap \{2, 3, 10, 11\} = \{11\}$
3	True	$t_2 = \{2, 3, 4, 11\}$	$S_{temp} = S_{temp} \cap S_{11} = \{11\} \cap \{2, 3, 4, 6, 8\} = \{\}$
4	False	–	–

placed in set S_r, these items may or may not share a bin with the items in set S_t. In this example, $S_r = I \backslash \{S_{\text{alone}} \cup S_t\} = \{6,\ 8,\ 10\}$. At the end, the procedure returns the sum of the cardinality of sets S_{alone} and S_t. $\text{LB}_2 = |S_{\text{alone}}| + |S_t| = 9$. Note that in this example $\text{LB}_2 > \text{LB}_c$, where

$$\text{LB}_c = \max\left(\lceil 4239/1000 \rceil,\ \lceil 4552/1000 \rceil,\ \lceil 6475/1000 \rceil,\ \lceil 6507/1000 \rceil\right) = 7$$

The complexity of the lower bound LB_2 procedure is $O(n^2)$. This LB is suited for problem instances where several items have high values for one or more number of dimensions. Evidently, the overall LB is equal to the maximum of the two.

$$\text{LB} = \max\{\text{LB}_c, \text{LB}_2\} \tag{12}$$

6 Performance evaluation

In this section, a performance evaluation of the proposed approach with respect to solution quality and runtime is provided. First, the proposed approach is compared with the Norm Based-FFD ($\texttt{FFD}_{\texttt{NB}}$) and Dot Product-Based FFD ($\texttt{FFD}_{\texttt{DP}}$) that are implemented in Microsoft's Virtual Machine Manager [25,26], and a well-known iterative heuristic, SA [28]. Then the solution quality and performance of SimE heuristic are discussed.

6.1 Simulation setup

The programs for the proposed SimE algorithm, SA, $\texttt{FFD}_{\texttt{NB}}$ and $\texttt{FFD}_{\texttt{DP}}$ heuristics were coded in MATLAB and run on an Intel Xeon E5405 with 2.00 GHz CPU (2 processors) and 4 GB RAM. The quality of the solution obtained by SimE improves when the number of iterations is increased. The improvement is quite steep in the early iterations. It becomes less steep in later iterations until it becomes almost insignificant. The number of required iterations can be easily tuned after a few initial experimental runs [28]. In this work, SimE algorithm was set to stop exploring the search space if no improvement was observed in the last 75 iterations. And the maximum size ($maxSelection$) of the selection set was restricted to 40% of the total items (reasons are discussed in Sect. 3.3.1). Two goodness measures were proposed in Sect. 3.3.1. The one in Eq. 6 exhibited better performance and hence was used in all experimentations.

6.1.1 Work load

Problem instances were generated with following different parameter values, using the procedure discussed in Sect. 4, Fig. 5.

Bin capacity in each dimension: $C = 1000$
Problem size: $n = 250, 500$
Resource dimensions: R = 4
Correlation: Corr = Negative, Zero, Positive
Lower limit of item size: $v_1 = 0.001, 0.05, 0.15, 0.25, 0.35$
Upper limit of item size: $v_2 = 0.1, 0.2, 0.3, \ldots, 1.0$

Due to the nondeterministic behavior, the average of results obtained from 20 independent runs are reported.

6.2 Results and discussion

To evaluate the efficiency of the proposed SimE algorithm, its performance was compared to that of $\texttt{FFD}_{\texttt{NB}}$, $\texttt{FFD}_{\texttt{DP}}$, and SA. The comparison metrics are the time to find the solution and consolidation ratio (q/LB). A value q/LB closer to 1.0 represents a higher efficiency. Table 5 lists the time (in seconds) to find the solution and average value q/LB obtained by these algorithms for different correlation and lower and upper limits of item size values.

From Table 5 the following can be noted:

- For all algorithms applied, consolidation ratio increases with a change of correlation from positive to negative. However, a deviation from this trend is observed in the test cases where the average value of the item size is high. The reason is, for the negatively correlated instances of these test cases, most packing solutions have only one or two items per bin, and hence obtaining the optimum solution becomes trivial.
- Timing performance of both deterministic heuristics is better than SimE and SA as expected. SimE takes far less time than the other non-deterministic heuristic, SA.

Table 5 Performance comparison of $\mathtt{FFD}_{\mathtt{NB}}$, $\mathtt{FFD}_{\mathtt{DP}}$, SA and SimE

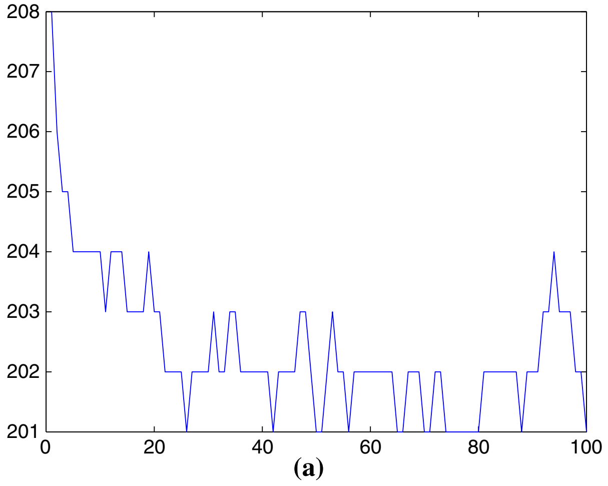
v_1	v_2	Avg.	Algo.	$n = 250$						$n = 500$					
				Neg. corr.		No corr.		Pos. corr.		Neg. corr.		No corr.		Pos. corr.	
				q/LB	Time	q/LB	Time	q/LB	Time	q/LB	Time	q/LB	Time	q/LB	Time
0.05	0.2	0.125	$\mathtt{FFD}_{\mathtt{NB}}$	1.127	0.293	1.11	0.293	1.065	0.285	1.112	1.11	1.111	1.105	1.065	1.084
			$\mathtt{FFD}_{\mathtt{DP}}$	1.12	0.314	1.104	0.31	1.067	0.303	1.114	1.182	1.118	1.179	1.065	1.151
			SA	1.545	21.978	1.527	21.732	1.521	21.676	1.664	143.2	1.678	141.601	1.636	144.019
			SimE	1.056	2.527	1.045	2.59	1.02	1.474	1.049	12.119	1.051	12.43	1.02	9.873
	0.5	0.275	$\mathtt{FFD}_{\mathtt{NB}}$	1.204	0.48	1.205	0.485	1.143	0.444	1.206	1.828	1.194	1.808	1.142	1.699
			$\mathtt{FFD}_{\mathtt{DP}}$	1.203	0.524	1.205	0.53	1.142	0.489	1.206	2.008	1.195	2	1.14	1.855
			SA	1.299	24.572	1.336	24.454	1.257	23.786	1.34	155.349	1.371	154.12	1.291	154.093
			SimE	1.111	5.089	1.072	4.816	1.042	3.998	1.111	29.351	1.066	24.287	1.039	20.971
	0.6	0.325	$\mathtt{FFD}_{\mathtt{NB}}$	1.336	0.573	1.261	0.556	1.194	0.491	1.322	2.191	1.241	2.073	1.177	1.883
			$\mathtt{FFD}_{\mathtt{DP}}$	1.331	0.635	1.261	0.611	1.197	0.54	1.322	2.426	1.248	2.301	1.179	2.048
			SA	1.393	26.836	1.333	26.309	1.245	25.044	1.424	165.748	1.368	162.359	1.277	160.187
			SimE	1.172	7.449	1.093	6.158	1.052	4.367	1.158	39.936	1.081	34.45	1.047	23.821
	0.9	0.475	$\mathtt{FFD}_{\mathtt{NB}}$	1.2	0.777	1.182	0.785	1.229	0.676	1.237	2.897	1.196	2.946	1.215	2.567
			$\mathtt{FFD}_{\mathtt{DP}}$	1.202	0.866	1.187	0.868	1.225	0.752	1.233	3.242	1.196	3.299	1.219	2.842
			SA	1.176	40.107	1.164	40.247	1.237	32.771	1.277	226.709	1.23	229.117	1.257	241.198
			SimE	1.059	5.568	1.046	7.018	1.073	5.393	1.063	27.673	1.051	35.054	1.055	24.684
0.15	0.3	0.225	$\mathtt{FFD}_{\mathtt{NB}}$	1.109	0.396	1.112	0.398	1.097	0.39	1.108	1.502	1.113	1.502	1.102	1.499
			$\mathtt{FFD}_{\mathtt{DP}}$	1.112	0.425	1.111	0.425	1.103	0.424	1.11	1.629	1.114	1.633	1.101	1.618
			SA	1.23	22.87	1.24	22.721	1.225	22.572	1.271	146.324	1.287	146.16	1.269	177.612
			SimE	1.101	2.916	1.06	3.112	1.019	2.453	1.097	15.042	1.064	17.736	1.017	10.674

Table 5 continued

v_1	v_2	Avg.	Algo.	$n = 250$						$n = 500$					
				Neg. corr.		No corr.		Pos. corr.		Neg. corr.		No corr.		Pos. corr.	
				q/LB	Time	q/LB	Time	q/LB	Time	q/LB	Time	q/LB	Time	q/LB	Time
	0.5	0.325	$\texttt{FFD}_{\texttt{NB}}$	1.314	0.573	1.248	0.477	1.154	0.5	1.301	2.152	1.255	2.084	1.15	1.93
			$\texttt{FFD}_{\texttt{DP}}$	1.32	0.628	1.249	0.503	1.152	0.553	1.308	2.405	1.25	2.312	1.149	2.106
			SA	1.383	26.314	1.312	25.715	1.21	24.879	1.403	162.245	1.351	160.452	1.233	155.373
			SimE	1.148	6.718	1.076	4.581	1.083	5.442	1.121	32.171	1.066	31.506	1.082	28.959
	0.6	0.375	$\texttt{FFD}_{\texttt{NB}}$	1.312	0.541	1.278	0.615	1.269	0.571	1.318	2.394	1.276	2.34	1.251	2.197
			$\texttt{FFD}_{\texttt{DP}}$	1.313	0.571	1.277	0.672	1.252	0.637	1.316	2.629	1.272	2.575	1.246	2.417
			SA	1.31	27.684	1.283	27.571	1.245	26.667	1.323	168.111	1.307	168.263	1.266	163.977
			SimE	1.29	6.665	1.146	7.525	1.07	5.379	1.287	52.965	1.144	40.799	1.062	27.05
	1.0	0.575	$\texttt{FFD}_{\texttt{NB}}$	1.204	0.53	1.202	0.525	1.148	0.501	1.199	2.023	1.193	2.014	1.151	1.938
			$\texttt{FFD}_{\texttt{DP}}$	1.203	0.585	1.195	0.576	1.149	0.551	1.199	2.235	1.195	2.212	1.151	2.112
			SA	1.291	25.614	1.272	25.324	1.2	24.687	1.325	176.725	1.311	190.82	1.218	183.514
			SimE	1.036	3.535	1.033	3.457	1.039	4.32	1.026	16.857	1.024	16.884	1.034	23.898
0.25	0.4	0.325	$\texttt{FFD}_{\texttt{NB}}$	1.311	0.619	1.3	0.623	1.206	0.576	1.318	2.395	1.303	2.391	1.199	2.237
			$\texttt{FFD}_{\texttt{DP}}$	1.311	0.679	1.3	0.688	1.201	0.636	1.318	2.662	1.304	2.639	1.196	2.457
			SA	1.314	27.17	1.308	26.979	1.204	26.191	1.325	185.728	1.322	201.133	1.224	192.838
			SimE	1.311	6.243	1.251	7.66	1.089	4.633	1.318	34.955	1.244	56.622	1.088	26.159
	0.7	0.475	$\texttt{FFD}_{\texttt{NB}}$	1.203	0.71	1.218	0.757	1.235	0.701	1.196	2.657	1.272	2.834	1.221	2.655
			$\texttt{FFD}_{\texttt{DP}}$	1.207	0.792	1.215	0.835	1.236	0.766	1.196	2.952	1.276	3.154	1.223	2.93
			SA	1.18	35.198	1.193	37.529	1.19	32.109	1.223	229.548	1.303	266.469	1.196	228.65
			SimE	1.036	3.49	1.053	5.257	1.02	2.876	1.017	13.83	1.066	23.826	1.001	4.073

Table 5 continued

v_1	v_2	Avg.	Algo.	$n = 250$						$n = 500$					
				Neg. corr.		No corr.		Pos. corr.		Neg. corr.		No corr.		Pos. corr.	
				q/LB	Time	q/LB	Time	q/LB	Time	q/LB	Time	q/LB	Time	q/LB	Time
	0.9	0.575	$\texttt{FFD}_{\texttt{NB}}$	1.001	1.076	1.019	1.008	1.189	0.82	1.002	4.174	1.027	3.92	1.187	3.201
			$\texttt{FFD}_{\texttt{DP}}$	1.001	1.198	1.017	1.131	1.189	0.913	1.002	4.686	1.028	4.391	1.187	3.562
			SA	1.001	95.013	1.012	62.908	1.179	39.942	1.003	527.84	1.028	415.137	1.185	240.335
			SimE	1	1.653	1.004	6.321	1.167	7.686	1	14.198	1.007	65.411	1.167	42.681
0.35	0.5	0.425	$\texttt{FFD}_{\texttt{NB}}$	1.166	0.615	1.164	0.618	1.167	0.618	1.17	2.398	1.166	2.407	1.17	2.414
			$\texttt{FFD}_{\texttt{DP}}$	1.166	0.683	1.164	0.682	1.167	0.688	1.17	2.651	1.166	2.657	1.17	2.653
			SA	1.17	26.953	1.167	23.753	1.173	25.335	1.177	168.27	1.172	168.807	1.178	166.176
			SimE	1.166	4.9	1.164	4.848	1.167	4.833	1.17	25.999	1.166	25.731	1.17	25.878
	0.7	0.525	$\texttt{FFD}_{\texttt{NB}}$	1	1.081	1.007	1.042	1.124	0.839	1.001	4.21	1.01	4.053	1.125	3.292
			$\texttt{FFD}_{\texttt{DP}}$	1	1.203	1.007	1.166	1.125	0.938	1.001	4.727	1.011	4.561	1.125	3.668
			SA	1	0.085	1.004	71.519	1.103	43.325	1.001	205.851	1.011	604.484	1.122	324.597
			SimE	1	0.435	1.001	2.796	1.079	6.602	1.001	23.751	1.002	41.736	1.088	36.552



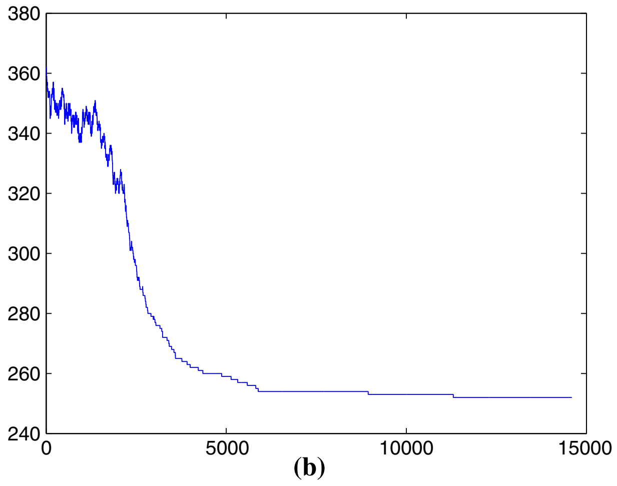


Fig. 7 Change in number of open bins with iterations in **a** SimE and **b** SA

- For all cases considered, the engineered SimE heuristic gives better consolidation efficiency when compared to `FFD`$_{\text{DP}}$, `FFD`$_{\text{NB}}$, and SA. With regards to the reduction in the number of bins, for example, from Table reftable:comp, corresponding to test case ($v1 = 0.05$; $v2 = 0.9$; $n = 500$; Corr = Neg.), SimE gave a consolidation ratio of 1.06, while the other algorithms gave a consolidation ratio higher than 1.23. These ratios translate into 285 and 331 bins respectively. In the context of VM placement problem at cloud-based data centers, this difference of 46 bins means 46 lesser physical machines and hence large savings in energy.

SimE performs better than the two deterministic algorithms `FFD`$_{\text{DP}}$ and `FFD`$_{\text{NB}}$ because these heuristics are single pass and pack the items with the best effort in one go. While the proposed SimE is a multi-pass heuristic that probabilistically picks a

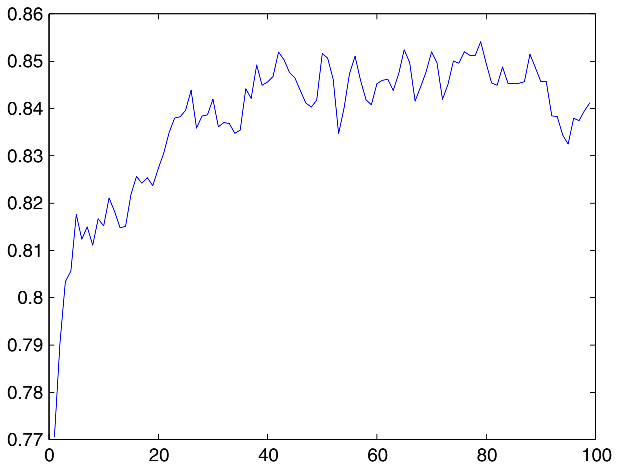


Fig. 8 SimE: Change in the average goodness of items with iterations

small number of items with a low goodness value after each pass and reallocates them after sorting them in a certain order. The precise selection of a small number of items and proper sorting plays a key role in improving the solution quality. Although SimE and SA both are iterative nondeterministic heuristics, SimE is more intelligent, and thus requires fewer iterations to converge towards a desirable solution [28]. Change in the cost of SimE and SA with iteration is illustrated in Fig. 7. SimE quickly finds a good solution through a few initial iterations. The plot of the average goodness of the solution with iterations is shown in Fig. 8. This graph reflects the behavior of SimE. The average goodness increases with iterations. As the algorithm progresses, more and more items are approaching their respective near optimal assignments in the solution is validated. It also shows that the algorithm possesses the hill-climbing phenomena.

7 Conclusion

MVBPP has several applications in different fields. The engineering of SimE search heuristic to find better solutions for this combinatorial NP-hard optimization problem is presented. The solutions in the SimE heuristic evolve based on the current goodness value of the items packed in bins. Goodness measures that enable the SimE heuristic to quickly find the near optimal solution was developed in this study. Its performance was evaluated for a wide range of different problem instances that vary in item sizes, and in the correlation between different dimensions. In terms of the consolidation efficiency, optimization results obtained by SimE are better than those published in the literature. We believe that the newly proposed heuristic should be the algorithm of choice for many applications.

Acknowledgements The authors acknowledge King Fahd University of Petroleum & Minerals (KFUPM) for all the support provided. The work was conducted as part of project No. COE-572132-4.

References

1. Jhawar R, Piuri V, Samarati P (2012) Supporting security requirements for resource management in cloud computing. In: Proceedings of the 2012 IEEE 15th International Conference on Computational Science and Engineering, pp.170–177, December 05–07, 2012. doi:10.1109/ICCSE.2012.32
2. Xu J, Fortes JA (2010) Multi-objective virtual machine placement in virtualized data center environments. In: International Conference on Green Computing and Communications (GreenCom), IEEE/ACM
3. Gao Y, Guan H, Qi Z, Hou Y, Liu L (2013) A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. J Comput Syst Sci 79(8):1230–1242
4. Alharbi F, Tain YC, Tang M, Sarker TK (2016) Profile-based static virtual machine placement for energy-efficient data center. In: High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2016 IEEE 18th International Conference on, IEEE, 2016, pp 1045–1052
5. Garey MR, Graham RL, Johnson DS, Yao AC-C (1976) Resource constrained scheduling as generalized bin packing. J Comb Theory Ser A 21(3):257–298
6. Kou L, Markowsky G (1977) Multidimensional bin packing algorithms. IBM J Res Dev 21(5):443–448. doi:10.1147/rd.215.0443
7. Sait SM, Bala A, El-Maleh AH (2016) Cuckoo search based resource optimization of datacenters. Appl Intell 44(3):489–506
8. Han BT, Diehr G, Cook JS (1994) Multiple-type, two-dimensional bin packing problems: applications and algorithms. Ann Oper Res 50(1):239–261
9. Sarin S, Wilhelm W (1984) Prototype models for two-dimensional layout design of robot systems. IIE Trans 16(3):206–215
10. Cook JS, Han BT (1994) Optimal robot selection and workstation assignment for a CIM system. IEEE Trans Robot Autom 10(2):210–219
11. Chang SY, Hwang H-C, Park S (2005) A two-dimensional vector packing model for the efficient use of coil cassettes. Comput Oper Res 32(8):2051–2058
12. Vercruyssen D, Muller H (1987) Simulation in production, Technical report of the University of Gent, Belgium, p 23
13. Spieksma FC (1994) A branch-and-bound algorithm for the two-dimensional vector packing problem. Comput Oper Res 21(1):19–25
14. Woeginger GJ (1997) There is no asymptotic PTAS for two-dimensional vector packing. Inf Process Lett 64(6):293–297. doi:10.1016/S0020-0190(97)00179-8. http://www.sciencedirect.com/science/article/pii/S0020019097001798
15. Caprara A, Toth P (2001) Lower bounds and algorithms for the 2-dimensional vector packing problem. Disc Appl Math 111(3):231–262
16. Kellerer H, Kotov V (2003) An approximation algorithm with absolute worst-case performance ratio 2 for two-dimensional vector packing. Oper Res Lett 31(1):35–41. doi:10.1016/S0167-6377(02)00173-6. http://www.sciencedirect.com/science/article/pii/S0167637702001736
17. Panigrahy R, Talwar K, Uyeda L, Wieder U (2011) Heuristics for vector bin packing. Technical report, Microsoft Research (2011)
18. Yao A (1980) New algorithms for bin packing. J ACM (JACM) 27(2):207–227
19. De La Vega WF, Lueker GS (1981) Bin packing can be solved within 1+ ε in linear time. Combinatorica 1(4):349–355
20. Chekuri C, Khanna S (2004) On multidimensional packing problems. SIAM J Comput 33(4):837–851
21. Shi L, Furlong J, Wang R (2013) Empirical evaluation of vector bin packing algorithms for energy efficient data centers. In: Symposium on computers and communications (ISCC), IEEE, pp 9–15
22. Stillwell M, Schanzenbach D, Vivien F, Casanova H (2010) Resource allocation algorithms for virtualized service hosting platforms. J Parall Distrib Comput 70(9):962–974
23. An-ping X, Chun-xiang X (2014) Energy efficient multiresource allocation of virtual machine based on PSO in cloud data center. Math Prob Eng 2014:1–8. doi:10.1155/2014/816518
24. Liu X-F, Zhan Z-H, Deng JD, Li JD, Gu T, Zhang J (2016) An Energy Efficient Ant Colony System for Virtual Machine Placement in Cloud Computing. IEEE Trans Evol Comput. doi:10.1109/TEVC.2016.2623803
25. Microsoft systems center virtual machine manager. http://www.microsoft.com/systemcenter/virtualmachinemanager

26. 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
27. Kling R-M, Banerjee P (1987) ESP: a new standard cell placement package using simulated evolution. In: Proceedings of the 24th Design Automation Conference, ACM/IEEE, pp 60–66
28. Sait S M, Youssef H (1999) Iterative computer algorithms with applications in engineering: solving combinatorial optimization problems. IEEE Computer Society Press, Washington, DC
29. Sait S M, Youssef H (1994) VLSI physical design automation: theory and practice. McGraw-Hill, Inc., New York
30. Hartmann A (2005) Phase transitions in combinatorial optimization problems—basics, algorithms and statistical mechanics. Wiley-VCH, Weinheim
31. Ajiro Y, Tanaka A (2007) Improving packing algorithms for server consolidation. In: International CMG Conference, 2007, pp 399–406