Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

Optimizing large scale bin packing problem with hybrid harmony search algorithm

Amol C. Adamuthe[a*] and Tushar R. Nitave[b]

[a]*Department of Computer Science & Information Technology, Rajarambapu Institute of Technology, Rajaramnagar, Dist. Sangli, MS, India*
[b]*Department of Computer Science, Illinois Institute of Technology, Chicago, United States*

C H R O N I C L E

Article history:
Received August 26 2020
Received in Revised Format October 7 2020
Accepted November 23 2020
Available online November, 23 2020

Keywords:
Harmony search algorithm
Bin packing problem
Combinatorial optimization
Constraint satisfaction problem
Heuristics

A B S T R A C T

Bin packing problem (BPP) is a combinatorial optimization problem with a wide range of applications in fields such as financial budgeting, load balancing, project management, supply chain management. Harmony search algorithm (HSA) is widely used for various real-world and engineering problems due to its simplicity and efficient problem solving capability. Literature shows that basic HSA needs improvement in search capability as the performance of the algorithm degrades with increase in the problem complexity. This paper presents HSA with improved exploration and exploitation capability coupled with local iterative search based on random swap operator for solving BPP. The study uses the despotism based approach presented by Yadav et al. (2012) [Yadav P., Kumar R., Panda S.K., Chang, C. S. (2012). An intelligent tuned harmony search algorithm for optimisation. *Information Sciences, 196*, 47-72.] to divide Harmony memory (HM) into two categories which helps to maintain balance between exploration and exploitation. Secondly, local iterative search explores multiple neighborhoods by exponentially swapping components of solution vectors. A problem specific HM representation, HM re-initialization strategy and two adaptive PAR strategies are tested. The performance of proposed HSA is evaluated on 180 benchmark instances which consists of 100, 200 and 500 objects. Evaluation metrics such as best, mean, success rate, acceleration rate and improvement measures are used to compare HSA variations. The performance of the HSA with iterative local search outperforms other two variations of HSA.

1. Introduction

The Bin Packing Problem (BPP) is an NP-hard problem and it is considered as one of the most important optimization problems in computer science. The objective of BPP is to allocate every item to a bin in such a way that the combined weight of all the items in a bin is not more than bin's capacity and the minimum number of bins are used. There are numerous applications of bin packing in the real world. Industries that are involved in cutting wood, glass and paper (Hopper & Turton, 1999). Perboli et al. (2014) showed the application of bin packing problem in packing and routing. Song et al. (2013) used bin packing problem to solve resource provisioning in the cloud. Coffman et al. (1978) solved multiprocessor scheduling using bin packing. De Cauwer et al. (2016) applied bin packing for managing workloads in data centers. Leinberger et al. (1999) provided packing algorithms for multi-resource allocation and scheduling solutions. Paquay et al. (2016) solved multiple bin size bin packing problem from air transportation. The BPP problem formulation is well known and presented in many papers (Fleszar & Hindi, 2002; Alvim et al., 2004). The mathematical problem formulation of 1D BPP can be modelled in the following manner. Given a set of x items $k = k_1, k_2, \ldots, k_x$ and y bins $b = b_1, b_2, \ldots, b_y$ with capacity $c \in \forall y$ the objective is to find an optimal assignment that minimizes the number of bins to be used. For every b_i the assignment should

* Corresponding author
E-mail: amol.admuthe@gmail.com (A. C. Adamuthe)

2021 Growing Science Ltd.
doi: 10.5267/j.ijiec.2020.11.002

be in such a way that the sum of weights of items in b_i should not exceed the capacity c. In mathematical terms it can be written as

$$min \qquad \sum_{i=1}^{y} b_i \tag{1}$$

subject to

$$\sum_{j=1}^{x} k_j \,.\, y_{ij} \leq c_i \forall i \tag{2}$$

$$y_{ij} = 1 \; if \; object \; j \; is \; assigned \; \text{to} \; bin \; i, 0 \; otherwise$$

$$b_i = 1 \; if \; i^{th} \; bin \; is \; used, 0 \; otherwise$$

Various deterministic approaches were proposed for solving the bin packing problem. Coffman et al. (1980) analyzed Next-Fit Decreasing-Height (NFDH) and First-Fit Decreasing-Height (FFDH) algorithms. A Hybrid First Fit (HFF) algorithm presented in (Chung et al., 1982). Simchi-Levi (1994) showed that Best-Fit (BF) and First-Fit (FF) algorithms have a worst-case ratio of 7/4. Optimization problems are NP in nature which means that there are no polynomial-time algorithms and therefore these algorithms often do not provide optimal solutions for large instances. Computational time for exact methods to solve optimization problems increases exponentially that leads to issues in practical applications (Stutzle, 1999; Gary & Johnson, 1979). In the last four-decades multiple papers showed the benefits of approximate methods which provide sub-optimal solutions in finite time (Stutzle, 1999). In literature, several meta-heuristic algorithms were investigated for solving the bin packing problem. An evolutionary randomized heuristic algorithm was presented for solving 2D bin packing (Blum & Schmid, 2013). Wong & Lee (2009) proposed an improved Lowest Gap Fill heuristic placement routine for solving the 2D bin packing problem. Various hybrid heuristic algorithms for solving bin packing problems can be found in (Frenk & Galambos, 1987; Monaci & Toth, 2006; Baker et al., 1980). Hybrid tabu search approaches were presented for solving two-dimensional bin packing problems (Lodi et al., 1999a; Lodi et al., 1999b; Lodi et al., 1999c). Parreño et al. (2010) proposed GRASP/VND algorithm for solving container loading problem which is a three-dimensional bin-packing problem and results proved to be better than existing algorithms. The HSA is an optimization algorithm based on population heuristic in which the improvisation strategy is similar to that of music players. The HSA has been utilized in several research problems. Hasanipanah et al. (2020) proposed novel hybrid artificial neural network with adaptive HSA for approximating blast-induced flyrock. Chen et al. (2012) presented dynamic harmony search to minimize makespan for identical parallel machines scheduling problem. Manjarres et al., (2013) shown several applications of HSA. In order to solve BPP, a variant of harmony search algorithm which uses adaptive PAR strategy with improved intensification-diversification and random operator is proposed. A problem specific HM representation, HM re-initialization strategy is also proposed. Two different PAR strategies are also tested. Finally, the performance of this proposed algorithm is tested on 180 benchmark instances and compared with two variants of HSA. These benchmark instances consist of 100, 200 and 500 objects of varying sizes. Various evaluation metrics are being used to measure the effectiveness of the proposed algorithm.

The remainder of this paper is structured as follows: Section 2 introduced basic HSA and related work. In section 3, proposed HSA is discussed. Section 4 provides details on the dataset used and compares the results obtained during experiments using different evaluation metrics. Finally, this paper is concluded in section 5.

2. Harmony search algorithm and related work

The HSA first presented by Geem et al. (2001) used the improvisation strategy used by music players. Musician's goal is to find a perfect harmony which is analogous to finding a global solution determined by objective function in optimization problems. HSA is an evolutionary stochastic global optimization method which can be related to particle swarm optimization, genetic algorithms etc. In the process of improvising music, notes represent components of the harmony vector which is a solution vector S of size N representing all the notes. If all the notes contribute to a good harmony then that solution vector is kept in the memory as shown in Fig. 1.

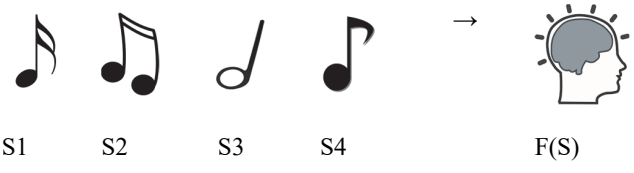

Fig. 1. Harmony vector representation with musical notes

Good harmonies are kept in the memory whereas worst harmonies are replaced by new solutions that are better than previous solutions. This is how improvisation occurs. These harmonies are similar to the fitness function of an optimization problem. Fig. 2 shows the optimization procedure for basic harmony search algorithm. There are total 5 steps involved.

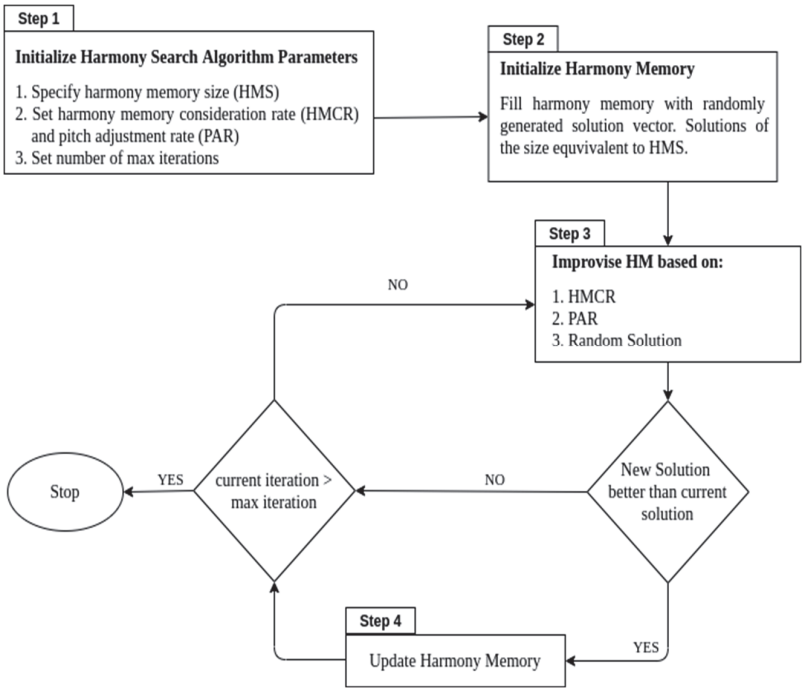

Fig. 2. Basic harmony search algorithm procedure

Step 1: This is the primary step of basic HSA where initialization of various harmony search parameters like harmony memory consideration rate (HMCR), pitch adjustment rate (PAR), max iterations i.e. the termination criteria and harmony memory size (HMS) is done.

Step 2: In this step harmony memory i.e. solution vector (S1, S2, …, SN) are initialized randomly. The Fig. 3 shows the general harmony memory matrix, where N is the HMS.

$$HarmonyMemory = \begin{bmatrix} S_1 \\ S_2 \\ . \\ . \\ . \\ S_N \end{bmatrix}$$

Fig. 3. Harmony memory matrix

Step 3: Once the harmony memory is filled with solution vectors then it is improvised in this step. Improvisation is based on HMCR, PAR and randomization. HMCR helps to maintain the balance between intensification and diversification which plays an important role in HSA. PAR and randomization maintain diversification in HSA Yang (2009). Based on these three parameters, a new solution vector S' is generated. A solution from HM is selected by the probability of HMCR and then each member of that solution is adjusted by the probability of PAR. HMCR ∈ [0, 1] and PAR ∈ [0, 1].

Step 4: If the newly generated solution vector S' in the step 3 is better than the solution vector present in the HM then the HM is updated with the new solution.

Step 5: Repeat step 3 and step 4 until maximum iterations are reached.

As stated earlier HSA has two important parameters: HMCR and PAR. These are probabilistic parameters in which HMCR is concerned with exploration and exploitation rate and PAR is the probability with which the selected value from memory is being adjusted. Fine tuning HSA with proper parameter values is a challenging task. This leads to variations of HSA with

adaptive parameter setting (Moh'd Alia & Mandava, 2011). In literature, a large number of papers improved the performance of HSA by fine tuning parameter setting and hybridization with other heuristic algorithms (Wang et al., 2011; Wu et al., 2012; Pandi & Panigrahi, 2011). Various variations of HSA have been proposed for solving optimization problems. An improved harmony search (IHS) was proposed by Mahdavi et al. (2007) which uses linearly increasing adaptive PAR. Omran & Mahdavi (2008) addressed the drawbacks of IHS and proposed global-best harmony search (GHS) which intuitively improvises harmony memory by replacing bandwidth parameters. Wang & Huang (2010) proposed dynamic selection of parameters in which PAR value is decreased linearly. A mutation operator inspired from the Differential Evolution algorithm is replaced by the PAR operator (Chakraborty et al., 2009). Hasançebi et al. (2010) and Saka & Hasançebi (2009) proposed dynamic tuning of HMCR and PAR parameters in the improvisation process. HMCR and PAR values start with zero and then dynamically adapt during the HS process. Geem et al. (2005) and Al-Betar et al. (2010) presented multiple PAR strategies in HS. Simulated annealing strategy is used to update the PAR values (Taherinejad, 2009). To update the PAR values Dispersed PSO component is used (Coelho & Bernert, 2009). El-Abd (2013) presented a novel strategy for adaptive change in PAR and BW values. A binary harmony search algorithm with adaptive parameter setting is applied to solve large-scale 0–1 knapsack problems (Kong et al., 2015). ABHS tunes HMCR and PAR based on the feedback from the evolutionary search (Guo et al., 2018).

3. Hybrid adaptive harmony search algorithm

This section presents the harmony memory representation, harmony memory initialization strategy. It also discusses re-initialization strategy and the proposed HSA with improved intensification-diversification capability with random operator based iterative local search.

3.1 Harmony memory representation

In 1D BPP we have k bins B1, B2, …, Bk with similar or varying capacities and O objects each with different weights. Objective is to allocate all objects to these bins in such a way that a minimum number of bins are used. A real coded solution representation for solving 1D BPP is shown in (Adamuthe & Nitave, 2020). Fig. 4 shows the solution representation with 4 objects and 2 bins where Object1 and Object3 are being assigned to Bin1 and remaining two objects are being assigned to Bin2. This forms one solution vector where every object is assigned to exactly one bin without violating the constraints and objective is to minimize the number of bins being used.

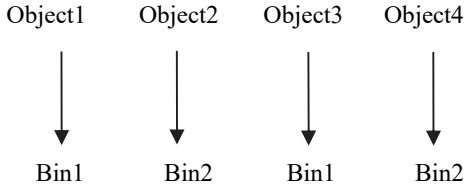

Fig. 4. Solution representation

The Fig. 5 shows the harmony memory matrix representation where HMS is equivalent to N and number of objects to be packed are equal to M. In harmony memory every row corresponds to one solution vector and there are N such solutions vectors.

$$\begin{pmatrix} S_{11} & \cdots & S_{1M} \\ \vdots & \ddots & \vdots \\ S_{N1} & \cdots & S_{NM} \end{pmatrix}$$

Number of Objects

Fig. 5. HM representation for 1D BPP

3.2 Harmony memory initialization

For initializing the harmony memory a first-fit strategy with randomization is used. A bin 'B' is assigned to object 'O' such that,

$$\text{bin\_size}_x + \text{obj\_size}_y \leq \text{max\_bin\_size}_x \tag{3}$$

where, bin_size_x is the current size of the bin B and obj_size_y is the size of the object O. If this constraint in Eq. (3) fails then a bin is randomly assigned to the object. If the randomly assigned bin B cannot allocate object O without satisfying the above constraint then a penalty equivalent to Eq. (4) is applied.

$$\text{bin\_size}_x = -\left(\text{bin\_size}_x + \text{obj\_size}_y\right) \tag{4}$$

3.3 Harmony search with increasing PAR

In this variant of HSA the value of pitch adjusting rate is increased along with every generation. The value of PAR changes as per equation (5) given in (Mahdavi et al., 2007).

$$PAR = PAR + \left(PAR_{min} - \left(\frac{PAR_{max} - PAR_{min}}{\text{NI}}\right) \times gen\right) \tag{5}$$

where,

PAR	pitch adjusting rate for every iteration/generation	NI	maximum number of iterations
PAR_{min}	initial and minimum value of pitch adjusting rate	Gen	iteration/generation number
PAR_{max}	maximum value of pitch adjusting rate		

In step 1 i.e. initialization of parameters the value of PAR_{min} and PAR_{max} is set to 0.45 and 1.0 respectively whereas initial value of PAR is set similar to that of PAR_{min}. This variant of harmony search is referred to as version 1 in this paper.

3.4 Harmony search with decreasing PAR

In this version of adaptive harmony search the value of PAR is decreased with increase in iteration according to the Eq. (6).

$$PAR = PAR - \left(PAR_{min} - \left(\frac{PAR_{max} - PAR_{min}}{\text{NI}}\right) \times gen\right) \tag{6}$$

For this variant of harmony search the value of PAR_{min} and PAR_{max} is set to 0.85 and 1.0 respectively. This variant of HSA is reffered to as version 2 in this paper. The Fig. 6 shows the change in values of PAR with the iteration for HSA with version 1 and version 2. As compared to traditional HSA, these two versions use dynamic PAR which enhances the performance of the HSA.

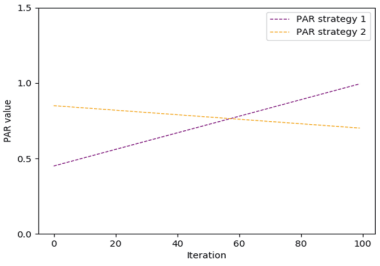

Fig. 6. Change in PAR with iteration

The pitch adjustment rate parameter in the HSA helps to boost the diversification capability as it controls the calibration of the components of the solution vector in HM. The calibration works either by adding or removing the assigned bins from an existing component. Good harmonies are retained while others are fine-tuned.

3.5 Re-initialization strategy

As the complexity of the problem increases it gets difficult to find optimal solutions because the algorithms don't converge as the components of solution vector gets saturated and the scope for exploration decreases. So, in order to induce diversity in solution vectors a re-initialization strategy is proposed by Adamuthe & Nitave, (2020). For re-initializing the harmony memory, n solution vectors are selected from HM and re-initialized according the strategy of HM initialization discussed previously. Deciding the value of n is very important because too high value will destroy previously converged solution vectors and too low value will not effectively create diversity. Setting the threshold parameter and updating the threshold while the algorithm is converging is also a crucial task. This threshold can be different for different datasets. Fig. 7 shows the diversity in the harmony memory after the first iteration and Fig. 8 shows the diversity in the harmony memory after 1000 iterations on sample instance with 500 objects and harmony memory size equivalent to 100. As we can see in Fig. 8 the solution vectors get converged and look identical to one another and thus fail to converge further. The algorithm 1 shows the pseudocode for re-initialization strategy where n is the number of solution vectors from HM that should be selected for re-initialization.

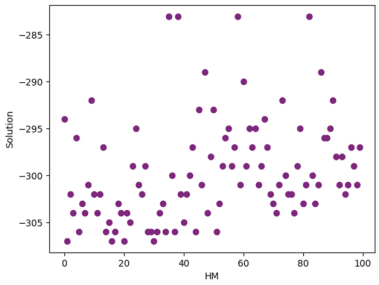

Fig. 7. Diversity of solution vectors in HM after first iteration

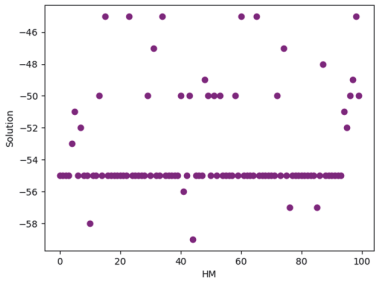

Fig. 8. Diversity of solution vector in HM after 1000th iteration

As described earlier the value of 'n' is important for re-initialization strategy. There is one more parameter namely *threshold* which decides how often the re-initialization strategy is executed. If the threshold is too low then re-initialization module is not executed frequently which results in delayed convergence and if it is too high then all the solutions in the HM will be destroyed before converging. The value of threshold highly depends on size and complexity of the optimization problem. The value of threshold is nothing but the iteration value that will execute re-initialization module. In random consideration, the new vector's components are generated at random mode, has the same level of efficiency as in other algorithms that handle randomization, where this property allows HS to explore new regions that may not have been visited in the search space.

Algorithm 1: Pseudocode for Re-initializing harmony memory

```
    Re-initialize (n):
1   for i = 0 to n
2      for all objects
3         assign bin using HM initialization strategy
4      update new solution in the HM
```

The algorithm 2 shows the detailed pseudocode for HSA with dynamic PAR. Similar to basic harmony search we first initialize all the parameters – HMCR, PAR, threshold, max_iterations, of PAR_{min} and PAR_{max}. The value of PARmin is set to 0.45 and 0.85 for version 1 and version 2 respectively whereas of PAR_{max} is set to 1.0 for both the versions. Also, the update equation for PAR changes according to harmony search variant.

Algorithm 2: Pseudocode for Harmony search algorithm with Dynamic PAR

```
Input: Maximum number of objects (no_obj), capacity of bins (bin_cap)
Output: Minimum number of bins required

Initialize HMS, HMCR, max iterations (itrmax), threshold
Set PARmin and PARmax
Initialize harmony memory step 1
1.  for itr = 0 to itrmax
2.     for hms = 0 to HMS
3.        for obj=0 to num_obj
4.           if rand[0,1] < HMCR
5.              call memory_consideration() method for current value of obj and hms
6.              if rand[0, 1] < PAR
7.                 if bin size is more than it's capacity
8.                    adjust the pitch of the current hms and obj
9.                 end
10.             end
11.          end
12.          else
13.             randomly select bin for current obj
14.          end
15.       end
16.       update the memory of HMS
17.    end
18.    update PAR using adaptive equation
19.    if itr > threshold
20.       Re-initialize the harmony memory to introduce randomization
21.       update threshold
22.    end
23. end
```

However this variant of harmony search is not able to find optimal solutions as the dimension of the optimization problem increases. In order to solve 1D-BPP with 500 objects another variant of HSA is proposed and discussed in the next section.

3.6 Harmony search with iterative local search

As the number of objects to be packed increases the minimum number of required bins also increases. This makes the search space bigger and therefore HSA in version 1 and version 2 fail to converge. In order to solve the 1D bin packing problem with a large number of objects, an harmony search algorithm with iterative local search is proposed which works on the concept of despotism which means that the decision is executed by the dominant (best) solution vector so we use the objective value represented by Eq. (7).

$$X^{best} = HM(best, 1{:}n) \tag{7}$$

where *best* represents the index of the best solution vector in the harmony memory. This strategy divides all the solution vectors into two categories such that solutions in first category have values less than HM_{mean} and the solutions in the second category have values more than that of HM_{mean} where HM_{mean} is the mean of the entire harmony memory. First category is responsible for intensification and diversification whereas latter is responsible for diversification. In the Eq. (8) and Eq. (9) y_i^{best} and y_i^{worst} are the i^{th} element of the best and worst solution vector in the harmony memory. Eq. (8) allows the algorithm to search between selected element y_i and the y_i^{best} search space and it is responsible for intensification. Equation (9) is responsible for diversification where selected element y_i is close to y_i^{worst} which makes (y_i^{worst} - y_i) smaller so we add y_i^{best} which moves it closer to the y_i^{best}. On the other hand if the selected element y_i is far from y_i^{worst} then the value moves away from y_i^{best}. The solution vectors belonging to the second category which are responsible only for diversification the pitch adjustment strategy is given by Eq. (10) where *m* is used to store the index of solution vector from harmony memory (Yadav et al., 2012).

$$z_i = y_i^{best} - \left(y_i^{best} - y_i\right) \times rand[0,1] \tag{8}$$

$$z_i = y_i^{best} + (y_i^{worst} - y_i) \times rand[0,1] \tag{9}$$

$$z_i = y_i + (y_m^{best} - y_i) \times rand[0,1] \tag{10}$$

The intensification and diversification strategy proposed in equations 8, 9 and 10 improves the convergence of the HSA and helps in finding the region of best solution but fails to provide the best solution. In order to further solve the problem an iterative local search is proposed which is based on random swap operation. This hybridization is relatively simple yet very effective strategy for solving BPP. The local iterative search enhances the performance by exploring multiple neighborhoods in the region of best solution. In this strategy, N swaps are performed on components of a selected solution vector. The value of N is adaptive and changes with the objective values of solution vectors in HM. The Fig. 9 shows that as the proposed algorithm approaches the best solution the number of swaps performed increases exponentially. This exponential increase in the number of swaps helps the algorithms to find the solution. The Fig. 10 shows the number of swaps performed for six different datasets with 500 objects. The algorithm 3 shows the detailed pseudocode for the iterative local search strategy. The variable row_index represents the solution vector which needs to be selected from the HM. After this the value of N is calculated, two objects obj_1 and obj_2 are selected randomly and then the swap is performed betweeen the bins of two selected objects. The newly generated solution is updated in the HM if it is better than previous solution. Exponential swaps are performed to find the optimal solutions. The algorithm 4 shows the detailed pseudocode for the proposed HSA.

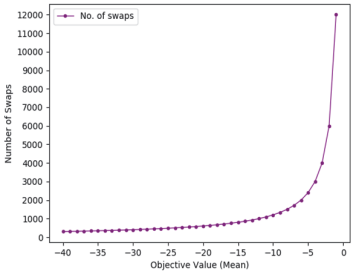

Fig. 9. Number of swaps performed by local iterative search on sample instance

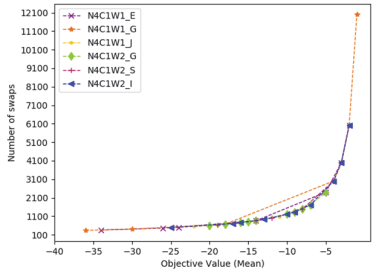

Fig. 10. Iterative local search on 6 difference instances

Algorithm 3: Pseudocode for iterative local search

swap(row_index)
$N \leftarrow$ *computed based on global* HM_{mean} and *population* size
1 **for** i=0 to N
2 $obj_1 \leftarrow \left(1 + \left(num_{obj} - 1\right) \times rand[0,1]\right)$
3 $obj_2 \leftarrow \left(1 + \left(num_{obj} - 1\right) \times rand[0,1]\right)$

```
5              swap bin of obj_1 with the bin of obj_2 and vice-versa
6          if new solution is better than previous solution
7              update memory
8   end
```

Algorithm 4: Pseudocode for Hybrid harmony search algorithm

```
Input: Maximum number of objects (no_obj), capacity of bins
Output: Minimum number of bins required

Initialize HMS, HMCR, max iterations (itrmax) : step 1
1   Set PAR_min → 0.85, PAR_max → 1.0
2   Initialize harmony memory : step 2
3   Improvisation : step 3
4   for itr = 0 to itrmax
5      for hms = 0 to HMS
6         for obj=0 to num_obj
7            if rand[0,1] < HMCR
8               call memory_consideration() method and store a solution vector from the harmony
9               memory in var d
10              if rand[0, 1] < PAR
11                 if bin size is less than mean weight of HM
12                 // Category 1: Both Intensification and Diversification
13                    if rand[0,1] <= 0.5
14                       perform intensification as per equation (8)
15                    else
16                       perform diversification as per equation (9)
17                    end
18                 // Category 2: Diversification
19                 else
20                    m = (1+ (number_of_obj - 2)*rand[0, 1])
21                    if obj weight is less than 70% of bin capacity
22                       perform diversification as per equation (10)
23                    end
24                 end
25              end
26              if 0 <= val <= max_number_bins
27                 assign val to HM[d][obj]
28              end
29           end
30           else
31              randomly assign at bin for current obj
32           end
33        end
34        perform random swap iteratively
35        update the memory of HMS
36     end
37  end
```

4. Results and performance evaluation

This section gives a detailed explanation about the dataset used for experiments and the results obtained. The proposed algorithm is implemented in C and tested on a computer with following specifications: Ubuntu 18.04 LTS, Intel core i5-8265U CPU 1.6 GHz with 8 GB RAM. Every experiment was executed for 10 times. A set of benchmark datasets are used for evaluating the proposed algorithm. The dataset can be found at https://www2.wiwi.uni-jena.de/Entscheidung/binpp/index.htm. These datasets are prepared as per Martello and Toth (1990) and best known solutions are already provided. The naming conventions of the dataset are as follows.

$N_xC_yW_{zi}$ where,
x=3 (200 objects), x = 4 (500 objects)
y=1 (bin capacity = 100), y=2 (bin capacity = 120), y=3 (bin capacity = 150)
z=1 (W_j from [1, 100]), z=2 (W_j from [20, 200]), z=4 (W_j from [30, 100])
i=A to T with 20 instances of each class

In all there were 180 instances which consist of x =3 and 4, y=1, z=1,2,4. Every instance consists of a number of objects, bin capacity and size of each object separated by a new line. Table 1 shows the structure of input data. The optimal solutions for these datasets can be found in (Flezar and Hindi, 2002; Schoenfield, 2002; Alvim et al., 2004; Schwerin and Wäscher, 1997).

Table 1
Input format of the benchmark instances

500← Number of objects	240← Capacity of bin	100 100 100 } Size of items

To evaluate the performance of all the variants of harmony search, this paper uses mean, standard deviation and best where best is the most converged solution in the harmony memory. Suganthan et al. (2005) presented six metrics for evaluating the performance of the algorithms. These six parameters are namely convergence graph, average number of function evaluations (NFEs), success rate (SR), acceleration rate (AR) and improvement. Parameters are calculated using the following equations.

$$Success\ rate = \frac{number\ of \text{ times } best\ solution\ obtained}{number\ of \text{ times } program\ executed} \tag{11}$$

$$NFEs = HM \times Number\ of\ iterations\ required\ for\ converging \tag{12}$$

$$Acceleration\ rate = \frac{NFE\ of\ HSA\ variant(v1 \text{ or } v2)}{NFE\ of\ proposed\ HSA\ (v3)} \tag{13}$$

$$Improvement = \left(NFE_{other} - NFE_{proposed}\right) \times \frac{100}{NFE_{other}} \tag{14}$$

Table 2 and 3 show the performance of three HS variations on 60 datasets with 200 objects and 60 datasets with 500 objects respectively. For evaluation, metrics such as best, mean and standard deviation are used. The negative objective values of best and mean indicates the penalties applied for constraint violations. The proposed harmony search algorithm (V3) outperforms the other two variations for all the instances. Whereas the performance of version 1 and version 2 are approximately the same.

Table 2
Best, mean and standard deviation values for proposed HS variations for 60 datasets with 200 objects

Dataset	Metrics	Algorithm			Dataset	Metrics	Algorithm		
		V3	V2	V1			V3	V2	V1
N3C1W1_A.BPP	Best	-2	-34	-2	N3C1W2_K.BPP	Best	-8	-43	-14
	SD	0	17.58	49.84		SD	0	18.30	43.37
	Mean	-2	-84	-41		Mean	-8	-93	-41
N3C1W1_B.BPP	Best	0	-9	0	N3C1W2_L.BPP	Best	0	-4	0
	SD	0	20.66	46.92		SD	0	18.78	45.69
	Mean	0	-65	-12.49		Mean	0	-49	-12.59
N3C1W1_C.BPP	Best	0	0	0	N3C1W2_M.BPP	Best	0	0	0
	SD	0	0	56.71		SD	0	22.01	46.06
	Mean	0	0	-17.51		Mean	0	-48	-17.69
N3C1W1_D.BPP	Best	0	-3	0	N3C1W2_N.BPP	Best	0	-18	0
	SD	0	16.14	57.83		SD	0	30.03	46.92
	Mean	0	-47	-19.04		Mean	0	-105	-19.04
N3C1W1_E.BPP	Best	0	-3	0	N3C1W2_O.BPP	Best	-20	-6	0
	SD	0	6.46	70.4		SD	0	17.89	49.84
	Mean	0	-15	-28.75		Mean	-20	-52	-28.75
N3C1W1_F.BPP	Best	0	-2	0	N3C1W2_P.BPP	Best	0	-1	0
	SD	0	14.03	66.12		SD	0	20.03	54.12
	Mean	0	-22	-21.84		Mean	0	-45	-21.65
N3C1W1_G.BPP	Best	0	0	0	N3C1W2_Q.BPP	Best	0	0	0
	SD	0	0	43.37		SD	0	23.08	56.71
	Mean	0	0	-10.76		Mean	0	-48	-1025
N3C1W1_H.BPP	Best	0	-10	0	N3C1W2_R.BPP	Best	-15	-52	-8
	SD	0	6.67	45.69		SD	0	22.14	57.83
	Mean	0	-26	-12.32		Mean	-22.96	-120	-12.32
N3C1W1_I.BPP	Best	0	-1	0	N3C1W2_S.BPP	Best	0	0	0
	SD	0	14.52	46.05		SD	0	12.49	66.12
	Mean	0	-20	-10.47		Mean	0	-20	-10.48
N3C1W1_J.BPP	Best	0	0	0	N3C1W2_T.BPP	Best	0	-1	0
	SD	0	15.24	55.172		SD	0	14.12	72.4
	Mean	0	-21	-15.6		Mean	0	-22	-15.2
N3C1W1_K.BPP	Best	0	-4	0	N3C1W4_A.BPP	Best	0	-8	-2
	SD	0	11.35	52.65		SD	0	17.56	43.37
	Mean	0	-41	-15.43		Mean	0	-51	-41
N3C1W1_L.BPP	Best	0	-50	0	N3C1W4_B.BPP	Best	0	-1	0
	SD	0	6.91	43.37		SD	0	14.11	45.69
	Mean	0	-69	-10.76		Mean	0	-22	-12.49

N3C1W1_M.BPP	Best	0	0	0	N3C1W4_C.BPP	Best	0	-12	0
	SD	0	14.38	45.69		SD	0	23.17	46.05
	Mean	0	-21	-12.32		Mean	0	-73	-17.51
N3C1W1_N.BPP	Best	0	-38	0	N3C1W4_D.BPP	Best	0	-13	0
	SD	0	8.35	46.05		SD	0	20.16	46.92
	Mean	0	-53	-10.47		Mean	0	-65	-19.04
N3C1W1_O.BPP	Best	-20	-79	-12	N3C1W4_E.BPP	Best	0	-2	0
	SD	0	12.77	52.65		SD	0	11.42	49.84
	Mean	-20	-117	-15.43		Mean	0	-25	-28.75
N3C1W1_P.BPP	Best	0	-10	0	N3C1W4_F.BPP	Best	0	-8	0
	SD	0	20.53	55.172		SD	0	14.68	55.172
	Mean	0	-42	-15.6		Mean	0	-44	-21.84
N3C1W1_Q.BPP	Best	0	-13	0	N3C1W4_G.BPP	Best	0	-10	0
	SD	0	6.23	45.69		SD	0	19.16	56.71
	Mean	0	-36	-12.32		Mean	0	-69	-10.76
N3C1W1_R.BPP	Best	0	-2	0	N3C1W4_H.BPP	Best	0	-4	0
	SD	0	12.67	0		SD	0	16.53	57.83
	Mean	0	-31	0		Mean	0	-62	-12.32
N3C1W1_S.BPP	Best	0	-27	0	N3C1W4_I.BPP	Best	0	-24	0
	SD	0	15.52	0		SD	0	20.29	66.12
	Mean	0	-69	0		Mean	-4.55	-86	-10.47
N3C1W1_T.BPP	Best	0	-15	0	N3C1W4_J.BPP	Best	0	-21	0
	SD	0	13.62	0		SD	0	20.81	70.4
	Mean	0	-62	0		Mean	0	-76	-15.6
N3C1W2_A.BPP	Best	-2	-2	-111	N3C1W4_K.BPP	Best	0	0	0
	SD	0	11.59	169.46		SD	0	0	66.12
	Mean	-2	-28	-157.02		Mean	0	0	-80.58
N3C1W2_B.BPP	Best	0	0	-113	N3C1W4_L.BPP	Best	0	0	0
	SD	0	12.81	196.42		SD	0	22.48	56.71
	Mean	0	-19	-182.3		Mean	0	-45	-10.76
N3C1W2_C.BPP	Best	0	-42	-98	N3C1W4_M.BPP	Best	0	-8	0
	SD	0	14.47	162.68		SD	0	29.17	57.83
	Mean	0	-86	-140.5		Mean	0	-98	-12.32
N3C1W2_D.BPP	Best	0	0	-111	N3C1W4_N.BPP	Best	0	-4	0
	SD	0	15.81	169.46		SD	0	24.88	45.69
	Mean	0	-22	-157.0		Mean	0	-61	-12.49
N3C1W2_E.BPP	Best	0	-1	-1	N3C1W4_O.BPP	Best	-20	-1	0
	SD	0	18.62	163.36		SD	0	15.78	70.4
	Mean	0	-47	-74.6		Mean	-20	-34	-15.6
N3C1W2_F.BPP	Best	0	-30	-37	N3C1W4_P.BPP	Best	0	-3	0
	SD	0	19.30	217.29		SD	0	22.92	46.05
	Mean	0	-84	-106.1		Mean	0	-46	-17.51
N3C1W2_G.BPP	Best	0	-14	-30	N3C1W4_Q.BPP	Best	0	-23	0
	SD	0	13.85	174.79		SD	0	18.96	46.92
	Mean	0	-72	-87.75		Mean	0	-78	-19.04
N3C1W2_H.BPP	Best	0	-39	-79	N3C1W4_R.BPP	Best	0	0	0
	SD	0	16.14	165.43		SD	0	18.04	55.172
	Mean	-6.93	-86	-125.7		Mean	0	-22	-21.84
N3C1W2_I.BPP	Best	0	-25	-48	N3C1W4_S.BPP	Best	0	-21	0
	SD	0	13.19	164.27		SD	0	23.75	49.84
	Mean	0	-72	-132.4		Mean	0	-75	-28.75
N3C1W2_J.BPP	Best	0	-20	-1	N3C1W4_T.BPP	Best	0	-1	0
	SD	0	14.61	192.38		SD	0	12.36	43.37
	Mean	0	-59	-81.66		Mean	0	-21	-41

Table 3
Best, mean and standard deviation values for proposed HS variations for 60 datasets with 500 objects

Dataset	Metrics	Algorithm			Dataset	Metrics	Algorithm		
		V3	V2	V1			V3	V2	V1
N4C1W1_A.BPP	Best	0	-113	-111	N4C1W2_K.BPP	Best	0	0	0
	SD	0	0	169.469		SD	0	0	205.78
	Mean	-6.97	-113	-157.02		Mean	0	0	-36.12
N4C1W1_B.BPP	Best	0	-5	-113	N4C1W2_L.BPP	Best	0	0	0
	SD	0	9.37	196.42		SD	0	0	240.41
	Mean	0	-80.96	-182.3		Mean	0	0	-48.97
N4C1W1_C.BPP	Best	0	-100	-98	N4C1W2_M.BPP	Best	0	0	0
	SD	0		162.689		SD	0	0	141.89
	Mean	0	-110.8	-140.53		Mean	0	0	-18.43
N4C1W1_D.BPP	Best	0	-117	-111	N4C1W2_N.BPP	Best	0	0	0
	SD	0	6.557	169.469		SD	0	0	204.244
	Mean	0	-139.9	-157.02		Mean	0	0	-35.85
N4C1W1_E.BPP	Best	0	-1	-1	N4C1W2_O.BPP	Best	0	0	0

	SD	0	12.36	163.369		SD	0	0	234.16
	Mean	0	-44.26	-74.6		Mean	0	0	-49.74
	Best	0	-22	-37		Best	0	0	0
N4C1W1_F.BPP	SD	0	7.28	217.29	N4C1W2_P.BPP	SD	0	0	252.81
	Mean	0	-57.57	-106.14		Mean	0	0	-55.81
	Best	0	-29	-30		Best	0	0	0
N4C1W1_G.BPP	SD	0	6.48	174.79	N4C1W2_Q.BPP	SD	0	0	250.39
	Mean	0	-57.06	-87.75		Mean	0	0	-53.16
	Best	0	-65	-79		Best	0	0	0
N4C1W1_H.BPP	SD	0	3.74	165.43	N4C1W2_R.BPP	SD	0	0	161.322
	Mean	0	-98.2	-125.78		Mean	0	0	-22.98
	Best	-9	-40	-48		Best	0	-1	-2
N4C1W1_I.BPP	SD	0	13.19	164.277	N4C1W2_S.BPP	SD	0	6.32	210.6
	Mean	-9	-103.4	-132.43		Mean	0	-20.	-57.66
	Best	-13	-1	-1		Best	0	0	0
N4C1W1_J.BPP	SD	0	13.07	192.38	N4C1W2_T.BPP	SD	0	0	193.05
	Mean	-13	-44.06	-81.66		Mean	0	0	-33.87
	Best	0	-1	-15		Best	0	0	0
N4C1W1_K.BPP	SD	0	6.08	179.61	N4C1W4_A.BPP	SD	0	0	211.95
	Mean	0	-47.37	-80.58		Mean	0	0	-39.19
	Best	0	-36	-35		Best	0	0	0
N4C1W1_L.BPP	SD	0	8.48	160.59	N4C1W4_B.BPP	SD	0	0	217.407
	Mean	0	-96.62	-122.87		Mean	0	0	-40.24
	Best	0	-53	-54		Best	0	0	0
N4C1W1_M.BPP	SD	0	3.31	178.69	N4C1W4_C.BPP	SD	0	0	201.92
	Mean	0	-65.2	-100.35		Mean	0	0	-35.43
	Best	0	0	0		Best	0	0	0
N4C1W1_N.BPP	SD	0	0	0	N4C1W4_D.BPP	SD	0	0	174.41
	Mean	0	0	0		Mean	0	0	-26.89
	Best	0	0	0		Best	0	0	0
N4C1W1_O.BPP	SD	0	0	0	N4C1W4_E.BPP	SD	0	0	271.58
	Mean	0	0	0		Mean	0	0	-62.1
	Best	0	0	0		Best	0	0	0
N4C1W1_P.BPP	SD	0	0	0		SD	0	0	197.43
	Mean	0	0	0	N4C1W4_F.BPP	Mean	0	0	-36.59
	Best	0	0	0		Best	0	0	0
N4C1W1_Q.BPP	SD	0	0	0	N4C1W4_G.BPP	SD	0	0	182.7
	Mean	0	0	0		Mean	0	0	-28.13
	Best	0	0	0		Best	0	0	0
N4C1W1_R.BPP	SD	0	0	0	N4C1W4_H.BPP	SD	0	0	200.36
	Mean	0	0	0		Mean	0	0	-33.08
	Best	0	0	0		Best	0	0	0
N4C1W1_S.BPP	SD	0	0	0	N4C1W4_I.BPP	SD	0	0	237.4
	Mean	0	0	0		Mean	0	0	-48.36
	Best	0	0	0		Best	0	0	0
N4C1W1_T.BPP	SD	0	0	0	N4C1W4_J.BPP	SD	0	0	225.17
	Mean	0	0	0		Mean	0	0	-41.72
	Best	0	0	0		Best	0	0	0
N4C1W2_A.BPP	SD	0	0	160.04	N4C1W4_K.BPP	SD	0	0	210.47
	Mean	0	0	-22.79		Mean	0	0	-36.93
	Best	0	0	0		Best	0	0	0
N4C1W2_B.BPP	SD	0	0	259.65	N4C1W4_L.BPP	SD	0	0	261.87
	Mean	0	0	-59.44		Mean	0	0	-57.74
	Best	0	0	0		Best	0	0	0
N4C1W2_C.BPP	SD	0	0	217.25	N4C1W4_M.BPP	SD	0	0	161.89
	Mean	0	0	-42.19		Mean	0	0	-23.07
	Best	0	0	0		Best	0	0	0
N4C1W2_D.BPP	SD	0	0	150.72	N4C1W4_N.BPP	SD	0	0	199.6
	Mean	0	0	-20.09		Mean	0	0	-32.97
	Best	0	0	0		Best	0	0	0
N4C1W2_E.BPP	SD	0	0	222.51	N4C1W4_O.BPP	SD	0	0	228.64
	Mean	0	0	-43.3		Mean	0	0	-44.49
	Best	0	0	0		Best	0	0	0
N4C1W2_F.BPP	SD	0	0	170.622	N4C1W4_P.BPP	SD	0	0	192.58
	Mean	0	0	-26.31		Mean	0	0	-31.81
	Best	-2	-1	-1		Best	0	0	0
N4C1W2_G.BPP	SD	0	7.74	186.72	N4C1W4_Q.BPP	SD	0	0	210.5
	Mean	-2	-23.32	-55.24		Mean	0	0	-38.9
	Best	0	0	0		Best	0	0	0
N4C1W2_H.BPP	SD	0	0	199.15	N4C1W4_R.BPP	SD	0	0	221.69
	Mean	0	0	-34.92		Mean	0	0	-41.08
	Best	-1	-1	-1		Best	0	0	0

	SD	0	6.164	214.289		SD	0	0	186.45
N4C1W2_I.BPP	Mean	-1	-17.2	-59.19	N4C1W4_S.BPP	Mean	0	0	-30.73
	Best	0	-32	-31		Best	0	0	0
N4C1W2_J.BPP	SD	0	11.48	187.18	N4C1W4_T.BPP	SD	0	0	199.42
	Mean	0	-69.2	-100.56		Mean	0	0	-35

The success rate of three versions of harmony search algorithm is shown in Table 4. For experimentation 180 instances are used. Versions V1, V2, V3 works well for instances with 100 objects. Results show that the success rate of V1 and V2 decreases for instances with 500 objects. Version 3 has a success rate of 100% for more than 80% of the instances whereas V2 has a success rate of approximately 56% and V1 has a success rate of approximately 6%.

Table 4
Success rate for proposed HS variations for 180 datasets

Dataset with 500 objects	Success rate (in %)			Dataset with 200 objects	Success rate (in %)			Dataset with 100 objects	Success rate (in %)		
	V3	V2	V1		V3	V2	V1		V3	V2	V1
N4C1W1_A	100	20	10	N3C1W1_A.BPP	100	80	80	N2C1W1_A.BPP	100	100	100
N4C1W1_B	80	60	30	N3C1W1_B.BPP	100	100	60	N2C1W1_B.BPP	100	100	100
N4C1W1_C	100	70	40	N3C1W1_C.BPP	100	100	70	N2C1W1_C.BPP	100	100	80
N4C1W1_D	100	70	30	N3C1W1_D.BPP	100	100	80	N2C1W1_D.BPP	100	100	100
N4C1W1_E	80	40	20	N3C1W1_E.BPP	100	100	50	N2C1W1_E.BPP	100	100	100
N4C1W1_F	100	100	60	N3C1W1_F.BPP	100	100	40	N2C1W1_F.BPP	100	100	100
N4C1W1_G	100	90	50	N3C1W1_G.BPP	100	100	90	N2C1W1_G.BPP	100	100	90
N4C1W1_H	100	100	90	N3C1W1_H.BPP	100	100	70	N2C1W1_H.BPP	100	100	100
N4C1W1_I	100	50	30	N3C1W1_I.BPP	100	100	80	N2C1W1_I.BPP	100	100	50
N4C1W1_J	70	70	50	N3C1W1_J.BPP	100	100	30	N2C1W1_J.BPP	100	100	100
N4C1W1_K	100	50	70	N3C1W1_K.BPP	100	100	70	N2C1W1_K.BPP	100	100	70
N4C1W1_L	90	90	50	N3C1W1_L.BPP	100	100	100	N2C1W1_L.BPP	100	100	100
N4C1W1_M	80	80	40	N3C1W1_M.BPP	100	100	70	N2C1W1_M.BPP	100	100	100
N4C1W1_N	100	100	80	N3C1W1_N.BPP	100	100	90	N2C1W1_N.BPP	100	100	90
N4C1W1_O	100	30	10	N3C1W1_O.BPP	100	100	100	N2C1W1_O.BPP	100	100	100
N4C1W1_P	100	100	30	N3C1W1_P.BPP	100	100	50	N2C1W1_P.BPP	100	100	100
N4C1W1_Q	100	100	10	N3C1W1_Q.BPP	100	100	100	N2C1W1_Q.BPP	100	100	100
N4C1W1_R	100	90	0	N3C1W1_R.BPP	100	100	80	N2C1W1_R.BPP	100	100	80
N4C1W1_S	100	100	90	N3C1W1_S.BPP	100	100	40	N2C1W1_S.BPP	100	100	100
N4C1W1_T	100	100	70	N3C1W1_T.BPP	100	100	80	N2C1W1_T.BPP	100	100	100
N4C1W2_A	100	30	60	N3C1W2_A.BPP	100	100	40	N2C1W2_A.BPP	100	100	100
N4C1W2_B	100	100	30	N3C1W2_B.BPP	100	100	100	N2C1W2_B.BPP	100	100	100
N4C1W2_C	100	100	10	N3C1W2_C.BPP	100	100	90	N2C1W2_C.BPP	100	100	90
N4C1W2_D	40	20	0	N3C1W2_D.BPP	100	100	100	N2C1W2_D.BPP	100	100	100
N4C1W2_E	100	100	20	N3C1W2_E.BPP	100	100	100	N2C1W2_E.BPP	100	100	100
N4C1W2_F	100	100	100	N3C1W2_F.BPP	100	100	80	N2C1W2_F.BPP	100	100	80
N4C1W2_G	50	20	0	N3C1W2_G.BPP	100	100	100	N2C1W2_G.BPP	100	100	100
N4C1W2_H	100	100	20	N3C1W2_H.BPP	100	100	70	N2C1W2_H.BPP	100	100	100
N4C1W2_I	100	100	100	N3C1W2_I.BPP	100	100	100	N2C1W2_I.BPP	100	100	100
N4C1W2_J	70	10	80	N3C1W2_J.BPP	100	100	60	N2C1W2_J.BPP	100	100	60
N4C1W2_K	100	100	10	N3C1W2_K.BPP	100	100	100	N2C1W2_K.BPP	100	100	100
N4C1W2_L	100	70	30	N3C1W2_L.BPP	100	100	50	N2C1W2_L.BPP	100	100	50
N4C1W2_M	100	100	70	N3C1W2_M.BPP	100	100	100	N2C1W2_M.BPP	100	100	100
N4C1W2_N	100	100	40	N3C1W2_N.BPP	100	100	90	N2C1W2_N.BPP	100	100	90
N4C1W2_O	100	100	60	N3C1W2_O.BPP	100	100	80	N2C1W2_O.BPP	100	100	100
N4C1W2_P	100	100	0	N3C1W2_P.BPP	100	100	100	N2C1W2_P.BPP	100	100	100
N4C1W2_Q	80	70	10	N3C1W2_Q.BPP	100	100	60	N2C1W2_Q.BPP	100	100	100
N4C1W2_R	100	100	70	N3C1W2_R.BPP	90	70	50	N2C1W2_R.BPP	100	100	80
N4C1W2_S	100	70	30	N3C1W2_S.BPP	100	100	20	N2C1W2_S.BPP	100	100	100
N4C1W2_T	100	100	70	N3C1W2_T.BPP	100	100	50	N2C1W2_T.BPP	100	100	100
N4C1W4_A	100	100	50	N3C1W4_A.BPP	100	100	70	N2C1W4_A.BPP	100	100	70
N4C1W4_B	100	20	0	N3C1W4_B.BPP	100	100	100	N2C1W4_B.BPP	100	100	100
N4C1W4_C	100	100	20	N3C1W4_C.BPP	100	100	100	N2C1W4_C.BPP	100	100	100
N4C1W4_D	100	80	0	N3C1W4_D.BPP	100	100	40	N2C1W4_D.BPP	100	100	40
N4C1W4_E	100	100	80	N3C1W4_E.BPP	100	100	50	N2C1W4_E.BPP	100	100	100
N4C1W4_F	100	30	30	N3C1W4_F.BPP	100	100	80	N2C1W4_F.BPP	100	100	100
N4C1W4_G	100	100	30	N3C1W4_G.BPP	100	100	80	N2C1W4_G.BPP	100	100	80
N4C1W4_H	80	60	40	N3C1W4_H.BPP	100	100	60	N2C1W4_H.BPP	100	100	100
N4C1W4_I	100	100	60	N3C1W4_I.BPP	100	100	60	N2C1W4_I.BPP	100	100	100
N4C1W4_J	100	100	100	N3C1W4_J.BPP	100	100	80	N2C1W4_J.BPP	100	100	90
N4C1W4_K	70	70	0	N3C1W4_K.BPP	100	100	70	N2C1W4_K.BPP	100	100	100
N4C1W4_L	100	100	70	N3C1W4_L.BPP	100	100	60	N2C1W4_L.BPP	100	100	100
N4C1W4_M	100	60	100	N3C1W4_M.BPP	100	100	80	N2C1W4_M.BPP	100	100	80
N4C1W4_N	100	100	60	N3C1W4_N.BPP	100	100	50	N2C1W4_N.BPP	100	100	50
N4C1W4_O	100	100	10	N3C1W4_O.BPP	100	100	100	N2C1W4_O.BPP	100	100	100
N4C1W4_P	100	30	90	N3C1W4_P.BPP	100	100	70	N2C1W4_P.BPP	100	100	70
N4C1W4_Q	100	80	30	N3C1W4_Q.BPP	100	100	100	N2C1W4_Q.BPP	100	100	100
N4C1W4_R	100	100	70	N3C1W4_R.BPP	100	100	30	N2C1W4_R.BPP	100	100	100
N4C1W4_S	100	100	70	N3C1W4_S.BPP	100	100	100	N2C1W4_S.BPP	100	100	100
N4C1W4_T	100	100	40	N3C1W4_T.BPP	100	100	90	N2C1W4_T.BPP	100	100	100

Fig. 11, 12 and 13 show the convergence of the algorithm for version 1, version 2 and version 3 respectively. It is evident that the proposed algorithm converges faster as compared to the other two algorithms. From the results it can be concluded that the proposed version V3 requires less number of iterations as compared to the other two variants to get optimal values. On the other hand there is not much difference in the performance of V1 and V2. Fig. 14, 15 and 16 shows the box plot for three harmony search algorithm variants on three instances with 500 objects. The box plots show that the objective values of the proposed algorithm variation V3 lies in the range of approximately 0 to -50 whereas the range for the other two variants is approximately from 0 to -175. From this it can be concluded that solution vectors in harmony memory of the proposed algorithm are more converged.

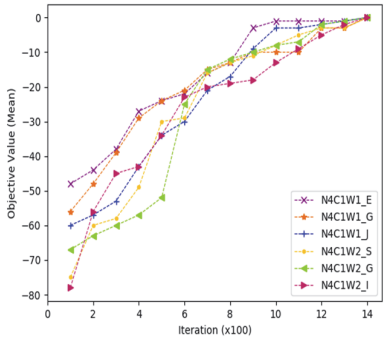

Fig. 11. Convergence of Version 1

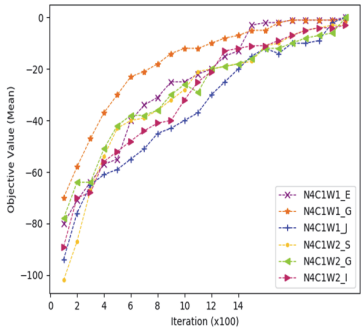

Fig. 12. Convergence of Version 2

Table 5 and 6 shows the function evaluations, acceleration rate and improvements of proposed harmony search algorithm w.r.t. other two versions. Sample results on 16 instances out which 6 instances with 500 objects and 10 instances with 200 objects is presented. Results show that the acceleration rate of proposed variation V3 is better than other two variants for instances with 500 objects. Acceleration rate of V3 and V2 is approximately similar for instances with 200 objects.

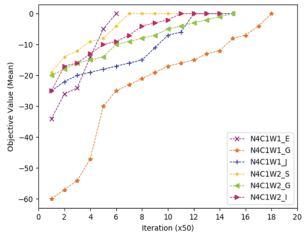

Fig. 13. Convergence of Version 3

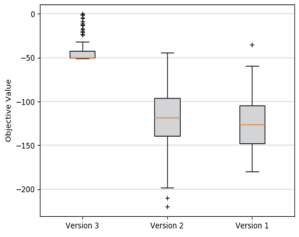

Fig. 14. Performance comparison of HS variations

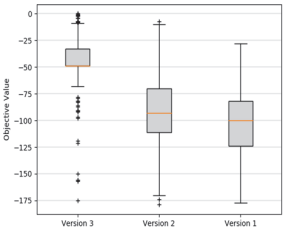

Fig. 15. Performance comparison of HS variations on input 2

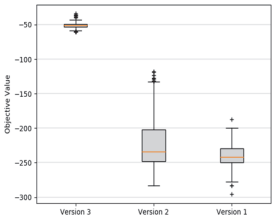

Fig. 16. Performance comparison of HS variations on input 3

Table 5

Acceleration Rate (AR) of Version 4 for sample datasets with 500 objects

Dataset with 500 objects	Function evaluations (NFEs)			Acceleration Rate (AR) of		Improvement in Version 3 (in %)	
	V3	V2	V1	V2	V1	wrt V3	wrt V2
N4C1W1_E	800	2800	3600	3.5	4.5	71.42	77.77
N4C1W1_G	600	2400	3800	4	6.33	75	84.21
N4C1W1_J	1000	1800	2800	1.8	2.8	44.44	64.28
N4C1W2_G	800	1600	2000	2	2.5	50	60
N4C1W2_I	800	1800	3000	2.25	3.75	55.55	73.33
N4C1W2_S	600	2600	2000	4.33	3.33	76.9	70

Table 6

Acceleration Rate (AR) of Version 4 for sample datasets with 200 objects

Dataset with 200 objects	Function evaluations (NFEs)			Acceleration Rate (AR) of Version 4 Vs		Improvement in Version 4 (in %)	
	V3	V2	V1	V2	V1	wrt V2	wrt V1
N3C1W1_A	200	200	600	1	3	0	66.66
N3C1W1_B	100	200	300	2	3	50	66.66
N3C1W1_C	100	200	200	2	2	50	50
N3C1W1_D	100	100	100	1	3	0	0
N3C1W1_E	100	200	200	2	2	50	50
N3C1W1_F	100	100	300	1	3	0	66.66
N3C1W1_G	100	200	300	2	3	50	66.66
N3C1W1_H	100	100	300	1	3	50	66.66
N3C1W1_I	100	100	400	1	4	0	75
N3C1W1_J	100	100	300	1	3	0	66.66

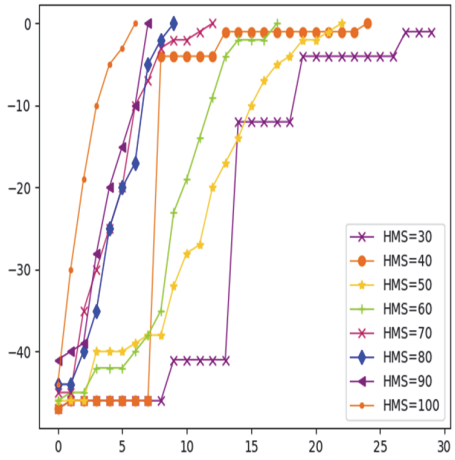

Fig. 17. Effect of HMS on performance of V3

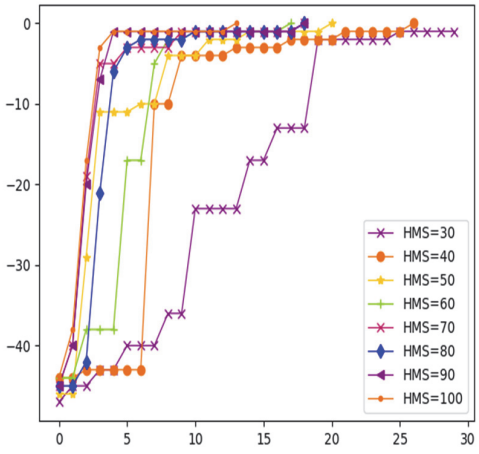

Fig. 18. Effect of HMS on performance of V2

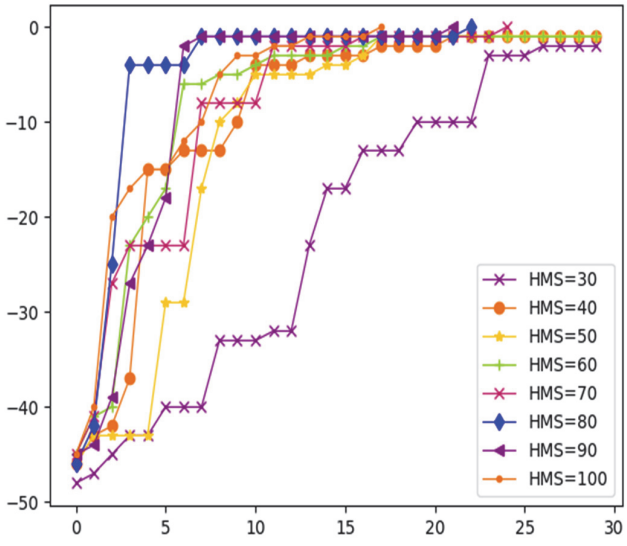

Fig. 19. Effect of HMS on performance of V1

The effect of harmony memory size on performance of the HS variations is tested with changing the size from 30 to 100. Fig. 17, 18 and 19 shows the results of HS variations with changing harmony memory size. Results show that all three versions of harmony search with problem specific representation, adaptive PAR, re-initialization strategy performs well with small harmony memory size.

5. Conclusions

Bin packing problem is a NP-hard combinatorial optimization problem and has several applications in the real world. Due to its large search space deterministic algorithms fail to provide optimal solutions. This paper presents a variant of HSA which is a meta-heuristic algorithm for solving 1D bin packing problem. HSA is also a widely used algorithm for solving various optimization problems. HSA has two parameters namely HMCR and PAR which controls the exploration and exploitation capability. In basic HSA these parameters are static and therefore fail to give optimal solutions for problems with higher complexity. In this paper a variant of HSA is proposed which uses adaptive PAR strategy and iterative local search which enhances the performance of the algorithm. Problem specific harmony memory representation, initialization and improvisation improved the results of BPP. The performance of the proposed HSA is evaluated on 180 benchmark datasets which consists of 100, 200 and 500 objects to be packed. In order to compare the performance of proposed HSA with other variants, metrics such as best, mean, standard deviation, success rate, acceleration rate are used. Here best is the most converged solution present in the harmony memory matrix. Version 3 is the proposed method, version 2 uses decreasing PAR strategy and version1 uses increasing PAR strategy Mahdavi et al. (2007). Performance of version 3 (proposed method) outperforms version 2 and version 1 in all the aspects. Also, the box plot presentation shows that values of the solution vector

in harmony memory for version 3 are far more converged than other two variants. An experiment was conducted to show the effect of harmony memory size on the performance of the algorithm.

In future there is a scope to investigate different strategies for controlling exploration and exploitation capabilities by tuning PAR and using hybridization approaches to obtain optimal solutions.

References

Adamuthe A., & Nitave T. (2020). Harmony search algorithm with adaptive parameter setting for solving large bin packing problems, *Decision Science Letters, 10*(2).

Al-Betar, M. A., Khader, A. T., & Liao, I. Y. (2010). A harmony search with multi-pitch adjusting rate for the university course timetabling. In Recent advances in Harmony search algorithm (pp. 147-161). Springer, Berlin, Heidelberg.

Alvim, A. C., Ribeiro, C. C., Glover, F., & Aloise, D. J. (2004). A hybrid improvement heuristic for the one-dimensional bin packing problem. *Journal of Heuristics, 10*(2), 205-229.

Baker, B. S., Coffman, Jr, E. G., & Rivest, R. L. (1980). Orthogonal packings in two dimensions. *SIAM Journal on computing, 9*(4), 846-855.

Blum, C., & Schmid, V. (2013). Solving the 2D bin packing problem by means of a hybrid evolutionary algorithm. *Procedia Computer Science, 18*, 899-908.

Chakraborty, P., Roy, G. G., Das, S., Jain, D., & Abraham, A. (2009). An improved harmony search algorithm with differential mutation operator. *Fundamenta Informaticae, 95*(4), 401-426.

Chen, J., Pan, Q. K., Wang, L., & Li, J. Q. (2012). A hybrid dynamic harmony search algorithm for identical parallel machines scheduling. *Engineering Optimization, 44*(2), 209-224.

Chung, F. R., Garey, M. R., & Johnson, D. S. (1982). On packing two-dimensional bins. *SIAM Journal on Algebraic Discrete Methods, 3*(1), 66-76.

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

Coffman, Jr, E. G., Garey, M. R., Johnson, D. S., & Tarjan, R. E. (1980). Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing, 9*(4), 808-826.

De Cauwer, M., Mehta, D., & O'Sullivan, B. (2016). The temporal bin packing problem: an application to workload management in data centres. *In 2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI) (pp. 157-164). IEEE*.

dos Santos Coelho, L., & de Andrade Bernert, D. L. (2009). An improved harmony search algorithm for synchronization of discrete-time chaotic systems. *Chaos, Solitons & Fractals, 41*(5), 2526-2532.

El-Abd, M. (2013). An improved global-best harmony search algorithm. *Applied Mathematics and Computation, 222*, 94-106.

Fleszar, K., & Hindi, K. S. (2002). New heuristics for one-dimensional bin-packing. *Computers & Operations Research, 29*(7), 821-839.

Frenk, J. G., & Galambos, G. (1987). Hybrid next-fit algorithm for the two-dimensional rectangle bin-packing problem. *Computing, 39*(3), 201-217.

Gary, M. R., & Johnson, D. S. (1979). Computers and Intractability: A Guide to the Theory of NP-completeness.(1979). WH Freman and Co.

Geem Z.W., Tseng CL., Park Y. (2005) Harmony search for generalized orienteering problem: best touring in China. In: Wang L., Chen K., Ong Y.S. (eds) Advances in Natural Computation. ICNC 2005. *Lecture Notes in Computer Science, vol 3612. Springer, Berlin, Heidelberg*.

Geem, Z. W., Kim, J. H., & Loganathan, G. V. (2001). A new heuristic optimization algorithm: harmony search. *Simulation, 76*(2), 60-68.

Guo, Z., Yang, H., Wang, S., Zhou, C., & Liu, X. (2018). Adaptive harmony search with best-based search strategy. *Soft Computing, 22*(4), 1335-1349.

Hasançebi, O., Erdal, F., & Saka, M. P. (2010). Adaptive harmony search method for structural optimization. *Journal of Structural Engineering, 136*(4), 419-431.

Hasanipanah, M., Keshtegar, B., Thai, D. K., & Troung, N. T. (2020). An ANN-adaptive dynamical harmony search algorithm to approximate the flyrock resulting from blasting. *Engineering with Computers*, 1-13..

Hopper, E., & Turton, B. (1999). A genetic algorithm for a 2D industrial packing problem. *Computers & Industrial Engineering, 37*(1-2), 375-378.

Kong, X., Gao, L., Ouyang, H., & Li, S. (2015). A simplified binary harmony search algorithm for large scale 0–1 knapsack problems. *Expert Systems with Applications, 42*(12), 5337-5355.

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

Lodi, A., Martello, S., & Vigo, D. (1999a). Approximation algorithms for the oriented two-dimensional bin packing problem. *European Journal of Operational Research, 112*(1), 158-166.

Lodi, A., Martello, S., & Vigo, D. (1999b). Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing, 11*(4), 345-357.

Lodi, A., Martello, S., & Vigo, D. (1999c). Neighborhood search algorithm for the guillotine non-oriented two-dimensional bin packing problem. In Meta-Heuristics (pp. 125-139). Springer, Boston, MA.

Mahdavi, M., Fesanghary, M., & Damangir, E. (2007). An improved harmony search algorithm for solving optimization problems. *Applied Mathematics and Computation, 188*(2), 1567-1579.

Manjarres, D., Landa-Torres, I., Gil-Lopez, S., Del Ser, J., Bilbao, M. N., Salcedo-Sanz, S., & Geem, Z. W. (2013). A survey on applications of the harmony search algorithm. *Engineering Applications of Artificial Intelligence, 26*(8), 1818-1831.

Moh'd Alia, O., & Mandava, R. (2011). The variants of the harmony search algorithm: an overview. *Artificial Intelligence Review, 36*(1), 49-68.

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

Omran, M. G., & Mahdavi, M. (2008). Global-best harmony search. *Applied Mathematics and Computation, 198*(2), 643-656.

Pandi, V.R., Panigrahi, B.K. (2011) Dynamic economic load dispatch using hybrid swarm intelligence based harmony search algorithm. *Expert Systems with Applications, 38*, 8509-8514. https://doi.org/10.1016/j.eswa.2011.01.050

Paquay, C., Schyns, M., & Limbourg, S. (2016). A mixed integer programming formulation for the three-dimensional bin packing problem deriving from an air cargo application. *International Transactions in Operational Research, 23*(1-2), 187-213.

Parreño, F., Alvarez-Valdés, R., Oliveira, J. F., & Tamarit, J. M. (2010). A hybrid GRASP/VND algorithm for two-and three-dimensional bin packing. *Annals of Operations Research, 179*(1), 203-220.

Perboli, G., Gobbato, L., & Perfetti, F. (2014). Packing problems in transportation and supply chain: new problems and trends. *Procedia-Social and Behavioral Sciences, 111*, 672-681.

Saka M.P., & Hasançebi O. (2009). *Adaptive Harmony Search Algorithm for Design Code Optimization of Steel Structures. In: Geem Z.W. (eds) Harmony Search Algorithms for Structural Design Optimization. Studies in Computational Intelligence, vol 239*. Springer, Berlin, Heidelberg

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

Schwerin, P., & Wäscher, G. (1997). The bin-packing problem: A problem generator and some numerical experiments with FFD packing and MTP. *International Transactions in Operational Research, 4*(5-6), 377-389.

Simchi-Levi, D. (1994). New worst-case results for the bin-packing problem. *Naval Research Logistics (NRL), 41*(4), 579-585.

Song, W., Xiao, Z., Chen, Q., & Luo, H. (2013). Adaptive resource provisioning for the cloud using online bin packing. *IEEE Transactions on Computers, 63*(11), 2647-2660.

Stützle, T. (1999). *Local search algorithms for combinatorial problems: analysis, improvements, and new applications*.

Suganthan, P. N., Hansen, N., Liang, J. J., Deb, K., Chen, Y. P., Auger, A., & Tiwari, S. (2005). Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. KanGAL report, 2005005(2005), 2005.

Taherinejad, N. (2009, August). Highly reliable harmony search algorithm. *In 2009 European Conference on Circuit Theory and Design (pp. 818-822). IEEE.*

Wang, C. M., & Huang, Y. F. (2010). Self-adaptive harmony search algorithm for optimization. *Expert Systems with Applications, 37*(4), 2826-2837.

Wang, L., Pan, Q. K., & Tasgetiren, M. F. (2011). A hybrid harmony search algorithm for the blocking permutation flow shop scheduling problem. *Computers & Industrial Engineering, 61*(1), 76-83.

Wong, L., & Lee, L. S. (2009). Heuristic placement routines for two-dimensional bin packing problem. *Journal of Mathematics and Statistics, 5*(4), 334.

Wu, B., Qian, C., Ni, W., & Fan, S. (2012). Hybrid harmony search and artificial bee colony algorithm for global optimization problems. *Computers & Mathematics with Applications, 64*(8), 2621-2634.

Yadav P., Kumar R., Panda S.K., Chang, C. S. (2012). An intelligent tuned harmony search algorithm for optimisation. *Information Sciences, 196*, 47-72.

Yang, X. S. (2009). *Harmony search as a metaheuristic algorithm. In Music-inspired harmony search algorithm* (pp. 1-14). Springer, Berlin, Heidelberg