J Grid Computing (2024) 22:49
https://doi.org/10.1007/s10723-024-09761-7

RESEARCH

A Hybrid Discrete Grey Wolf Optimization Algorithm Imbalance-ness Aware for Solving Two-dimensional Bin-packing Problems

Saeed Kosari · Mirsaeid Hosseini Shirvani · Navid Khaledian · Danial Javaheri

Received: 31 December 2023 / Accepted: 24 March 2024 / Published online: 10 May 2024

Abstract In different industries, there are miscellaneous applications that require multi-dimensional resources. These kinds of applications need all of the resource dimensions at the same time. Since the resources are typically scarce/expensive/pollutant, presenting an efficient resource allocation is a very favorable approach to reducing overall cost. On the other hand, the requirement of the applications on different dimensions of the resources is variable, usually, resource allocations have a high rate of wastage owing to the unpleasant resource skew-ness phenomenon. For instance, micro-service allocation in the Internet of Things (IoT) applications and Virtual Machine Placement (VMP) in a cloud context are challenging tasks because they diversely require imbalanced all resource dimensions such as CPU and Memory bandwidths, so inefficient resource allocation raises issues. In a special case, the problem under study associated with the two-dimensional resource allocation of distributed applications is modeled to the two-dimensional bin-packing problems which are categorized as the famous NP-Hard. Several approaches were proposed in the literature, but the majority of them are not aware of skew-ness and dimensional imbalances in the list of requested resources which incurs additional costs. To solve this combinatorial problem, a novel hybrid discrete gray wolf optimization algorithm (*HD-GWO*) is presented. It utilizes strong global search operators along with several novel walking-around procedures each of which is aware of resource dimensional skew-ness and explores discrete search space with efficient permutations. To verify *HD-GWO*, it was tested in miscellaneous conditions considering different correlation coefficients (*CC*) of resource dimensions. Simulation results prove that *HD-GWO* significantly outperforms other state-of-the-art in terms of relevant evaluation metrics along with a high potential of scalability.

Keywords Hybrid discrete grey wolf optimization algorithm · Two-dimensional bin-packing problem · Cloud computing · Resource allocation

S. Kosari
Institute of Computing Science and Technology, Guangzhou University, Guangzhou 510006, China

M. Hosseini Shirvani (✉)
Department of Computer Engineering, Sari Branch, Islamic Azad University, Sari, Iran
e-mail: mirsaeid_hosseini@iausari.ac.ir; mirsaeid_hosseini@yahoo.com

N. Khaledian
Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg, Esch-sur-Alzette, Luxembourg

D. Javaheri (✉)
Department of Computer Science and Engineering, Korea University, Seoul 02841, Republic of Korea
e-mail: javaheri@korea.ac.kr

1 Introduction

Resource management is a very important phase of the business process regarding each industry because it

indirectly impacts the system's performance and overall costs. Allocation of limited/expensive/pollutant resources to a variety of applications is a very intricate and sensitive task in different industries for the sake of performance improvement, overall expenditure reduction, and pollution as well. It is necessary to allocate resources efficiently to applications to reach different potentially conflicting objectives. For instance, in the information technology industry, virtualization technology enables each physical server to host multiple virtual machines (VMs) each of which runs one application relevant to users' requests [1–3]. Each physical server must supply two-dimensional resources that are CPU cycles and memory bandwidth at the same time for each requested VM associated with each application. Bear in mind that because of the different nature of applications, there is a high rate of skew-ness in resource vector requests. Inefficient resource allocations lead to additional usage of active servers that burdens more power costs and implicates environmental impacts [4]. Since a large part of the total cost of ownership (*TCO*) associated with distributed system owners such as cloud providers is owing to power consumption, utilizing the minimum number of active physical servers helps both *TCO* reduction and green computing objectives [5]. On the other hand, in distributed systems such as cloud data centers different VMs are periodically requested which may have a high degree of skew-ness in each resource dimension of the requested list of resources. So, inefficient resource allocation may lead high rate of resource wastage and high illogical usage of physical servers. To obviate the challenges, several approaches were proposed in the literature to solve the resource allocation issues which are abstracted to the two-dimensional bin-packing problems, but most of them do not pay attention to the imbalance-ness in resource requests for each VM. Therefore, it burdens more costs because of the high rate of resource wastage, underutilized servers, and using illogical additional servers [6]. This paper concentrates on the allocation of two-dimensional resources to applications that simultaneously need both resource dimensions. For instance, VM placement and server consolidation techniques are pervasively exploited in cloud data centers to lower down system's power consumption and resource dissipation [7, 8]. So, the novelty of the current paper is as follows:

- It models the two-dimensional resource allocation problem to a two-dimensional bin-packing problem.
- It proposes a total resource wastage model regarding each dimension to utilize it in the main algorithm for precluding additional illogical resource usage.
- It formulates the aforementioned packing problem to an integer linear programming (ILP) model which is an NP-Hard problem.
- To solve this combinatorial optimization problem, a hybrid discrete gray wolf optimization algorithm that engages some novel walking around procedures is proposed; the proposed walking around procedures are aware of dimensional skew-ness in which it efficiently permutes search space toward main objective functions. Moreover, it takes a tricky approach to evading from local optimum trap.

The focus of the current paper is to propose a novel algorithm for solving a especial two-dimensional bin-packing problem in which each bin is a two dimensional resource. All dimensions of a resource are multiplexed between different applications so the resource is allocated up to a determined threshold in each dimension. This paper formulates this problem as an integer linear programming problem (*ILPP*) which is NP-Hard. Since existing approaches are not commensurate with the discrete nature of search space and also are not aware of the imbalance-ness of requested resources regarding all dimensions, the novel hybrid discrete grey wolf optimization algorithm (*HD-GWO*) which takes benefit of different advantages is proposed. It utilizes several discrete walking-around heuristics used in the exploitation phase to improve exploration results. During the search process, it exploits special heuristics which are aware of resource imbalance-ness. Consequently, it efficiently permutes solutions so it reduces resource wastage and also leads to utilizing fewer physical servers or bins. It also applies the temperature and cooling concepts in the main body of the proposed algorithm similar to the simulated annealing (SA) algorithm to examine non-efficient solutions either opening new good solutions or running away from the local trap. To verify the proposal, it has been tested in different variable scenarios. The simulation results prove the superiority of the proposed *HD-GWO* against other state-of-the-art in terms of evaluation parameters. The rest of the paper is structured as below. Section 2 reviews related works and focus on existing gaps. Section 3 presents problem statement. Section 4 elaborates the proposed imbalance-ness-aware algorithm for solving

two-dimensional bin-packing problem. Section 5 evaluates the current paper in different scenarios. Finally, Section 6 concludes the paper and provides comments on future direction.

2 Related Works

Packing problems are very important issues in computation domains because of their wide variety of applications. For instance, one-dimensional, two-dimensional, and multi-dimensional bin-packing problems, and strip packing problems are some examples to cite; each of which has its own application in different industries. In this section, several packing problems are surveyed with the most concentration of the two-dimensional bin-packing problems. Finally, the gaps are outlined.

In steel industries, the cutting stock problem is modeled to the one-dimensional packing problem. Each item can be split into smaller pieces; then, they can be welded for recombination. Tanir et al. have proposed a heuristic algorithm to solve this packing problem with the aim of minimizing both trim loss and number of welds [9]. Munien and Ezugwu prepared a survey study on the applications of one-dimensional bin-packing problems with heuristic-based approaches [10]. The most applications are in cutting industries, transportation, warehousing, and supply chain management [10]. Another case of packing issue is the circle bin packing problem in which the items are circular. The circle items must be packed into multiple identical circle bins with the objective of minimizing the number of used circle bins [11]. To solve this problem, an adaptive simulated annealing algorithm which utilizes greedy search was proposed by Yaun et al. [11]. For the sake of the problem nature, the resource wastage rate is high in these kinds of packing problems. In production systems, packing and assembling are two prominent operations in the production process. Hao et al. proposed a hybrid algorithm to solve this configuration-dependent bin-packing problem in an aviation manufacturing factory [12]. One of the most two-dimensional packing issues is strip packing problems without guillotine constraints. It is used in cutting sheets of iron and steel industries with the aim of minimizing resource wastage [13]. An effective discrete gray wolf optimization algorithm with incorporating some strategies was suggested to solve this problem. One of the most important applications of two-dimensional bin-packing problems is in either VM placement or server consolidation of cloud and fog computing data centers [14–16]. Both of them aim to pack VMs into the minimum number of physical servers subject to some constraints. For instance, the sum of the requested CPU bandwidth and memory bandwidth of all co-hosted VMs must not exceed from available CPU and RAM capability of the host physical machine. An optimal VM placement-based GWO algorithm was proposed by Al-Moalmi et al. to minimize the number of active servers and power consumption as well [17]. This paper proposed a novel binary version of the GWO algorithm to solve the VMP problem. During the course of optimization, it utilizes the sigmoid function to generate valid vectors from the continuous domain. A multi-objective monarch butterfly algorithm was proposed for solving the VMP problem in a cloud computing environment by the number of used servers, power consumption, and maintenance cost minimization perspectives which is a version of the multi-bin packing problem [18]. A VMP algorithm based on multi-objective reinforcement learning was propounded to figure out VM deployment on cloud data centers with optimization on power consumption and resource wastage simultaneously. To improve the quality of non-dominated solutions, they applied the Chebyshev scalarization function in multi-objective reinforcement learning algorithms [19]. A reinforcement learning algorithm as a machine learning approach was added to the grey wolf optimizer to enhance engineering optimization problems [20]. Nasr et al. proposed a novel water pressure change optimization (WPCO) to solve the scheduling problem that pays for efficient resource allocation in cloud computing platforms [21]. They used the balancing degree (BD) factor to measure the amount of the system's load balancing. By using the BD metric, WPCO conducts its operators to schedule tasks and VMs to meet objectives and increase resource utilization. The same work has been published by Amer et al. [22] with multi-objective viewpoint. Since the multi-objective scheduling problem is NP-Hard, the authors proposed an elite learning Harris hawks optimizer (ELHHO) to solve multi-objective optimization problem with *makespan*, throughput, resource utilization, cost, etc. at the same time. Low time complexity heuristic traveler salesman approach for Cloudlet scheduling (TSACS) algorithm was proposed to solve Cloudlet scheduling problem in

cloud platforms with the aim of resource utilization and other objectives [23]. The proposed TSACS has three phases, namely, the clustering, converting, and assignment phases. In the clustering phase, it groups a number of Cloudlets in a cluster; then, in the converting phase, it uses a reduction algorithm to map a Cloudlet instance to Traveler Salesman Problem (TSP) instance; finally, in the assignment phase, it assigns a new instance to VMs in a data center to meet objectives. An evolutionary heuristic algorithm was used to solve multi-dimensional vector bin packing problem that is encountered in different industrial applications such as production planning, steel fabrication, and assignment of VMs onto physical hosts at cloud and fog data centers [24]. A primal decomposition algorithm was designed for a two-dimensional bin packing problem which needs packing a set of rectangular items into a minimum set of larger rectangular bins. In this classic bin packing, the items must be packed with their edges to be parallel to the borders of the bins, cannot be rotated and cannot overlap among them [25]. An adaptive heuristic algorithm was proposed to solve VM deployment for IoT applications that utilize cloud data centers [26]. The proposed heuristic considers four thresholds of CPU utilization parameters to classify requested resources for applications to reduce power consumption and service level agreement (SLA) violation rate. It is done by a customized *K*-means approach [26]. An energy-efficient VM cluster placement (EVCT) algorithm was extended in Software-defined Data Centers (SDDC) [27]. Specifically, the proposed heuristic is utilized in Connected and Autonomous Vehicle (CAV) as a large-scale IoT application. To this end, EVCT uses graph theory and abstracts the problem into the "maximum flows and minimum cut theory" issue. By applying the clustering method to VMs' similarity in resource usage, resource allocation is performed so that the power consumption costs are minimized and the requested Quality of Service (QoS) to users is met [27]. Zhou et al. proposed an adaptive three-threshold VM placement algorithm in cloud data centers [28]. This heuristic considers workload variation for VMs' deployment so that it minimizes total power consumption while it pays not to violate agreed SLA. It categorizes workloads into four-class hosts. Three predetermined thresholds specify the workload class. Accordingly, the proposed heuristic decides to deploy VMs for user workload to meet objectives. Cloud manufacturing is a new paradigm with unified manufacturing models such as ASP and MGrid; and enterprise information technologies under supporting cloud computing, IoT, and virtualization to deliver manufacturing services. To facilitate service delivery to enterprises, an intelligent energy consumption model using machine learning methods was presented in the literature to deliver green manufacturing services. To this end, prediction models are used to forecast near-exact resource usage and make service deployments based on that prediction model. The proposed intelligent model utilizes support vector machine, random forest, and Grid search algorithms [29]. One of the most applicable distributed systems that supports CPU-intensive and delay-sensitive IoT applications is Mobile Edge Computing (MEC). Resource management issues in such variable environments are very important in meeting users' required QoS and reducing overall costs. To obviate the challenges, a new edge intelligent energy modeling scheme mixing by Elman Neural Network (ENN) and feature selection of ML models was propounded to optimize the power consumption of edge servers [30]. This scheme considers load fluctuations and sorts tasks based on CPU-intensive, I/O-intensive, and online transaction-intensive; then, the resource allocation is done with the lowest estimation error. A modified genetic algorithm mixing greedy strategy (MGGS) approach was proposed in the literature to solve task scheduling problems in cloud computing platforms [31]. The proposed MGGS tries to find optimal solutions in the minimum execution time. It returns the scheduling solutions with higher system performance and user satisfaction. The proposed scheduling approach improves performance in terms of total completion time, average response time, and QoS parameters that the user experiences.

Some heuristic algorithms were published in literature to solve classic bin-packing problems with the lowest time complexity which are first fit decreasing (FFD), best fit decreasing (BFD), and worst fit decreasing (WFD) strategies. The time complexity is bounded to O(*nlogn*) that refers to only sorting cost [32]. Fatima et al. proposed virtual machine placement via bin packing in cloud datacenters by utilizing an enhanced levy-based particle swarm optimization algorithm with the incorporation of best fit strategy [33]. A virtual machine placement algorithm that combines the NSGAII algorithm and bin-packing heuristic was proposed by Wie et al. in applying multi-dimensional resources which

makes a balance between resource dimensions [34]. Table 1 is dedicated to reviewing literature with a comparative viewpoint.

Table 1 condenses key points of literature comparisons. It points out that less works concentrate on resource imbalance-ness which leads to additional resource usage incurring more costs. Moreover, the majority of the proposed algorithms are not scalable in larger search spaces which consequently reaches sub-optimal solutions. To fill the gap, the current proposal is aware of the resource skew-ness and suggests a discrete meta-heuristic incorporating some novel walking around procedures that balances exploration and exploitation in the course of the optimization process. The simulation results witness a significant improvement in packing items with the least dissipation amount in comparison with other existing state-of-the-art.

3 Problem Statement

The current paper focuses on a special kind of two-dimensional bin-packing problem that multiplexes resource dimensions among applications. The dedicated multiplexed resources cannot exceed from available resources in each dimension. One of its most common applications is in VM placement and server consolidation of cloud and fog data centers. The prominent being used resources are CPU, memory, and hard disk bandwidths. Since all physical servers uniformly access the storage attached networks (SANs), the reason why the vector (CPU, Memory) has been taken into consideration for a two-dimensional bin-packing problem. To formally state the problem, this section includes three sub-sections. Firstly, the resource wastage model is provided which is then utilized in proposed procedures to cautiously lower the number of used bins (resources). Secondly, the problem is formally stated. Thirdly, an illustrative example is brought to show how the proposal is effective. For the sake of ease of following, Table 2 provides a list of used symbols and notations for problem formulation.

3.1 Problem Formulation

We are given a set of two-dimensional n objects, $Objects = \{O_1, O_2, \ldots, O_n\}$; each object O_k has two specific attributes width (w_k) and height (h_k) in which the rotation of objects in any dimension is not permitted. So, the sets $W = \{w_k | k = 1, \ldots, n \text{ and } 0 < w_k \leq T_W\}$ and $H = \{h_k | k = 1, \ldots, n \text{ and } 0 < h_k \leq T_H\}$ are dedicated for all widths and heights respectively. Note that T_W and T_H are used for a *threshold* or ceiling of the first and second dimensions of associated resources respectively. For the sake of simplicity, all *threshold* values are normalized to 1. Moreover, there exists m number of bins; so, the set of homogeneous bins is $Bins = \{b_1, b_2, \ldots, i_m\}$ where $m \leq n$; each bin b_k is characterized (w_k,h_k) without rotate permission. It is a especial bi-packing problem in which resources in each dimension are multiplexed among applications. For instance, in the IT domain, the set of objects is a set of applications that request its own VM including CPU and Memory bandwidth requests as two prominent dimensions. In this regard, the bins are a set of physical servers that can host multiple VMs so that the sum of resources associated with all co-hosted VMs does not exceed from available physical resources in each dimension. Then, the smart scheduler in the broker of clouds can utilize this proposal to reduce resource wastage. The goal is to pack all objects in the minimum number of used bins to reduce overall expenditures because each additional resource incurs more costs for service providers. To formulate the problem, a variety of binary decision variables are used. The binary variable z_j is used to indicate whether j-th two-dimensional resource or a bin b_j is engaged or not. In this regard, another binary variable x_{kj} is used to indicate whether the k-th object (O_k) is packed to a bin b_j or not. Note that, each object must be packed only in one bin; the reason why a constraint is incorporated in Eq. (4). So, the main objective function is to minimize the sum of used bins which Eq. (1) shows subject to preserve limitations of Eq. (2) through Eq. (6).

$$
\begin{cases}
min\ \mathrm{F(z)} = \sum_{\mathrm{j=1}}^{\mathrm{m}} z_j & (1)\\
\text{Subject to :} & \\
\sum_{\mathrm{j=1}}^{\mathrm{m}} \sum_{\mathrm{k=1}}^{n} w_k \times \mathrm{x_{kj}} \leq T_W \times z_j & (2)\\
\sum_{\mathrm{j=1}}^{\mathrm{m}} \sum_{\mathrm{k=1}}^{n} h_k \times \mathrm{x_{kj}} \leq T_H \times z_j & (3)\\
\sum_{\mathrm{j=1}}^{\mathrm{m}} \sum_{\mathrm{k=1}}^{n} x_{kj} = 1 & (4)\\
z_j, x_{kj} \in \{0, 1\} & (5)\\
0 < w_k, h_k \leq 1,\ k = 1, \ldots, n; & (6)
\end{cases}
$$

Table 1 A comparative literature review

Author(s)/ Ref	Problem	Application	Class	Algorithm	Merit	Limitation
Tanir et al. [9]	One-dimensional	Cutting stock in steel industry	Heuristic	Dynamic programming	It quickly provides a solution	Although it has low time complexity, in most large cases it returns sub-optimal solutions
Yaun et al. [11]	Circle packing	Cylinder packing and logistic industry	Hybrid meta-Heuristic	Simulated annealing and greedy search	Its operators are conducted in such a way as to avoid the local traps	It is not scalable, the reason why it returns poor results in large search space
Hao et al. [12]	One-dimensional	Aviation manufacturing factory	Hybrid Heuristic	Bi-level dynamic programming	Before it plummets into the main algorithm it calculates the lower and upper bounds of the solution	It cannot amend the worst solution during the optimization process; for this, it leads to sub-optimal in large-scale problems
Wang et al. [13]	Two-dimensional	Iron and steel production systems	Meta-Heuristic	Single discrete GWO	It improves the BFD strategy by utilizing the GWO algorithm	Since it engages limited exploration operators, it cannot efficiently permute discrete search space
Al-Moalmi et al. [17]	Two-dimensional	VM placement in cloud for IT industry	Meta-Heuristic	GWO	It utilizes the sigmoid function to discretize continuous value to valid arrays	Since it exploits limited searching operators, a big portion of search space remains unexplored which suffers from early convergence
Ghetas [18]	Multi-dimensional vector packing	Server consolidation in cloud for IT industry	Hybrid Meta-Heuristic	Monarch butterfly optimization	It works similarly to the cuckoo search algorithm which utilizes the levy flight concept to explore search space uniformly	The proposal is not aware of skew-ness in requested resources the reason why it does not return efficient solutions when the rate of skew-ness in requested resources very is high
Qin et al. [19]	two-dimensional	VM placement in cloud for IT industry	Heuristic	Reinforcement learning	It utilizes the Chebyshev function to weight selection for each objective function to prepare the scalarization function	It also is not aware of resource imbalance-ness which leading utilizing more resources

Table 1 (continued)

Author(s)/ Ref	Problem	Application	Class	Algorithm	Merit	Limitation
Fatima et al. [33]	Two-dimensional	Server consolidation in cloud for IT industry	Meta-Heuristic	Levy-based PSO	It utilizes the levy flight concept to explore search space uniformly	The quality of discrete levy flight is not similar to continuous levy flight. To obviate the problem, devising more efficient operators are necessary
Wie et al. [34]	Multi-dimensional	Server consolidation in cloud for IT industry	Hybrid Meta-Heuristic	NSGAII and bin-packing heuristic	It provides balanced index to preclude unbalanced resources	It suffers from early convergence because it does not concentrate on exploration and exploitation tuning in the course of optimization
Zhou et al. [26]	Multi-dimensional	IoT application	Heuristic	K-means & AFED-EF	It considers different thresholds and load fluctuations; then, the VM deployment leads to better performance	It can be improved by re-deployment some VMs in case of resource imbalance-ness
Zhou et al. [27]	Multi-dimensional	Connected & Autonomous Vehicles (CAV)	Heuristic	Clustering & Graph Theory	It utilizes VMs' affinity and makes efficient cluster of VMs for applications which need same resources	It seems it is well-suited for applications which have seasonal resource usage pattern
Zhou et al. [31]	Two-dimensional	Task Scheduling in Cloud Computing	Hybrid Meta-heuristic	Modified GA mixing Greedy Strategy (MGGS)	It adds greedy strategy to GA algorithm to reach optimal solutions in short time	Since it is customized for task scheduling problems, it needs further operators for other kinds of applications that require multi-dimensional resources

Table 2 The nomenclature for used symbols and notations

Symbols	Description
n	Number of objects (VMs)
m	Number of bins (servers)
w_k	Width of $k-th$ object (CPU_k) processing requirement of (VM_k)
x_{kj}	The binary decision variable indicating $k-th$ object is assigned to $j-th$ bin
C_1	The coefficient indicating the importance of the first dimension of the required resource
$Dis(b_k)$	The resource dissipation of bin b_k(physical machine: PM_k)
$S_W^{b_k}$	Sum of the used first dimension (W like CPU) of the resource b_k (physical machine PM_k) by different objects (different VMs)
T_0	Initial temperature
ε	Balancing parameter (small value)
T_W	Normalized threshold for the first dimension of a resource
$MaxIteration$	Maximum iterations
$Freeze$	Freeze temperature
O_i	$i-th$ object (VM_i)
b_j	$j-th$ bin (PM_j)
h_k	Height of $k-th$ object (Mem_k) memory requirement of (VM_k)
Z_j	The binary decision variable indicating $j-th$ bin is in use
C_2	The coefficient indicating the importance of the second dimension of the required resource
P_k	Power consumption of bin b_k (physical machine: PM_k)
$S_H^{b_k}$	Sum of the used second dimension (H like RAM) of the resource b_k (physical machine PM_k) by different objects (different VMs)
ΔT	Temperature gradual decrement
ToD	Total dissipation of all bins
T_H	Normalized threshold for the second dimension of a resource
$MaxProcess$	Maximum processes in each temperature
$PopSize$	Wolves population size

In the cloud industry, the goal is to minimize the number of active servers at the same time the resources are available for applications that request VM resources. In such a way, the operational cost including power consumption cost is controlled and residual physical machines are set to hibernate mode to save energy. Note that the first and the second dimensions are CPU (known as weight) and Memory (known as height) bandwidths in terms of Million Instructions per Second (MILPS) and Giga Byte (GB) respectively which must be multiplexed among VMs. Equation (1) is an optimization model which is an Integer Linear Programming (ILP) model; it is computationally NP-Hard. The solution must be returned provided some constraints to be considered. The constraints are used to show the resource limitation for serving the services. Since each bin can serve to applications up to its available resources, the sum of co-hosted items in a bin must not exceed from bin's capacity in each dimension. So, Eq. (2) indicates that the sum of the width of all co-hosted items is limited to the *threshold* dedicated for width threshold (T_W). In the same manner, Eq. (3) is used to indicate that the sum of the height of all co-hosted items is limited to the *threshold* dedicated for height threshold (H_W). For instance, the sum of used memory associated with all co-hosted VMs on a server must not exceed more than that server's memory capacity. Equation (4) determines that each item should only be assigned to one bin which means that one VM must be deployed on one server. Binary decision variables Z_j and x_{kj} are applied to show active used bin b_j and assigning the item O_k to bin b_j respectively. Equation (5) indicates that both variables are binary integers. In addition, Eq. (6) implies that input vectors, namely, w_k and h_k are real normalized values because CPU and memory capacities

and their units differ. For the sake of simplicity, their values are normalized to 1.

3.2 Resource Dissipation Model

Since the proposed algorithm is aware of resource imbalance-ness, in this section the novel resource wastage model is presented which takes both resource dimensions into account along with their importance in the whole system. To this end, the weight coefficients C_1 and C_2 are incorporated for the first and the second dimension's importance respectively where $0 < C_1, C_2 \leq 1$ and $C_1 + C_2 = 1$. A bin b_j with maximum capacity T_W and T_H respectively in the first and second dimensions that co-host some objects may suffer from resource wastage. Note that in many distributed computing systems, there are CPU-intensive applications whereas the counterparts are memory-intensive applications. For the first case, the designer can consider a bigger weight for the coefficient associated with the CPU dimension of the used server (bins) as a two-dimensional resource. For now, both importance is the same; so, the coefficients C_1 and C_2 are considered 0.5. The resource dissipation model of bin b_k, $Dis(b_k)$, is presented by Eq. (7) regarding all available dimensions. The term ε is a small value as a balancing parameter. In this paper, it is taken 0.0001.

$$Dis(b_k) = \frac{\left| C_1.\left(T_W - S_W^{b_k}\right) - C_2.(T_H - S_H^{b_k}) \right| + \varepsilon}{C_1 \times S_W^{b_k} + C_2 \times S_H^{b_k}} \tag{7}$$

In Eq. (7), the terms $S_W^{b_k}$ and $S_H^{b_k}$ are used to show the sum of occupied resources of the first and second dimensions of bin b_k by co-hosted objects respectively. The values of $S_W^{b_k}$ and $S_H^{b_k}$ are calculated by Eqs. (8) and (9) respectively.

$$S_W^{b_k} = \sum_{i=1}^{n} w_i \times x_{ik} \tag{8}$$

$$S_H^{b_k} = \sum_{i=1}^{n} h_i \times x_{ik} \tag{9}$$

The intention of suggesting the dissipation model is to apply it in the proposed algorithm to lower down resource wastage. To do so, the efficient algorithm is presented to permute search space efficiently following the proposed dissipation model. The next subsection proves the effectiveness of the proposal illustratively.

3.3 An Illustrative Example

Take a data center containing ample homogeneous HP ProLiant servers each of which has 1800 MIPS and 12 GB as CPU and main memory capacity; each server can multiplex its available resources between VMs requested for applications owing to virtualization technology. Set of resource requests for applications (objects) is $Apps = \{App_1(CPU = 900\ MIPS, RAM = 4GB), App_2(CPU = 600\ MIPS, RAM = 1.2GB), App_3(CPU = 600\ MIPS, RAM = 6GB), App_4(CPU = 180\ MIPS, RAM = 4GB), \ldots, App_n\}$. For now, there are four applications that request some two-dimensional resources. For the sake of normalization, the 1800 MIPS and 12 GB are normalized to 1 in each dimension. So, the set of normalized items are $Objects = \{O_1(App_1), O_2(App_2), O_3(App_3), O_4(App_4) \ldots, i_n\}$; also, set of width as for the first dimension is $W = \{w_1 = 0.5, w_2 = 0.3, w_3 = 0.3, w_4 = 0.1, \ldots\}$ and set of height as for the second dimension is $H = \{h_1 = 0.3, h_2 = 0.1, h_3 = 0.5, h_4 = 0.3, \ldots\}$. An inefficient resource allocation scheme that is not aware of imbalance-ness in the resource request list assigns resources to applications with a high wastage rate as Fig. 1 depicts.

According to Eq. (7), the resource dissipation for the first and second bins are 33% and 33% respectively which totally is near to 66% of a full server (bin). In this case, the fifth application which needs 600 MIPS and 3GB as CPU and memory bandwidth inevitably launches the third additional physical server (third bin) to host a new application (object) that incurs more additional cost because the new request cannot be met by the first server nor the second one owing to skewed resource assignment. On the other hand, the proposed imbalance-ness aware scheme permutes search space to lower down resource dissipation to open new room for adopting the fifth application without applying additional bins. Figure 2 shows the discrete permutation before co-hosting the fifth application.

This permutation lowers all resource dissipation from 66% (of Fig. 1) to 0% which can host newcomer applications (objects) in the second servers

(bins) without launching additional servers. The normalized resource request for App_5 is a vector ($w_5 = 0.3, h_5 = 0.4$) that can be co-hosted with App_2 and App_4 on the second server. Figure 3 illustrates all co-hosted applications on two physical servers.

The dissipation model is utilized in the proposed scheme which cautiously permutes discrete search space to pack objects as much as possible and lowers the resource wastage; so, it potentially reduces used bins in the larger number of requests.

4 Proposed Imbalance-ness-aware Algorithm for Solving Two-dimensional Bin-packing Problems

One of the most successful meta-heuristic algorithms that solves scientific problems is the grey wolf optimization algorithm (GWO) [35]. Each wolf is in a herd of wolves that is an agent or representative of a candidate solution. The democratic behavior is governed by the social wolves' population. The trajectory, assimilation toward an optimal solution, of a wolf is conducted by the position of three experienced wolves in the population. The three individuals are the first best wolf known as alpha (W_α), the second best wolf known as (W_β), and the third best wolf that is (W_δ). The rest wolves are named omega wolves (W_π) that are followers. They coordinate their path based on the position of three conductor wolves. Since the original GWO was designed for continuous optimization problems, the position changes of each wolf by reproduction operators do not necessarily yield efficient solutions in discrete optimization problems. The reason why this proposed discrete version presents several beneficial discrete operators to permute discrete search space uniformly and efficiently. The profound study of meta-heuristic approaches reveals that there is not a comprehensive algorithm to solve all kinds of NP-Hard problems. For instance, the Hill climbing and SA algorithms have local search trends; so, they seldom examine the far distant areas. They are recently being used as auxiliary algorithms in the hybrid algorithms to locally improve the global search results [36,

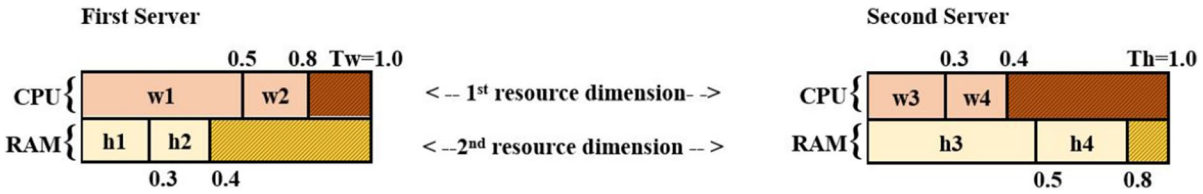

Fig. 1 Inefficient resource allocation scheme

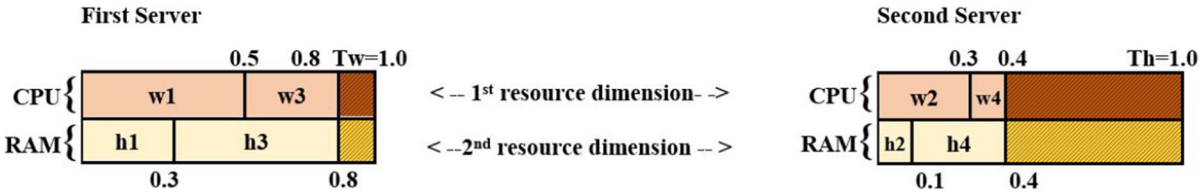

Fig. 2 Resource allocation imbalance-ness aware scheme

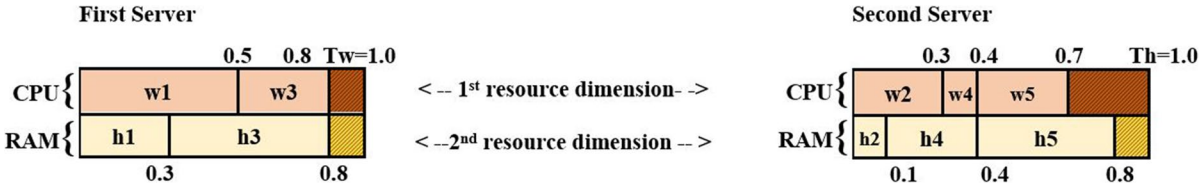

Fig. 3 Proposed imbalance-ness aware resource allocation scheme

37]. The genetic algorithm (GA) has limited operators for exploration also it is endangered to get stuck in the local optimal trap also it utilizes mutation. For another example, the particle swarm optimization (PSO) algorithm suffer from early convergence; also, the trajectory of the swarm is conducted by one experienced leader who has limited experience. The art is to devise a customized hybrid algorithm that heirs all existing plus points and avoids drawbacks. To this end, this proposed algorithm even examines worse solutions for a short time similar to the simulated annealing (SA) algorithm to run away from getting stuck in a local trap [38]. To strengthen the exploration, different exploring operators are presented. To keep random behavior, each of which is randomly called to permute search space uniformly. To make a balance in exploration and exploitation, a customized Hill-climbing algorithm is randomly incorporated to potentially improve gained results. Since the suggested algorithm works similarly to SA, the Hill-climbing tends to either uphill or downhill movement. The customized Hill-climbing algorithm calls one of the five possible new walking-around hill positions whether uphill or downhill. To reach a global solution, the bad trajectory may potentially lead to a good solution. This bad trajectory is examined even for a short time similar to the SA algorithm. This trick is implemented by the temperature concept in the annealing process. As the permutation operators may disperse objects in possibly more bins, finding a denser solution is possible. So, before the exploitation is ended the integration of solutions is done. For each solution, the algorithm finds the sparse source bin and the target bin (being used bin) that has ample capacity in both dimensions to adopt all of the objects of the source bin; then, the number of used bins is decreased by releasing the source bin. The simulation results prove all claims. To illustrate the big picture of the proposed hybrid approach, the Fig. 4 as a block diagram elaborates the details of the current suggested algorithm schematically.

Similar to each swarm-based optimization algorithm, it starts with a bunch of initial population of wolves each of which is an agent (a wolf) or candidate solution. The swarm of wolves has a *PopSize* number of individuals (wolves). Firstly, the two fitness values are calculated for each wolf. The first value counts the number of packed bins. The second value indicates the amount of accumulated wastage of used bins which is a clue for the next rounds in the proposed algorithm. Since the solution with the lowest first fitness value is the best, the second value is used for the comparison of two solutions that give the same first value. In the case of the same first fitness values of two solutions, the solution with the lowest accumulated wastage value is the better solution because it potentially uses fewer bins in the future. The whole algorithm is iterated until the termination criteria are met. The main loop follows the three phases. Firstly, it passes exploration by updating wolves in regard to three best wolves' positions. Then, the exploitation phase is started. In this phase, each solution examines Hill-climbing approaches whether upward or downward. If it passes the better way and meets a better solution (uphill), it definitely accepts it; otherwise, it probability accepts a bad solution (downhill). In the early stage, the chance of accepting the bad solution is high whereas in the late stage of algorithm, the chance is near zero. In other words, it works similarly to another evolutionary algorithm at the end of algorithm processing when it becomes familiar with the search space. This concept is done by the temperature definition. Note that, in each temperature, the algorithm searches several times to find a stable position; the stable point is a thermodynamic concept when the metal is near to freeze [39]. In the third phase, the complementary integration procedure is performed to possibly return a denser solution.

4.1 Basic Concepts (Encoding, Wolf, Fitness, and Termination)

One important process in meta-heuristics is how to encode the solution agents. If n requested applications (known as objects) to be placed in m number of two-dimensional available bins is received, the objective is to pack objects into the minimum number of bins so the overall cost is minimized. To encode each solution, the integer numbers in [1..m] are selected as genes where m is the number of available bins. The number of genes makes a solution with the length of n which is the number of objects [14, 40]. For example, 10 two-dimensional objects (applications) must be packed in 5 available two-dimensional bins (resources). Then, the length of the agent is 10 and gene values are selected from [1..5]. Figure 5 draws a valid encoding of a candidate solution. It means the objects O_2, O_4, O_7, and O_8 are placed on bin b_1 etc. In other words, each wolf W[i] from a swarm of wolves is encoded to a vector (4,1,2,1,4,4,1,1,3,3).

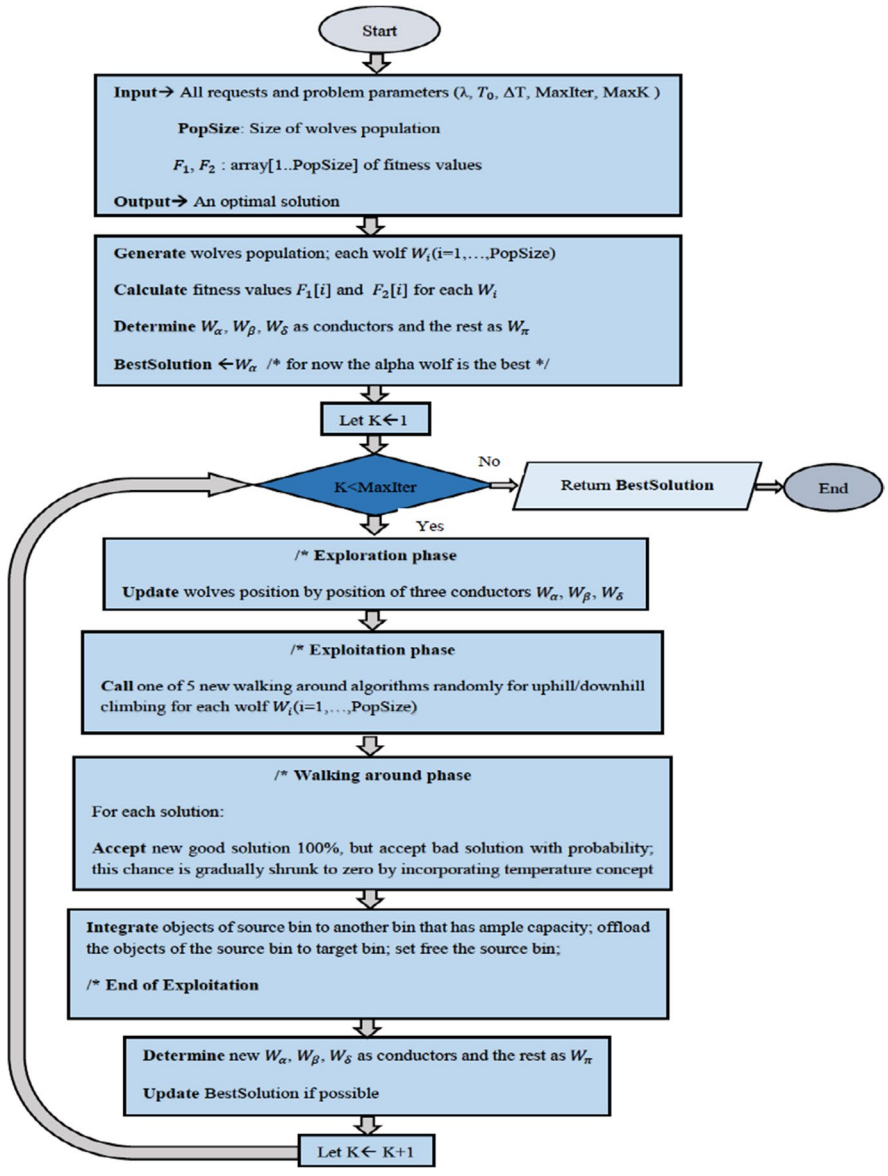

Fig. 4 Block diagram of the proposed hybrid algorithm

In this paper, for each solution, two values are calculated as fitness values. The first value counts the number of used bins whereas the second one is relevant to the total dissipation (*ToD*). The second value is used when a comparison of two solutions gives the same number of used bins. For the encoded wolf depicted in Fig. 5, the first fitness value is 4 bins whereas the second value calculated by Eq. (10) that is the accumulated dissipation of resources of used bins.

$$ToD = \sum_{k=1}^{m} Dis(b_k) \times x_k \tag{10}$$

If the k-th bin is used, the binary decision variable x_k is set to 1. The term $Dis(b_k)$ calculated by Eq. (7) is

Object	1	2	3	4	5	6	7	8	9	10
Bin	4	1	2	1	4	4	1	1	3	3

Fig. 5 A candidate encoded wolf

applied to measure the overall resource dissipation of *k*-th bin. The algorithm is stopped once the *MaxIteration* round is reached. The maximum iteration rounds are experimentally set.

4.2 Description of the Proposed *HD-GWO* Algorithm

This subsection pays to the description of the proposed HD-GWO algorithm for solving the famous NP-Hard two-dimensional bin-packing problem. As Algorithm 1 depicts, the proposed algorithm starts with the generation of the number of valid wolves each of which is an agent or a candidate solution. Then, two fitness values are calculated for each solution. The first value counts the used bins for packing objects whereas the second value calculates the amount of total resource dissipation in regard to two dimensions of available resources. Once the comparison of two solutions is needed, the solution that is better in the first fitness value is the dominant solution otherwise the better solution is selected based on the second fitness value. Since the proposed algorithm is imbalance-ness-aware, the second fitness value is used as a clue to preclude additional resource usage in future rounds. A meta-heuristic algorithm that works based on a wolf optimization algorithm needs to determine three conductors in which the follower wolves adjust their trajectory in accordance with the conductors' position and their experiences. However, the leader wolves are changed in the course of the algorithm's iterations. In the primary stage, the three leaders known as alpha, beta, and delta wolves are specified based on the current fitness values; then, the best so far solution is determined in the *BestSolution* variable which has undergone changes within the course of the algorithm. The main loop of the algorithm is between lines 6 to 61. It is iterated until no improvement is achieved. In each iteration of the main loop, it covers three phases. The first phase is for exploration that is an assimilation of encircling the prey in canonical of GWO [35]. In the second phase, it applies exploitation to potentially improve the gained solutions of the exploration phase similar to attack. In the third phase as a complementary phase, the integration procedure is called to possibly dense the disperse solution. One of the most intricate points which makes the novelty of the proposed hybrid algorithm brilliant is to utilize efficient local search. The novel diverse exploitation procedures, walking around procedures, are designed each of which is randomly called to traverse search space uniformly after a global search; then, Algorithm 3 is called in the third stage to integrate the current solution so the exploitation is ended. In other words, once the walking-around procedure is called; then, Algorithm 3 is called to deliver a possibly denser solution which reduces problem's cost. To this end, Algorithm 3 utilizes a circular queue as a data structure for finding sparse bins to integrate so possibly reducing the number of used bins. As the majority of meta-heuristic algorithms are endangered to get stuck in the local optimum, the walking-around procedures are called in such a way similar to the simulated annealing (SA) algorithm to get rid of the local optimum trap [39]. In other words, it definitely accepts good solutions resulting the new changes by incorporating calling one of the exploitation procedures. As the problem has a minimization trend, it is called downhill inclination. On the other hand, it accepts a worse solution with the probability; this probability is high at the early stage and is near zero at the end stage of the algorithm running. Therefore, it is called uphill inclination because a worse solution is selected. To avoid aggressive worse solution acceptance, the exponential function $e^{\frac{-\Delta F}{T}}$ is utilized. The term ΔF is the difference between two solutions belonging to two consecutive generations and the term T is used for the temperature concept similar to SA. The algorithm starts with the high temperature near to melt point when the molecule can freely move everywhere it wants. Then, it gradually cools down to reach the freezing temperature where the solution finds its stable point [39]. In each temperature, the *max* process (*MaxProcess* variable in Algorithm 1) is done to help for finding the stable point for each degree. To compare solutions for the acceptance stage, the discrepancy between the first fitness values is considered. If the new solution is better in terms of used bins, it definitely is accepted because of strongly downhill walking around. If the first fitness values of two comparative solutions are the same, the second fitness value is in line. In this case, the solution with less dissipation value is a better solution. It is a gentle downhill walking around. Otherwise, the solution with more used bins (worse solution) is selected with probability which may have a high acceptance rate in the first stage of running the algorithm but is near zero in the end of the algorithm. The best so far solution is returned as the final solution after the termination criteria are met.

Algorithm 1 HD-GWO procedure;

Algorithm 1. HD-GWO procedure;

Input
PopSize : Size of the swarm of wolves population;
n : Number of objects;
m : Number of available bins;
W[1.. *PopSize*] : Population of individual wolves;
F_1[1..PopSize] and F_2[1..PopSize] : The first and second fitness values;
(* first fitness indicates number of used bins the second is used for resource wastage;
Note that, the second value is used once the two first values are the same *)
Set of Parameters ={ T_0, λ, MaxK, *n, m, ΔT, MaxIteration, Freeze* };
T_k[1..PopSize] : temperature of wolves in *k*-th round;
Process: the stage of cooling phase in each temperature;
MaxProcess: the maximum defined process to reach stable point;

Output
An optimal bin-packing solution

1: **Generate** an initial population of wolves; each wolf is *W*[i] where i=1,…, *PopSize*
(* each wolf W[i] is encoded by (w_{i1}, w_{i2},…, w_{in}) where w_{ij} ∈ [1..*m*], $1 \le j \le n$ *)
2: **Calculate** fitness values F_1[i] and F_2[i] for wolf *W*[i] where i=1,…, *PopSize*
3: **Select** three best wolves W_α, W_β, and W_δ; the rest wolves are named omega wolves (W_π) that are followers.
4: **BestSolution** ≻ W_α,
5: *K*=1;
6: **while** *K* < *MaxIteration* **do**
7: /* start of exploration */
8: **For** i=1 To *PopSize* **do**
9: Call **Algorithm 2** for the sake of encircling the prey (* update the position of W[i] towards the prey *)
10: **/* in the next step, two fitness values is considered for each solution */**
11: $F_{11} = F_1$(W[i]) for the first fitness value; $F_{12} = F_2$(W[i]) for the second fitness value;
12: **if** Fitness(W[i]) is better than BestSolution **then**
13: **BestSolution** ≻ W[i]
14: **end-if**
15: /* start of exploitation */
16: T_k[i] ≻ T_0 (* high temperature *)
17: Process ≻ 0;
18: **while** T_k[i] > Freeze **do**
19: Process ≻ Process+1; /* start Walking Around */

As mentioned earlier, Algorithm 2 is designed for exploration in Algorithm 1. In line 9 of Algorithm 1, Algorithm 2 is invoked for each wolf in the main iteration. Algorithm 2 is used for updating the position of each solution by the concept of encircling the prey; this part is used for the exploration of the search space. Since the problem is discrete in nature, the proposed algorithm is a discrete version of a customized GWO algorithm. Each wolf changes its trajectory by looking at the positions of the three leader wolves. To do so, in the encoded solution each gene of a changing solution is derived from the genes of leaders; it is done by determining the probability PB_i made by the fitness of leaders. The fitness is better, and the chance of that leader's gene is higher to be selected. After this changes, the probable impaired solution is corrected by the Check&Correct(.) procedure in line 10. This procedure works similar to Algorithm 3 which utilizes a circular queue and offloads overloaded bins to the available used bins that have

```
20:          Draw an integer number q in ~U [1..5] for Walking Around procedure
21:          if R=1 then Call UDHC1 procedure for W'[i] = UDHC1(W[i]) ;
22:          elseif if R=2 then Call UDHC2 procedure for W'[i] = UDHC2(W[i]) ;
23:          elseif if R=3 then Call UDHC3 procedure for W'[i] = UDHC3(W[i]) ;
24:          elseif if R=4 then Call UDHC4 procedure for W'[i] = UDHC4(W[i]) ;
25:          else (* R=5 *) then Call UDHC5 procedure for W'[i] = UDHC5(W[i]) ;
26:          end-if;
27:          F21 = F1(W'[i]);
28:          ΔF= F21-F11
29:          if ΔF < 0 then
30:            /* Accept new better solution, it goes downhill strongly;
31:            W[i] ≻ W'[i]
32:          else if ΔF==0 then   /* number of used bins are equal
33:            F12 = ToD(W[i]) based on Eq. (10) as the second fitness; (* it means focus on wastage *)
34:            F22 = ToD(W'[i]) based on Eq. (10) as the second fitness; (* it means focus on wastage *)
35:            ΔD= F22-F12 ;
36:            if ΔD <= 0 then
37:              /* Accept the new better solution, it goes downhill gently;
38:              W[i] ≻ W'[i] ;
39:            else
40:              λ1 ≻ Draw a real number in ~ U(0,1);
41:              if e^(-ΔD/TK[i]) > λ1 then /* Accept worse solution with probability, it goes uphill;
42:                W[i] ≻ W'[i];
43:              end-if
44:            end-if
45:          else
46:            λ2 ≻ Draw a real number in ~ U(0,1)
47:            if e^(-ΔF/TK[i]) > λ2 then /* Accept worse solution with probability, it goes uphill;
48:              W[i] ≻ W'[i]
49:            end-if
50:          end-if
51:        if Process > MaxProcess then
52:          TK[i] ≻ TK[i] - ΔT;
53:          K ≻ K +1;
54:          Process ≻ 0;
55:        end-if
56:        end-while-Freeze
57:        Call Algorithm 3 for Integration (W[i])
58:        /* End of Exploitation Phase
59:    end-For{i}
60:    K≻K+1;
61: end-while-K
62: return BestSolution
63: End {Algorithm 1}
```

Algorithm 1 (continued)

ample resources, otherwise, it opens a new bin to pack residual object(s). Therefore, its time complexity is $O(m^2)$.

In line 9 of Algorithm 1, for each candidate solution, Algorithm 2 a metaphor for encircling of GWO is called to change the position of the current solution. To

Algorithm 2. Encircling the prey procedure;

Input:
W[i], W_α, W_β, W_δ : Wolf; (* each wolf W=(w_{i1}, w_{i2},…, w_{in}) *);
n : number of objects;
m : number of bins;
Output:
A new updated wolf W[i];

(* the trajectory of each wolf is conducted by three leaders W_α, W_β, and W_δ *)
1: Let F_1=Fitness (W_α), F_2=Fitness (W_β), and F_3=Fitness (W_δ)
(* $F_1 \le F_2 \le F_3$ *)
2: Let $PB_1=\frac{F_1}{F_1+F_2+F_3}$, $PB_2=\frac{F_2}{F_1+F_2+F_3}$, and $PB_3=\frac{F_3}{F_1+F_2+F_3}$
(* $PB_1 \le PB_2 \le PB_3$ and $PB_1 + PB_2 + PB_3 = 1$ *)
3: **For** j=1 To n **Do** (* n is the encoding size *)
4: Draw a real value q *in* ~ U(0,1)
5: **if** $q < PB_1$ **Then**
6: W[i].j=w_{ij} ⇝ W_α.j;
7: **elseif** $q < PB_2$ **Then**
8: W[i].j=w_{ij} ⇝ W_β.j;
9: **else**
10: W[i].j=w_{ij} ⇝ W_δ.j;
11: **end-if**
12: **End-For**
13: **Check&Correct** (W[i]);
14: return W[i]
15: **End** {**Algorithm 2**}

change the position of the current solution (input wolf), Algorithm 2 uses three leader wolves as conductors. The new probabilities for each conductor wolf are calculated by incorporating their fitness values in line 2 of Algorithm 2. Then, the position of each encoded wolf, for next movement, is determined by each part of the conductor's body with a probability. After the changes are done by calling Algorithm 2, potentially impaired solutions are amended by calling the Check&Correct(.) procedure. Therefore, the best solution so far is updated in line 13 of Algorithm 1. Then, the exploitation phase is started. For each solution (each wolf) a high temperature (T_0) is set; then, the temperature is gradually decreased by the amount of ΔT to reach the freezing point. In each temperature, the algorithm randomly calls one of the walking-around procedures in the limited times to potentially improve the current solution. After all operations of Algorithm 1 are performed, Algorithm 3 is invoked to return a possible denser solution. Once the last round iteration is terminated, the best so far solution is returned as the optimal solution. In fact, Algorithm 3 uses a circular queue as a data structure. For each used bin, it searches in a circular manner to find a bin that has ample capacity for that used bin. If so, the items from the source bin are completely offloaded to the destination bin. Then, the source bin is released. In this way, the possible denser solution is gained with fewer used bins. Note that, the Check&Correct(.) procedure is a heuristic approach that works akin to Algorithm 3. Once either exploration or exploitation operators are performed, the solutions may have impairment because the genes are substituted. The possible impairments occur when a bin adopts more than its capacity. The Check&Correct(.) procedure first checks the solution. If it finds a bin that is overweight, it selects an overhead item; then, it searches in the used bins in a circular manner to find a bin that has ample capacity to adopt that overhead. If so, it is offloaded. This approach is iterated until the solution is corrected.

Algorithm 3 inputs the current solution and returns a possible denser solution. The time complexity of Algorithm 2 is $n + m^2$ which belongs to $O(m^2)$. With a closer look, the time complexity of Algorithm 3 is $O(m^2)$ because of the two nested loops each of which iterates at

Algorithm 3. Integration procedure;

Input
n : Number of objects;
m : Number of available bins;
W_i : The *i*-th solution with length n using *K* bins;
/* $W_i = (w_{i1}, w_{i2}, \ldots, w_{in})$; */
Output
A new possible denser solution W_i which uses less than *K* bins;

1: **For i ← 1 To *m* Do**
2: **if** the bin_i is being used **Then**
3: **j ← i+1;**
4: **while** j ≤ *m* **Do**
5: **if** the bin_j is being used and has ample capacity for objects bin_i **Then**
6: Offload objects bin_i of to bin_j;
7: Release bin_i;
8: Decrease *K* as number of being used bins;
9: **end-if**
10: **j ← j+1;**
11: **end-while**
12: **end-if**
13: **End-For**
14: return new solution $W_i[1..n]$**;**
15: End {**Algorithm 3**}

most *m* times. The parameter *m* is the number of maximum bins as the number of all resources in the system. Algorithm 3 is a heuristic that is called to reduce the number of active being used bins as possible. For the sake of disperse-ness of the objects over bins during the optimization process by calling different changing procedures, there may exist a contingent fortune to offload all of the co-hosted objects of the source bin to the destination active bin which has ample capacity to adopt all of the source objects. Then, it releases the source bin and the number of being used bins is decreased. During the walking around process, calling one of the exploitation procedures may lead the direction to either an upward or downward trajectory. Here, the uphill/downhill walking around procedures are introduced.

4.3 Novel Uphill/Downhill Walking Around Procedures

In this part, five types of uphill/downhill procedures are introduced. The procedures' names are $UDHC_1$, $UDHC_2$, $UDHC_3$, $UDHC_4$, and $UDHC_5$. The result of worse/better determines whether it is uphill climbing or downhill climbing; the reason why the name uphill/downhill climbing (*UDHC*) was selected. All of the uphill/downhill procedures search gently around a current solution. In $UDHC_1$, each pair of genes is exchanged by starting from the even position with its next gene. Algorithm 4 elaborates on the details of changing genes.

Figure 6 shows how this procedure works. If the length of a candidate solution is *n*, the time complexity of $UDHC_1$ is O(*n*).

In $UDHC_2$, each pair of genes is exchanged by starting from the odd position with its next gene.

Algorithm 5 elaborates the functionality of the $UDHC_2$ procedure.

Figure 7 shows how this procedure works. If the length of a candidate solution is *n*, the time complexity of $UDHC_2$ is O(*n*).

Algorithm 6 is dedicated to preparing the $UDHC_3$ procedure. In $UDHC_3$, a random substring *W*[S..E] is selected. Then, a point (C) within this substring is determined to halve the substring into two sections, namely, *W* [S..C] and *W*[C + 1..E]. Afterward, the subsections are changed similarly to a single-point crossover in the genetic algorithm; in other words,

the section W[C + 1..E] is placed from W[S], then W [S..C] is placed in the head.

Figure 8 shows how this procedure works. If the length of a candidate solution is n, the time complexity of $UDHC_3$ is O(n).

Algorithm 7 presents another walking-around procedure that potentially changes the results.

In $UDHC_4$, a random substring W[S..E] is selected. Then, this substring is completely revered. Figure 9 shows how this procedure works. If the length of a candidate solution is n, the time complexity of $UDHC_4$ is O(n).

Algorithm 8 presents the last walking-around procedure $UDHC_5$. In the $UDHC_5$, two random genes are selected. Then, they are substituted similarly to the mutation procedure in the genetic algorithm.

Figure 10 shows how this procedure works. If the length of a candidate solution is n, the time complexity of $UDHC_5$ is O(*1*).

4.4 Time Complexity of the Proposed HD-GWO

Since the exact execution time of an algorithm deeply depends on several factors such as underlying hardware and used compiler which may be apart from the logic of the designed algorithm, the time complexity

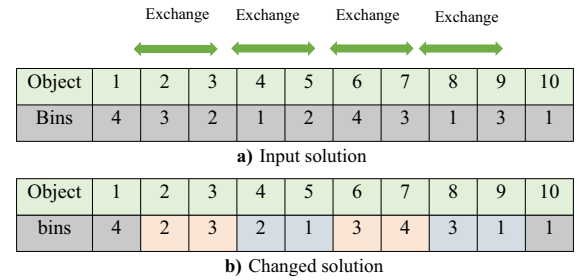

Fig. 6 Application of $UDHC_1$ as the first type of walking around

of the proposed algorithm is presented which is the best reflection of the sense. In the suggested algorithms two parameters n and m are the number of objects (also the length of the encoded wolf) and bins respectively. The time complexity of Algorithm 2 is mainly relevant between lines 3 through 12 which is O(n), but it is possible that the impaired encoded solution needs to be amended; the Check&Correct(.) procedure is called that consumes m^2; so, the time complexity of Algorithm 2 is O($n + m^2$). The time complexity of Algorithm 3 is also O($n + m^2$) because of using two nested loops each of which in length of m. All of the walking-around procedures $UDHC_1$ through $UDHC_5$ known as Algorithm 4 through

Algorithm 4. $UDHC_1$ procedure;

Algorithm 4. UDHC$_1$ procedure;

Input
n : Number of objects;
m : Number of available bins;
W_i : The i-th solution with length n using K bins;
/* $W_i = (w_{i1}, w_{i2}, \ldots, w_{in})$; */

Output
NewW [1..n] : A new changed solution;

```
1: For j ← 1 To floor(n/2) Do
2:     NewW [2j-1] ← Wi [2j];
3:     NewW [2j] ← Wi [2j-1];
4: end-For
5: if n is odd Then
6:     NewW [n] ← Wi [n];
7: end-if
8: return new solution NewW[1..n];
9: End {Algorithm 4}
```

Algorithm 5. *UDHC*$_2$ procedure;

Input
n : Number of objects;
m : Number of available bins;
W_i : The *i*-th solution with length n using *K* bins;
/* $W_i = (w_{i1}, w_{i2}, \ldots, w_{in})$; */
Output
NewW [1..*n*] : A new changed solution;

1: NewW [1] ≻ W_i [1];
2: **For j ≻ 1 To Ceil(*n*/2-1) Do**
3: NewW [2j] ≻ W_i [2j+1];
4: NewW [2j+1] ≻ W_i [2j];
5: **end-For**
6: **if** *n* is even **Then**
7: NewW [*n*] ≻ W_i [*n*];
8: **end-if**
9: return new solution NewW[1..*n*];
10: End {**Algorithm 5**}

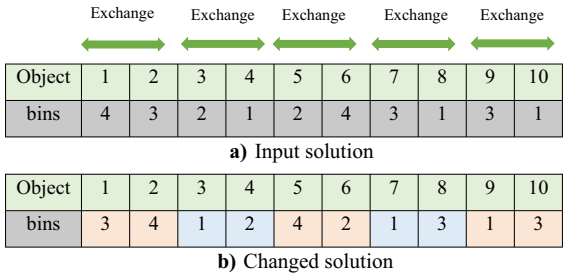

Fig. 7 Application of *UDHC*$_2$ as the second type of walking around

Algorithm 8 consume O(*n*) in the worst case. Now that the time complexity associated with all of the sub-algorithms has been measured, the time complexity of Algorithm 1 can be simply computed. This main algorithm performs its instructions between lines 6 and 61 with *MaxIteration* times. In each iteration, all of the wolves are undergone changes. The number of wolves is *PopSize*. For each wolf, the exploration, exploitation, and integration process are performed. For exploration, the Algorithm 2 is called. For exploitation, the semi-SA behavior is done. In this process, the high-temperature T_0 is used which is gradually decreased to the Freeze point by declining ΔT amount. At each temperature, a number of limited changes (*MaxProcess* times) are performed so the solution reaches its stable thermodynamic point [40]. During the gradually decreasing temperature, one of the neighboring procedures or walking-around is called O(*n*) complexity. So, the exploitation phase consumes $MaxProcess \times \frac{(T_0 - Freeze)}{\Delta T} \times n$ times for each individual. The time of all instructions in the Algorithm 1 is $t = MaxIteration \times PopSize \times [(n + m^2) + MaxProcess \times \frac{(T_0 - Freeze)}{\Delta T} \times n + (n + m^2)]$. Therefore, the time complexity of Algorithm 1 is gained by (11).

$$O\left(MaxIteration \times PopSize \times [MaxProcess \times \frac{(T_0 - Freeze)}{\Delta T} \times n + m^2\right] \quad (11)$$

5 Performance Evaluation

To evaluate the effectiveness of the proposed algorithm for solving the two-dimensional bin-packing problems, extensive experiments are done in different conditions to reach trustable results. Take there are ample homogeneous two-dimensional resources (bins or physical servers) in the systems to run requested applications

Algorithm 6. *UDHC*$_3$ procedure;

Input
n : Number of objects;
m : Number of available bins;
W_i : The *i*-th solution with length n using *K* bins;
/* $W_i = (w_{i1}, w_{i2}, \ldots, w_{in})$; */
Output
NewW [1..*n*] : A new changed solution;

```
1: Draw two random integers S<E ∈ [1..n];
2: Draw a random integers C ∈ [S+1..E-1];
3: NewW [1..S-1] ≻ Wi [..S-1];
4: idx ≻ S;
5: j ≻ C+1;
6: while j<=E do
7:    NewW [idx] ≻ Wi [j];
8:    idx++;
9:    j++;
10: end-while
11: j ≻ S+1;
12: while j<=C do
13:    NewW [idx] ≻ Wi [j];
14:    idx++;
15:    j++;
16: end-while
17: NewW [1..S-1] ≻ Wi [1..S-1];
18: return new solution NewW[1..n];
19: End {Algorithm 6}
```

(objects or virtual machines); note that, each application needs all of the resource dimensions at the same time. For instance, co-hosted applications on a single physical machine require CPU and memory bandwidth at the same time. Therefore, the VMP for applications is bounded to the resource capacity of PM in all dimensions. Since the units of resources are different, for the sake of simplicity, the vector of normalized resource request is taken into account in each dimension that is in (0..1] interval. The normalized requests are 20, 40, 70, 100, and 200 two-dimensional objects (VMs) to be placed on two-dimensional available resources that are abstracted to bins (PMs). The objective is to use a minimum number of available bins to reduce the overall costs. Meanwhile, the resource wastage must be declined to preclude possible additional and residual resource usage for the next rounds. The last input, 200 requests, is considered for analyzing the proposed algorithm's scalability. To assess the effectiveness of the proposed *HD-GWO*, a wide range of inputs are generated with different pairwise correlation coefficient values in all dimensions to make sense of real-world data. Therefore, three correlation coefficient values {+0.05 (independent dimensions), +0.75 (strongly dependent dimensions), and -0.75(strongly anti-dependent dimensions)} are incorporated. In total, 15 experiments in different circumstances are conducted which are five groups of data each of which with three kinds of correlation coefficients between each dimension. Since the cloud and fog environments are dynamic in nature, the circumstances are always changing. Therefore, in different time intervals, our proposed VMP scheme can be called to reduce overall costs. The time interval and upper threshold for each physical server can be adjusted by experts and system admins. For now, we run one snapshot of a time interval. Meanwhile, the SLA terms must be always met otherwise the penalty should be paid to customers. To this end, the (SLA_k) as the SLA of each physical machine (PM_k) is met provided Eq. (12) is satisfied.

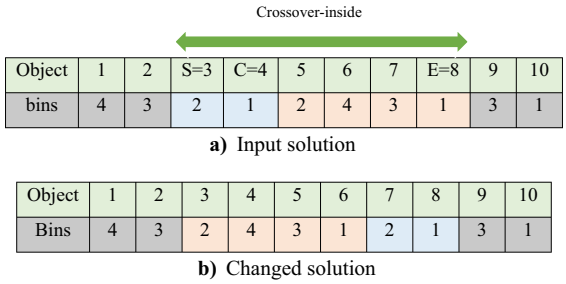

Fig. 8 Application of $UDHC_3$ as the third type of walking around

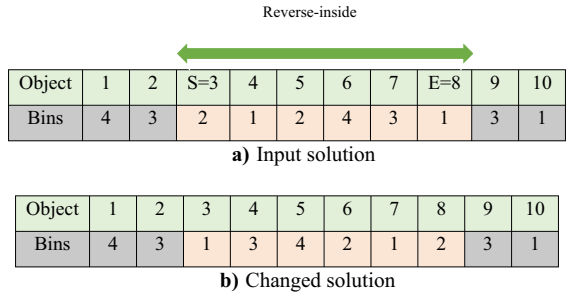

Fig. 9 Application of $UDHC_4$ as the third type of walking around

$$\mathrm{SLA}_j = \frac{1}{1 + e^{\mathrm{Thr} - S_{CPU}^{PM_k}}} < 0.5, \text{for each } \mathrm{PM}_j, j = 1, \ldots, m \tag{12}$$

In Eq. (12), the terms Thr and $S_{CPU}^{PM_k}$ are the CPU threshold of a server and the sum of accumulated CPU bandwidths that are being used by the co-hosted VMs known as CPU utilization. The SLA violation occurs if the SLA value exceeds 0.5 based on a presented formula. In this case, re-deployment VMP is performed. For now, the paper concentrates on primary VM deployment by considering Eq. (12) not to violate the agreed SLA to supply requested resources for applications proactively. Another important thing to mention is associated with threshold (Thr) determination. In this case, several ML approaches such as the clustering methods presented in [28] can be used which access data history. For instance, the upper threshold can be updated such as in [41]. In fact, the new threshold is updated based on the previous threshold, requested SLA, and experienced (delivered) SLA by Eq. (13).

$$\mathrm{Thr(new)} \propto \frac{Requested\ SLA}{Delivered\ SLA} * Thr(old) \tag{13}$$

For now, this paper considers a static upper threshold although it can be updated based on Eq. (13) in an environment with high fluctuation rates.

Algorithm 7. $UDHC_4$ procedure;

Input
n : Number of objects;
m : Number of available bins;
W_i : The i-th solution with length n using K bins;
/* $W_i = (w_{i1}, w_{i2}, \ldots, w_{in})$; */
Output
NewW [1..n] : A new changed solution;

1: Draw two random integers S<E ∈ [1..n];
2: NewW [1..S-1] ≻ W_i [1-S-1];
3: NewW [E+1.. n] ≻ W_i [1-S-1];
4: idx ≻ S;
5: j ≻ E;
6: **while** j ≥ S **do**
7: NewW [idx] ≻ W_i [j];
8: idx++;
9: j--;
10: **end-while**
17: return new solution NewW [1..n];
18: **End** {**Algorithm 7**}

Algorithm 8. *UDHC*$_5$ procedure;

Input
n : Number of objects;
m : Number of available bins;
W_i : The *i*-th solution with length n using *K* bins;
/* $W_i = (w_{i1}, w_{i2}, \ldots, w_{in})$; */
Output
NewW [1..*n*] : A new changed solution;

1: Draw two random integers S<E ∈ [1..*n*];
2: NewW [1.. *n*] ⇝ W_i [1..*n*];
3: NewW [S] ⇝ W_i [E];
4: NewW [E] ⇝ W_i [S];
5: return new solution NewW [1..*n*];
6: End {**Algorithm 8**}

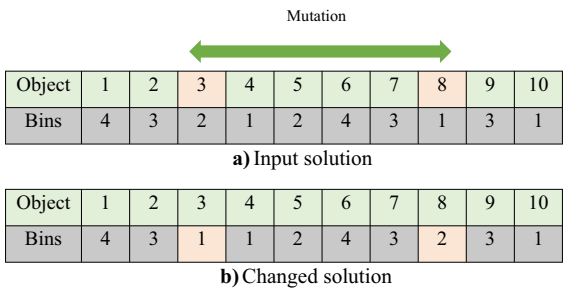

Fig. 10 Application of *UDHC*$_5$ as the fifth type of walking around

5.1 Comparative Algorithms and Parameter Settings

Upon investigating the literature review, the most effective state-of-the-art and newly published from reputed publications have been selected for competition against the proposed *HD-GWO*, in solving two-dimensional bin-packing problems in extensive scenarios. To this end, some famous and effective algorithms in each artificial intelligence category are opted, namely, the first fit decreasing order (FFD) [32] which is a strong heuristic algorithm, simulated annealing-based bin-packing algorithm [11] that is a non-evolutionary algorithm, genetic-based bin-packing [34] that is a strong population-based meta-heuristic algorithm, and GWO-based [17] that is a strong swarm-based meta-heuristic algorithm. The selected algorithms are customized based on the stated problem under-study and this paper's conditions. All of the algorithms are run in the same condition on the same platform and on the same datasets to have fair comparison and trustable results to rely on. Note that, different heuristic and meta-heuristic algorithms have been tried on the same problems, but the most effective ones were selected for final demonstration. Bear in mind that the FFD algorithm is applied to the one-dimensional bin-packing problems, the smarter FFD in comparison with the original FFD has been incorporated. For this reason, the sorting of objects is in decreasing order based on the effect of both equal dimensions similar to the heuristic proposed in [32]. In other words, the decreasing arrangement of objects is done based on the reverse dissipation value of objects which was demonstrated in Eq. (7). In the same dissipation value, the bigger object is selected. So, the comparative algorithms are named smart FFD (S-FFD) customized of Ref. [32] published on 2022, simulated annealing-based two-dimensional bin-packing (SA2DBP) customized of Ref. [11] published on 2022, genetic-based two-dimensional bin-packing (GA2DBP) customized of Ref. [34] published on 2020, grey wolf optimization-based two-dimensional bin-packing (GWO2DBP) customized of Ref. [17] published on 2020, and one of the most applicable swam-based optimizers is a particle swarm optimization (PSO) algorithm that is customized to PSO2DBP derived from newly published for cloud resource management in Ref. [42]. Note that the proposed algorithm is named hybrid discrete grey wolf 2D bin packing solver (HD-GWO2DBP) in these comparisons. Recall that customized GA2DBP uses single-point crossover for exploration and mutation

for running away from local trap. In addition, all of the comparative algorithms use the same encoding that was mentioned in subsection 4.1. Since the state-problem is discrete optimization and majority of comparatives such as PSO and GWO are continuous optimizer, either the S-shape or V-shape operators are incorporated provided return better solutions in terms of objective function. The S-shape and V-shape operators were presented by Mirjalili et al. to discretize solutions [43]. Table 3 provides the parameter settings of comparative algorithms. The parameters determine how to tune the comparative approaches commensurate with the under-study experiments.

All of the scenarios were run in fair conditions on a windows 8 platform with a dual core Intel Corei3 380 M and 2.53 GHZ processor clock rate, equipped with four logical processors and 8 GB as main memory in its hardware; by applying the MATLAB 2018 programming language environment.

5.2 Datasets and evaluation metrics

As mentioned earlier, the random normalized values in (0..1] interval are generated for both dimensions of requested objects. The number of objects is 20, 40, 70, 100, and 200 respectively. Three different datasets are generated for each scenario in such a way that the correlation coefficient values between resource dimensions are $+0.05, +0.75$, and -0.75 respectively for independent, strongly dependent, and strongly anti-dependent dimensions. In total, 15 extensive experiments are run. For each experiment, twenty independent executions are run; then, the average results are reported in terms of determined evaluation metrics. One of the most important objective metrics is to count a number of used bins to which Eq. (1) is dedicated. Note that, the two-dimensional bin-packing problem is an abstract problem in mathematics and has a variety of applications in different industries. To have tangible results, an important application of the two-dimensional bin-packing problem is considered for the information technology (IT) domain. For instance, the number of used servers in a typical data center has drastic impacts on the overall cost. In addition, power consumption puts side effects on both overall cost and climate changes [3, 4]. So, apart from the number of used bins (physical servers), the amount of power consumption is calculated by Eq. (14). Recall that, the most part which is incorporated in the total power consumption of each

Table 3 Parameter settings of comparative algorithms

Customized Comparative Algorithms /Ref	Specific parameters				Number of Objective	Population Size	Max Iterations
GA2DBP [34]	Crossover Percentage:		80%		2	100 ~ 200 depending on scenario	40 ~ 50 depending on scenario
	Mutation Percentage:		10%				
GWO2DBP [17]	Archive size: nGrid:	100 10	α: β: δ:	0.1 4 2			
Proposed HDG-WO2DBP	Archive size: nGrid:	100 10	α: β: δ:	0.1 4 2			
	T_0 :1000 ΔT: 20	*Freeze:* 10 *MaxK*: 20	λ_1 : (0..1] λ_2 : (0..1]				
SA2DBP [11]	T_0 :1000 ΔT: 20	*Freeze:* 10 *MaxK*20	λ_1 : (0..1] λ_1 : (0..1]				
PSO2DBP [42]	**ω**: 0.4 C1: 2 C2: 2						

α, Grid Inflation parameter; β, Leader selection pressure parameter; δ, Extra repository member selection pressure, *Archive size,* Repository size; *nGrid*, Number of Grids per each Dimension; *maxvel*, Maxmium velocity in percentage (search space percentage); *u_mut*, Uniform mutation percentage; ω, Inertia weight; *C1*, Individual confidence factor; *C2*, Swarm confidence factor

server strongly depends on the CPU utilization. The CPU utilization of a server (a bin) is calculated by the sum of resource usage (dimensions) of objects (virtual machines).

$$P_j = [(\lambda \times P_{j,Full} + (1-\lambda) \times P_{j,Full} \times U_W^j) + \varepsilon_0 \times U_H^j] \times Z_j \quad (14)$$

The binary variable Z_j indicates whether j-th bin (here is the physical server) is in use or not. The power consumption of a server depends on the first dimension (W: width) or CPU utilization (significantly) and the second dimension (H: height) or RAM utilization (negligible). The parameters U_W^j and U_H^j are the first and the second dimension utilization of a j-th bin that can be computed by the sum of dimension values of all placed objects in each dimension. The parameters $P_{j,Full}$ is the power consumption of a full-loaded physical server and λ is about 70% [3, 4]. The total power consumption of all applied two-dimensional resources (two-dimensional bins) is calculated via Eq. (15).

$$\text{Total Power(Sol)} = \sum_{j=1}^{m} P_j \times z_j \quad (15)$$

In this experiment, the physical servers are HP ProLiant servers each of which has 1800 MIPS and 12 GB as CPU and main memory capacity in which every full-loaded server consumes about 300 Watt/hour as power consumption. Table 4 demonstrates all of the experiment specifications.

5.3 Comparison and data analysis

In this subsection, the performance comparison and data analysis between state-of-the-art are done. The scenarios are conducted in five groups each of which needs the same number of objects (VMs) to be packed in the minimum number of bins (PMs) and they are tested in three different conditions. The conditions are relevant to pairwise values in input dimensions. These conditions are specified by the *CC* parameter. Except for S-FDD which is a heuristic approach, other meta-heuristic-based algorithms are run several times. Then, the average values of evaluation metrics are reported in figures. In the three first experiments, the number of requested objects is 20. Figure 11 demonstrates the effectiveness of each solution in regard to solving the stated problem. As Fig. 11 shows the S-FFD works the worst, but in some cases it competes with SA2DBP. The GA2DBP and PSO2DBP compete with GWO2DBP and the proposed HD-GWO2DBP algorithm. Since the search space is very small, all of the comparative algorithms approximately reach the same results.

For the sake of the illustration of iterative comparisons, Fig. 12 is dedicated to showing how comparative algorithms reach to stable point. Since S-FDD is a heuristic and works with a predefined criterion, the reason why it has steady behavior in every iteration which Fig. 12 depicts the proposed HDGWO2DBP returns the best results. Note that Fig. 12 returns the number of used bins (PMs); then, the power consumption is calculated by Eq. (15) incorporated in Fig. 11 One important thing to mention about the step-style of drawing of Figs. 12, 14, 16, 18, and 20 in the continuity of drawing after each iteration is that the stated problem is not continuous optimization instead it is a discrete optimization issue, the reason why the drawing is shown in step-like shape. In other words, the shape is not a curve, but is step-like shape.

For the next three scenarios where the number of object requests is 40, the contrast between comparative algorithms is drawn in Fig. 13. When the search space grows, the discrepancy amongst the performance of the comparative algorithms has high fluctuation. Again, the S-FFD has the worst result because it is a greedy algorithm the reason why it cannot compensate for its previous inefficient decision in the course of optimization. In contrast, other meta-heuristic-based algorithms have enough time to compensate for previous decisions toward objectives. In some cases, S-FFD competes against SA2DBP and GA2DBP marginally competes with the GWO2DBP algorithm. Moreover, the PSO2DBP works akin to GWO2DBP. In the scenario where the *CC* is + 0.75, the GA only beats the GWO algorithm. In most cases, the proposed HD-GWO2DBP algorithm leads the better results. This result revolves around the fact that it conducts solutions in such a way as to balance resource dimensions during placing objects. It indirectly causes less resource wastage and less number of used bins; and, consequently, it leads the lower power consumption.

Figure 14 illustrates to showing how comparative algorithms reach to stable point. For instance, the SA2DBP and PSO2DBP suffer from early convergence, namely, the former in CCR = -0.75 and the latter in CCR = + 0.05 experiments respectively. Since

Table 4 All experiments scenarios

Number of Experiment	Number of requested objects	Data range x: first dimension y: second dimension	Correlation Coefficient (CC) between dimensions
1	20	x ϵ (0..1] y ϵ (0..1]	- 0. 75 (Anti-dependent)
2	20	x ϵ (0..1] y ϵ (0..1]	+0. 05 (independent)
3	20	x ϵ (0..1] y ϵ (0..1]	+0. 75 (strongly dependent)
4	40	x ϵ (0..1] y ϵ (0..1]	- 0. 75 (Anti-dependent)
5	40	x ϵ (0..1] y ϵ (0..1]	+0. 05 (independent)
6	40	x ϵ (0..1] y ϵ (0..1]	+0. 75 (strongly dependent)
7	70	x ϵ (0..1] y ϵ (0..1]	- 0. 75 (Anti-dependent)
8	70	x ϵ (0..1] y ϵ (0..1]	+0. 05 (independent)
9	70	x ϵ (0..1] y ϵ (0..1]	+0. 75 (strongly dependent)
10	100	x ϵ (0..1] y ϵ (0..1]	- 0. 75 (Anti-dependent)
11	100	x ϵ (0..1] y ϵ (0..1]	+0. 05 (independent)
12	100	x ϵ (0..1] y ϵ (0..1]	+0. 75 (strongly dependent)
13	200	x ϵ (0..1] y ϵ (0..1]	- 0. 75 (Anti-dependent)
14	200	x ϵ (0..1] y ϵ (0..1]	+0. 05 (independent)
15	200	x ϵ (0..1] y ϵ (0..1]	+0. 75 (strongly dependent)

S-FDD is a heuristic and works with a predefined criterion, the reason why it has steady behavior in every iteration which Fig. 14 depicts the proposed HDGWO2DBP returns the best results. Note that Fig. 14 returns the number of used bins (PMs); then, the power consumption is calculated by Eq. (15) incorporated in Fig. 13.

For the three scenarios where the number of object requests is 70, when the search space grows to more extent, the discrepancy between the solutions and proposed HD-GWO2DBP shows more extent. Figure 15 is dedicated to comparing the performance of comparative algorithms once the number of requested objects is 70. In contrast, after HD-GWO2DBP, the GWO2DBP, GA2DBP, PSO2DP, SA2DBP, and S-FFD are placed in the next ranking from the best to the worst. Note that, some of them compete marginally with each other.

Figure 16 illustrates to showing how comparative algorithms reach to stable point. In contrast to other state-of-the-art, the proposed algorithm returns the minimum used bins and also the minimum total power consumption owing to using PMs in data centers that Fig. 15 elaborates in the bar chart drawn on the right side.

For the next three scenarios where the number of object requests is 100 when the search space grows with more extent, the discrepancy between the solutions of proposed HD-GWO2DBP shows more extent even more than in Fig. 17. Figure 19 is dedicated to comparing the performance of comparative algorithms once the number of requested objects are 200.

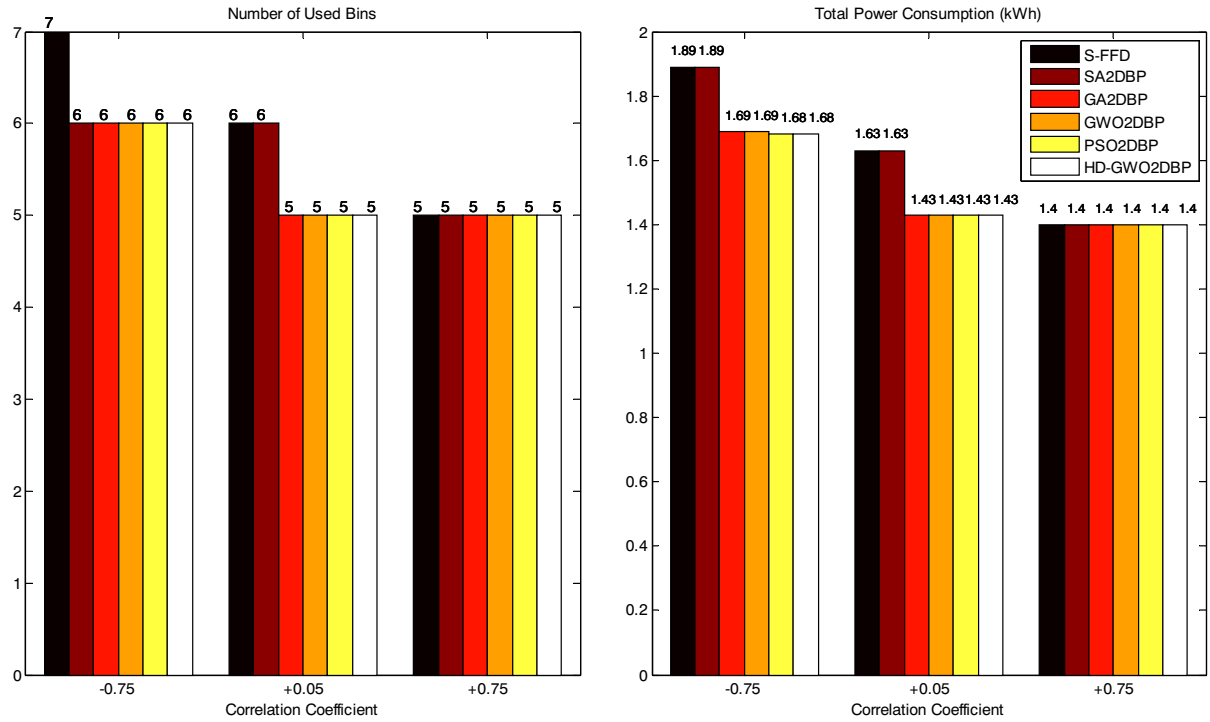

Fig. 11 Comparative evaluation plots of state-of-the-art in requesting 20 objects in different CC values

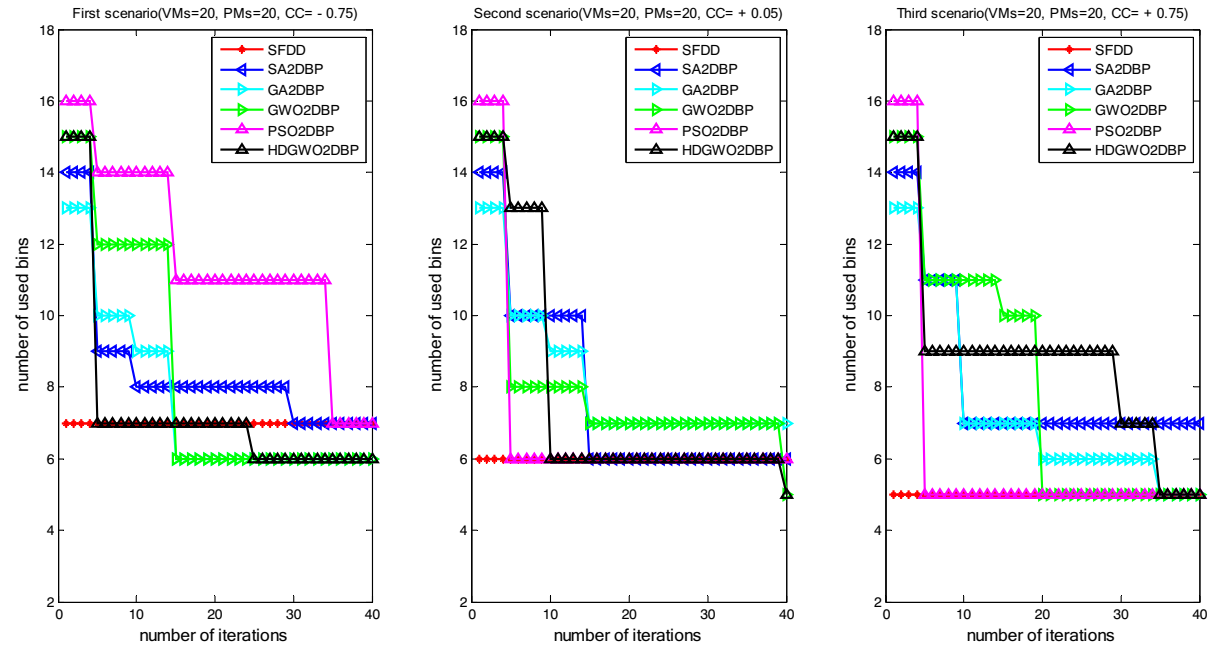

Fig. 12 Convergence of Comparative algorithms to calculate objective function in different CC

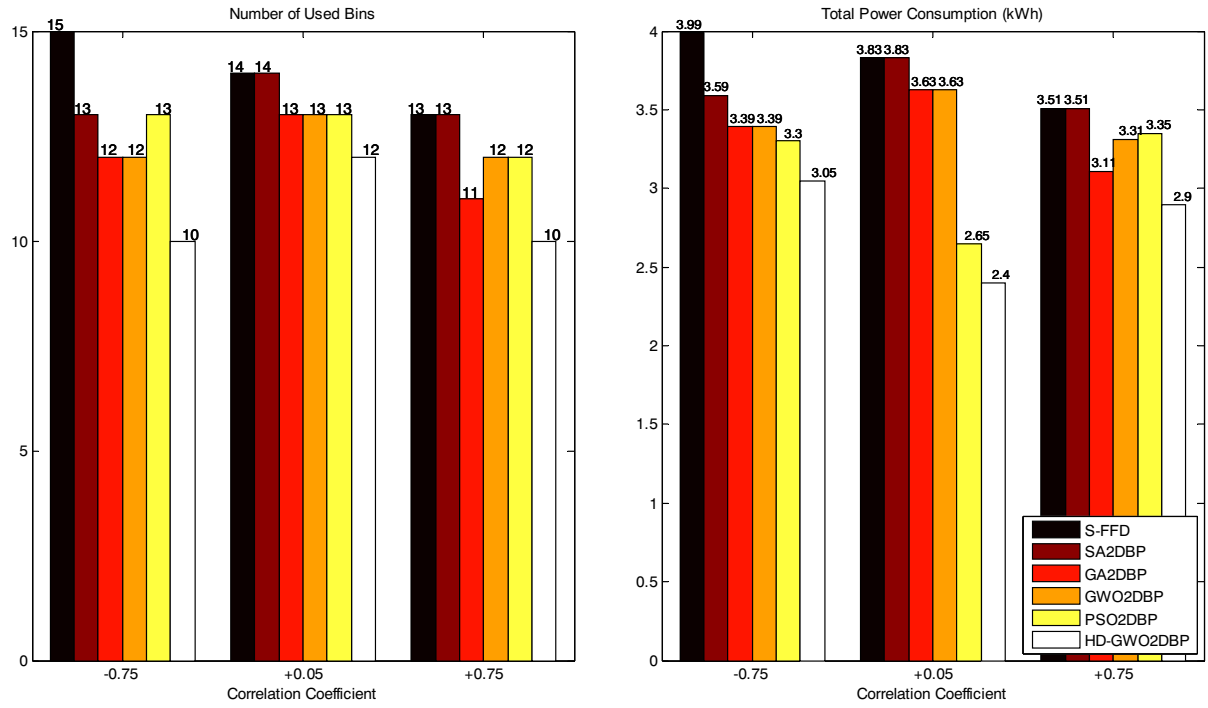

Fig. 13 Comparative evaluation plots of state-of-the-art in requesting 40 objects in different CC values

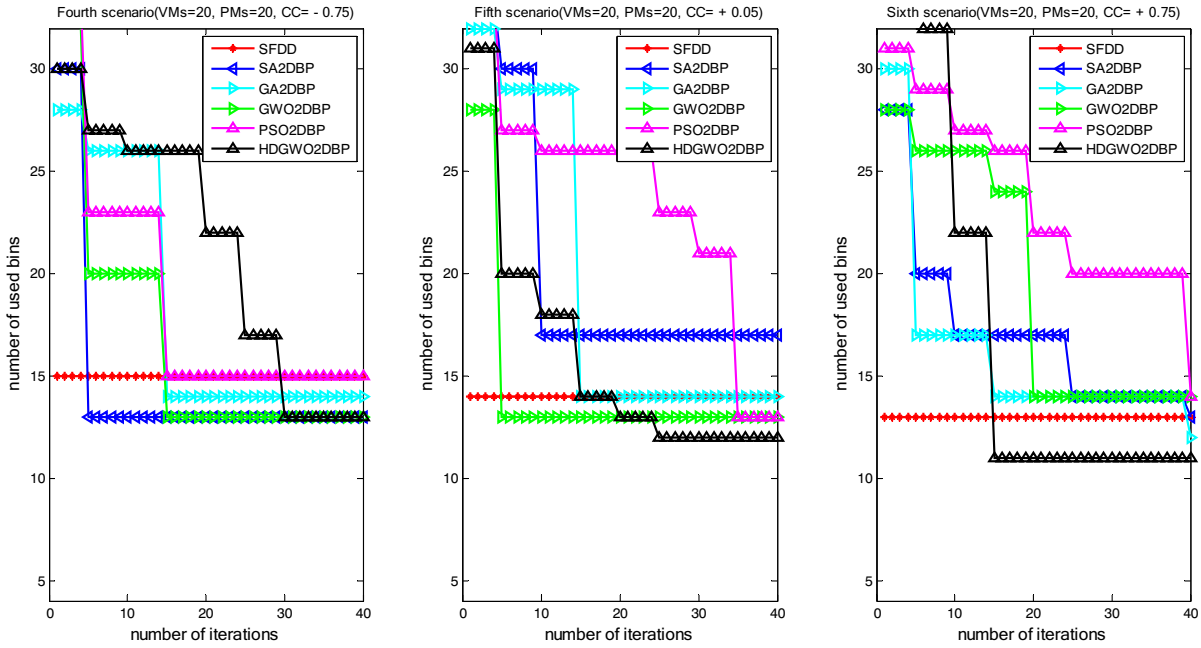

Fig. 14 Convergence of Comparative algorithms to calculate objective function in different CC

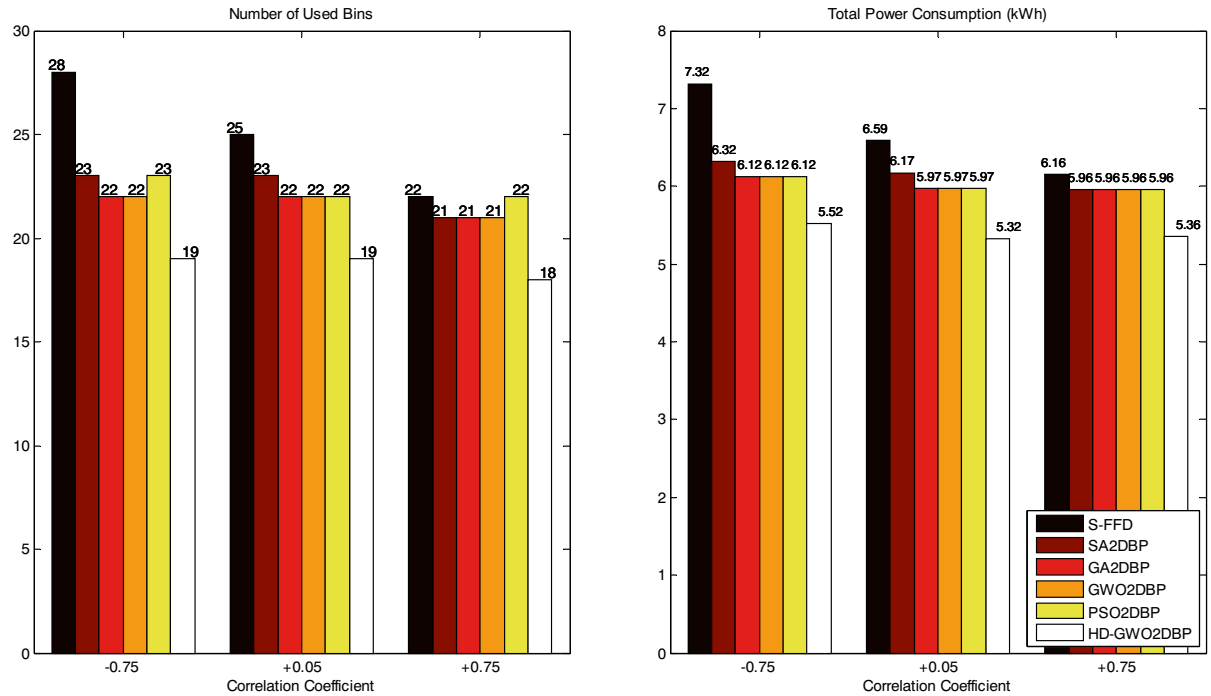

Fig. 15 Comparative evaluation plots of state-of-the-art in requesting 70 objects in different CC values

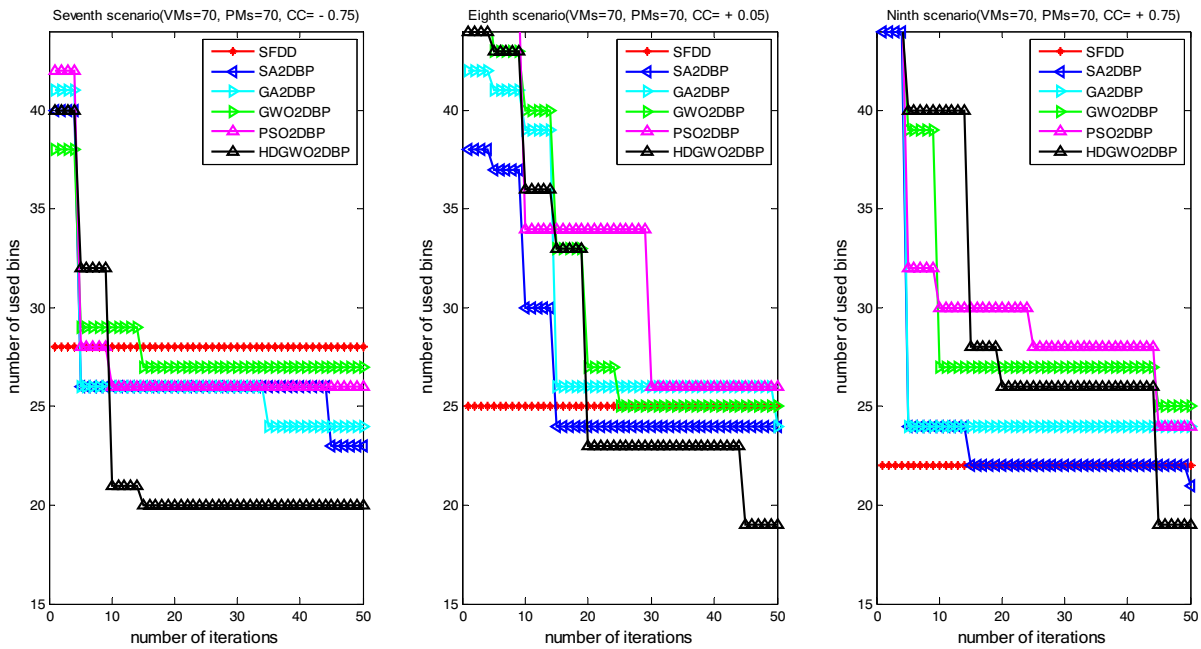

Fig. 16 Convergence of Comparative algorithms to calculate objective function in different CC

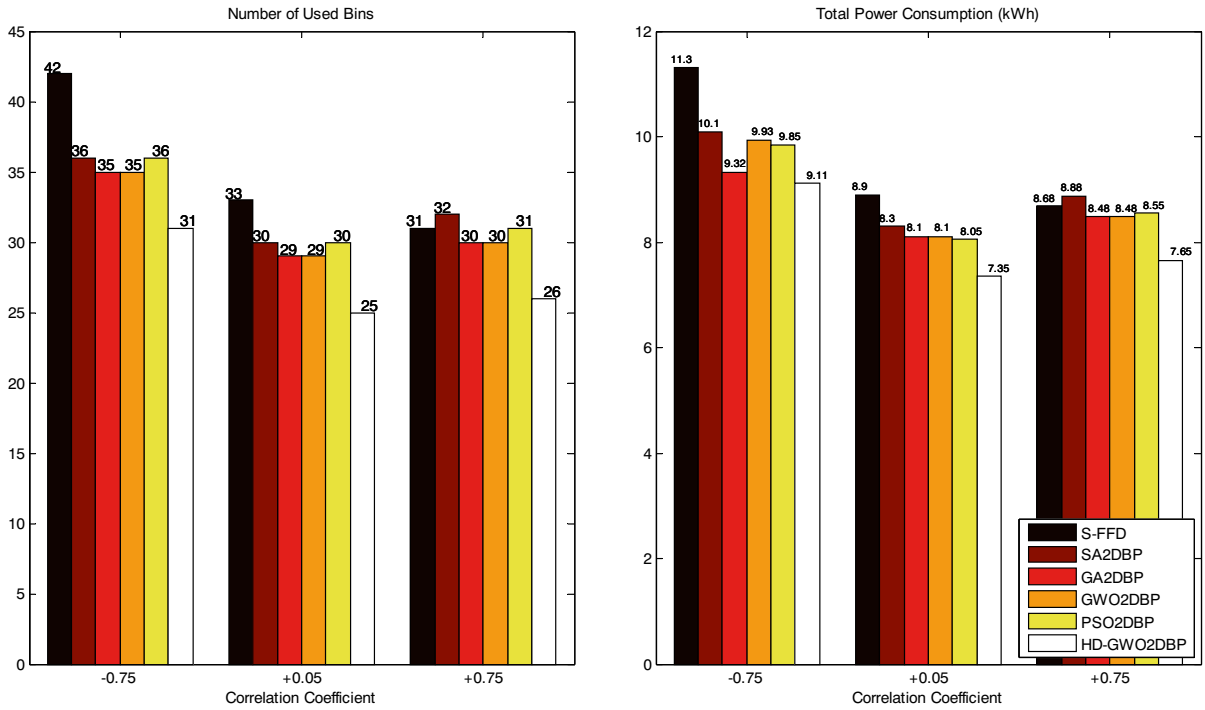

Fig. 17 Comparative evaluation plots of state-of-the-art in requesting 100 objects in different CC values

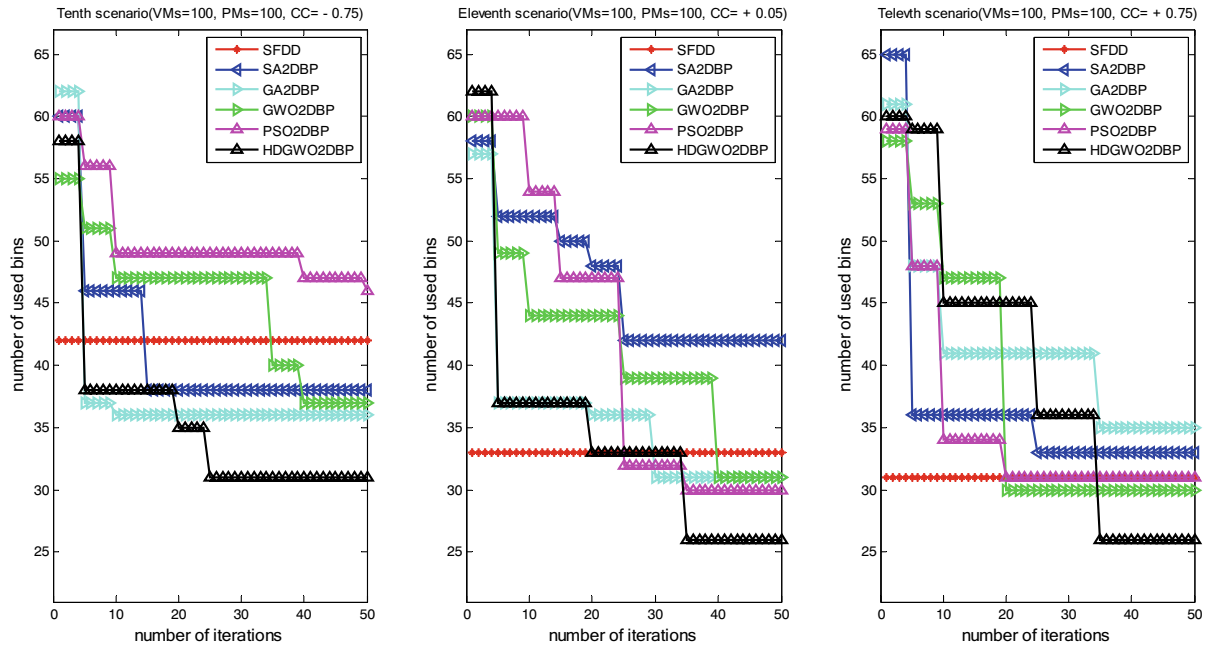

Fig. 18 Convergence of Comparative algorithms to calculate objective function in different CC

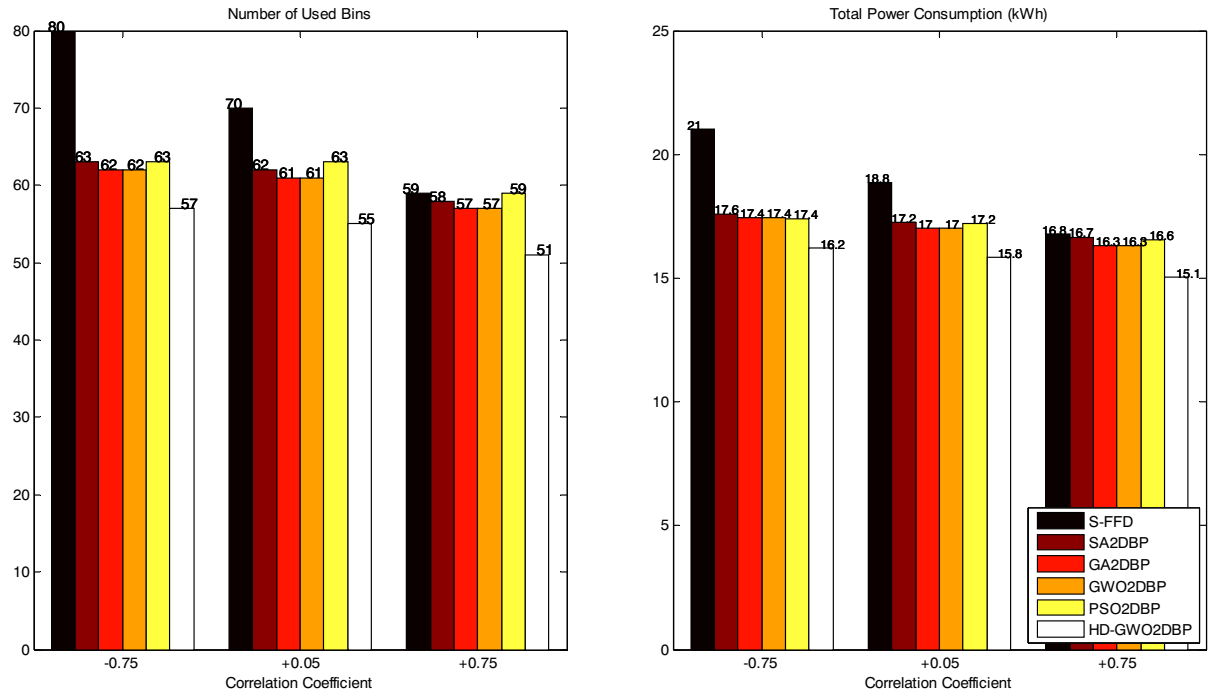

Fig. 19 Scalability evaluation plots of state-of-the-art in requesting 200 objects in different CC values

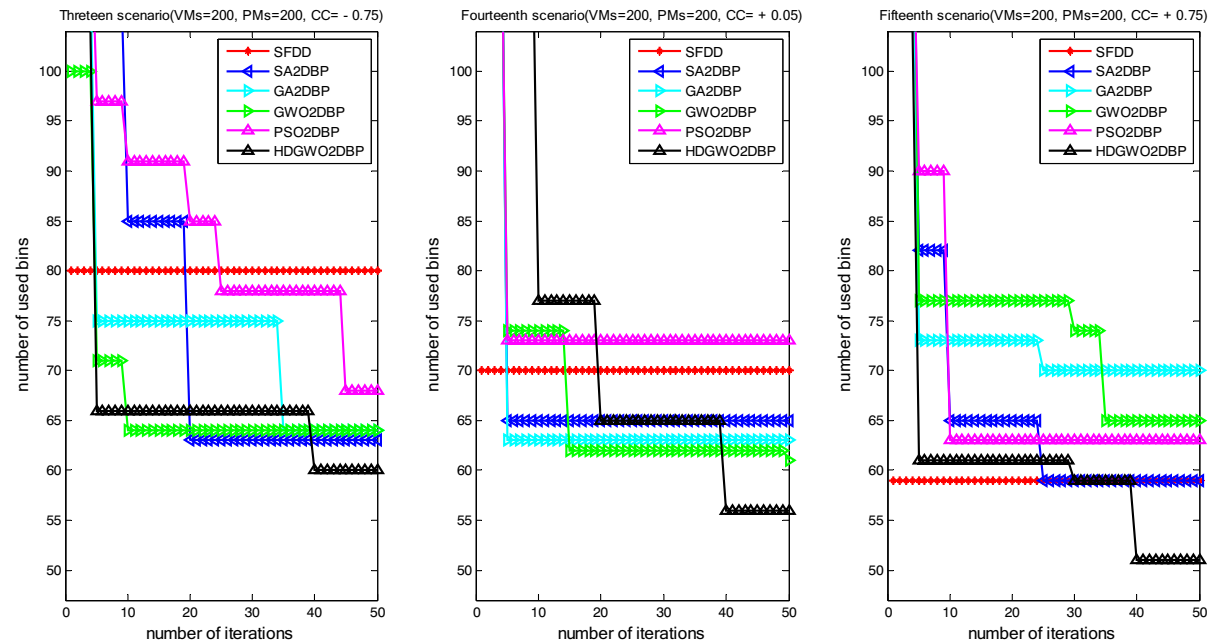

Fig. 20 Convergence of Comparative algorithms to calculate objective function in different CC

Table 5 All experiments scenarios

Number of Experiment	Number of requested objects	Correlation Coefficient (CC) between dimensions	Cost Score (Algorithms)					
			S-FFD	SA2DBP	GA2DBP	GWO2DBP	PSOO2DP	HDGWO2DBP
1	20	- 0. 75 (Anti-dependent)	0.48	0.43	0.42	0.40	0.41	**0.38**
2	20	+ 0. 05 (independent)	0.43	0.39	0.38	0.37	0.38	**0.34**
3	20	+ 0. 75 (strongly dependent)	0.40	0.37	0.36	0.36	0.36	**0.32**
4	40	- 0. 75 (Anti-dependent)	0.49	0.47	0.45	0.44	0.44	**0.41**
5	40	+ 0. 05 (independent)	0.47	0.44	0.43	0.43	0.45	**0.40**
6	40	+ 0. 75 (strongly dependent)	0.50	0.47	0.44	0.44	0.46	**0.39**
7	70	- 0. 75 (Anti-dependent)	0.54	0.50	0.49	0.48	0.48	**0.43**
8	70	+ 0. 05 (independent)	0.56	0.53	0.51	0.50	0.51	**0.45**
9	70	+ 0. 75 (strongly dependent)	0.51	0.46	0.45	0.44	0.44	**0.41**
10	100	- 0. 75 (Anti-dependent)	0.57	0.50	0.48	0.47	0.48	**0.43**
11	100	+ 0. 05 (independent)	0.55	0.51	0.49	0.48	0.48	**0.45**
12	100	+ 0. 75 (strongly dependent)	0.54	0.51	0.49	0.49	0.49	**0.43**
13	200	- 0. 75 (Anti-dependent)	0.66	0.61	0.59	0.57	0.58	**0.48**

Table 5 (continued)

Number of Experiment	Number of requested objects	Correlation Coefficient (CC) between dimensions	Cost Score (Algorithms)						
			S-FFD	SA2DBP		GA2DBP	GWO2DBP	PSOO2DP	HDGWO2DBP
14	200	+0. 05 (independent)	0.60	0.59	0.56		0.54	0.55	**0.46**
15	200	+0. 75 (strongly dependent)	0.56	0.54	0.53		0.50	0.52	**0.45**

This extent proves the high potential of the scalability of the proposed HD-GWO2DBP algorithm. With a closer look, in all of the objective functions, the proposed algorithm beat others significantly. Note that Figs. 18 and 20 are considered for showing the convergence of comparative algorithms in solving the stated problems respectively for scenarios of 100 bins and 200 bins.

The scalability of the proposed HD-GWO2DBP, the next three scenarios are conducted where the number of requested objects is 200. Figure 19 demonstrates the comparison between comparative algorithms in the same condition for solving a real large-scale problem where the number of requests is rather high in regard to typical requests.

The last scenario reflects the scalability behavior of comparative algorithms. As Figs. 19 and 20 show, the proposed HD-GWO2DBP algorithm has a high potential for scalability because it has a meaningful distance in terms of the number of used bins and total power consumption against other state-of-the-art in all evaluation parameters in all *CC* values. It revolves around the fact that all operations of the proposed algorithm were conducted in the smartest trajectory. It cautiously co-hosts objects in the same bin so that resource wastage is minimized; this technique leads to potentially using fewer bins. Also, the handful of discrete operators were devised to permute discrete search space very slowly and patiently. This part is the assimilation of simulated annealing science to reach a stable thermodynamic state gradually and slowly through cooling process. In the main part of the algorithm, the temperature concept was simulated to examine the local search for avoiding from the local trap. The final result proves the effectiveness of intricate points integrated into the proposed HD-GWO2DBP algorithm.

Since the sole number of used bins cannot be an exact determinant for performance evaluation, applying auxiliary metrics is necessary. To this end, the power consumption and new score metrics are defined and used. For instance, two solutions yield the same B number of used bins but with different resource dissipation. Although the additional amount of power consumption indirectly and approximately shows the amount of high resource wastage rate, the new fitness value shows what approach exactly dominates others in regard to all

of the objectives. Therefore, the new *Cost score* is defined via Eq. (16). The term *cost* is coined because the lower value proves the dominance against other comparative solutions.

$$Cost\ score\ (sol) = w_1 \times \frac{\sum_{k=1}^{m} Dis(b_k) \times x_k}{\sum_{k=1}^{m} x_k} + w_2 \times \frac{\sum_{k=1}^{m} P_k \times x_k}{\sum_{k=1}^{m} P_{j,Full} \times x_k} \tag{16}$$

The coefficients w_1 and w_2 determine the importance value of each part where w_1+w_2=1. The first part indicates the normalized amount of resource wastage amortized on the number of used bins and the second term determines the amount of average power consumption against the full-loaded server power usage. The lowest score means the highest dominance. These coefficients can be adjusted by the systems admin who is familiar with the details. For the sake of considering the same importance, the coefficients w_1 and w_2 are set to 0.5. After the introduction of the new parameter *cost score*, it was examined in all experiments. Table 5 is dedicated to showing the amount of measured *cost score*. As Table 5 demonstrates the proposed algorithm outperforms against others in all experiments. The smallest value inserted boldly in Table 5 is a determinant for selecting the best solution which considers all of the objective functions in its solution. It provides information about the effectiveness of a minimum number of used bins and minimum total power consumption with a good balance between the used resources because the new score function distributes effective metrics in the comprehensive weighted formula. Note that the weight of each is considered the same to indicate the same importance.

5.4 Execution Time Comparison

The time complexity of comparative algorithms have been placed in subsection 4.4 which directly depends on their instructions as a function of input parameters. To calculate the real execution time of comparative algorithms, all of them were run in the same platform on the same datasets to reach concrete and fair results. All of the scenarios were run in fair conditions on a windows 8 platform with a dual core Intel Corei3 380 M and 2.53 GHZ processor clock rate, equipped with four logical processors and 8 GB as main memory in its hardware; by applying the

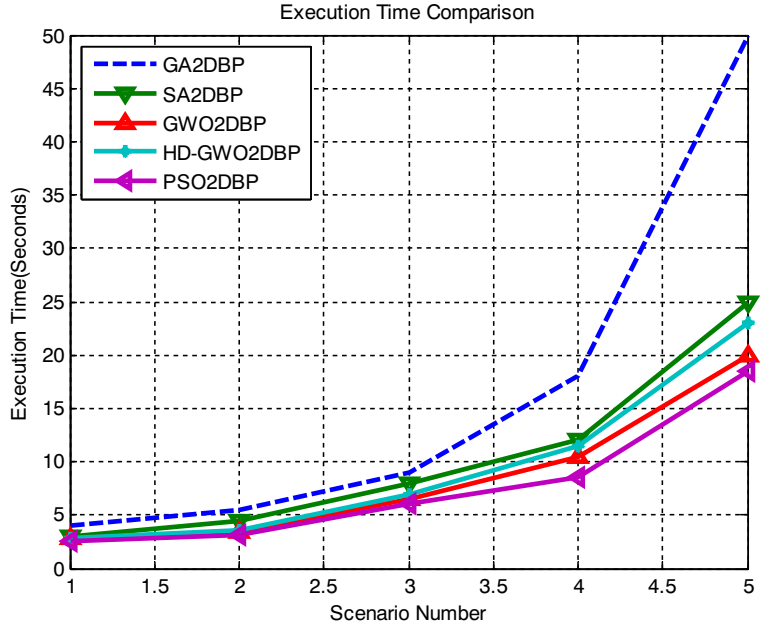

Fig. 21 Comparison of execution time of all comparative algorithms in regarding to input size

MATLAB 2018 programming language environment. The average execution time of each comparative algorithm in regard to the input size is schematically depicted in Fig. 21.

Note that, HD-GWO2DBP works faster than customized GA2DBP and SA2DBP algorithms. Although the canonical SA is very fast in the customized SA2DBP, the inner loop was added to SA2DBP algorithm for each temperature for additional searches to reach stable points; it precludes stopping premature convergence. The customized SA2DBP is incorporated because the canonical SA led the poor results. On the other hand, the GA2DBP works very slowly; it revolves around the fact that it suffers from costly operators; meanwhile, it fails to keep individual experiences gained from previous rounds despite swarm-based intelligence algorithms. In comparison between GWO2DBP and HD-GWO2DBP, the HD-GWO2DBP works slower because it calls for more walking-around procedures to lead a good balance between exploration and exploitation phases. In other words, it runs away from early convergence in which it contingently returns better solutions in more cases. Nevertheless, in terms of real execution time running on the same platform, the PSO2DBP algorithm is the fastest in comparison with all, but in the quality of the produced solutions it stands in the third ranking place after the proposed HD-GWO2DBP and GWO2BP algorithms. It is worth mentioning that the PSO2DBP algorithm suffers from an early convergence phenomenon in which it returns fast solutions but with rather low quality.

6 Conclusion and Future Works

Several real-world industrial applications require multi-dimensional resources. To run such applications, they need all the dimensions at the same time which always are scarce/expensive/pollutant resources. Inefficient resource usage incurs additional costs for cash flow even pollution in some industries. For the sake of the importance of the issue, it made sense of high motivation to find an efficient solution that led to the preparation of the current paper. This paper models the two-dimension resource allocation issue as a two-dimensional bin-packing problem to an integer linear programming which is a famous NP-Hard issue. To this end, a hybrid discrete version of the grey wolf optimization (HD-GWO) algorithm was presented which is imbalance-ness-aware in resource requests in packing objects in the bins. It also takes benefit of all available artificial intelligence approaches integrated into the proposed HD-GWO algorithm. Extensive scenarios have been conducted and the most successful state-of-the-art were selected from the literature for competitions. The simulation results prove the dominance of the proposed algorithm against other comparative algorithms in terms of eminent evaluation metrics. It revolves around the fact that it cautiously utilized discrete operators and sub-procedures so it reduces resource wastage during the optimization process in regard to the main objective. In addition, the proposed algorithm shows the high potential of scalability behavior once the size of the input is extremely huge. For future work, we envisage proposing a multi-dimensional bin-packing algorithm in online industrial applications with a limited time window and call re-deployment procedure if necessary.

Author Contributions Programming & Testing: Saeed Kosari

Blueprint, Writing, Algorithm design, Test & Analysis, and Supervisor of project: Mirsaeid Hosseini Shirvani

Conceptualization, Classification, and Advisor: Navid Khaledian

Conceptualization, Resources, Validation, Formal Analysis, Revising the Comments, Review & Writing, and Proofreading: Danial Javaheri

Funding Not Applicable.

Data Availability No datasets were generated or analysed during the current study

Declarations

Competing Interests The authors declare no competing interests.

References

1. Hormozi, E., Hu, S., Ding, Z., Tian, Y., Wang, Y., Yu, Z., Zhang, W.: Energy-efficient virtual machine placement in data centres via an accelerated Genetic Algorithm with improved fitness computation. Energy **252**, 123884 (2022). https://doi.org/10.1016/j.energy.2022.123884
2. Thabet, M., Hnich, B., Berrima, M.: A sampling-based online Co-Location-Resistant Virtual Machine placement strategy. J. Syst. Softw. **187**, 111215 (2022). https://doi.org/10.1016/j.jss.2022.111215

3. Tavana, M., Shahdi-Pashaki, S., Teymourian, E., Santos-Arteaga, F.J., Komaki, M.: A discrete cuckoo optimization algorithm for consolidation in cloud computing. Comput. Ind. Eng. (2017). https://doi.org/10.1016/j.cie.2017.12.001
4. Reddy, M.A., Ravindranath, K.: Virtual machine placement using JAYA optimization algorithm. Appl. Artif. Intell. **34**(1), 31–46 (2020). https://doi.org/10.1080/08839514.2019.1689714
5. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Stoica, I., Zaharia, M.: Above the clouds: A Berkeley view of cloud computing. University of California, Berkeley (2009)
6. Wei, C., Hu, Z.H., Wang, Y.G.: Exact algorithms for energy-efficient virtual machine placement in data centers. Future Gener. Comput. Syst. **106**, 77–91 (2020). https://doi.org/10.1016/j.future.2019.12.043
7. Gao, Y., Guan, H., Qi, Z., Hou, Y., Liu, L.: A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. J. Comput. Syst. Sci. **79**(8), 1230–1242 (2013). https://doi.org/10.1016/j.jcss.2013.02.004
8. Dashti, S.E., Rahmani, A.M.: Dynamic VMs placement for energy efficiency by PSO in cloud computing. J. Exp. Theor. Artif. Intell. **28**(12), 97–112 (2016). https://doi.org/10.1080/0952813X.2015.1020519
9. Tanir, D., Ugurlu, O., Guler, A., Nuriyev, U.: One-dimensional cutting stock problem with divisible items. J. Appl. Eng. Math. **9**(3), 473–484 (2019). https://doi.org/10.48550/arXiv.1606.01419
10. Munien, C., Ezugwu, A.E.: Metaheuristic algorithms for one dimensional bin-packing problems: a survey of recent advances and applications. J. Intell. Syst **30**, 636–663 (2021). https://doi.org/10.1515/jisys-2020-0117
11. Yuan, Y., Tole, K., Ni, F., et al.: Adaptive simulated annealing with greedy search for the circle bin packing problem. Comput. Oper. Res. **144**, 105826 (2022). https://doi.org/10.1016/j.cor.2022.105826
12. Hao, X., Zheng, L., Li, N., Zhang, C.: Integrated bin packing and lot-sizing problem considering the configuration-dependent bin packing process. Eur. J. Oper. Res. **303**(2), 581–592 (2022). https://doi.org/10.1016/j.ejor.2022.03.012
13. Wang, P., Rao, Y., Luo, Q.: An effective discrete grey wolf optimization algorithm for solving the packing problem. IEEE Access **8**, 115559–115571 (2020). https://doi.org/10.1109/ACCESS.2020.3004380
14. Ramzanpoor, Y., Shirvani, M.H., Golsorkhtabaramiri, M.: Multi-objective fault-tolerant optimization algorithm for deployment of IoT applications on fog computing infrastructure. Complex Intell. Syst. **8**, 361–392 (2022). https://doi.org/10.1007/s40747-021-00368-z
15. Taneja, M., Davy, A.: Resource-aware placement of IoT application modules in fog-cloud computing paradigm. In: Proc. of the IFIP/IEEE Symposium on Integrated Network and Service Management, IM '15, pp. 1222–1228. IEEE (2017) https://doi.org/10.23919/INM.2017.7987464
16. Brogi, A., Forti, A.: QoS-aware deployment of IoT applications through the fog. IEEE Internet Things J. **4**, 1185–1192 (2017). https://doi.org/10.1109/JIOT.2017.2701408
17. Al-Moalmi, A., Luo, J., Salah, A., Li, K.: Optimal virtual machine placement based on grey wolf optimization. Electronics **8**(3), 283 (2019). https://doi.org/10.3390/electronics8030283
18. Ghetas, M.: A multi-objective Monarch Butterfly Algorithm for virtual machine placement in cloud computing. Neural Comput. & Applic. **33**, 11011–11025 (2021). https://doi.org/10.1007/s00521-020-05559-2
19. Qin, Y., Wang, H., Yi, S., et al.: Virtual machine placement based on multi-objective reinforcement learning. Appl Intell **50**, 2370–2383 (2020). https://doi.org/10.1007/s10489-020-01633-3
20. Yu, X., Xu, W., Wu, X., et al.: Reinforced exploitation and exploration grey wolf optimizer for numerical and real-world optimization problems. Appl. Intell. **52**, 8412–8427 (2022). https://doi.org/10.1007/s10489-021-02795-4
21. Nasr, A.A., Chronopoulos, A.T., El-Bahnasawy, N.A., Attiya, G., El-Sayed, A.: A novel water pressure change optimization technique for solving scheduling problem in cloud computing. Clust. Comput. **22**, 601–617 (2019)
22. Amer, D.A., Attiya, G., Zeidan, I., Nasr, A.A.: Elite learning Harris hawks optimizer for multi-objective task scheduling in cloud computing. J. Supercomput. 1–26 (2022)
23. Nasr, A.A., El-Bahnasawy, N.A., Attiya, G., El-Sayed, A.: Using the TSP solution strategy for cloudlet scheduling in cloud computing. J. Netw. Syst. Manage. **27**, 366–387 (2019)
24. Sait, S.M., Shahid, K.S.: Optimal multi-dimensional vector bin packing using simulated evolution. J. Supercomput. **73**, 5516–5538 (2017). https://doi.org/10.1007/s11227-017-2100-0
25. Côté, J.F., Haouari, M., Iori, M.: A primal decomposition algorithm for the two-dimensional bin packing problem. Optim. Control (2019). https://doi.org/10.48550/arXiv.1909.06835
26. Zhou, Z., Shojafar, M., Alazab, M., Abawajy, J., Li, F.: AFED-EF: an energy-efficient VM allocation algorithm for IoT applications in a cloud data center. IEEE Trans. Green Commun. Netw. **5**(2), 658–669 (2021). https://doi.org/10.1109/TGCN.2021.3067309
27. Zhou, Z., Shojafar, M., Li, R., Tafazolli, R.: EVCT: an efficient VM deployment algorithm for a software-defined data center in a connected and autonomous vehicle environment. IEEE Trans. Green Commun. Netw. **6**(3), 1532–1542 (2022). https://doi.org/10.1109/TGCN.2022.3161423
28. Zhou, Z., Abawajy, J., Chowdhury, M., Hu, Z., Li, K., Cheng, H., Alelaiwi, A.A., Li, F.: Minimizing SLA violation and power consumption in cloud data centers using adaptive energy-aware algorithms. Future Gener. Comput. Syst. **86**, 836–850 (2018). https://doi.org/10.1016/j.future.2017.07.048
29. Zhou, Z., Shojafar, M., Alazab, M., Li, F.: IECL: an intelligent energy consumption model for cloud manufacturing. IEEE Trans. Industr. Inf. **18**(12), 8967–8976 (2022). https://doi.org/10.1109/TII.2022.3165085
30. Zhou, Z., Shojafar, M., Abawajy, J., Yin, H., Lu, H.: ECMS: an edge intelligent energy efficient model in mobile edge computing. IEEE Trans. Green Commun. Netw. **6**(1), 238–247 (2022). https://doi.org/10.1109/TGCN.2021.3121961

31. Zhou, Z., Li, F., Zhu, H., et al.: An improved genetic algorithm using greedy strategy toward task scheduling optimization in cloud environments. Neural Comput. Applic. **32**, 1531–1541 (2020). https://doi.org/10.1007/s00521-019-04119-7
32. Nehra, P., Kesswani, N.: Efficient resource allocation and management by using load balanced multi-dimensional bin packing heuristic in cloud data centers. J. Supercomput. (2022). https://doi.org/10.1007/s11227-022-04707-w
33. Fatima, A., Javaid, N., Sultana, T., Hussain, W., Bilal, M., Shabbir, S., Asim, Y., Akbar, M., Ilahi, M.: Virtual machine placement via bin packing in cloud data centers. Electronics **7**, 389 (2018). https://doi.org/10.3390/electronics7120389
34. Wei, W., Wang, K., Wang, K., Gu, H., Shen, H.: Multi-resource balance optimization for virtual machine placement in cloud data centers. Comput. Electr. Eng. **88**, 106866 (2020). https://doi.org/10.1016/j.compeleceng.2020.106866
35. Mirjalili, S., Mirjalili, S.M., Lewis, A.: Grey wolf optimizer. Adv. Eng. Softw. **69**, 46–61 (2014). https://doi.org/10.1016/j.advengsoft.2013.12.007
36. Hosseini Shirvani, M.: A hybrid meta-heuristic algorithm for scientific workflow scheduling in heterogeneous distributed computing systems. Eng. Appl. Artif. Intell. **90**, 1–20 (2020)
37. Dordaie, N., JafariNavimipour, N.: A hybrid particle swarm optimization and hill climbing algorithm for task scheduling in the cloud environments. ICT Press **4**(4), 199–202 (2018). https://doi.org/10.1016/j.icte.2017.08.001
38. Moschakis, I.A., Karatza, H.D.: Multi-criteria scheduling of bag-of-tasks applications on heterogeneous interlinked clouds with simulated annealing. J. Syst. Softw. (2014). https://doi.org/10.1016/j.jss.2014.11.014
39. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**, 671–680 (1983)
40. Tanha, M., Hosseini Shirvani, M.S., Rahmani, A.M.: A hybrid meta-heuristic task scheduling algorithm based on genetic and thermodynamic simulated annealing algorithms in cloud computing environments. Neural Comput. Appl. **33**, 16951–16984 (2021). https://doi.org/10.1007/s00521-021-06289-9
41. Blaglazov, A., Buyya, R.: Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurr. Comput. Pract. Exp. **24**(13), 1397–1420 (2011). https://doi.org/10.1002/cpe.1867
42. Zhou, H.: A novel approach to cloud resource management: hybrid machine learning and task scheduling. J Grid Comput. **21**, 68 (2023). https://doi.org/10.1007/s10723-023-09702-w
43. Mirjalili, S.: Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems. Neural Comput. Appl. **27**, 1053–1073 (2016). https://doi.org/10.1007/s00521-015-1920-1