Journal of Artificial Intelligence and Data Mining (JAIDM), Vol. 13, No. 1, 2025, 1-10.

Shahrood University of Technology

Journal of Artificial Intelligence and Data Mining (h)
Journal homepage: http://jad.shahroodut.ac.ir

Research paper

An Evaluation of New Meta-heuristics for Virtual Machine Placement in Cloud Data Centers

Mohsen Kiani[1] and Mohammad Reza Khayyambashi[2*]

1. Department of Computer Science, Khansar Campus, University of Isfahan, Isfahan, Iran.
2. Department of Software Engineering, Faculty of Computer Engineering, University of Isfahan, Isfahan, Iran.

Article Info

Article History:
Received 03 September 2024
Revised 20 December 2024
Accepted 19 January 2025

DOI:10.22044/jadm.2025.14987.2597

Keywords:
Cloud Computing, Coati Optimization Algorithm, Cloud Date Center, Virtual Machine

**Corresponding authot: m.r.khayyambashi@comp.ui.ac.ir (M. Khayyambashi).*

Abstract

The present study investigates the effectiveness of several new meta-heuristic (MH) methods in solving virtual machine (VM) to physical machine (PM) placement (VMP) in cloud data centers. More specifically, Coati optimization algorithm (COA) is properly adapted for solving VMP by introducing several operators for the phases of the algorithm. Several emerging and classic meta-heuristics are also included in the evaluations, including genetic algorithm, chemical reaction optimization, Harris hawk optimization (HHO), and electron valley optimizer (EVO). Two main parameters are included in our evaluations, including power consumption and resource wastage. The algorithms are evaluated in terms of their ability to reduce power consumption and resource wastage in VMP, and also in terms of their execution times. A set of evaluations with synthetic VMs are performed. The results indicate that all MHs perform almost similarly, while emerging methods (COA, HHO, EVO) have a marginal benefit.

1. Introduction

Numerous enterprises today rely on the cloud for their required services in terms of software and infrastructure. Cloud computing brings many benefits such as cost reduction and scalability. At the infrastructure level, cloud serves its users by creating virtual machines (VMs) [1], enabled by the virtualization technology [2, 3]. A VM can be executed on any of the available physical machines (PMs) in the data center of the cloud service provider. Further, a VM provides a given level of service for each resource (usually processing, memory, and storage). A VM is usually created with less resources than that of a PM. Thus, a PM can host and run more than one VM at the same time. Therefore, the way of assigning a set of VMs to a set of PMs becomes a substantial problem to solve. This problem, which is called VM to PM placement (VMP), is proven to be NP-hard [4]. VMs provided by a cloud service provider come into several flavors to serve users with different resource requirements. This is also true for PMs since PMs in a data center are not all identical. Another aspect of the problem that increases its difficulty is that the mapping of VMs needs the consideration of several resource constraints (e.g., CPU, RAM, disk, etc.). Energy consumption of the PMs in a data center is a main contributor to the cost of the data center. Since any active PM consumes energy, one main goal of a VMP method is to minimize the number of active PMs (PMs with at least one VM to execute), known as VM consolidation. Then, all inactive PMs can be turned off to reduce power consumption. Therefore, VMP can be pictured as a bin packing problem where VMs are the items, and PMs are the bins. Other metrics that are used to decide the effectiveness of a VMP method include resource utilization and resource wastage.

To solve VMP efficiently, many heuristics and meta-heuristic (MH) algorithms are developed by the researchers in the area of cloud computing. MHs have shown promising results for solving VMP [5]. Further, several new MH algorithms emerge every year that can be utilized to solve

VMP more optimally. Based on the no-free-lunch (NFL) theorem [6], no MH is superior in solving all optimization problems. This motivates us to employ every emerging MH algorithm for solving VMP. Therefore, the present study adopts several newly-proposed MH algorithms for solving VMP. The goal is to decide to what extent new meta-heuristics can benefit VMP, in terms of effectiveness (energy of the data center, resource utilization, resource wastage) and execution time.

Employing an MH for solving VMP requires to modify the algorithm for VMP by introducing a proper solution encoding (to encode a solution to VMP as a member of the population of the MH algorithm) and parameter settings in the algorithm. This is required since most MHs are first introduced for optimizing continuous functions while VMP is a discrete problem. Second, as each algorithm includes a set of steps each of which operating on the population of the algorithm to create new members iteratively, one or more operators should be provided for each step. One main focus of the present study is on the latter. A set of operators are first introduced. Then, the operators are employed in different phases of several MH algorithms. Further, this study adopts a new MH, Coati optimization algorithm (COA) to solve VMP. COA is an emerging algorithm that has shown promising results in many optimization problems, as it can properly balance between exploration and exploitation in solution space. Further, a set of proper operators are designed to adopt COA for solving VMP. Several new and classic MHs are also included in the evaluations. Then, energy consumption, resource utilization, and resource wastage of all the algorithms are reported. Moreover, the execution time of the algorithms are compared. Our evaluations show that the results provided by the MHs are generally similar yet much better than heuristics.

The rest of this paper is structured as follows. In the next section, Section 2, a brief literature review is given. Next, Section 3 discusses the preliminary materials in the field of VMP and MHs. Then, in Section 4 COA is introduced and then the modifications and required operators for COA are proposed so that it can be utilized for solving VMP. Then, the algorithm of COA is presented. Afterwards, Section 5 introduces the evaluation settings and the obtained results. The paper is then concluded in Section 6.

2. Related Work

There are plentiful studies in the field of VMP using MHs [5, 7-10]. As our work focuses on emerging MHs, here, we briefly review several similar recent studies.

Various MHs are used to solve VMP. Two scenarios are considered for VMP including static [11-13] and dynamic [14, 15] VMP. The former includes mapping a given set of VMs onto PMs, which is modeled as a constraint optimization problem. The latter models a more comprehensive system and includes more parameters such as utilization and service-level agreements. Investigating VMP as both static and dynamic have merits. In early stages of adopting an emerging algorithm, a static scenario provides the ability to examine the considered algorithms with different VM counts, which is the focus of this study.

In [11], the authors used Ant colony system (ACS) to deal with VMP, aiming at PM energy reduction in the data center and significant improvements were observed in terms of energy reduction. The work of Duan et al. [16] is another study that used ACS for VMP. Chemical Reaction Optimization (CRO) is another nature-inspired algorithm which is employed for solving VMP in [13]. Recently, Harris Hawk Optimization (HHO) is employed for VMP in [14]. HHO is adopted for VMP, which the presented evaluations confirm its benefits over heuristics and several MHs. Another recent work [17] employed Energy Valley Optimizer (EVO) for solving VMP. In the present study, we adopt COA and include recent works (HHO and EVO) and classic methods (Genetic Algorithm (GA) and CRO) in our comparison to investigate the effectiveness of recent MHs in solving VMP. Further, several new operators are devised to be used in phases of COA.

3. Problem Formulation

In the present study, five MH algorithms along with two heuristics are experimented with. The employed MHs in this work include GA, CRO [18], HHO [19], Coati Optimization Algorithm (COA [20]), and EVO [21]. EVO is inspired by the interactions of atoms at sub-atomic (electron) level. Previous studies have employed GA and CRO [12, 13], for solving VMP. Recently, EVO [17] and HHO [14] were also used for solving VMP. To the best of the authors' knowledge, this is the first research attempt in employing COA in solving VMP. This section presents a background on VMP and then introduces COA and EVO algorithms.

3.1. VM Placement

The purpose of a VMP algorithm is to optimally assign VMs to PMs. The optimality is decided based on one or more criteria, chief among which are energy consumption, resource wastage, and load balancing. Energy cost in data centers is a main concern from both monetary and environmental impact points of view. Therefore, energy consumption is always a primary concern in VMP.

The optimization goal of a VMP algorithm is to minimize power consumption and resource wastage. Power consumption of the data center is the accumulation of the power consumed by all PMs and resource wastage is first calculated for each PM and then accumulated. Therefore,

$$\min \sum_{j=0}^{N-1} \alpha_P P_j + \alpha_W W_j, \tag{1}$$

subject to

$$\sum_{j=0}^{N-1} \pi_{i,j} = 1 \quad \forall i \in \{0,\ldots,M-1\}, \tag{2}$$

$$\sum_{i=0}^{M-1} VM_i^r \times \pi_{i,j} \le PM_j^r \;\forall\, j \in \{0,\ldots,N-1\}, \quad r \in \{CPU, RAM\}. \tag{3}$$

Here, P_j and W_j are power consumption and resource wastage of the *j*th PM, respectively. In addition, α_P and α_W are the weight factors for power and resource wastage, respectively. The optimization constraint represented by (2) ensures the assignment of a VM to only one PM. The second constraint ((3)) dictates that the total resource request of all the VMs mapped to a PM must not exceed the PM's capacity (for both processing and memory resources). In the next section, we discuss the formulation required to calculate P_j and W_j. Then, In Section 3.3, we discuss how to adopt several meta-heuristics for solving VMP ((1)-(3)).

3.2. Power and Wastage Calculation

Let *M* be the number of VMs and *N* be the number of PMs in a cloud data center. Each VM is a 3-tuple in the form of $\{id, CPU, RAM\}$ where $id \in \{0,1,\ldots,M-1\}$ shows the VM's unique identifier. Further, *CPU* and *RAM* are the requested CPU and memory of the VM, respectively. Similarly, each PM is a 5-tuple $\{id, CPU, RAM, P_{\text{idle}}, P_{\max}\}$, with $id \in \{0,1,\ldots,N-1\}$ being the PM's unique identifier, *CPU* and *RAM* being the processing power and memory capacity of the PM. In addition, P_{idle} and $P_{\max}$ are the idling power consumption and maximum power consumption of the PM, respectively. The maximum power consumption is associated with the PM with full CPU utilization. A VMP algorithm allocates all VMs to a subset of PMs so that each VM is only allotted to one PM. Moreover, the accumulated resource requests of all the VMs placed on a PM must not exceed the PM's capacity. Let $\pi_{i,j}$ be a Boolean that equals one when VM_i is mapped to PM_j, and zero otherwise. The mapping solution is then shown as a matrix, as shown in (4) with *N* rows and *M* columns in each row. On the *i*th row, only the *j*th column is one and the rest of the elements on the row are zero, showing that VM_i is placed on PM_j.

$$\Pi = \begin{bmatrix} \pi_{0,0} & \cdots & \pi_{0,M-1} \\ \ldots & \ldots & \ldots \\ \pi_{N-1,0} & \cdots & \pi_{N-1,M-1} \end{bmatrix} \tag{4}$$

Resource utilization, calculated for each resource of a PM, is obtained by summing up the resource values of all the VMs placed on the PM, divided by the capacity of the PM's respective resource. Let τ_j be the set of VMs on PM_j. Then, CPU and RAM utilization values of PM_j are calculated by (5).

$$U_j^r = \frac{\sum_{i \in \tau_j} VM_i^r}{PM_j^r}, \tag{5}$$

where, $r \in \{CPU, RAM\}$ shows the resource for which the utilization is being calculated, U_j^r is the utilization value of resource *r* in PM_j, and U_i^r is the requested value of resource *r* in VM_i, a VM mapped to PM_j. For a given PM, its power consumption, which is the main optimization goal, is calculated based on (6) [22].

$$P_j = P_{\text{idle}} + U_j^{CPU}(P_{\max} - P_{\text{idle}}), \tag{6}$$

where P_j is power consumption of PM_j. Note that the total power consumption of a data center is simply calculated by summing up the power consumption of all active PMs.

Resource wastage of PM_j is calculated using (7).

$$W_j = \frac{|L_j^{CPU} - L_j^{RAM}|}{U_j^{CPU} - U_j^{RAM}} + \varepsilon \tag{7}$$

Where L_j^{CPU} and L_j^{RAM} are the residual CPU and memory resources of PM_j, respectively. Recall that U_j^{CPU} and U_j^{RAM} are CPU and memory utilization of PM_j, respectively. Further, ε is a small positive fraction (set to 0.000001).

3.3. Meta-heuristics for Solving VMP

Solving VMP, as an NP-hard problem [4], with exact methods is not feasible, especially for big numbers of VMs and PMs. Thus, heuristics and MH methods are considered. A benefit of heuristic methods is their fast execution times. On the other hand, there is usually a gap between the optimal solution and the solutions obtained by heuristic methods. What's more, a heuristic method may operate well in one situation while performing poorly in others (e.g., when the correlation between resources requested by the VMs is small). These motivate the research community to apply MH algorithms. Further, the NFL theorem motivates researchers to examine emerging MH algorithms in solving VMP. However, in order to apply an MH algorithm to a problem such as VMP, as each algorithm requires a set of operators to appropriately manipulate candidate solutions, proper operators should be adopted for different phases of the algorithm. A typical MH algorithm mainly consists of a given number iterations where each iteration uses several operators to manipulate the members of the population (candidate solutions) of the algorithm to create a better population. In the next section, solution encoding (how to represent a candidate solution to VMP) and different operators used in various MH algorithms are introduced.

3.4. Solution Encoding

An MH algorithm has a population, each of which representing a candidate solution (solution, for the sake of simplicity) to the problem. For an MH algorithm, an encoding scheme is required to represent a solution for a specific problem. A solution can be an intermediate or a final representation of a VM to PM mapping. The former, in which each solution is formed by a sequence of VM indices, is the one that needs the application of an additional operation to get to the final mapping. For instance, {5,4,2,1,3,0,7,6} is an intermediate mapping that shows a permutation of VMs and the final mapping can be obtained using a heuristic such as the first fit (FF) mapping. Conversely, the solution {{0,5,2}{7,1,4}{3,6}} represents a final mapping where VMs {0,2,5} are mapped to PM_0 and sets {7,1,4} and {3,6} are mapped to PM_1 and PM_2, respectively. Some algorithms represent solutions in the form of intermediate solutions while others use the final representation. The former simplifies the operations in terms of time complexity. Further, in the case of a final representation, applying some operators to achieve a new solution can lead to an invalid solution, which requires a complex repair operator. However, the solution space represented by an intermediate representation is more limited. This study considers the intermediate representation for all of the algorithms, where each solution is a sequence of VM indices. Then, the FF algorithm is applied to obtain the final mapping.

3.5. Operators for Intermediate Solution Encoding

A major part of employing an MH algorithm for VMP is designing proper operators for different stages of the algorithm. An operator is used to receive one or more solutions to create one or more new solutions. Each phase in an MH algorithm is either aimed at inducing exploration or exploitation. The former induces drastic changes to get new solutions while the latter attempts to perform local searches without considerable modifications. This section first introduces several existing operators proposed in the literature. Then, a set of new operators are proposed.

3.5.1. Crossover

A crossover operator creates one or more solutions out of (usually) two input solutions. Two kinds of crossover exist including single- and double-pint crossovers. Typically, in a single-point crossover, two solutions with an index, *c*, are provided. Then, the new solutions are formed by exchanging their first *c* VMs. Another version of the operator creates only one solution that is created from fractions of the two input solutions. In the double-point crossover, two indices c_1 and c_2 are provided and used to cut a portion (from index c_1 to c_2) of the solutions to exchange the portions. When a crossover is applied, the resultant solutions are likely to be invalid since some VM indices are not present while some other are repeated. This should be fixed. First, all the repeated VM indices are removed. Then, all the missing VM indices are inserted at random into the empty locations of the solutions. This creates valid solutions. Let's assume X_1={5,2,4,7,0,1,6,3} and X_2= {7,0,3,6,1,2,5,4}. A single-point crossover operator on index 2 first yields X_{new}= {5,2,4,**6**,**1**,**2**,**5**,**4**}. Then, repeated indices are removed and randomly reinserted into the solution. Removing the repeated indices results in X_{new}= {5,2,4,6,1,_,_,_} and absent VM indices of {0,3,7}. The empty locations without any VM index are shown with an "_". A sample results is X_{new}={5,2,4,6,1,0,7,3} which is obtained after randomly reinserting absent indices into the free locations.

3.5.2. VM Exchange

A new solution is formed from an existing solution by selecting two VM indices and exchanging their positions. In its simplest form, the VMs are selected at random. Another VM selection method that can induce a higher change into the solution, is exchanging the smallest VM (in terms of resources) with the biggest one. As an example, let

X_1={5,2,4,7,0,1,6,3}. A sample result would be X_{new}={5,2,**1**,7,0,**4**,6,3}, obtained by exchanging the VM indices at the third and sixth locations (highlighted with bold face). This operator can be extended so that more than one exchange take place, or each exchange includes a sequence of VMs.

3.5.3. Shift Operator

A new solution is obtained by shifting out the VM indices and reinserting them at random. A shift operator is defined by a direction (left or tight), and a count to show how many shifts should be performed. Letting X_1={5,2,4,7,0,1,6,3}, the right shift operator by one yields X_{new}={5,2,4,**3**,7,0,1,6}.

3.5.4. Rotate Operator

The rotate operator is similar to the shift operator. However, when a VM is shifted out, it goes back to the beginning of the solution (the first or the last VM location, depending on the rotation direction). Letting X_1={5,2,4,7,0,1,6,3}, applying two rotations yield X_{new}={**6**,**3**,5,2,4,7,0,1}.

3.5.5. Interval Sorting

The interval sorting operator takes two indices, denoted by c_1 and c_2, to determine an interval in the solution. Then, all the VMs located within the interval are sorted. The sort operation can be increasing or decreasing and by one or more resources. For instance, the sort can be done based on CPU or memory resource values, or a parameter that represents both CPU and memory resources. While crossover and VM exchange operators have been used in previous studies, to the best of our knowledge, we are the first to propose and employ shift and rotate operators for VMP.

4. Coati Optimization Algorithm

In this section, first, COA is introduced. Then, it is adopted for solving VMP.

4.1. COA Algorithm

COA [20] is an inspiration from Coatis' (a mammal native to South America) social behavior in hunting for preys (Iguanas), while escaping predators. Coatis gather in a pack to hunt Iguanas in a cooperative fashion. The prey is on a tree and some Coatis ascend the tree to scare the Iguana to jump from the tree to the ground where the rest of the Coatis awaiting to trap the Iguana. Further, when predators attack a Coati, they should escape and move to a safe place. In COA, the population is formed by Coatis and each Coati represents a solution to the problem. The position of the Coati shows its fitness. The position of a Coati is manipulated based on two phenomena:

1. Coatis' praying strategies to hunt Iguana
2. Coatis' escaping strategy from predators

The population of Coatis is updated through two phases (the total of three steps), as follows.

Phase 1, Step 1, Hunting strategy on tree. In Phase 1, Coatis are divided into two groups of equal size. The first group ascends the tree where the Iguana perched. At the end of this step, the Iguana scares and falls to the ground. The operator for the first step is represented by (8).

$$X_i^{P1}: x_{i,j}^{P1} = x_{i,j} + r(Iguana_j - I \times x_{i,j}),$$
$$i \in \{1, 2, \ldots, \left\lfloor \frac{n}{2} \right\rfloor\},\ j \in \{1, 2, \ldots, m\} \tag{8}$$

Here, X_i^{P1} is the new position of the Coati and $x_{i,j}^{P1}$ is the value of the jth variable in the ith Coati. Further, m and n are the dimension size and the number of Coatis (population size), respectively. In addition, $Iguana_j$ is the Coati with the best solution and r is a random in [0,1] and I is a random in [0,2].

Phase 1, Step 2, Hunting strategy. The second group of Coatis await the Iguana on the ground. When the Iguana falls to the ground, Coatis conduct a cooperative attack to finally hunt the Iguana. The position of the Coati with the best fitness value is considered as the location of the Iguana.

$$Iguana^G : Iguana_j^G = lb_j + r(ub_j - lb_j),$$
$$j \in \{1, 2, \ldots, m\}, \tag{9}$$

where $Iguana^G$ is the Iguana on the ground and $Iguana^G_j$ is the value of the jth dimension in the solution. Further, lb_j and ub_j are the lower and the upper bounds of the jth dimension in the problem, respectively. Parameter r is a random in [0,1]. Then, the position of each Coati is manipulated according to (10).

$$X_i^{P1}: x_{i,j}^{P1} = \begin{cases} x_{i,j} + r(Iguana_j - I \times x_{i,j}) & F_{Iguana^G} < F_i \\ x_{i,j} + r(Iguana_j) & \text{otherwise} \end{cases}$$
$$i \in \{\left\lfloor \frac{n}{2} \right\rfloor + 1, \ldots, n\},\ j \in \{1, 2, \ldots, m\}, \tag{10}$$

where, X_i^{P1} is the solution presented by the ith Coati and $x_{i,j}^{P1}$ is the value of the jth dimension of the ith Coati. Moreover, $Iguana_j$ is the value of the jth dimension of the Iguana on the ground. The parameters r and I are two random values in [0,1] and [0,2], respectively.

Phase 2, Escaping strategy. This phase is inspired by the behaviour of Coatis in escaping predators. When a predator attacks, the Coati moves to a safe place. This phase is to induce exploitation by employing a local search operator. To do so, the Coati moves to a location close to its current location. This operator is conducted based on (11) and (12).

$$lb_j^{local} = \frac{lb_j}{t},\ ub_j^{local} = \frac{ub_j}{t}, t \in \{1,2,...,T\}, \qquad (11)$$

where, *t* is the current iteration of the algorithm and *T* is the maximum number of iterations allowed in the algorithm. Moreover, lb_j^{local} and ub_j^{local} are the local lower and upper bounds for the *j*th variable. The new position assigned to Coati *i* in the second phase of the algorithm, denoted by X_i^{P2} is determined by (12).

$$X_i^{P2} : x_{i,j}^{P2} = x_{i,j}(1-2r) \times (lb_j^{local} + r(ub_j^{local} - lb_j^{local})), \quad i \in \{1,2,...,n\},\ j \in \{1,2,...,m\}. \qquad (12)$$

Here, *r* is a random in [0,1]. Next, if the new position optimizes the fitness of the Coati, the movement is accepted by replacing the solution of the Coati with the new solution, as presented by (13).

$$x_i = \begin{cases} X_i^{P1} & F_i^{P1} < F_i \\ X_i & \text{otherwise} \end{cases} \qquad (13)$$

Here, F_i^{P1} is the fitness of the *i*th Coati in the second phase and F_i is the old fitness of the Coati.

4.2. Adopting COA for VMP

The equations presented in the previous chapter are designed to solve functions of continuous variables with given bounds. This section presents the operators developed for COA in order to employ it for VMP, as a discrete problem.

Phase 1, Step 1, Alternate selection. As mentioned earlier, each solution to the problem is represented as a sequence of VMs. To generate a new solution for a Coati, the old solution and the best solution (Iguana) are used. The following explains the operator, called alternate selection, used to create a new Coati. VM indices are sequentially inserted into the new solution. The locations of even VM indices are defined based on that of the Iguana and the odd VM indices from that of the Coati. For instance, VM_0 in the solution of the new Coati is placed at the same location of the Iguana while VM_1 is placed at the location where VM_1 is located in the original Coati. When attempting to place a VM to a location that is already taken, it will be placed at the closest free location. For instance, let's assume X_i={1,4,5,0,2,3,7,6} and *Iguana*={7,4,6,3,1,5,2,0} be the solution of the original Coati and the Iguana, respectively. X_i^{P1} is empty at first thus X_i^{P1}= {_,_,_,_,_,_,_,_}, where each underline ("_") shows an empty location to be filled with a VM index. First, VM_0 is placed at the same location of VM_0 in *Iguana* thus we get X_i^{P1}={_,_,_,_,_,_,_,0}. The five subsequent steps lead to X_i^{P1}= {1,4,5,_,_,3,2,0}. Then, to place VM_6, which is located at the third location of the Iguana, the third location of the new solution is already filled, hence VM_6 is placed at the fourth location (the closest free location), hence we get X_i^{P1}={1,4,5,6,_,3,2,0}. Finally, VM_7 is placed at the only remaining free location to obtain X_i^{P1}={1,4,5,6,7,3,2,0}.

Phase 1, Step 2, Random cut. If the fitness of the Coati is worse than that of $Iguana^G$ (the Iguana on the ground, which is first constructed randomly), then, the operator is applied to the Coati to create a new solution. Two random numbers, r_1 and r_2 are generated in the range [0,*M*-1] and [r_1,*M*-1], respectively. Then, the chunk of $Iguana^G$ from index r_1 to r_2 is copied to the same location or the new Coati. The rest of the locations are taken from the original Coati. For instance, let us assume X_i={1,4,5,0,2,3,7,6} and *Iguana*={7,4,6,3,1,5,2,0}. Further, let r_1=1 and r_1=4. Thus, X_i^{P1}= {1,**4**,**6**,**3**,**1**,3,7,6} (the VM indices with bold font inserted from Iguana). However, this operation may result in absent and repeated VM indices, which represents an invalid solution. To construct a valid solution out of an invalid solution, a repairing operator is applied, as discussed below.

Repair operator. This operator receives a solution with repeated or absent VM indices along with r_1 and r_2 to create a valid solution. Repeated indices are replaced with absent ones. To replace a given repeated VM index with an absent one, the most similar repeated VM index is selected for replacement. Similarity is decided based on (14).

$$D_{i,i'} = \sqrt{(CPU_i - CPU_{i'})^2 + (RAM_i - RAM_{i'})^2} \qquad (14)$$

where, $D_{i,i'}$ is the calculated similarity between two VMs, VM_i and $VM_{i'}$. Further, CPU_i and $CPU_{i'}$ are respectively the CPU requests of VM_i and $VM_{i'}$. Additionally, RAM_i and $RAM_{i'}$ are memory requests of VM_i and $VM_{i'}$, respectively.

Phase 2, Selective interval sort. This step is to implement a local search. Therefore, no drastic changes are made to the solutions and only small changes are applied to slightly move the solutions in search of a better solution nearby. To do so, two random numbers are generated, called *l* and *s* so that we have $1 \le l \le k$ and $0 \le s \le M-1-l$. *k* is a controllable parameter selected between 0 and *M*-1. After generating *s* and *l*, the chunk of *l* indices in the solution, starting from index *s s* (i.e., {*s*,*s*+1,…,*s*+*l*-1}) is sorted decreasingly by the processing values of the respective VM indices in the locations. For instance, let X_i={1,4,5,0,2,3,7,6} and and CPU requests of the VMs be X_i^{CPU}=

{100,120,75,150,200,80,50,125}. Further, assume s=2 and l=4. Thus, all VM indices between location 2 and location 5 are sorted decreasingly based on the CPU values (X_i^{CPU}) of the VMs. Therefore, this operator yields the new solution as X_i^{P2}={1,4,2,0,3,5,7,6}. Finally, (15) is evaluated to decide whether the new solution is worthy to accept.

$$x_i = \begin{cases} X_i^{P2} & F_i^{P2} < F_i \\ X_i & \text{otherwise} \end{cases} \quad (15)$$

4.3. COA Algorithm

Algorithm 1 shows the used algorithm for COA. First, at the initialization phase, the population is created and the parameters are set (Line 2). Then, the algorithm runs until the termination criteria is met (here, 200 iterations). In each iteration, the population is divided in two halves. The first half performs the first phase (Iguana on tree) (Lines 5-7). This phase includes the application of the alternate selection to all members of the first half. Then, the algorithm enters the second phase (Lines 8-10), which is applied to the population members of the second half. This phase employs the introduced random cut operator. Next, the escape phase is performed (Line 11) by applying the selective interval sort operator to the new population. The algorithm returns the fittest solution as the output.

Algorithm 1. COA algorithm for VMP.

```
1  begin Algorithm
2  Pop ← initialize(VM,PM);
3  while(termination criterion not met)
4      Pop.fit ← calculateFitness(Pop);
5      for all Pop[i], $i\in\{0,\dots,\lfloor \frac{n}{2}\rfloor-1\}$
6          PopNew ← IguanaOnTree(Pop,i);
7      end for
8      for all Pop[i], $i\in\{\lfloor \frac{n}{2}\rfloor,\dots,n-1\}$
9          PopNew ← IguanaOnGround(Pop,i);
10     end for
11     PopNew ← escapePredators(PopNew);
12     Pop ← PopNew;
13 end while
14 return(CoatiWithBestFitness(Pop));
15 end Algorithm
```

5. Evaluation Results

This section gives the evaluation results of COA and other algorithms. All of the algorithms were implemented using C/C++ and ran on a system with Core™ i5 CPU and 8GB of RAM. For VM tasks, a standard synthetic task generator introduced in [23] was used. Three important parameters are provided for the task generator. The first two parameters include $\overline{R^{CPU}}$ and $\overline{R^{RAM}}$ that define reference values for CPU and memory. In this study, $\overline{R^{CPU}}$ and $\overline{R^{RAM}}$ both were set to 0.45. Further, a correlation parameter P controls the correlation between VM's CPU and memory resource values, where $P \in \{0, 0.25, 0.5, 0.75, 1\}$. P=0 shows the case where the values of CPU and memory in the VMs have strong negative correlation while P=1 indicates strong positive correlation. Our simulation results are reported for all the values of P. Further, the number of VMs was set to one of 100, 200, 500, 800, and 1200 values, and the number of PMs and VMs were the same and their resource capacities were assigned randomly.

5.1. Evaluated Algorithms and Metrics

The evaluated algorithms include two categories. The first category includes two heuristics, first-fit decreasing (FFD) and best-fit (BF). The second category includes several MHs as introduced below. The selected algorithms both include well-established ones with promising results (GA and CRO) as well as two emerging algorithms (HHO and EVO).

GA The algorithm in [12] was used. GA is a popular algorithm that has rather simple parameter settings and it is efficient in exploring search space. For crossover, the crossover operator introduced in Section 3 is used. For mutation, the exchange operator introduced in Section **3** is employed. In GA, the crossover and mutation rates were set to 0.8 and 0.1, respectively.

CRO The CRO algorithm proposed in [12] was employed. CRO is efficient but its parameter setting is rather tedious. However, as it employs four operators, it can perform well in diverse applications. The values of α , β and the initial Kinetic energy were set to 1, 0.25, and 1, respectively. Further, Molecular collision and KE loss rate were set to 0.6 and 0.2, respectively. The value for initial buffer was 5.

HHO Harris hawk optimization (HHO) is a new MH algorithm with promising performance. This algorithm was modified for VMP. The algorithm has three phases including exploration, exploitation, and attack. For the exploration phase, a crossover operator was used to combine two solutions and create a new solution. For the exploitation phase, a random is generated to decide between applying a crossover operator or a VM exchange operator. Finally, for the attack phase, multiple VM exchange operator was used. In HHO, the exploit and attack rates were set to 0.5 and 0.1, respectively.

EVO Electron valley optimizer (EVO), which has recently been used for VMP [17] was used. It should be noted that for an Alpha reaction, a single-point crossover was employed to create a new atom from an existing atom and the best atom. Further, for a Beta decay, a selective rotate or shift operator was applied where a random is generated to select between shift or rotate operation. For a Gamma reaction, a crossover operator is used to generate a new atom from the atom itself and a neighbor atom. Finally, a Positron reaction uses a shift operator on the best atom to generate a new atom.

The number of iterations and the population size were set to 100 and 200 for all MH algorithms.

In terms of evaluated metrics, we consider power consumption, resource wastage, and execution time of the algorithms.

5.2. Results and Discussion

Power consumption. Figure 1 shows the obtained results for power consumption reduction with respect to FFD. As it can be seen, when there is low correlation between CPU and memory resources in VMs, the benefit of MHs is high. On an average, GA, CRO, HHO, EVO, and COA performed 16.7%, 16.2%, 16.6%, 16.7%, and 16.5% better than FFD, respectively. Further, BF performed 14.3% better than FFD. However, as the correlation value increases, the benefit of the algorithms with regard to FFD diminishes. The average power reduction values for *P*=0.5 is 8.3%, 7.8%, 8.4%, 8.6%, and 8.2% for GA, CRO, HHO, EVO, and COA, respectively. For *P*=1.0, which represents full correlation, the benefit declines to 3.2%, 2.9%, 3.2%, 3.3%, and 3.2%.

Overall, GA performs well as it is efficient in exploring search space. Further, HHO, EVO, and COA perform slightly better than CRO.

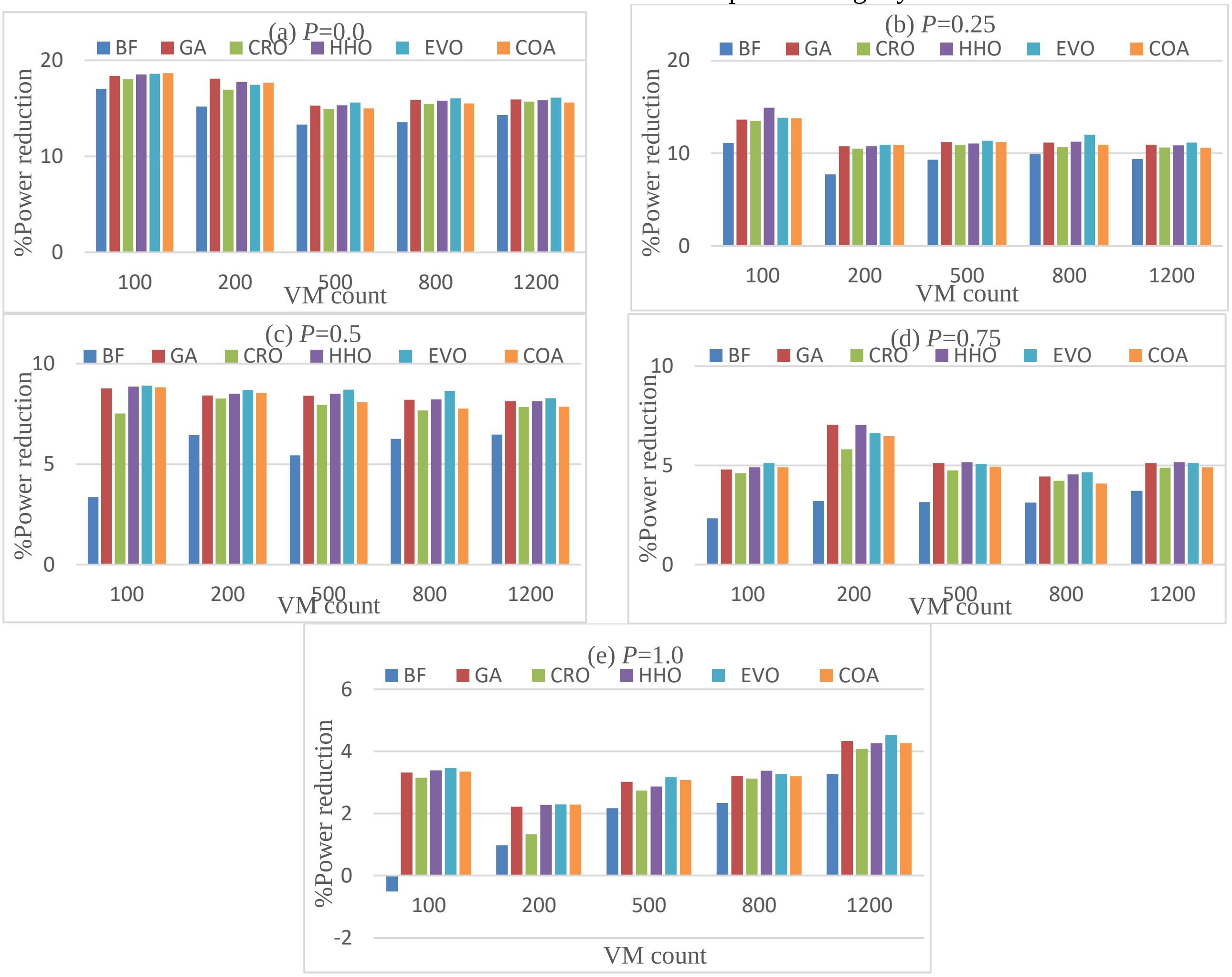

Figure 1. Power consumption reduction of different algorithms with respect to FFD for different numbers of VMs.

Resource wastage. Figure 2 gives the results of resource wastage reduction (in percent) of the algorithms, which are calculated with FFD results as the baseline. As it can be seen all algorithms perform significantly better than FFD. Note that BF has also performed well and it is to no surprise. BF attempt to fit a VM into the PM with the closest resource capacity available. Therefore, it emphasizes on reduction of resource wastage.

Execution time. Execution time of MH algorithms are presented in Figure 3. The execution times are given for *P*=0.5 and different numbers of VMs. CRO has the best scalability and lowest execution time and GA has the worst scalability. COA performed significantly better than GA, EVO, and HHO.

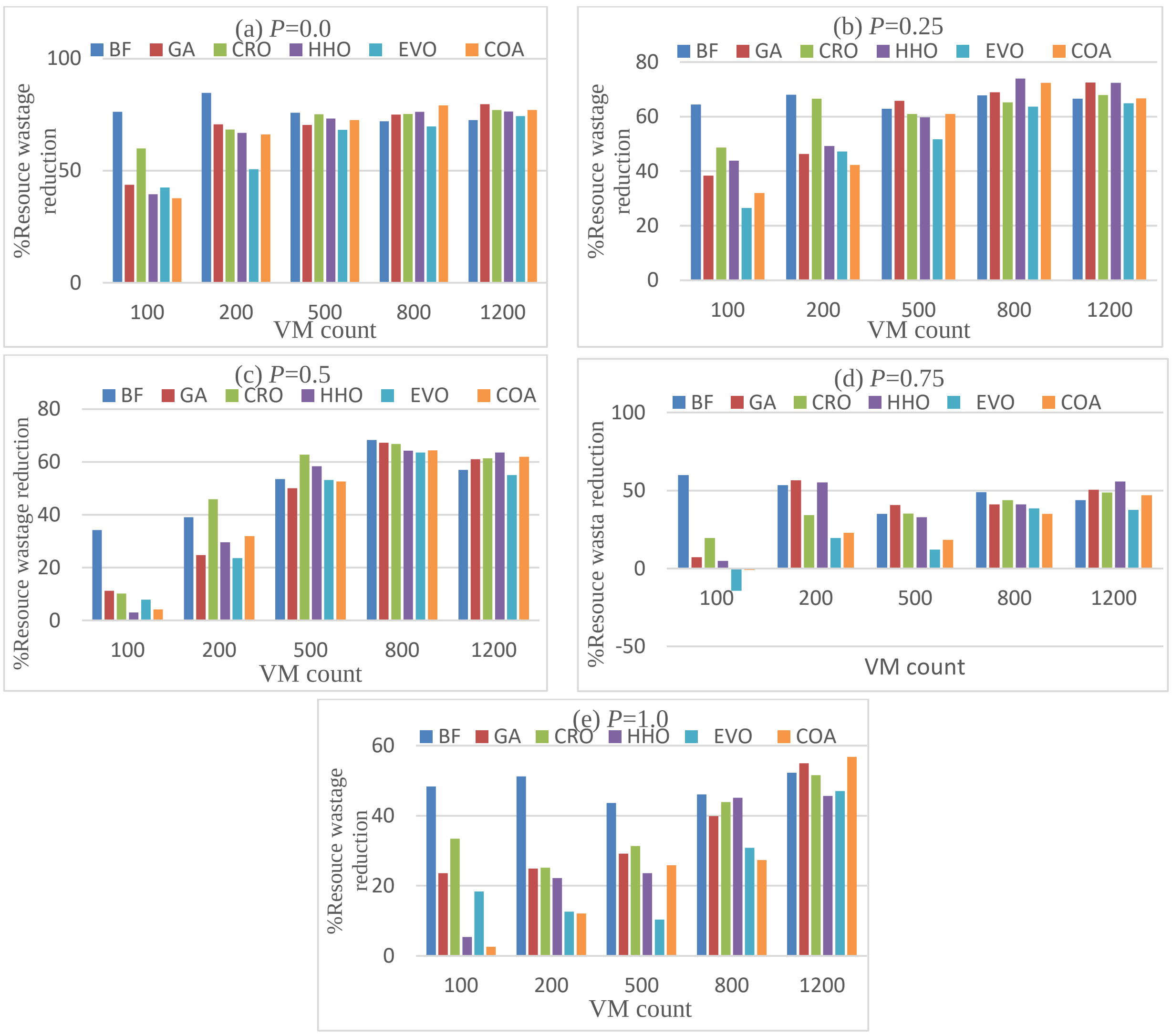

Figure 2. Resource wastage reduction of different algorithms with respect to FFD for different number of VMs.

6. Concluding Remarks

In the present study, several classic and new MHs were evaluated in solving VMP. Specifically, COA was adopted for VMP by introducing a set of operators. The evaluation results indicate that the evaluated MHs generally perform similarly in terms of power consumption reduction and outperform heuristics. However, their execution times can be different significantly, which is mainly due to the operators they use and also the phases of the algorithm. What's more, the evaluation results show that COA, EVO, and HHO as new MHs performed well with respect to GA, as a classic MH, but the benefit is not significant. One benefit of COA, which is also true about EVO, was its simple parameter settings while, e.g. for CRO, there are several parameters that should be set properly which requires many trials and errors for a specific problem.

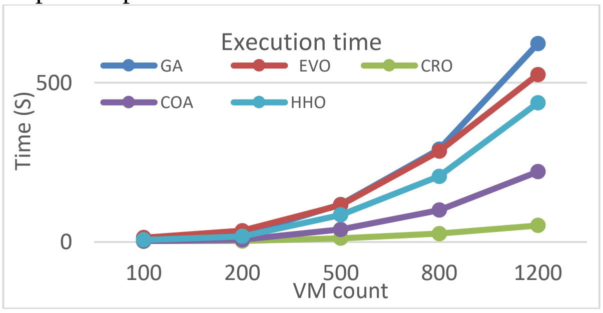

Figure 3. Running times of the MHs for various VM counts.

In terms of execution time, COR and COA performed better, showing better scalability for higher VM counts. In the future, the authors plan to implement the algorithm for solving VMP in dynamic scenarios where more important parameters such as overbooking and service level agreement (SLA) and network bandwidth are modeled. Further, investigating the effectiveness of COA in green data centers is another work planned for the future. Another future work is to design and experiment with new operators to be used in COA.

References

[1] M. Singh, "Virtualization in cloud computing-a study," in *2018 International Conference on Advances in Computing, Communication Control and Networking,* ICACCCN, 2018, pp. 64-67.

[2] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems,* vol. 25, no. 6, pp. 599-616, 2009.

[3] Y. Xing and Y. Zhan, "Virtualization and cloud computing," in *Future Wireless Networks and Information Systems, Lecture Notes in Electrical Engineering,* 2012, pp. 305-312.

[4] J. D. Ullman, "NP-complete scheduling problems," *Journal of Computer and System Sciences,* vol. 10, no. 3, pp. 384-393, 1975.

[5] J. P. Gabhane, S. Pathak, and N. M. Thakare, "Metaheuristics algorithms for virtual machine placement in cloud computing environments—a review," in *Computer Networks, Big Data and IoT*, ICCBI, 2020, pp. 329-349.

[6] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," IEEE Transactions on Evolutionary Computation, vol. 1, no. 1, pp. 67-82, 1997.

[7] B. Pourghebleh, A. Aghaei Anvigh, A. R. Ramtin, and B. Mohammadi, "The importance of nature-inspired meta-heuristic algorithms for solving virtual machine consolidation problem in cloud environments," *Cluster Computing,* vol. 24, no. 3, pp. 2673-2696, 2021.

[8] K. Rajwar, K. Deep, and S. Das, "An exhaustive review of the metaheuristic algorithms for search and optimization: taxonomy, applications, and open challenges," *Artificial Intelligence Review*, vol. 56, pp. 1-71, 2023.

[9] H. Talebian, A. Gani, M. Sookhak, A. A. Abdelatif, A. Yousafzai, A. V. Vasilakos, and F. R. Yu, "Optimizing virtual machine placement in iaas data centers: taxonomy, review and open issues," *Cluster Computing,* vol. 23, pp. 837-878, 2020.

[10] A. Safari-Bavil, S. Jabbehdari, and M. Ghobaei-Arani, "An Efficient Approach to Solve Software-defined Networks based Virtual Machines Placement Problem using Moth-Flame Optimization in the Cloud Computing Environment," *Journal of AI and Data Mining,* vol. 9, no. 3, pp. 309-320, 2021.

[11] F. Alharbi, Y.-C. Tian, M. Tang, W.-Z. Zhang, C. Peng, and M. Fei, "An ant colony system for energy-efficient dynamic virtual machine placement in data centers," *Expert Systems with Applications,* vol. 120, pp. 228-238, 2019.

[12] M. Kiani and M. R. Khayyambashi, "A network-aware and power-efficient virtual machine placement scheme in cloud datacenters based on chemical reaction optimization," *Computer Networks,* vol. 196, pp. 108270, 2021.

[13] Z. Li, Y. Li, T. Yuan, S. Chen, and S. Jiang, "Chemical reaction optimization for virtual machine placement in cloud computing," *Applied Intelligence,* vol. 49, pp. 220-232, 2019.

[14] M. HS, P. Gupta, and G. McArdle, "A Harris Hawk Optimisation system for energy and resource efficient virtual machine placement in cloud data centers," *Plos One,* vol. 18, no. 8, p. e0289156, 2023.

[15] B. Zhang, X. Wang, and H. Wang, "Virtual machine placement strategy using cluster-based genetic algorithm," *Neurocomputing,* vol. 428, pp. 310-316, 2021.

[16] L. T. Duan, J. Wang, and H. Y. Wang, "An energy-aware ant colony optimization strategy for virtual machine placement in cloud computing," *Cluster Computing,* vol. 27, no. 10, pp. 1-14, 2024.

[17] M. Kiani, "Virtual machine placement in cloud data centers using energy valley optimizer algorithm," *Tabriz Journal of Electrical Engineering*, in press, 2024.

[18] A. Y. Lam and V. O. Li, "Chemical reaction optimization: a tutorial," *Memetic Computing,* vol. 4, pp. 3-17, 2012.

[19] A. A. Heidari, S. Mirjalili, H. Faris, I. Aljarah, M. Mafarja, and H. Chen, "Harris hawks optimization: Algorithm and applications," *Future Generation Computer Systems,* vol. 97, pp. 849-872, 2019.

[20] M. Dehghani, Z. Montazeri, E. Trojovská, and P. Trojovský, "Coati optimization algorithm: A new bio-inspired metaheuristic algorithm for solving optimization problems," *Knowledge-Based Systems,* vol. 259, pp. 110011, 2023.

[21] M. Azizi, U. Aickelin, H. A. Khorshidi, and M. Baghalzadeh Shishehgarkhaneh, "Energy valley optimizer: a novel metaheuristic algorithm for global and engineering optimization," *Scientific Reports,* vol. 13, no. 1, pp. 226, 2023.

[22] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," *ACM SIGARCH Computer Architecture News,* vol. 35, no. 2, pp. 13-23, 2007.

[23] Y. Ajiro and A. Tanaka, "Improving packing algorithms for server consolidation," in *International Conference on Mathematical Geophysics*, CMG, 2007, vol. 253, pp. 399-406.

بررسی الگوریتم‌های فراابتکاری برای حل مسئله جانمایی ماشین مجازی در مراکز داده ابری

محسن کیانی[۱] و محمدرضا خیام‌باشی[۲،*]

[۱] گروه علوم کامپیوتر، دانشکده ریاضی و کامپیوتر خوانسار، دانشگاه اصفهان، اصفهان، ایران.

[۲] گروه مهندسی نرم‌افزار، دانشکده مهندسی کامپیوتر، دانشگاه اصفهان، اصفهان، ایران.

ارسال ۲۰۲۴/۰۹/۰۳؛ بازنگری ۲۰۲۴/۱۲/۲۰؛ پذیرش ۲۰۲۵/۰۱/۱۹

چکیده:

در این پژوهش، کارایی چند الگوریتم فراابتکاری در حل مسئله جانمایی ماشین‌های مجازی روی ماشین‌های فیزیکی در مراکز داده ابری مورد بررسی قرار گرفته است. به طور خاص، الگوریتم بهینه سازی کواتی با معرفی چند عملگر برای مراحل مختلف الگوریتم، برای حل مسئله جانمایی ماشین‌های مجازی انطباق یافته است. همچنین، چند الگوریتم فراابتکاری نوظهور و چند الگوریتم فراابتکاری قدیمی و محبوب، شامل الگوریتم ژنتیک، بهینه سازی مبتنی بر واکنش شیمیایی، بهینه سازی شاهین هریس، و بهینه سازی دره الکترون، مورد ارزیابی قرار گرفته‌اند. دو پارامتر اصلی، شامل توان مصرفی و هدررفت منابع، در ارزیابی‌ها بررسی شدند. الگوریتم‌های مذکور از نظر توانمندی شان در کاهش توان مصرفی و کاهش هدررفت منابع، و همچنین زمان اجرایشان در حل مسئله جانمایی ماشین‌های مجازی ارزیابی شدند. مجموعه‌ای از ارزیابی‌ها با ماشین‌های مجازی تولید شده با یک مولد وظیفه تصادفی انجام شد. نتایج مبین این واقعیت است که الگوریتم‌های فراابتکاری به طور عمومی مشابه یکدیگر عمل می‌کنند، هرچند الگوریتم‌های نوظهور کمی دارای مزیت هستند.

کلمات کلیدی: رایانش ابری، الگوریتم بهینه‌سازی کواتی، مراکز داده ابری، ماشین مجازی، بهینه‌سازی، الگوریتم‌های فراابتکاری.