Appl Intell (2016) 44:489–506
DOI 10.1007/s10489-015-0710-x

Cuckoo search based resource optimization of datacenters

Sadiq M. Sait[1] · **Abubakar Bala**[2] · **Aiman H. El-Maleh**[2]

Published online: 12 September 2015

Abstract With advancements in virtualization technology, datacenters are often faced with the challenge of managing large numbers of virtual machine (VM) requests. Due to this large amount of VM requests, it has become practically impossible to search all possible VM placements in order to find a solution that best optimizes certain design objectives. As a result, managers of datacenters have resorted to the employment of heuristic optimization algorithms for VM placement. In this paper, we employ the cuckoo search optimization (CSO) algorithm to solve the VM placement problem of datacenters. Firstly, we use the CSO to optimize the datacenter for the minimization of the number of physical machines used for placement. Secondly, we implement a multiobjective CSO algorithm to simultaneously optimize the power consumption and resource wastage of the datacenter. Simulation results show that both CSO algorithms outperform the reordered grouping genetic algorithm (RGGA), the grouping genetic algorithm (GGA), improved least-loaded (ILL) and improved FFD (IFFD) methods of VM placement.

✉ Sadiq M. Sait
sadiq@kfupm.edu.sa

Abubakar Bala
g201201620@kfupm.edu.sa

Aiman H. El-Maleh
aimane@kfupm.edu.sa

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

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

Keywords Cuckoo search · Datacenter · Lévy flight · Cloud computing · NP-Hard · Combinatorial optimization

1 Introduction

Recently, datacenters have become more robust. This is due to the employment of virtualization technology which enables the resources of a single large server to be divided into several isolated execution environments running on virtual machines. This has resulted in the creation of datacenters with fewer physical servers, high per-server utilization, higher availability, enhanced flexibility, as well as reduced hardware and operational costs. However, this flexibility provided by virtualization has caused the client base of most cloud service providers to significantly grow over the years. This has compelled large cloud service providers (such as: Amazon, Google and Microsoft) to deploy datacenters that consume huge amount of energy [1]. Consequently, the energy cost of operating and cooling infrastructure within such datacenters has significantly increased and in some cases even exceeding the cost of acquiring hardware. Recent studies have shown that the energy cost of datacenters around the world is around $20 billion [2, 3]. Thus, cutting down the energy cost of such datacenters causes a significant cost savings for both clients and providers of cloud services. Furthermore, other than enormous energy costs, rise in power consumption has been shown to cause considerable increase in heat dissipation which in turn increases hardware failure rates [4]. Hence, reducing energy consumption significantly affects the availability, productivity, as well as reliability of datacenters. Additionally, the methods of generating such large amount of energy could cause increase in carbon footprint and nuclear wastes disposed into our surroundings. There have been several attempts

to conserve energy within datacenters, which includes the employment of energy efficient hardware. However, cutting down energy wasted due to over-provisioning of hardware has been shown to provide great amount of savings [2]. In order to sustain service availability during peak resource demands, present day datacenters are typically over provisioned. However, nowadays resource requests are often bursty in nature. This causes a low average utilization of resources in the range of 15-20 % [5]. Thus, energy conservation techniques such as turning off idle servers can lead to huge energy savings.

The virtual machine placement problem is a Multi-Capacity Bin Packing Problem (MCBPP) which is a variant of the well-known bin-packing problem [6]. In such problems, there exist items of different sizes which are to be packed into bins of a given capacity, such that a minimum number of such bins are used. In the case of the VM placement problem, items to be packed are the VMs, and their sizes are the resource utilizations. The bins are the physical servers and their capacity is the utilization threshold of the servers. Finally, the dimensions are represented by the kind of resources (e.g. CPU and memory). Figure 1 shows the packing of three VMs into a single server with two dimensions, i.e., CPU and memory.

A two-level control approach (shown in Fig. 2) for automating the management of resources in datacenters was developed by Tolia et al. [7]. The local controller in Fig. 2 is responsible for estimating the amount of compute resources required by applications to guarantee their performance. This estimation usually involves either some form of approximation or profiling in which an application is run on a server for few weeks and then the peak utilization of resources are taken as the resource utilization request for such application. In general, the local controller maps applications to physical resource requirements. In contrast, the global controller (shown in Fig. 2) is in charge of final VM placement and resource allocation. The global controller receives resource utilization requests in the form of VM requests from the local controller and then finds the placement and amount of physical resource to allot to each VM. However, this initial placement carried-out by the global controller may have to change over time. This is due to

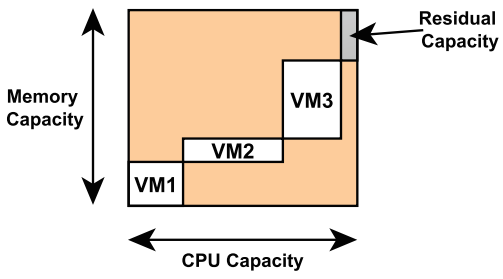


Fig. 1 A typical example of VM placement on a single server

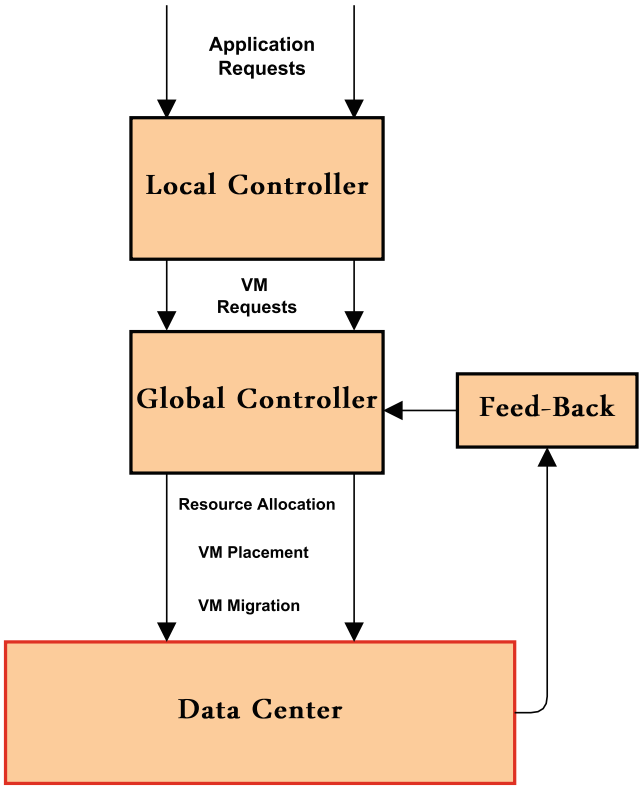


Fig. 2 Two-level control architecture for automating resource management in datacenters. [8]

the fact that workloads represented as VMs are dynamic in nature. This means that some VMs may change their resource requirements or release resources due to task completion. Moreover, additional VM requests may enter into the datacenter. In each of the aforementioned cases, the global controller should be able to receive some feedback on the condition of workloads within the datacenter. From the Feed-Back block (shown in Fig. 2), the global controller is capable of finding better placements for existing as well as incoming VM requests. VM migration technology [8] is often employed to enable the global controller perform dynamic placement of VMs. Our proposed CSO algorithm is used for VM placement, which is the job of the global controller. Thus, the CSO algorithm will be used by the global controller for both VM placement and also replacement in the case of dynamic placement of VMs. The CSO algorithm enables the global controller to find better placements for the VM requests it receives from the local controller.

The cuckoo search optimization (CSO) algorithm is inspired from the aggressive reproduction strategy of the wonderful cuckoo bird [9]. Some species of cuckoos engage in obligate brood parasitism behaviour, in which a bird (the parasite) lays its egg in the nest of another bird (the host) so that the host becomes responsible for the incubation and

hatching of the parasites' egg(s). However, some of these host birds are very vigilant and could engage the intruding bird cuckoo in direct conflict. Moreover, if the host discovers that some of the eggs in its nest are not its own, it either throws the alien ones away or discards the entire nest and builds a new one elsewhere. Other species of cuckoo such as New World Brood-Parasitic *Tapera* have evolved in such a way that they lay eggs that mimic the color and size of that of their hosts. This reduces the chance that their eggs will be identified and eventually discarded, thus increasing their reproductivity. Most parasitic cuckoos lay their eggs in fresh host nests. This increases the chance that their eggs will get hatched earlier than that of their hosts. Once the first cuckoo bird is hatched, it blindly propels other eggs out of the nest to increase its share of food provided by the host bird. The CSO algorithm employs the use of lévy flight for both local and global searching. The lévy flight is a random walk characterized by sudden jumps. Studies have shown that such characteristic is demonstrated by the flight behaviour of many insects and animals. Recent applications of such behaviour in optimization and optimal search has yielded interesting results [10]. Figure 3 shows a typical plot of the lévy flight. Assuming the imaginary rectangular border to be the design space of an optimization problem, it can be clearly seen that the lévy process randomly searches the design space with a high degree of exploitation and diversification. Walton et al. [11] presented two modifications to the original cuckoo search by Yang *et al.* [9]. These improvements enable the cuckoo search achieve faster convergence rate as well as wider application. The CSO algorithm proposed by Walton et al. [11] is shown in Fig. 4. It is based on partitioning the nests into top and bottom nests according to their cost such that the number of bottom nests is equal to P_a of the number of nests. Then each of the bottom nests is replaced using a lévy flight-based perturbation. For each of the top nests, a new nest is created either using a lévy flight-based perturbation or based on a crossover of two randomly selected top nests. If the cost of the new nest is less than a randomly selected nest among the population it replaces it.

The CSO algorithm offers many advantages over other metaheuristics. One is that its design involves the tuning of few parameters, such as the percentage of bottom nets and the generation of lévy flight step size. Moreover, the CSO algorithm together with lévy flight has been shown to outperform the genetic algorithm (GA) and particle swarm optimization (PSO) for solving complex optimization problems [9].

In this paper, we employ the cuckoo search optimization (CSO) algorithm [9] to solve the virtual machine placement problem. Firstly, we implemented a CSO-based server consolidation algorithm. The CSO for server consolidation uses a VM-based fitness measure to determine the quality

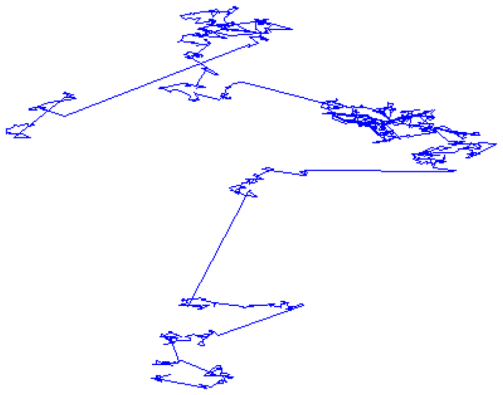


Fig. 3 A typical plot of levy flight

of solutions in the population. Moreover, we propose new perturbation functions that enable the CSO algorithm to traverse the large design space effectively. The fitness measure and perturbation functions used have shown to perform well. Furthermore, we implemented a multiobjective CSO algorithm for VM placement. In this case, we aim to simultaneously optimize the datacenter for reduced power and resource wastage. A fuzzy evaluation function is used to combine the two objectives into a single objective which we aim to optimize. To the best of our knowledge, this is the first attempt to use the cuckoo search optimization algorithm to solve the VM placement problem. Moreover, we have shown that the CSO algorithm can be used to solve both the server consolidation problem as well as multiobjective optimization of the datacenter. The rest of the paper is organised as follows. In Section 2 we discuss works related

1: $P_a \leftarrow 0.75$, $\psi \leftarrow 1.62$, $MAXiter \leftarrow 350$
2: Initialize the Population
3: Rank the entire population according to cost
4: **for** $G = 1$ **to** $MAXiter$ **do**
5: partition the population into top and bottom nests
6: **for all** X_i in bottom nests **do**
7: use Lévy flight to create a new nest X_k from X_i
8: replace X_i with X_k
9: **end for**
10: **for all** X_i such that X_i is in top nests **do**
11: Select another random top nest X_j
12: **if** $(X_i = X_j)$ **then**
13: perform Lévy flight from X_i to create a new nest X_k
14: Select a random nest X_l
15: **if** $(Cost(X_k) < Cost(X_l))$ **then**
16: replace X_l with X_k
17: **end if**
18: **else**
19: Move a distance $d_x = |X_i - X_j|/\psi$ from X_i to create a new nest X_k
20: validate X_k
21: Select a random nest X_l
22: **if** $(Cost(X_k) < Cost(X_l))$ **then**
23: replace X_l with X_k
24: **end if**
25: **end if**
26: **end for**
27: Rank the nests according to their costs
28: **end for**
29: Save the best achieved nest and its cost

Fig. 4 Modified CSO algorithm by Walton et al. [11]

to the VM placement problem. Section 3 presents the CSO algorithm for server consolidation. In Section 4, the multiobjective CSO algorithm for VM placement is presented. Finally, Section 5 concludes the paper.

2 Related work

Most algorithms designed to solve the VM placement problem can be broadly classified into two: those that attempt to optimize the datacenter for reduced number of physical machines (servers), which is referred to as server consolidation and those that attempt to optimize the datacenter for other objectives such as: power consumption, thermal dissipation and resource wastage. As stated earlier, the VM placement problem is a variant of the vector bin-packing problem; as such heuristics developed to solve the vector bin-packing problem can be adapted to solve the VM placement problem. The vector bin-packing problem has many applications, such as: scheduling, material cutting, loading and layout design [12]. However, even the simple one-dimensional vector bin-packing is known to be NP-hard [13]. Deterministic heuristics have been developed to solve the vector bin-packing problem [14]. The First-Fit-Decreasing (FFD) heuristic and Least-Loaded heuristic (LL) are among the prominent of these methods [15]. In the FFD algorithm, items to be packed are initially sorted in decreasing order of their sizes. In a single dimensional bin-packing, this sorting process is trivial. However, in the case of multidimensional bin-packing, Maruyama et al. [12] have proposed eight different methods for sorting the items according to their multidimensional sizes. Subsequently, each item in the sorted list is then packed in the first existing bin that can accommodate it. However, if all the existing bins are sequentially scanned and none could accommodate an item, a new bin is introduced into the solution to accommodate such an item and the algorithm keeps packing remaining items till all the items have been successfully packed.

In contrast, the LL algorithm tries to balance loads among the existing bins by assigning an item to the least loaded bin. Ajiro et al. [15] proposed a new technique for improving the standard FFD and LL. Experiments conducted proved that the improved versions of FFD (IFFD) and LL (ILL) outperform the standard FFD and LL algorithms. Non-deterministic algorithms that can be employed to solve the server consolidation problem include Falkenauer's grouping genetic algorithm GGA [16]. The proposed GGA was developed to address the failure of the classic genetic algorithm (GA) in solving most grouping problems. The GGA employs the use of a group-based encoding scheme rather than the individual or item-based encoding of the classic GA. However, due to this new encoding scheme, Falkenauer [16] designed new mutation and crossover operations that apply to the new encoding. An enhancement of the GGA is the reordered grouping genetic algorithm (RGGA) proposed by Wilcox et al. [17]. The RGGA employs the use of two solution encoding schemes-the group-based encoding as well as the packing sequence encoding. Moreover, RGGA employs a modified version of GGA's fitness function in order to solve Multi-Capacity Bin Packing Problems. Additionally, the authors designed new mutation and crossover functions. With equal probability, the RGGA performs three mutation functions: the swap, move and remove. Additionally, RGGA uses the steady state genetic algorithm [18]. Another form of the GGA is the exon shuffling genetic algorithm proposed by Rohlfshagen et al. [19].

In contrast, other works attempt to optimize the datacenter for objectives other than minimizing the number of physical servers used. Such objectives include: low power consumption, reduced resource wastage, and low thermal dissipation [3]. Additionally, other studies have attempted to simultaneously optimize more than one objective in what is popularly known as multi-objective optimization. Examples of such works include multi-objective ant colony system (VMPACS) of Gao et al. [20]. VMPACS optimizes the datacenter for both reduced power consumption and resource wastage. A similar work is the modified grouping genetic algorithm (MGGA) proposed by Xu et al. [8].

3 CSO for server consolidation

In this section we explain the implementation of a cuckoo search optimization (CSO) algorithm for solving the virtual machine placement problem targeting the minimization of the number of physical machines used for placement. This is often referred to as server consolidation. As discussed in Section 1, in server consolidation, VMs are placed onto the datacenter in such a way that a minimum number of physical servers are employed. Subsequently, any machine that does not host any VM is switched off in order to cut down power consumption. Moreover, this section compares the performance of the CSO algorithm for server consolidation with that of RGGA [17], GGA [16], improved least-loaded (ILL) [15] and improved FFD (IFFD) [15].

3.1 Problem definition

This section details the server consolidation problem and also elaborates on the optimization equation and constraints. In this work, the datacenter is considered to be fully virtualized such that all applications run on virtual machines. Thus the virtual machine placement problem is that of assigning

these VMs to physical machines such that certain design objectives are optimized. In the server consolidation problem the design objective is the minimization of the number of physical servers used for placement. CPU and memory dimensions are used to characterize a VM and a server node. Disk size dimension is not considered because it is assumed that network-attached-storage (NAS[1]) is used as the main storage. If a server hosts more than one VM, the CPU utilization of the server is estimated as the sum of CPU utilizations of the VMs it hosts. Similarly, the memory utilization of the server is approximated as the sum of memory utilizations of the hosted VMs. Moreover, we assume a homogeneous datacenter, where all the servers have identical capacities.

Next, the VM placement equation and constraints are defined. Suppose that there are n number of VMs that can be placed on m servers with the assumption that there exist no VM request that cannot be handled by a single server. Variables i and j represent a VM and server, respectively. Thus sets I and J represent all VMs and servers, respectively. Let ρ_i^c denote CPU request of VMi and ρ_i^m denote the memory request of VMi. Additionally, let T_{cj} and T_{mj} be the maximum capacity of CPU and memory utilization of server j, respectively. In addition we define the following two binary decision variables:

- Server allocation variable y_j, equals 1 if server j is in use and 0 otherwise.
- VM allocation variable $x_{i,j}$, equals 1 if VMi is placed in server j, and 0 otherwise.

Since the fundamental aim of the server consolidation algorithm is to place the given VMs such that the minimum number of servers is used, the placement problem can be formulated as:

Minimize $f(y) = \sum_{j=1}^{m} y_j$

subject to:

$$\sum_{i=1}^{n} \rho_i^c \cdot x_{i,j} \leq T_{cj} \cdot y_j \quad \forall j \in J \tag{1}$$

$$\sum_{i=1}^{n} \rho_i^m \cdot x_{i,j} \leq T_{mj} \cdot y_j \quad \forall j \in J \tag{2}$$

$$\sum_{j=1}^{m} x_{i,j} = 1 \quad \forall i \in I \tag{3}$$

$$y_j, x_{i,j} \in 0, 1 \quad \forall j \in J \quad \text{and} \quad \forall i \in I \tag{4}$$

Constraints (1) and (2) guarantee that the capacity threshold of each server is not exceeded. Moreover, constraint (3) ensures that a VM is placed in exactly one server. Finally, constraint (4)represents the domain of variables $x_{i,j}$ and y_j.

3.2 Proposed CSO for Server Consolidation

The CSO algorithm employs a similar chromosome representation as the GGA, where genes represent groups (servers) instead of individual items (VMs). An illustration of such encoding is: A = {4,5,6}, B = {1,2,7}, C = {3,8}, D = {0,9,10}. Where A,B,C and D represent servers and numbers 0-10 represent VMs. Thus, this nest[2] can be represented as: {4,5,6} {1,2,7} {3,8} {0,9,10}. The modified CSO algorithm for VM placement shown in Fig. 5 can be divided into three parts: the initialization phase, bottom nest perturbation and top nest perturbation. In the initialization phase, initial population and algorithm parameters are set. The population is then partitioned into top and bottom nests. In the second phase, each bottom nest is then perturbed and replaced by its perturbed version. In the top nest perturbation phase, each top nest is also perturbed and then a random nest is picked from the entire population, if the fitness of perturbed top nest is better than the randomly selected nest, it replaces it in the population. The best nest seen is then saved and the algorithm keeps iterating till the maximum iteration is reached.

In detail, the algorithm begins by setting initial parameters such as fraction of population in the bottom nests P_a and the maximum number of iterations $MAXiter$. The initial population of size S is generated by initially obtaining S random permutations of the VM requests. Subsequently, for each of these random permutations, the first fit (FF) heuristic [16] is run to obtain a placement solution. Thus, at the end we are able to obtain S different placement solutions to serve as the initial population. Next, the fitness of each nest is found by computing the average fitness of placed VMs based on the fitness of a VM given in (5). Subsequently, a procedure (line 6) commences and is repeated $MAXiter$ times. This procedure starts with the partitioning of the population into top and bottom nests. This partitioning is done by first obtaining the 75^{th} percentile of the population fitness. The first 75 % of the population that have fitness less than or equal to the 75^{th} percentile form members of the bottom nests, and the remaining nests are selected as members of the top nest. This ranking method is used in order to avoid the computationally expensive sorting used in the modified cuckoo search algorithm [11]. Subsequently, for each nest belonging to the bottom nest (line 8-12) we generate a new nest using the $Perturb\_1$ function (described in

[1] NAS device is a dedicated server used for storing and sharing files.

[2] The terms "nests" and "egg" are used interchangeably to denote a VM placement solution

1: $P_a \leftarrow 0.75$ {P_a is the fraction of population belonging to bottom nests}
2: Set $MAXiter$ {$MAXiter$ is the maximum number of iteration}
3: Generation number $G \leftarrow 1$
4: Initialize the population
5: Calculate the fitness of each nest
6: **for** $G = 1$ **to** $MAXiter$ **do**
7: Partition the population into top and bottom nests
8: **for all** X_i such that X_i is in bottom nests **do**
9: $X_j \leftarrow Perturb\_1(X_i)$
10: $X_i \leftarrow X_j$
11: $f(X_i) \leftarrow f(X_j)$
12: **end for**
13: Partition the population again into top and bottom nests
14: **for all** X_i such that X_i is in top nests **do**
15: $X_k \leftarrow Perturb\_2(X_i)$
16: Select a random nest X_l from the entire population
17: **if** $f(X_k) > f(X_l)$ **then**
18: $X_l \leftarrow X_k$
19: $f(X_l) \leftarrow f(X_k)$
20: **end if**
21: **end for**
22: Store the best nest seen so far and its fitness
23: **end for**

Fig. 5 Modified CSO algorithm for VM placement

Section 3.2.2). The new nest generated is made to replace the old bottom nest. The population is then partitioned again into top and bottom nests.

Next, the algorithm moves to the top nests procedure (line 14-21). For each top nest, a new nest X_k is generated from the current top nest by using the $Perturb\_2$ function (explained in Section 3.2.3). Subsequently, a random nest X_l is picked from the entire population. If the fitness of the newly created nest (X_k) is better than that of X_l, it replaces X_l in the population. After the top nest procedure, the best nest found so far is stored and the outer-loop keeps repeating until $MAXiter$ is reached.

It should be observed that the differences between our proposed CSO algorithm and that proposed by Walton *et al.* [11] is that we rank the population again after perturbing the bottom nests as we have found that this step improves the results. In addition, we perturb the top nests differently as illustrated in Section 3.2.3.

3.2.1 Fitness evaluation

One of the main characteristics of a good heuristic algorithm is the definition of a good fitness evaluation function. For the VM placement problem, the first fitness measure that comes to mind is the employment of number of servers used for placement, which is correct from a mathematical point of view. However, as highlighted by Falkenauer [16] this fitness measure is not practical. This is because it leads to an extremely unfriendly landscape of the design space. By using such fitness measure in a population-based heuristic like CSO algorithm, individuals (nests) in the population will all have the same fitness prematurely. Considering these limitations, a fitness measure which is item (VM) based rather than the group (server) based is employed. This fitness measure is intuitively speaking and considers the fitness of a packed VM as a function of how well it utilizes the space remaining[3] in the server. A VM that fills this remaining space well receives a high fitness while the one that fails to fill the space well gets a lower fitness. Equation 5 shows this fitness measure of a given VM. The numerator denotes the CPU and memory requests of the VM in question, while the denominator denotes the remaining CPU and memory capacity of the server assuming the VM is removed from the server. It should be noted that in the denominator, variable k cannot take the value of i.

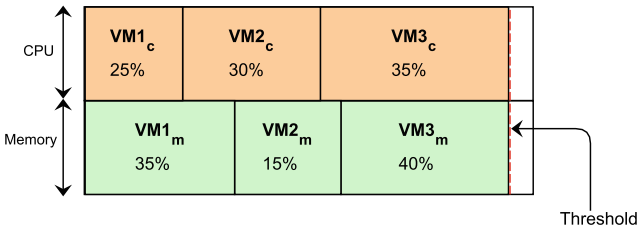


Fig. 6 perfect placement of VMs on a server

$$\frac{\rho_i^c + \rho_i^m}{\left(T_c - \sum_{k=1,k\neq i}^{n} \rho_k^c\right) + \left(T_m - \sum_{k=1,k\neq i}^{n} \rho_k^m\right)} \tag{5}$$

As an illustration, consider the placement of three VMs in a server shown in Figs. 6 and 7 with thresholds T_c and T_m set as 90 % . In the case of Fig. 6, from (5) the fitness of VM1 is 1 and so is the finesses of the other two VMs since they completely fill the server capacity. In contrast, in Fig. 7, from (5) the fitness of VM1 is computed as 0.46. This low fitness resulted because in Fig. 7 VM1 does not utilize the remaining space within the server well. Thus (5) is able to completely model how well VMs are placed within a server. Subsequently, the fitness for a server is computed as the average fitness of the VMs placed in the server. The overall fitness of a nest (placement solution) is taken to be the average fitness of all VMs contained in the solution.

3.2.2 Perturb_1 function

The CSO *perturb_1* function is executed in line 9 of the CSO algorithm. The function receives a nest as input, it then generates a number x from a lévy distribution [9]: $x = (1-u)^{-1/\alpha}$, where u is a uniform random variable in the range [0, 1] and $\alpha = G^{1/6}$ with G as generation number. Subsequently, x number of randomly selected servers are

[3] This means the residual space within the server assuming that the particular VM is removed from the server

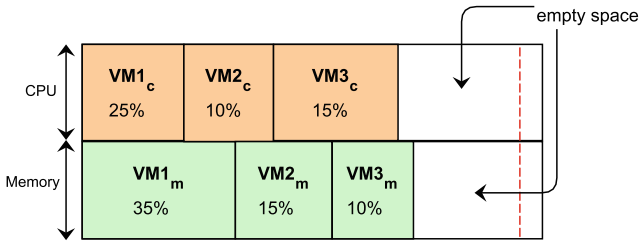


Fig. 7 Placement of VMs on a server with wasted resources

deleted from the nest. However due to this deletion, VMs contained in the deleted servers are missing from the nest and need to be reinserted. One of the 8 sorting methods of Maruyama et al. [12] is then randomly selected to sort the missing VMs. Finally, the sorted VMs are reinserted into the solution by using the first-fit (FF) heuristic and the function returns the completed solution.

3.2.3 Perturb_2 function

The *perturb_2* function is executed in line 15 of the CSO algorithm within the top nests. The function receives a single nest as input. It then finds the fitness of each server within the nest. The fitness of each server is estimated as the average fitness of the VMs it hosts (calculated from (5)). The servers are then partitioned into top and bottom groups based on their fitness. Experimental testing has shown that by selecting the best 75 % of servers as members of the top group and the remaining 25 % as bottom group, better results are obtained. In order to avoid the sorting of servers within the nest, the partitioning process adopted is similar to that of partitioning the population into top and bottom nest explained in Section 3.2. Subsequently, the servers belonging to the bottom group are deleted from the nest. The VMs in the deleted servers are then sorted according to one of the sorting methods of Maruyama et al. [12]. They are then replaced into the nest by using the first-fit (FF) heuristic. The final resulting nest is then returned by the function.

3.3 Experimental results

In this section we discuss the experimental findings. All algorithms are implemented in MATLAB R2012b (8.0.0.783) running on Windows 7 Ultimate 64-bit (6.1, build 7601). It is run on a computer with 200GB RAM and Intel(R) Xeon(R) CPU X5690 3.5GHz (24CPUs). In order to generate the dataset for this kind of experiment, all that is needed is to develop sequences of random CPU and memory requests for VMs in an experiment that has several correlations. However as pointed out by Ajiro et al. [15] there is no available standard method for generating such sequences. Thus the method adopted is to use the probability P that both the CPU and memory utilization of a server would be equal to or greater than some set reference values or that both utilizations would be less than the reference values. The pseudocode of Fig. 8 is used to generate such instances of CPU and memory utilization requests.

In the procedure above, the function rand(x) returns random numbers that are uniformly distributed in the range [0, x). R_c and R_m represent the reference CPU and memory utilization, respectively. These reference values are set as: $R_c = R_m = 25$ % and $R_c = R_m = 45$ %. P is a probability that is used to control the correlations between memory utilizations and that of CPU. The value of P is set to: 0.0, 0.25, 0.50, 0.75 and 1.0, respectively. These values correspond to strong negative, weak negative, zero, weak positive and strong positive correlations, respectively. Random VM requests are generated while incrementing the values of probability P. Thus for each reference value, five sets of VM requests are generated with each set representing different correlations. VM requests are produced for the case where the number of VM requests (n) is set to 200 and the case where it is set to 500, respectively. Thus, for each of the two cases ($n = 200$ and $n = 500$) there are five VM request sets, with each representing different correlations between the CPU and memory dimension of VM requests. For each correlation value we generated 100 instances of VM requests, e.g., for the settings $n = 200$, $R_c = R_m = 25$ % and $P = 0$ (strong negative correlation) we have 100 instances of 200 VM requests. The same is repeated for other settings. All algorithms are used to solve the placement of each of these 100 instances of VMs into physical servers and the average results are reported in Table 1. In all experiments, the resource capacity threshold (T_c and T_m) of all servers was set to 90 %.

The population size and maximumm iteration number of the CSO, RGGA [17] and GGA [16] algorithms is set to 25 and 100, respectively. The RGGA implemented is the steady state genetic algorithm [18] with crossover rate and mutation rates of 0.8 and 0.1, respectively as proposed by Wilcox et al. [17]. Similarly, the GGA implemented is a generational genetic algorithm with crossover rate of 0.8 and mutation rate of 0.1. Since the GGA fitness measure was designed to solve only single dimensional bin packing problem, the GGA implemented uses the RGGA fitness

```
for i = 1 to n
  ρ_i^c ← rand(2R_c)
  ρ_i^m ← rand(R_m)
  r = rand(1.0)
  if (r <  P and ρ_i^c ≥ R_c) or (r ≥ P and ρ_i^c <  R_c)
    ρ_i^m ← ρ_i^m + R_m
  end if
end for
```

Fig. 8 Pseudocode for Dataset. [15]

Table 1 Comparison of the CSO algorithm with other Techniques for server consolidation

			n = 200			n = 500		
Reference value	Corr.	Algorithm	m	m/LB	Time(s)	m	m/LB	Time(s)
$R_c = R_m = 25\ \%$	strong -ve	CSO	59.40	1.03	5.66	145.86	1.03	22.68
		RGGA	62.26	1.08	2.03	153.63	1.08	9.58
		GGA	61.32	1.06	5.94	152.79	1.08	25.03
		ILL	62.89	1.09	3.05	155.59	1.10	43.22
		IFFD	66.10	1.14	5.33	164.15	1.16	202.61
	weak -ve	CSO	58.98	1.02	5.64	144.31	1.02	22.64
		RGGA	60.91	1.05	1.95	149.22	1.05	8.99
		GGA	60.44	1.04	5.80	149.27	1.05	24.53
		ILL	61.60	1.06	2.26	152.69	1.08	30.88
		IFFD	64.46	1.11	4.25	160.73	1.13	176.98
	zero	CSO	58.75	1.02	5.80	143.76	1.01	23.21
		RGGA	60.44	1.05	1.96	147.67	1.04	8.85
		GGA	59.84	1.04	5.72	148.00	1.04	25.05
		ILL	60.80	1.05	1.82	150.41	1.06	25.88
		IFFD	63.87	1.11	3.94	160.00	1.13	172.56
	weak +ve	CSO	57.96	1.02	5.80	143.19	1.01	23.95
		RGGA	59.53	1.04	1.91	146.15	1.03	8.82
		GGA	59.08	1.04	5.65	146.75	1.04	25.39
		ILL	59.79	1.05	1.56	149.18	1.05	22.18
		IFFD	62.21	1.09	3.22	155.50	1.10	130.21
	strong +ve	CSO	57.74	1.01	5.83	142.19	1.01	24.30
		RGGA	58.87	1.03	1.88	144.74	1.03	8.69
		GGA	58.59	1.03	5.61	145.30	1.03	25.15
		ILL	59.20	1.04	1.30	147.75	1.05	19.26
		IFFD	60.77	1.07	2.19	150.69	1.07	86.50
$R_c = R_p = 45\%$	strong -ve	CSO	121.01	1.17	9.56	290.75	1.14	42.07
		RGGA	123.79	1.20	3.82	299.30	1.17	18.41
		GGA	122.34	1.18	10.02	297.89	1.17	50.01
		ILL	123.59	1.20	11.34	296.84	1.16	332.94
		IFFD	122.37	1.19	17.24	293.77	1.15	563.04
	weak -ve	CSO	118.48	1.15	9.25	286.27	1.12	41.08
		RGGA	121.40	1.17	3.72	294.50	1.15	18.00
		GGA	120.05	1.16	9.82	293.69	1.15	48.71
		ILL	120.85	1.17	9.56	292.10	1.14	248.56
		IFFD	120.65	1.17	15.78	291.66	1.14	458.57
	zero	CSO	116.07	1.13	8.99	278.96	1.10	39.76
		RGGA	118.96	1.15	3.65	287.73	1.13	17.66
		GGA	117.91	1.14	9.62	287.75	1.13	47.59
		ILL	118.54	1.15	8.24	285.71	1.12	247.50
		IFFD	118.71	1.15	13.74	288.40	1.13	425.66
	weak +ve	CSO	114.11	1.11	8.90	272.39	1.08	38.72
		RGGA	117.05	1.14	3.52	280.58	1.11	17.08
		GGA	116.05	1.13	9.48	281.35	1.11	46.33
		ILL	116.56	1.13	7.09	278.99	1.10	190.17
		IFFD	117.53	1.14	12.29	285.10	1.13	382.67
	strong +ve	CSO	109.64	1.07	8.66	267.71	1.06	37.93
		RGGA	112.32	1.10	3.38	274.52	1.09	16.59

Table 1 (continued)

	GGA	111.43	1.09	9.16	275.27	1.09	45.36
	ILL	111.54	1.09	4.75	271.41	1.07	102.23
	IFFD	114.02	1.12	9.48	281.37	1.11	333.76

measure which is a modification of the GGA fitness for solving multidimensional bin packing problems. In contrast, for the deterministic methods of ILL and IFFD, the maximum number of repetition (MAXR) [15] parameter was set as follows: 30 % of number of VM requests (n) for IFFD and 10 % for ILL as recommended by Ajiro et al. [15].

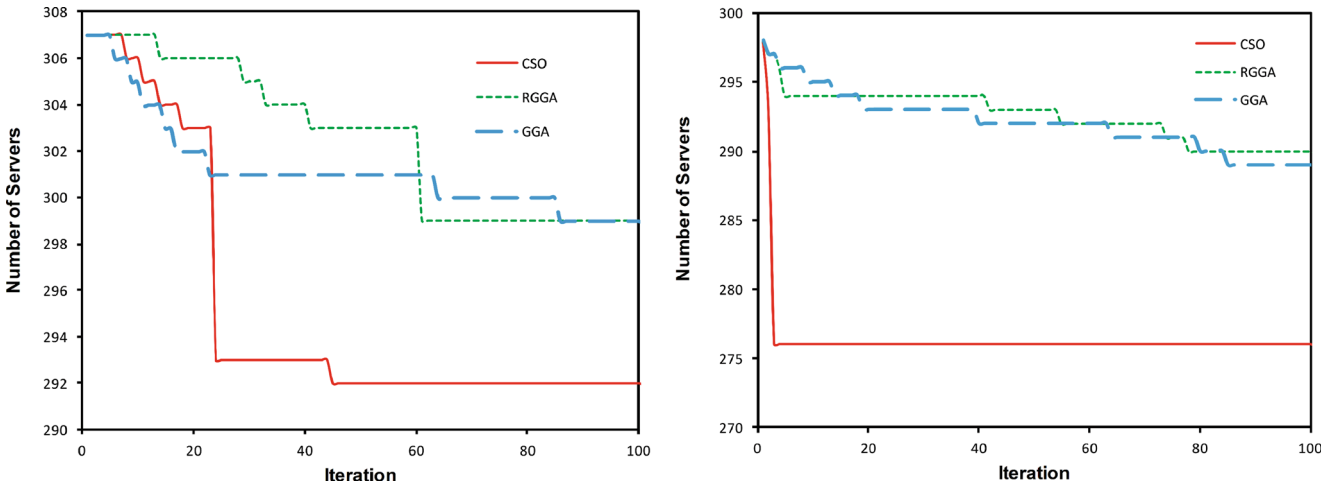
Table 1 compares the performance of the proposed CSO algorithm with that of RGGA [17], GGA [16], ILL [15] and IFFD [15] algorithms. The performance measures are: The average number of physical machines used for placement (m), average consolidation ration (m/LB) and execution time in seconds. LB is the theoretical lower bound on the number of servers that can be used for placement given as:

$$LB = \max\left\{\left\lceil\left(\sum_{i=1}^{n}\rho_i^c\right)/T_c\right\rceil, \left\lceil\left(\sum_{i=1}^{n}\rho_i^m\right)/T_m\right\rceil\right\}$$

Thus, as the number of physical machines used for placement (m) approaches the theoretical lower bound (LB) the server consolidation ratio (m/LB) converges to a value of 1. From Table 1 we observe the following:

- For both scenarios reported i.e., where $n = 200$ and $n = 500$, the CSO algorithm outperforms the RGGA, GGA, ILL and IFFD in terms of both average number of machines used for placement (m) and average server consolidation ratio (m/LB) in all reported cases. Thus, we can conclude that placement solutions obtained by employing the CSO algorithm have less number of physical machines (m) than those obtained from other reported algorithms.
- From the last column (Time (s)), it can be observed that the good results of the CSO algorithm are obtained at competitive run times.
- As correlation increases (from strong negative to strong positive), the number of machines needed for placement decreases. This is because when there is a negative correlation between the CPU and memory requests of VMs, the VM placement generally results in plenty of wastage in each server and as a result a large number of servers are needed to accommodate the VMs.
- It can be observed that more number of servers (m) are needed for placement when the reference values R_c and R_m are set to 45 % than when they are set to 25 %. This is because in the first scenario, the VM utilization requests are in the range [0,90 %) while in the second case the range is [0,50 %). Thus due to the VM sizes in the first case ($R_c = R_m = 45$ %) more number of server are needed for placement.

In the plots of Figs. 9a –10a we aim to compare the convergence rate of the proposed CSO algorithm with that



(a) Strong neg. corr. (b) Zero corr.

Fig. 9 Convergence of CSO with other Techniques

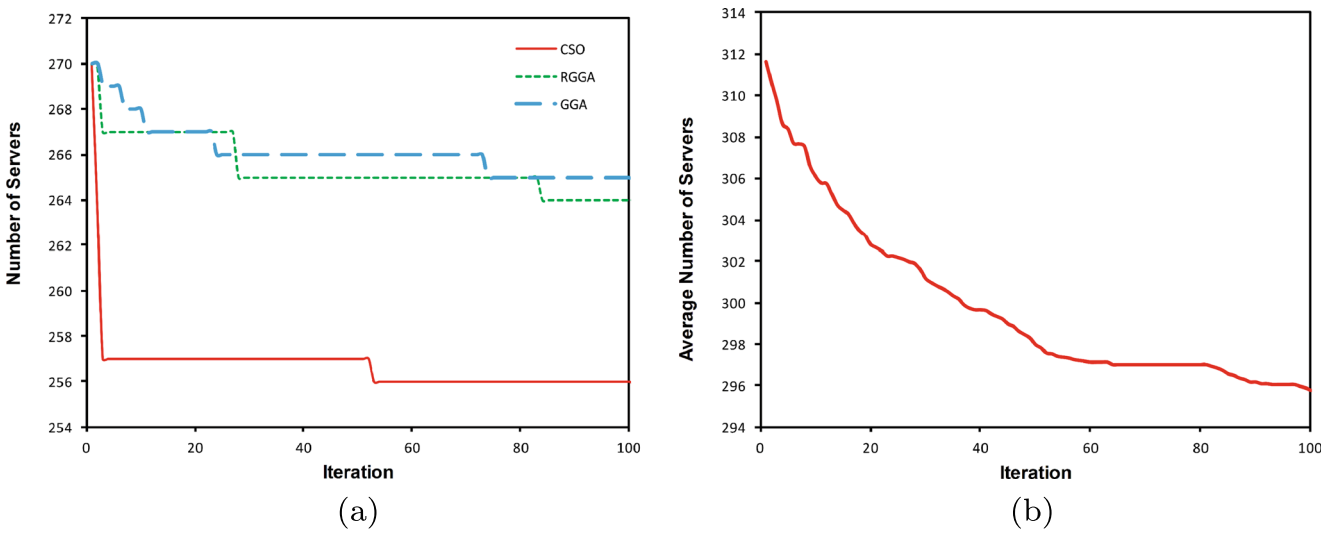


Fig. 10 **a** Convergence of CSO with other Techniques (strong post. corr.) and (**b**) Average Number of Servers Per Iteration

of the RGGA and GGA algorithms. From our generated dataset, we selected one of the 100 instances of the case where $R_c = R_m = 45\%$, $n = 500$, at $P = 0$ (strong negative correlation; Fig. 9a), $P = 3$ (zero correlation; Fig. 9b) and $P = 5$ (strong positive correlation; Fig. 10a), respectively. For each experiment, the same initial population was fed into the algorithms and each of the algorithms is run 10 times on the same initial population and then the best traces obtained are plotted. From the plots in Figs. 9a–10a we can observe that after few iterations the CSO algorithm has faster convergence rate than the RGGA and GGA algorithms. Moreover, looking at the number of servers used for placement at the end of the 100^{th} iteration, we can see that the CSO algorithm clearly outperforms the RGGA and GGA algorithms.

Similarly, Fig. 10b shows a plot of the average number of servers in the population of the CSO algorithm per iteration. The curve is for the case where: $R_c = R_m = 45\ \%, n = 500$ and $P = 0$. The main aim of this experiment is to see how the average qualities of nests evolve with iteration. From the

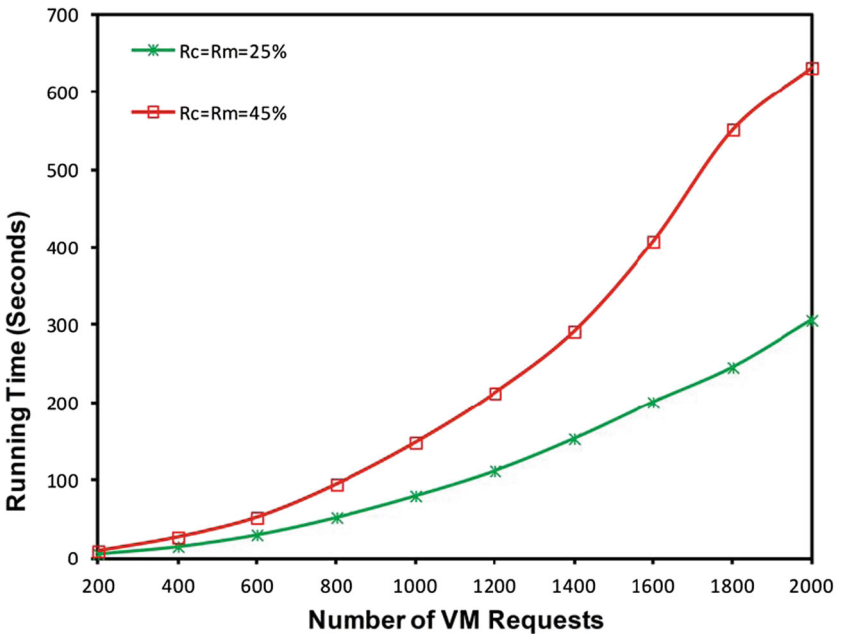


Fig. 11 Testing the CSO for Scalability

figure, it can be observed that the average number of servers per iteration does not decrease uniformly with iteration. The curves show a hill-climbing-like behavior, which is one of the attractive features of the CSO algorithm (line 8-12) that enables it to escape the local minima of most optimization problems.

In Fig. 11, the CSO algorithm is tested for scalability. The aim of this experiment is to investigate how long the CSO algorithm takes to perform the placement of a large number of VMs. In the pseudocode of Fig. 8, we set $P = 0$ i.e., strong negative correlation. This case was chosen because it is the most difficult to solve. The number of VMs (n) is then gradually incremented from 200 to 2000 in steps of 200. From the graph of Fig. 11 it can be observed that even in the case where the reference values are set to 45 %, the placement of 2000 VMs took approximately 11 minutes, which shows that the proposed CSO algorithm is scalable.

4 Multiobjective CSO for virtual machine placement

In this section we present the implementation of a multiobjective cuckoo search optimization (CSO) algorithm for solving the virtual machine placement problem. In the multiobjective CSO, the datacenter is optimized for the simultaneous minimization of power consumption and resource wastage. As highlighted in Section 1, power consumption as well as resource wastage are among the major factors that contribute to the capital cost of running large datacenters. Therefore, reducing both power consumption and resource wastage causes a significant reduction in cost as well as potential decrease in carbon footprint. The implemented multiobjective CSO is compared with reordered grouping genetic algorithm (RGGA) [17], grouping genetic algorithm (GGA) [16], improved least-loaded (ILL) [15] and improved FFD (IFFD) [15] algorithms for VM placement.

4.1 Problem definition

This section discusses the optimization equations of the multiobjective VM placement.

4.1.1 Resource wastage model

Different VM placement solutions often result in varied resource wastages on each server. Thus to fully utilize multidimensional resources, potential cost of wasted resources is computed using the following equation [20]:

$$W_j = \frac{\mid L_j^m - L_j^c \mid + \varepsilon}{U_j^m + U_j^c} \tag{6}$$

Where: W_j represents the resource wastage of the j^{th} server, U_j^m and U_j^c denotes the normalized memory and CPU resource usage (i.e., the ratio of used resource to total available resource), respectively. L_j^m and L_j^c represent the normalized remaining memory and CPU resource, respectively, while ε is a constant with value set as 0.0001. This constant (ε) is added to prevent the case where the resource wastage of a server is returned as zero. The main idea of (6) is to make efficient utilization of resources in all dimensions and balance the remaining resources on each server along different dimensions. Thus, from (6), a server with almost equal CPU and memory utilization will have a low resource wastage. This means that it has a high probability of receiving additional VMs. In contrast, when the difference between the CPU and memory utilization of a server is large, the resource utilization becomes higher, which means that such a server is less likely to accept new incoming VMs.

4.1.2 Power consumption model

Fan et al. [21] proposed a method for accurately determining the power consumption of a server from the linear relationship between power consumed and the utilization of CPU. This research has been further verified by Gao et al. [20] in experiments they conducted on a Dell server. In order to save energy, servers that are not utilized[4] are usually turned off. Thus their power consumption in idle state is not part of the total energy consumed by the CPU. The power consumed by the j^{th} server can be defined by (7) [20].

$$P_j = \begin{cases} \left[\left(\bar{p}_{busy} - \bar{p}_{idle}\right) \times U_j^c\right] + \bar{p}_{idle}, & \text{if } U_j^c > 0. \\ 0, & \text{elsewhere.} \end{cases} \tag{7}$$

Since we are considering physical servers to be homogeneous, $\bar{p}_{busy}$ and $\bar{p}_{idle}$ are the average power consumption of a server when it is fully utilized and when it is idle, respectively. Gao et al. [20] conducted experiments on a Dell server and came up with $\bar{p}_{busy} = 215$ Watts and $\bar{p}_{idle} = 162$ Watts.

4.2 Optimization equations

In this section the VM placement optimization problem is formulated. Using the same notation used in Section 3, the overall placement problem can be formulated as [20]:

[4] In the server idle-state, the CPU is not used, thus power is consumed by other server peripherals other than the CPU

$$Minimize \sum_{j=1}^{m} P_j = \sum_{j=1}^{m} \left\{ y_j \times \left[\left(\bar{p}_j^{\,busy} - \bar{p}_j^{\,idle} \right) \times \sum_{i=1}^{n} \left(x_{ij} \cdot \rho_i^c \right) + \bar{p}_j^{\,idle} \right] \right\}$$

$$Minimize \sum_{j=1}^{m} W_j = \sum_{j=1}^{m} \left\{ y_j \times \frac{\left| \left(T_{cj} - \sum_{i=1}^{n} (x_{ij} \cdot \rho_i^c) \right) - \left(T_{mi} - \sum_{i=1}^{n} (x_{ij} \cdot \rho_i^m) \right) \right| + \varepsilon}{\sum_{i=1}^{n} (x_{ij} \cdot \rho_i^c) + \sum_{i=1}^{n} (x_{ij} \cdot \rho_i^m)} \right\}$$

Subject to:

$$\sum_{j=1}^{m} x_{ij} = 1 \quad \forall i \in I \tag{8}$$

$$\sum_{i=1}^{n} \rho_i^c \cdot x_{ij} \leq T_{cj} \cdot y_j \quad \forall j \in J \tag{9}$$

$$\sum_{i=1}^{n} \rho_i^m \cdot x_{ij} \leq T_{mj} \cdot y_j \quad \forall j \in J \tag{10}$$

$$y_j, x_{ij} \in 0, 1 \quad \forall j \in J \quad \text{and} \quad \forall i \in I \tag{11}$$

4.3 Proposed multiobjective CSO for virtual machine placement

The multiobjective CSO algorithm for VM placement is similar to the CSO algorithm for server consolidation discussed in Section 3. Both algorithms have the same encoding and $Perturb\_1$ functions. However, they differ in two main areas, which are: the fitness evaluation method and $Perturb\_2$ function. While the CSO for server consolidation uses the fitness evaluation method detailed in Section 3.2.1, the multiobjective CSO uses a fuzzy evaluation method (explained in Section 4.3.1) to combine both the power consumption and resource wastage objectives into one objective which we aim to maximize. Moreover, the $Perturb\_2$ function of the CSO for server consolidation (explained in Section 3.2.3) and $Perturb\_2$ of the multiobjective CSO are the same in all aspects, except in the method used in finding the fitness of a server.

4.3.1 Fitness measure

The fitness of a particular nest or placement solution is obtained by fuzzy logic [22]. Fuzzy logic enables different criteria to be mapped into linguistic values which characterize the designer's level of satisfaction with the numerical values of the objectives [23]. These linguistic values operate over the interval $[0 \sim 1]$. The following fuzzy rule can be used to express the evaluation of a solution:

If solution x has low power consumption (lp), **AND** small resource wastage (sw) **THEN** it is a good solution. Thus, the solution with the best quality is the one with highest membership in the fuzzy sets of $\{lp, sw\}$. We use the And-like-Fuzzy-Aggregation (AFA) fuzzy rule proposed by Khan et al. [23] to obtain the fitness of solutions. However, in order to obtain the fitness of a solution we have to find its membership in fuzzy set of $\{lp\}$ and $\{sw\}$, respectively. This membership can be obtained by first defining upper and lower bound for power consumption and resource wastage.

Power consumption bounds

Lower bound The lower bound on power is the power consumed by a placement solution consisting of minimum number of servers that are required to pack VMs. This theoretical lower bound on number of servers M_{min} (LB) is computed using (12). Thus, assuming all these servers are busy, the lower bound on power is given by (13). In (13), the first term denotes the total power consumed by other server peripherals, while the second term represents the total power consumed by the CPUs of M_{min} servers.

$$M_{min} = \max \left\{ \left\lceil \left(\sum_{i=1}^{n} \rho_i^c \right) / T_c \right\rceil, \left\lceil \left(\sum_{i=1}^{n} \rho_i^m \right) / T_m \right\rceil \right\} \tag{12}$$

$$Power_{lower} = M_{min} \bar{P}_{idle} + \bar{P}_{cpu} \sum_{i=1}^{n} \left(\rho_i^c \right) \tag{13}$$

Upper bound We assume the upper bound on power consumption to be the real power consumed by the datacenter when the maximum possible number of servers is used for

placement i.e., when we have one VM per server denoted by $M_{max} \equiv$ number of VMs. Thus, the upper bound on power is:

$$Power_{upper} = M_{max}\bar{P}_{idle} + \bar{P}_{cpu}\sum_{i=1}^{n}\left(\rho_i^c\right) \tag{14}$$

$\bar{P}_{cpu} = \bar{P}_{busy} - \bar{P}_{idle}$ is the average power consumed by the CPU and ρ_i^c is the CPU utilization of VMs. T_c and T_m represent the CPU and memory utilization capacity of a server.

Wastage bounds

Lower bound In the case of resource wastage, we assume the lower bound to be the wastage calculated when we assume that there exists a large single server that can accommodate all VMs. The wastage that is generated by the placement of VMs in this large server is the lower bound on resource wastage and is given by:

$$Wastage_{lower} = \frac{\left|\sum_{i=1}^{n}\left(\rho_i^c\right) - \sum_{i=1}^{n}\left(\rho_i^m\right)\right| + \varepsilon}{\sum_{i=1}^{n}\left(\rho_i^c\right) + \sum_{i=1}^{n}\left(\rho_i^m\right)} \tag{15}$$

Upper bound The upper bound on resource wastage is the wastage obtained from the worst placement of VMs i.e., one VM per server. This wastage is given by:

$$Wastage_{upper} = \sum_{i=1}^{n}\left\{\frac{\left|\rho_i^c - \rho_i^m\right| + \varepsilon}{\rho_i^c + \rho_i^m}\right\} \tag{16}$$

Subsequently, from these computed bounds, we are able to find the membership of a solution say x in the fuzzy set of $\{lp, sw\}$. The *Fitness* function receives inputs of upper and lower bounds on power and resource wastage. Moreover, it also receives input of the resource wastage ($Wastage_x$) and power consumption ($Power_x$) of solution x computed from (6) and (7), respectively. From these inputs, the function is able to compute the membership of solution x in the set $\{sw\}$ and set $\{lp\}$ given by $\mu_{x,w}$, $\mu_{x,p}$ in (17).

$$\mu_{x,w} = \frac{Wastage_{upper} - Wastage_x}{Wastage_{upper} - Wastage_{lower}} \tag{17a}$$

$$\mu_{x,p} = \frac{Power_{upper} - Power_x}{Power_{upper} - Power_{lower}} \tag{17b}$$

By employing the equations of Khan et al. [23], we compute the weights of each objective.

Equation 18 is used to compute such weights.

$$\bar{w}_{x,w} = \frac{\bar{\mu}_{x,w}}{\bar{\mu}_{x,w} + \bar{\mu}_{x,p}} \qquad \bar{w}_{x,p} = \frac{\bar{\mu}_{x,p}}{\bar{\mu}_{x,p} + \bar{\mu}_{x,w}} \tag{18}$$

where: $\bar{\mu}_{x,w} = 1 - \mu_{x,w}$ and $\bar{\mu}_{x,p} = 1 - \mu_{x,p}$.

Figure 12 shows the graph of the normalized membership function of solution x in the set $\{sw\}$ and $\{lp\}$, respectively. From these weights we find the complement of the overall membership of solution x or fuzzy fitness (FF) in the fuzzy set $\{lp, sw\}$ as:

$$\bar{\mu}_x = (\bar{w}_{x,w})(\bar{\mu}_{x,w}) + (\bar{w}_{x,p})(\bar{\mu}_{x,p}) \tag{19}$$

Finally, we are able to obtain the fitness evaluation of solution x from (19).

$$\mu_x = 1 - \bar{\mu}_x \tag{20}$$

As an illustration of how the overall membership of a particular solution is found, take for example a nest x with overall power consumption and resource wastage calculated from (6) and (7) as ($Power_x = 300$) and ($Wastage_x = 65$), respectively. To find the membership of such nest, we start by computing upper and lower bounds from (13) through 16. Let us assume that these bounds were found to be; $Power_{lower} = 200$, $Power_{upper} = 650$, $Wastage_{lower} = 50$ and $Wastage_{upper} = 100$. Thus, from (17b) we find that for nest x, $\mu_{x,w} = 0.7$ and $\mu_{x,p} = 0.78$. Subsequently, from (18) we find the weights to be $\bar{w}_{x,w} = 0.58$ and $\bar{w}_{x,p} = 0.423$. Consequently, from (19) we find that $\bar{\mu}_x$ is 0.27. Finally, from (20) we obtain $\mu_x = 0.73$ which is

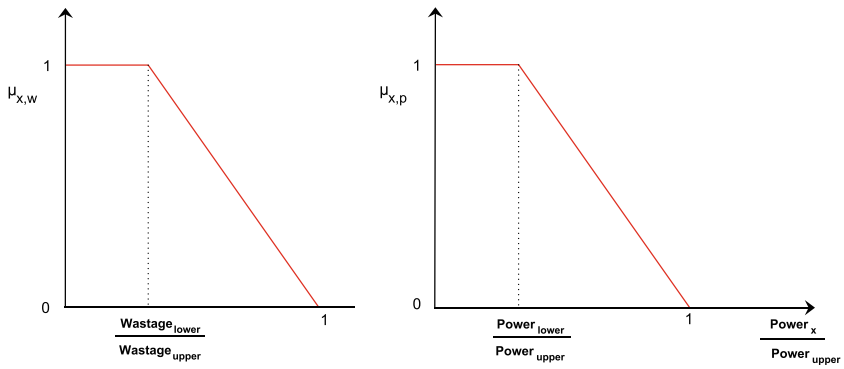


Fig. 12 Normalized Membership Function of Solution x in the Set $\{sw\}$ and $\{lp\}$

Table 2 Comparison of the multiobjective CSO algorithm with other Techniques for average power consumption and resource wastage

			n = 200				n = 500			
Reference value	Corr.	Algorithm	Power(W)	Wastage	FF ($\times 10^3$)	Time (s)	Power(W)	Wastage	FF ($\times 10^3$)	Time (s)
$R_p = R_m = 25\ \%$	strong -ve	CSO	12410	2.90	974	4.68	30491	5.48	980	18.56
		RGGA	12739	5.14	956	2.03	31514	10.22	963	9.58
		GGA	12587	4.95	959	5.94	31378	10.08	964	25.03
		ILL	12841	15.90	870	3.05	31831	39.55	872	43.22
		IFFD	13361	17.15	863	5.33	33218	42.67	865	202.61
	weak -ve	CSO	12319	2.63	975	4.63	30190	4.37	983	18.50
		RGGA	12528	4.39	959	1.95	30801	7.95	970	8.99
		GGA	12452	4.39	959	5.80	30809	7.84	970	24.53
		ILL	12640	13.85	867	2.26	31363	33.78	871	30.88
		IFFD	13103	14.97	861	4.25	32666	37.08	863	176.98
	zero	CSO	12251	1.99	978	4.60	30086	3.70	984	18.15
		RGGA	12463	3.81	958	1.96	30556	6.71	971	8.85
		GGA	12366	3.41	963	5.72	30610	6.62	971	25.05
		ILL	12521	11.67	867	1.82	31000	28.66	869	25.88
		IFFD	13019	12.82	860	3.94	32554	32.31	859	172.56
	weak +ve	CSO	12109	1.48	980	4.60	29966	3.42	982	18.11
		RGGA	12300	3.05	960	1.91	30304	5.77	970	8.82
		GGA	12227	2.76	964	5.65	30402	6.01	969	25.39
		ILL	12342	9.25	871	1.56	30795	23.10	872	22.18
		IFFD	12734	10.02	866	3.22	31819	25.61	864	130.21
	strong +ve	CSO	12060	0.98	983	4.56	29776	1.98	987	17.97
		RGGA	12198	2.00	966	1.88	30096	3.91	974	8.69
		GGA	12152	1.76	970	5.61	30187	3.91	974	25.15
		ILL	12251	5.90	895	1.30	30584	14.96	895	19.26
		IFFD	12506	6.62	885	2.19	31060	16.57	886	86.50
$R_m = R_p = 45\%$	strong -ve	CSO	24432	18.31	815	7.69	59258	37.95	850	34.19
		RGGA	24841	23.62	777	3.82	60391	48.11	816	18.41
		GGA	24607	23.63	784	10.02	60162	48.19	819	50.10
		ILL	24809	48.04	618	11.34	59992	112.95	641	332.94
		IFFD	24611	47.70	621	17.24	59495	110.74	648	563.04
	weak -ve	CSO	23983	15.05	834	7.49	58512	31.41	864	33.53
		RGGA	24423	19.81	794	3.72	59656	40.09	831	18.00
		GGA	24204	20.17	797	9.82	59525	40.58	831	48.71
		ILL	24334	41.96	615	9.56	59267	98.88	638	248.56
		IFFD	24302	41.86	616	15.78	59196	98.62	639	458.57
	zero	CSO	23608	12.44	847	7.28	57356	24.19	881	33.50
		RGGA	24049	16.39	806	3.65	58519	31.46	848	17.66
		GGA	23879	17.18	803	9.62	58522	32.77	844	47.59
		ILL	23981	35.21	620	8.24	58192	81.70	644	247.50
		IFFD	24009	35.55	617	13.74	58627	84.01	635	425.66
	weak +ve	CSO	23300	9.17	866	7.21	56223	17.41	900	32.40
		RGGA	23770	12.81	820	3.52	57346	23.73	865	17.08
		GGA	23608	13.18	819	9.48	57471	25.56	857	46.33
		ILL	23691	27.801	635	7.09	57088	64.31	657	190.17
		IFFD	23848	28.60	625	12.29	58078	67.95	641	382.67
	strong +ve	CSO	22565	5.18	904	7.01	55483	10.76	922	30.35
		RGGA	22985	7.76	862	3.38	56418	15.67	889	16.59

Table 2 (continued)

	GGA	22841	7.61	867	9.16	56540	16.58	883	45.36
	ILL	22859	19.00	676	4.75	55914	44.38	693	102.23
	IFFD	23260	19.89	665	9.48	57528	48.64	673	333.76

the overall membership value of nest x in fuzzy set of good solutions.

As mentioned earlier the $Perturb\_2$ functions of the CSO for server consolidation (Section 3) and multiobjective CSO are similar in all aspects except in the method used in finding the server fitness. In the multiobjective CSO, the function for finding server fitness operates as follows: Firstly, the function receives a server as input, say server x. Upper and lower bounds are then defined for server power and resource wastage. The lower bound on power is taken to be the power consumed by the server when it is fully loaded i.e., when the CPU utilization is 90 % (threshold). This is the best power that can be obtained since the server is fully utilized. The upper bound on power is estimated as $\tilde{P}_{idle} = 162W$ i.e., power consumed when the server is almost idle. This is the worst power that can be obtained because the server is highly underutilized. In contrast, the resource wastage lower bound is taken to be resource wastage generated when the server is fully loaded i.e. both CPU and memory utilization of the server is 90 %. The upper bound on resource wastage is taken to be the case where the resource utilization of a server is skewed i.e., the case in which one dimension is fully utilized while the other is almost zero e.g., CPU utilization is 90 % and memory utilization is 0 %. The wastage generated from such a server is taken as upper bound on server resource wastage. The function then computes the actual power and resource wastage of server x. Equation 17 is then used to find the membership of server x in fuzzy set of $\{lp\}$ and $\{sw\}$. From these membership values, (18–20) are used to obtain the overall fitness of server x in the set $\{lp, sw\}$. This membership is taken as the fitness of server x.

4.4 Experimental results

In this section, we present our experimental results on the employment of the multiobjective CSO algorithm to solve the VM placement problem of datacenters. The experimental platform and parameters used are similar to that discussed in Section 3.3. Table 2 shows comparison results of the multiobjective CSO, RGGA, GGA, ILL and IFFD algorithms. The measures for comparison are: the average power consumption (Power), average resource wastage (RW), average fuzzy fitness (FF) and CPU times (Times). From Table 2 the following points can be observed:

- In terms of average power consumption (W), in both scenarios i.e., where $n = 200$ and $n = 500$ the CSO algorithm performs better than other algorithms in all reported cases. As a result, we can deduce that placement solutions obtained by employing the proposed CSO algorithm generally have less power consumption than those obtained from RGGA, GGA, ILL, and IFFD.
- Comparing the average resource wastage, it can be seen that for both the case where $n = 200$ and $n = 500$, the CSO algorithm outperforms other algorithms in all reported results. From these results, we can say that placement solutions obtained by the proposed CSO algorithm generally have less resource wastages than those of the other reported algorithms.
- Similarly, considering the fuzzy fitness (FF), it can be observed that the CSO algorithm outperforms other algorithms in all reported cases. This shows that placement solutions found using our proposed CSO algorithm achieves better balance between the power consumption and resource wastage objective.
- As correlation increases i.e., from strong negative to strong positive, the average power and average resource wastage decrease for all algorithms. This is because as correlation increases, less number of servers are needed for VM placement thus resulting in low power consumption and resource wastage.
- More power consumption and resource wastage are generated for the case where $n = 500$ than the case where $n = 200$. This is because with more number of VMs, more servers are needed for placement and as a result more power and resource wastage is produced.

Figure 13 shows how the fuzzy fitness and average fuzzy fitness of solutions (nests) improve with iteration. Without the loss of generality, the case reported has the following values: $n = 500$, $R_c = R_m = 25$ % and $P = 0$ (strong negative correlation). However, similar results are obtained for other cases. From Fig. 13, we can see that the best fuzzy fitness curve increases with iteration. While in the case of the average fuzzy fitness curve, it can be seen that it rises and falls with iteration. This is because in the CSO algorithm (line 8-12), all bottom nests are replaced irrespective of their quality, thus the average quality of nests is bound to rise and fall with iteration. This attribute is similar to hill climbing and it is one of the attractive features of the CSO algorithm

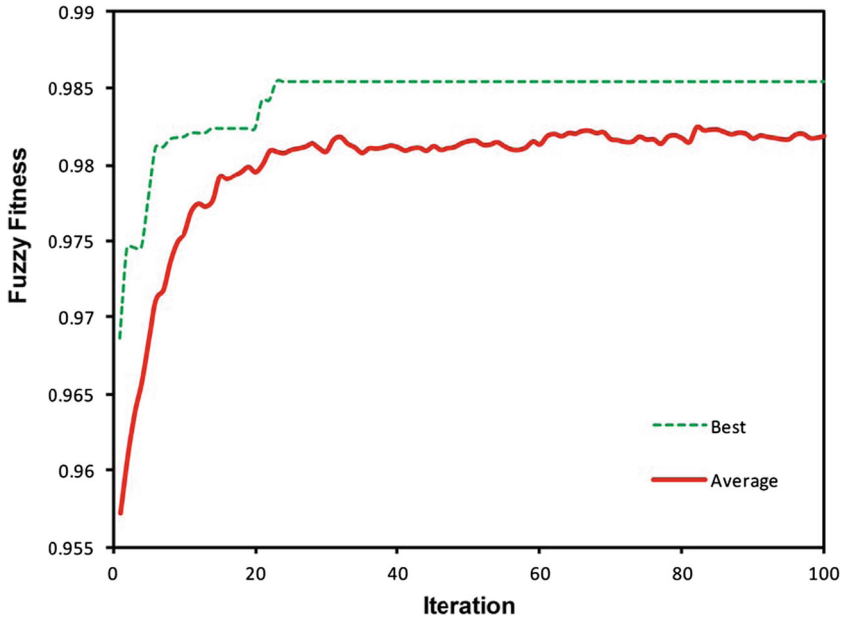


Fig. 13 Plot of best and average fuzzy fitness per iteration

that helps it escape the local minima of most optimization problems.

5 Conclusions

In this paper we have proposed a cuckoo search optimization (CSO) based algorithm for server consolidation in datacenters. In server consolidation, the datacenter is optimized for the minimization of the number of physical servers used for placement. Moreover, we have also proposed a multiobjective CSO algorithm for the simultaneous optimization of the power consumption as well as the resource wastage of datacenters. Experimental results have shown that both the CSO algorithm for server consolidation and multiobjective CSO algorithm are able to outperform the reordered grouping genetic algorithm (RGGA), grouping genetic algorithm (GGA), improved least-loaded (ILL) and improved first-fit-decreasing (IFFD) methods of VM placement. Additionally, our implemented CSO algorithm for virtual machine placement can also be employed in solving the dynamic placement of VMs in datacenters. When a batch of VM requests is received by the datacenter, the CSO algorithm can be used to resolve the placement problem in order to find a good placement for both the incoming and existing VMs. Moreover, because VM requests can timeout, it is recommended that after several weeks, the CSO algorithm should be used again to resolve the placement problem so that better placements can be found for existing VMs. However this resolving process should not be done frequently, due to the expensive nature of VM-migration. In the case that only few VMs arrive at the datacenter, it is better to use the FFD heuristic to place them.

Acknowledgements The authors acknowledge King Fahd University of Petroleum & Minerals (KFUPM) for all support. The work was conducted as part of project COE-572132-2.

References

1. Make IT Green (2010) Cloud computing and its contribution to climate change. Greenpeace International
2. Feller E, Rilling L, Morin C (2011) Energy-aware ant colony based workload placement in clouds. In: Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing, IEEE Computer Society, pp 26–33
3. Moore JD, Chase JS, Ranganathan P, Sharma RK (2005) Making scheduling cool: temperature-aware workload placement in data centers. In: USENIX annual technical conference. General Track, pp 61–75
4. Beloglazov A, Buyya R (2010) Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers. In: Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science, ACM, p 4
5. Vogels W (2008) Beyond server consolidation. Queue 6(1):20–26
6. Karmarkar N, Karp RM (1982) An efficient approximation scheme for the one-dimensional bin-packing problem. In: Proceedings of 23rd Annual Symposium on Foundations of Computer Science. IEEE, pp 312–320
7. Tolia N, Wang Z, Marwah M, Ranganathan B, Zhu X (2008) Delivering energy proportionality with non energy-proportional systems-optimizing the ensemble. HotPower 8: 2–2
8. Xu J, Fortes JAB (2010) Multi-objective virtual machine placement in virtualized data center environments. In: Proceedings of the 2010 IEEE/ACM Conference on Green Computing and Communications. IEEE, pp 179–188
9. Yang X-S, Deb S (2009) Cuckoo search via lévy flights. In: World Congress on Nature & Biologically Inspired Computing (NaBIC 2009). IEEE, pp 210–214
10. Pavlyukevich I (2007) Lévy flights, non-local search and simulated annealing. J Comput Phys 2:1830–1844

11. Walton S, Hassan O, Morgan K, Brown MR (2011) Modified cuckoo search: a new gradient free optimisation algorithm. Chaos, Solitons & Fractals 44(9):710–718
12. Maruyama K, Chang SK, Tang DT (1977) A general packing algorithm for multidimensional resource requirements. Int J Comput Inf Sci 6(2):131–149
13. Sait SM, Youssef H (1999) Iterative computer algorithms with applications in engineering: solving combinatorial optimization problems, IEEE Computer Society Press
14. Coffman Jr EG, Garey MR, Johnson DS (1984) Approximation algorithms for bin-packing-an updated survey. In: Algorithm design for computer system design, Springer, pp 49–106
15. Ajiro Y, Tanaka A (2007) Improving packing algorithms for server consolidation. In: Int. CMG Conference, pp 399–406
16. Falkenauer E (1996) A hybrid grouping genetic algorithm for bin packing. J heuristics 2(1):5–30
17. Wilcox D, McNabb A, Seppi K (2011) Solving virtual machine packing with a reordering grouping genetic algorithm. In: 2011 IEEE Congress on Evolutionary Computation (CEC). IEEE, pp 362–369
18. Luke S (2013) Essentials of Metaheuristics.Lulu, second edition. http://cs.gmu.edu/~sean/book/metaheuristics/
19. Rohlfshagen P, Bullinaria JA (2007) A genetic algorithm with exon shuffling crossover for hard bin packing problems. In: Proceedings of the 9th annual conference on genetic and evolutionary computation, ACM, pp 1365–1371
20. Gao Y, Guan H, Qi Z, Hou Y, Liu L (2013) A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. J Comput Syst Sci 79(8):1230–1242
21. Fan X, Weber W-D, Barroso LA (2007) Power provisioning for a warehouse-sized computer. In: Proceedings of the 34th Annual International Symposium on Computer Architecture, ACM, pp 13–23
22. Yager RR (1988) On ordered weighted averaging aggregation operators in multicriteria decisionmaking. IEEE Transactions on Systems Man and Cybernetics 18(1):183–190
23. Khan JA, Sait SM (2002) Fuzzy aggregating functions for multi-objective VLSI placement. In: Proceedings of the IEEE International Conference on Fuzzy Systems, vol 2. IEEE, pp 831–836

Sadiq M. Sait obtained a Bachelor's degree in Electronics from Bangalore University in 1981, and Master's and PhD degrees in Electrical Engineering from King Fahd University of Petroleum & Minerals (KFUPM), Dhahran, Saudi Arabia in 1983 & 1987 respectively. Since 1987 he has been working at the Department of Computer Engineering where he is now a Professor. In 1981 Sait received the best Electronic Engineer award from the Indian Institute of Electrical Engineers, Bangalore (where he was born). In 1990, 1994 & 1999 he was awarded the 'Distinguished Researcher Award' by KFUPM. In 1988, 1989, 1990, 1995 & 2000 he was nominated by the Computer Engineering Department for the 'Best Teacher Award' which he received in 1995, and 2000. Sait has authored over 200 research papers, contributed chapters to technical books, and lectured in over 25 countries. Sadiq M. Sait is the principle author of the books (1) VLSI PHYSICAL DESIGN AUTOMATION: Theory & Practice, published by McGraw-Hill Book Co., Europe, (and also co-published by IEEE Press), January 1995, and (2) ITERATIVE COMPUTER ALGORITHMS with APPLICATIONS in ENGINEERING (Solving Combinatorial Optimization Problems): published by IEEE Computer Society Press, California, USA, 1999. He was the Head of Computer Engineering Department, KFUPM from January 2001 - December 2004, Director of Information Technology and CIO of KFUPM between 2005 and 2011, and now is the Director of the Center for Communications and IT Research at the Research Institute of KFUPM.

Abubakar Bala is an Assistant Lecturer with Bayero University Kano, Nigeria. He holds a master's degree in computer engineering from King Fahd University of petroleum & Minerals, in 2015. Before then he had obtained a bachelor degree in computer engineering from Bayero University Kano in 2011 with first class honors. His research interests include: optimization, cloud computing and renewable energy.

Dr. Aiman El-Maleh is an Associate Professor in the Computer Engineering Department at King Fahd University of Petroleum & Minerals. He holds a Ph.D in Electrical Engineering, with the Dean's honor list, from McGill University, Canada, in 1995. He was a Member of Scientific Staff with Mentor Graphics Corp., a leader in design automation, from 1995-1998. Dr. El- Maleh's research interests are in the areas of synthesis, testing, and verification of digital systems. In addition, he has research interests in defect and soft error-tolerant design, VLSI design, design automation, and error correcting codes. He has published over seventy papers in refereed journals and conference proceedings, and holds two US patents. Dr. El-Maleh is the winner of the best paper award for the most outstanding contribution in the field of test for 1995 at the European Design & Test Conference.