Research Article

Using gravitational search algorithm enhanced by fuzzy for resource allocation in cloud computing environments

Rahim Gholami Shooli[1] · **Mohammad Masoud Javidi**[1]

Received: 6 October 2019 / Accepted: 8 January 2020 / Published online: 11 January 2020

Abstract
The aim of this paper is to allocate resources to tasks and scheduling tasks on existing virtual machines (VMs) in cloud environments, so that the time to finish the last work and average of all tasks execution time are minimized, and loads are distributed balanced on virtual machines. Since task scheduling in the cloud environment is a continuous process, so scheduling improvements, although slight, play an important role in cloud efficiency. On the other hand the resource allocation problem in cloud computing and user tasks scheduling on existing virtual machines is a NP-hard problem, and traditional algorithms requires exponential time to examine search space of this problem in sequence and finding the best answer, therefore we used Gravitational Search Algorithm (GSA) that has a high efficiency in solving nonlinear problems, for solving this problem. To do this, we create masses by combining sequences of tasks assigned to all machines. Each mass position is a solution of the problem. Then we find the best possible assignment using the gravitational search algorithm. We used fuzzy logic to determine the number of masses that affect one another during the implementation of the GSA. To calculate the cost, we use a combination of Make_ span (Time to finish the last task) and Mean_ Flow_Time (Average of all tasks execution time) and Load_ imbalance. The results show that the proposed method achieves more optimal response than genetic algorithm and GSA without fuzzy for resource allocation. It means that proposed algorithm allocated resources to tasks with less make span and mean_ flow time and more load balancing than other two algorithms.

Keywords Cloud computing · Gravitational search algorithm (GSA) · Resource allocation · Tasks scheduling · Fuzzy logic

1 Introduction

Cloud computing is a computational model based on Internet that provides a fresh model for the supply, consumption and delivery of computing services (including infrastructure, software, platform, and other computing resources) by utilizing the network. In cloud computing, resources such as disk, network, RAM, and processor, or services such as a database, according to customer needs, are provided online.

Clouds are divided into several categories based on the type of service they provide: Infrastructures as a Service (IaaS), Platforms as a Service (PaaS) and Software as a Service (SaaS). Iaas is a combination of the hardware and services needed to run the cloud. Paas is a computing platform and a set of applications to an enterprise by a cloud provider. SaaS is a software distribution that is hosted by the service provider and used by users online [1].

One of the most important issues in cloud computing is how to allocate resources (such as CPU, Memory) to user requests. Virtualization is generally used to allocate resources in the cloud environment. A tasks scheduler is used to map each task to virtual machines (VMs) to minimize a given cost function. Cost can be power/energy

✉ Mohammad Masoud Javidi, javidi@uk.ac.ir; Rahim Gholami Shooli, gholami.r@math.uk.ac.ir | [1]Department of Computer Science, Shahid Bahonar University of Kerman, Kerman, Box No. 76169133, Iran.

consumption [2] or make-span, that is, the time when finishes the latest task, or flow-time, that is, the sum of initialization times of all the tasks [3].

The problem of resource allocation in cloud computing and user tasks scheduling on existing virtual machines is a NP-hard problem [4]. So the search space for this problem is so large that if an algorithm wants to examine this space in sequence and find the best answer, it requires exponential time. Therefore, intelligent and heuristic methods and algorithms are used to solve this problem.

Main issue in cloud computing is resource deficiency. Therefore, maximizing the utilization of resources at same time minimizing the make span is an important object [5]. One way to increase utilization of resource and consequently increase cloud throughput is to avoid overloading on resources and balancing load on them. On the other hand, the service level agreement (SLA) for cloud environment uses the average response time to reflect the Quality of Service (QoS) and cloud users want this time to be minimal. Therefore, in a good scheduling method, minimizing make span (time to finish the last task) and minimizing average of all tasks execution time and maximizing load balancing should be considered.

Task scheduling and resource allocation in the cloud environment are continuous processes, so scheduling improvements, although slight, play an important role in cloud efficiency. Since the task scheduling problem is a nonlinear problem and Rashedi et al. [6] have shown that the Gravitational Search Algorithm (GSA) has a high efficiency in solving nonlinear problems, and results obtained by GSA in most cases provide superior results and in all cases are comparable with other algorithms such as Particle Swarm Optimization (PSO) and Genetic algorithm (GA), so in this study we used GSA for task scheduling to achieve mentioned purposes. To increase the accuracy of the algorithm, we used fuzzy logic in our proposed method to determine the number of masses that affect one another during the implementation of the GSA. The results show that proposed method, in comparison with the Genetic Algorithm (GA) and GSA without fuzzy enhancement, receives roughly more optimal responses for resource allocation and it allocated resources to tasks with less make span and mean_ flowtime and more load balancing than other two algorithms.

In next section, a brief summary of tasks scheduling in Cloud environments is presented. In Sect. 3, we explain necessity to improve scheduling methods and describe problem statement. Section 4 gives an overview of GSA. In Sect. 5, how using enhanced GSA by fuzzy for resource allocation is mentioned, and in Sect. 6 achieved results are given.

2 Related works

Scheduling is one of the key issues of optimization and has an important role in increasing the reliability of the system. The main purpose of the scheduling is to allocate resources to the tasks and to find the proper sequence of the tasks to execute with appropriate time [7]. Since the application of cloud computing is increasing andas mentioned finding an optimal solution for the tasks scheduling is a NP-hard problem, in recent years, tasks scheduling techniques for cloud environment received great attention from the researchers.

In [8] a scheduler has been proposed using the particle swarm optimization (PSO) algorithm to schedule tasks and allocate resources to tasks, which uses this scheduler less time consuming than the Best Resource Selection (BRS) algorithm.

Lakra and Yadav [9], proposed a multi-purpose tasks scheduling algorithm for tasks mapping to VMS to increase the efficiency of the data center and reduce costs without violating the SLA (Service Level Agreement). This method is simulated using Cloud Sim simulator and results show throughput improvement.

Li et al. [10] presented a method for scheduling tasks in cloud environments based on ant colony optimization (ACO) algorithm. The main goal of this proposed algorithm is to balance the entire system load while minimizing the time it takes to end the last task.

Priya et al [11] proposed a fuzzy multidimensional resource scheduling model to increase the resource scheduling efficiency in the cloud by introduce a resource scheduling and load balancing algorithm.

In [12], Mansouri et al. proposed a hybrid method using fuzzy system and particle swarm optimization (PSO) algorithm to increase load balancing and cloud throughput. In their study, they used fuzzy system for calculating fitness with some input factors such as tasks length, speed of CPU, size of RAM, and execution time. In their paper, the combination of crossover and mutations operators with POS algorithm is used to improve optimization performance. The experimental results show that the proposed algorithm has a better performance comparing to other methods in some terms such as imbalance degree and make span.

Jena and Mohanty [13] using Genetic algorithm to task scheduling in multi-cloud computing. The aim of their paper is to map the tasks to VMs in order to have maximum customer consent and minimum time that needs to finish the last task. They first using Genetic algorithm to map tasks to the virtual machines and then schedule tasks by using shortest job algorithm. The

results show that the proposed algorithm efficiency is more than existing algorithms.

Muthulakshmi and Somasundaram [14] integrate the simulated annealing (SA) and artificial bee colony (ABC) algorithm to scheduling the tasks according to their size and priority of the request and distance between client nodes to a server in the cloud environment. They use Cloud Sim tool too simulation the results. The results show that the proposed algorithm is more efficient in terms of reduced make span.

3 Problem statement

As mentioned in the introduction section, tasks scheduling is one of the most important issues in cloud computing. For cloud providers, a good scheduling should reduce costs and avoid overloading on resources to increase system efficiency. From the cloud user's view, make span and the average of all tasks execution time should be minimized. Therefore, to satisfy the interest of both groups, the proposed algorithm should consider minimizing make span and average of all tasks execution time and maximizing utilization of resource by balancing load on them.

Task scheduling and resource allocation in the cloud environment are continuous processes, so scheduling improvements, although slight, play an important role in cloud efficiency. Since task scheduling is a nonlinear problem and GSA has a high efficiency in solving nonlinear problems, therefore in this paper we used GSA for task scheduling to satisfy optimization constraints.

In our proposed method, physical resources are shared among several tasks using virtualization. Virtual resource requests are described by a set of parameters, including CPU, Memory and other resources requirements. The cloud provider satisfies a request by mapping virtual resources to physical ones. The resources are allocated to tasks on demand basis. Each VM can process several tasks at a time, but no two VM process the same task at a time.

Here we apply GSA to allocate virtual machines to tasks and used fuzzy logic to improve GSA Performance. We used combination of Make _span (Time to finish the last task) and Mean _Flow_Time (Average of all tasks execution time) and Load _imbalance as cost function in GSA and attempt to minimize this function value and consequently maximize fitness value of masses.

4 Gravitational search algorithm (GSA)

There are four main forces in nature. Gravity, weak force, electromagnetic force and strong force [15]. Among these forces, the gravitational force is weaker than the others, but it has the fate of the universe. The gravitational force is very comprehensive and covers the entire universe while other forces are local.

In the gravitational search algorithm (GSA), optimization is done with the aid of a plan of gravitational laws and motion in a discrete time artificial system [6]. The system environment is the same as the range of the problem definition. Under gravity law, each mass recognizes the location and condition of other masses through gravitational law. Therefore, this force can be used as a means of exchanging information.

In the first step, the system space is determined. The environment consists of a multi-dimensional coordinate system in the problem space. Every point in space is a solution to the problem. The search agents are a collection of masses. Each mass has four characteristics: (a) mass position, (b) active gravity mass, (c) inactive gravity mass, and (d) inertia mass. The amounts of gravitational and inertial masses are determined by the fitness of each mass.

After the formation of the system, the rules governing it are determined. It is assumed that only the law of gravitation and rules of motion is established. To begin, imagine the system as a collection of N masses. The position of d dimension of the mass i is represented by x_d^i(Eq. (1)). In this equation, m is the problem dimension.

$$X_i = \left(x_i^1, x_i^2, \ldots, x_i^m\right) \quad For \quad i = 1, 2, \ldots, N \tag{1}$$

To locate masses, it is assumed that in the search space, all dimensions have the same span. In this system, at time t, the mass i forces to mass j in the direction d in size $f_{ij}^d(t)$. The value of this force is calculated as Eq. (2). M_{aj} and M_{pi} are the active gravitational mass of the mass j and the passive gravitational mass of the mass i respectively. $G(t)$ is the gravitational constant at time t and R_{ij} is the distance between i and j masses. We use Euclidean distance to determine the distance between masses in accordance with Eq. (3). ε is a very small number. p is the distance exponent, which is a real number greater than one. This value is often considered to be one.

$$F_{ij}^d(t) = G(t) \times \frac{M_{pi}(t) \times M_{aj}(t)}{R_{ij}(t)^p + \varepsilon} \times \left(x_j^d(t) - x_i^d(t)\right) \tag{2}$$

$$R_{ij}(t) = ||x_i(t), x_j(t)||_2 \tag{3}$$

The force on the mass i in the direction d at time t is shown by $F_i^d(t)$ and is equal to the sum of the random coefficients of the forces that k best masses enter on the mass i (Eq. (4)). In this equation, $rand_j$ is a random number with uniform distribution in interval [0–1], which is used to maintain the randomness feature of the search algorithm.

$$F_i^d(t) = \sum_{j \in kbest, j \neq i} rand_j \times F_{ij}^d(t) \tag{4}$$

According to Newton's second law, the mass *i* accelerates in the direction of dimension *d* at time *t*, and this acceleration is calculated by Eq. (5) in which M_{ii} is the inertia mass of the mass *i*.

$$a_i^d(t) = \frac{F_i^d(t)}{M_{ii}(t)} \tag{5}$$

The next velocity of each mass is equal to the sum of the coefficients of the current velocity of the mass and the acceleration of the mass (Eq. (6)). The new position of the dimension *d* of the mass *i* is calculated by Eq. (7).

$$V_i^d(t+1) = rand_i \times V_i^d(t) + a_i^d(t) \tag{6}$$

$$x_i^d(t+1) = x_i^d(t) + V_i^d(t+1) \tag{7}$$

In the Eq. (6), $rand_i$ is a random number that distributed uniformly in the interval [0–1], which is used to maintain the randomness feature of the search algorithm.

To set the gravity constant, start from a primitive value and the value will be reduced over time. The gravitational constant, according to Eq. (8), is a function of the initial gravitational constant and time. This is true in the real world and the gravitational constant decreases very slowly over time. A suggestion for this function is to use an exponential relation to reduce the gravity constant (Eq. (9)).

$$G(t) = G\left(G_0, t\right) \tag{8}$$

$$G(t) = G_0 e^{-\alpha \frac{t}{T}} \tag{9}$$

In Eq. (9), G_0 isthe initial gravitational constant, α is a positive constant and *t* is the total of algorithmic repetitions.

In this algorithm, the gravitational and inertial masses are considered equal in accordance with Eq. (10); for their adjustment, the value of the mass target function is used (Eq. (11)). The amount of masses is normalized using Eq. (12).

$$M_{ai} = M_{pi} = M_{ii} = M_i \tag{10}$$

$$q_i(t) = \frac{fit_i(t) - worst(t)}{best(t) - worst(t)} \tag{11}$$

$$M_i(t) = \frac{q_i(t)}{\sum_{j=1}^{N} q_j(t)} \tag{12}$$

In Eq. (11), $fit_i(t)$ represents the amount of fitness of mass *i* at time *t*.

In minimizing problems, we can use Eqs. (13) and (14) to calculate the best and worst value of fitness.

$$best(t) = \min_{j \in \{1,\ldots,N\}} fit_j(t) \tag{13}$$

$$worst(t) = \max_{j \in \{1,\ldots,N\}} fit_j(t) \tag{14}$$

In maximization problems, the best and worst are defined according to Eqs. (15) and (16).

$$best(t) = \max_{j \in \{1,\ldots,N\}} fit_j(t) \tag{15}$$

$$worst(t) = \min_{j \in \{1,\ldots,N\}} fit_j(t) \tag{16}$$

At the beginning of the algorithm, each mass is randomly positioned in a point of space, which is the answer to the problem. Then, at each moment of time, the masses are evaluated, and the position of each mass is determined after calculating the Eqs. 2 to 7. Also, gravitational and inertia masses and Newton gravity constant are updated in each step according to Eqs. 8 to 16. The stop condition can be determined by the number of repetitions. The block diagram of the GSA is shown in Fig. 1 [6].

5 Using GSA enhanced by fuzzy for resource allocation in cloud computing environments

As mentioned the problem of tasks scheduling on existing virtual machines in cloud environment is a NP-hard problem. If we show the number of tasks with *Tas knum* and the number of virtual machines with *VMnum*, then the number of possible allocations is $VMnum^{Tasknum}$. The duty of scheduler is finding an allocation of virtual machines to input tasks such that the computational and memory requirements of all tasks are satisfied with the lowest cost.

This cost can be defined in a variety of ways. The most important costs are:

1. Make span: Time to finish the last task
2. Mean (Flow-Time): Average of all tasks execution time
3. Load imbalance

Cost can be a combination of these parameters.

For scheduling, we can use traditional algorithms such as Round Robin algorithm, but because they have a serial structure, offer only possible responses that are not

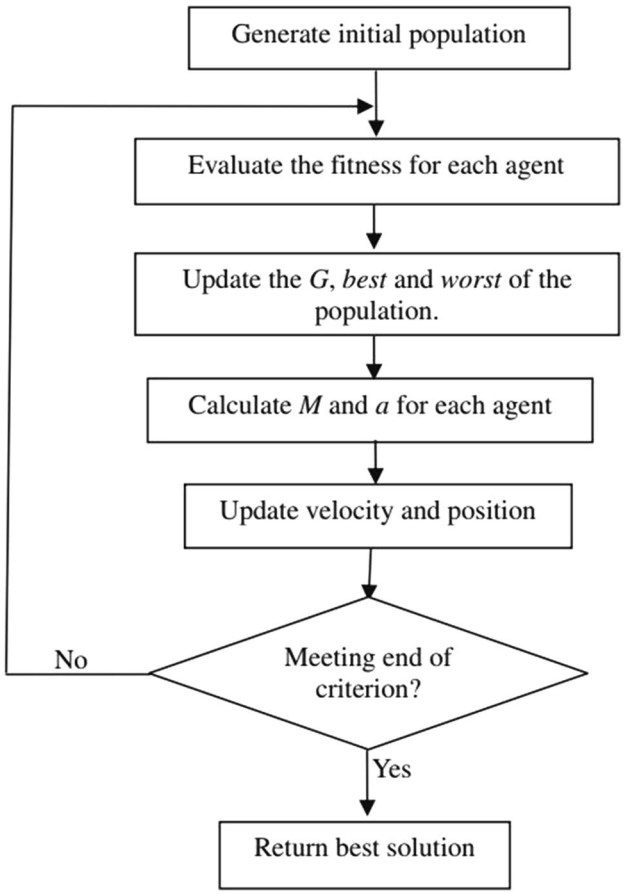


Fig. 1 Block diagram of the GSA

necessarily optimal or relatively optimal. Also do not meet the load balance criterion.

As noted above, the problem of scheduling and allocating resources in the cloud is a NP-Hard problem and due to the size of the search space, the time needed to check the entire space sequentially and find the best answer is required exponential time. Because of the good performance of GSA in nonlinear problems, we used this algorithm for tasks scheduling in Cloud environment.

First, to code problem responses, we create a sequence of tasks assigned to each machine and then we combine the created sequences to each other and store in an array named *Allocated*. The length of this array is equal to the number of tasks (*Task num*), and in each section, allocated task numbers in each machine are stored. Also, in an array named *Tasks In VM* with the length of the number of virtual machines (*V Mnum*), the number of tasks assigned to each machine is kept. Figure 2 shows an example of coding for the 10 tasks scheduling on 6 machines. In this figure as mentioned in problem state section, each VM can process several tasks at a time, for example task#1 and task#3 are executed on VM#1 and task#6, task#8 and task#10 are executed on VM#5. Since we have 10 tasks here, so the length of *Allocated* array is 10. The *Allocated* array, along with the *Tasks InV M*array, determines each task executes on which virtual machine. For example, *Tasks InVM*[1] = 2 means that two tasks are running on VM#1, and since *Allocated*[1] = 1 *and Allocated*[2] = 3, so tasks 1 and 3 are running on VM#1. Likewise *TasksInVM*[2] = 1 means that one task is running on VM#2, and since two tasks are running on VM#1, therefore the first two cells of the *Allocated* array store the tasks numbers that runs on the VM#1, and the third cell shows the executed task number on the VM#2. Here*Allocated*[3]= 2 and it means that task#2is running on VM#2.

If the number of available resources is equal to *Resnum*, then the capacities of the virtual machines in an array called *VMsCapacity* with *Resnum* × *VMnum* dimensions are stored. The column *i* of this array shows the amount of *i-th* virtual machine resources. Also resource requirements for tasks are stored in an array named *Tasks Requirements* with dimensions *Resnum* × *Tasknum*. The j column of this array shows the amount of resources requested by *j-th*task. The structure of the *VMs Capacity* and *Tasks Requirements* arrays is shown in Fig. 3.

Given that at any time, depending on the resources of each virtual machine, it can be assigned to more than one task, therefore, if part of *Allocated* array that stores the number of executed tasks on the *y* machine is as in Fig. 4, then the Eq. (17) at the time *t* is established.

$$\sum_{j=x}^{x+TasksInVM[y]-1} (TasksRequirements[r][Allocated[j]]) \le VMsCapacity[r][y] \forall r, 1 \le r \le Resnum \tag{17}$$

To calculate the allocation cost at any moment, we use Eq. (18):

$$cost = \alpha \times (\beta \times make\_span + (1-\beta) \times mean\_flowtime) + (1-\alpha) \times Load\_imbalance \tag{18}$$

Constants α and β shows the importance of each criterion. For example, if α = 1 and β = 1, then only *Make span* parameter is important and if α = 1 and β = 0, then only second parameter is important. The fitness of

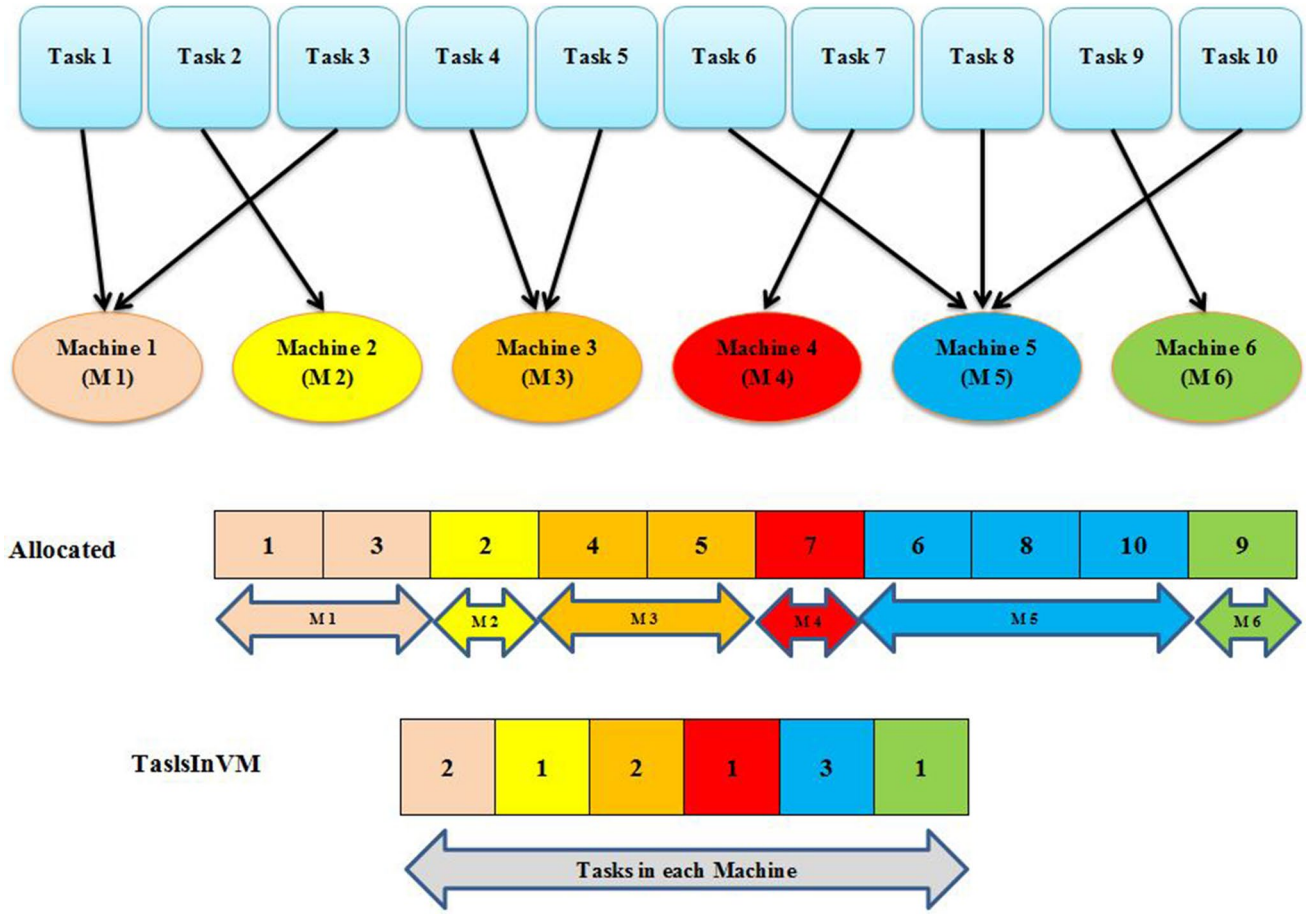


Fig. 2 Coding for the scheduling of 10 tasks on 6 machines

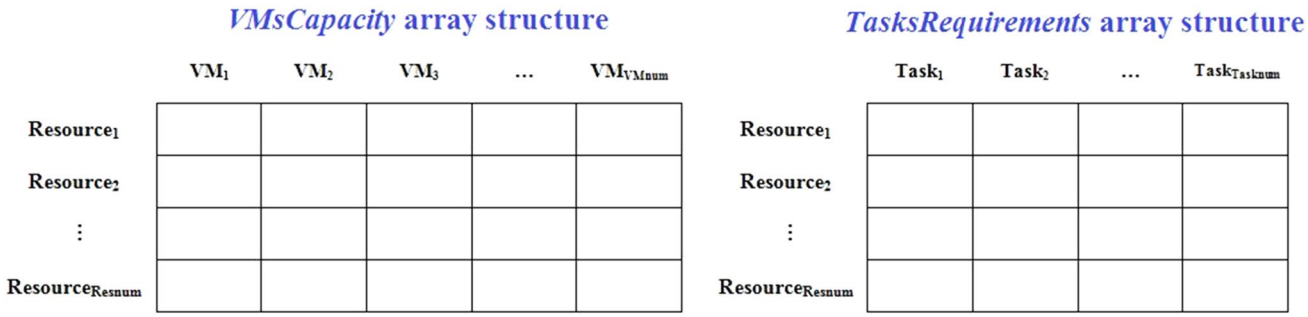


Fig. 3 Structure of VMs capacity and tasks requirements arrays

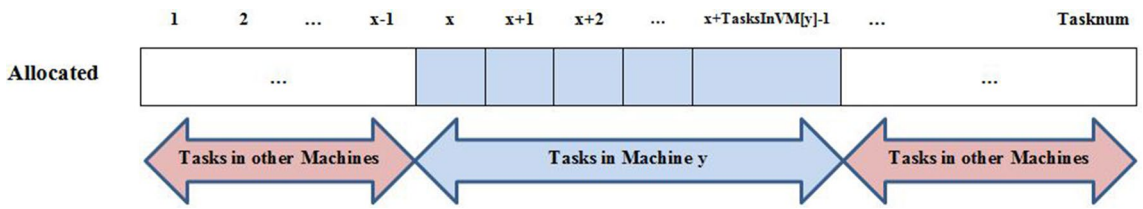


Fig. 4 A part of the allocated array that stores the scheduled tasks numbers on the y machine

appropriateness of the allocation at any time is obtained from Eq. (19):

$$fitness = \frac{1}{cost} \tag{19}$$

Any allocation that has more fitness will be a better allocation.

After formulating allocations, we use the gravitational search algorithm to find an optimal allocation.

Since the *fitness* function is defined in such a way that a better allocation is more competent, so at the time *t*, the best and the worst allocation is obtained from Eqs. (20) and (21), respectively, where *N* is the number of problem responses in time *t*.

$$best(t) = \max_{j\in\{1,\ldots,N\}} fitness_j(t) \tag{20}$$

$$worst(t) = \min_{j\in\{1,\ldots,N\}} fitness_j(t) \tag{21}$$

After calculating the best and worst response, the mass of each response in the time t is calculated using relations Eqs. (22) and (23):

$$q_i(t) = \frac{fitness_i(t) - worst(t)}{best(t) - worst(t)} \tag{22}$$

$$M_i(t) = \frac{q_i(t)}{\sum_{j=1}^{N} q_j(t)} \tag{23}$$

If the system is a collection of *N* masses, then the position of each mass is a point of space, which is the answer to the problem. In our problem, the answer is coded in *Allocated* array therefore, the position of the dimension *d* of the mass *i* is represented by $Allocated_i^d$(Eq. (24)). In this equation the number of tasks (*Task num*) is the dimension of the problem.

$$Allocated_i = \left(Allocated_i^1, \ldots, Allocated_i^d, \ldots, Allocated_i^{Tasknum}\right) \tag{24}$$

Euclidean distance (Norm 2) is used to determine the distance between masses, according to Eq. (25).

$$R_{ij}(t) = Allocated_i(t), Allocated_j(t)_2 \tag{25}$$

In this system, at time *t*, to mass *i* by mass *j*, a force of F_{ij}^d is entered in the direction *d* dimension. The value of this force is calculated from Eq. (26).

$$F_{ij}^d(t) = G(t) \times \frac{M_i(t) \times M_j(t)}{R_{ij}(t) + \varepsilon} \times \left(Allocated_j^d(t) - Allocated_i^d(t)\right) \tag{26}$$

The sum of all forces applied to mass *i* is obtained from Eq. (27). To improve the power of the algorithm, only the *k best* set containing the *k* best members is allowed to affect other members. Because in the initial repetition of the algorithm, there is a need for pervasive search, so the start time affects all the masses on one another, and over time, the number of members affecting the population is reduced by a linear ratio, until the end, only 2% of the best of the population affect to other members.

$$F_i^d(t) = \sum_{j\in kbest, j\neq i} rand_j \times F_{ij}^d(t) \tag{27}$$

In this equation $rand_j$ is a random number with a uniform distribution in the interval [0, 1] and used to maintain the randomness of the search.

Because in the initial repetition of the algorithm, there is a need for pervasive search, so in most papers often the start time affects all the masses on one another, and over time, the number of members affecting the population is reduced by a linear ratio, until the end, few of the best of the population affect to other members. Here we used fuzzy logic to control and adjust the *k* parameter. Our general policy for determining *k* is that if the optimal answer of the algorithm does not change significantly in much iteration, then we increase the value of *k* so that more masses can affect each mass. This makes it possible for a mutation to occur and the algorithm does not get stuck in the local optimal. Also, when the problem response variations are high, we reduce the *k* value to avoid wasting time and increasing the convergence speed of the algorithm.

To do this, we used a fuzzy inference system with two inputs and one output. The first input is the number of iterations where the current response remains unchanged and named fixed number. The second input is the ratio of the number of response changes to the time elapsed since the start of the algorithm and named *variation*. The output of our fuzzy system shows the *k* parameter that determines number of members that allowed affecting other members. Figure 5 shows structure of proposed fuzzy system.

Our fuzzy inputs and output membership functions are shown in Fig. 6.

Figure 7 shows our fuzzy inference system rules. As shown in this figure, when the optimal answer of the algorithm does not change significantly in much iteration, the value of *k best* increased because the algorithm does not get stuck in the local optimal, and when the problem response variations are high, the *k* value reduced to avoid wasting time and increasing the convergence speed of the algorithm.

Plot of the output surface of proposed fuzzy inference system using *fixed number* and *variation* inputs and *k best* output is shown in Fig. 8.

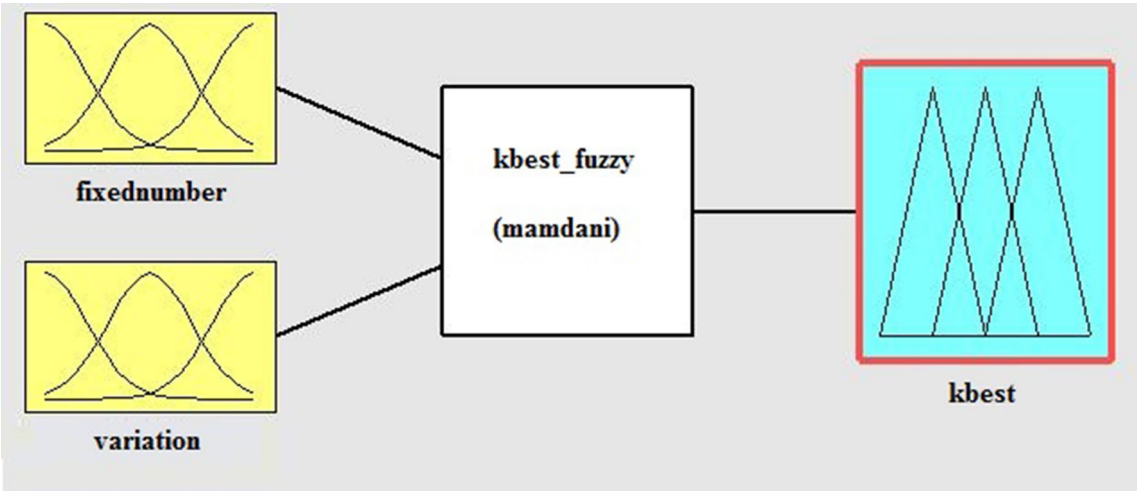


Fig. 5 Structure of proposed fuzzy system

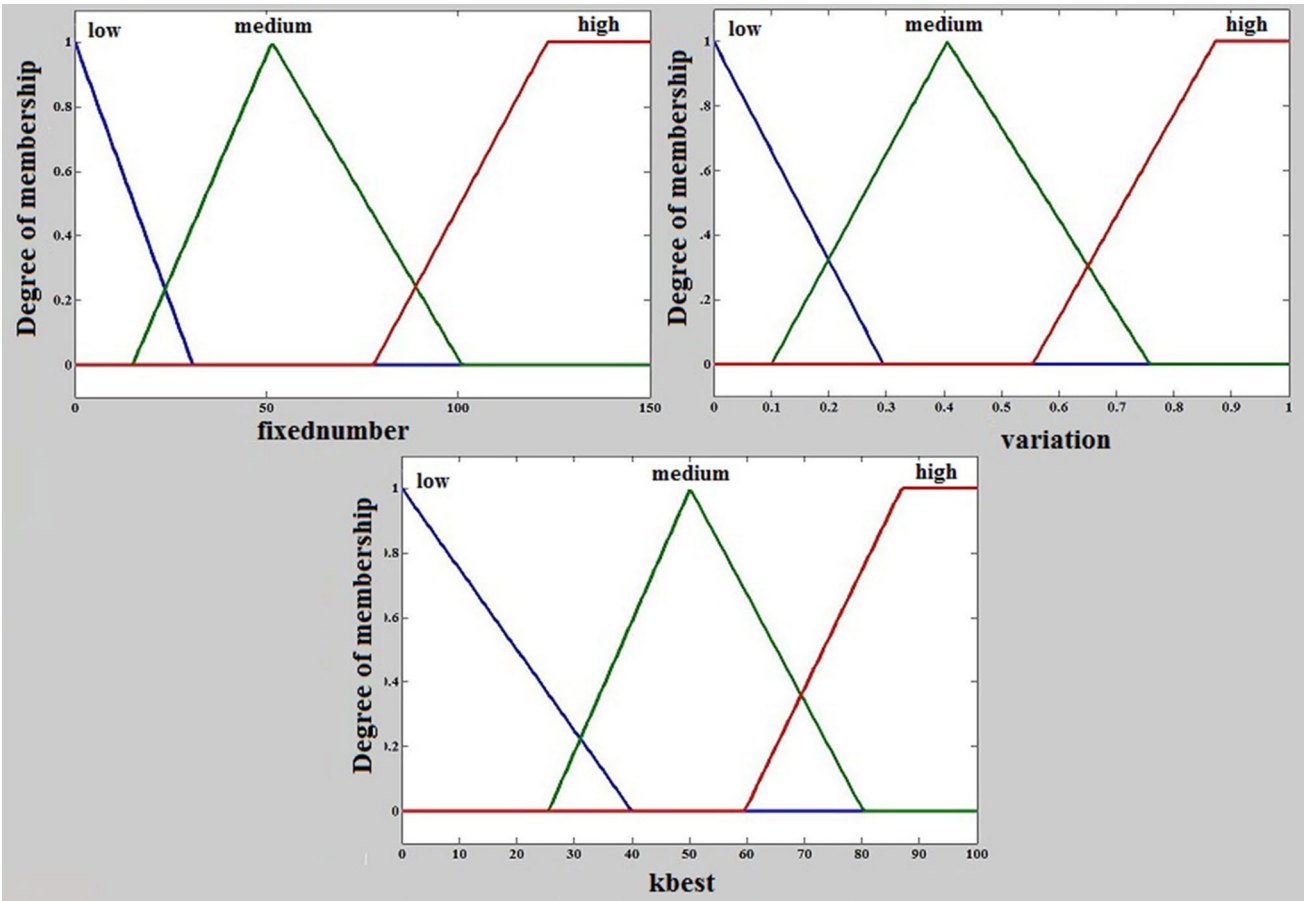


Fig. 6 Our fuzzy inputs and output membership functions

After calculating the force applied to the mass, the acceleration of the mass *i* in the direction of dimension *d* at time *t* is obtained by using Eq. (28).

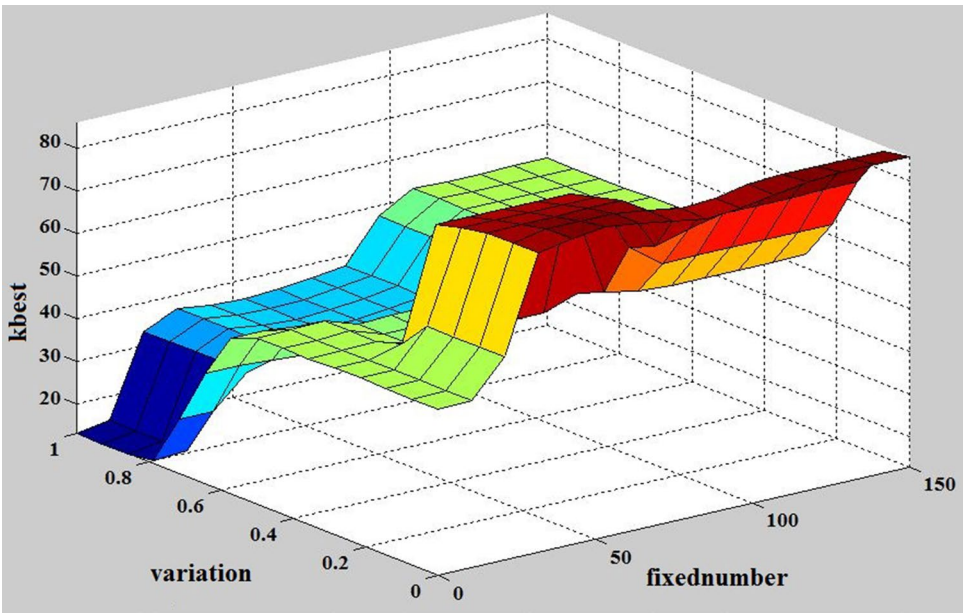
$$a_i^d(t) = \frac{F_i^d(t)}{M_i(t)} \tag{28}$$

The next velocity of each mass is equal to the sum of the coefficients of the current velocity and the acceleration of

Fig. 7 Our fuzzy inference system rules

1. If (*fixednumber* is low) and (*variation* is low) then (*kbest* is medium).
2. If (*fixednumber* is low) and (*variation* is medium) then (*kbest* is medium).
3. If (*fixednumber* is low) and (*variation* is high) then (*kbest* is low).
4. If (*fixednumber* is medium) and (*variation* is low) then (*kbest* is high).
5. If (*fixednumber* is medium) and (*variation* is medium) then (*kbest* is medium).
6. If (*fixednumber* is medium) and (*variation* is high) then (*kbest* is low).
7. If (*fixednumber* is high) and (*variation* is low) then (*kbest* is high).
8. If (*fixednumber* is high) and (*variation* is medium) then (*kbest* is medium).
9. If (*fixednumber* is high) and (*variation* is high) then (*kbest* is medium).

Fig. 8 Output surface of proposed fuzzy inference system



the mass in accordance with Eq. (29). The new position of dimension *d* of the mass *i* is calculated according to Eq. (30).

$$V_i^d(t+1) = rand_i \times V_i^d(t) + a_i^d(t) \tag{29}$$

$$Allocated_i^d(t+1) = Allocated_i^d(t) + V_i^d(t+1) \tag{30}$$

In Eq. (29), $rand_i$ is a random number with a uniform distribution in the interval [0, 1] and used to maintain the randomness of the search.

As mentioned, in the *Allocated* array, the tasks numbers are stored, and since each task only runs on one machine, we scaled the values obtained in Eq. (30) in such a way that they are placed in the interval [1, *Tasknum*] and there are no duplicate values in the $Allocated_i$ array members.

In Eq. (26), the gravity constant starts from an initial value and decreases with time. To calculate this constant, Eq. (31) is used.

$$G(t) = G_0 e^{-\alpha \frac{t}{T}} \tag{31}$$

In this Equation, G_0 is the initial gravitational constant, α is a positive constant and T is the total of algorithm repetitions, that is, the system lifetime.

The proposed method is summarized in the pseudocode as follows:

Resource allocation with GSA

ResourceAllocation with GSA

Input:

Tas knum, VMnum, Resnum,VMsCapacity, Tasks Requirements

Tasknum->Number of tasks

VMnum->Number of virtual machines

Resnum->Number of resource types

VMsCapacity->The amount of resources per virtual machine

TasksRequirements->The amount of resources that tasks require

output:

The optimal allocation of resources to tasks, which has minimum cost

Initialization:

Nsol=100. // Number of masses at the start of the algorithm

T=1000. // Total of algorithm repetitions

G0=100. // Initial gravitational constant

α =2. // A positive constant for decrease gravity

ConvIter=150. // Iteration that shows Convergence condition

Procedure:

1: Create *Nsol* random solutions. //Each solution indicates which tasks are executed on which virtual machines and stores in its *Allocated* array and has its own *TasksInVM* array.

2: **for**(t =1…T) **{**

3: Obtain each solution mass.

4: Determine *kbest* by using fuzzy logic

5: Calculate sum of forces that *kbest* masses applied to each mass.

6: Calculate each mass acceleration by the formula: $a(t) = \frac{F(t)}{m(t)}$

7: Obtain masses velocity by below formula:

$$V(t) = rand \times V(t-1) + a(t-1).$$

8: Update each mass position by below formula:

$$Allocated(t) = Allocated(t-1) + V(t).$$

9: **if** ($Cost(Allocated(t)) < MinCost$)**{**

10: MinCost=Cost(Allocated(t)).

11: BestSolution=Allocated(t). **}** // **End if**

12: **If** (in the past *ConvIter* iterations, MinCost has not changed)**{**

13: Break.**}** //**End if**

14: **}** // **End for**

15: return BestSolution.

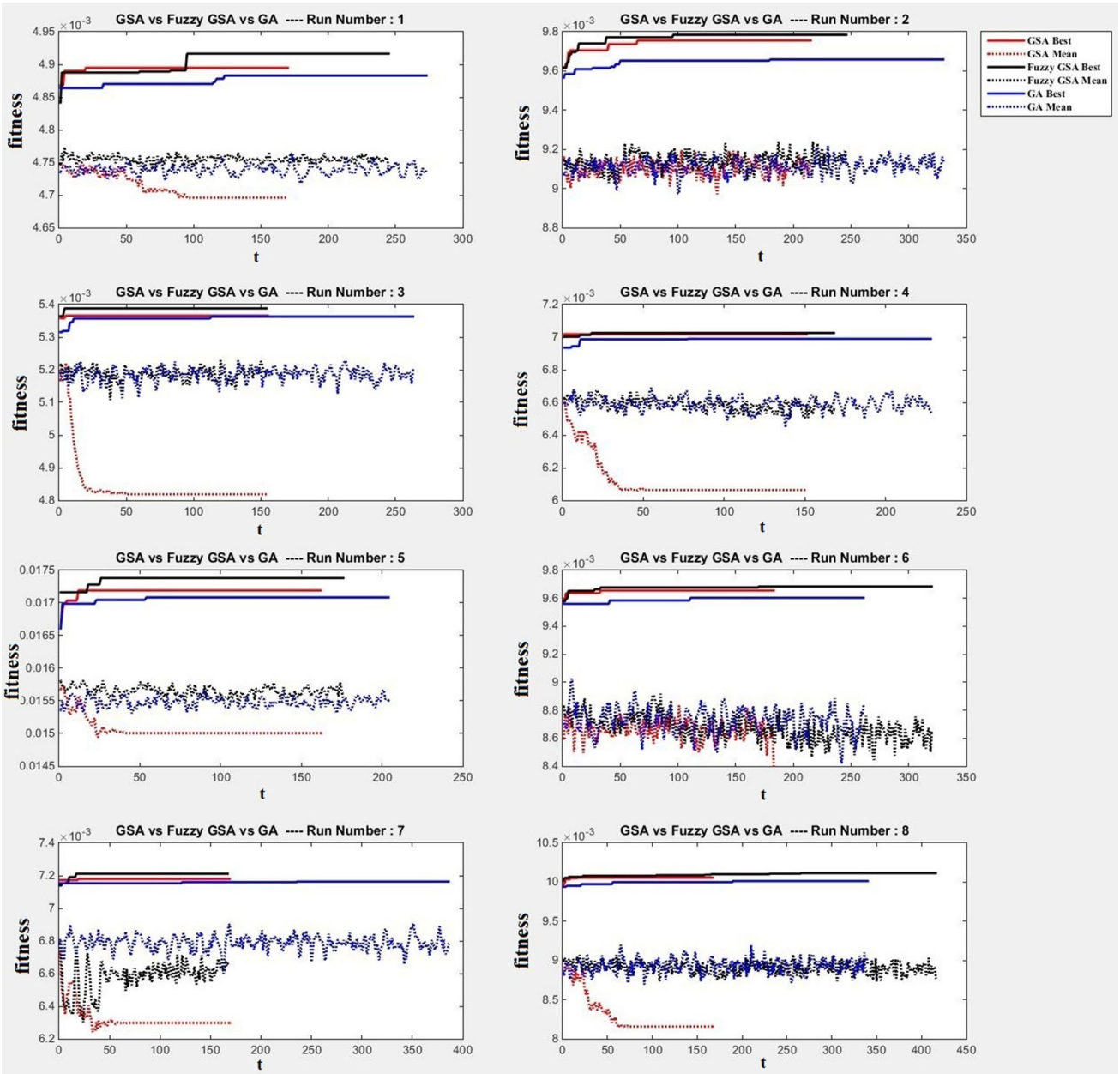


Fig. 9 Compare fitness value and convergence of proposed method with GSA without fuzzy enhancement and GA

6 Results

In this section, we compare the responses from the implementation of the proposed method to the responses obtained using the GSA without fuzzy enhancement and using genetic algorithm. To run the algorithm we used a system with CPU = core i7 4800MQ, RAM = 8 GB, VGA = AMD Radeon HD 8790 M with 2 GB Dedicated Memory. To test the proposed method, we first create virtual machines with random numbers of each of the available resources using a function called *create vms*. Also, by creating a function called *create tasks*, we create number of arbitrary tasks with random numbers of each of the available resources.

Each time we compare the proposed algorithm with the mentioned algorithms, we use the same virtual machines and the same tasks. In this way, before each comparison, we call the mentioned functions for creating virtual machines and tasks.

For the first comparison, we put $G_0 = 100$ and $\alpha = 2$. First, we create 100 masses randomly. For the genetic algorithm, we also consider the Crossover's probability to be 0.3 and the mutation's probability to be 0.2. We also consider the initial population to be 100.

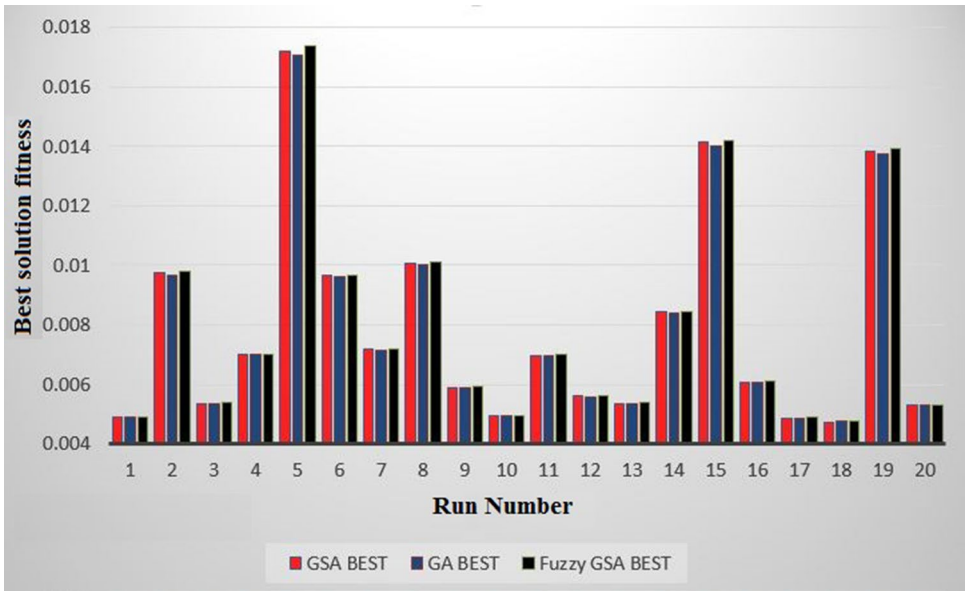


Fig. 10 Compare proposed method best solution with GSA without Fuzzy and GA best solution

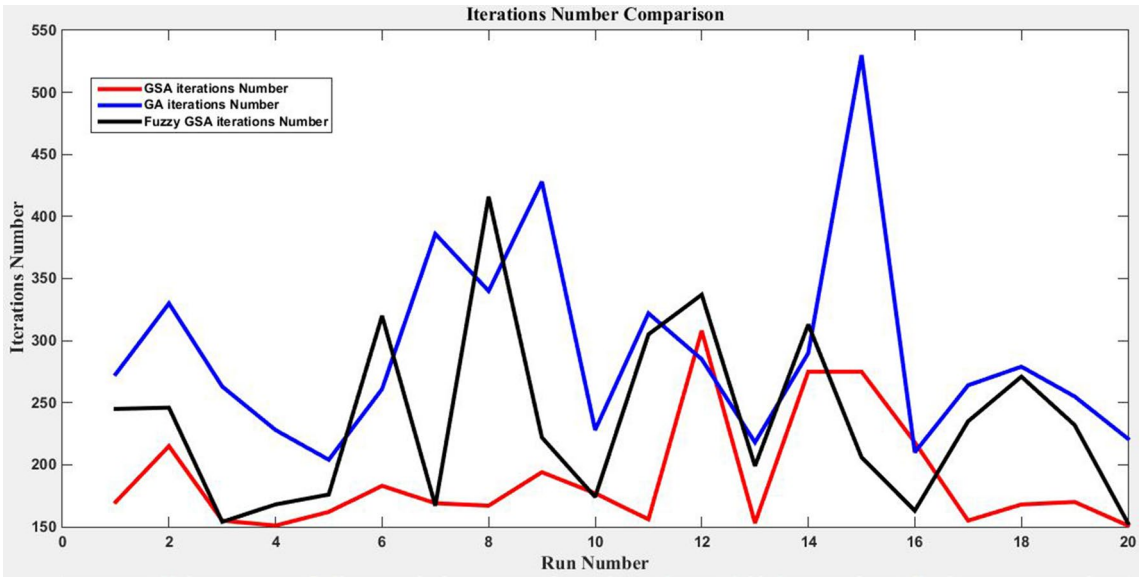


Fig. 11 Compare proposed method number of iteration with GSA without Fuzzy and GA number of iteration

Figure 9 shows the results of running the algorithm with eight categories of virtual machines and various tasks. The convergence condition is the steady state of the best answer in the last 150 repetitions of the algorithm. For example in Run_Number#1, from the iteration#94 to the iteration#244, fitness of our proposed method remained the same and in 150 iterations, no improvement was observed. So this algorithm stop in iteration#244. The convergence condition for the other two algorithms is the same as the convergence condition of this algorithm.

As Eq. (18) showed, cost of each solution is calculated based on the combination of make span and mean_ flow time and Load_ imbalance. On the other hand, according to Eq. (19), the fitness of each solution is inversely related to its cost. So being fuzzy _GSA fitness higher than other two algorithms in Run_Number#1 means that it allocated resources to tasks with less make span and mean _flow time and more load balancing than other two algorithms. Overall the results show that the proposed method obtains roughly more optimal responses than genetic algorithm and GSA without fuzzy. The

comparison of the results of these eight runs with the twelve other runs is illustrated in Fig. 10.

As showed in Fig. 9, the proposed method often converges to the optimal response in less number of iterations than the genetic algorithm, but more number of iterations than the GSA without fuzzy. This increment is due to *k best* tuning by fuzzy and makes the GSA algorithm not get stuck in the local optimum. Figure 11 shows number of iterations of these algorithms from twenty times execution.

7 Conclusion

Since resource allocation and tasks scheduling problem in cloud environment is a NP-hard problem and previous studies showed that GSA has a high efficiency in solving nonlinear problems, so in this study we used GSA for resource allocation in cloud environment. For this, first we coded problem responses in some arrays that each code showed a sequence of tasks assigned to each machine. Here we used combination of Make_ span and Mean _Flow_ Time and Load_ imbalance as cost function in GSA and attempt to minimize this function value and consequently maximize fitness value of masses. To increase the accuracy of the algorithm, we used fuzzy logic to determine the number of masses that affect one another during the implementation of the GSA and then continue the other steps of the GSA. The results show that proposed method, in comparison with the Genetic Algorithm (GA) and GSA without fuzzy enhancement, receives roughly more optimal responses for resource allocation and it allocated resources to tasks with less make span and mean_ flow time and more load balancing than other two algorithms. Although the proposed method often converges to the optimal response in less number of iterations than the GA and makes the GSA algorithm not get stuck in the local optimum. Future works can include other parameters such as the number of tasks to be completed per unit of time in terms of cost. Also to increase the speed of the algorithm and reduce the number of iterations, for *k best* determining, time is also considered as the input of the fuzzy system so that time has verse effect on *k best*.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Manvi SS, Shyam GK (2014) Resource management for infrastructure as a service (IaaS) in cloud computing: a survey. J Netw Comput Appl 41:424–440. https://doi.org/10.1016/j.jnca.2013.10.004
2. Nguyen T, Quang-Hung N, Tuong NH, Tran VH, Thoai N (2013) Virtual machine allocation in cloud computing for minimizing total execution time on each machine. In: International conference on computing, management and telecommunications (ComManTel), Ho Chi Minh City, pp 241–245. https://doi.org/10.1109/ComManTel.2013.6482398
3. Carretero J, Xhafa F, Abraham A (2006) Genetic algorithm based schedulers for grid computing systems. Int J Innov Comput Inf Control 3:1053–1071
4. Maqableh M, Karajeh H, Masa'deh R (2014) Job scheduling for cloud computing using neural networks. Commun Netw 6:191–200. https://doi.org/10.4236/cn.2014.63021
5. Sasikaladevi N (2016) Minimum makespan task scheduling algorithm in cloud computing. Int J Grid Distrib Comput 9(11):61–70. https://doi.org/10.14257/ijgdc.2016.9.11.05
6. Rashedi E, Nezamabadi-Pour H, Saryazdi S (2009) GSA: a gravitational search algorithm. Inf Sci 179(13):2232–2248. https://doi.org/10.1016/j.ins.2009.03.004
7. Zhao C, Zhang S, Liu Q, Xie J, Hu J (2009) Independent tasks scheduling based on genetic algorithm in cloud computing. In: 5th International conference on wireless communications, networking and mobile computing, Beijing, pp 1–4. https://doi.org/10.1109/WICOM.2009.5301850
8. Pandey S, Wu L, Guru SM, Buyya R (2010) A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In: 24th IEEE international conference on advanced information networking and applications, Perth, WA, pp 400–407. https://doi.org/10.1109/AINA.2010.31
9. Lakra AV, Yadav DK (2015) Multi-objective tasks scheduling algorithm for cloud computing throughput optimization. Procedia Comput Sci 48:107–113. https://doi.org/10.1016/j.procs.2015.04.158
10. Li K, Xu G, Zhao G, Dong Y, Wang D (2011) Cloud task scheduling based on load balancing ant colony optimization. In: Sixth annual chinagrid conference, Liaoning, pp 3–9. https://doi.org/10.1109/chinagrid.2011.17
11. Priya V, Kumar CS, Kannan R (2019) Resource scheduling algorithm with load balancing for cloud service provisioning. Appl Soft Comput 76:416–424. https://doi.org/10.1016/j.asoc.2018.12.021
12. Mansouri N, Zade BMH, Javidi MM (2019) Hybrid task scheduling strategy for cloud computing by modified particle swarm optimization and fuzzy theory. Comput Ind Eng 130:597–633. https://doi.org/10.1016/j.cie.2019.03.006
13. Jena T, Mohanty JR (2018) GA-based customer-conscious resource allocation and task scheduling in multi-cloud computing. Arab J Sci Eng 43:4115. https://doi.org/10.1007/s13369-017-2766-x
14. Muthulakshmi B, Somasundaram K (2017) A hybrid ABC-SA based optimized scheduling and resource allocation for cloud environment. Clust Comput. https://doi.org/10.1007/s10586-017-1174-z
15. Holliday D, Resnick R, Walker J (1993) Fundamentals of physics. Wiley, Hoboken