Computers and Electrical Engineering 95 (2021) 107419

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

Hybrid ant genetic algorithm for efficient task scheduling in cloud data centers

Muhammad Sohaib Ajmal [a], Zeshan Iqbal [a], Farrukh Zeeshan Khan [a], Muneer Ahmad [b], Iftikhar Ahmad [c], Brij B. Gupta [d,e,*]

[a] Department of Computer Science, University of Engineering and Technology Taxila, Pakistan
[b] Department of Information Systems, Faculty of Computer Science and Information Technology, Universiti Malaya, Kuala Lumpur, Malaysia
[c] Department of Information Technology, Faculty of Computing and Information Technology, King Abdulaziz university, Jeddah, Saudi Arabia
[d] Department of Computer Engineering, National Institute of Technology, Kurukshetra, India
[e] Department of Computer Science and Information Engineering, Asia University, Taiwan

ARTICLE INFO

Editor: Dr. M. Malek

Keywords:
Ant colony algorithm
Cloud computing
Data center cost
Evolutionary algorithm
Genetic algorithm
Multi-objective optimization
Quality of service (QoS)
Resource allocation
Service level agreement (SLA)
Task scheduling

ABSTRACT

Cloud computing is a computing paradigm which meets the computational and storage demands of end users. Cloud-based data centers need to continually improve their performance due to exponential increase in service demands. Efficient task scheduling is essential part of cloud computing to achieve maximum throughput, minimum response time, reduced energy consumption and optimal utilization of resources. Bio-inspired algorithms can solve task scheduling difficulties effectively, but they need a lot of computational power and time due to high workload and complexity of the cloud environment. In this research work, Hybrid ant genetic algorithm for task scheduling is proposed. The proposed algorithm adopts features of genetic algorithm and ant colony algorithm and divides tasks and virtual machines into smaller groups. After allocation of tasks, pheromone is added to virtual machines. The proposed algorithm effectively reduces solution space by dividing tasks into groups and by detecting loaded virtual machines. Due to the minimum solution space of proposed algorithm, convergence and response time is significantly decreased. It finds a feasible scheduling solution to minimize the running time of workflows and tasks. The proposed algorithm achieved 64% decrease in execution time and 11% decrease in overall data center costs.

1. Introduction

Cloud computing is a rapidly emerging field that aims to provide computing resources on demand. Its tremendous growth is due to its scalable, dynamic, and customizable environment. Moreover, it provides virtual resource abstraction and hides technical aspects of management. Currently, cloud computing has three administrative models that provide software, infrastructure and platform services [23–25]. However, researchers have also proposed other models, such as cloud and sensor integration platform (SC-iPaaS), health platform as a Service (HPas), and social cloud. These models operate on a pay-per-use policy; thus, they do not require business

This paper is for special section VSI-bioc. Reviews processed and recommended for publication by Guest Editor Dr. Xiaochun Cheng.

* Corresponding author at: Department of Computer Engineering, National Institute of Technology, Kurukshetra, India.
E-mail addresses: sohaib.ajmal1993@gmail.com (M.S. Ajmal), zeshan.iqbal@uettaxila.edu.pk (Z. Iqbal), farrukh.zeeshan@uettaxila.edu.pk (F.Z. Khan), mmalik@um.edu.my (M. Ahmad), iakhan@kau.edu.sa (I. Ahmad), gupta.brij@gmail.com (B.B. Gupta).

https://doi.org/10.1016/j.compeleceng.2021.107419
Received 23 August 2020; Received in revised form 19 January 2021; Accepted 27 August 2021
Available online 6 September 2021
0045-7906/

enterprises to invest in their network infrastructure. Instead, organizations get their computing and storage resources from public cloud providers. Some organizations also have private clouds, but they can opt for public clouds if they need additional resources. The cloud provider is responsible for providing resources on demand. A cloud can deal with abrupt changes in workload demands by offering auto-scaling feature. There are two types of scaling in data centers, vertical scaling and horizontal scaling. A cloud data center may contain thousands of machines. Energy consumption is a critical concern due to unprecedented growth of the cloud data centers. High energy consumption increases operational costs of data centers and emission of carbon dioxide (CO_2) gases into the environment. Resource optimization is required for the reduction in energy usage and minimizing the operational cost of data centers.

Task scheduling in data centers is a critical research area. Load balancing, scalability, and performance of the data centers depend on efficient task scheduling mechanism. Task scheduling aims to map the best resources to the workload to minimize the running time of workflow and improve resource utilization. In cloud data centers, many applications execute in parallel. Parallelism cannot be fully achieved if jobs are not properly scheduled. Inefficient scheduling results in high execution time, high cost and low resource utilization, reducing the overall performance of the cloud. Efficient scheduling algorithms are used to achieve the objectives of better resource utilization. Scheduling algorithms map tasks to virtual machines in a fashion that reduces the cost and execution time of workflows. Cloud providers' services operate according to the pay-per-use policy. Quality-of-service (QoS) provided to the users is written in the Service level agreement (SLA). Every task has different computation, storage, bandwidth, and response time requirements. If a task cannot obtain its required resources, it results in SLA violation. If SLA violations are high in a data center, then the cloud provider's QoS is compromised.

The task scheduling problem is NP hard. It is assumed that NP-hard problems are difficult to solve by any algorithm in polynomial time. Researchers have proposed many heuristics to solve the problem of task scheduling. Scheduling algorithms have two types, the first is heuristic-based, and the second is a random guided search based. Heuristic-based algorithms achieve good results in task scheduling problems and can be executed on both homogeneous and heterogeneous systems. These can be further divided into various types: list-based heuristic, rule-based heuristic, and meta-heuristic. Rule-based heuristics algorithms are traditional algorithms, for example Max-Min [1], Min-Min [2], and Minimum Execution Time (MCT) [3]. Max-Min algorithms first allocate tasks with maximum execution time to the processor, followed by smaller tasks. Min-Min algorithms give first preference to the execution of smaller tasks. MCT algorithms allocate tasks to the processor according to what it can execute in a short time. Rule based heuristic algorithms are simple and their performance is very good, but when the workload is higher, their performance is compromised. Famous list-based heuristic algorithms are Heterogeneous earliest fish time (HEFT) [4], Critical path on processor (CPOP) [5], and Parental prioritization-based task scheduling in heterogeneous systems (PPEFT) [6]. In these algorithms, tasks are prioritized and allocated to the processors for the execution. Dynamic frequency scaling technique (DVFS) is also used for task scheduling to decrease energy utilization. DVFS works according to the fact that power consumption is directly proportional to frequency and voltage. These algorithms reduce the frequency of servers when tasks are not time critical, and low voltage is applied to machines. List-based heuristic algorithms perform better than rule-based heuristic algorithms when the workload is high.

Meta-heuristic algorithms, also called swarm intelligent algorithms, include bio-inspired algorithms like Ant colony optimization (ACO), Artificial bee colony algorithm (ABC), Genetic algorithm (GA), and Particle swarm optimization (PSO). In nature, each individual can solve the problem with limited capacity. Therefore, these algorithms can solve NP-hard problems effectively. For example, bees, ants, and birds can search for food with very little intelligence but feature self-organization and parallelism. All individuals can work without central control and can find an optimized solution. With bio-inspired algorithms, we can find an optimal solution of complex problems. GA works on the rule of inheritance, where features are inherited to the child from the parents. With the evolution of time, populations with good features survive. In GA [7], we begin with a random population then this population passes its properties to new offspring. The population with an optimized solution is selected and others are removed. Over time, the algorithm reaches the global optimal solution. ACO [8] works according to the nature of ants. Ants leave their nests for the food. They add pheromones on their path. More pheromone is added to the shortest path because ants return along that path most quickly. Other ants start to follow the path which has more pheromones. In this way, after some time a global optimum is achieved. ABC [9] simulates the foraging behavior of honeybees. Honeybees are divided into two groups: scout bees that find an optimal solution from food sources, and onlooker bees that find the new food source. PSO [10] is an optimization algorithm inspired by flocks of birds to find a global optimum.

With the increase in usage of mobile devices, social media, and low-cost connectivity to the Internet, cloud usage is increased. Cloud computing is used by the scientists to solve scientific problems because of its high computational power and storage capacity [20–22]. Many companies have billions of users, for example, in the 3rd quarter of 2020, Facebook records 2.7 billion active users. This means that 80 million people used the platform on a daily basis. To handle parallel requests from such a huge number of clients, it is necessary to schedule tasks to appropriate servers and VMs. Inappropriate scheduling can increase the cost of using cloud services, workload execution time, SLA violations, cost, and energy consumption, while decreasing the overall quality of cloud services. It is caused due to inefficient utilization of resources. According to one study of 5000 servers for six months, utilization was only 10 to 50%. As discussed earlier, the cloud computing model works according to a pay-per-use policy, and the above-mentioned problems can result in high costs for both users and cloud providers. Motivated by these issues, a Hybrid ant genetic algorithm (HAGA) for efficient task scheduling in the cloud data centers is proposed. The contributions of this paper are as follows:

(1) Proposed a HAGA algorithm for efficient task scheduling in clouds data centers.
(2) Designed a mutation technique.
(3) Introduced a new evaporation technique.
(4) Perform load balancing in datacenter.

(5) Provide large-scale comparisons of different algorithms.
(6) Results show that the HAGA has less computational time, response time and convergence time as compared to other algorithms used in comparison like Min-Min, Max-Min, Adaptive incremental genetic algorithm AIGA [11], and GA.

The proposed work was compared with a wide range of algorithms, including Max-Min, Min-Min, GA, and AIGA. The proposed HAGA algorithm outperforms other algorithms in the literature for different performance metrics like execution time, response time, fitness value, convergence time, and SLA violations.

The rest of the paper is organized as follows. In Section 2, relevant literature is reviewed and existing techniques are compared. In Section 3, objective function is defined. Section 4 describes the proposed HAGA algorithm, in detail. Section 5 evaluates the performance of the proposed algorithm, while Section 6 concludes the work.

2. Related work

In literature, many researchers have proposed different scheduling algorithms to achieve the maximum performance of the cloud. Jang et al. [12] presented a model in which a scheduling function is called by the task scheduler after every fixed scheduling interval. The proposed scheduler function evaluates resources required by the tasks and compares them with available resources to satisfy user's SLA. The function iterates and generates the optimal schedule. Duan et al. [11] proposed an incremental genetic algorithm, called Adaptive incremental genetic algorithm by dividing tasks into smaller groups. They proposed a new mutation technique that automatically changes crossover and mutation rates. This algorithm obtains the optimal solution in less time than the standard GA algorithm. Zuo et al. [13] proposed a scheduling algorithm using ACO and has multiple objectives. It takes user's budget cost and time span as the objective functions and minimizes these constraints. Two constraint functions check the quality of the solution based on feedback after a fixed interval to reach the global optimal solution in less time. Young Moon et al. [14] proposed new methodology in Ant colony algorithm for task scheduling. This algorithm uses slave ants to avoid selecting longer paths. Different strategies are used by slave ants to detect long paths created by leading ants and move toward the optimal solution. This algorithm uses two pheromones' matrices: one is a local pheromone, and the second is a global pheromone to reach global optimum instead of stacking at local optima.

Milan et al. [15] presented a bacterial swarm intelligent algorithm that schedules tasks according to their priority. This model uses the foraging and chemotactic behavior of bacteria and reduces the idle time of virtual machines (VMs). This algorithm reduces workflow execution time and saves energy usage. Feller et al. [16] proposed a novel methodology for scheduling workload. They modeled workload as an instance of a multidimensional bin packaging problem and used ACO to optimize workload scheduling. Their approach utilizes resources in an optimized way and saves power. Shu et al. [17] proposed an energy-efficient algorithm using efficient resource allocation. Their approach applies the immune clonal optimization technique which uses the optimization and searching technique of clonal algorithms. Dashti and Rahmani [18] proposed a VM placement strategy using PSO. They proposed task scheduling in the SaaS layer and used PSO to relocate VMs, which are migrated from one server to another when overloading occurs. This approach maintains QoS and minimizes energy.

Most researchers used tasks from a few hundred to a little more than one thousand for experimentation and compared their work with a smaller number of techniques. Table 1 presents the comparison of scheduling objectives, the number of metrics used for comparison, and the tasks used in the work discussed in the literature. Swarm intelligent algorithms perform very well for the scheduling problem; however, these algorithms find an optimal solution by checking every possible solution. Therefore, their convergence time is very high. Hence, minimum timespan and the low convergence time are objectives of this work. Accuracy of the proposed algorithm is assessed after evaluating it for SLA violations, convergence time, and response time.

Table 1.
Comparison of work discussed in the literature.

Refs.	Scheduling objective	Comparison metrics	Total tasks	Algorithms compared
[1]	Make span	Tasks pending time tasks response ratio	N/A	3
[2]	Make span	Make span	N/A	2
[3]	Power	Power consumption datacenter load	3337	3
[4]	Make span	Schedule length ratio (SLR) running time speedup	N/A	5
[5]	Make span	Make span SLR	25	3
[6]	Make span	SLR Speedup efficiency	100	3
[7]	Make span	Make span	N/A	1
[8]	Load balancing	Make span transmission delay load deviation value	200	4
[9]	Convergence time	Efficiency	N/A	3
[10]	Make span	Make span resource utilization	100	3
[12]	Task scheduling	Response time cost average utilization threshold	2000	4
[11]	Optimal fitness	Convergence optimal fitness make span	90000	6
[13]	Cost make span	Make span cost SLA violation resource utilization	600	4
[14]	Make span	Make span	700	3
[15]	Energy efficiency	Make span energy consumption	1200	5
[16]	Energy efficiency	Energy consumption	N/A	6
[17]	Energy efficiency	Response time make span energy consumption	800	4
[18]	Energy efficiency	Make span energy consumption migrations	N/A	4

3. Task scheduling problem

Cloud data centers consist of thousands of physical servers on which virtual machines are running. User tasks run on virtual machines. A virtual machine is chosen on the basis of demands. The cloud environment has two types of scheduling. One is the selection of servers to run virtual machines on them. This type of scheduling directly affects the efficiency of data centers, energy consumption and utilization of resources. In the second type of scheduling, virtual machines are selected to run tasks. Large tasks are often divided into smaller parts and then distributed to available virtual machines in the resource pool for processing. In this scheduling type, VMs are chosen according to the service requirements of user and status of VMs. This type of scheduling affects the completion time and execution costs of tasks. The first type of scheduling affects cloud providers, while the second type affects users, especially in terms of quality of service and cost. Therefore, cloud service providers need to improve cloud efficiency and reduce unnecessary costs through appropriate scheduling strategies.

Data center Broker and Cloud information service (CIS) schedules user tasks to virtual machines on the basis of individual user's requirements and quality of service. Users want to keep service costs low while cloud providers aim at keeping energy consumption low with maximum performance and utilization of servers. This is because execution time is directly related to these factors. With the increase in running time of tasks, cost and energy consumption also increases. Therefore, in this research work, makespan is the primary scheduling objective. As already discussed, the task scheduling problem is NP-hard. Evolutionary algorithms behave well for the NP-hard problem, but their convergence time is very slow because they check every possible and feasible solution. Therefore, convergence time is a secondary objective in this work. A brief abbreviation of used terms is defined in Table 2.

Suppose there are *m* virtual machines and *n* tasks in the system then set of virtual machines *V* and tasks *T* can be represented as:

$$V = \{v_1,\ v_2, v_3,\ ....,\ v_m\}$$

$$T = \{t_1, t_2,\ t_3,\ ....,\ t_n\}$$

These tasks can be allocated to virtual machines in n^m ways. *R* is a set of virtual machine resources and can be represented as$R = \{r_1, r_2, r_3, ..., r_k\}$. These resources include CPU ability to execute millions of instructions per second (MIPS), bandwidth, RAM, and storage. Task completion time depends on these resources of virtual machines. Cloud service providers use some measuring units, called Compute Unit (CU) to calculate the capacity of the virtual machine. One Amazon CU is equivalent to 1.0 to 1.2 GHz of Opteron or Xeon processor. The execution time of the tasks and the estimated running cost of the workflow can be calculated by CU. Therefore, execution time ET of task *t* can be expressed as,

$$ET_{ij} = \frac{L_i}{CU_j} \tag{1}$$

Where, $i \in \{1, 2, 3, ..., n\}$, $j \in \{1, 2, 3, ...m\}$, L_i is the length of the task t_i and CU_j represents CU of a virtual machine v_j. Busy time of VM is the time during which tasks are running on it. Busy time BT of vm_j can be expressed as,

$$BT_j = \sum_{i=1}^{n} ET_{ij} \times c_{ij} \tag{2}$$

Where, $c_{ij} \in \{0, 1\}$ and x_{ij} represents the relationship between virtual machine and task. If x_{ij} is 1 then the task t_i is scheduled on a virtual machinev_j. As virtual machines run in parallel, the running time of workflow is equal to the maximum busy time of a VM. Therefore, we can calculate the makespan MT of the system by,

$$MT = max(BT_j) \tag{3}$$

Table 2.
Table of abbreviations.

Term	Abbreviation
Service level agreement	SLA
Execution time of task	ET
Busy time of VM	BT
Makespan	MT
Convergence time	CT
Objective function	O
Number of Tasks in group	TG
Chromosome	*X*
Cuts	C1 & C2
Pheromone	*f*
Selection probability	*P*
Evaporation rate	ρ
VM Selection constant	γ
Crossover rate	CR
Mutation rate	MR

Evolutionary algorithms iterate over all possible solutions to get an optimal solution. Their convergence time depends on solution space and the number of iterations. Larger the solution space, the more time they take to converge. The relation between convergence time and solution space can be expressed as:

$$CT \propto (l_x,\ k) \tag{4}$$

Where CT represents convergence time, l_x is the length of the solution x, and k is the total iterations to find an optimal solution. From Eqs. (3) and (4), we can derive our objective function O to find solution x as:

$$O(x) = min\ (MT),\ min\ (CT) \tag{5}$$

4. Proposed approach

The genetic algorithm is an evolutionary algorithm and works on the principles of nature. In nature, individuals that can adapt themselves to the changing environment can survive while others cannot survive. Features of individuals are written on genes that are stored in chromosomes. Individuals with good environmental adaptability survive easily as compared to individuals with less adaptability to the environment. In genetic algorithm, young individuals are produced by the crossover and genetic mutation. Every solution in a population has a fitness value associated with it. Fitness value is used to calculate quality of solution. Convergence time and direction of evolution greatly depend on selection strategies. The genetic algorithm follows a series of selection, crossover, and mutation operations. First, individuals with the best fitness value are selected. In the crossover, genes are recombined to produce a new generation. Ideally, the good features of parents should be inherited by the new generation. Finally, mutation creates a new individual by changing a gene.

When we use GA to solve various problems, every chromosome represents an entire solution. A solution is evaluated by the fitness value; the small fitness value means a good solution. Genetic algorithms perform crossover, mutation, and selection operations for finding an optimized solution. The selection operation selects the best chromosomes and discards other chromosomes based on their fitness value. After a few generations, the population comprises of individuals with the best fitness values, which is an optimal solution.

ACO is a probabilistic technique inspired by ants. When ants search for food, they have the natural ability to find the shortest and optimized path. Initially, ants select random paths and then deposit pheromones along the paths according to the fitness of each path. If a path is good, the pheromone quality will be high. Next, ants have a high probability of choosing the path with more pheromones. After some time, they find the shortest path and start to follow that path. With time, the pheromone evaporates, and the process continues.

In task scheduling, chromosomes are encoded with numbers. Each chromosome contains a feasible solution. As a data center can have thousands of tasks and crossover and mutation operations require large computation power, the GA algorithm takes a long time to find the right solution. To address this issue, we divided the tasks into smaller groups to minimize the solution space. HAGA algorithm take advantage of the ACO algorithm to keep track of previous tasks that are scheduled on the server and have not finished their execution yet. This helps in detecting loaded virtual machines. Overloaded virtual machines are not selected for the next group of tasks. This approach reduces solution space and there is a high probability that global optimum is achieved. Details of the proposed HAGA are given below.

4.1. Encoding

Many encoding strategies are used for different problems, such as symbol encoding, binary number encoding, and real number encoding. Binary encoding maps a solution space to a set of string, while symbol encoding represents a solution space according to some symbols. Similarly, real number encoding represents a solution space according to some real numbers. The gene value of the chromosome represents the VM on which the task will be scheduled, and chromosome length is equal to total tasks. In this work, tasks are divided into smaller groups and real numbers are used for chromosome encoding. Each group is allocated to a set of virtual machines of size w. Suppose, G is a set of groups then$G = \{g_1,g_2,g_3....,g_y\}$. The number of tasks TG assigned to a group $g_k$can be expressed as:

$$TG_k = \frac{n}{y} \tag{6}$$

Where n is the total number of tasks, y is the total number of groups. Suppose there are z chromosomes/solutions, then set of chromosomes X can be represented as$X = \{x_1,x_2,x_3,...., x_z\}$. The chromosome length will be same as total tasks TG for a group k. For example, if 5 tasks and 3 virtual machines are assigned to a group, then Fig. 1 represents a simple chromosome.

Where, each column in a chromosome represents a gene and set of chromosomes is called a population.

t0	t1	t2	t3	t4
v0	v1	v2	v2	v0

Fig. 1.. A chromosome.

4.2. Fitness

In a genetic algorithm, crossover and mutation operations are performed in each step of iteration and new solutions are created. It is necessary to check the feasibility of the solution. The fitness value is used to check the quality and feasibility of the solution. In this algorithm, fitness value is a make span of a solution and can be calculated by Eq. (3). The lesser the fitness value the better solution will be.

4.3. Selection

In selection, solutions with the best fitness are selected while others are discarded. Several selection strategies are used for selection, for example, tournament, roulette, and sorting selection. We use a roulette selection scheme. In the first step, the selection probability of each chromosome is calculated. Selection probability P of a chromosome x_j is calculated by,

$$P_j = \frac{MT_j}{\sum_{i=0}^{z} MT_i} \tag{7}$$

After calculating selection probabilities, we divide a circular disk into z parts and the width of each part is proportional to selection probability. In the last step, a random number is generated and a chromosome of the region in which the random number lies is selected.

4.4. Crossover

Crossover is used to create new offspring from individuals selected in the selection phase. Various crossover techniques are used in the literature. We use two-point crossover. In this technique, chromosomes are first divided into two cuts. Cuts position, on the chromosomes, can be randomly selected as,

$$[C1, C2] = rand(2) \times (TK - 1) + 1 \tag{8}$$

Where, C1 and C2 are two cut points and $rand(2) \times (TK - 1) + 1$ generate a random number in the range of 1 to the total length of the chromosome. Then the values of both chromosomes between these cuts are interchanged to two new chromosomes. Fig. 2 shows crossover in detail.

4.5. Mutation

Mutation operation introduces new features in chromosomes. It includes swap mutation, uniform mutation, and interchange mutation. In these techniques, a position in the solution is chosen randomly and its gene value is replaced with a gene value of another position. To ensure that each position has an equal chance for a new feature, we introduce a new mutation strategy. In this technique, first, we choose the starting position and ending position, then interchange genes on these positions. In the next iteration, position value from start is incremented and decremented from last, and then genes values are interchanged at these positions, and so on. Fig. 3 shows the mutation technique in detail.

4.6. Add pheromone

For load balancing and minimizing the response time of finding an optimal solution, we introduce a novel approach. After allocating virtual machines to a group of tasks, a pheromone is added to all virtual machines in a group. Pheromone f of a VM j in group k can be calculated by,

$$f_{jk} = \sum_{i=0}^{TG_k} ET_i \tag{9}$$

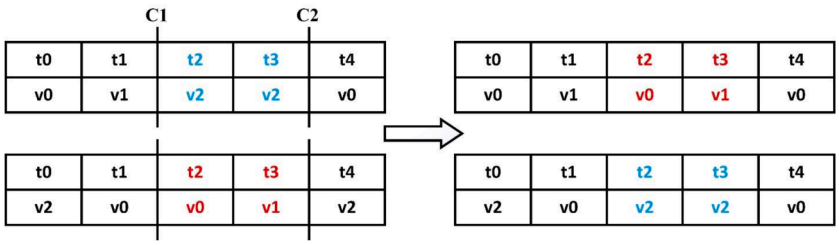

Fig. 2.. Crossover.

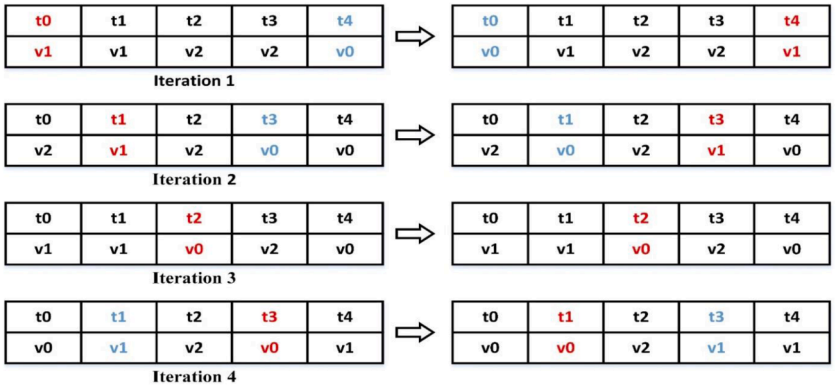

Fig. 3.. Mutation technique.

Where, TG_k is total tasks assigned to a group k, ET_i is the running time of task i. Pheromone of the VM depends on tasks scheduled on it. If a virtual machine has a high pheromone then it means it is a loaded virtual machine and it will delay the execution of new tasks. This helps to detect loaded virtual machines.

4.7. VM selection

To reduce solution space, virtual machines are assigned to each group g_i of tasks. When tasks are scheduled on VMs, a pheromone is added to detect loaded VMs. The algorithm first sorts all virtual machines on the base of pheromone then set of w virtual machines from m virtual machines is selected by,

Algorithm 1.
HAGA

Input: Stop criteria, Control parameters, Tasks list, Virtual machines list, Initial population size				
Output: Optimal tasks to VM mapping				
1: Choose tasks equal to group size TG from set of tasks T				
2: for each group of tasks do				
3:	if group number is 1 then			
4:		select all virtual machines from set V		
5:	else			
6:		Evaporate pheromone$\rho$from all virtual machines		
7:		Sort virtual machines by order of their pheromone level		
8:		Select w virtual machines from set V		
9:	end if			
10:	Initialize population z			
11:	while iterations $<$ stop criteria do			
12:		Calculate fitness of population z		
13:		Sort population by order of their fitness		
14:		Select individuals from the population by roulette selection		
15:		Generate crossover rate and mutation rate		
16:		if crossover rate $<$ threshold then		
17:				do crossover
18:		end if		
19:		if mutation rate $<$ threshold then		
20:				do mutation
21:		end if		
22:		update the population		
23:		Iterations $++$		
24:	end while			
25:	Add pheromone f to all virtual machines selected in a group			
26: end for				
27: update the occupation time of virtual machines.				

$$w = \gamma * m \tag{10}$$

Where, γ is a constant and its range is from 0.1 to 1. Loaded virtual machines have a low chance of selection for the next group of tasks. This helps to minimize workflow execution time and reduces solution space.

4.8. Pheromone evaporation

We add pheromone to virtual machines to detect the load on them. If a virtual machine has more pheromone then there is less chance that the virtual machine will be selected for scheduling in the next group of tasks. But with time, virtual machines finish the execution of tasks and it is necessary to select them for the next scheduling solutions. Hence, we proposed a novel approach for pheromone evaporation ρ, and can be expressed as:

$$\rho = \frac{\sum_{i=0}^{TG_k} ET_i}{TG_k} \tag{11}$$

Where, TG_k is total tasks assigned to a group k, ET_i is the execution time of task i. Pheromone evaporation of a virtual machine can be expressed as:

$$f_i = f_i - \rho \tag{12}$$

Where, f_i is pheromone deposited in a virtual machine i and it should be greater than 0. In this way, pheromone automatically decreases with the execution of tasks on virtual machines. When pheromone evaporates, the virtual machine is again selected in solution space for the next group of tasks. The pseudo code of HAGA is shown in Algorithm 1.

5. Results and discussion

The proposed algorithm, HAGA, is simulated and results are compared for performance evaluation. The proposed algorithm is compared with two well-known scheduling algorithms Max-Min [1], Min-Min [2], and two famous meta-heuristics algorithms Standard genetic algorithm (GA) [7] and Adaptive incremental genetic algorithm (AIGA) [11]. Cloudsim plus [19] simulator is used to simulate algorithms. Cloudsim plus is a simulation tool which is built on the famous Cloudsim simulator, designed for large scale simulation of cloud computing application services and infrastructure.

In the simulation, a set of heterogeneous processors is used for experimentation. Task length ranges from 100 to 1000. Virtual machines instances have the same specification as Amazon elastic compute instance. CU and cost of virtual machines are shown in Table 3. Genetic algorithms give different solutions each time as random factors are involved. For fair experimentation, every experiment is simulated 10 times and an average of their results is presented.

Suppose there are 10 tasks, and their length is [343, 140, 483, 843, 415, 619, 928, 534, 406, 558]. Table 4 shows the running time of each task on a VM. Above mentioned 10 tasks and VM5 to VM9, from Table 3, are used for simulation. Fig. 4 represents the scheduling strategy of tasks on virtual machines by different algorithms. Note that Fig. 4Error! Reference source not found, represents only one possible solution for HAGA. The make span is the maximum time taken by a virtual machine among all virtual machines. The make span of this simulation obtained by Min-Min, Max-Min, GA, AIGA and HAGA is 44, 22, 24, 26, and 18, respectively. The HAGA algorithm has a low make span and fitness value. This is because genetic algorithms evaluate different solutions to find the best solution. HAGA algorithm has a high possibility to find the best solution because its solution space is small as compared to other algorithms.

In the next simulation, the performance of HAGA, AIGA, and GA algorithms is evaluated on different parameters. The performance of genetic algorithms is highly influenced by different parameters like Crossover Rate (CR), Mutation Rate (M.R), and Group Size. Typical values of CR and MR lie between 0.1 and 1. In the HAGA algorithm, another important parameter i.e selection rate (γ) is introduced and its value is also in the range of 0.1–1. To evaluate the performance of the HAGA algorithm on different input parameters, we performed four experiments. The first experiment changes the CR, the second changes MR, the third changes selection

Table 3.
Amazon EC2 virtual machine CU and costs.

No.	Instance type	CU	Costs
0	c5.1large	2	0.6
1	c5.1xlarge	4	0.17
2	c5.2xlarge	8	0.34
3	c5.4xlarge	16	0.68
4	c5.9xlarge	36	1.53
5	c5.18xlarge	72	3.888
6	m5d.8xlarge	32	2.48
7	m5d.12xlarge	48	3.72
8	m5d.16xlarge	64	4.96
9	m5d.24xlarge	96	7.44

Table 4.
Running time of tasks on VM.

Task	VM0	VM1	VM2	VM3	VM4	VM5	VM6	VM7	VM8	VM9
0	171.50	85.75	42.87	21.43	9.52	4.76	10.71	7.14	5.35	3.57
1	70.00	35.00	17.50	8.75	3.88	1.94	4.37	2.91	2.18	1.45
2	241.50	120.75	60.37	30.18	13.41	6.70	15.09	10.06	7.54	5.03
3	421.50	210.75	105.37	52.68	23.41	11.70	26.34	17.56	13.17	8.78
4	207.50	103.75	51.87	25.93	11.52	5.76	12.96	8.64	6.48	4.32
5	309.50	154.75	77.37	38.68	17.19	8.59	19.34	12.89	9.67	6.44
6	464.00	232.00	116.00	58.00	25.77	12.88	29.00	19.33	14.50	9.66
7	267.00	133.50	66.75	33.37	14.83	7.41	16.68	11.12	8.34	5.56
8	203.00	101.50	50.75	25.37	11.27	5.63	12.68	8.45	6.34	4.22
9	279.00	139.50	69.75	34.87	15.50	7.75	17.43	11.62	8.71	5.81

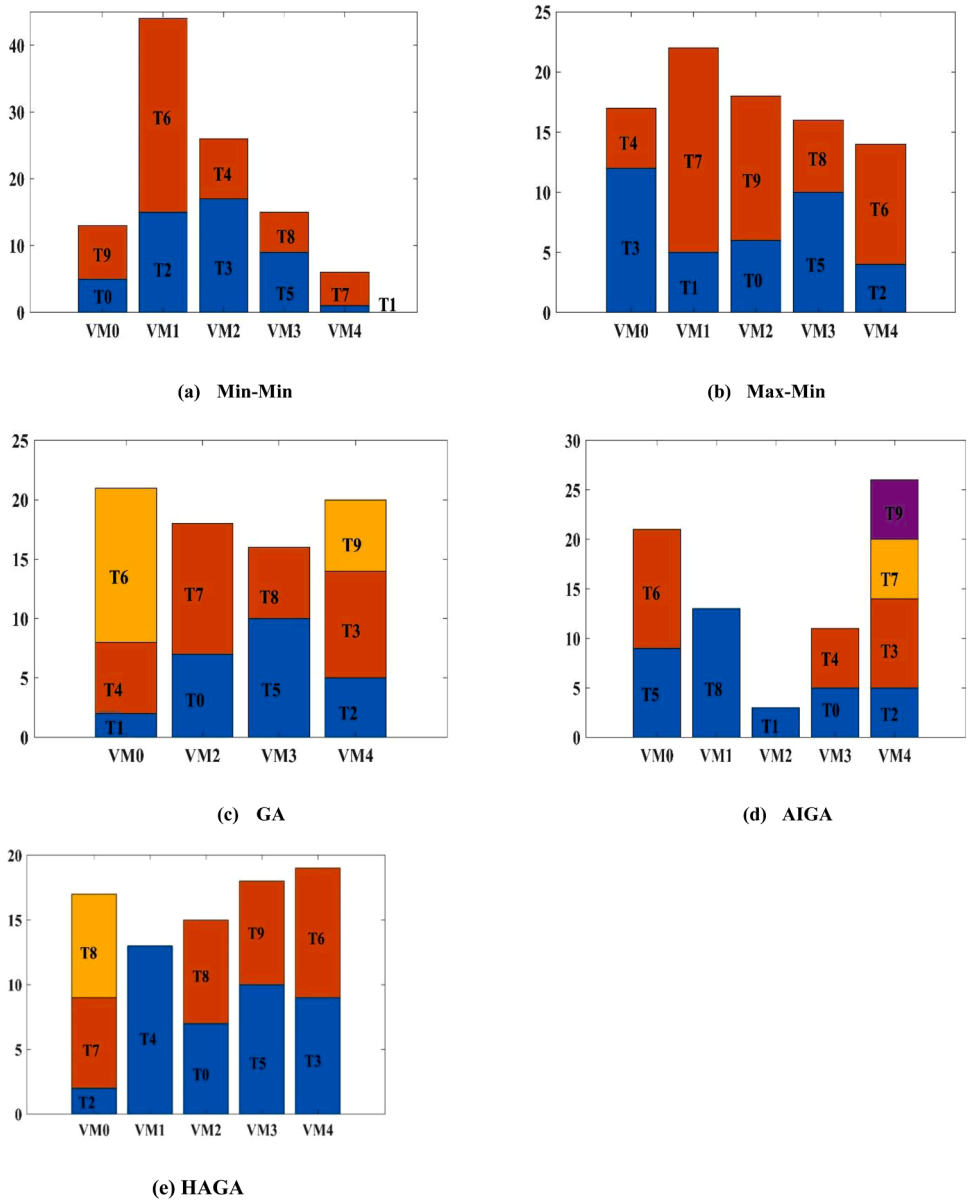

Fig. 4.. Scheduling Scheme by: (a) Min-Min (b) Max-Min (c) GA (d) AIGA (e) HAGA.

rate, and the fourth changes group size. All experiments use 1000 tasks, and results are collected by running each experiment 10 times.

Table 5 represents the optimal solution, when the CR changes (MR and γ are 0.5). It shows that the overall global fitness of HAGA is better than AIGA and GA. Table 6 shows the optimal solution when mutation rate changes (CR and γ is 0.5). It shows that the HAGA algorithm outperforms AIGA and GA. Fig. 5(a) and (b) shows the make span of algorithms according to different CR and MR. It shows that the HAGA algorithm is more flexible than other algorithms. It finds better solutions even if parameters are changed.

shows the fitness values of HAGA algorithms when γ is changed (CR and MR are 0.5). When γlow then fitness value is very high. This is due to the small number of virtual machines selected for a scheduling solution. When the number of virtual machines is too small then a large number of tasks are scheduled to a few virtual machines from set *V*. This increases fitness and makespan of a workflow. When γ is increased, fitness value starts to decrease. This pattern continues until γ is 0.6. After this solution space becomes large and the HAGA algorithm cannot find an optimal solution due to which fitness value starts to increase.

Fig. 6 shows the make span of the HAGA algorithm when γ changes. It shows that the HAGA algorithm finds good solutions when γ ranges from 0.5 to 0.7. Table 8 shows the fitness value of HAGA and the AIGA algorithm when group size changes. It shows that fitness value is high when the group size is too small. When the group size is small solution space increases and algorithms stuck at local optima instead of global optima. When group size increases, solution space decreases and algorithms have a chance to find a good solution. In the last, the fitness value increases with a very small value because the number of tasks selected are nearly equal to virtual machines due to which algorithms have the possibility to stuck at local optima. Fig. 7 shows the makespan of HAGA and AIGA algorithm when group size changes. The results show that HAGA performs better in finding an optimal solution because it has less solution space as compared to AIGA.

In the next simulation, the performance of HAGA, AIGA, GA is measured on different metrics like cost, SLA violation, convergence time, and response time. In this experiment, number of tasks range from 1000 to 10,000, CR and MR are 0.5 and γ is 0.7 and each experiment is simulated 10 times. Each live virtual machine has a cost associated with it and shown in

Table 3 and Fig. 8 represents the data center cost when a running workflow. It can be seen that in the start, all algorithms have the same cost, when the number of tasks increases, the cost of HAGA decreases as compared to other algorithms. This is because the cost of the VM depends on its running time. When tasks are small in numbers, all algorithms find solutions close to global optima due to which their cost is the same.

When tasks increase, the HAGA algorithm finds a better solution due to the small solution space as compared to other algorithms. Due to which the running time of virtual machines is decreased and as a result cost of virtual machines is reduced. Fig. 9 shows the SLA violations of algorithms. It shows that HAGA has fewer violations as compared to other algorithms. As seen earlier, HAGA can find a good scheduling solution. When tasks are executed on a VM, then it can execute them in less time, and hence, there is a less chance of SLA violation.

Fig. 10 gives the convergence curve of algorithms. HAGA algorithm converges better than AIGA and GA. It can be seen that AIGA and GA both converged to local optima and cannot find better fitness solutions, while HAGA converges to a better solution. Fig. 11 represents the response time of algorithms. The response time of the algorithm is the time taken by the algorithm to generate the first response [17].

When the number of tasks is small, the response of all algorithms is close to each other. When tasks are increased, response time of GA increases exponentially, due to the large solution space of GA. Although, AGA and HAGA response time gradually increase but when tasks are very large, then the HAGA algorithm outperforms AIGA because in less solution space HAGA can check different combinations in less time as compared to AIGA. Suppose there are 100 tasks, 6 virtual machines. If we divide tasks into 5 groups and γ= 0.5. GA has $100^6 = 1e + 12$ possible ways to schedule tasks on virtual machines. AIGA has $20^6 + 20^6 + 20^6 + 20^6 + 20^6 = 3.2e + 8$ ways to allocate tasks to virtual machines. In case of HAGA there are $20^6 + 20^3 + 20^3 + 20^3 + 20^3 = 6.4032e + 7$ ways to allocate tasks to virtual machines. It can be seen that HAGA has to check minimum combinations due to which its response time is very low as compared to GA and AIGA. Due to less solution space it takes less time to perform genetic operations and finds an optimal solution.

In the next simulation, we compared the performance of HAGA, AIGA, GA, Min-Min, and Max-Min algorithm on a large scale. Number of tasks range from 10,000 to 50,000 ((M.R CR are 0.5 and γ is 0.7). Table 9 shows the fitness values of all algorithms. Fig. 12 depicts that the make span of the HAGA algorithm is less than AIGA, GA, Min-Min, and Max-Min. When the workload is small, Max-Min and Min-Min perform better. However, with an increase in the workload, HAGA performs better gradually and has less computation time. This is because with an increase in the workload Max-Min and Min-Min algorithms do not perform well. While GA and AIGA have large solution space than HAGA due to which HAGA finds a good scheduling solution as compared to other algorithms (Table 7).

Table 5.
Fitness value when crossover rate changes.

CR	GA	AIGA	HAGA
0.1	1180.2	552.3	418.7
0.2	645.3	574.2	381.4
0.3	777.9	543.3	416.5
0.4	985.4	522.1	396.8
0.5	829.7	547.6	404.2
0.6	677.7	536.6	369.1
0.7	871.6	525.8	403.2
0.8	537.2	537.8	393.3
0.9	684.3	536.5	396.5

Table 6.
Fitness value when mutation rate changes.

MR	GA	AIGA	HAGA
0.1	997.7	560.5	381.5
0.2	992.8	551.4	411.4
0.3	1065.4	500.5	393.5
0.4	1174.4	540.4	405.1
0.5	920.5	523.1	416.5
0.6	1207.5	563.1	395.6
0.7	1152.8	524.3	395.8
0.8	1252.8	550.3	412.8
0.9	1270.9	545.5	400.1

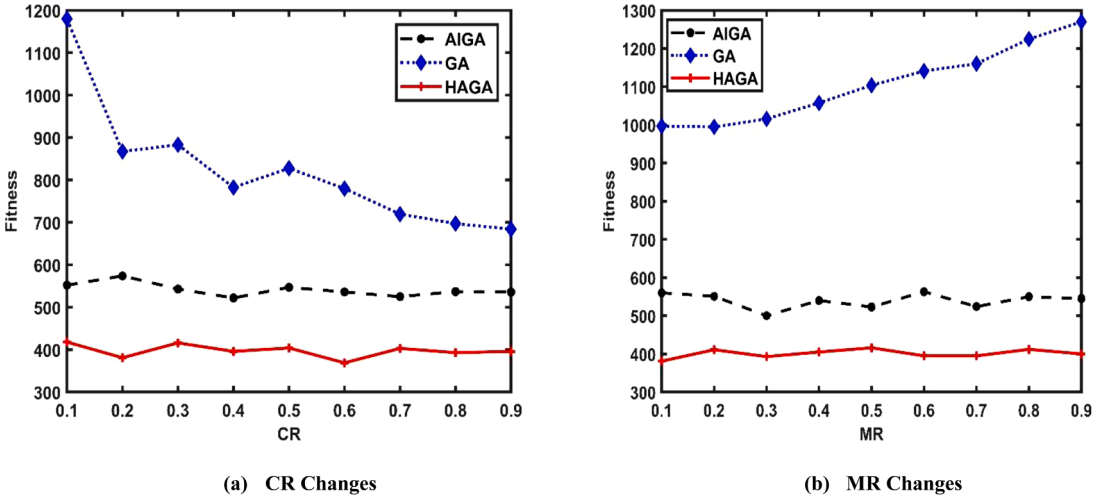

Fig. 5.. Makespan when (a) CR changes (b) MR changes.

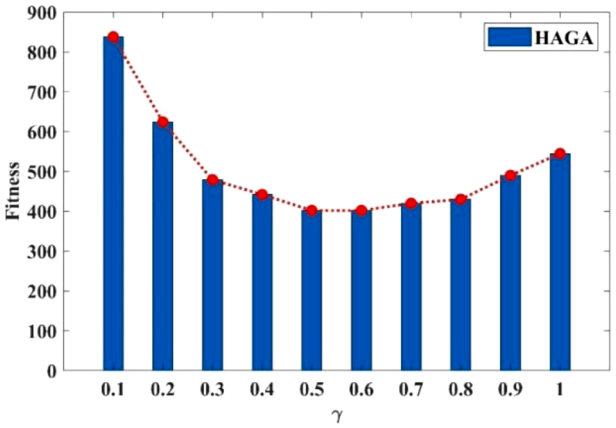

Fig. 6.. Makespan when γ changes.

6. Conclusion and future work

In a cloud computing environment, running time of tasks and workflows affects the performance of the data center. An increase in task execution time results in high user costs and operational costs. Evolutionary algorithms can efficiently solve NP-hard problems. These algorithms have a large solution space and take a lot of time and computational power during genetic operations. Their convergence time increases, and the response time of algorithms also increases exponentially. Proposed algorithm uses a novel scheme to add and evaporate VM pheromone to detect the load on the virtual machine. We redesigned the genetic algorithm and mutation operation and used pheromone to detect loaded virtual machines. If virtual machines are loaded, the proposed algorithm does not

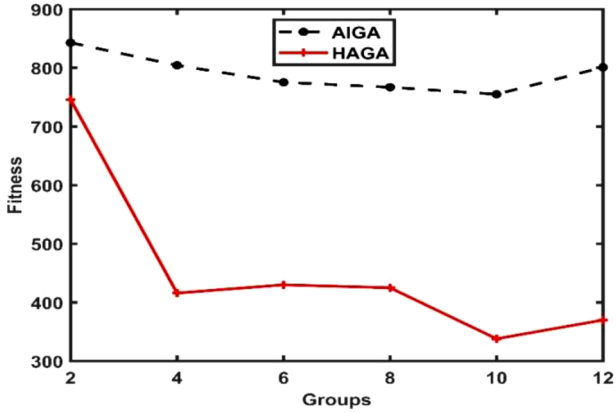

Fig. 7.. Makespan when number of group changes.

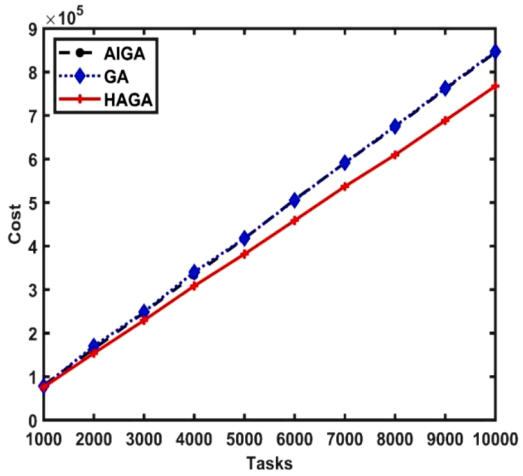

Fig. 8.. Datacenter cost.

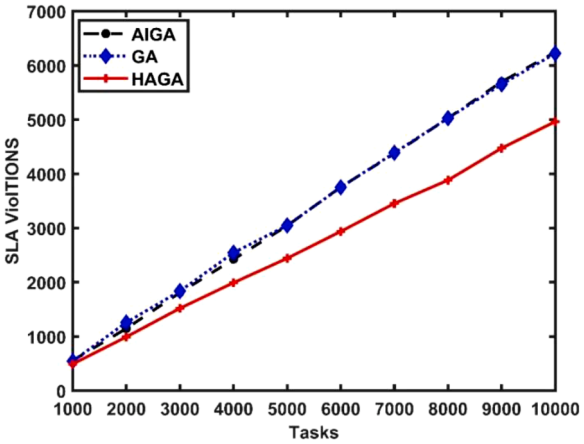

Fig. 9.. Number of SLA violations.

include it in solution space. This results in less solution space due to which the proposed algorithm can find an optimal solution with less computational power. Removal of loaded virtual machines from solution space also reduces execution time because virtual machines in solution space can start execution of next tasks early as compared to loaded virtual machines. We model tasks scheduling

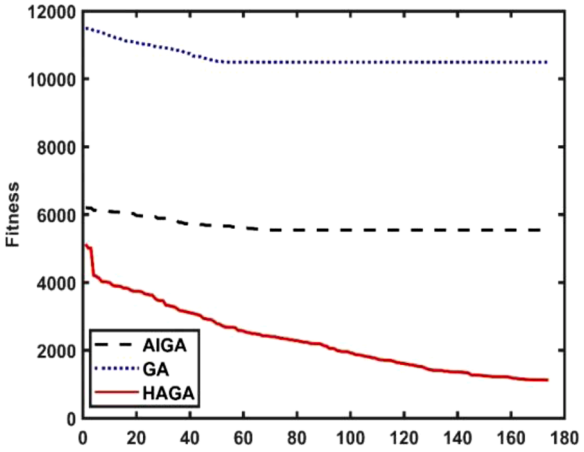

Fig. 10.. Convergence of GA, AIGA, HAGA.

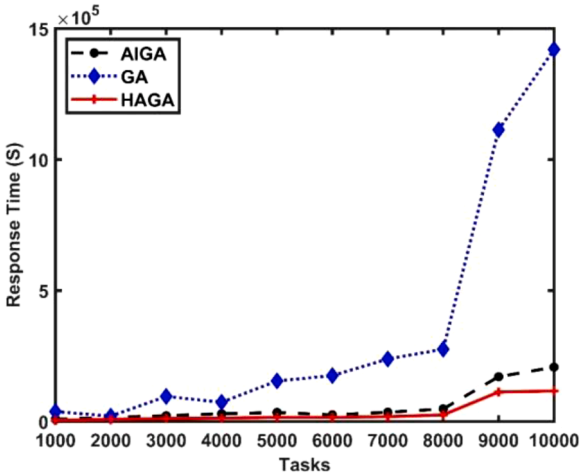

Fig. 11.. Response time of GA, AIGA and HAGA.

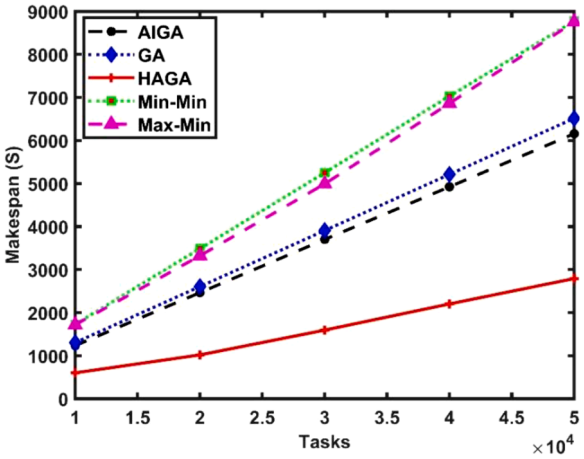

Fig. 12.. Makespan of Max-Min, Min-Min, GA, AIGA and HAGA.

Table 7.
Fitness of optimal solution when γ changes.

γ	Fitness
0.1	838.5
0.2	624.5
0.3	479.4
0.4	442.6
0.5	402.9
0.6	402.8
0.7	420.7
0.8	430.8
0.9	490.8
1.0	545.7

Table 8.
Fitness of optimal solution when group changes.

Group size	AIGA	HAGA
2	843.7	746.8
4	794.2	416.9
6	776.2	430.9
8	732.1	425.4
10	732.0	338.2
12	801.9	370.1

Table 9.
Fitness of optimal solution.

Tasks	Max-Min	Min-Min	GA	AIGA	HAGA
10,000	1718.6	1728.1	1304.5	1233.8	601.5
20,000	3322.0	3488.2	2606.6	2465.7	1018.6
30,000	4991.5	5251.2	3908.4	3700.9	1594.5
40,000	6863.9	7029.4	5212.5	4923.5	2202.4
50,000	8754.8	8776.5	6515.5	6158.4	2729.4

problem as objective optimization. We compared the proposed algorithm with traditional algorithms Min-Min, Max-Min, and genetic algorithms GA, AIGA and performed extensive experimentation. HAGA algorithm steadily outperforms other algorithms in optimizing convergence and make span. We also tested algorithms using different control parameters. Experimental results proved that the HAGA algorithm performs better in terms of minimizing make span, convergence time, response time, SLA violations, and cost of running workflow.

In the future, we aim to manage servers and schedule VMs on the server, through the HAGA algorithm. We also aim to design a mathematical model to address dependencies between tasks. Furthermore, we will evaluate the HAGA algorithm for other parameters such as energy consumption, resource utilization, and scaling of data center.

CRediT authorship contribution statement

Muhammad Sohaib Ajmal: Conceptualization, Methodology, Data curation, Visualization, Software, Validation, Writing – original draft. **Zeshan Iqbal:** Methodology, Data curation, Software, Visualization, Validation. **Farrukh Zeeshan Khan:** Conceptualization, Methodology, Data curation, Visualization. **Muneer Ahmad:** Data curation, Visualization, Software, Validation, Supervision, Writing – original draft. **Iftikhar Ahmad:** Investigation, Methodology, Visualization, Validation. **Brij B. Gupta:** Conceptualization, Investigation, Methodology, Validation, Supervision, Writing – review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Mao Y, Chen X, Li X. Max-min task scheduling algorithm for load balance in cloud computing. Proc. Int. Conf. Comput. Sci. Inf. Technol. 2014:457–65.
[2] He X, Sun X, Von Laszewski G. QoS guided min-min heuristic for grid task scheduling. J Comput Sci Technol 2003;18:442–51.

[3] Mehdi NA, Mamat A, Amer A, Abdul-Mehdi ZT. Minimum completion time for power-aware scheduling in cloud computing. In: Proceedings of the 4th international conference on developments in Esystems engineering DeSE; 2011. p. 484–9. https://doi.org/10.1109/DeSE.2011.30.
[4] Topcuoglu H, Hariri S, Wu MY. Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans Parallel Distrib Syst 2002; 13:260–74. https://doi.org/10.1109/71.993206.
[5] Mazrekaj A, Sheholli A, Minarolli D, Freisleben B. The Experiential Heterogeneous Earliest Finish Time Algorithm for Task Scheduling in Clouds. CLOSER 2019: 371–9. May.
[6] Arif MS, Iqbal Z, Tariq R, Aadil F, Awais M. Parental prioritization-based task scheduling in heterogeneous systems. Arab J Sci Eng 2019;44:3943–52. https://doi.org/10.1007/s13369-018-03698-2.
[7] Chenhong Z, Shanshan Z, Qingfeng L, Jian X, Jicheng H. Independent tasks scheduling based on genetic algorithm in cloud computing. In: Proceedings of the 5th international conference on wireless communications, networking and mobile computing WiCOM; 2009. p. 1–4. https://doi.org/10.1109/WICOM.2009.5301850.
[8] Li G, Wu Z. Ant colony optimization task scheduling algorithm for SWIM based on load balancing. Futur Internet 2019;11:90.
[9] Li X, Yang G. Artificial bee colony algorithm with memory. Appl Soft Comput J 2016;41:362–72. https://doi.org/10.1016/j.asoc.2015.12.046.
[10] Agarwal M, Srivastava GMS. A PSO algorithm based task scheduling in cloud computing. Int J Appl Metaheuristic Comput 2019;10:1–17.
[11] Duan K, Fong S, Siu SWI, Song W, Guan SSU. Adaptive incremental genetic algorithm for task scheduling in cloud environments. Symmetry 2018;10:1–13. https://doi.org/10.3390/sym10050168 (Basel).
[12] Jang SH, Kim TY, Kim JK, Lee JS. The study of genetic algorithm-based task scheduling for cloud computing. Int J Control Autom 2012;5:157–62.
[13] Zuo L, Shu L, Dong S, Zhu C, Hara T. A multi-objective optimization scheduling method based on the ant colony algorithm in cloud computing. IEEE Access 2015;3:2687–99. https://doi.org/10.1109/ACCESS.2015.2508940.
[14] Moon YJ, Yu HC, Gil JM, Lim JB. A slave ants based ant colony optimization algorithm for task scheduling in cloud computing environments. Hum Centric Comput Inf Sci 2017;7. https://doi.org/10.1186/s13673-017-0109-2.
[15] Milan ST, Rajabion L, Darwesh A, Hosseinzadeh M, Navimipour NJ. Priority-based task scheduling method over cloudlet using a swarm intelligence algorithm. Cluster Comput 2019:0123456789. https://doi.org/10.1007/s10586-019-02951-z.
[16] Feller E, Rilling L, Morin C. Energy-aware ant colony based workload placement in clouds. In: Proceedings of the 12th IEEE/ACM international conference on grid computing; 2011. p. 26–33. https://doi.org/10.1109/Grid.2011.13.
[17] Shu W, Wang W, Wang Y. A novel energy-efficient resource allocation algorithm based on immune clonal optimization for green cloud computing. EURASIP J Wirel Commun Netw 2014;2014:1–9. https://doi.org/10.1186/1687-1499-2014-64.
[18] Dashti SE, Rahmani AM. Dynamic VMs placement for energy efficiency by PSO in cloud computing. J Exp Theor Artif Intell 2016;28:97–112. https://doi.org/10.1080/0952813X.2015.1020519.
[19] Filho MCS, Oliveira RL, Monteiro CC, Inácio PRM, Freire MM. CloudSim plus: a cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness. In: Proceeding of the IM IFIP/IEEE symposium on integrated network and service management; 2017. p. 400–6. https://doi.org/10.23919/INM.2017.7987304.
[20] Tewari A, Gupta BB. Secure Timestamp-Based Mutual Authentication Protocol for IoT Devices Using RFID Tags. International Journal on Semantic Web and Information Systems (IJSWIS) 2020;16(3):20–34.
[21] Al-Qerem A, Alauthman M, Almomani A, Jararweh Y, Gupta BB. IoT transaction processing through cooperative concurrency control on fogcloud computing environment. Soft Computing 2020;24(8):5695–711.
[22] Wang H, Li Z, Li Y, Gupta BB, Choi C. Visual saliency guided complex image retrieval. Pattern Recognit Lett 2020;130:64–72.
[23] Ali MB. Multiple perspective of cloud computing adoption determinants in higher education a systematic review. Int J Cloud Appl Comput 2019;9:89–109.
[24] Aouzal K, Hafiddi H, Dahchour M. Policy-driven middleware for multi-tenant SaaS Services configuration. Int J Cloud Appl Comput 2019;9:86–106.
[25] Herzfeldt A, Floerecke S, Ertl C, Krcmar H. Examining the antecedents of cloud service profitability. Int J Cloud Appl Comput 2019;9:37–65.

Muhammad Sohaib Ajmal: is student of MS Computer Science in University of Engineering and Technology Taxila, Pakistan. He is currently working as network design engineer. His research interests are focused on Optimization Algorithms, Networks, Virtualization, Cloud computing and Software Defined Networks.

Zeshan Iqbal received his M.Sc. and Ph.D. Degree in Computer Engineering from University of Engineering and Technology (UET), Taxila, Pakistan, in 2006 and 2014, respectively. Currently he is working as Assistant Professor at Department of Computer Science, UET Taxila, Pakistan. His research interests focus on Distributed Systems, Computer and Network Virtualization, Machine Learning and Software Defined Networks in Cloud.