Genetic Algorithms for Job Scheduling in Cloud Computing*

Mohammed-Albarra HASSAN[1,2], Imed KACEM[1], Sébastien MARTIN[1], Izzeldin M. OSMAN[3]

[1] Université de Lorraine,
LCOMS EA 7306, Metz, 57000, FRANCE
imed.kacem@univ-lorraine.fr, (*corresponding author*)

[2] University of Gezira,
Wadmedani, SUDAN,
barra.hassan@univ-lorraine.fr

[3] University of Sciences and Technology,
Khartoum, SUDAN
izzeldin@acm.org

Abstract: Efficient job scheduling algorithms needed to improve the resource utilization in cloud computing, the role of a good scheduling algorithm on cloud computing is to minimize the total completion time for last job on the system. In this paper, we present a genetic-based task scheduling algorithms in order to minimize Maximum Completion Time *Makespan*. These algorithms combines different techniques such as list scheduling and earliest completion time (ECT) with genetic algorithm. We reviewed, evaluated and compared the proposed algorithms against one of the well-known Genetic Algorithms available in the literature, which has been proposed for task scheduling problem on heterogeneous computing systems. After an exhaustive computational analysis we identify that the proposed Genetic algorithms show a good performance overcoming the evaluated method in different problem sizes and complexity for a large benchmark set of instances.

Keywords: Task scheduling, Genetic Algorithm, Cloud Computing, Unrelated Parallel machines with precedence Constraints.

1. Introduction

In Cloud environments a task scheduling is a process that manages and maps the execution of inter-dependent tasks on the data centers (resources) [6]. It allocates appropriate tasks to the virtual resources which is virtual machines (VMs) so the execution is often completed to satisfy objective functions imposed by customers. Efficient task scheduling algorithm will have important impact on the performance of the system. The scheduling problem in cloud computing can be generalized as an unrelated parallel machine with different speeds and precedence constraints. We consider VMs as an unrelated parallel machine because the cloud computing providers offer their services virtually by sharing their physical resources through a large number of virtual machines in parallel. These virtual machines, allocated with different CPU capacities, so it can be considered as unrelated parallel machines.

In cloud computing users may face hundreds of thousands of virtualized resources to utilize. It is hard to allocate user's tasks on the available resources. Due to the virtualization properties, cloud computing leaves task scheduling complexity to the virtual machine layer through resource virtualization.

Hence, to allocate the resources to each task efficiently, scheduling plays more important role in cloud computing [3]. It is quite difficult to achieve an optimal solution with traditional optimization methods. Mathematical optimization techniques can give an optimal solution for a reasonably sized problem, however, in the case of a large scale problem, their application is limited [13]. Dispatching rules (LPT, SPT, EDD,...) are suitable only for small scale problems and no single dispatching rule guarantees good result in various problems [5]. Research efforts in scheduling are concentrated on heuristic approaches. Many

* This paper recalls the Task Scheduling Genetic Algorithm (GATS), published in [20]. In the current paper, we improve GATS and we propose an advanced version GATS+. Moreover, we propose two new algorithms: Genetic Algorithm Based on Cut-point (GACP) and Genetic Algorithm Based on The List of Available Jobs (GAAV), as well as its improved version (GAAV+). We also tested and compared different versions of these genetic algorithms (GAAV -> GATS, GAAV+ -> GATS, GATS -> GAAV, GATS -> GAAV+).

heuristics and meta-heuristics have been proposed such as simulated annealing (SA), tabu search, branch and bound (B&B) and genetic algorithm (GA) [3]. Among these various approaches to different scheduling problems, there has been an increasing interest in applying GA in view of its adaptability. The important difference between GA and other heuristics is that GA maintains a set of solutions (population) rather than a unique solution, which leads to a better diversity.

This paper considers the problem of task scheduling in cloud computing as the problem of unrelated parallel machines with precedence constraints in order to minimize makespan (C_{max}). In scheduling problems, C_{max} is equivalent to the completion time of the last task leaving the system. The small C_{max} usually implies a high utilization. Therefore, reducing the C_{max} should also lead to a higher throughput rate [5]. Three genetic algorithms have been applied to solve this problem. The rest of the paper is organized as follows. Section 2 reports the literature review. In Section 3, we formulate the problem. Our genetic algorithms are represented in Section 4. In Section 5 we discussed the results, and Section 6 concludes the paper.

2. Literature Review

The task scheduling problem in the distributed systems is known to be NP-hard [2], since, for allocating n jobs to m virtual machines (VMs), the number of allocation will be $|n|^{|m|}$ and the number of states for running will be $n!$. One of the goals of scheduling is to determine an assignment of jobs to computing machines in order to optimize the completion time of the final task in the system. Job scheduling problem in heterogeneous distributed system like cloud computing [17] has been widely studied in the last few years. The job scheduling problem has two forms: *static* and *dynamic*. When all information needed for scheduling, such as execution times of jobs and data dependencies between jobs are known in advance, the scheduling problem is described as *static*. This type of scheduling jobs place during compile time. On the other hand, in the *dynamic* model jobs are allocated to processors upon their arrival, and scheduling decisions must be made at run time [18], [19]. In this section we focus our attention on the available algorithms for static scheduling in cloud environment and unrelated parallel machine with precedence constraints scheduling problem. In the literature numerous approaches have been developed for solving this problem (heuristic-based and evolutionary-based algorithms). A survey on scheduling in cloud computing can be found in [6] and [3]. Different methods for solving this problem exist. Some researchers proposed efficient meta-heuristics based on genetic algorithm: Zhou *et al* in [9] proposed a genetic algorithm based on earliest completion time (ECT) to minimize completion time (we represent it in sub section 4.1. Arash and Yalda also developed hybrid genetic algorithm for work flow scheduling in cloud system (HSGA). It merges best-fit and Round Robin methods to make an optimal initial population to obtain a good solution quickly. HSGA at first makes job prioritization in complex graph considering their impact on others. A particle swarm optimization (PSO) have used in [11] for workflow scheduling in cloud environment, which considers not only execution cost but also the cost for transmitting dependent data. In [12] a PSO is also formulated as a model for the multi-objective task assignment to optimize the time and cost. To the best of our knowledge, none of the existing Genetic algorithms have considered the idea of scheduling jobs with a high number of successors in order to optimize the makespan. Unrelated parallel machine scheduling problem receives a great deal of attention in the academic and engineering circle. The literature on parallel machine scheduling is fairly large, we focus mainly on the non-preemptive unrelated parallel machine problem with precedence constraints to minimize makespan criterion. There are many applications for this scheduling problem specially in distributed computing systems [12],[9]. Several heuristics and meta-heuristics have been proposed to solve this problem for optimizing different objectives. In [15] Vallada and Ruiz proposed a genetic algorithm to minimize the makespan. Their GA includes a fast local search and a local search enhanced crossover operator. In [13] Balin proposed a new crossover operator for genetic algorithm to minimize makespan. His algorithm achieved a high computational speed for large-scale problems. In [14] Tavakkoli-Moghaddam *et al* proposed a genetic algorithm to solve bi-objective unrelated parallel machine scheduling problem. In this paper, we deal with the problem of job scheduling in cloud environment and we generalized this problem as an unrelated parallel machines scheduling with precedence constraints for optimizing the makespan. We proposed four genetic algorithms. Three of them show a very good performance for small and medium benchmarks.

3. Problem Formulation

The problem under consideration is to schedule n jobs on m machines which are arranged in parallel with the aim of minimizing the total completion time. Let J be the set of the jobs and M be the set of the parallel machines. A precedence constraint between two jobs *j₁* and *j₂* is denoted by ($j_1 = j_2$) and it requires that job *j₂* cannot start to be processed until job *j₁* will finish its processing. The type of the precedence constraint is a graph type, which is denoted by *D = (V, A)*, where *V* is the set of vertices associated at each job and denotes the set of edges associated with each precedence constraint. We called this graph the precedence graph. We take also the case where $\{\upsilon, \varpi, \omega\} \subseteq V$ such that *υ* before ϖ and ϖ before *ω* then *υ* before *ω*. We consider also the speeds for all machines denoted by σ_ι, where $\iota \in M$. Every job $\varphi \in \vartheta$ has a processing time π_φ and its effective processing time depends on the selected machine *ι*, where $\pi_{\iota\varphi} = \pi_\varphi \times \sigma_\iota$. Each machine $\iota \in M$ cannot process more than one job at a given time. Furthermore, machines have different speeds and preemption of jobs is not allowed. According to the well-known *α/β*|γ scheduling problem classification scheme proposed initially by Graham *et al* [1], this problem can be denoted as $P|prec|X_{\mu\alpha\xi}$. We denote by C_ι the completion time of machine *ι*, where $\iota \in M$, and denote by X_φ the completion time of job *φ*, where $\varphi \in \vartheta$, in the rest of this paper. Thus, the problem can be reduced to the following mathematical formulation proposed in [11]:

For all $i \in M,\ j \in J,\ r \in \{1,\dots,n\}$

$$x_{jir} \begin{cases} 1 & \text{if job } j \text{ in the position } r \text{ on machine } i \\ 0 & \text{otherwise} \end{cases}$$

For all $j \in J, C_i \in \aleph^+$ is the completion time of *j*. $C_{max} \in \aleph^+$ is the maximum of C_j.

$$\min C_{max}\ , \tag{1}$$

$$C_j \leq C_{max}, \qquad \forall j \in J\ , \tag{2}$$

$$\sum_{i \in M} \sum_{r \in \{1,\dots,n\}} x_{jir} = 1, \qquad \forall j \in J\ , \tag{3}$$

$$\sum_{j \in J} x_{jir} \leq 1, \qquad \forall i \in M, \forall r \in \{1,\dots,n\}\ , \tag{4}$$

$$\sum_{j_1 \in J} x_{j_1 ir} - \sum_{j_2 \in J} x_{j_2 ir-1} \leq 0 \qquad \forall i \in M, \forall r \in \{2,\dots,n\}\ , \tag{5}$$

$$C_{j_1} - C_{j_2} + C(2 - x_{j_1 ir} - x_{j_2 ir-1}) \geq p_{j_1 i}, \quad \forall i \in M, \forall r \in \{2,\dots,n\}, \forall j1 \neq j_2 \in J\ , \tag{6}$$

$$C_j \geq \sum_{r \in \{1,\dots,n\}} p_{ji} x_{jir}, \qquad \forall i \in M, \forall j \in J\ , \tag{7}$$

$$\sum_{r \in \{1,\dots,n\}} \sum_{i \in M} p_{j_1 i} x_{j_1 ir} \leq C_{j_1} - C_{j_2}, \quad \forall (j1, j_2) \in A\ , \tag{8}$$

$$x_{jir} \in \{0,1\}, \quad \forall j \in J, \forall i \in M, \forall r \in \{1,\dots,n\}\ , \tag{9}$$

$$C_j \geq 0, \qquad \forall j \in J\ . \tag{10}$$

where *C* in (6) is a large positive number.

The objective function minimizes the makespan. Inequalities (2) ensure that the global makespan is greater or equal the completion time of all the jobs. Inequalities (3) ensure that each job is assigned to one of the existing positions on the machines. Inequalities (4) guarantee that at most one job can be assigned to each position. Inequalities (5) ensure that until one position on a machine is occupied, jobs are not assigned to subsequent positions. Inequalities (6) ensure that the completion time of a job in sequence on a machine is at least equal to the sum of the completion time of the preceding job and the processing time of the present job. Inequalities (7) measure completion time for each job on each machine. Inequalities (8) observe precedence relationships. Inequalities (9) define the type of decision variables. Inequalities (10) bounds C_j. This mathematical model will be used later in computational experiments.

4. Genetic Algorithm (GA)

The GA is a general search approach inspired by the process of the natural evolution. It has been widely exploited for solving combinatorial optimization problems [16]. It is introduced in the 1970s by Holland [15]. The basic idea of our algorithm is to exploit the advantages of the both of the evolutionary and heuristic based algorithms while avoiding their drawbacks. A potential solution to this problem will be represented as a chromosome containing a series of genes, its fitness value is related to its objective function and constraints for that solution.

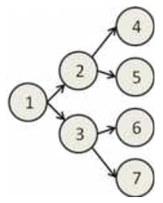


Figure 1. Precedence constraints

The population P of generation g, denoted by (P_g), consists of a set of chromosomes. GA utilizes a population of solutions in its search in order to find a better solution. The efficiency of GA depends largely on the presentation of a chromosome which is composed of a series of genes. In this paper we proposed two encoding methods random ordering method and list scheduling method to formulate the chromosome. During each iteration step (Generation), genetic operations, that is *crossover*, *mutation* and *selection* are processed to search potential better solutions. *Crossover* combines two chromosomes to generate next generation P_{g+1}. Mutation reorganizes the structure of genes in a chromosome randomly so that a new combination of genes may appear in the next generation. It manages the search by jumping form out of local optimal solutions. Reproduction is to copy a chromosome to the next generation directly so that chromosomes from various generations could cooperate in the evolution and the quality of the population may be improved after each generation [13]. The general schema of GA is illustrated in the code below.

The instances in Figure 1 and Table 1, will be considered for numerical example. The rest of

Algorithm 1:
Pseudo-code of a genetic algorithm.

```
Generate randomly an initial population of solutions.
Calculate the fitness of the initial population.
while Stopping Criteria Not Satisfied do
        Select a pair of parents based on fitness.
        Create two offspring using crossover.
        Apply mutation to each child.
        Evaluate the mutated offspring.
All the offspring will be the new population.
end
```

genetic algorithms tested and compared under the following proposed benchmark of instances.

The processing times are uniformly distributed between 1 and 100 as it is common in the literature [4]. We keep the processing time for a specific size of problem, and we changed the density of graph. Regarding the precedence constraints,

Table 1. Processing Time

J	1	2	3	4	5	6	7
Processing Time	1	2	3	1	2	3	4

we generated three subsets of DAG where the graph density is high, medium and low respectively, with the following combinations of number of jobs $n = \{50, 100, 200, 500\}$ and number of machines $m = \{2, 5, 10, 20\}$. The speed of machines generated randomly between 10 and 20. In total $4\times4\times3$ instances are generated.

Table 2 Represents the Average and the standard deviation of jobs processing time. Regarding to the graph density (GD) is calculated as follow: $GD=\frac{|E|}{|V|(|V|-1)}$ where E is the set of edges associated with precedence constraints between jobs, and V is the set of vertices associated with jobs.

Table 2. Average and Standard deviation for Instances Processing time

J	***Average***	***Stdev***
50	48.11	25.98
100	52.26	28.62
200	50.81	27.61
500	49.72	27.70

We generated the instances of three density sets {low, medium, high} corresponding to the values {0.1, 0.15, 0.25} respectively. Regarding the parameters of the algorithms, we tested the combinations of population size $P_{size} \in \{50,100\}$ crossover ratio $P_c \in \{1,0.5\}$ and mutation ratio $P_m \in \{1, 0.5\}$. The stopping criterion is set to the computation time of 1, 10, 60 and 600 seconds for all combinations of instance.

5 Modelling the Problem using Genetic Algorithm

In this section, we present the modeling of our GAs for DAGs in cloud environment. These scheduling algorithms effectively addresses the issues of minimizing the makespan.

5.1 Task scheduling genetic algorithm (GATS)

This GA has been proposed on heterogeneous computing systems by Zhou *et al* in [9]. They call it, "*task scheduling based Genetic Algorithm (GATS)*". It has been modeled as follows: The linear order of all jobs forms the chromosome. Each chromosome represents a solution for the problem by scheduling the jobs in the order given by the permutation, the order of the jobs should be a valid topological order as the associated nodes in the DAG, where start nodes should be placed at the beginning of the chromosome, while the exit nodes should be placed at the end. The initial population is produced by making a random perturbation to the order of jobs in the first chromosome to produce a valid chromosome, until the desired size of the initial population reached. Indeed, a linear crossover from a single random position applied to the two selected parents. The mutation operation operated for all individuals of the new population considering the precedence constraints topologically.

Then, the objective function is evaluated by using the Earliest Completion Time (ECT) technique, which schedules a candidate job onto machines (processors) on which the completion time of the job is the earliest. The robust characteristic in this GA is the generation of a valid chromosome in the initial population.

At the next generation, we modify GATS in GATS+ by just making a random mutation for two genes selected randomly and if the candidate chromosome is not valid, then we throw it out by assigning a big value as Cmax to this candidate. Since we have valid chromosomes in the initial population, the robust characteristics of the GATS can still be maintained and we will not spend a lot of computation time in the mutation operator. This small change increases the chance to find a best result, especially when the computation time is less than one minute, because GATS spends a lot of time in mutation procedure if the candidate is not valid. Table 3 shows the results obtained by GATS and GATS+ in one second with P_{size} = 100, P_c = 1.0 and P_m = 0.5. The dashed results means that GATS does not find a solution during one second and also when we run the instances for 10 seconds GATS cannot find a solution with the problems of large number of instances in all of the three density sets.

Table 3. GATS vs GATS$^+$ C_{max} Comparison

Inst.	**L-Density**		**H-Density**	
m-n	**GATS**	**GATS$^+$**	**GATS**	**GATS$^+$**
2- 50	17274	17251	17274	17927
2- 100	37667	37667	37667	38636
2 -200	-	74590	-	75702
2 -500	-	181679	-	183294
5 -50	10349	10527	7813	7355
5 -100	20791	19766	20049	19747
5 -200	-	37731	-	32981
5 -500	-	88668	-	82615
10 -50	7820	7820	6460	6460
10 -100	10860	10768	10970	10900
10 -200	-	22236	-	18182
10 -500	-	48365	-	40901
20 -50	6238	6090	4540	4540
20 -100	9775	9604	9628	9593
20 -200	-	20892	-	15189
20 -500	-	39090	-	30158

5.2 Genetic Algorithm based on Cut-point (GACP)

For this genetic algorithm (GACP), the chromosome coding composed of two rows: the first represents a valid order of jobs according to the precedence constraints, and the second row gives an information on job positioning according to the cut-point.

We generate $m-1$ random cut-points $(cp)=cp_1, cp_2, \ldots, cp_{m-1}$, to assign jobs to its VMs. The solution provided as follows: The sequence of jobs from j_0 to j_{cp_1} will be assigned on VM_1 and the sequence of jobs from j_{cp_1+1} to j_{cp_2} will be assigned on VM_2 and the sequence of jobs from j_{cp_2+1} to j_{cp_3} on VM_3 and so on. In other words, we assign a valid sub-sequence of a random length of jobs on a specific VM.

In this genetic algorithm we carried out one point *crossover* between two parents and an exchange between two random points carried as *mutation* operator. However, this genetic algorithm gave bad results. The best result obtained by GACP is at least two times the results obtained by GATS.

5.3 Genetic Algorithm based on the list of Available jobs (GAAV)

In this section we will propose a simple idea to generate the population with lowest computational cost, where the chromosome coding depends on VMs and places the job in its associated VM, and the computational efforts will be taken in the evaluation function. In this section we will describe our second genetic algorithm, based on the list of available jobs (GAAV), which depends mainly on the available-list scheduling heuristic.

5.3.1 Coding an initial population

The assignment of VMs to the list of jobs is a candidate solution to the problem. Therefore, the chromosome can be represented by a linear list of integers, each integer representing a VM, here mi considered as gene. The series of genes with the length of n are generated randomly by assigning each job of J to a random mi from the set of M. Figure 2 shows the chromosome representation for ten jobs on five VMs.

1	4	2	1	3	3	5	2	1	3

Figure 2. Chromosome representation for GAAV

In GAAV there is no computational effort to produce the initial population (*IP*) because it is produced by making a random number of permutations to the integer-list to produce a chromosome until the size of *IP* (IP_{size}) reached. Hence, all chromosomes give a valid solution.

5.3.2 Fitness evaluation

In GAAV, to evaluate the chromosome, first we search the virtual machine with minimum completion time Ci. For this machine we take its available jobs according to the precedence constraints, from these available jobs we schedule the job with the maximum number of successors will be placed to the selected virtual machine first. Then, we update Available list, and search again for machine with the minimum completion time and repeat this process until we finish the evaluation process. Simply, at each placement iteration we select the machine with minimum completion time and its available job with highest number of successors. Then, we assign the job which could lead to a late schedule of some jobs in the future to its VM, maybe this job will affect the C_{max} of the whole system. At the end of this process, a valid schedule will be obtained and the fitness function (C_{max}) also will be calculated. Algorithm 2 illustrates the GAAV fitness function steps.

Algorithm 2.
Fitness Evaluation Function for GAAV.

```
Let Available be the current set of jobs without
predecessors.
while |Available| > 0 do
    Selectedmachine =
      the machine with   minimum $C_i$.
    Selectedjob=
        the job of greater number of successors in
        Selectedmachine.
    Add Selectedjob to Selectedmachine
    Update $C_i$.
    Update Available.
end
```

5.3.3 Crossover

The matter of replacing some of the genes in one parent by corresponding ones on the other parent is known as crossover. Here this operator is carried out based on a linear crossover from a single point. This operator is applied to the selected parents (parent1, parent2), then a new two offsprings are obtained as offspring1 and offspring2, the crossover operated between two chromosomes one with higher fitness value and the other with the P_c ratio. Figure 3 illustrates the crossover operator.

5.3.4 Mutation

Mutation can be thought as an effectively escape method for premature convergence by randomly change the value of an individual. For maintaining the feasibility of the new generated individual. During the mutation process, one gene selected randomly and we put it on a different random mi from the set of M to obtain a new offspring, Figure 4 represents the mutation operator.

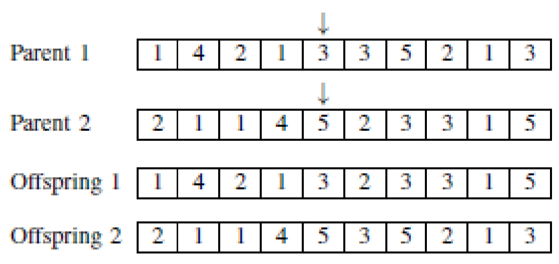


Figure 3. GAAV Crossover Operator

5.3.5 Selection

Finally, the best chromosome of the first population is stored as in a linear ranking.

5.3.6 Stopping rule

The Genetic Algorithm is stopped when the execution time is greater than the maximum execution time allowed.

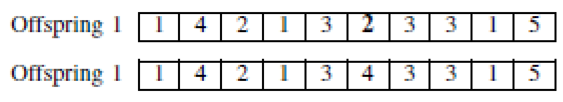


Figure 4. GAAV Mutation Operator

Figure 5, represents an example of chromosome encoding solution for GATS, which is (1-2-3-6-4-7-5), and a chromosome encoding solution for GAAV, which is (0-1-0-1-0-1-0), for the instances in Table 1 according to the precedence constraints in Figure 1, run on two VMs of different speeds which are: $s_1 = 1, s_2 = 2$.

5.4 Genetic Algorithm (GAAV$^+$)

When GATS depends mainly on ECT technique and GAAV based on the local density of the DAG, according to the effectiveness of these two techniques in the optimization of scheduling unrelated parallel machine problem with precedence constraints, we combined these two techniques in GAAV$^+$.

In this genetic algorithm GAAV$^+$, the modification occurred in the Fitness

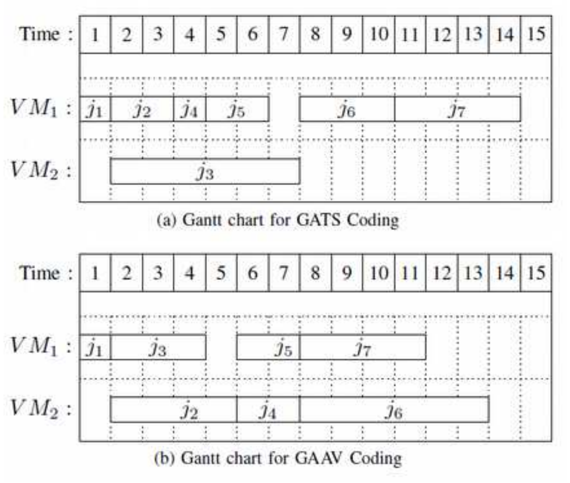

Time : 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
VM_1 :
j_1 j_2 j_4 j_5 j_6 j_7
VM_2 :
j_3
(a) Gantt chart for GATS Coding
Time : 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
VM_1 :
j_1 j_3 j_5 j_7
VM_2 :
j_2 j_4 j_6
(b) Gantt chart for GAAV Coding

Figure 5. Example of the GATS and GAAV encoding

Evaluation. Therefore, the chromosome representation as in Figure 2, according to this coding we know the VM for each job, this fitness evaluation will select the job in machine that will give the minimum C_j from the available jobs $j \in AV$, where AV is the current set of jobs without predecessors, this is ECT technique. From the other hand, at the same time we considered the number of successors of this job, and this is the local density of the DAG technique. Thus, the evaluation can be taken by selecting job j from AV with the minimum value produced by the following function:

$$\alpha \times (C_j) - (1-\alpha) \times |Succ_j|$$

where $\alpha \in [1,0]$ and $|Succ_j|$ is the number of successors of job j. We schedule the job with the minimum value of this function first to generate a valid schedule for calculating C_{max}.

6. Experiments and Results

This section presents the experiment setup and the results for GATS, GATS$^+$, GAAV, and GAAV$^+$. A set of simulations have been performed on Dell Intel, core i5 running at 3.4 GHz, and 8 GB of RAM. The GAs have been coded in C++, compiled with g++ compiler, and tested under Ubuntu 14.02 64-OS. The entries in the Table 5 and 6 are:

– m : number of machines,
– n : number of jobs,
– GATS : C_{max} value for GATS,

– GATS$^+$: C_{max} value for GATS$^+$,
– GAAV : X_{max} value for GAAV,
– GAAV$^+$: X_{max} value for GAAV$^+$,

Genetic search is implemented through genetic operators. Tables 5 and 6 show the results given by our proposed GAs (GATS$^+$, GAAV, and GAAV$^+$) compared to GATS.

Table 5. Makespan(C_{max})for the proposed algorithms and GATS (H density)

Inst.	**L- Density**			
m-n	**GATS**	**GATS$^+$**	**GAAV**	**GAAV$^+$**
2- 50	17274	17251	17251	17251
2- 100	37667	37667	37492	37521
2 -200	73839	74508	73024	73412
2 -500	180246	180089	179612	216190
5 -50	7813	7416	7649	7438
5 -100	18980	18837	18689	18020
5 -200	31902	32342	33652	31600
5 -500	80214	80537	89516	124257
10 -50	6560	6460	6493	6460
10 -100	10632	10871	10264	9544
10 -200	17030	17558	17600	17998
10 -500	38147	38649	45846	75764
20 -50	4540	4540	4540	4540
20 -100	9320	9320	9320	9320
20 -200	14092	14340	15804	13735
20 -500	27765	28179	33921	50949

From this simulation study we fixed the parameters with the combination of (100, 1.0, 0.5, 600), Population size, Crossover ratio, mutation ratio and the computation time respectively. We have tested out different values of α in GAAV$^+$ to find the best value. Therefore, we took α = 0.5. From the results we have noticed that when population size in GAAV is larger than 100, any increase of it has no significant influence on the performance of the genetic algorithm.

In Table 6 we can see the genetic algorithm GAAV can improve 31% of the solutions obtained by GATS in low density problems, 43% in medium density problems and 43% in high density problems. One other interesting outcome is that GAAV can be considered as an efficient algorithm with the problems of small and medium number of VMs. GAAV$^+$ mostly outperforms GATS when the number of jobs 100 and 200 in high density. This may improve 50% of solutions obtained by GATS. It can also improve 50% of medium density problems and 31% of low density problems. If we focused our attention to genetic algorithm GATS$^+$, we can see that for low density showed a good performance and for medium and high density problems is really far from the best solutions, because when we thrown out the invalid candidates we lost some information about some generations. According to the GATS operators behaviors, it always needs more time than the specified stop criterion, with the problems of large numbers of machines and jobs.

Another interesting factor to study in the experiments is the count of best solutions obtained by GAs. In Figure 6 we can notice that GATS$^+$ can find maximum number of best solutions overall instances in one second and ten seconds, whereas GATS cannot find best solution within the specified computation time, we noticed that GATS cannot obtain solution when we run it for 1 and 10 seconds, it needs at least 77 seconds to obtain solution with few number of iterations. We also noticed that GATS need a lot of time to find the first population and for other genetic algorithm operators. We can also see the similarity of a behavior for GATS and GATS+ when we run them for 10 and 60 seconds with the improvement of GATS+. Therefore, we can say GATS+ outperforms GATS in terms of best solutions for sizes and densities. In Figure 6 it is clear that GAAV+ has a positive relationship with the computation time, and has the ability to improve the counts of best solutions for different problems. The other positive thing is that, it can also obtain a solution within the specified computation time.

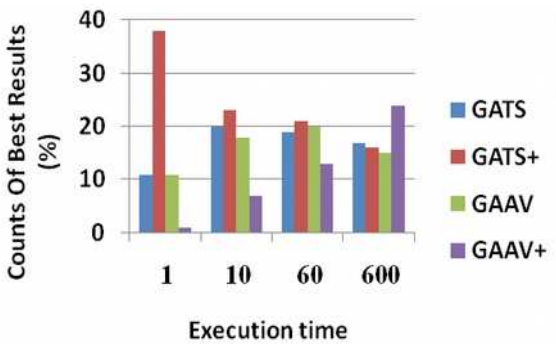


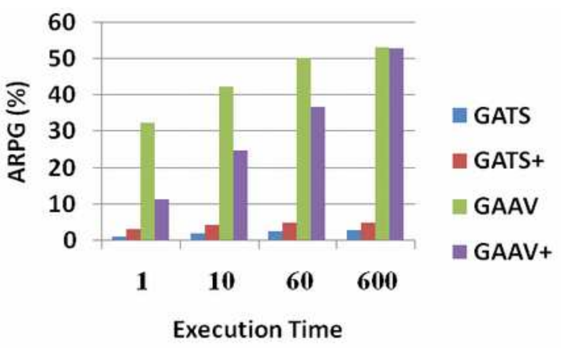
Figure 6. Counts of best results

Table 6. Makespan(C_{max})for the proposed algorithms and GATS, (L&M Density)

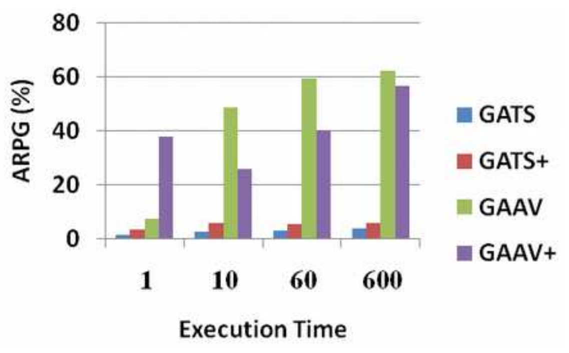
Inst.	**H- Density**				**M-Density**			
m-n	**GATS**	**GATS$^+$**	**GAAV**	**GAAV$^+$**	**GATS**	**GATS$^+$**	**GAAV**	**GAAV$^+$**
2- 50	17927	17927	17927	17927	17251	17251	17251	17251
2- 100	38577	38577	37790	37539	37661	37667	37505	37492
2 -200	75595	75682	74659	74474	73847	73956	73210	74850
2 -500	180908	180861	180849	195618	180375	180466	179120	190116
5 -50	10349	9672	10073	9591	8318	8228	7997	8013
5 -100	19921	19776	19921	19320	18467	18213	17793	17640
5 -200	36521	36645	36036	36205	35621	35577	35575	34890
5 -500	85015	85268	92482	114689	80365	81039	87801	113390
10 -50	7820	7820	7820	7820	4850	4850	4887	4850
10 -100	10494	10591	10378	10310	10580	10552	10118	9898
10 -200	20656	21755	22980	21803	18858	19521	20010	18148
10 -500	44935	46714	57585	73291	40309	41062	48073	70042
20 -50	6156	6090	6156	6090	6430	6430	6430	6430
20 -100	9310	9431	9801	9495	8206	8141	8186	8198
20 -200	20112	19964	21151	20707	14477	14348	15395	14430
20 -500	36364	36346	43463	54919	30117	31063	41631	51731

The efficiency of GAAV appears when we run it for one second; it can always obtain the best solution for the minimum and medium problems.

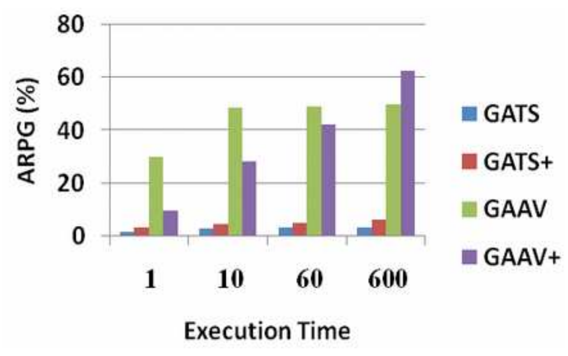
Figure 8 indicates the Average Relative Percentage Gap (ARPG) between the first and the best solution for genetic algorithm. Indeed, ARPG is computed as follows:



(a) High Density



(b) Medium Density



(c) Low Density

Figure 8. Average Relatives Percentage Gap

$100 \times \frac{C_{max}^{First} - C_{max}^{Best}}{C_{max}^{Best}}$, where C_{max}^{Best} is the best known C_{max}, obtained by the given GA, and C_{max}^{First} is the first C_{max} obtained by this GA. We noticed that GATS with low, medium and high density DAG problems cannot improve its solutions. This means GATS starts with a good initial population and the computation time will not affect this solution positively.

This behavior inherited also by GATS$^+$, whereas GAAV and GAAV$^+$ start with random solutions, but they can obtain a better solutions when we increase the computation time.

Figure 7 represents the convergence traces for processing the problem of high density of a randomly generated DAG with 5 VMs and 100 jobs. It can be observed from this figure GAAV$^+$ decreases quickly. GAAV also shows a quick decreasing and provides a best solution when it runs for 10 and 60 seconds.

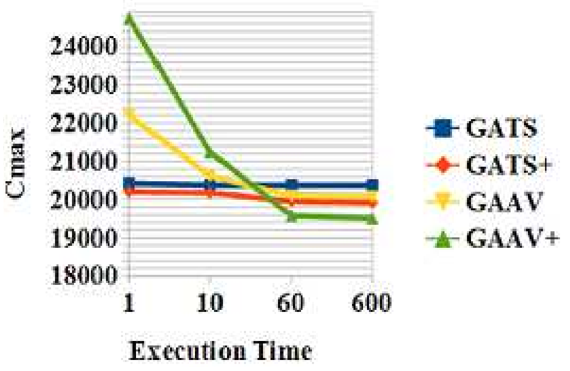


Figure 7. Genetic algorithms convergence.

Whereas GATS remains in the same trend, this behavior followed with most of our problems. Hence, we can say that the techniques used in GAAV and GAAV$^+$ can improve the solution and we can find a better upper bound for this problem. The modification of GATS in GATS$^+$ also has a good outcome.

6.1 Integral Linear Programming Solution (ILP)

The mathematical model is applied for small instances. It is implemented with CPLEX 12.4, on an Intel, core i5 running at 3.4 GHz, and 8 GB of RAM under a computation time limit of one hour (if after one hour no optimal solution is obtained, the current integer solution is returned). In Table 4, columns LB and UB represent the lower bounds and the upper bounds respectively for some problems, for which CPLEX is not able to find the linear relaxation value. Indeed, we have limited the use of the RAM to 6 GB and for the most of instances this amount is not sufficient for the linear relaxation with all these constraints and variables in the model.

Table 4. Results obtained by the Integral Linear Program

Inst.	H-Density		M-Density		L-Density	
m-n	**LB**	**UB**	**LB**	**UB**	**LB**	**UB**
2-20	8892	38480	8008	38480	8190	38480
5-50	8887	40885	5482	40885	5575	40835
10-50	7263	45695	4746	45695	5888	45695

6.2 Transformations between GAs

In spite of the variety between GATS and GAAV encoding's, we tried to investigate the ability of each genetic algorithm to improve the solution obtained by the other genetic algorithm. We transformed the best population generated by the first genetic algorithm, to be the first population for the second genetic algorithm. This also provides interesting observations, about the differences between our proposed genetic algorithms and GATS, by doing all transformations from GATS to GAAV and GAAV$^+$, and from GAAV,GAAV $^+$ to GATS. We noticed that, because of the differences of the encoding and the genetic operators between GATS in comparing to GAAV and GAAV$^+$ the ARPG between the best solution obtained by the first GA before transformation and the best solution obtained by the second genetic algorithm after the transformation.

The ARPG is computed as follows:

$100 \times \frac{C_{max}^{FirstBest} - C_{max}^{SecondBest}}{C_{max}^{SecondBest}}$, where $C_{max}^{FirstBest}$ is the best known C_{max}, obtained by the first GA, and $C_{max}^{SecondBest}$ is the best known C_{max}, obtained by the second GA. Table 7 shows the ARPG of the transformation processes: the negative values mean that the second best solution is worse than the first best. We observed that the

Table 7. The Average Relative Percentage Gap(ARPG) for the transformation process between GAs

Inst.	**H-Density**				**M-Density**				**L-Density**			
m-n	**ATSV**	**ATSV$^+$**	**AVTS**	**AV$^+$TS**	**ATSV**	**ATSV$^+$**	**AVTS**	**AV$^+$TS**	**ATSV**	**ATSV$^+$**	**AVTS**	**AV$^+$TS**
2- 50	1.1	-0.1	0.4	0.2	0.3	-3.1	3.3	0.1	2.0	0.1	0.3	0.3
2-100	2.1	2.5	0.9	0.2	0.8	-0.1	0.3	0.5	-1.9	0.4	1.4	0.0
2-200	1.6	1.2	0.9	0.6	0.7	-1.3	0.1	0.3	0.9	1.0	0.5	0.5
2-500	-0.2	-12.0	-0.5	12.7	0.0	-10.2	0.5	12.0	0.6	-17.3	0.2	17.4
5-50	10.8	5.8	1.7	8.8	16.0	6.6	-1.6	2.3	6.0	-0.2	6.9	4.6
5-100	4.1	3.0	6.6	5.1	11.1	-0.2	7.7	-3.3	14.8	2.7	3.4	3.5
5-200	-3.2	0.9	-0.7	0.1	1.0	-1.7	-3.8	1.2	-8.4	-0.9	-0.6	5.6
5-500	-9.7	-8.0	-0.1	31.0	-10.0	-37.7	-2.0	46.2	-9.7	-42.2	-6.8	52.2
10-50	1.3	-0.9	1.6	-3.4	-7.5	0.0	6.9	4.7	5.6	-1.9	3.9	1.1
10-100	-8.0	-1.1	2.7	2.7	3.8	4.0	3.2	3.3	1.1	6.0	6.9	4.6
10-200	-12.0	-5.8	0.1	1.3	-8.6	-2.9	-0.3	3.2	-15.5	-8.6	-1.3	-4.8
10-500	-27.8	-37.7	-14.4	52.0	-26.0	-42.0	-16.5	74.0	-26.1	-54.5	-15.6	93.1
20-50	4.2	1.3	-0.6	2.0	1.1	0.0	1.7	0.0	-6.0	0.0	1.4	3.6
20-100	0.3	1.6	10.0	0.8	3.3	-2.43	15.6	3.8	-2.0	-5.2	11.9	4.2
20-200	-1.4	-2.4	-2.6	5.0	-11.0	-4.1	-2.0	7.0	-7.4	-1.6	-2.5	10.0
20-500	-23	-39	-17.5	57.5	-29.0	-48.8	-12	83.0	-26.0	-52.7	-10.6	9.0

Table 8. Best Makespan (*Χμαξ*) obtained among All GAs Comparing to AV το TS and AV$^+$ το TS

Inst.	**H-Density**			**M-Density**			**L-Density**		
m-n	**Best**	**AVTS**	**AV$^+$TS**	**Best**	**AVTS**	**AV$^+$TS**	**Best**	**AVTS**	**AV$^+$TS**
2- 50	17927	17927	17937	17251	17251	17251	17251	17251	17251
2-100	37539	37488	37488	37492	37488	37667	37488	37488	37488
2 -200	74474	74425	74438	73210	73112	73358	73024	72891	72891
2 -500	180849	179768	180377	179120	178412	178919	179612	178334	179580
5 -50	9591	9308	9217	7997	7365	7440	7416	7280	7435
5 -100	19320	19318	18579	17640	16929	18597	18020	17973	17830
5 -200	36036	36011	36418	34890	33995	34122	31600	30835	30796
5-500	85015	86133	89414	80365	82007	84885	38147	81178	85589
10 -50	7820	7820	8101	4850	4850	4850	6460	6460	6460
10 -100	10310	10736	10306	9898	9952	9920	9544	9570	9704
10 -200	20656	21045	21218	18148	18951	19301	17030	16251	19321
10 -500	44935	45262	46536	40309	40841	41663	38147	38744	42580
20 -50	6090	6156	6156	6430	6430	6430	4540	4540	4540
20 -100	9310	9428	9262	8141	8249	8189	9320	9054	9372
20 -200	19964	20208	20054	14348	14318	13880	13735	14235	13054
20 -500	36346	38198	37252	30117	32289	30835	27765	28950	28707

behavior of the proposed GAs and GATS is not the same. From the transformations, GAAV and GAAV$^+$ cannot make an improvement to the best generation obtained by GATS. However, for the solutions obtained by the transformations from GAAV to GATS and from GAAV$^+$ to GATS sometimes these transformations can lead to solutions better than those obtained by GAAV, GAAV$^+$ and GATS. Table 8 represents a comparison between the best solutions obtained among all GAs in column labeled ”Best” and the transformations solutions. We can see also, AV$^+$TS can improve the best solution obtained by the first genetic algorithm for instances of large number of jobs, but this improvement did not improve the best solution obtained among all genetic algorithms.

7. Conclusion and Future Work

In this paper we have proposed genetic algorithms for job scheduling problem in cloud computing with the objective of minimizing the makespan, which is considered as an unrelated parallel-machine scheduling problem under precedence constraints. Our contribution has consisted in new genetic algorithms. GAAV includes a new local search procedure for local graph density to evaluate the chromosome. GATS$^+$ which is based on a permutation coding and ECT, and GAAV$^+$ which is combined the innovative characteristics of GAAV with the (ECT) technique. The performances of our proposed genetic algorithms have been compared against one of the best existing genetic algorithm for the same problem. After extensive comparisons, we can conclude that the proposed algorithms can improve the solutions obtained by GATS for small and medium problems. Moreover, they can get better results than GATS within the specific running time (stop criterion) for a high and medium DAG density problem. In the future work, we will enhance the mathematical model by adding new constraints for further improvement. Another interesting topic regarding scheduling problem in cloud computing is to consider the multiobjective optimization context.

REFERENCES

1. GRAHAM, R. L., E. L. LAWLER, J. K. LENSTRA, A. R. KAN, **Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey**, Annals of Discrete Mathematics, vol. 5, 1979, pp. 287-326.

2. RAHMANI, A. M., M. REZVANI, **A Novel Genetic Algorithm for Static Task Scheduling in Distributed Systems**, Intl J. of Computer Theory and Engineering, vol. 1, no. 1, 2009, pp. 1793-8201.

3. SHENAI, S., **Survey on Scheduling Issues in Cloud Computing**, Procedia Engineering, vol. 38, 2012, pp. 2881-2888.

4. HALL, N. G., E. MARC, **Generating Experimental Data for Computational Testing with Machine Scheduling Applications**, Operations Research, vol. 49, no. 7, 2001, pp. 854-865.

5. CHENG, W., L. MIN, **A Genetic Algorithm for Minimizing the Makespan in the Case of Scheduling Identical Parallel Machines**, Artificial Intelligence in Eng., vol. 13, no. 4, 1999, pp. 399-403.

6. BALA, A., I. CHANA, **A Survey of Various Task Scheduling Algorithms in Cloud Environment**, IJCA, 2011, pp. 26-30.

7. HUANG, Q., X. HUANG, J. LI, K. SHUANG, S. SU, J. WANG, **Cost-efficient Task Scheduling for Executing Large Programs in the Cloud**, Parallel Computing, vol. 39(4), 2013, pp. 177-188.

8. GUO, L., C. JIANG, S. ZHAO, S. SHEN, **Task Scheduling Optimization in Cloud Computing based on Heuristic Algorithm,** Journal of Networks, vol. 7, no. 3, 2012, pp. 547-553.

9. TIAN, S., Y. XU, H. ZHAO, G. ZHOU, **A Genetic-based Task scheduling Algorithms on Heterogeneous Computing Systems to Minimize Makespan**, Journal of Convergence Information Technology(JCIT), vol. 8, no. 5, 2013, pp. 547-555.

10. BILGAIYAN, S., M. DAS, S. SAGNIKA, **An Analysis of Task Scheduling in Cloud Computing using Evolutionary and Swarm-based Algorithms**, International Journal of Computer Applications, vol. 89, no. 2, 2014, pp. 11-18.

11. LIU, C., YANG, S., **A Heuristic Serial Schedule Algorithm for Unrelated Parallel Machine Scheduling with Precedence Constraints**, Journal of Software, vol. 6(6), 2011, pp. 1146-1153.

12. HE, J., Y. KANG, H. LU, **A PSO-based Genetic Algorithm for Scheduling of Tasks in a Heterogeneous Distributed System**, Journal of software, vol. 8, no. 6, 2013, pp. 1443-1450.

13. BALIN, S., **Non-Identical Parallel Machine Scheduling using Genetic Algorithm**, Expert Sys. with Applications, vol. 38(6), 2011, pp. 6814-6821.

14. BAZZAZI, M., M. IZADI, F. SASSANI, F. TAHERI, R. TAVAKKOLI-MOGHADDAM, **Design of a Genetic Algorithm for Bi-objective Unrelated Parallel Machines Scheduling with Sequence-dependent Setup Times and Precedence Constraints**, Computers & Operations Research, vol. 36, no. 12, 2009, pp. 3224-3230.

15. RUIZ, R., E. VALLADA, **A Genetic Algorithm for the Unrelated Parallel Machine Scheduling Problem with Sequence Dependent Setup Times**, European Journal of Operational Research, vol. 211(3), 2011, pp. 612-622.

16. BENZIANI, Y., I. KACEM, P. LAROCHE, A. NAGIH, **Exact and Heuristic Methods for Minimizing the Total Completion Time in Job-shops**, Studies in Informatics and Control, ISSN 1220-1766, vol. 23(1), 2014, pp. 31-40.

17. ARYAN, Y., A. G. DELAVAR, **HSGA: A Hybrid Heuristic Algorithm for Workflow Scheduling in Cloud Systems**. Cluster Computing, vol. 17, no. 1, 2014, pp. 129-137.

18. AHMAD, I., Y. K. KWOK, **Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors**, ACM Computing Surveys, vol. 31, no. 4, 1999, pp. 406-471.

19. DEEPA, S. N., S. N. SIVANANDAM, **Introduction to Genetic Algorithm**, Springer Berlin Heidelberg, 2008.

20. HASSAN, M.-A., I. KACEM, S. MARTIN, **Unrelated Parallel Machines with Precedence Constraints: Application to Cloud Computing**, Cloudnet 2014, pp. 438-442.