A GENETIC ALGORITHM FOR INDEPENDENT JOB SCHEDULING IN GRID COMPUTING

Muhanad Tahrir Younis and Shengxiang Yang

De Montfort University
School of Computer Science and Informatics
Centre for Computational Intelligence (CCI)
The Gateway, Leicester LE1 9BH
United Kingdom
P14017957@my365.dmu.ac.uk
syang@dmu.ac.uk

Abstract: *Grid computing refers to the infrastructure which connects geographically distributed computers owned by various organizations allowing their resources, such as computational power and storage capabilities, to be shared, selected, and aggregated. Job scheduling is the problem of mapping a set of jobs to a set of resources. It is considered one of the main steps to efficiently utilise the maximum capabilities of grid computing systems. The problem under question has been highlighted as an NP-complete problem and hence meta-heuristic methods represent good candidates to address it. In this paper, a genetic algorithm with a new mutation procedure to solve the problem of independent job scheduling in grid computing is presented. A known static benchmark for the problem is used to evaluate the proposed method in terms of minimizing the makespan by carrying out a number of experiments. The obtained results show that the proposed algorithm performs better than some known algorithms taken from the literature.*

Keywords: *evolutionary algorithms; genetic algorithm; job scheduling; grid computing; makespan.*

1 Introduction

Grid Computing has been defined as a type of parallel and distributed infrastructure which allows the geographically distributed autonomous and heterogeneous resources to be shared, selected and aggregated dynamically depending on their availability, capability, performance, cost, and users' quality-of-service requirements. Grid systems exploit the information technology resources by connecting loosely coupled computers and various networks together to offer the same processing capabilities provided by supercomputers to their users. This connection is achieved through computer software, called Middleware, which provides the necessary services for resource management, monitoring, security, and so forth. These computational resources are owned by different administrative organizations. Therefore, local policies are defined to specify what is shared, who is allowed to access what and when, and under what conditions. The grid architecture uses the concept of Virtual Organizations (VOs) to control resource sharing [4]. A physical organization can be part of one or more VOs by allowing all or some of its resources to be shared [6]. Nowadays, computational grid is a common tool for developing a large scale of commercial and non-commercial applications [5].

Job scheduling, which is the problem of mapping a set of jobs to a set of resources, is considered one of the main issues in grid computing systems [8]. This mapping is known to be computationally hard. The quality of resource allocation can be computed using an objective function such as minimizing the makespan, maximizing load balancing, and maximizing resources utilization [14]. The scheduler's efficiency strongly relies on the approach applied to find the mapping. Different algorithms could be used to find such mapping, which vary from simple heuristic methods to meta-heuristic methods. However, to enhance the overall performance of the grid, meta-heuristic approaches are more likely preferred [12]. One of these methods is the Genetic Algorithm (GA), a population-based meta-heuristic search method inspired by the evolution of living beings. A GA starts with a group of solutions, called the initial population, which is usually generated randomly, and then seeks to evolve the population by applying selection, crossover and mutation operators in order to find the optimal or near-optimal solutions. GAs have been widely used for solving many combinatorial optimization problems. Eleven static heuristic methods have been presented in [2] to solve the problem of job scheduling in heterogeneous environment. The results show that the GA studied outperforms the other ten methods used in the study in terms of minimizing the makespan. A population of 200 individuals, which are generated either randomly or by seeding the population with one individual generated using the Min-Min heuristic method [7] and 199 individuals generated randomly, was used.

A study proposed by [3] studied the use of GAs for efficient multi-objective job scheduling on computational grid. Two heuristics methods, which are the Longest Job to Fastest Resource, Shortest Job to Fastest Resource (LJFR- SJFR) [1] and Minimum Completion Time (MCT) [11], have been used beside the random method to initialize the initial population. Two encoding schemes, namely the direct and permutation methods, have been considered as well as several GA operators have been implemented.

A fuzzy particle swarm optimisation (PSO) based scheduler for job scheduling on computational grid was proposed by [10]. Minimization of the makespan time was the main goal of the proposed scheduler. The authors used small to large scale resource-job pairs problems to test the performance. Their method has been compared with a GA and a simulating annealing (SA) approach. The results showed that the fuzzy PSO scheduler has the ability to find faster and feasible solutions over the GA and SA.

The authors in [16] developed a differential evolution (DE) algorithm to generate schedules which efficiently utilises the resources and completes the jobs with a minimum period of time. The results have been compared with findings in [10] and it has been found that PSO outperforms DE in three instances. However, the authors in [16] have claimed that the solutions found by DE have better resource utilisation.

A Variable Neighbourhood Search algorithm, called TPVNS, was introduced in [15] to solve the problem of job scheduling on heterogeneous computing and grid environment. The proposed algorithm consists of two modules: exploration and diversification. The former module is achieved by using GVNS while the latter is achieved by applying another two modules, namely BVNS and Crossover heuristic. The proposed method has been compared against several methods from the literature and the results showed that it outperforms them in many cases.

In this work, a GA for the independent job scheduling problem in grid computing is introduced. The proposed algorithm uses a new mutation procedure to improve the quality of solutions in terms of minimising the makespan. The performance of the proposed GA is compared with other methods, including a GA, SA, Particle Swarm Optimisation (PSO), Differential Evolution (DE), and Variable Neighbourhood Search (VNS).

The rest of this paper is structured as follows. Section 2 presents the scheduling problem formulation. Section 3 explains the use of GA for job scheduling in grid computing while Section 4 presents the results of applying the proposed GA in grid computing. Finally, the conclusions and future work are provided in Section 5.

2 Scheduling Problem Formulation

In this study, a simulation model is considered rather than real grid computing systems, which allows us to capture the important characteristics of job scheduling on computational grid. One such model is the Expected Time to Compute (ETC) model [12]. The expected execution time of the jobs on each machine is assumed to be available in advance in a two dimensional array. This assumption is realistic since it is easy to gather information about the jobs requirements and the computation power of resources from the users, by predications or from historic data [18]. Table 1 shows a 10 x 4 subset of the ETC matrix.

The problem description under the ETC model is defined as follows:

1. A set of n jobs that have to be scheduled. These jobs are independent to each other (i.e., any job can be processed by any resource) and are non-preemptive, which means that a job must be processed entirely by a single resource.

2. A set of m resources to process the submitted set of independent jobs. These resources are heterogeneous.

3. The ETC matrix of size $n \times m$, where ETC[i][j] represents the estimated time for executing job i on resource j.

The job scheduling problem in grid computing is usually known to be multi-objective since several objective functions can be considered, such as the makespan, load balancing, and flow time [17]. In this study, the minimization of makespan will be considered, which is defined as the finishing time of the latest job and can be calculated by Equation (1).

$$makespan = min_{s \in S} max_{j \in J}(Finish_j), \tag{1}$$

where S is the set of all possible solutions, J is the set of all jobs submitted to the system and $Finish_j$ represents the time when job j is finished [9].

3 Applying GAs to the Job Scheduling Problem

GAs are a population-based heuristic search method, which is inspired by natural evolution. A GA starts with an initial population, a group of solutions usually generated randomly, then seeks to find the approximately best solution by applying selection, crossover and mutation operators. GA has been quite used for solving many

Table 1: A 10 x 4 subset of the Expected Time to Compute (ETC) matrix, where r_i $(1 \leq i \leq 4)$ is a resource

job	r_1	r_2	r_3	r_4
j_1	4.3	135.9	194.5	223.8
j_2	353.5	472.9	478.4	117.5
j_3	17.2	18.9	24.5	33.2
j_4	182.2	358.3	180.1	539.6
j_5	55.3	99.7	107.0	198.0
j_6	405.8	88.7	59.9	82.9
j_7	55.8	79.0	84.7	110.2
j_8	166.6	334.4	310.1	194.7
j_9	108.2	119.3	138.7	144.7
j_{10}	109.8	127.7	30.2	125.0

Algorithm 1 The Genetic Algorithm

1: $t \leftarrow 0$
2: Generate the initial generation Gen(t) of k individuals
3: Evaluate the fitness of each individual in the initial generation, i.e., compute Fitness(Gen(t))
4: **while** (the end criterion is not true) **do**
5: $t \leftarrow t + 1$
6: Select Parent(t) from Gen(t-1)
7: With probability p_c, recombine individuals in Parent(t) to produce Offspr1(t)
8: With probability p_m, mutate individuals in Offspring1(t) to produce Offspr2(t)
9: Evaluate the fitness of each individual, i.e. , compute Fitness(Offspr2(t))
10: Replace Gen(t) from Offspr2(t) and/or Gen(t-1)
11: **end while**
12: return the best solution found

optimization problems closely related to the job scheduling problem. The pseudo-code for the GA is illustrated in Algorithm 1.

3.1 The Solution Representation

A key issue in GAs is the representation of individuals. Two types of encodings have been reported in the literature, namely, the direct representation and the permutation-based representation [3]. In this study, the direct representation is used to encode the individuals. In the direct representation, each individual is represented as a list with size equals to the number of jobs. The value of allele i represents the resource where job i is allocated. Therefore, the values in this list are integers in the range$[0, r-1]$, where r is the total number of resources.

3.2 The Initial Generation

One way to generate the initial population of GAs is the random method. However, several studies showed that seeding the initial population of a GA with solutions from other heuristic methods may introduce more diversity and hence produce better solutions [19]. In this study, the initial population is generated by seeding the population with one individual generated using the Min-Min heuristic method [7] and the rest individuals generated randomly. The Min-Min heuristic [7] used in generating the initial population starts by computing the minimum completing time $CT[i, j]$ for all jobs and resources. Then, it finds the job x with the minimum $CT[i, j]$ and allocates it to the resource that obtains it. After assigning job x, the $CT[i, j]$ matrix is updated. The same steps are repeated until all jobs are assigned. The pseudo-code for the Min-Min heuristic is illustrated in Algorithm 2.

3.3 The Fitness Evaluation

The makespan is used to evaluate the fitness of individuals, see Section 2 (Scheduling problem formulation).

3.4 The Selection Operator

The selection operator refers to the process that determines which individuals are to be continued and allowed to reproduce and which ones deserve to be eliminated. Several selection techniques are available in the literature.

Algorithm 2 The Min-Min Algorithm

```
For every job in the job set, calculate the completion time (CT)
jobs_removed ← 0
while (jobs_removed<total number of jobs) do
    Find the job i in the job set with the earliest completion time and the resource j which obtains it
    Assign i to j
    Delete i from the job set
    jobs_removed ← jobs_removed + 1
    Update the ready time and the completion time (CT) of resource j
end while
```

In the study, the N-tournament method is used with $N = 3$. In a tournament selection, several tournaments are run among a few individuals which have been selected randomly from the population. The winner of each tournament (the one with the best fitness) is selected for next stages.

3.5 The Crossover Operator

The crossover operator is equivalent to reproduction and biological crossover. New solutions (offspring) are generated by selecting individuals from the parental generation and exchanging their genes. Crossover enables the search process to explore new regions of the solution space, which have been not explored yet and provide the next generation with good quality individuals. Several types of crossover operators exist in the evolutionary computation literature, which mainly depends on the solution representation. Therefore, in our case, a crossover operator for the direct representation, which is the one-point crossover, will be considered. Given two parent solutions, the one-point crossover operator starts by generating a random position between 1 and the total number of jobs. This position serves as an exchange point which divides each parent into two parts. Two new offspring are obtained by exchanging the two first segments of the parents. The one-point scheme is used with a probability of 0.8, which is the same probability reported in [10].

3.6 The Mutation Operator

The mutation operator is one of the most important elements of GAs, which is related to the exploration of the search space. By mutation, individuals are randomly altered to maintain and introduce diversity in the subsequent generations [13].

Any solution S will have at least one resource with a local makespan time equals to the overall makespan of the solution, this resource is called the 'problem resource'. New solutions can be obtained from S by swapping a job currently allocated to the problem resource with a job allocated to other resource, or by transferring of a job currently allocated to the problem resource to any other resource. Therefore we can define two new mutation operators based on the concepts of swap and transfer, namely, the best swap mutation and the best transfer mutation. The best swap mutation alters the solution by finding the best resource swap between one of jobs assigned to the problem resource and all other jobs which best minimises the makespan. The pseudo-code for the best swap mutation is illustrated in Algorithm 3. On the other hand, the best transfer mutation alters the solution by transferring the job assigned to the problem resource which has the maximum expected completion time to the resource which has the minimum processing time for it. Algorithm 4 demonstrates the pseudo-code for the best transfer mutation. The best swap mutation is done with a probability of 0.5; otherwise, the best transfer mutation is applied.

3.7 The Replacement Operator

The replacement operator is the process of deciding which individuals in the population should be eliminated to make room for the new offspring. In this paper, the Steady State strategy is used, that is, parents and offspring compete for survival then the best of them are selected.

3.8 The Stopping Condition

To make fair comparison, the proposed GA uses the same number of iterations as used in [10], which is (50 x the number of jobs x the number of resources) iterations.

Algorithm 3 The best swap move mutation

```
Find the problem resource (pr) which has the maximum local makespan time
Find the list of jobs assigned to pr (pr_list)
for All pj ∈ pr_list do
    for All oj ∈ all jobs and oj ≠ pj do
        new_solution ← swap the resource assigned to pj with the resource assigned to oj
        Calculate the fitness of new_solution
        Add new_solution to the list of all solutions
    end for
end for
Find the swap move in the list of all solutions which best minimises the overall makespan
```

Algorithm 4 The best transfer move mutation

```
Find the problem resource (pr) which has the maximum local makespan time
Find the list of jobs assigned to pr (pr_list)
Find the job j in pr_list which has the maximum expected completion time
Assign j to resource r which has the minimum processing time for j
```

Table 2: Performance comparison between the proposed GA and some algorithms from the literature

Algorithm	Criteria	Instance			
		(3, 13)	(5, 100)	(8, 60)	(10, 50)
GA [10]	Avg	47.1167	85.7431	42.9270	38.0428
	time	302.9210	2415.9000	2263.0000	2628.1000
SA[10]	Avg	46.6000	90.7338	55.4594	41.7889
	time	332.5000	6567.8000	6094.9000	6926.4000
PSO [10]	Avg	46.2667	**84.0544**	41.9489	37.6668
	time	106.2030	1485.6000	1521.0000	1585.7000
DE [16]	Avg	46.0500	86.3600	42.4800	38.3900
	time	22.4400	1550.3227	430.0000	285.2600
TPVNS [15]	Best	**46.0000**	85.4345	41.7227	35.1586
	Avg	46.2500	85.4357	41.7412	35.2478
proposed GA	Best	**46.0000**	85.5281	**41.5808**	**35.1438**
	Avg	**46.0000**	85.5333	**41.5941**	**35.1613**
	time	4.3787	195.1741	76.2255	70.4771

4 Experimental Results

In order to simulate several heterogeneous scheduling scenarios in a realistic way and to allow a fair comparison of the presented methods, the same dataset used in [10] has been considered. The dataset consists of four instances of different sizes. The notation (the number of resources, the number of jobs) has been used to describe each instance. The resource job pairs vary from small scale instance (3, 13) to large scale instances, such as (5, 100), (8, 60) and (10, 50). Experiments have been carried out using an Intel i5-4570 CPU @ 3.20 GHz with 8 GB RAM and all programs were written in Java language. To obtain the best and average values, each algorithm was executed 10 times for each instance. Table 2 provides the performance comparison between the proposed GA and other methods from the literature in terms of makespan. Fig. 1 illustrates the average makespan of GA, SA, PSO, DE, TPVNS, and our GA, while Fig. 2 shows the performance of the proposed GA. In Table 2, the first column represents the algorithm applied, the second column represents the criteria used in comparison, namely Avg (average), time (in seconds), and Best (best makespan found). There is no information provided about the best makespan achieved by the algorithms proposed in [10] and [16] nor the time the algorithm proposed in [15] needed to finish. the third, fourth, fifth, and sixth columns represent the four different instances. The best results are indicated in bold.

The results in Table 2 show clearly that the proposed GA outperforms the other approaches in three instances, namely (3, 13), (8, 60) and (10, 50), while PSO outperforms the other methods in the (5, 100) instance. Table 2 also shows the time needed to finish the search process. It is clear that the proposed GA is faster than all other methods. Moreover, the performance of the proposed GA shows that it finds better solutions faster than other approaches. For the (3, 13) instance, the results in [10] indicates that the optimal solution for this instance is the one with a makespan of 46. For ten trials, GA provided the optimal solution twice, SA and

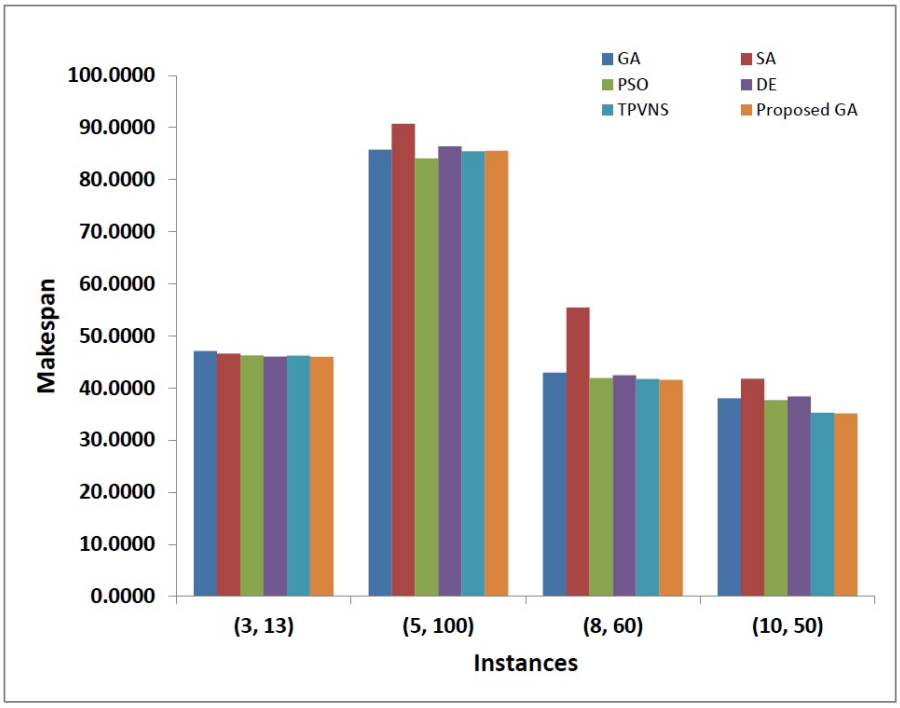


Figure 1: The average makespan of GA, SA, PSO, DE, TPVNS, and our GA.

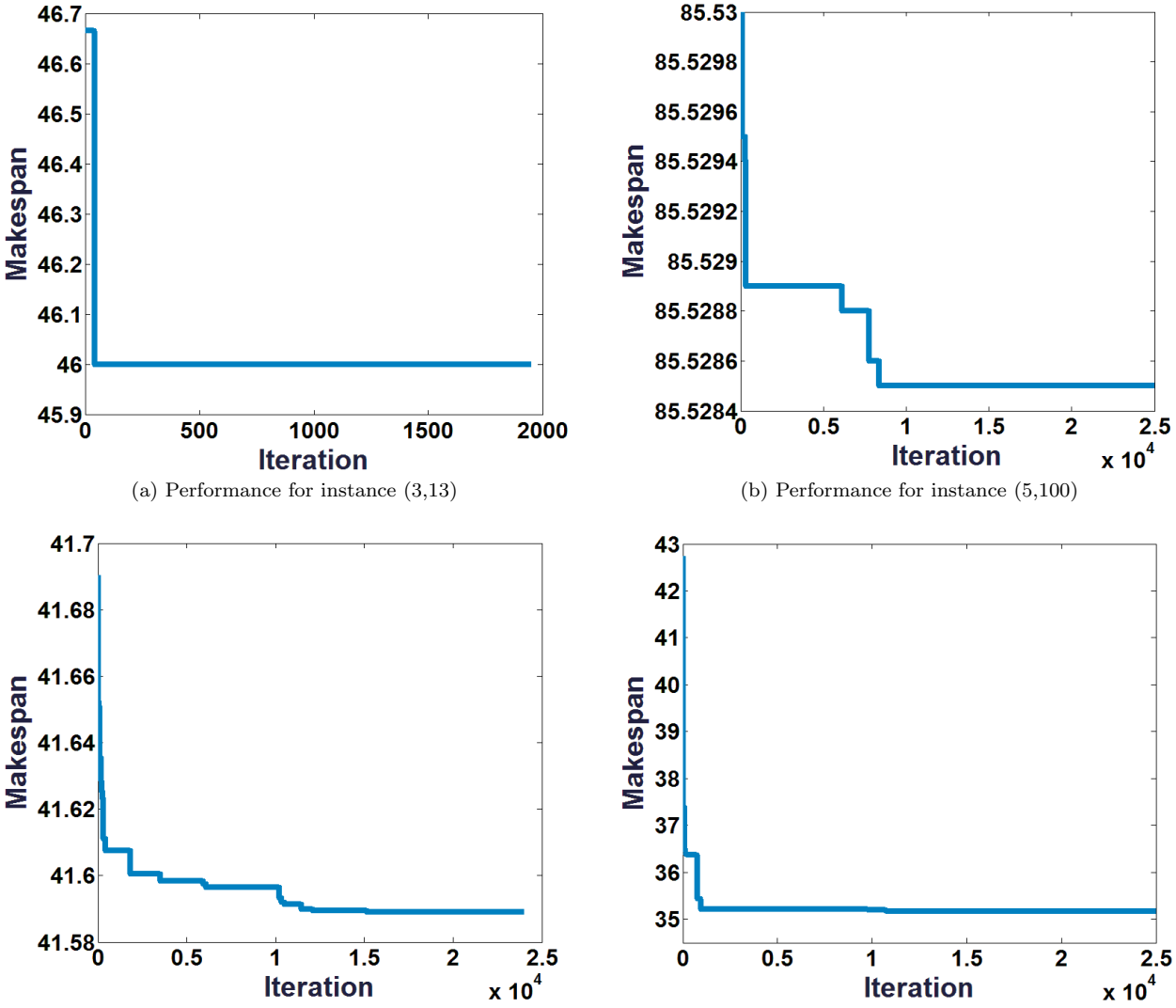
PSO provided the optimal solution 3 and 5 times, respectively. Our proposed GA provided the optimal solution 10 times. Moreover, PSO was able to find the best solution after approximately 1,800 iterations, the DE in [16] was able to find the best solution after approximately 170 iterations while our proposed GA was able to find the optimal solution after 40 iterations only as indicated in Fig. 2(a). For the (5, 100) instance, PSO and DE found the best schedule after approximately 23,000 and 400 iterations respectively, while our GA found the best solution after 8370 iterations as indicated in Fig. 2(b). For the (8, 60) instance, after approximately 22,500 iterations, PSO was able to find a solution with the makespan of 41.9489, DE found a schedule with the makespan of 42.4800 after 400 iterations, while our GA found the a solution with the makespan of 41.6111 after 400 iterations as indicated in Fig. 2(c). Finally, for the (10, 50) instance, PSO, DE, and our GA found solutions with tha makespan of 37.6668, 83.3900, and 36.3751, respectively after 22,500, 400, 400 iterations respectively. Fig. 2(d) indicates the performance for instance (5, 50).

5 Conclusions and Future Work

One of the significant tasks in grid computing systems is the mapping of jobs to resources. Designing efficient job scheduler means improving the overall performance of grid computing systems. Although the job scheduling problems in conventional distributed systems is known to be NP-complete, it is much more complex in grid computing systems as the jobs and resources in these environments have a high degree of heterogeneity, the environment is dynamic, and the problem is multi-objective. Therefore, it is necessary to use meta-heuristics, such as GAs in order to cope in practice with its complexity and difficulty. GAs as a robust search method, have been used successfully to solve the problem of job scheduling in computational grid. However, the solution found by GA could be improved by using good genetic operators. In this paper, a new mutation procedure, which uses the concept of swap and transfer to alter individuals, has been introduced. The results show that the proposed GA can find better mappings than other approaches found in the literature in terms of minimizing the makespan as it outperforms them in most of the cases. In addition, it has been found that our GA has the minimum runtime and it requires less number of iterations among all methods investigated in this study.

The proposed GA seems a promising approach to scheduling in grid computing systems. However, there is much space for further improvements. These improvements include checking other solution representations, defining other crossover operators, using other selection methods, applying other replacement strategies, adding another objective (such as the flow time) so that the problem will be multi-objective, and finally testing it in a dynamic environment.

Acknowledgement: This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) of U.K. under Grant EP/K001310/1.



(a) Performance for instance (3,13)

(b) Performance for instance (5,100)

(c) Performance for instance (8,60)

(d) Performance for instance (10,50)

Figure 2: The performance of the proposed genetic algorithm.

References

[1] Abraham, A., Buyya, R., Nath, B.: Natures heuristics for scheduling jobs on computational grids. In: The 8th IEEE international conference on advanced computing and communications (ADCOM 2000), pp. 45–52 (2000)

[2] Braun, T.D., Siegel, H.J., Beck, N., Bölöni, L.L., Maheswaran, M., Reuther, A.I., Robertson, J.P., Theys, M.D., Yao, B., Hensgen, D., et al.: A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. Journal of Parallel and Distributed computing **61**(6), 810–837 (2001)

[3] Carretero, J., Xhafa, F., Abraham, A.: Genetic algorithm based schedulers for grid computing systems. International Journal of Innovative Computing, Information and Control **3**(6), 1–19 (2007)

[4] Foster, I., Kesselman, C.: The Grid 2: Blueprint for a new computing infrastructure. Elsevier (2003)

[5] Foster, I., Kesselman, C.: The history of the grid. computing **20**(21), 22 (2010)

[6] Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: Enabling scalable virtual organizations. International journal of high performance computing applications **15**(3), 200–222 (2001)

[7] Ibarra, O.H., Kim, C.E.: Heuristic algorithms for scheduling independent tasks on nonidentical processors. Journal of the ACM (JACM) **24**(2), 280–289 (1977)

[8] Izakian, H., Abraham, A., Snasel, V.: Performance comparison of six efficient pure heuristics for scheduling meta-tasks on heterogeneous distributed environments. Neural Network World **19**(6), 695 (2009)

[9] Kołodziej, J., Xhafa, F.: Enhancing the genetic-based scheduling in computational grids by a structured hierarchical population. Future Generation Computer Systems **27**(8), 1035–1046 (2011)

[10] Liu, H., Abraham, A., Hassanien, A.E.: Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm. Future Generation Computer Systems **26**(8), 1336–1343 (2010)
[11] Maheswaran, M., Ali, S., Siegel, H.J., Hensgen, D., Freund, R.F.: Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. Journal of parallel and distributed computing **59**(2), 107–131 (1999)
[12] Nesmachnow, S., Alba, E., Cancela, H.: Scheduling in heterogeneous computing and grid environments using a parallel chc evolutionary algorithm. Computational Intelligence **28**(2), 131–155 (2012)
[13] Osaba, E., Carballedo, R., Diaz, F., Onieva, E., de la Iglesia, I., Perallos, A.: Crossover versus mutation: a comparative analysis of the evolutionary strategy of genetic algorithms applied to combinatorial optimization problems. The Scientific World Journal **2014** (2014)
[14] Pacini, E., Mateos, C., Garino, C.G.: Distributed job scheduling based on swarm intelligence: A survey. Computers & Electrical Engineering **40**(1), 252–269 (2014)
[15] Selvi, S., Manimegalai, D.: Task scheduling using two-phase variable neighborhood search algorithm on heterogeneous computing and grid environments. Arabian Journal for Science & Engineering (Springer Science & Business Media BV) **40**(3) (2015)
[16] Selvi, S., Manimegalai, D., Suruliandi, A.: Efficient job scheduling on computational gridwith differential evolution algorithm. International Journal of Computer Theory and Engineering **3**(2), 277 (2011)
[17] Xhafa, F., Abraham, A.: Computational models and heuristic methods for grid scheduling problems. Future generation computer systems **26**(4), 608–621 (2010)
[18] Xhafa, F., Kolodziej, J., Barolli, L., Fundo, A.: A ga+ ts hybrid algorithm for independent batch scheduling in computational grids. In: Network-Based Information Systems (NBiS), 2011 14th International Conference on, pp. 229–235. IEEE (2011)
[19] Younis, M.T., Yang, S., Passow, B.: Meta-heuristically seeded genetic algorithm for independent job scheduling in grid computing. In: European Conference on the Applications of Evolutionary Computation, pp. 177–189. Springer (2017)