Modified genetic algorithm for multiobjective task scheduling on heterogeneous computing system

O.L. Sathappan and P. Chitra*

Department of Computer Science and Engineering,
Thiagarajar College of Engineering,
Madurai – 625015, Tamilnadu, India
E-mail: ol.sathappan@gmail.com
E-mail: pccse@tce.edu
*Corresponding author

P. Venkatesh

Department of Electrical and Electronics Engineering,
Thiagarajar College of Engineering,
Madurai – 625015, Tamilnadu, India
E-mail: pvee@tce.edu

M. Prabhu

Department of Computer Science and Engineering,
Thiagarajar College of Engineering,
Madurai – 625015, Tamilnadu, India
E-mail: Mprabhucse@tce.edu

Abstract: This paper addresses the problem of task scheduling in heterogeneous distributed systems with the goal of maximising the system reliability and decreasing the makespan. The task scheduling problem in heterogeneous systems an NP-complete problem. A modified genetic algorithm which combines the characteristics of bacteriological algorithm (BA) and genetic algorithm is proposed. This modified algorithm (MA) is used in the weighted sum approach of multiobjective genetic algorithm (MOGA). The proposed algorithm is applied for random and real-time numerical application graphs and compared with the biobjective genetic algorithm (BGA) in the literature. The simulation results confirm that the proposed algorithm produces near optimal solutions at reduced computational times.

Keywords: task scheduling; makespan; reliability; bacteriological algorithm; BA; multiobjective genetic algorithm; MOGA; biobjective genetic algorithm; BGA.

Reference to this paper should be made as follows: Sathappan, O.L., Chitra, P., Venkatesh, P. and Prabhu, M. (2011) 'Modified genetic algorithm for multiobjective task scheduling on heterogeneous computing system', *Int. J. Information Technology, Communications and Convergence*, Vol. 1, No. 2, pp.146–158.

Biographical notes: O.L.Sathappan is finishing his final year for the degree in BE Computer Science at the Thiagarajar College of Engineering, Madurai, Tamilnadu, India.

P. Chitra obtained his ME in Computer Science and Engineering from Thigagarajar College of Engineering in 2000. She is pursuing her research in the area of high performance computing. Her current areas of interest are evolutionary algorithms, distributed computing and multicore architectures. She has published more than two textbooks on computer languages and basic communications. She has also published research papers in international journals and conferences.

P. Venkatesh obtained his Degree in Electrical and Electronics Engineering, his Masters in Power System Engineering and his PhD in Application of Evolutionary Computation Methods to Power System Optimizations Problems from the Madurai Kamaraj University, India in 1991, 1994 and 2003 respectively. He has received the Boyscast Fellowship Award from the Department of Science and Technology, India for carrying out Post Doctoral Research Work at the Pennsylvania State University, USA. His areas of interest are application of evolutionary computation techniques to power system problems and power system restructuring.

M. Prabhu is finishing his final year for the degree in BE Computer Science at the Thiagarajar College of Engineering, Madurai, Tamilnadu, India.

1 Introduction

Heterogeneous distributed computing systems, such as peer-to-peersystems and grids, have been widely deployed for executing computationally intensive applications. These systems usually comprise large number of geographically distributed resources which are more susceptible to failure. As these systems become larger and more widely dispersed, the reliability of an application running on these systems decreases due to the system's inherent unreliability. Hence, the scheduling of an application in such environments must take into account the reliability of the application besides the execution time (makespan) which is usually considered as the main objective of scheduling.

Given an application modelled by a dependence graph, the scheduling problem deals with mapping each task of the application onto the available heterogeneous systems in order to minimise makespan. The task scheduling problem has been solved for years and is known to be NP-complete (Gary and Johnson, 1979). Several heuristic algorithms are proposed in literature to solve this problem. These heuristics are classified into different categories such as list scheduling algorithms, clustering algorithms, duplication-based algorithms, guided random search algorithms (Topcuoglu et al., 2002). These algorithms are applicable for homogeneous systems. List scheduling algorithms (Kwok and Ahmad, 1996; El-Rewini and Lewis, 1990) assign priority to each task. Based on their priorities, tasks are allocated to processors to minimise a predefined cost function. The basic idea of clustering algorithm (Topcuoglu et al., 2002; Yang and Gerasoulis, 1994; Liou and Palis, 1996) is to group heavily communicated tasks into the same cluster. Tasks grouped into the same cluster are assigned to the same processor in an effort to avoid communication costs. Duplication-based scheduling algorithms (Chang and Ranka, 1992; Ahmad and

Kwok, 1994) use the idle time slots on certain processors to execute duplicated predecessor tasks that are also being run on some other processors, such that communication delay and network overhead can be minimised. Genetic algorithms (Hou et al., 1994; Shroff et al., 1996; Wang et al., 1997; Braun et al., 2001; Topcuoglu et al., 2002) are the most widely studied guided random search techniques for task scheduling problems. The task scheduling problems have also been studied for heterogeneous computing systems in El-Rewini and Lewis (1990), Maheswaran and Siegel (1993), Singh and Youssef (1996), Wang et al. (1997) and Braun et al. (2001). Evolutionary programming (EP) is a mutation-based evolutionary algorithm applied for discrete search spaces and used in task scheduling (Fogel and Fogel, 1996)

Multi-objective evolutionary algorithm (MOEA) combines two major disciplines: evolutionary computation and the theoretical frameworks of multi-criteria decision making. Defining the multiple objectives for the task scheduling problem for generating efficient schedules at reduced computational times are of research interest in the recent days. The objectives such as makespan, average flow time, robustness and reliability of the schedule are considered for solving task scheduling problem. Multiobjective GA has been used for task scheduling in Dogan and Ozguner (2005), Ye et al. (2006) and Carretero et al. (2007). Dogan and Ozguner (2005) proposed a bi-objective genetic algorithm (BGA) and bi-criteria extension of the DLS algorithm by estimating a cost function that takes into account the decrease in reliability from a particular decision and generates schedules to optimise makespan and reliability. In Ye et al. (2006), the dependent relationships of tasks, and multi-dimensional QoS metrics of completion time and execution cost of tasks, are considered as multiple objectives. In Carretero et al. (2007), independent tasks are considered for minimising the multiple objectives of minimising the makespan and average flow-time.

This paper considers the two objectives for multiobjective task scheduling: minimising the makespan, and maximising the reliability of the schedules. A modified algorithm (MA) is proposed for handling two objectives, using the weighted sum approach. We have tested the performance of the algorithm using parallel Gaussian elimination task graph (Topcuoglu et al., 2002) and compared the results obtained by MA and BGA.

2 Task scheduling problem

The scheduling system model consists of an application, heterogeneous computing system and two of the objectives namely, makespan and reliability. The application can be represented by a directed acyclic graph (DAG) *G (V, E, C, W)*, where:

- V, the set of v nodes. Each node v_i in V represents a subtask of the application task.
- E is the set of communication edges. The directed edge $e_{i,j}$ joins nodes v_i and v_j, where node v_i is called the parent node and node v_j is called the child node. This implies the precedence constraint that v_j cannot start until v_i finishes and sends its data to v_j.
- C is the set of communication costs. Edge $e_{i,j}$ has a communication cost $c_{i,j}$ in C.
- W, a $v \times p$ computation costs matrix in which each $w_{i,j}$ gives the estimated time to execute task v_i on machine p_j.

A task without any parent is called an entry task and a task without any child is called an exit task. In the DAG model, each node label gives the execution time for the corresponding task and each edge label gives communication time required to pass data from one node to the next if the two nodes are assigned to different processors during program execution. A task cannot start until all of its predecessor tasks are complete.

The heterogeneous computing system is a set P of p heterogeneous machines (processors) connected in a fully connected topology. Figure 1(a) represents an example of an application modelled by a DAG $G = (V, E)$, where $V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ represents the set of six subtasks to be executed, and the set of weighted, directed edges E represents the communication requirement between subtasks. In this example, E consists of $\{e_{12}, e_{13}, e_{24}, e_{34}, e_{35}, e_{46}, e_{56},\}$ with the values set to {6, 5, 7, 7, 8, 3, 9} respectively. Figure 1(b) represents the computation cost matrix (W), of the heterogeneous computing system (composed of three processors p_0, p_1 and p_2).

A schedule is a function $S : V \to P$ maps a task to the processor that executes it. Let $V(j,s) = \{ i | s(i) = j \}$ be the set of tasks mapped to processor j.

The completion time of a processor j is given by

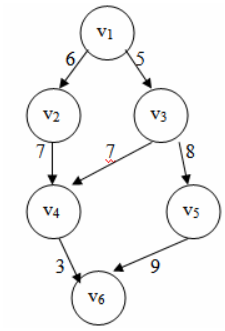
$$c_j(s) = \sum_{i \in v(j,s)} W_{ij} \tag{1}$$

The communication cost, e_{ij} is added if the dependent tasks are allocated to different processors. The makespan of a schedule is the time where all tasks are completed.

$$C_{\max}(S) = \max_j c_j(s) \tag{2}$$

Figure 1(c) shows an example schedule for the DAG.

Figure 1 (a) DAG of an application, (b) computation cost matrix, and (c) an example schedule(with idle time and communication time due to dependencies) with makespan = 70

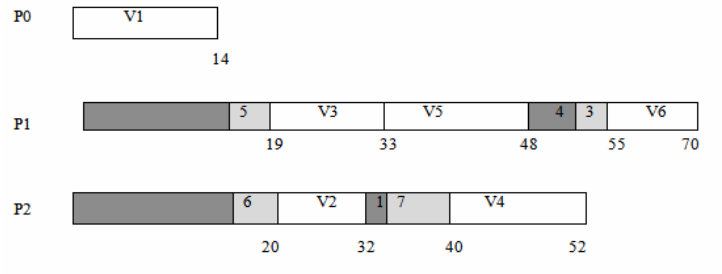


(a)

	P_0	P_1	P_2
V_1	14	13	15
V_2	13	14	12
V_3	15	14	16
V_4	11	13	12
V_5	18	15	16
V_6	16	15	14

(b)

Figure 1 (a) DAG of an application, (b) computation cost matrix, and (c) an example schedule(with idle time and communication time due to dependencies) with makespan = 70 (continued)



(c)

Every processor has a constant failure rate and let λ_j denote the failure rate of processor j. The probability that a processor j executes all its tasks successfully is given by an exponential law:

$$p_{succ}^{j}(S) = e^{-\lambda_j c_j(s)} \tag{3}$$

It is assumed that faults are independent, therefore, the probability that schedule S finished correctly is:

$$p_{succ} = e^{-\sum_j \lambda_j c_j(s)} \tag{4}$$

The reliability index (rel) is defined by:

$$rel(s) = \sum_j \lambda_j c_j(s) \tag{5}$$

Minimising the objective function *rel* is equivalent to maximising the probability of success of the schedule on a parallel heterogeneous system subject to failure. For solving the task scheduling problem the objectives C_{max} and *rel* are to be minimised simultaneously (i.e., minimising the makespan and maximising the probability of success of the whole schedule).

3 Multi-objective problem

The objective of the task scheduling problem is to minimise the two objective functions: makespan and reliability index simultaneously. This problem is formulated as a non-linear multi-objective optimisation problem as follows:

$$Min\ f = [F1, F2] \tag{6}$$

3.1 Objective functions

1 Makespan objective. The makespan of a schedule S is calculated as:

$$F_1 = C_{\max}(S) \tag{7}$$

where $C_{max}(S)$ is the time at which the last task completes for a particular schedule S. It is calculated using equation (2).

2 Reliability index objective. The reliability index of a schedule S as:

$$F_2 = rel(s) \tag{8}$$

4 Implementation of MOGA to task scheduling problems

4.1 Step 1 (initialisation)

The chromosome representation consists of two parts: the matching string and the scheduling string (Wang et al., 1997). The matching string is obtained by randomly assigning each subtask to a processor. The matching string *ms* is a vector of length V, the number of subtasks. $ms(i) = j$ indicates task v_i is assigned to processor P_j. The scheduling string is formed by performing a topological sort of the DAG, i.e., a total ordering of the nodes in the DAG that obeys the precedence constraints. The scheduling string, *ss* is a vector of length V, where $ss(k) = i$ indicates v_i is the kth subtask in the scheduling string. The scheduling string gives an ordering of the subtasks that is used by the evaluation step.

The initial population comprises of a predefined number of chromosomes. Some random scheduling strings are generated and for each chosen scheduling string randomly a subtask is selected and is moved to another position without violating data dependency constraint. Similarly, for some random number of matching strings the processor assignment for the selected pair is changed randomly to another processor. Each newly generated chromosome is checked against those previously generated for uniqueness.

4.2 Step2 (crossover)

Different crossover operators are developed for scheduling strings and matching strings. The crossover operator for the scheduling strings randomly chooses some pairs of the scheduling strings. For each pair, it randomly generates a cut-off point, which divides the scheduling strings of the pair into top and bottom parts. Then, the subtasks on each bottom part are reordered. The new ordering of the subtasks in one bottom part is the relative positions of these subtasks in the other original scheduling string in the pair. For matching string, randomly some pairs are chosen and for each pair cut-off point are generated to divide both matching strings of the parts into two parts. Then the processor assignments of the bottom parts are exchanged. The probability for performing crossovers was determined by experimentation and set to 0.8.

4.3 Step 3 (mutation)

Different mutation operators are developed for scheduling strings and matching strings. The scheduling string mutation operator randomly chooses some scheduling strings. Then for each chosen scheduling string, it randomly selects a victim subtask. The valid range

of the victim subtask is the set of the positions in the scheduling string at which this victim subtask can be placed without violating any data dependency constraints. Specifically, the valid range is after all source subtasks of the victim subtask and before any destination subtask of the victim subtask. After a victim subtask is chosen, it is moved randomly to another position in the scheduling string within its valid range. Then the matching string mutation operator is applied on its corresponding matching string. On each chosen matching string, it randomly selects a subtask/processor pair. Then the processor assignment for the selected pair is changed randomly to another machine. The probability for performing mutations was determined by experimentation and set to 0.4.

4.4 Step 4 (evaluation)

For the task scheduling problem, the combination of parent and offspring solutions are evaluated by two fitness functions F_1 *and* F_2 using the weighted sum approach. In this method, the makespan function given in (2) and reliability index function in (5), are weighted according to their relative importance and the two weighted functions are added together to produce the objective function as:

$$f(s) = \omega \cdot \frac{F_1 - \min\_F_1}{\max\_F_1 - \min\_F_1} + (1-\omega) \cdot \frac{F_2 - \min\_F_2}{\max\_F_2 - \min\_F_2} \tag{9}$$

where ω, is the weighting coefficient in the range 0 and 1. When ω = 1, only the makespan objective is considered and when ω = 0, the reliability index is considered. min_F_1 and max_F_1 are the minimum and maximum values of the makespan objective in the current generation respectively. Similarly, min_F_2 and max_F_2 are the minimum and maximum value of the reliability index objective. By varying the values of ω, the tradeoff between the makespan and reliability index can be determined over the range of values of ω. Actually, in this method the bi-objective task scheduling problem is converted to a single objective scheduling problem using the linear combination of both objectives.

4.5 Step 5 (selection)

Sort the solutions in the ascending order with respect to the *f* values in equation (9) and select the first *I* solutions as parents.

Steps 2 to 5 are repeated until their stopping criterion (maximum number of iterations) is met.

5 Proposed modified algorithm

In this paper, an MA has been proposed which takes the good characteristics of bacteriological algorithm (Kim et al., 2007) and genetic algorithms.

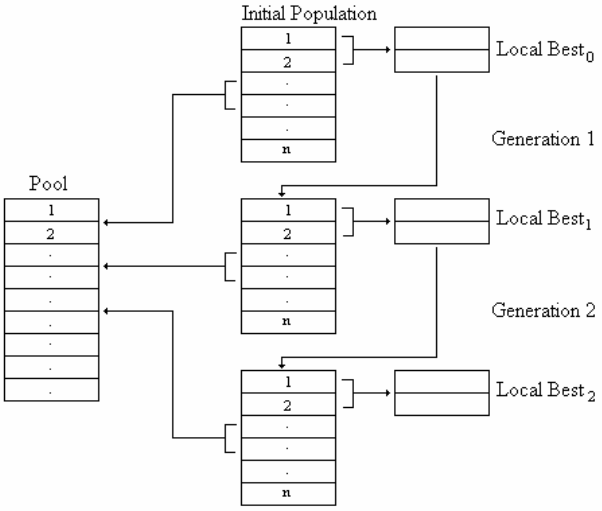
Initial population is generated as given in Section 4 above. One of the chromosomes of the initial population is derived from a list scheduling heuristic and added to the initial population.

Fitness of the chromosomes in initial population is evaluated. Best chromosomes are those with minimum *f* value mentioned in equation (9). The two best chromosomes from

initial population are selected and stored in local best. Hence, local best consists of two chromosomes which are best among the entire population for the current generation.

The genetic algorithm operations like crossover, mutation are performed on these two parents and the offspring for the next generation is produced. After generating the *n* offspring, fitness is evaluated for each of the offspring. Now the two best chromosomes obtained are compared with the local best. If the latest best chromosomes are found to be better, then the local best is replaced with these best chromosomes. If the latest best chromosomes are not better than the local best, the local best chromosomes are retained for next generation. Each generation begins with consideration of local best chromosomes as the parent chromosomes for that generation as illustrated in the Figure 2.

Figure 2 Local best and pool maintenance

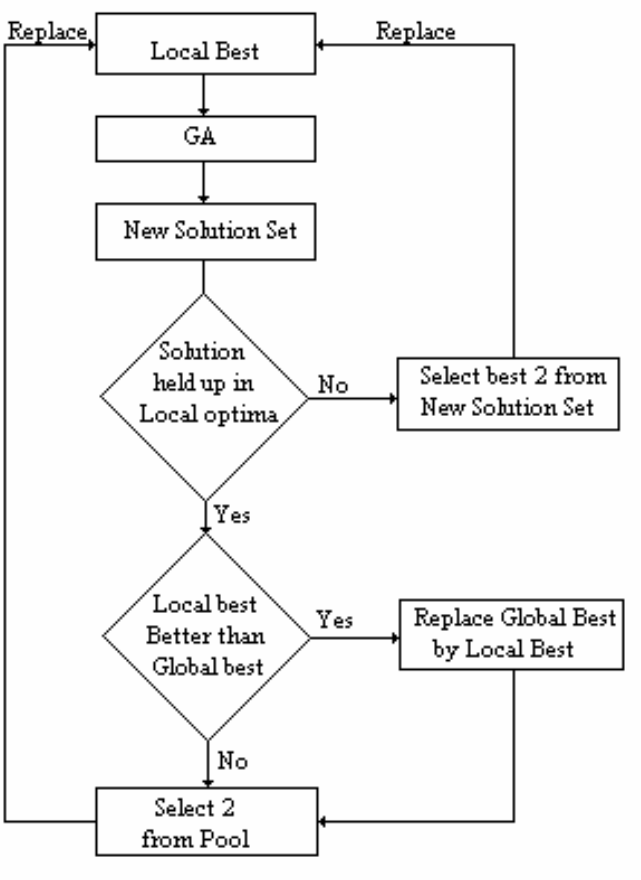


5.1 Pool maintenance and backtracking

Generally, the solution converges towards the optimal solution in every generation. In some cases, the solution stops converging before it reaches optimal solution. To overcome this, back tracking method, in which the next available best solutions of previous generations are back tracked and considered as parents for current generation. Hence, for back tracking, it is needed to store the next available best solutions of each generation. This process of storing the next available best solutions of each generation in a separate pool for future usage is called pool maintenance. This is shown in Figure 2.

If the solution starts being held in local optima, two of the chromosomes are taken from the pool and considered as parents or local best for the next generation. In order to avoid the loss of the previous local best values, the local best are stored in global best before applying backtracking. This is illustrated in Figure 3. For the next time, when backtracking is applied, the local best is compared and replaced in global best only if the local best value is better than the chromosomes already existing in global best. At the end of all the generations, the best solution obtained is available in global best.

Figure 3 Backtracking



6 Results and discussion

To test the effectiveness of the MOEP and MOGA for the task scheduling, the solutions were obtained for real time numerical application graph, parallel Gauss elimination algorithm (Topcuoglu et al., 2002). We have considered the matrix size, m = 10 and hence the number of nodes =54. The CCR values are {0.1, 1,5, and 10}.The value of ACC is set as 20. The heterogeneity factor, β = 0.4. A similar experimental set-up was used in our previous study (Chitra and Venkatsh, 2010) to compare the MOGA and MOEP algorithms for task scheduling.

We have generated randomly a set of four processors, where λ is chosen uniformly in the range $[10^{-2}, 10^{-3}]$ and these numbers are not very realistic but provide comprehensive results that are easy to read on the graphs. In Dongarra et al. (2007), authors have shown that for general task graphs, it is easy to extend most of the heuristics designed for optimising the makespan by taking into account the reliability. Hence, we have used MA to solve the task graph for optimising the makespan by considering the reliability.

For comparing the results we have implemented the weighted sum approach, BGA (Dogan and Ozguner, 2005). Table 1 gives the results of the best makespan and reliability index value obtained for the 54 node Gaussian elimination test graph with m =10 and CCR =0.1 using MA and BGA. From Table 1 we can infer that MA is capable of producing optimal value at a lower time compared to the BGA (Dogan and Ozguner, 2005). Though the fitness evaluation remains same for MA and BGA the difference in computation time stems from the fact that we preserve the good solution and extend them for populating the parent population for the next generation. Figure 3 demonstrates the capability of convergence of MA compared to BGA. The solution starts converging to near optima at 250th generation itself compared to 485th generation using BGA.

Table 2 gives the results of the best makespan and reliability index value obtained by executing MA and BGA for CCR = 1. The table gives the best makespan and reliability index for three weight assignments 0, 0.5, 1. It can be observed that when w = 1, the importance is given to makespan and weight w = 0, the importance is for reliability. The obtained non-dominated solutions for 11 runs of MA and GA for the same test problem of m = 5 is shown in Figure 5.

Figure 4 Convergence graph for Gauss elimination task graph with CCR = 1 and weight w = 1 (see online version for colours)

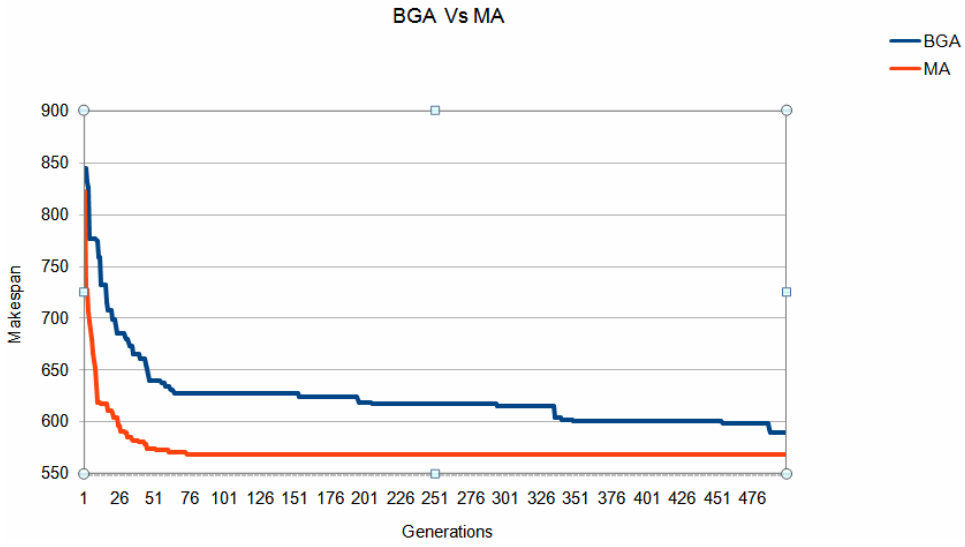


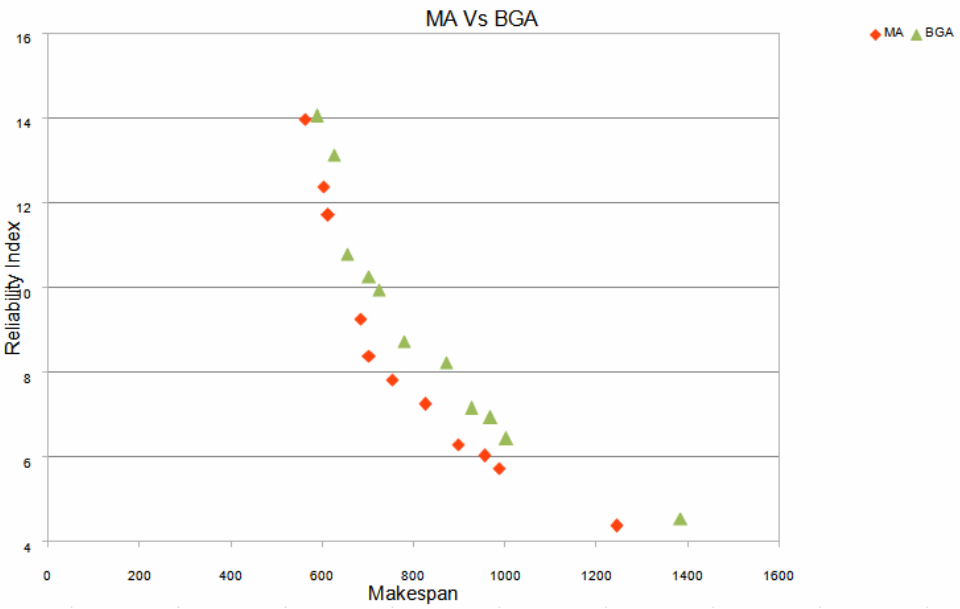
Table 1 Best solutions for MA, BGA

CCR	*Modified algorithm(MA)*			*BGA*		
	Makespan	*Reliability index*	*Execution time*	*Makespan*	*Reliability index*	*Execution time*
1	591	13.17	274.42	616	9.48	427.41
5	1,070	15.93	279.02	1,103	12.20	414.75
10	1,426	23.54	308.44	1,490	22.47	425.18

Table 2 Best solutions for MA, BGA for CCR = 1

Weight	*Modified algorithm(MA)*			*BGA*		
	Makespan	*Reliability index*	*Execution time*	*Makespan*	*Reliability index*	*Execution time*
0	1,244	4.36	274.37	1,383	4.54	425.62
0.5	752	7.82	274.42	616	9.48	427.41
1	562	13.95	273.13	587	14.04	426.11

Figure 5 Obtained non-dominated points for Gauss elimination task graph with CCR = 1 and weight w in the increments of 0.1 (see online version for colours)



7 Conclusions

This paper proposes an MA which inherits the good characteristics of bacteriological algorithm and genetic algorithm and is used for solving the task scheduling problem considering two objectives. The two objectives: makespan and reliability index are considered in this paper. The result obtained from the MA is compared with BGA for parallel Gaussian elimination task graph. The simulations confirm that MA can produce the optimal solutions at faster execution times.

References

Ahmad, I. and Kwok, Y. (1994) 'A new approach to scheduling parallel programs using task duplication', *Proc. Int'l Conf. Parallel Processing*, Vol. 2, pp.47–51.

Braun, T. et al. (2001) 'A comparison of study of static mapping heurisitcs for a class of meta-tasks on heterogeneous computing systems', *Journal of Parallel and Distributed Computing,* Vol. 61, No. 6, pp.810–837.

Carretero, J., Xhafa, F. and Abraham, (2007) A. 'Genetic algorithm based schedulers for grid computing systems', *International Journal of Innovative Computing, Information and Control*, December, Vol. 3, No. 6.

Chang, Y. and Ranka, S. (1992) 'Applications and performance analysis of a compile-time optimization approach for list scheduling algorithms on distributed memory multiprocessors', *Proc. Supercomputing*, November, pp.512–521.

Chitra, P. and Venkatsh, P. (2010) 'Multiobjective evolutionary computation algorithms for solving task scheduling problem on heterogeneous systems', *International Journal of Knowledge-Based and Intelligent Engineering Systems*, January, Vol. 14, No. 1, pp.21–30.

Dogan, A. and Ozguner, F. (2005) 'Biobjective scheuling algorithms for execution time-reliability trade-off in heterogeneous computing systems', Technical Report 2005-001, Department of Electrical and Electronics Engineering, Anadolu University, Eskisehir, Turkey.

Dongarra, J.J., Jeannot, E., Saule, E. and Shi, Z. (2007) 'Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems', *Proceedings of the Nineteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, pp.280–288.

El-Rewini, H. and Lewis, T.G. (1990) 'Scheduling parallel program tasks onto arbitrary target machines', *Journal of Parallel and Distributed Computing,* Vol. 9,pp.138–153.

Fogel, D.B. and Fogel, L.J. (1996) 'Using evolutionary programming to schedule tasks on a suite of heterogeneous computers', *Computers & Operation Research*, June, Vol. 23, No. 6, pp.527–534.

Gary, M.R. and Johnson, D.S. (1979) *Computers and Intractability: A Guide to the Theory of NP Completeness,* W.H. Freman and Co.

Hou, E.S.H. et al. (1994) 'A genetic algorithm for multiprocessor scheduling', *IEEE Trans. Parallel Distrib Systems*, Vol.5, No. 2, pp.113–120.

Kim, D.H., Abraham, A. and Cho, J.H. (2007) 'A hybrid genetic algorithm and bacterial foraging approach for global optimization', *An International Journal of Information Sciences*, pp.3918–3937.

Kwok, Y.K. and Ahmad, I. (1996) 'Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors', *IEEE Trans. Parallel Distributed Systems,* May, Vol. 7, No. 5, pp.506–521.

Liou, J. and Palis, M.A. (1996) 'An efficient clustering heuristic for scheduling DAGs on multiprocessors', *Proc. Symp. Parallel and Distributed Processing*.

Maheswaran, M. and Siegel, H.J. (1993) 'A dynamic matching and scheduling algorithm for heterogeneous computing systems', *Proc. Heterogeneous Computing Workshop*.

Shroff, P., Watson, D.W., Flann, N.S. and Freund, R. (1996) 'Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments', *Proc. Heterogeneous Computing Workshop*, pp.98–104.

Singh, H. and Youssef, A. (1996) 'Mapping and scheduling heterogeneous task graphs using genetic algorithms', *Proc. Heterogeneous Computing Workshop*, pp.86–97.

Topcuoglu, H., Hariri, S. and Wu, M-Y. (2002) 'Performance-effective and low-complexity task scheduling for heterogeneous computing', *IEEE Trans. on Parallel and Distributed Systems,* March, Vol. 13, No. 3, pp.260–274.

Wang, L. et al. (1997) 'A genetic-algorithm-based approach for task matching and scheduling in heterogeneous computing environments', *Journal of Parallel and Distributed Computing*, Vol. 47, No. 1, pp.8–22.

Yang, T. and Gerasoulis, A. (1994) 'DSC: scheduling parallel tasks on an unbounded number of processors', *IEEE Trans. Parallel and Distributed Systems*, September, Vol. 5, No. 9, pp.951–967.

Ye, G., Rao, R. and Li, M. (2006) 'A multiobjective resources scheduling approach based on genetic algorithms in grid environment', *Procs of the Fifth International Conference on Grid and Cooperative Computing Workshops,* pp.504–509.