Sādhanā Vol. 36, Part 2, April 2011, pp. 167–180. © Indian Academy of Sciences

Comparison of evolutionary computation algorithms for solving bi-objective task scheduling problem on heterogeneous distributed computing systems

P CHITRA[1,*], P VENKATESH[2] and R RAJARAM[3]

[1]Department of Computer Science and Engineering,
[2]Department of Electrical and Electronics Engineering,
[3]Computer Science and Information Technology Departments, Thiagarajar College of Engineering, Madurai 625 015, India
e-mail: pccse@tce.edu; pveee@tce.edu; rrajaram@tce.edu

MS received 22 August 2009; revised 8 December 2010; accepted 18 February 2011

Abstract. The task scheduling problem in heterogeneous distributed computing systems is a multiobjective optimization problem (MOP). In heterogeneous distributed computing systems (HDCS), there is a possibility of processor and network failures and this affects the applications running on the HDCS. To reduce the impact of failures on an application running on HDCS, scheduling algorithms must be devised which minimize not only the schedule length (makespan) but also the failure probability of the application (reliability). These objectives are conflicting and it is not possible to minimize both objectives at the same time. Thus, it is needed to develop scheduling algorithms which account both for schedule length and the failure probability. Multiobjective Evolutionary Computation algorithms (MOEAs) are well-suited for Multiobjective task scheduling on heterogeneous environment. The two Multi-Objective Evolutionary Algorithms such as Multiobjective Genetic Algorithm (MOGA) and Multiobjective Evolutionary Programming (MOEP) with non-dominated sorting are developed and compared for the various random task graphs and also for a real-time numerical application graph. The metrics for evaluating the convergence and diversity of the obtained non-dominated solutions by the two algorithms are reported. The simulation results confirm that the proposed algorithms can be used for solving the task scheduling at reduced computational times compared to the weighted-sum based biobjective algorithm in the literature.

Keywords. Task scheduling; makespan; reliability; directed acyclic graph (DAG); MOGA; MOEP.

*For correspondence

1. Introduction

Heterogeneous systems consist of heterogeneous suite of machines, high speed interconnections interfaces, communication protocols and programming environments and provide a variety of architectural capabilities that has diverse execution requirements. One of the key challenges of heterogeneous systems is the scheduling problem. 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 minimize makespan. The task scheduling problem has been solved several years ago and is known to be NP-complete (Gary & 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. These algorithms are applicable for homogeneous systems. List scheduling algorithms (Kwok & Ahmad 1996) assign priority to each task. Based on their priorities, tasks are allocated to processors to minimize a predefined cost function. The basic idea of clustering algorithm (Topcuoglu *et al* 2002) 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 algorithm (Ahmad & Kwok 1994; Chang & Ranka 1992) 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 minimized. Genetic Algorithm (Hou *et al* 1994; Shroff *et al* 1996; Wang *et al* 1997; Braun *et al* 2001) is one of the widely studied guided random search techniques, for task scheduling problems. The task scheduling problems have been studied for heterogeneous computing systems (Rewini-El & Lewis 1990; Singh & Youssef 1996; Wang *et al* 1997; Topcuoglu *et al* 2002). Evolutionary Programming (EP) is a mutation-based evolutionary algorithm applied for discrete search spaces and used in task scheduling (Fogel & Fogel 1996; Kwok & Ahmad 1999).

Reliability of the systems in the heterogeneous systems has a vital role in scheduling the application on to the processors (Girault *et al* 2009; Qin & Jiang 2006). As heterogeneous systems become larger and larger, the issue of reliability of such systems needs to be addressed. Indeed, the number of possible failures increases with the size of the hardware. Therefore, nowadays, it is not possible to ignore the fact that an application running on a very large system can crash due to hardware failure. Several approaches can be employed to solve this problem. One approach is based on task duplication where each task is executed more than once in order to decrease the probability of failure. The main problem of this approach is that it requires a large number of processors. Alternatively, it is possible to checkpoint the application and restart the application after a failure (Dogan & Ozguner 2002, 2005; Dongarra *et al* 2007). However, in case of failure, the application is slowed down by the restart mechanism, which requires the user to restart the application on a subset of processors and repeat some communications and computations. Hence, in order to minimize the impact of the restart mechanism, it is important to reduce the risk of failure. Moreover, even in the case where there is no checkpoint-restart mechanism, it is important to guarantee that the probability of failure of the application is kept as low as possible. Unfortunately, increasing the reliability implies, an increase of the execution time (a fast schedule is not necessarily a reliable one). This justifies the search for algorithms that both minimize makespan and maximize reliability (Dogan & Ozguner 2005).

Multi-Objective Evolutionary Algorithm (MOEA) combines two major disciplines: Evolutionary computation and theoretical frameworks of multi-criteria decision making. Defining multiple objectives for the task scheduling problem for generating efficient schedules at reduced computational times are of interest in 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 & Ozguner (2005).

The biobjective algorithm (Dogan & Ozguner 2005) uses weighted-sum approach for tackling the two objectives (makespan and reliability). The population has both valid and invalid chromosome, where the invalid chromosomes are to be treated separately in the selection phase.

This paper considers only the two objectives of minimizing the makespan (schedule length) and maximizing the reliability in the multiobjective task scheduling problem. Multiobjective GA and Multiobjective EP algorithms both based on non-dominated ranking for selection are developed and compared for solving the task scheduling problem. The convergence and diversity among the obtained non-dominated solutions for the two algorithms are evaluated. The non-dominated solutions obtained by the MOEAs are compared with those obtained by the weighted-sum based biobjective algorithm in the literature. We have also simulated random task graphs using the task graph generator (Topcuoglu *et al* 2002) and compared the results obtained by MOGA, MOEP.

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 *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. $e_{i,j}$ is a directed edge between 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 times. Edge $e_{i,j}$ has a communication time $c_{i,j}$ 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 Directed Acyclic Graph (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 completed.

The heterogeneous computing system is a set P of p heterogeneous machines (processors) connected in a fully connected topology.

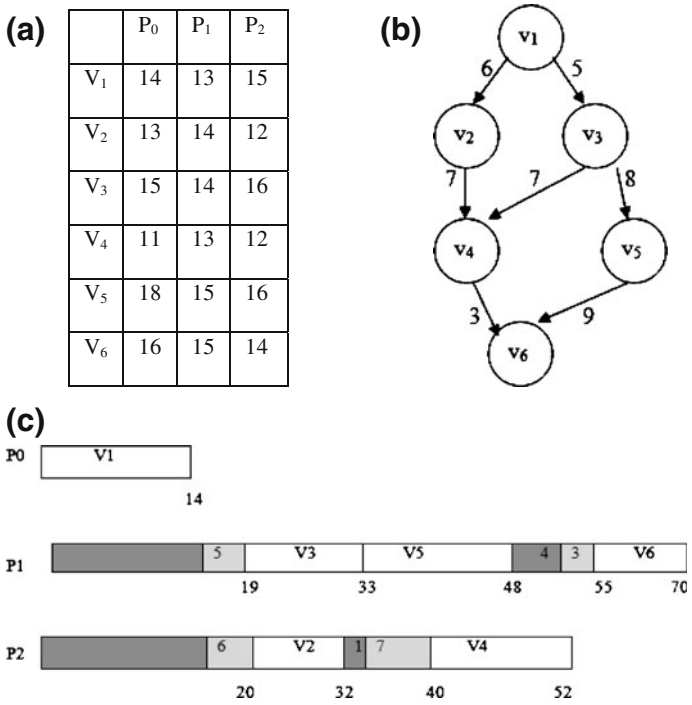
Figure 1a represents an example of an application modelled by DAG $G = (V, E)$, where $V = \{v_1, v_2, v_3, v_4, v_5, v6\}$ represents the set of 6 subtasks to be executed, and the set of weighted, directed edges E represents the communication requirement between subtasks. In this example, E consists 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 1b 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: $V \rightarrow P$ 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

$$c_j(s) = \sum\nolimits_{i \in v(j,s)} st_{ij} + w_{ij},$$



	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

Figure 1. (**a**) DAG of an application; (**b**) computation cost matrix; (**c**) an example schedule with idle time due to dependency and communication time due to predecessors on other processors.

where st_{ij} denotes the start time of the task i on processor j. The start time of the entry task is assumed to be zero. Other tasks' start time can be computed by considering the completion time of all immediate predecessors of the task. The communication time, st_{ij} is added to the start time, if the dependent tasks are allocated to different processors. The makespan of a schedule is the time where all tasks are completed.

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

Figure 1c shows an example schedule for the DAG. 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

$$p^j_{succ}(S) = e^{-\lambda_j c_j(s)}.$$

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

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

The reliability index (rel) is defined by

$$rel(s) = \sum_j \lambda_j c_j(s).$$

Minimizing the objective function *rel* is equivalent to maximizing 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 minimized simultaneously (i.e., minimizing the makespan and maximizing the probability of success of the whole schedule).

3. Optimization problem

The task scheduling problem is formulated by considering the objectives of minimizing the makespan and maximizing the reliability of the schedules

3.1 *Multi-objective problem*

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

$$Min\ f = [F_1, F_2].$$

3.2 *Objective functions*

(i) Makespan objective. The makespan of a schedule S is calculated as

$$F_1 = C_{\max}(S),$$

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

(ii) Reliability index objective. The reliability index of a schedule S as

$$F_2 = rel(s),$$

where *rel(s)* is calculated using the equation (5).

3.3 *Weighted-sum method*

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

$$Min\ f = \omega F_1 + (1 - \omega)F_2,$$

where $F1$ is the makespan function, $F2$ is the reliability index function and ω is the weighting coefficient in the range 0 and 1. Since the objectives have varying range, they are normalized in the range 0 to 1 (Dogan & Ozguner 2005). When $\omega = 1$, only the makespan objective is considered and when $\omega = 0$, only the reliability index objective is accounted. By varying the values of ω, the trade-off 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.

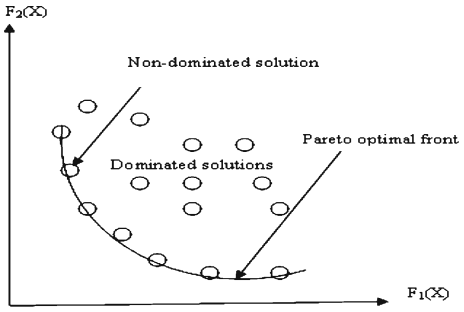


Figure 2. Pareto optimal front for a MOP.

4. Principles of multi-objective optimization

Many real world problems involve simultaneous optimization of multiple objectives that often are competing. In a multi-objective optimization problem (MOP), one solution that is best with respect to all objectives may not be achieved. Usually the aim is to determine the trade-off surface, which is a set of non-dominated solution points, known as Pareto optimal solutions. Every individual in this set is an acceptable solution. Mathematically a MOP can be formulated as follows:

$$\text{Minimize} \quad f_i(x) \quad i = 1, 2, \ldots N_{obj},$$

where f_i is the i^{th} objective function, x is a decision vector that represents a solution, N_{obj} is the number of objectives. For a MOP problem, any two solutions x^1 and x^2 can have one of two possibilities: one dominates the other or none dominates the other. In a minimization problem, without loss of generality, a solution covers x^1, or dominates x^2 , if the following two conditions are satisfied:

1. $\forall i \in \{1, 2, \ldots N_{obj}\} \rightarrow f_i(x^1) \leq f_j(x^2)$
2. $\exists\, j \in \{1, 2, \ldots N_{obj}\} \rightarrow f_j(x^1) \leq f_j(x^2)$.

If any of the above conditions is violated, the solution x^1 does not dominate the solution x^2. If x^1 dominates the solution x^2, x^1 is called the non-dominated solution. The solutions that are non-dominated within the entire search space are denoted as Pareto optimal solution and constitute the Pareto optimal set. This set is also known as Pareto optimal front. The concept of Pareto optimal front for a MOP is shown in figure 2.

Recently, studies on evolutionary algorithms have shown that these algorithms can be efficiently used to eliminate most of the problems confronting of classical methods (Deb 2001), such as multiple runs in order to get the Pareto optimal front. The goal of a multi-objective optimization algorithm is not only to guide the search toward the Pareto optimal front, but also to maintain population diversity in the set of non-dominated solutions. Various researchers have proposed the multi-objective evolutionary algorithms such as NPGA, NSGA-II, and SPEA for multiobjective optimization problems (Deb 2001).

5. Implementation of MOGA and MOEP to task scheduling problem

MOGA approach is a probabilistic, global search technique. This starts with a population of randomly generated candidate solutions and evolves towards better set of solutions (Pareto optimal

front solutions) over number of generation or iteration. Its implementation for task scheduling problems is as given below.

5.1 *Step 1 (initialization)*

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 k_{th} subtask in the scheduling string. The scheduling string gives an ordering of the subtasks that is used by the evaluation step. Figure 3 illustrates one of the chromosomes for the DAG in figure 1a for V = 6 and P = 3.

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. Since every time we consider a topological sort of the DAG, there is a non-zero probability that a chromosome will represent an infeasible schedule (Chitra & Venkatesh 2010). Thus, this representation conforms to any possible (valid) solution, even after the crossover and mutation operations are carried out.

5.2 *Step 2 (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 pair of cut-off point, which divides the scheduling string into top and bottom parts. Then, the subtasks in 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, thus guaranteeing that the newly generated scheduling strings are valid schedules. For matching string, some pairs are chosen randomly and for each pair cut-off point is 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.

5.3 *Step 3 (mutation)*

Different mutation operators are developed for scheduling strings and matching strings. The scheduling string mutation operator chooses some scheduling strings. Then for each chosen scheduling string, it 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

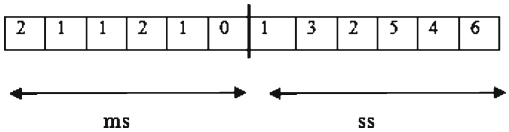


Figure 3. One of the chromosomes for the DAG in figure 1a.

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 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 to more reliable machine. The probability for performing mutations was determined by experimentation and set to 0.4.

5.4 *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 separately and stored in F_{1i} and F_{2i}, for $i = 1,2,\ldots, 2*I$.

5.5 *Step 5 (non-dominated ranking)*

For the task scheduling problem, the ranking procedures consist of finding the non-dominated solution in the current population $2*I$ using their fitness F_{1i} and F_{2i}. Assign a rank to each solution by counting the number of solutions that dominates each solution.

5.6 *Step 6 (selection)*

Sort the $2*I$ solutions in the ascending order with respect to the rank assigned to each solutions. Now the first I solutions are selected as parents without their rank values for the next generation.

Steps 2 to 6 are repeated until their stopping criterion (Maximum number of iterations) is met, and then the present parent solutions are the optimal Pareto optimal front solutions.

In the case of MOEP technique the steps involved are initialization, mutation, evaluation, non-dominated solution ranking, and selection.

6. Simulation results and discussion

To test the effectiveness of the MOEP and MOGA for the task scheduling, the solutions were obtained for random task graph. Random task graph was generated using the method as proposed by Topcuoglu *et al* (2002). There are few parameters involved in the random task graph generation such as: number of nodes, shape parameter, out degree, average computation cost, computation to cost ratio, average communication cost, heterogeneity factor.

In this paper, the input parameters of the directed acyclic graph generation algorithm were set with the following values:

- Number of nodes (tasks) in the graph, V. The value of V is assigned from the set of {6,50,100,200,500}.
- Shape parameter of the graph, α. The height of a DAG is generated from a uniform distribution with mean equal to $\sqrt{\frac{V}{\alpha}}$. The width for each level in a DAG is selected from a uniform distribution with mean equal to $\alpha \times \sqrt{V}$. α is assigned value from the set {0.2,0.4,1.0}.
- Out degree value for all nodes are randomly generated from a uniform distribution with mean equal to out degree.
- Communication to computation ratio, CCR. CCR is the ratio of the average communication cost to the average computation cost. CCR = {0.1,1,2,10}.

- Average computation cost in the graph, ACC. This is the average time required to complete the task on all of the available processors. Computation costs are generated randomly from a uniform distribution with mean equal to ACC. The value of ACC is from set {10,15,100,200}.
- Average communication cost is derived as avg_comm = CCR × ACC.
- Heterogeneity factor, β. This value indicates the range percentage of computation cost on processors. The computation cost of each task vi on each processor p_j in the system is randomly set from the range:

$$\text{ACC} \times \left(1 - \frac{\beta}{2}\right) \leq w_{ij} \leq \text{ACC} \times \left(1 + \frac{\beta}{2}\right).$$

The figures 1a and b are the output of the random task graph generator with the following parameters: V = 6, $\alpha = 0.4$, CCR = 2, ACC = 15, $\beta = 0.2$ and the number of processors P = 3.

We have also generated randomly a set of 4 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. Dongarra *et al* (2007) have shown that for general task graphs, it is easy to extend most of the heuristics designed for optimizing the makespan by taking into account the reliability. Hence, we have used GA and EP to solve the task graph for optimizing the makespan by considering the reliability.

In addition to randomly generated task graphs, we also considered application graphs of a real world problem: Parallel Gauss elimination algorithm (Kwok & Ahmad 1996). The task graph for this application is shown in figure 4.

Table 1 gives the results of the best makespan and reliability index value obtained for the 18 node Gaussian elimination test graph (Kwok & Ahmad 1996) using simple GA and simple EP. The GA and EP algorithms were implemented in its simple form to consider the

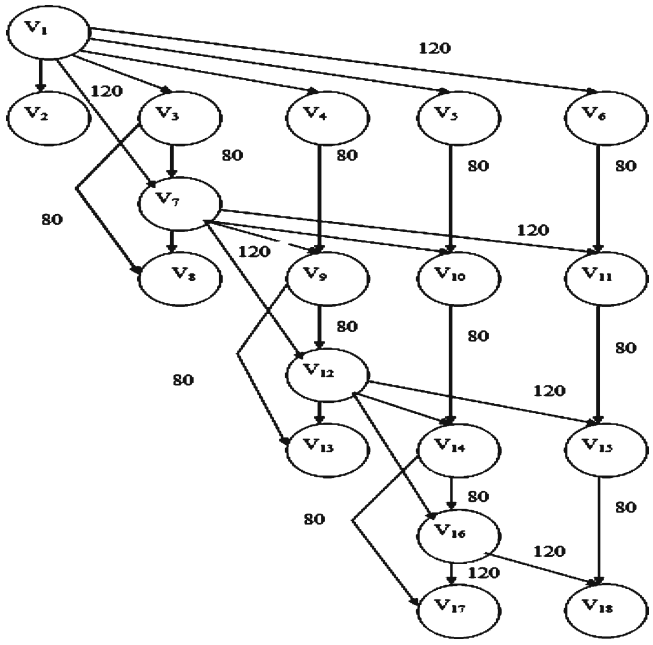


Figure 4. 18 Node standard task graph.

Table 1. Best values of makespan and reliability index for 18 nodes test graph.

	Best makespan		Best reliability index	
	GA	EP	GA	EP
Makespan	467	472	600	604
Reliability index	6.01	6.81	4.16	4.54

individual objectives separately. This result was useful for assessing the minimal value of the fitness corresponding to the individual objectives in the obtained pareto optimal front. Table 2 gives the output for the weighted-sum (WS) approach (Dogan & Ozguner 2005) for the 18 node test graph. Various linearly distributed weights (ω) from 0 to 1 in the interval of 0.1 are used in the weighted-sum approach. The trade-off optimal solutions are obtained by executing the weighted-sum method program in 11 separate runs and shown in figure 5. This weighted-sum method (Dogan & Ozguner 2005) gives the trade-off solutions for the two objectives. Though, this method is simple and flexible, the selection of the weight values is critical in deciding the trade-off between the two objectives. The difficulty in selecting the appropriate weights is overcome by the MOEAs since the objectives are considered as such for finding the non-dominated points. Further, to determine the effectiveness of MOGA method, the test results of the MOGA method is compared with MOEP for the task scheduling problem and with weighted-sum method.

MOGA and MOEP methods are applied to the same problem to obtain the pareto optimal solutions. The obtained global Pareto optimal front using MOGA and MOEP method in a single run is compared with weighted-sum method and it is as shown in figure 6. It is exactly matching with the 11 optimal solution points those obtained using weighted-sum method. Furthermore, the other intermediate solutions are also available which is quite useful. To verify the effectiveness and the convergence capability of the proposed MOGA method, the solutions status at intermediate iteration level such as at the end of 20^{th} iteration is shown in figure 7. This gives a better understanding of the proposed MOGA method. Table 3 shows the comparison of the makespan values and the minimum value for reliability index obtained for different random task graphs using GA and EP. Table 4 shows the parameter settings for GA, EP, Weighted-sum, MOGA and MOEP methods for solving the 18 node test graph. The number of generations was fixed to the figures given in the table if the quality of the elite chromosomes did not improve over 20 generations. The number of tasks is 18 and the number of processors is set to 4 and the heterogeneity factor was set to 0.2. The computation time for executing the Evolutionary algorithms are also reported in the table 4. The simulation results show that the failure probability of the random task graph increases in proportion to the size of the application. This is due to the fact that when the size of an application increases, processors have to be failure-free for longer periods of time. Since the failure probability of a processor increases exponentially within the time period for which the processor must remain failure-free, the failure probability of the application increases. Hence, bigger applications will be more susceptible to failures, unless tasks of the application

Table 2. Weighted-sum results for 18 node test graph.

Weight ω	1	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1	0
Makespan	467	470	472	475	485	495	523	524	557	560	600
Reliability index	6.01	5.32	5.25	5.11	4.9	4.73	4.46	4.3	4.24	4.21	4.16

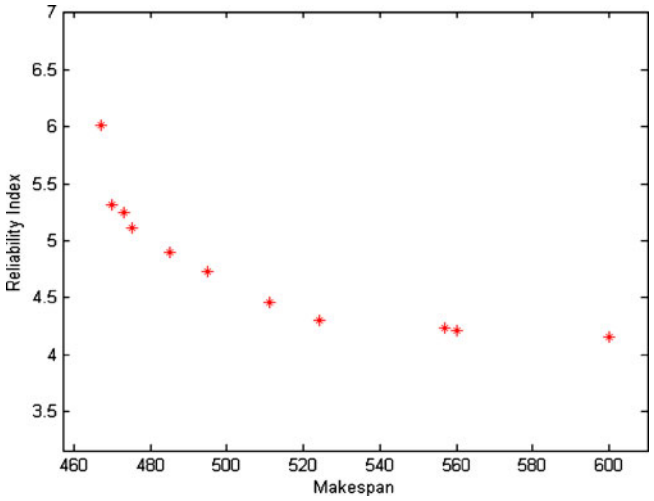


Figure 5. Pareto optimal front of linear combination in 11 separate runs for 18 node test graph.

are assigned to processors in such a way that the reliability of the processor committed for the execution of the application is accounted for.

There are many performance metrics proposed in the literature for evaluating the multiobjective optimization algorithms. One of the performance metrics namely, *spacing* is used to measure the diversity among non-dominated solutions obtained by MOGA, MOEP and Weighted-sum methods.

Spacing is calculated as a relative distance measure between consecutive solutions in the obtained non-dominated set using the following equation 12 as

$$S = \sqrt{\frac{1}{|Q|}\sum_{i=1}^{|Q|}(d_i - \overline{d})^2},$$

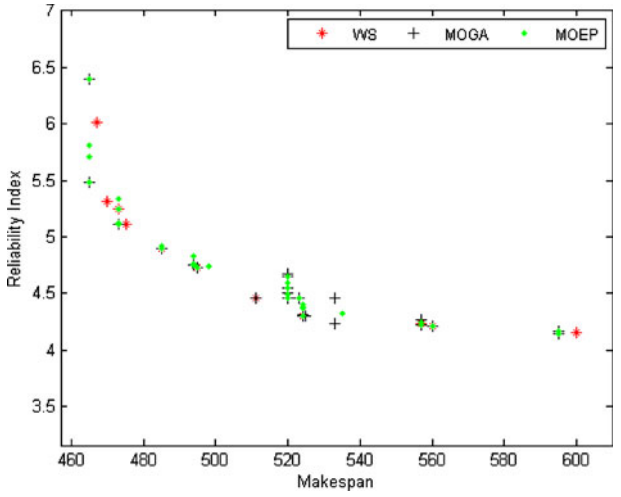


Figure 6. Pareto optimal solution of MOEAs and weighted-sum method for 18 node test graph.

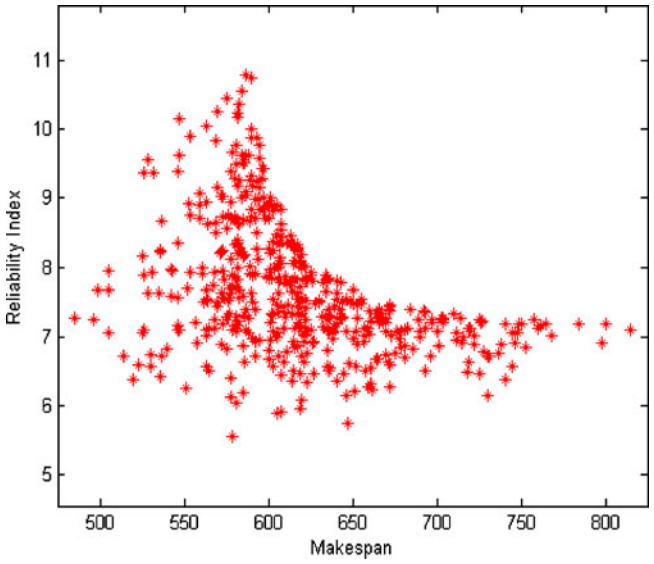


Figure 7. Solution status in MOGA for 20^{th} genertation for 18 node test graph.

Table 3. Best values of makespan and reliability index for random task graphs.

No.of nodes	Two objectives	Best makespan		Best reliability index	
		GA	EP	GA	EP
50	Makespan	1471	1496	2373	2105
	Reliability index	27.43	27.89	21.67	23.46
100	Makespan	4114	4164	5330	4801
	Reliability index	79.84	80.55	71.78	75.51
200	Makespan	10615	10664	11217	11201
	Reliability index	208.68	209.22	201.67	203.97
500	Makespan	30471	30562	31544	30721
	Reliability index	602.71	606.39	590.09	595.52

Table 4. Parameter used in GA, EP, Weighted-sum, MOGA, and MOEP methods along with computation time for 18 node test graph.

Method	GA	EP	Weighted-sum	MOGA	MOEP
Population size	500	500	500	500	500
No.of iteration	100	100	100	1000	1000
Crossover probability	0.8	-	0.8	0.8	-
Mutation probability	0.4	0.4	0.4	0.4	0.4
Computation time (secs)	28.8	26.5	321.2	286.16	263.56

Table 5. Metrics for evaluating the diversity and convergence of MOEAs.

No.of nodes	Spacing		SCM	
	MOEP	MOGA	MOEP	MOGA
50	10.27	9.12	0.42	0.65
100	8.75	7.38	0.51	0.72
200	6.15	4.23	0.59	0.78
500	7.15	5.43	0.61	0.84

where $d_i = \min_{k \in Q \wedge k \neq i} \sum_{m=1} \left| f_m^i - f_m^k \right|$ and $\overline{d}$ is the mean value of the above distance measure $\overline{d} = \sum_{i=1}^{|Q|} d_i \Big/ |Q|$ and Q is the number of solutions in the pareto optimal set. The value f_m^i denotes the m^{th} objective function value of the i^{th} member in the pareto curve. The distance measure is the minimum value of the sum of the absolute difference in objective function values between the i^{th} solution and any other solution in the obtained non-dominated set. This metric measures the standard deviations of different d_i values. When the solutions are near uniformly spaced, the corresponding distance measure will be small (Deb 2001). The spacing metric is evaluated for the 18 node task graph to study the diversity among the points in the pareto optimal front, with the non-dominated solutions of weighted-sum method, as the known pareto optimal points. The spacing value for the MOGA and MOEP are 12.83 and 15.81 respectively. Since, the algorithm finding a set of non-dominated solutions having a smaller spacing is better (Deb 2001), the results shows that MOGA is best suited for the task scheduling problem.

The Set Coverage Metric (SCM) is used for measuring the convergence of the obtained non-dominated points. Given two set of solution vectors A and B, the SCM calculates the proportion of solutions in B, which are weakly dominated by solutions of A. The SCM value for MOGA and MOEP algorithms for the task graph under study are 0.97 and 0.82 respectively. These values confirm that MOGA is better for the problem under study. The spacing and SCM values for the random task graphs are given in table 5.

7. Conclusion

This paper considers the two objectives of makespan and reliability index in the task scheduling problem on heterogeneous systems. The best values for makespan and reliability index obtained using GA and EP for 18 nodes test graph is reported, and compared with the weighted-sum method in the literature. The difficulty in selecting appropriate weights for the objectives to solve the multiobjective problem is overcome by the proposed MOEAs. The pareto optimal front of MOGA and MOEP for the same test system is obtained. The parameters used in GA, EP, weighted-sum, MOGA, MOEP methods along with computation time are given in this paper. Results showed that MOEA algorithms are well-suited for obtaining good pareto optimal solutions in a single run for task scheduling problem by considering the above two objectives. Further MOGA solution is obtained with reduced computation time compared with weighted-sum method in the literature, and it is validated with performance metrics.

References

Ahmad I, Kwok Y 1994 A new approach to scheduling parallel programs using task duplication. *Proc. Int. Conf. Parallel Processing* 2: 47–51

Braun T, et al 2001 A comparison of study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems. *J. Parallel and Distributed Computing* 61(6): 810–837

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

Chitra P, Venkatesh P 2010 Multiobjective evolutionary computation algorithms for solving task scheduling problem on heterogeneous systems. *Int. J. Knowledge-based and Intelligent Eng. Systems* 14(1): 21–30

Deb F K 2001 *Multi-objective optimization using evolutionary algorithms*, New York: Wiley

Dogan A, Ozguner F 2002 Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogenous computing. *IEEE Trans. Parallel Distributed Systems* 13(3): 308–323

Dogan A, Ozguner F 2005 Biobjective scheduling algorithms for execution time-reliability trade-off in heterogenous computing systems. *The Computer Journal* 48(3): 300–314

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

Fogel D B, Fogel L J 1996 Using evolutionary programming to schedule tasks on a suite of heterogeneous Computers. *Computers and Operation Res.* 23(6): 527–534

Freund R F, Siegel H J 1993 Heterogeneous processing. *IEEE Computer* 26(6): 13–17

Gary M R, Johnson D S 1979 *Computers and intractability: A guide to the theory of NP-completeness*. San Francisco: W.H. Freeman and Co

Girault A, Saule E, Trystram D 2009 Reliability versus performance for critical applications. *J. Parallel and Distributed Computing* 69(3): 326–336

Hou E S H, Ansari N, Ren H 1994 A genetic algorithm for multiprocessor scheduling. *IEEE Trans. on Parallel Distributed Systems* 5(2): 113–120

Kwok Y K, Ahmad I 1996 Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors. *IEEE Trans. Parallel and Distributed Systems* 7(5): 506–521

Kwok Y K, Ahmad I 1999 Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys (CSUR)* 31(4): 406–471

Qin X, Jiang H 2006 A novel fault-tolerant scheduling algorithm for precedence constraint tasks in real time heterogeneous systems. *J. Parallel Computing* 32: 331–356

Rewini-El H, Lewis T G 1990 Scheduling parallel program tasks onto arbitrary target machines. *J. Parallel and Distributed Computing* 9: 138–153

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

Singh H, Youssef A 1996 Mapping and Scheduling Heterogeneous Task Graphs Using Genetic Algorithms. Proc. Heterogeneous Computing Workshop, pp. 86–97

Topcuoglu H, Hariri S, Wu M-Y 2002 Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. on Parallel and Distributed Systems* 13(3): 260–274

Wang L et al 1997 A genetic-algorithm-based approach for task matching and scheduling in heterogeneous computing environments. *J. Parallel and Distributed Computing* 47(1): 8–22