Sādhanā Vol. 37, Part 6, December 2012, pp. 675–694. © Indian Academy of Sciences

Comparison of multi-objective evolutionary approaches for task scheduling in distributed computing systems

G SUBASHINI* and M C BHUVANESWARI

Department of Information Technology, PSG College of Technology, Coimbatore 641 004, India
e-mail: suba@ity.psgtech.ac.in

MS received 5 July 2011; revised 13 June 2012; accepted 21 August 2012

Abstract. Parallel and distributed systems play an important part in the improvement of high performance computing. In these type of systems task scheduling is a key issue in achieving high performance of the system. In general, task scheduling problems have been shown to be NP-hard. As deterministic techniques consume much time in solving the problem, several heuristic methods are attempted in obtaining optimal solutions. This paper presents an application of Elitist Non-dominated Sorting Genetic Algorithm (NSGA-II) and a Non-dominated Sorting Particle Swarm Optimization Algorithm (NSPSO) to schedule independent tasks in a distributed system comprising of heterogeneous processors. The problem is formulated as a multi-objective optimization problem, aiming to obtain schedules achieving minimum makespan and flowtime. The applied algorithms generate Pareto set of global optimal solutions for the considered multi-objective scheduling problem. The algorithms are validated against a set of benchmark instances and the performance of the algorithms evaluated using standard metrics. Experimental results and performance measures infer that NSGA-II produces quality schedules compared to NSPSO.

Keywords. Multi-objective; genetic algorithms; particle swarm optimization; non-dominated sorting.

1. Introduction

Distributed heterogeneous computing systems (DHCS) consist of multiple heterogeneous computing entities or nodes interconnected to solve large computational problems. The efficiency and use of the system depends on the capability of the system to satisfy diverse computational needs of large scale computational problems (Bora *et al* 2006). The computing entities being heterogeneous, execution time of tasks vary on different processors. The available processing entities must be orchestrated to solve complex applications atomized to set of independent subtasks (Maheswaran *et al* 1998). To gain advantage over the large computing capacity of DHCS, scheduling in heterogeneous environments is an important issue. The productivity of the system

*For correspondence

is increased by employing a suitable scheduling method that assigns the tasks to the available processing nodes ensuring good usage of the resources. Scheduling strategies may be static or dynamic. Static scheduling may be useful for analysis of heterogeneous computing systems (Braun *et al* 2001). Static methods can also be employed in high-powered computational grids to distribute computation resources (Foster & Kesselman 2003). Hence the importance of scheduling demands research focus on this issue.

To utilize the system resources effectively, several factors like throughput, resource usage, response time, network traffic and other overheads are to be optimized. To account for its performance the scheduling technique employed in this paper considers minimization of makespan and flowtime (Liu *et al* 2010). Throughput of the system is measured in terms of the makespan and flowtime refers to the response time in execution of the user's request. Since both the requirements contradict with each other this problem requires a multi-objective formulation. There are two general approaches to multiple objective optimization problems (MOOP). One approach involves combining of the multiple objectives into a scalar cost function making the problem single-objective prior to optimization. The other approach is to determine the entire set of Pareto optimal solutions or a subset. In considering real-life problems such solution sets are practically preferable, as a trade-off exists between crucial parameters (Chankong & Haimes 2008).

From a computational complexity perspective, task scheduling on DHCS is computationally hard. Therefore using heuristics to generate schedules at a minimal amount of time is not practically possible. This encourages the research of adapting multi-objective evolutionary computing methods to solve this problem. Recently, Genetic Algorithms (GA) and particle swarm optimization (PSO) appeared as promising evolutionary techniques for handling the task scheduling optimization problem (Salman *et al* 2002; Page & Naughton 2005). In this paper, an elitist non-dominated sorting genetic algorithm and Particle swarm optimization algorithm is applied on the task scheduling problem to generate a set of Pareto optimal solutions. Both algorithms employ the concept of non-dominated sorting. This approach will ensure more non-dominated solutions discovered through the domination comparison operations. The crowding distance calculation enables to maintain the Pareto front size at the chosen limit, without destroying its characteristics. The optimization procedure is applied with the aim of minimizing makespan and flowtime.

The rest of the paper is structured as follows. Section 2 reviews related algorithms for task scheduling problem. The problem formulation is given in section 3. Section 4 describes the application of NSGA-II and and NSPSO for task scheduling. Experimental results are reported in section 5. Finally, in section 6 the conclusions are presented.

2. Previous work

A number of heuristic algorithms are designed to schedule independent tasks in DHCS environments. To optimize selected objectives, each heuristic maps tasks to processors based on certain perception. Unfortunately, the performance of different heuristics differs under various circumstances (Izakian *et al* 2009a; Munir *et al* 2007). Some of the efficient heuristics used for scheduling independent tasks include min–max (Munir *et al* 2007), Sufferage (Maheswaran *et al* 1999), min–min, max–min (Freund *et al* 1998), LJFR-SJFR (Abraham *et al* 2000), etc. Due to the recent developments in metaheuristic optimization theory, several such algorithms are found to be efficient in solving computation intensive problems. The most popular among them being genetic algorithm (Page & Naughton 2005), simulated annealing (Yarkhan & Dongarra 2002), ant colony optimization (Ritchie & Levine 2004) and particle swarm optimization (Salman *et al* 2002). Braun *et al* 2001 described eleven heuristics and compared them on different

types of heterogeneous environments illustrating the performance of GA scheduler in comparison with others. All the above referred heuristics and meta-heuristics aimed at minimizing a single criteria, the makespan of the schedule.

Very few cases illustrate the attempts made in minimizing multiple criteria while scheduling tasks on heterogeneous environments. The method proposed in Liu *et al* 2010 aims at simultaneously minimizing makespan and flowtime. Izakian *et al* (2009a) investigate five popular heuristics for minimizing makespan and flowtime on heterogeneous environments with various characteristics of both machines and tasks. However, the objectives are evaluated separately here. Abraham *et al* (2008) illustrate the usage of several nature inspired meta-heuristics (SA, GA, PSO, and ACO) for scheduling tasks in grids using single and multi-objective optimization approaches. Variants of fuzzy particle swarm optimization for minimizing makespan and flowtime are used in Liu *et al* (2010), Izakian *et al* (2009b). GA-based schedulers are implemented in Xhafa *et al* (2007). However, these methods combine the multiple objectives into a scalar cost function, ultimately making the problem single-objective prior to optimization.

From the previous work, it is concluded that the task scheduling problem considered in this paper is generally formulated as a single-objective optimization problem, transforming several objectives to a single objective by aggregating them. This paper investigates the task scheduling problem as a real multi-objective optimization problem. In doing so, a non-dominated sorting genetic algorithm (NSGA-II) and a non-dominated particle swarm optimization (NSPSO) methods are employed to determine optimal schedules.

3. Problem formulation

A heterogeneous distributed environment is composed of computational units that can be a single personal computer, a group of workstations or a supercomputer. The problem is formally defined as follows. Let P be the set of n processors in the heterogeneous computing system and T be the set of m tasks to be assigned to the processors. The tasks are assumed to be independent of each other and there is no precedence relation among tasks. Each task can only be assigned to a single processor and this is executed exclusively. It is also assumed that pre-emption of task is not allowed. An estimation of the computational load of each task, the computing capacity of each resource, and an estimation of the prior load of each one of the resources is required. Having the computing capacity of the resources and the workload of the tasks, an Expected Time to Compute matrix (ETC) can be built. An ETC matrix is a m, x, n matrix, where each position ETC[m][n] indicates the expected time to compute task m in processor n. One row of the ETC matrix specifies the estimated execution time for a given task on each processor. Similarly, one column of the ETC matrix consists of the estimated execution time of a given processor for each task. A simple example ETC matrix for a system containing 4 tasks and 2 processors is given in table 1.

Table 1. An example ETC matrix.

	Processor 1	Processor 2
Task 1	3	2
Task 2	2	4
Task 3	7	5
Task 4	6	7

An instance of the problem specifies the number of independent tasks that must be scheduled, number of heterogeneous processors in DHCS and the expected time to compute, ETC, where a position ETC[m][n] indicates the expected execution time of task t in processor p. This matrix is explicitly provided.

3.1 *Objectives*

The objectives to be optimized are the makespan and the flowtime, which is defined as follows for an optimal schedule.

$$\text{Minimize F} = (\text{MS},\ \text{FT})\,,$$

where MS the makespan is the time taken to complete the last task and FT is the flowtime, the sum of completion time of all the tasks given by

$$\text{MS} : \min_{S_i \varepsilon \text{Sched}} \left\{ \max_{t \varepsilon \text{Tasks}} \text{Finish}_t \right\} \tag{1}$$

$$\text{FT} : \min_{S_i \varepsilon \text{Sched}} \left\{ \sum_{t \varepsilon \text{Tasks}} \text{Finish}_t \right\}, \tag{2}$$

where *Sched* is the set of all possible schedules, *Tasks* stands for the set of all tasks and $Finish_t$ represents the time in which task t finalizes.

Makespan is an indicator of the general throughput of the system. Small values of makespan mean that the scheduler is providing good and efficient planning of tasks to resources. On the other hand, flowtime refers to the response time to the user requests for task executions. Minimizing the value of flowtime therefore means reducing the average response time of the system. It is essential to maximize the throughput of the system and at the same time offer a quality of service acceptable to the users. To minimize flowtime it requires scheduling of Shortest Job on the Fastest Resource (SJFR) and makespan minimization includes scheduling the Longest Job on the Fastest Resource (LJFR). Minimization of makespan thus results in maximization of flowtime. The relationship between the objectives being contradictory makes the problem multi-objective.

4. Elitist multi-objective evolutionary algorithms for task scheduling

Elitism ensures that the fitness of the best solution in a population does not deteriorate as the generation advances. In fact, using elite parents increases the probability of creating better offsprings. For multi-objective optimization problems, individuals found in the non-dominated front are considered as elites. Pareto-based fitness assignment proposed by Goldberg (1989) uses the non-dominated ranking and selection to move a population to the Pareto front in MOOP. This preserves elitism in the newly found solutions. To maintain diversity among solutions, a crowding distance based selection strategy is employed. The algorithms NSGA-II (Deb *et al* 2002) and NSPSO (Li 2003) implemented in this paper for task scheduling designed on the Pareto based approach are discussed below. The fitness assignment procedure based on non-dominated sorting of individuals in the population and the crowded tournament selection process employed in the implemented algorithm is explained below.

4.1 *Non-dominated sorting*

Non-dominated sorting is used to classify the population to identify the solutions for the next generation. The procedure for sorting is implemented as follows.

Step 1: For each solution p in population N.
Step 2: For each solution q in population N.
Step 3: If q not equal to p

Compare p and q for all 'm' objectives.

Step 4: If for any q, p is dominated by q mark solution p as dominated.

The solutions, not marked, are called non-dominated solutions, which form the first non-dominated front in the population. The process is repeated with the remaining solutions for other higher non-domination fronts until the entire population is classified into u different fronts F_u.

4.2 *Crowded tournament selection*

Selection of individuals is done using Crowded Tournament Selection Operator as given below:

A solution x wins the tournament with another solution y if any of the following conditions are true.

1. If solution i has a better rank, i.e., $R_x < R_y$.
2. If x and y hold the same rank, and solution x has a better crowding distance than solution y, that is $R_x = R_y$ and $cd_x > cd_y$.

The first condition makes sure that the chosen solution lies on a better non-dominated front. The second condition resolves the tie of both the solutions being on the same non-dominated front by deciding on their crowded distance. The one residing in the less crowded area wins, that is, it has a larger crowding distance.

Crowding distance calculation is the determination of Euclidian distance between each individual in a front based on their m objectives in the m dimensional space. The following procedure is used in calculating the crowding distance of the solutions within a front.

Step 1: Initialize the number of solutions in front as $F = n$. initialize the crowding distance of every individual i to zero,

$$\text{ie } F(cd_i) = 0.$$

Step 2: For each objective function $m = 1, 2, \ldots\ldots\ldots\ldots, M$, do
Step 3: Sort the individuals in front F based on objective m, $F = \text{sort}(F, m)$.
Step 4: For $m = 1, 2, \ldots\ldots\ldots\ldots, M$ assign a large distance to the boundary solutions, $F(cd_1) = \infty$ and $F(cd_n) = \infty$.
Step 5: For $i = 2$ to $(n - 1)$

$$F(cd_i) = F(cd_i) + \frac{F(i+1)f_m - F(i-1)f_m}{f_m^{max} - f_m^{min}}$$

$F(i)f_m$ is the value of the m th objective function of the i th individual in F.

4.3 *Non dominated sorting genetic algorithm (NSGA-II)*

The main objective of NSGA-II is to find multiple Pareto-optimal solutions in one single simulation run. This algorithm has been demonstrated as one of the most efficient algorithms for multi-objective optimization on a number of benchmark problems. The advantages of NSGA-II are as follows.

1. It uses non-dominated sorting techniques to provide the solution as close to the pareto-optimal solution as possible.
2. It uses crowding distance techniques to provide diversity in solution.
3. It also uses elitist techniques to preserve the best solution of current population in the next generation.

The steps of NSGA-II algorithm are depicted in figure 1 and summarized below.

4.3a *Algorithm*: (i) A random initial population is generated of size N. (ii) The population produced above is sorted using fast non-dominated sorting till the whole population is classified into various fronts. (iii) Crowding distance assignment is done for each solution and crowded tournament selection is done. This selects a solution at a better rank if the solutions belong to different fronts or a solution with a higher crowding distance if they belong to the same front. (iv) Crossover and mutation is applied to the parent population generated above to produce child population. To create new offspring, simulated binary crossover (SBX) operator and polynomial mutation operators are used. (v) The parent and child population are combined together to produce a population of size 2N. The population updation procedure starts copying individuals into the new population considering first those belonging to the lowest index front as long as the number of individuals in the front does not exceed the population size N. In the last front to be copied, individuals are sorted according to their crowding distance, eliminating those individuals with smaller crowding distance until the total number of individuals N is completed. (vi) Stopping criteria is checked. If the Pareto optimal front is achieved, the algorithms is stopped else go to 2.

4.3b *Representation*: Each individual or chromosome is represented as a vector of length equal to the number of tasks. The values specified in this vector are in the range (1, number of processors) and the value corresponding to each position *n* in the vector represent the processor to which task *n* is allocated. It is to be noted that a processor number can appear more than once in the schedule if both the tasks are assigned to the same processor. Considering the availability of 7 tasks and 3 processors, a schedule representation and the task assignment is as shown in figure 2.

4.3c *Genetic operators*: As the solution is real-coded, the genetic operators Simulated Binary Crossover (SBX) and polynomial mutation (Deb 2001; Deb *et al* 2002) is implemented in this paper.

4.3d *Simulated binary crossover*: The SBX operator works with two parent solutions and creates two offspring. The difference between offspring and parent depends on crossover index η_c, a non-negative real number. A large value of η_c gives a higher probability for creating offspring

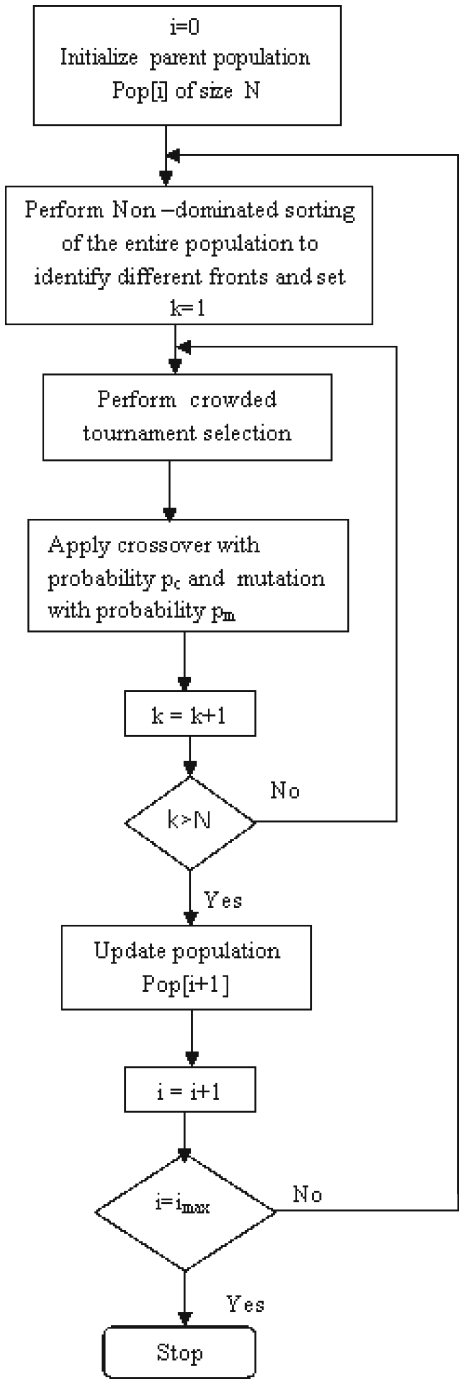


Figure 1. NSGA-II algorithm.

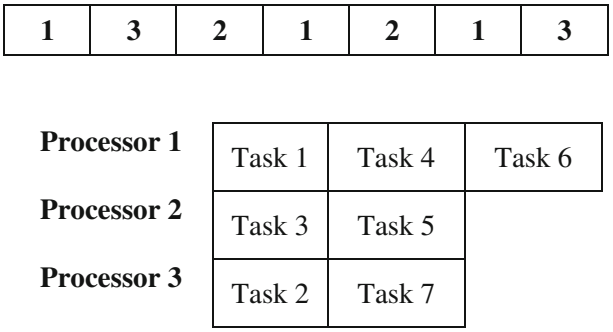


Figure 2. Schedule representation in NSGA-II.

closer to the parent. A small value of η_c allows distant solutions to be selected as offspring. The two offspring solutions created are symmetric about the parent solutions. Also, for a fixed value of η_c the offspring has a spread proportional to that of the parent solutions. It has two properties: (a) the difference between corresponding decision variables of the created offspring is proportional to the difference between corresponding decision variables of the parent solutions; (b) offspring having decision variables closer to the parent solutions are more likely to be selected.

4.3e *Polynomial mutation*: The mutation operator prevents the premature convergence at sub-optimal solutions in GA by restore lost or unexpected genetic material into a population. This ensures a higher probability of reaching all the points in the search space. The polynomial mutation operator has a higher probability of creating a solution closer to the parent than creating one away from it. The shape of the probability distribution is directly controlled by an external parameter, the mutation index η_m and the distribution remains same throughout the iterations.

4.4 *Non-dominated sorting particle swarm optimization (NSPSO)*

The original version of PSO can only be applied to single objective optimization tasks. However, with the adaptation of Pareto-optimal concepts and the non-dominated sorting process, PSO can be used to find the non-dominated solutions in multi-objective optimization effectively. This modified algorithm is non-dominated sorting PSO (NSPSO) (Li 2003). The standard PSO does not effectively utilize valuable non-domination comparisons while the personal best pbest of the particles are updated. To gain advantage of the non-dominance comparisons NSPSO combines the pbest of N particles and the N particles offspring to form a population of 2N particles. These 2N particles are subject to non-domination comparisons for identifying different non-domination fronts. Fitness values are assigned to individuals based on the front they belong to. Individuals that are completely non-dominant in the entire population lie in the first front and given a fitness value of 1. Individuals in the second front dominated by individuals in front one are given a fitness value of 2 and so on. To ensure the distribution of solutions crowding distance is also determined for each particle. The global best $gbest_n$ for the n^{th} particle P_n is selected randomly from the top part of the first front having the highest crowding distance. Based on fitness value and crowding distance, N particles are selected to act as pbest in updating the particle. The steps of basic NSPSO algorithm are shown in figure 3 and summarized below.

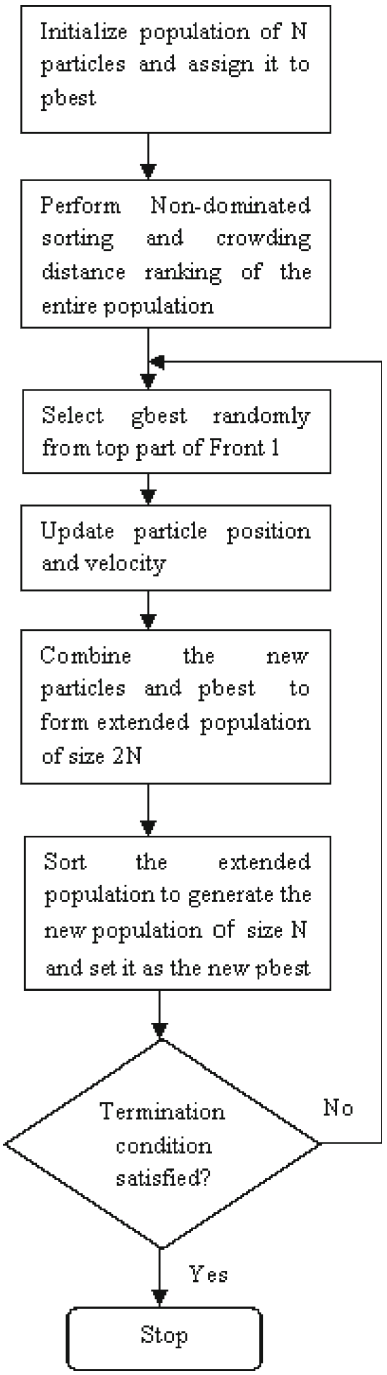


Figure 3. NSPSO algorithm.

4.4a *Algorithm*: (i) Generates an initial population of size N. The position and velocity of each particle in the swarm is initialized randomly within the specified limits. The personal best position of all the particles $Pbest_i$, is set to X_i. (ii) Sort the population based on the non-domination and crowding distance ranking. (iii) Assign each individual a fitness equal to its non-domination level. (iv) Randomly choose one individual as gbest for N times from the non-dominated solutions of the best front. (v) Calculate the new velocity and new position using the determined gbest and pbest. (vi) Create an extended population of size 2N by combining the new position and their pbest. (vii) Sort the extended population based on non-domination and crowding distance ranking. (viii) Fill the new population of size N with individuals from the sorted fronts in the ascending order. These N solutions form the pbest for the next iteration. (ix) Perform steps 4–8 till the stopping criterion is met.

4.4b *Particle representation*: A particle represents a possible solution in the population and dimension n corresponds to n tasks. The Smallest Position Value (SPV) rule is used first to find a permutation corresponding to the continuous position X_i. For the n tasks and m processors scheduling problem, each particle represents a feasible schedule. The position vector $X_k^i = [X_1^i, X_2^i, ..., X_n^i]$ has a continuous set of values. Based on the SPV rule, the continuous position vector can be transformed to a sequence permutation $S_k^i = [s_1^i, S_2^i, ..., S_n^i]$. Then the operation vector $r_k^i = [r_1^i, r_2^i, ..., r_n^i]$ is defined by the following formula: $r_k^i = S_k^i \mod m$. This sequence represents the computing processor number for n tasks. Table 2 illustrates the solution representation of particle *Xi* of NSPSO algorithm for 5 tasks and 3 processors.

4.4c *Particle updation*: During every iteration, the particle position is updated based on personal best and global best experiences. This feedback process increases the probability of targeting the optimal solution. The personal best experience, denoted by $pbest_i$, is the position when the particle X_i has the highest fitness value on flying. gbest represents the best particle found in the entire population of each generation. In each iteration, the particle X_i modifies its velocity V_{ij} and position X_{ij} through each dimension j by referring to $pbest_i$ and the swarm's best experience gbest using (3) and (4).

$$V_{ij} = WV_{ij} + c1rand1\,() \left(pbest_i - X_{ij}\right) + c2rand2\,() \left(gbest - X_{ij}\right) \tag{3}$$

$$X_{ij} = X_{ij} + V_{ij}, \tag{4}$$

c1 and c2 are the cognitive and interaction coefficients, rand1 and rand2 are random real numbers drawn from U(0,1). W, the inertia weight, a user-specified parameter controls the momentum of the particle. A larger inertia weight pushes towards global exploration while a smaller inertia

Table 2. Particle representation in NSPSO.

Dimension	X_k^i	S_k^i	r_k^i
0	2.01	3	0
1	4.96	4	1
2	−0.94	1	1
3	1.87	2	2
4	−1.23	0	0

weight helps in fine-tuning the current search area. The following weighting function is usually utilized:

$$w = w_{\max} - \frac{w_{\max} - w_{\min}}{iter_{total}} * iter_{curr}, \tag{5}$$

where

wmax: initial weight, wmin: final weight,

$\text{iter}_{\text{total}}$: total number of iterations, $\text{iter}_{\text{curr}}$: current iteration number.

5. Experimental set-up and results

The test data in Braun *et al* (2001) based on expected time to compute (ETC) matrix for 512 tasks and 16 processors is simulated for comparing the performances of NSPSO and NSGA-II. In this paper, a set of performance measures is applied to compare the multi-objective evolutionary methods.

5.1 *Data set description*

Firstly, the simulation model based on expected time to compute (ETC) matrix for 512 tasks and 16 processors is generated randomly using the following specifications. Initially, an n × 1 baseline column vector V is generated by repeatedly selecting m uniform random floating point values between 1 and φ_v, the upper bound on values in V. Each value V(i) in V is multiplied by a uniform random number r which has an upper bound of φ_r. Each row in the ETC matrix is then given by V(i) × r. One row requires m different values of r. This process is repeated for each row until the n × m matrix is full. The vector V is not used in the actual matrix. Hence the value in the ETC matrix is within the range (1, φ_v, φ_r).

Attempting to capture realistic aspects of heterogeneous environments, based on three metrics: task heterogeneity, processor heterogeneity and consistency different types of ETC matrices are simulated. The task heterogeneity is defined as the amount of variance possible among the execution times of the tasks with two possible values low and high. Machine heterogeneity is the variation of the execution time of a particular task across all the processors, which can be high and low. High task heterogeneity was represented by $\varphi_v = 3000$ and low task heterogeneity used $\varphi_v = 100$. A value of $\varphi_r = 1000$ models high processor heterogeneity and $\varphi_r = 10$ models low processor heterogeneity. Three different ETC consistencies namely; consistent, inconsistent and semi-consistent are used to model the features of a real heterogeneous system. An ETC matrix is considered consistent if a processor P_i executes task t faster than processor P_j; then P_i executes all the tasks faster than P_j. To model a consistent matrix, each row in the matrix was sorted independently, with processor P_i always being the fastest, and P_j being the slowest. Inconsistency means that a processor is faster for some tasks and slower for some others. Inconsistent matrices are left in the random state in which they are generated. A semi consistent ETC matrix is characterized by an inconsistent matrix which has a consistent sub-matrix of a predefined size. Semi-consistent matrices are generated by extracting the even column elements of each row, sorting them and replacing them while the odd column elements are left the same.

In the results the different problem instances are identified according to the following scheme: x- yy-zz, where x denotes the type of consistency (c–consistent, i–inconsistent and s means semi-consistent).

yy indicates the heterogeneity of the tasks (hi–high, and lo–low).

zz indicates the heterogeneity of the processors (hi–high, and lo–low).

Table 3. Test parameters.

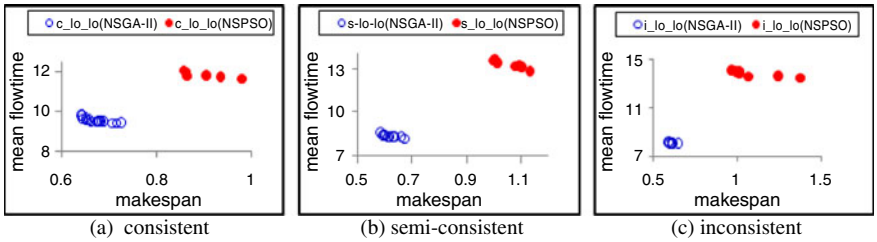
Method	Parameters	Values
NSGA-II	Population size	200
	Number of iteration	1000
	Pc, crossover probability	0.8
	Pm, mutation probability	0.02
	Cross over index η_c	2
	Mutation index η_m	20
NSPSO	Population size	200
	Number of iteration	1000
	C1	2.0
	C2	2.0
	w_{max}	0.9
	w_{min}	0.4

5.2 *Parameter description*

The results of all heuristic techniques are sensitive to parameters. Hence, in setting the parameters, it is required to do extensive simulations to find suitable values for various parameters. In this paper, parameters are determined through preliminary simulation for both the methods and summarized in table 3.

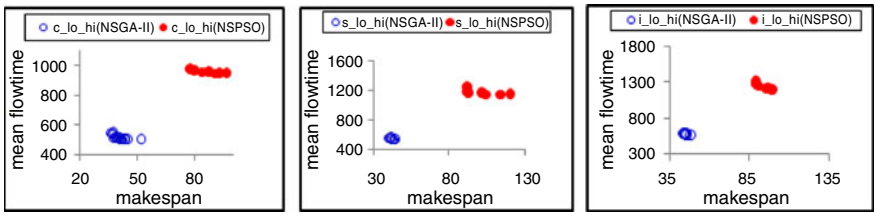
5.3 *Test results*

The algorithms NSGA-II and NSPSO are implemented and applied on all 12 problem instances simulated using the specifications as described in the above section. The test runs use the parameter settings specified in table 3. To compare the performance of multi-objective scheduling algorithms, the extent of minimization of the obtained non-dominated solutions produced by each algorithm for each objective and the spread of their solutions is to be examined. Figures 4, 5, 6 and 7 show the non-dominated solutions obtained at the end of simulation trial for all the instances.



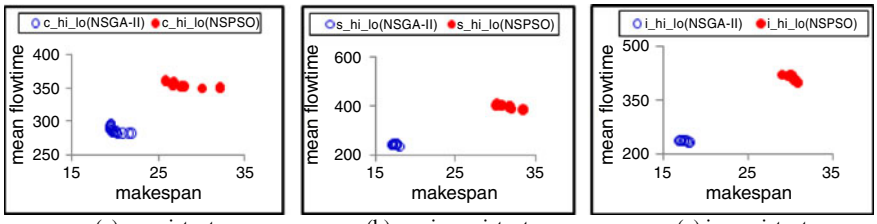
(a) consistent (b) semi-consistent (c) inconsistent

Figure 4. NSGA-II and NSPSO comparison for low task, low machine heterogeneity.



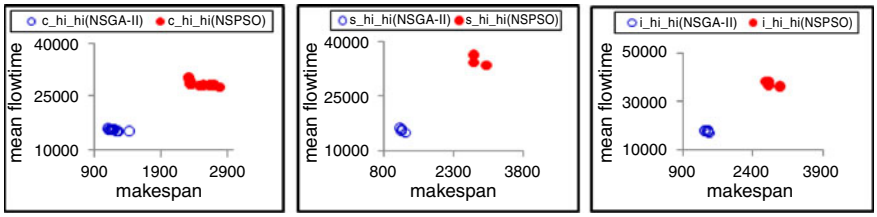
(a) consistent (b) semi-consistent (c) inconsistent

Figure 5. NSGA-II and NSPSO comparison for low task, high machine heterogeneity.



(a) consistent (b) semi-consistent (c) inconsistent

Figure 6. NSGA-II and NSPSO comparison for high task, low machine heterogeneity.



(a) consistent (b) semi-consistent (c) inconsistent

Figure 7. NSGA-II and NSPSO comparison for high task, high machine heterogeneity.

The values of makespan and mean flowtime are measured in same time units and plotted in a scale of ten thousands of measured time unit. However, the plotted graphs indicate that NSGA-II significantly outperforms NSPSO in terms of the minimization of objectives for all cases. As the problem tested comprises of 512 tasks, each schedule is represented by 512 positions. This takes more number of iterations for the algorithms solutions to generate a large set of pareto optimal solutions. However, the algorithm is run for a considerable number of iterations (1000) enabling to find nearly 15 non-dominated solutions. By increasing the population size also, more number of solutions may be found.

From the above, it is found that more number of non-dominated solutions is found for a consistent matrix with both NSGA-II and NSPSO. Comparatively few solutions result for the case of inconsistent and semi-consistent matrix. To get a similar performance it may be required to run the algorithm more number of times for the semi-consistent and inconsistent case.

5.4 *Performance comparisons of GA and NSGA-II*

A single objective GA is also implemented that considers both the objectives separately. The single objective GA uses an initial population of 200 solutions that are randomly generated, a single point crossover and a swap mutation with probability of 0.8 and 0.02, respectively. The algorithm is run for 2000 generations for each of the objectives the makespan and meanflowtime.

Further, to select the best compromise solution from the obtained non-dominated set of solutions a Fuzzy-based approach is applied. The triangular membership function is used to define the fuzzy sets. By taking into account of the minimum and maximum values of each objective function, the k^{th} objective function of a solution in a Pareto set f_k is represented by a membership function μ_k defined as

$$\mu_k = \begin{cases} 1, & f_k \le f_k^{\min}, \\ \dfrac{f_k^{\max} - f_k}{f_k^{\max} - f_k^{\min}}, & f_j^{\min} < f_k < f_k^{\max}, \\ 0, & f_k \ge f_k^{\max} \end{cases}. \tag{6}$$

The value of membership function suggests how far a non-dominated solution has satisfied the f_k objective. The sum of membership function values μ_k for $k = 1, 2, \ldots, m$ objectives can be computed in order to measure the accomplishment of each solution in satisfying the objectives. The accomplishment of each non-dominated solution can be rated with respect to all the N non-dominated solutions by normalizing its accomplishment over the sum of the accomplishments of N non-dominated solutions as follows:

$$\mu_i = \frac{\sum_{k=1}^{m} \mu_k^i}{\sum_{i=1}^{n} \sum_{k=1}^{m} \mu_k^i}, \tag{7}$$

where 'm' is the number of objectives functions and 'n' is the number of solutions. The solution that attains the maximum membership μ_i in the fuzzy set so obtained can be chosen as the best solution. The makespan and meanflowtime values of the solutions thus obtained is given in table 4. The percentage of reduction in makespan and meanflowtime achieved by NSGA-II over GA is also calculated as follows.

$$makespan\ reduction\,(\%) = 1 - \frac{makespan_{NSGA-II}}{makespan_{GA}} \tag{8}$$

$$mean\ \ flowtime\ reduction\,(\%) = 1 - \frac{mean\ \ flowtime_{NSGA-II}}{mean\ \ flowtime_{GA}}. \tag{9}$$

The values of makespan and meanflowtime for the best schedule obtained by the single objective GA and the proposed NSGA-II is given in table 4. The results show that NSGA-II produces quality schedules thereby minimizing both the objectives. NSGA-II achieves a reduction in makespan and flowtime by 23% and 28%, respectively over the values achieved by GA.

Table 4. Comparison of GA and NSGA-II.

	GA		NSGA-II		Reduction in makespan by	Reduction in mean flowtime
Instance	Makespan	Mean flowtime	Makespan	Mean flowtime	NSGA-II (%)	by NSGA-II (%)
c_lo_lo	7581.53	106456.40	6771.35	94980.49	10.68	10.77
c_lo_hi	484886.84	6558902.13	378653.26	5138918.39	21.90	21.64
c_hi_lo	222023.73	3165899.27	214915.50	2843648.47	3.20	10.17
c_hi_hi	13959354.01	192902646.60	12639068.84	154068968.86	9.45	20.13
s_lo_lo	7951.57	111028.67	6749.58	82425.70	15.11	25.76
s_lo_hi	661760.44	8526164.42	460817.50	5282076.98	30.36	38.04
s_hi_lo	242335.20	3329371.79	186560.16	2371762.68	23.01	28.76
s_hi_hi	19255703.63	253897333.21	12855976.47	159959678.5	33.23	36.99
i_lo_lo	8846.40	120342.41	6451.19	79886.54	27.07	33.61
i_lo_hi	804858.26	10256117.40	525548.58	5838798.12	34.70	43.07
i_hi_lo	264065.21	3570443.15	182241.14	2355441.18	30.98	34.02
i_hi_hi	23342636.92	303613189.01	13852895.81	176479328.36	40.65	41.87
Average					23.36	28.74

5.5 *Performance comparisons of PSO and NSPSO*

For analysing the performance of NSPSO, a single objective PSO is also implemented that considers both the objectives separately. The single objective PSO uses a similar parameter setting as that of NSPSO given in table 3 except that the algorithm is run for 2000 generations for each of the objectives the makespan and meanflowtime. The best solution achieved by NSPSO is determined using the Fuzzy-based approach described in section 5.4 and given in table 5.

The results in table 5 also show that NSPSO generates better schedules over PSO but the reduction in makespan and flowtime by NSPSO is only by 1.6% and 1.5%, respectively. This indicates that NSPSO does not show significant improvement in the quality of solutions as that of NSGA-II.

5.6 *Performance metrics*

Finally, this section describes the performance indices used for evaluating the multi-objective heuristics, NSGA-II and NSPSO and the result of solution quality for both the methods. To evaluate the solution set obtained by the heuristics this paper makes use of ratio of non-dominated individuals (RNI), maximum spread and coverage of two sets(C) (Tan *et al* 2002; Zitzler *et al* 2000). They may be described as follows.

5.6a *Ratio of non-dominated individuals (RNI)*: It is always desired to have as many possible candidate solutions known as the Pareto-front from a given population size. The performance measure is denoted here as the ratio of non-dominated individuals *(RNI)* for a given population X and is mathematically formulated as

$$RNI\,(P) = \frac{nondom_{indiv}}{N}, \tag{10}$$

where $\text{nondom}_{\text{indiv}}$ is the number of non-dominated individuals in population P and N is the size of population P. The value RNI = 1 means all the individuals in the population are non-dominated while RNI = 0 represents the situation where none of the individuals in the population are non-dominated.

5.6b *Coverage of two set (C)*: The *Coverage of Two Set (C)* is a measure to compare the domination of two populations in a pair-wise manner, i.e., how good population x dominates population y as well as how good population y dominates population x. This measure is defined as

$$C\left(N, N''\right) = \frac{\left|\left\{a \in N'; \exists a \in N : a \preceq a''\right\}\right|}{|N''|}. \tag{11}$$

The value C(N,N″) = 1 means that all solutions in N are dominated by or equal to solutions in N". C(N,N″) = 0 represents the situation when none of the solutions in N" are covered by the set N.

5.6c *Maximum spread*: The maximum spread is the index for measuring the extent of the search space. It calculates the Euclidean distance between the maximum and the minimum of

Table 5. Comparison of PSO and NSPSO.

	PSO		NSPSO		Reduction in makespan by	Reduction in mean flowtime
Instance	Makespan	Mean flowtime	Makespan	Mean flowtime	NSPSO (%)	by NSPSO (%)
c_lo_lo	9061.29	117340.49	8664.80	117821.78	4.37	−0.41
c_lo_hi	800174.51	9448456.40	779897.71	9403574.69	2.53	0.47
c_hi_lo	280472.76	3508367.36	277143.87	3488426.44	1.18	0.56
c_hi_hi	23717256.9	280113008.31	23572329.52	278368048.63	0.61	0.62
s_lo_lo	10861.21	135950.51	10842.05	135609.86	0.17	0.25
s_lo_hi	1045787.61	11908505.49	1015471.54	11646285.23	2.89	2.20
s_hi_lo	315714.63	4042059.73	308564.65	4017087.20	2.26	0.61
s_hi_hi	28917091.98	356144150.12	28165967.06	348987256.04	2.59	2.00
i_lo_lo	10164.29	137990.51	10141.67	135478.73	0.22	1.82
i_lo_hi	941143.95	13010902.2	932168.49	12759246.05	0.95	1.93
i_hi_lo	302129.04	4181565.18	299918.325	4171033.93	0.73	0.25
i_hi_hi	28445995.1	376070915.9	28165192.5	367546405.23	0.98	2.26
Average					1.62	1.05

Table 6. Performance metrics.

	RNI		Max spread		Set coverage (C)	
Instance	NSGA-II	NSPSO	NSGA-II	NSPSO	NSGA-II over NSPSO	NSPSO over NSGA-II
c_lo_lo	**0.15**	0.06	0.122	**0.125**	1	0
c_lo_hi	**0.17**	0.07	**0.301**	0.194	1	0
c_hi_lo	**0.16**	0.07	**0.185**	0.198	1	0
c_hi_hi	**0.16**	0.12	**0.179**	0.083	1	0
s_lo_lo	**0.1**	0.07	**0.136**	0.130	1	0
s_lo_hi	0.06	**0.07**	0.164	**0.247**	1	0
s_hi_lo	**0.09**	0.08	0.067	**0.116**	1	0
s_hi_hi	**0.05**	0.03	**0.113**	0.110	1	0
i_lo_lo	**0.11**	0.07	**0.087**	0.054	1	0
i_lo_hi	**0.06**	**0.06**	**0.201**	0.144	1	0
i_hi_lo	**0.08**	0.06	**0.276**	0.135	1	0
i_hi_hi	**0.07**	0.05	0.118	**0.125**	1	0

each function. The maximum spread may be written as

$$dist_{spread}=\sqrt{\sum_{i=1}^{n}\left|f_i^{\max}-f_i^{\min}\right|}, \tag{12}$$

where $dist_{spread}$: the maximum Euclidean distance,

$f_i^{\max(\min)}$: max (min) value of the ith objective function.

Greater the value of maximum spread indicates coverage of large search space thereby achieving greater diversity among solutions.

Table 6 shows the result of the solution quality obtained by NSGA-II and NSPSO for the three indices RNI, maximum spread and coverage of two sets. The results obtained infer that the solutions obtained by NSGA-II are of high quality exhibiting enhanced performance in comparison with NSPSO for most of the instances with respect to the above quality measures.

The value of RNI obtained for all the instances indicate that more number of non-dominated solutions are obtained by NSGA-II at the end of simulation run. NSPSO produces equal number of solutions for the i_lo_hi case and more number of solutions in the case of s_lo_hi.

On evaluating the algorithms based on the performance metric Coverage of Two Sets, it is seen that the quality of all the non-dominated solutions obtained by NSGA-II is far-better than that produced by NSPSO. The value of 1 and 0 clearly indicates that all solutions of NSPSO are dominated by NSGA-II.

Table 7. Execution time comparisons.

GA	NSGA-II	PSO	NSPSO
19.33 sec	35.851 sec	18.312	39.525 sec

With respect to the maximum spread, NSGA-II solutions has a higher value for maximum spread in most of the cases which indicates that diversity is preserved. Though NSPSO maintains diversity for few cases, the quality of solutions achieved by NSPSO for those instances is not effective in comparison with the other metrics.

The time taken by the algorithms to obtain the optimal schedule is tabulated in table 7. It is seen that GA and PSO have smaller execution times than NSGA-II and NSPSO, respectively. Since GA and PSO have to be run twice to deal with two objectives, the total running time of both GA and PSO doubles. Moreover, NSGA-II determines the Pareto-optimal set of solutions quickly than NSPSO as NSPSO incurs some time in determining the gbest that is different for each schedule.

6. Conclusion

In distributed computing systems, qualified assignment of tasks among processors is an important step for efficient utilization of resources and execution of the tasks. In this paper, two elitist multi-objective evolutionary algorithms, NSGA-II and NSPSO using Pareto-based fitness assignment is implemented and their performances compared against a standard set of metrics. These algorithms were implemented intending to schedule independent tasks in a DHCS environment of optimizing, makespan and flowtime simultaneously. From the results obtained, it is seen that NSGA-II provides a set of good quality solutions that offer more flexibility to users to estimate their preferences and choose a desired schedule. The results also verify that genetic operators used in NSGA-II help to evolve better solutions rather than controlling the movement of the particles in NSPSO for evolving better solutions. Though PSO is simpler to implement it is found that NSGA-II works well in the case of multi-objective problems. Further work could be extended by hybridizing GA with PSO to make use of better spread of solutions in the search space. Also tasks with different characteristics like dependency among them, pre-emption of tasks could also be considered. The schedules could also be generated dynamically as new tasks arrive into the system.

References

Abraham A, Buyya R and Nath B 2000 Nature's heuristics for scheduling jobs on computational grids, *The 8th IEEE International Conference on Advanced Computing and Communications* (ADCOM 2000), India

Abraham A, Liu H, Grosan C and Xhafa F 2008 Nature inspired meta-heuristics for grid scheduling: *single and multi-objective optimization approaches. Studies in computational intelligence*, Berlin Heidelberg: Springer Verlag, 247–272

Bora U, Cevdet A, Kamer K and Murat I 2006 Task assignment in heterogeneous computing systems, *J. Parallel and Distributed Computing* 66: 32–46

Braun T D, Siegel H J, Beck N, Boloni L L, Maheswaran M, Reuther A I, Robertson J P, Theys M D and Yao B 2001 A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, *J. Parallel and Distributed Computing* 61: 810–837

Chankong V and Haimes Y 2008 *Multiobjective decision making theory and methodology*, Mineola, NY: Dover Publications

Deb K 2001 *Multi-objective optimization using evolutionary algorithms*, Singapore: John Wiley & Sons Ltd

Deb K, Pratap A, Agarwal S. and Meyarivan T 2002 A Fast Elitist Multiobjective Genetic Algorithm: NSGA-II, *IEEE Transactions on Evolutionary Computation* 6: 182–197

Foster I and Kesselman C 2003 *The grid: Blueprint for a new computing infrastructure*, 2nd ed, New York: Morgan Kaufman

Freund R F, Gherrity M, Ambrosius S, Campbell M, Halderman M, Hensgen D, Keith D E, Kidd T, Kussow M, Lima J D, Mirabile F, Moore L, Rust B and Siegel H J 1998 Scheduling resources in multiuser, heterogeneous, computing environments with SmartNet, In: *7th IEEE Heterogeneous Computing Workshop*, 184–199

Goldberg D E 1989 *Genetic algorithms in search, optimization and machine learning*, Boston, MA, USA: Addison-Wesley

Izakian H, Abraham A and Snasel V 2009a Comparison of heuristics for scheduling independent tasks on heterogeneous distributed environments, *IEEE Control Systems Magazine* 1: 8–12

Izakian H, Tork Ladani B, Zamanifar K and Abraham A 2009b A novel particle swarm optimization approach for grid job scheduling, In: *Proceedings of the Third International Conference on Information Systems, Technology and Management*, 100–110

Li X 2003 A non-dominated sorting particle swarm optimizer for multiobjective optimization, In: *Proceedings of Genetic and Evolutionary Computation*, Springer-Verlag Lecture Notes in Computer Science, 2723, 37–48

Liu H, Abraham A and Hassanien A 2010 Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm, *Future Generation Computer Systems* 26(8): 1336–1343

Maheswaran M, Braun T D and Siegel H J 1998 High-performance mixed machine heterogeneous computing, *Sixth Euromicro Workshop on Parallel and Distributed Processing*, 3–9

Maheswaran M, Ali S, Siegel H J, Hensgen D and Freund R F 1999 Dynamic mapping of a class of independent tasks onto heterogeneous computing systems, *J. Parallel and Distributed Computing* 59: 107–131

Munir E U, Li J-Z, Shi S-F and Rasool Q 2007 Performance Analysis of Task Scheduling Heuristics in Grid, In: ICMLC'07: *Proceedings of the International Conference on Machine Learning and Cybernetics*, 6: 3093–3098

Page A and Naughton J 2005 Framework for task scheduling in heterogeneous distributed computing using genetic algorithms, *Artificial Intelligence Rev.* 24: 415–429

Ritchie G and Levine L 2004 A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments, In: *23rd Workshop of the UK Planning and Scheduling Special Interest Group*, PLANSIG 2004

Salman A, Ahmad I and Al-Madani S 2002 Particle swarm optimization for task assignment problem, *Microprocessors and Microsystems* 26(8): 363–371

Tan K C, Lee T H, and Khor E F 2002 Evolutionary Algorithms for Multi-Objective Optimization: Performance Assessments and Comparisons, *Artificial Intelligence Rev.* 17: 251–290

Xhafa F, Carretero J and Abraham A 2007 Genetic algorithm based schedulers for grid computing systems, *Int. J. Innovative Computing, Information and Control* 3: 1053–1071

Yarkhan A and Dongarra J 2002 Experiments with scheduling using simulated annealing in a grid environment, In: *Proceedings of the 3rd International Workshop on Grid Computing* (GRID2002), 232–242

Zitzler E, Deb K and Thiele L 2000 Comparison of multiobjective evolutionary algorithms: empirical results, *Evolutionary Computation* 8: 173–195