Comparison among four Modified Discrete Particle Swarm Optimization for Task Scheduling in Heterogeneous Computing Systems

S.Sarathambekai, K.Umamaheswari

Abstract— ***Task scheduling in heterogeneous multiprocessor systems is an extremely hard NP complete problem. Hence, the heuristic approaches must be used to discover good solutions within a reasonable time. Particle Swarm Optimization (PSO) is a population based new heuristic optimization technique developed from swarm intelligence. This paper presents a Modified Discrete PSO (MDPSO). PSO was originally designed for continuous optimization problems. Some conversion techniques are needed to operate PSO in discrete domain. In Discrete PSO, conversion techniques are not required. Here, the particles are directly represented as integer vectors. The MDPSO extends the basic form of DPSO which incorporates mutation, which is an operator of Genetic Algorithm, for the better diversity of the particles. In this paper, the scheduler aims at minimizing make span, reliability cost and flow time in heterogeneous multiprocessor systems for scheduling of independent tasks using four different MDPSO algorithms. The performance of PSO greatly depends on its control parameters such as inertia weight and acceleration coefficients. Slightly different parameter settings may direct to very different performance. This paper compares the formulation and results of four different MDPSO techniques: constant control parameters, random inertia weight with time varying acceleration coefficients, linearly decreasing inertia weight with time varying acceleration coefficients and constant control parameters with dependent random parameters. Benchmark instances of Expected Time to Complete (ETC) model is used to test the MDPSO. Based on this comparative analysis, MDPSO with linearly decreasing inertia weight provides better results than others.***

***Index Terms*— *Expected Time to Complete, Heterogeneous Multiprocessor systems, Task Scheduling, Particle Swarm Optimization*.**

I. INTRODUCTION

Heterogeneous Computing (HC) systems consist of mixed group of machines, communication protocols and programming environments and offer a diversity of architectural capabilities that has different execution requirements. One of the key challenges of HC system is the task scheduling problem. In general, scheduling is concerned with distribution of limited resources to certain tasks to optimize few performance criterions, like the completion time, waiting time. Task assignment problems can be classified into two categories based on the types of tasks [1]: scheduling a meta-task composed of independent tasks with no data dependencies and assigning an application composed of tasks with precedence constraints.

There are more than a few conflicting objectives in \ multi-objective optimization problems to be optimized and it is tough to identify the greatest solution. For example, a bike manufacturer wish to maximize its turnover, but in the meantime also to minimize its manufacturing cost. These objectives are classically conflicting to each other. A higher turnover would raise the manufacturing cost. There is no single optimal solution. The most traditional approach to solve a multi-objective optimization problem is to summative the objectives into a single objective by using a weighting sum. This paper uses Randomly Assigned Weighted Aggregation (RAWA) method [2] to calculate the fitness value.

Particle Swarm Optimization (PSO) [1, 3] is a population based intellect algorithm proposed by Kennedy and Eberhart [4] in 1995, motivated by the flocking behavior of birds. This has been applied effectively to a number of problems and its use is rapidly increasing. In PSO, each particle is a candidate solution in the search space. All particles have fitness values calculated by a fitness function, and have velocities to direct the flying of the particles. Compared with Genetic Algorithm (GA), PSO has some striking characteristics. It has memory, and the knowledge of good solutions is shared by all particles. In this way, PSO can update its particles' positions according to individuals' memory and swarm's greatest information iteratively. With the collective intelligence of these particles, the swarm can converge to an optimum or near-optimum. PSO has a flexible and well-balanced method to improve and adjust to the global and local exploration and exploitation abilities within a short computation time. These characteristics make PSO highly reasonable to be used for solving single objective and also multi-objective optimization problems.

A significant development in the performance of PSO with the time varying parameters (inertia weight and acceleration coefficients) over the generations was suggested by P K Tripathi [5]. The time varying parameter concepts has been used in previous works e.g., in GA [6], PSO [7, 8] etc, even though the majority of these works dealt with Single Objective Optimization (SOO) problems. This paper slightly modifies the existing DPSO for enhancing the performance of the algorithm to optimize multiple objectives.

The remainder of the paper is organized as follows: The problem statement is given in Section 2. Section 3 reviews related algorithms for task scheduling problem. Section 4 presents the MDPSO algorithm.
Experimental results are reported

Manuscript received on May, 2013.

S.Sarathambekai, Assistant Professor, Dept of IT/ PSG College of Technology/ Coimbatore, Tamilnadu, India.
Dr.K. Umamaheswari, Professor, Dept of IT/ PSG College of Technology/ Coimbatore,Tamilnadu, India

in Section 5. Finally, Section 6 concludes the paper.

II. PROBLEM DEFINITION

A Heterogeneous Computing (HC) system consists of a number of heterogeneous Processor Elements (PEs) connected with a mesh topology. Let T = {T_1, T_2…, T_n} denote the n number of tasks that are independent of each other to be scheduled on m processors P = {P_1, P_2..., P_m}. Because of the heterogeneous nature of the processors and disparate nature of the tasks, the expected execution times of a task executing on different processors are different. Every task has an Expected Time to Compute (ETC) on a specific processor. The ETC values are assumed to be known in advance. An ETC matrix is an *n x m* matrix in which m is the number of processors and n is the number of tasks and. One row of the ETC matrix represents estimated execution time for a specified task on each PE. Similarly one column of the ETC matrix consists of the estimated execution time of a specified PE for each task.

The task scheduling problem is formulated below the following assumptions:

1. All tasks are non-preemptive
2. Every processor processes only one task at a time
3. Every task is processed on one processor at a time.

This paper considers the scheduling of meta-tasks on a set of heterogeneous processors in a way that minimizes the make span, reliability cost and flow time at the same time.

Most popular optimization criterion is minimization of make span [1] i.e. the finishing time of the newest task. Make span computes the throughput of the HC system. Assume that $C_{i,j}$(i ε{1,2,...,n}, j ε {1,2,...,m}) is the execution time for performing ith task in jth processor and W_j (j ε {1,2,...,m}) is the previous workload of P_j. According to the above definition, make span can be estimated as follows:

$$\text{Make span}=\max \{ \sum C_{i,j} + W_j \} \quad j \in \{1, 2, 3\ldots, m) \qquad (1)$$
$$\forall \text{ task i allocated to processor j}$$

Reliability is defined to be the probability that the system will not fail during the time that it is executing the tasks. The Reliability Cost [9, 10] as like a meter of how reliable a given system is when a group of tasks are allocated to it. The lesser the reliability cost raises the reliability. In this model, processor failures are assumed to be independent, and follow a Poisson Process with a constant failure rate. Failures of communication links are not considered here. The reliability cost of a task T_i on a processor P_j is the product of P_j's failure rate(PFR) λ_j and T_i 's execution time on j .Thus, the reliability cost of a task schedule is the summation over all tasks' reliability costs based on the given schedule. According to the above definition, the reliability cost is defined in (2), where $pr(T_i) = j$ indicates that task T_i is allocated to P_j

$$\text{Reliability Cost} = \sum_{j=1}^{m} \sum_{pr(T_i)=j} \lambda_j C_{ij}(T_i) \qquad (2)$$

Flow time [11] is the sum of the finishing times of tasks. Flow time measures the Quality of Service of the HC system. The flow time can be estimated using (3), where $F_{i,j}$ is the finishing time of $task\ T_i$ on a processor P_j

$$\text{Flow time}= \sum_{j=1}^{m} \sum F_{i,j} \qquad (3)$$
$$\forall \text{ task i allocated to processor j}$$

III. RELATED WORK

In general, finding optimal solutions for the task assignment problem in a HC system is NP-complete [12]. Therefore, only small-sized instances of the problem can be solved optimally using precise algorithms. For large scale instances, most researchers have spotlighted on developing heuristic algorithms that give up near-optimal solutions within a reasonable computation time.

Braun [13] explained 11 heuristics for task scheduling and assessed them on different types of heterogeneous computing environments. The 11 heuristics examined are Opportunistic Load Balancing, Minimum Execution Time, Minimum Completion Time, Min_min, Max_min, Duplex, Genetic Algorithm, Simulated Annealing, Genetic Simulated Annealing, Tabu, and A* .The authors illustrated that the Genetic Algorithm can obtain better results in comparison with others. The above stated heuristics aimed at minimizing a single objective, the make span of the schedule.

Izakian [14] recommended an efficient heuristic called min-max for scheduling meta-tasks in heterogeneous computing systems. The effectiveness of proposed algorithm is investigated with 5 popular pure heuristics min-min, max-min, LJFR-SJFR, sufferage, and Work Queue for minimizing make span and flow time. The author also considers the effect of these pure heuristics for initializing Simulated Annealing (SA) meta-heuristic approach for task scheduling on heterogeneous environments.

To achieve a better solution quality, modern meta-heuristics have been commenced for the task scheduling problem such as SA, Tabu Search, GA and Swarm Intelligence (SI).SI consists of two successful techniques of Particle Swarm Optimization (PSO) and Ant Colony Optimization algorithm (ACO). Abraham [15] stated the usage of a number of nature inspired meta-heuristics (SA, GA, PSO, and ACO) for task scheduling in computational grids using single and multi-objective optimization techniques. PSO yields faster convergence when compared to GA, because of the balance between exploration and exploitation in the search space.

The main advantages of PSO algorithm are précised as: simple concept, easy implementation, robustness to control parameters, and computational effectiveness when compared with mathematical algorithm and other heuristic optimization techniques [16].However, these greater characteristics make PSO a highly feasible candidate to be used for solving multi-objective optimization problems. In fact, there have been several recent proposals to extend PSO to handle multi-objectives: The swarm metaphor of Ray and Liew [17], Dynamic neighborhood PSO proposed by Hu and Eberhart [7], the Multi-objective PSO (MOPSO) by Coello and Lechuga [18].

Different criteria can be used for evaluating the effectiveness of scheduling algorithms. So far, some of works have been completed for investigating a number of these heuristics for minimizing make span or make span and flow time, yet no attempts has been made to minimize make span, reliability cost and flow time for scheduling meta tasks on heterogeneous systems using MDPSO.

IV. MODIFIED DISCRETE PARTICLE SWARM OPTIMIZATION

PSO is like to the other evolutionary techniques. The system is initialized with a population of random solutions (particles). The population of the possible particles (solutions) in PSO is called a swarm. Each particle moves in the D-dimensional problem space with a velocity. The velocity is dynamically changed based on the flying knowledge of its own (Personal best) and the knowledge of the swarm (Global best). The velocity of a particle is controlled by three components, namely, inertial momentum, cognitive, and social. The inertial component simulates the inertial behavior of the bird to fly in the previous direction. The cognitive component models the memory of the bird about its previous best position, and the social component models the memory of the bird about the best position among the particles.PSO is different from other evolutionary techniques in a way that it does not apply the filtering operation (such as crossover and/or mutation) and the members of the entire swarm are preserved through the search procedure, so that information is socially shared among particles to direct the search towards the finest position in the search space.PSO can be easily implemented and it is computationally inexpensive, since its memory and CPU speed necessities are low [19].

The movement of the particle towards the best solution is directed by updating its velocity and position characteristics. The velocity and position update for MDPSO are given in (4) and (5)

$$V_i^{(t+1)}(j) = W V_i^t(j) + C_1 r_1 (Pbest_i^t(j) - present_i^t(j)) + C_2 r_2 (Gbest^t(j) - present_i^t(j)) \quad (4)$$

$$present_i^{(t+1)}(j) = V_i^{(t+1)}(j) + present_i^t(j) \quad (5)$$

where i=1, 2, 3…P, j=1, 2, 3…D, P is the number of particles in the swarm, W is the inertia weight which is used to control the impact of the previous history of velocities on the current velocity of a given particle, $V_i^t(j)$ is the *j*th element of the velocity vector of the *i*th particle in *t*th iteration which determines the direction in which a particle needs to move, $present_i^t(j)$ is *j*th element of *i*th particle (solution) in *t*th iteration. r_1 and r_2 are random values in range[0, 1] sampled from a uniform distribution, C_1 and C_2 are positive constants, called acceleration coefficients which control the influence of Personal best (Pbest) and Global best (Gbest) on the search process. Position updating of DPSO is different from classical PSO. Equation (6) shows that the updating the particle position in discrete domain. Each column of position matrix, value 1 is assigned to the element whose corresponding element in velocity matrix has the maximum value in its corresponding column. If there is more than one element with maximum value in a column of velocity matrix, then one of these elements is selected randomly and 1 assigned to its corresponding element in the position matrix. In DPSO, the above (5) is rewritten as follows [11]

$$present_i^{(t+1)}(j) = \begin{cases} 1 \; if \; V_i^{t+1}(j) = max\{V_i^{t+1}(j)\} \; \forall i \in \{1,2,\dots M\} \\ 0 \; otherwise \end{cases} \quad (6)$$

The Fig.1 shows the flow chart for MDPSO and Table 1 shows the pseudo code of the general MDPSO algorithm. Fig. 2 shows the block diagram for MDPSO.

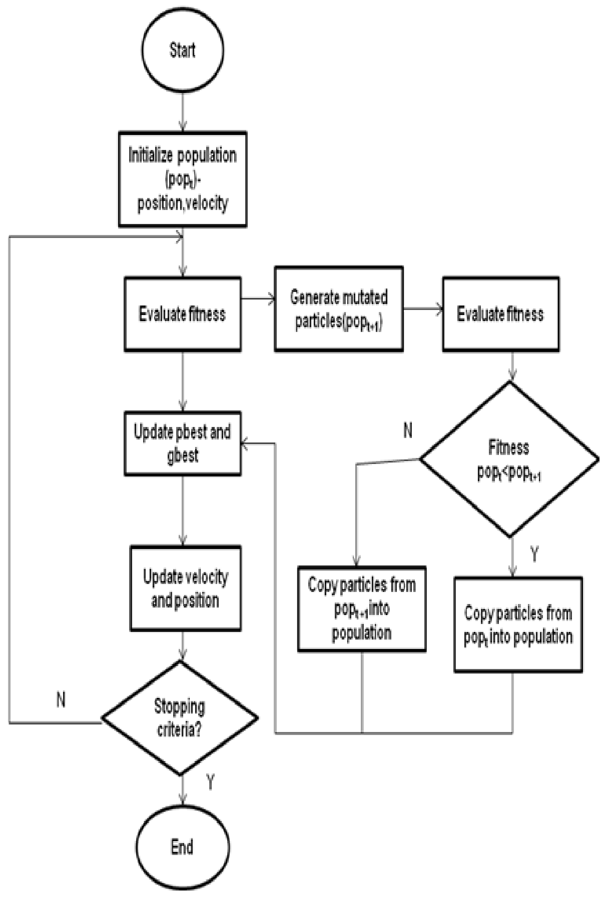

Start
Initialize population (pop$_t$)-position,velocity
Evaluate fitness
Generate mutated particles(pop$_{t+1}$)
Evaluate fitness
N
Fitness pop$_t$<pop$_{t+1}$
Y
Update pbest and gbest
Update velocity and position
Copy particles from pop$_{t+1}$ into population
Copy particles from pop$_t$ into population
N
Stopping criteria?
Y
End

Fig.1. Flow of execution for MDPSO

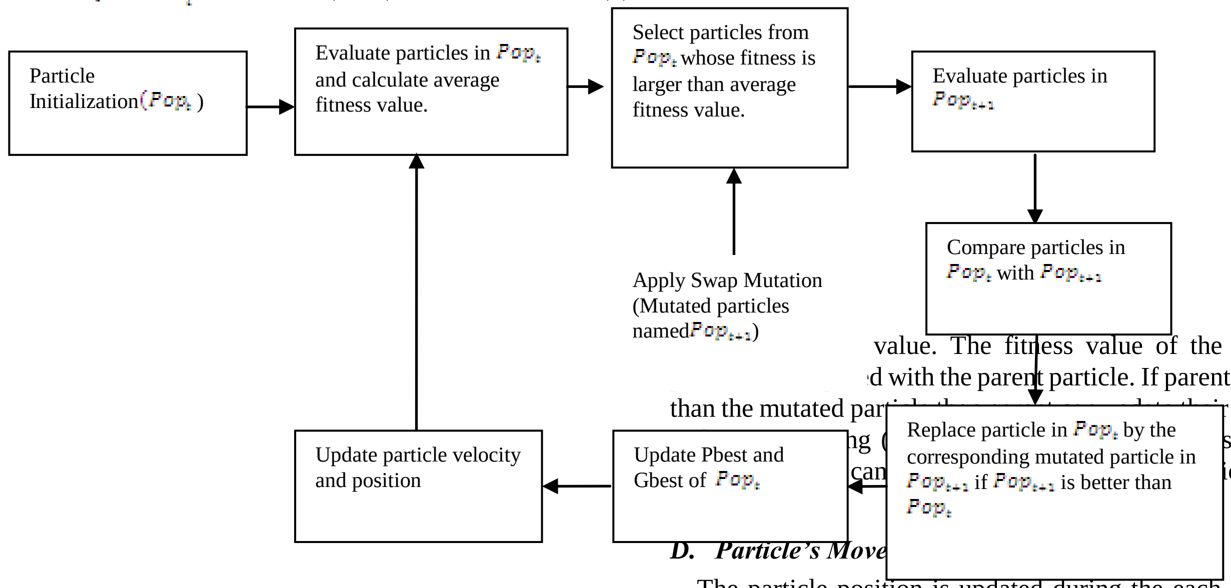
Table 1 Pseudo code of MDPSO

1. Initialize the population.
2. Evaluate the particles.
3. The average of the fitness value is calculated and swap mutation is applied to the particles whose fitness value is greater than the average fitness value. In swap mutation, two positions are randomly selected and are swapped.
4. Compare the fitness values and find the swarm then update Personal best and global best particles.
5. Update velocity and position using (4) and (6)
6. Goto step 2 till the termination condition is met

A. Particle representation and Initialize population

The particles are represented in position vector format in which the elements are integer numbers between 1 and m, where m is the number of processors. The initial population is randomly generated based on the equation (7)

$$present_i = \text{Random}(1, M) \quad (7)$$



Particle Initialization(Pop_t)
Evaluate particles in Pop_t and calculate average fitness value.
Select particles from Pop_t whose fitness is larger than average fitness value.
Evaluate particles in Pop_{t+1}
Compare particles in Pop_t with Pop_{t+1}
Apply Swap Mutation (Mutated particles namedPop_{t+1})
Replace particle in Pop_t by the corresponding mutated particle in Pop_{t+1} if Pop_{t+1} is better than Pop_t
Update Pbest and Gbest of Pop_t
Update particle velocity and position

Fig.2.Block diagram for MDPSO

where Random is a function which generates an integer uniformly distributed in the range [1,M]. Fig.3 shows an illustrative example for a particle which corresponds to a task assignment that assigns four tasks to two processors. Particle 1 (2) = 2 means that task 2 in particle 1 is assigned to processor 2, and so on.

Fig.3.An example for Population

B. Particle Evaluation

The three objectives, make span, reliability cost and flow time are calculated as given by (1), (2) and (3). RAWA is used to calculate the weights for MDPSO. For the RAWA [2], the weights can be generated in the following way:

$$W_1\,(t) = random(\lambda)/\lambda \quad (8)$$

$$W_2\,(t) = (1.0 - W_1\,(t))random(\lambda)/\lambda \quad (9)$$

$$W_3\,(t) = 1.0 - W_1\,(t) - W_2\,(t) \quad (10)$$

The function $fit(swarm)_{sum}$ is a sum of three objectives, the make span, reliability cost and flow time. For three objective functions, the weighted single objective function $fit(swarm)_{sum}$ is obtained as follows:

$$fit(swarm)_{sum} = W_1\ Makespan + W_2\ Reliability\ Cost + W_3\ Flowtime \quad (11)$$

C. Mutation

After finding the fitness value of all the particles, the average of the fitness value is calculated and swap mutation is applied to the particles whose fitness value is greater than the

value. The fitness value of the mutated d with the parent particle. If parent is better than the mutated par velocity ng sponding can ion using

D. Particle's Move

The particle position is updated during the each iteration based on two types of experiences: personal best and global best experiences. The personal best experience (Pbest$_i$) is the experienced position by particle present $_i$ which obtains the smallest fitness value during flying. The gbest represents the best particle found in the entire population of each generation. For each iteration, the particle modifies its velocity and position through each dimension j by referring to $Pbest_i^t$ and the swarm's best experience $Gbest^t$ using (4) and (6)

V. EXPERIMENTAL EVALUATION

The experimental results are attained using a set of benchmark instances [20] for the distributed heterogeneous systems. All algorithms are coded in C and executed on a Linux platform.

A. Benchmark description

The simulation model in [20] based on ETC matrix for 512 tasks and 16 PEs are used. The instances of the benchmark are categorized into 12 types of ETC‘s based on the 3 following metrics: task heterogeneity, machine heterogeneity and consistency are simulated. In this benchmark, quality of the ETC matrices are varied in an attempt to simulate various possible heterogeneous computing environments by setting the values of parameters mean_{task} , V_{task} and $V_{machine}$, which represent the mean task execution time, the task heterogeneity, and the machine heterogeneity respectively. In ETC matrices, the amount of variance among the execution time of tasks in the meta-task for a given processor is defined as task heterogeneity. Machine heterogeneity represents the distinction among the execution times for a given task across all the processors [20]. The Coefficient of Variation Based (CVB) ETC generation method gives a larger control over the spread of execution time values than the common range based method proposed by Braun [13].

The CVB type ETC matrices generation method works as follows. First, a column vector of the expected task execution time with the preferred task heterogeneity, s, is created following gamma distribution with mean mean_{task} and stand deviation $\text{mean}_{task} \times V_{task}$

The input parameter V_{task} is desired coefficient of variation of values in s. The value of V_{task} is high for high task heterogeneity, and small for low task heterogeneity. Each element of s is then used to produce one row of the ETC matrix following gamma distribution with mean q[i] and standard deviation s[i] $\times V_{machine}$ such that the desired coefficient of variation of values in each row is $V_{machine}$. The value of $V_{machine}$ is large for high machine heterogeneity, and small for low machine heterogeneity. Task and machine heterogeneities are modeled by using different V_{task} and $V_{machine}$ values: high heterogeneity is represented by setting V_{task} and $V_{machine}$ equal to 0.6, and low heterogeneity is modeled using V_{task} and $V_{machine}$ equal to 0.1.

TASK \ PARTICLE	1	2	3	4
Particle 1	1	2	1	2
Particle 2	1	1	2	2
Particle 3	1	1	1	2

To capture other possible characteristics of real scheduling problems, three different ETC consistencies namely consistent, inconsistent and semi-consistent are used. An ETC matrix is considered consistent if a processor P_i executes task T_j faster than processor P_j, then P_i executes all the jobs faster than P_j. Inconsistency indicates that a processor is quicker for a few jobs and slower for some others. An ETC matrix is considered semi-consistent if it includes a consistent sub-matrix. A semi consistent ETC matrix is characterized by an inconsistent matrix which has a consistent sub-matrix of a predefined size.

B. Algorithms compared

Simulations were carried out to compare the performance analysis among four different MDPSO.

In first MDPSO, the control parameters W, C_1 and C_2 are constant.

In second Algorithm, the inertia weight called random inertia weight is calculated using the equation (12) and acceleration coefficient C_1 and C_2 are calculated using the equation (13) and (14)

$$W = 0.5 + \text{Rand}()/2 \tag{12}$$

$$C_{1\_t} = \left(C_{1\_final} - C_{1\_initial}\right)\frac{Curr\_iter}{iter\_max} + C_{1\_initial} \tag{13}$$

$$C_{2\_t} = \left(C_{2\_final} - C_{2\_initial}\right)\frac{Curr\_iter}{iter\_max} + C_{2\_initial} \tag{14}$$

where Rand () function generates a random number between 0 and 1 and $iter\_max$ is the maximum number of iterations and $Curr\_iter$ is the current iteration number. Larger values of C_1 guarantee larger deviation of the particle in the search space, while the larger values of C_2 signify the convergence to the present global best (gbest). C_1 has been permitted to reduce from its initial value of C_{1_final} *to* $C_{1\_initial}$ while C_2 has been raised from C_{2_final} *to* $C_{2\_initial}$.

The Linearly Decreasing inertia Weight (LDW) is used in third Algorithm of MDPSO. The LDW is calculated using the equation (15) and acceleration coefficient C_1 and C_2 are calculated using the equation (13) and (14)

$$W_t = (W_{max} - W_{min})\frac{Curr\_iter}{iter\_max} + W_{min} \tag{15}$$

This adaptiveness permits to attain a good balance between the exploration and the exploitation of the search space. The inertia weight (W) is used to balance the global and local search abilities. A high inertia weight is more suitable for global search and a small inertia weight helps local search [21]. Typically, this algorithm started with a large inertia weight, which is decreased over time. The value of W_t is permitted to reduce linearly with iteration from W_{max} to W_{min}.

In fourth Algorithm, the control parameters W, C_1 and C_2 are constant. The two random parameters r_1 and r_2 of (4) are independent. If the two random parameters are high, both the personal and social experiences are over used and the particle is moved too far away from the local optimum. If both are low, both the personal and social experiences are not used fully and the convergence speed of the optimization technique is reduced. The fourth MDPSO creates a dependency between two random parameters r_1 and r_2 of (4) to control the balance of personal and social experiences. Instead of taking independent r_1 and r_2, one single random number r_1 is chosen so that when r_1 is large, $1 - r_1$ is small and vice versa. The (4) is rewritten as follows

$$V_i^{(t+1)}(j) = \chi\,[V_i^t(j) + C_1 r_1 (Pbest_i^t(j) - present_i^t(j)) + C_2(1 - r_1)(Gbest^t(j) - present_i^t(j))] \tag{16}$$

All the four algorithms are stochastic based algorithms.

Type of heterogeneity	Alg 1	Alg 2	Alg 3	Alg 4
i_hi_hi	6699.802	6699.802	**6689.802**	8532.202
i_hi_lo	**3601.311**	3790.541	3798.071	4646.161
i_lo_hi	8591.672	8591.672	**8581.672**	9410.282
i_lo_lo	6190.731	**6112.140**	6190.731	7529.961
s_hi_hi	4168.201	**4076.561**	4168.201	5360.801
s_hi_lo	3653.271	3653.271	**3643.271**	4502.801
s_lo_hi	6970.892	6970.892	**6960.892**	8707.761
s_lo_lo	**5945.452**	5986.451	5982.541	7401.602
c_hi_hi	3647.441	3973.801	**3554.801**	4584.800
c_hi_lo	3550.381	3550.381	**3545.381**	4411.321
c_lo_hi	3329.491	**3304.121**	3329.491	4037.080
c_lo_lo	5789.401	5892.992	**5782.992**	7295.601

Each independent run of the same algorithm on a particular problem instance may yield a different result. To make a good comparison of the algorithms each experiment was repeated 5 times with different random seeds and the average of the results are reported.

C. Parameter setup

- Population Size (N)=30 and Number of iteration =25 for all the algorithms.
- The Failure rate for each processor is uniformly distributed [10, 11] in the range from 0.95×10^{-6} /h to 1.05×10^{-6}/h .
- The MDPSO with constant control parameters set fixed inertia weight as W=0.8, C_1 = 1 and C_2=1 during the whole run of the algorithm
- Values for time varying inertia weight and acceleration coefficients: inertia weights W_{max} and W_{min} (0.8 and 0.4), the initial acceleration coefficients $C_{1\_initial}$ = 2.5, C_{1_final} = 0.5, $C_{2\_initial}$ = 0.5, C_{2_final} = 2.5. C_1 has been allowed to decrease from its initial value of 2.5 to 0.5, while C_2 has been increased from 0.5 to 2.5

D. Performance comparisons

To make the comparison fair, the swarms for all the methods were initialized using the same random seeds. All instances consisting of 20 tasks and 2 or 3 processors are classified into 12 different types of ETC matrices according to the 3 metrics. All the algorithms are applied on all 12 problem instances and the results plotted from Fig.4 to Fig.15.

Table 2 Comparison of four algorithms for 2 processors

The instances are labeled as g_a_bb_cc as follows:

- g means gamma distribution used in generating the matrices.
- a shows the type of inconsistency; c means consistent, i means inconsistent, and s means semi-consistent.
- bb indicates the heterogeneity of the tasks; hi means high and lo means low.
- cc represents the heterogeneity of the machines; hi means high and lo means low.

The table 2 shows the comparison of four MDPSO algorithms for 2 processors. From the results obtained, Algorithm 3 MDPSO is found to be the best for 2 processors which uses linearly decreasing inertia weights and varying acceleration coefficients.

The Table 3 shows the comparison of four MDPSO algorithms for 3 processors. From the results obtained, Algorithm 3 MDPSO is found to be the best for 3 processors which uses linearly decreasing inertia weights and varying acceleration coefficients. Thus the Algorithm 3 MDPSO is found to be efficient for both 2 and 3 processors. In table 2 and 3, Alg represents Algorithm.

Fig 4 shows the comparison of fitness values for high Task, high machine and inconsistent type. From the results obtained, Algorithm 1, 2 have the same fitness values. The fitness value of algorithm 3 is better than others. The fitness values for 3 processors are better than 2 processors. In Fig 5, Algorithm 1 has the smallest fitness value for 2 processors and Algorithm 2 has the smallest fitness value for 3 processors. In Fig 6, Algorithm 3 has the smallest fitness value. The fitness values for 3 processors are better than 2 processors.

Type of heterogeneity	Alg 1	Alg 2	Alg 3	Alg 4
i_hi_hi	4121.791	4121.791	**4120.091**	5131.361
i_hi_lo	2916.581	**2911.901**	2916.581	3420.481
i_lo_hi	4645.811	4645.811	**4643.081**	5478.441
i_lo_lo	4635.302	4635.302	**4634.010**	5715.602
s_hi_hi	3655.841	3655.841	**3652.041**	4638.761
s_hi_lo	**2673.611**	2686.301	2673.881	3244.441
s_lo_hi	**5168.711**	**5168.711**	**5168.711**	5874.441
s_lo_lo	**4668.992**	4673.091	4673.091	5869.642
c_hi_hi	3132.291	3132.291	**2910.591**	3776.601
c_hi_lo	2767.841	3792.371	**2763.041**	3489.081
c_lo_hi	3342.020	**3265.180**	3413.281	5113.001
c_lo_lo	**4627.662**	4683.182	4683.182	5863.722

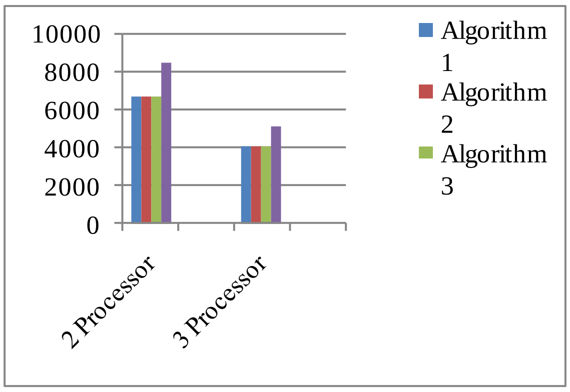


Fig.4. Comparison of fitness values for High Task-High Machine-Inconsistent

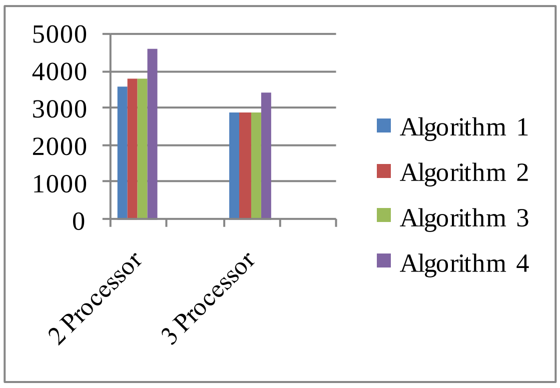


Fig.5. Comparison of fitness values for High Task-Low Machine-Inconsistent

Table 3 Comparison of four algorithms for 3 processors

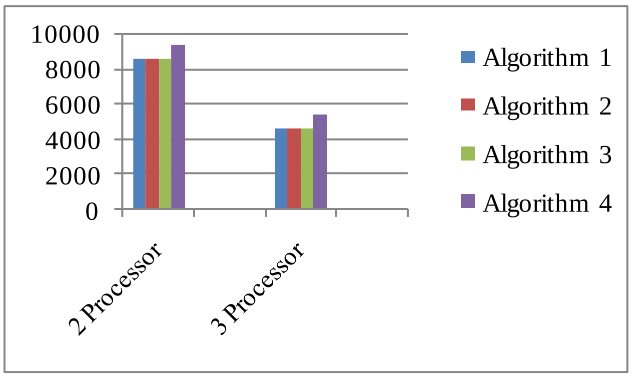


Fig.6. Comparison of fitness values for Low Task-High Machine-Inconsistent

In Fig 7 and 8, Algorithm 2 has the smallest value for 2 processors and the Algorithm 3 has smallest fitness value for 3 processors.

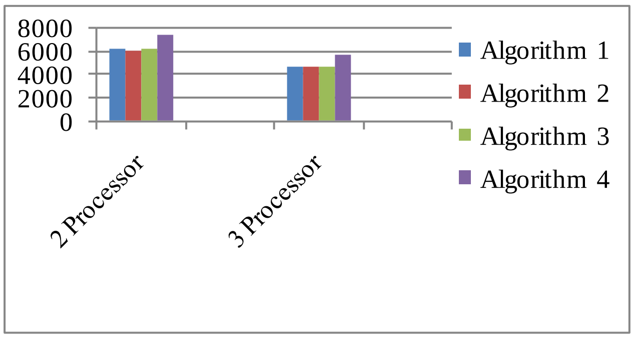


Fig.7. Comparison of fitness values for Low Task-Low Machine-Inconsistent

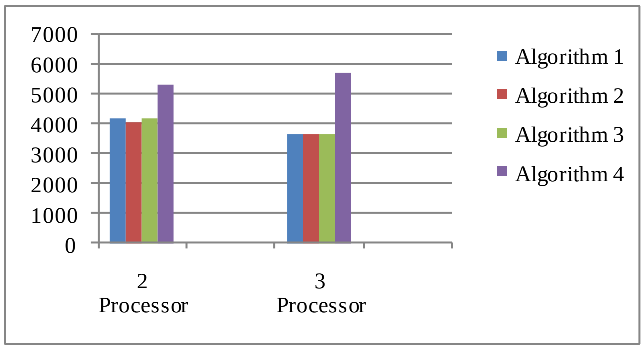


Fig.8. Comparison of fitness values for High Task-High Machine-Semi consistent

In Fig 9, Algorithm 3 has the smallest fitness value for 2 processors and Algorithm 1, 2 and 3 have the same values for 3 processors.

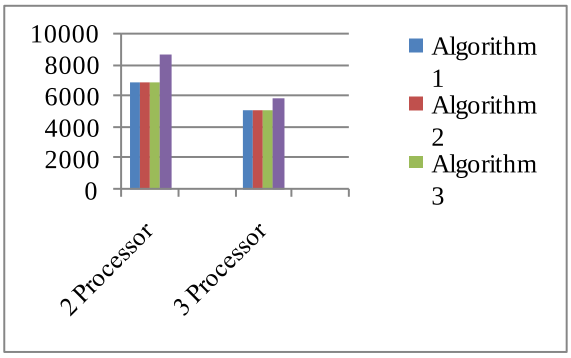


Fig.9.Comparison of fitness values for Low Task-High Machine-Semi consistent

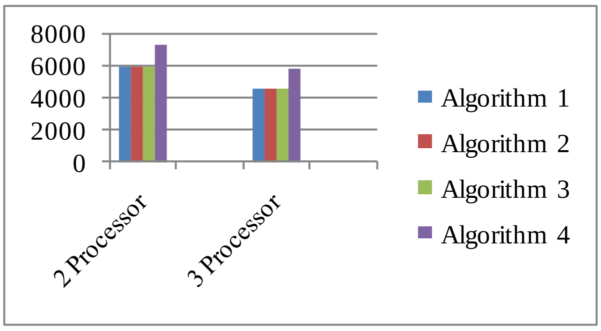


Fig.10.Comparison of fitness values for Low Task-Low Machine-Semi consistent

In Fig 10, Algorithm 1 has the smallest value for 2 processors and 3 processors.

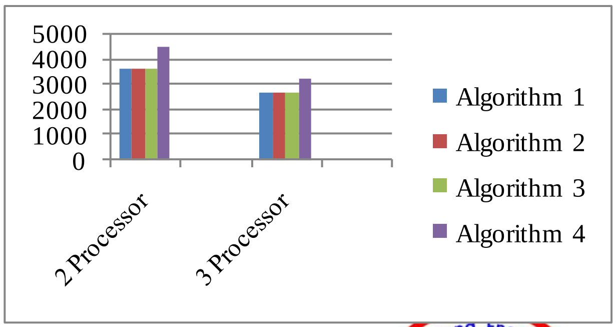


Fig.11. Comparison of fitness

values for High Task-Low Machine-Semi consistent

In Fig 11, Algorithms 3 has the smallest fitness value for 2 processors and Algorithm 1 has the smallest fitness value for 3 processors. In Fig 12, Algorithm 3 has the smallest fitness value for both 2 and 3 processors.

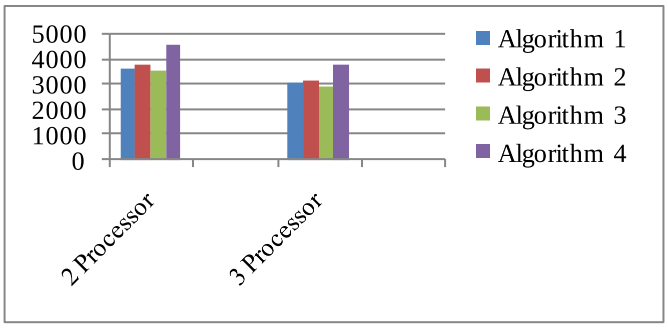
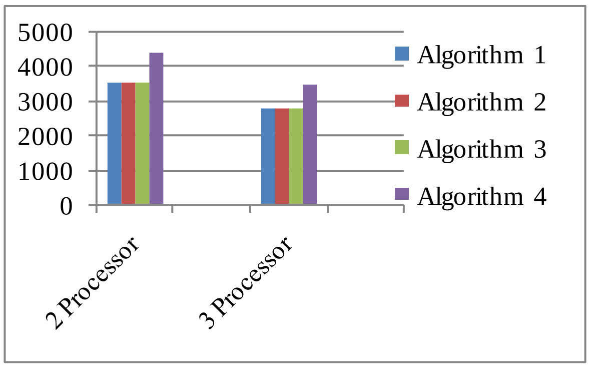


Fig.12. Comparison of fitness values for High Task-High Machine-Consistent



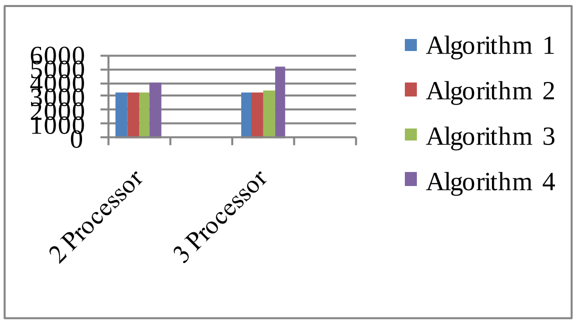


Fig.14. Comparison of fitness values for Low Task-High Machine-Consistent

In Fig 14, Algorithm 2 has the smallest fitness value for 2 processors and 3 processors. In Fig 15, Algorithm 3 has the smallest fitness value for 2 processors and Algorithm 1 has the smallest fitness value for 3 processors.

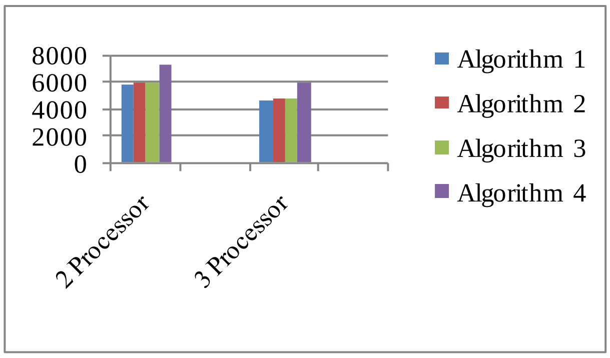


Fig.15. Comparison of fitness values for Low Task-Low Machine-Consistent

From the results obtained, algorithm 3 is found to be the best for 2 processors and 3 processors which uses linearly decreasing inertia weights and varying acceleration coefficients. Thus the Algorithm 3 is found to be efficient among others.

VI. CONCLUSION AND FUTURE WORK

This paper has presented four different modified DPSO algorithms. Modified Discrete PSO algorithm has been successfully applied for scheduling of independent tasks in a heterogeneous environment. Four different modified DPSO algorithms were developed for minimizing the objectives like make span, reliability cost and flow time. All the algorithms were compared with all the 12 different types of ETC matrices. From the various set of results obtained, Algorithm 3, which has a linearly decreasing inertia weight, is found to be efficient than other algorithms which have random inertia weight and constant inertia weights. The algorithm 3 is found to be more efficient when tasks are being scheduled in 3 processors.

The future work will investigate scheduling tasks with precedence constraint which are pre-emptive in nature or in dynamic environments.

REFERENCES

[1] Qinma Kang, and Hong He,"A novel discrete particle swarm optimization algorithm for meta-task assignment in heterogeneous computing systems", *Microprocessors and Microsystems, Elsevier*,pp 10–17,(2011)

[2] Yaochu Jin, Tatsuya Okabe and Bernhard Sendhoff ,"Solving three objective optimization problems using Evolutionary dynamic weighted aggregation: Results and Analysis", *Genetic and Evolutionary Computation Conference*. Springer, Berlin, pages 636—637(2003)

[3] G. Subashini, M.C. Bhuvaneswari, "Non Dominated Particle Swarm Optimization For Scheduling Independent Tasks On Heterogeneous Distributed Environments", *International Journal of Advance. Soft Computing Appl*., Vol. 3, No. 1, ISSN 2074-8523(2011)

[4] Kennedy,J, and Eberhart.R ,"Particle swarm optimization", *In proceeding of the fourth IEEE International conference on Neural Networks*,Perth,Australia.IEEE Service Center(1995)

[5] Praveen Kumar Tripathi , Sanghamitra Bandyopadhyay, and Sankar Kumar Pal, " Multi-Objective Particle Swarm Optimization with time variant inertia and acceleration coefficients", *Elsevier,International journal of Information Sciences*(2007)

[6] S.K. Pal, S. Bandyopadhyay, C.A. Murthy, "Genetic algorithms for generation of class boundaries", *IEEE Transaction on Systems Man and Cybernetics – Part B: Cybernetics,pp* 816–828(1998)

[7] A. Carlisle and G. Dozier, "Adapting particle swarm optimization to dynamic environments", *International Conference on Artificial Intelligence* , Las Vegas, Nevada, USA, pp. 429–434(2000)

[8] A. Carlisle and G. Dozier, "Tracking changing extrema with adaptive particle swarm optimizer", *5th Biannual World Automation Congress*, Orlando, Florida, USA, pp. 265–270(2000)

[9] Wei Luo, Xiao Qin, and Kiranmai Bellamssss,"Reliability-Driven Scheduling of Periodic Tasks in Heterogeneous Real-Time Systems", *IEEE International Symposium on Embedded Computing,*Ontario, Canada(2007)

[10] Xiao Qin,Hong Jiang,"Dynamic, Reliability-driven Scheduling of Parallel Real-time Jobs in Heterogeneous Systems", *IEEE International conference on Parallel Processing*,pp 113-122(2001)

[11] Hesam Izakian, Behrouz Tork Ladani, Ajith Abraham,Vaclav Snasel, “A Discrete Particle Swarm Optimization Approach For Grid Job Scheduling”, *International Journal of Innovative Computing, Information and Control ICIC International*, ISSN 1349-4198 Volume 6, Number 9(2010)

[12] D. Fernandez-Baca, “Allocating modules to processors in a distributed system”, *IEEE Trans. Software Eng*. 15 (11) -1427–1436(1989)

[13] H.J. Braun et al, “A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems”, *Journal of Parallel and Distributed Computing*, Vol 61, No.6, (2001).

[14] H. Izakian, A. Abraham and V. Snasel, “Performance comparison of six efficient pure heuristics for scheduling meta-tasks on heterogeneous distributed environments”, *Neural Network World*, vol.19,no.6, pp.695-710(2009)

[15] A. Abraham, H . Liu, C. Grosan, F. Xhafa, “Nature inspired meta-heuristics for grid scheduling: single and multi-objective optimization approaches”, *Studies in Computational Intelligence*, Springer Verlag: Heidelberg, Germany, pp. 247–272(2008)

[16] J. Park, K. Lee, J. Shin, and K. Y. Lee, "A Particle Swarm Optimization for Economic Dispatch with Nonsmooth Cost Function", *IEEE Trans. on Power Systems*, Vol. 20, No.1, pp. 34-42, (2005)

[17] T. Ray, and K. Liew, "A swarm metaphor for multiobjective design optimization", *Engineering Optimization*, Vol. 34, pp. 141-153(2002).

[18] C.A. Coello Coello, and M. Salazar Lechuga, "MOPSO: A proposal for multiple objective particle swarm optimization", *Congress on Evolutionary Computation IEEE Service Center*, Piscataway, New Jersey, pp. 1051-1056(2002)

[19] Ozgur Uysal1 and Serol Bulkan2, “Comparison Of Genetic Algorithm And Particle Swarm Optimization for Bicriteria Permutation Flowshop Scheduling Problem”, *International Journal of Computational Intelligence Research*, ISSN 0973-1873 Vol.4, No.2, pp.159–175 (2008)

[20] S. Ali, H.J. Siegel, “Representing task and machine heterogeneities for heterogeneous computing systems”, *Tamkang Journal of Science and Engineering*, Vol. 3, No. 3, pp. 195-207 (2000)

[21] Kaushik Suresh, Sayan Ghosh, Debarati Kundu, Abhirup Sen, Swagatam Das and Ajith Abraham, “Inertia-Adaptive Particle Swarm Optimizer for Improved Global Search”, *IEEE International conference on Intelligent systems and design*,pp 253-258(2008)