Computing and Informatics, Vol. 36, 2017, 950–970, doi: 10.4149/cai_2017_4_950

TASK SCHEDULING USING HAMMING PARTICLE SWARM OPTIMIZATION IN DISTRIBUTED SYSTEMS

Subramaniam Sarathambekai, Kandaswamy Umamaheswari

Department of Information Technology
PSG College of Technology, Coimbatore, Tamil Nadu, India
e-mail: vrs070708@gmail.com, uma@ity.psgtech.ac.in

Abstract. An efficient allocation of tasks to the processors is a crucial problem in heterogeneous computing systems. Finding an optimal schedule for such an environment is an NP-complete problem. Near optimal solutions are obtained within a finite duration using heuristics/meta-heuristics are used instead of exact optimization methods. Heuristics and meta-heuristics are the efficient technologies for scheduling tasks in distributed environment because of their ability to deliver high quality solutions in a reasonable time. Discrete Particle Swarm Optimization (DPSO) is a newly developed meta-heuristic computation technique. To enhance the final accuracy and improve the convergence speed of DPSO, this paper presents a modified DPSO algorithm by adjusting its inertia weight based on Hamming distance and also makes a dependency between the two random parameters r_1 and r_2 to control the balance of individual's and collective information in the velocity updating equation. Three criteria such as make span, mean flow time and reliability cost are used to assess the efficiency of the proposed DPSO algorithm for scheduling independent tasks on heterogeneous computing systems. Computational simulations are performed based on a set of benchmark instances to evaluate the performance of the proposed DPSO algorithm compared to existing methods.

Keywords: Distributed system, heuristics, inertia weight, meta-heuristic, task scheduling, particle swarm optimization

Mathematics Subject Classification 2010: 68M14, 68M20

1 INTRODUCTION

Unlike homogeneous systems, Heterogeneous Systems (HS) are composed of computers with different speeds [1]. Task Scheduling (TS) is one of the key challenges in HS. It can be classified into two categories based on the types of tasks [2]: tasks with no data dependencies (independent task) and tasks with precedence constraints (dependent task). In this paper, the scheduler schedules only the meta-task in HS. Meta-tasks composing of independent tasks occur in many circumstances. For example, all of the jobs submitted to a super computer-center by different users constitute a meta-task and another example of a meta-task is a group of image processing applications all operating on different images [2].

The HS may become larger and larger because of its scalability nature. As the numbers of interconnected heterogeneous resources are growing enormously, the need for an algorithmic solution for efficient use of such platform is growing as well. Different metrics may be used to evaluate the effectiveness of scheduling algorithms, such as make span, flow time, resource utilization [3, 4]. All the existing works investigated a number of scheduling algorithms for minimizing make span or flow time. The issue of reliability for such an environment needs to be addressed or else an application running on a very large system may crash because of the hardware failure. The previous research work [5] evaluated the scheduler with the reliability cost using DPSO algorithm. To enhance the reliability of the HS, this paper presents a modified DPSO algorithm.

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. Kim et al. presented an Adaptive Weighted Sum (AWS) method for multi-objective optimization problems [6]. The author demonstrated that the AWS method produces a well-distributed Pareto front and also finds solutions in non-convex regions. This paper presents the fitness value of each solution using AWS method that was proposed by Kim et al. [6].

Heuristics are one of the suitable approaches to solve the TS problem in HS. Some research works have been made in recent years using pure heuristics to discover near-optimal solutions [7]. These heuristics are fast, simple and easy to implement. Also, to improve the quality of solutions, meta-heuristics have been presented for the TS problem. The most popular meta-heuristic algorithms in the literature are Genetic Algorithm (GA), Differential Evolution (DE), Simulated Annealing (SA), Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO) [7].

PSO is a meta-heuristic algorithm proposed by Kennedy et al. [1] in 1995, motivated by the flocking behavior of birds. This has been applied in wide area in different fields such as engineering, physics, mathematics and chemistry. The performance of PSO greatly depends on its control parameters such as inertia weight. Slightly different parameter settings may direct to different performance in the algorithm. Kaushik et al. [8] proposed an adaptive inertia weight in continuous domain, which is calculated based on the Euclidean distance of the particles of a particular generation from the global best. This kind of the inertia factor ensures that the particles have not moved away from the global best. This paper makes the adaptive inertia weight more suitable for discrete domain. The proposed inertia weight is calculated based on the Hamming distance of the particles from global best.

The rest of the paper is organized as follows: Section 2 reviews the existing algorithms for task scheduling problem. The proposed DPSO is presented in Section 3. Experimental results are reported in Section 4. Finally, Section 5 concludes the paper.

2 RELATED WORK

Finding an optimal solution for the TS problem in a Heterogeneous Computing (HC) system is NP-hard. The precise algorithms can optimally solve the small-sized instances of the problems. For large scale instances, most of the researchers have spotlighted on developing heuristic algorithms that give up near-optimal solutions within a reasonable computation time.

Braun et al. [2] clarified 11 heuristics such as Opportunistic Load Balancing, Minimum Execution Time, Minimum Completion Time, Min-min, Max-min, Duplex, Genetic Algorithm, Simulated Annealing, Tabu, and A* for the TS problem and assessed them on different types of heterogeneous environments. These heuristics were evaluated by a single objective, the make span of the schedule. The authors illustrated that the Genetic Algorithm can obtain better results in comparison with 11 heuristics. Izakian et al. [9] recommended an efficient heuristic called min-max for scheduling meta-tasks in HS. The effectiveness of the recommended min-max algorithm was investigated with five popular pure heuristics min-min, max-min, LJFR-SJFR, suffrage, and work queue for minimizing make span and flow time.

To achieve a better solution quality, meta-heuristics have been commenced for the TS problem such as SA, Tabu Search, GA and Swarm Intelligence (SI). SI consists of two successful techniques: PSO and ACO. Abraham et al. [10] stated the usage of a number of nature inspired meta-heuristics (SA, GA, PSO, and ACO) for TS in computational grids using single and multi-objective optimization techniques. PSO yields faster convergence when compared with GA, because it has an inertia factor for providing balance between exploration and exploitation in the search space.

Kennedy et al. [1] developed PSO with no inertia weight. Shi et al. [11] presented the concept of inertia weight with constant value. Further, many researchers introduced dynamical adjusting of inertia weight that can increase the capabilities of PSO. Xin et al. [12] presented Linearly Decreasing Inertia weight (LDI) for enhancing the efficiency and performance of PSO. Bansal et al. [13] presented a comparative study on 15 strategies to set inertia weight in PSO. The four different modified DPSO variants based on changing the value of inertia weight were addressed in TS problem [14].

Hesam et al. [4] proposed DPSO approach for grid job scheduling problem to minimize make span and flow time. In this paper, the author redefined the velocity and position updating equations for both direct particle representation (position vector) and indirect particle representation (position matrix). Kang et al. [3] developed a new position update method for a particle that is represented in position vector format in DPSO for scheduling of meta-tasks in HS to minimize make span of the schedule. Further, the author applied variable neighborhood descent algorithm and migration mechanism to escape DPSO from local optimum and to balance the exploration and exploitation.

The previous research work [5] presented DPSO approach for scheduling problem to minimize make span, flow time and reliability cost. The make span and flow time values are in incomparable ranges and the flow time has a higher magnitude order over the make span. DPSO is tested with small data set such as twenty tasks and the number of processors is limited to two and three. Proposed work in this paper extends DPSO for scheduling of independent tasks to minimize make span, mean flow time and reliability cost using modified DPSO which also addresses the above stated problem. The modified DPSO incorporates Hamming inertia weight and dependent random parameters into DPSO, and proposes Hamming inertia weight with Dependent random parameters DPSO (HDDPSO) for scheduling independent tasks in HS. The HDDPSO is evaluated with benchmark ETC instances of 512 tasks and 16 processors.

3 THE PROPOSED DPSO ALGORITHM

The proposed work emphasizes the significance of representing the particles in Permutation Based Format (PBF) which conveys the flow of execution of tasks on the processors for optimizing flow time. Representation of a particle is one of the key issues in devising a successful PSO algorithm in discrete problems, since, the particles convey the essential information related to the problem domain. PSO is a prominent stochastic search method on continuous optimization problems because position of the particles (solutions) is real value. Scheduling task in distributed systems is discrete optimization problem because it has discrete decision variables such as tasks and processors. Position Vector Format (PVF) is a well-known particle representation for an independent task scheduling problem. PVF representation expresses only which task to be executed on which processor and it does not provide the flow of execution of tasks in processors. The flow is not required for the algorithm that evaluates only the makespan. However, Flow time is also an important metric to evaluate the efficiency of the algorithm for scheduling independent tasks in distributed systems. Flow of execution affects the flow time. The proposed DPSO uses permutation based format for representing the particles to address the above stated problem in PVF.

Linearly Decreasing Inertia (LDI) weight was used in the existing DPSO [3, 5] algorithm. LDI needs only maximum ($W_{\max}$) and minimum ($W_{\min}$) values of W and linearly decreased from $W_{\max}$ to $W_{\min}$. This kind of the inertia factor does not guarantee that the particles have not moved away from the global best. The proposed work designs the inertia weight based on the distance between the particles and their global best. This ensures that the particles have not moved away from their global best because the distance is also considered as a part of velocity to update the position of the particles. Here, the distance represents the count of dissimilar values in two particles such as current particle and global best particle. The Hamming distance is considerably suitable to the above stated distance. Therefore, the proposed inertia weight is entitled as Hamming inertia which is used to improve the performance of the algorithm. The flow of the proposed work is given in Figure 1.

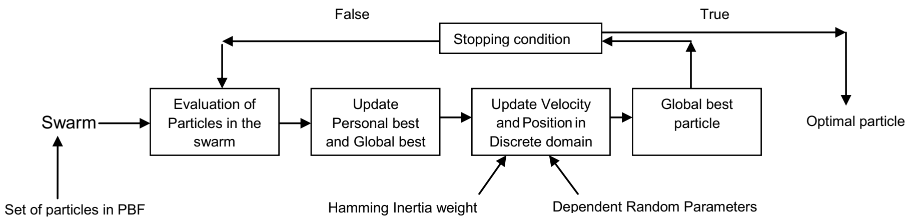

False
Stopping condition
True
Swarm
Evaluation of Particles in the swarm
Update Personal best and Global best
Update Velocity and Position in Discrete domain
Global best particle
Optimal particle
Set of particles in PBF
Hamming Inertia weight
Dependent Random Parameters

Figure 1. Flow diagram of the proposed DPSO

PSO is an optimization algorithm based on population. The system is initialized with a population of random particles. The population of the possible particles 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 significantly low [15].

In PSO, the particles are encoded as a set of real values, which represents the location of a particle in the search space. Task scheduling is one of the combinatorial optimization problems because it has discrete decision variables (e.g. task number or processor number). However, the classical PSO cannot be directly used in the task scheduling problem because their positions are continuous values. The most well-known encoding technique in the literature is Smallest Position Vector (SPV) [16] rule for mapping continuous positions of particles in PSO to the discrete values. The movement of the particle towards the best solution is directed by updating its velocity and position characteristics. The velocity and position updates for PSO are given in Equations (1) and (2). The pseudo code of PSO algorithm with SPV rule in task scheduling problem is given in Algorithm 1.

$$V_i^{(t+1)}(j) = WV_i^t(j) + C_1 r_1(\text{Pbest}_i^t(j) - \text{present}_i^t(j)) + C_2 r_2(\text{Gbest}^t(j) - \text{present}_i^t(j)), \tag{1}$$

$$\text{present}_i^{(t+1)}(j) = \text{present}_i^t(j) + V_i^{(t+1)}(j). \tag{2}$$

Algorithm 1 PSO with SPV for task scheduling problem

```
begin
    Initialize the swarm randomly.
    Initialize each particle position and velocity.
    Using SPV rule to map the particle's position from continuous space into
    discrete space.
    Evaluate each particle and find the Pbest and the Gbest.
repeat
    Update velocity of each particle using Equation (1).
    Update position of each particle using Equation (2).
    Using SPV rule to map the particle's position from continuous space into
    discrete space.
    Evaluate fitness value of each new particle.
    Update Pbest and Gbest for each new particle.
until stopping condition is true.
```

Kang et al. [3] and Izakian et al. [4] proposed PSO called Discrete PSO (DPSO) that can update their particles in a discrete domain directly. Here, the conversion techniques are not required for mapping continuous positions of particles into discrete values and hence much computation time can be saved. The authors [3, 4] developed a DPSO algorithm of a particle representing in position vector format. This representation does not provide the flow of execution of tasks. The flow is not needed, if the algorithm minimizes only the makespan. The proposed algorithm minimizes not only makespan, but also flow time. Therefore, the particles are represented as a permutation of integer values to identify the flow of execution. The velocity and position update methods in DPSO are redefined to operate directly in the permutation based format. The redefined velocity and position update [5] are given in Equations (3) and (4).

$$V_i^{(t+1)}(j) = WV_i^t(j) \cup C_1 r_1(\text{Pbest}_i^t(j) - \text{present}_i^t(j)) \cup C_2 r_2(\text{Gbest}^t(j) - \text{present}_i^t(j)), \tag{3}$$

$$\text{present}_i^{(t+1)}(j) = \text{present}_i^t(j)(\text{swap})V_i^{(t+1)}(j). \tag{4}$$

The HDDPSO uses Hamming inertia weight that is given in Equation (5), where W_0 is the random number between 0.5 and 1, H_i is the current Hamming distance of i^{th} particle from the global best and MDH is the maximum distance of a particle from the global best in that generation.

$$W = W_0 \left(1 - \frac{H_i}{\text{MDH}}\right), \tag{5}$$

$$H_i = \text{Hamming distance}(\text{Gbest}^t, \text{present}_i^t), \tag{6}$$

$$\text{MDH} = \text{Max}(H_i). \tag{7}$$

The two random parameters r_1 and r_2 in Equation (3) 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 completely and the convergence speed of the optimization technique is reduced. Mandal et al. [17] proposed dependent random parameters to control the balance of personal and social experiences in continuous domain. The HDDPSO also uses dependent random parameters r_1 and r_2 in discrete domain. In this method, one single random number r_1 is chosen so that when r_1 is large, $1 - r_1$ is small and vice versa.

The HDDPSO algorithm has taken the redefined velocity and position update methods from [5] and incorporates Hamming inertia weight and dependent random parameters to update the particles. The Equation (3) is rewritten in Equation (8).

$$V_i^{(t+1)}(j) = W(\text{SSO}_1) \cup C_1 r_1(\text{SSO}_2) \cup C_2(1 - r_1)(\text{SSO}_3). \tag{8}$$

An illustrative example of generating the new particle is shown below:

Assume,

$n = 5$; $W = C_1 = C_2 = r_1 = r_2 = 1$;

$V_{\text{old}} = 0$; Pbest $= \{2,5,3,4,1\}$; Gbest $= \{1,3,5,2,4\}$; present $= \{1,2,3,4,5\}$

- $V_{\text{new}} = \{2,5,3,4,1\} - \{1,2,3,4,5\} \cup \{1,3,5,2,4\} - \{1,2,3,4,5\}$

  $SSO_1 = 0$; $SSO_2 = \{(1,2)(2,5)\}$; $SSO_3 = \{(2,3),(3,5),(4,5)\}$

  $$V_{\text{new}} = \{(1,2)(2,5),(2,3),(3,5),(4,5)\}$$

- $\text{present}_{\text{new}} = \{1,2,3,4,5\}(\text{swap})\{(1,2)(2,5),(2,3),(3,5),(4,5)\}$

  $= \{2,1,3,4,5\}(\text{swap})\{(2,5),(2,3),(3,5),(4,5)\}$

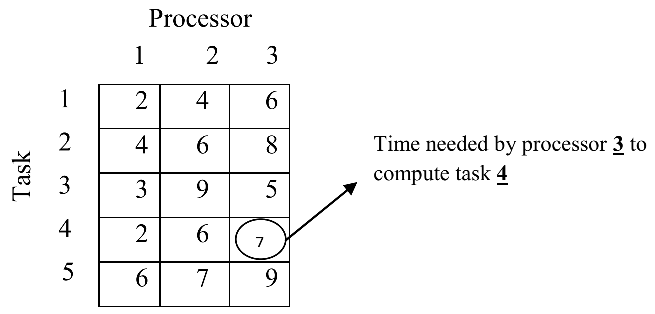
  $$\text{present}_{\text{new}} = \{2,3,1,5,4\}$$

The TS problem is formulated based on the following assumptions:

The n denotes the number of independent tasks $T = \{T_1, T_2, \ldots, T_n\}$ to be scheduled on m processors $P = \{P_1, P_2, \ldots, P_m\}$. All tasks are non-preemptive and every processor processes only a single task at a time. Each processor uses the First-Come, First-Served (FCFS) method for performing the received tasks and every task is processed on a single processor at a time [7]. At the time of submitting these tasks, m processors $P = \{P_1, P_2, \ldots, P_m\}$ are within the Heterogeneous Computing (HC) environment.

Because of the heterogeneous nature of the processors and disparate nature of the tasks in HC, 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 $n \times m$ matrix where m is the number of processors and n is the number of tasks. One row of the ETC matrix represents estimated execution time for a specified task on each processor. Similarly one column of the ETC matrix consists of the estimated execution time of a specified processor for each task. An illustrative example for the ETC model with 5 tasks and 3 processors is shown in Figure 2.

The asymptotic complexity of the HDDPSO algorithm is the same as the DPSO [5] because the algorithm follows the same pseudo code of the DPSO. The pseudo code of the proposed HDDPSO algorithm is given in Algorithm 2.



Task \ Processor	1	2	3
1	2	4	6
2	4	6	8
3	3	9	5
4	2	6	7
5	6	7	9

Time needed by processor **3** to compute task **4**

Figure 2. ETC model for 5×3 (5 tasks and 3 processors)

The following subsections describe in detail the steps of HDDPSO algorithm.

3.1 Particle Representation and Swarm Initialization

The DPSO algorithm [5] starts with a random initial swarm. The random initialization is a simple and straightforward technique. However, the problem in this technique is, some processors are busy with processing, while some processors are idle without any processing. To make better utilization of the processors, the HD-DPSO performs load sharing which assures no processor is idle initially.

Swarm initialization consists of two parts: Particle Initialization (PI) and Processor Allocation (PA). Number of tasks and population size are required to generate particles. Here, a particle is encoded in permutation based method instead of posi-

Algorithm 2 HDDPSO for TS problem

```
begin
    Initialize the swarm with load sharing and initialize each particle position and
    velocity
    Evaluate each particle and find the Personal best and the Global best
repeat
    Update velocity of each particle using Equation (8)
    Update position of each particle using Equation (4)
    Evaluate fitness value of each new particle
    Update Personal best and Global best for each new particle
    Apply VND to the Global best particle
until stopping condition is true
```

tion vector format. In the permutation vector, the position of a task represents the sequence the task is scheduled and the corresponding value indicates a task number. An example of permutation based method is shown in Figure 3.

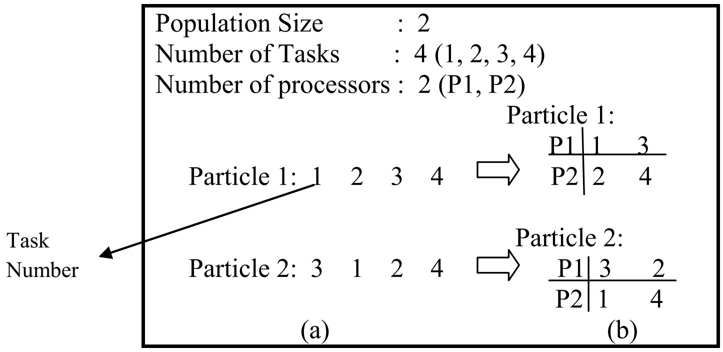


Figure 3. Swarm Initialization a) Particle Initialization b) Processor Allocation with sharing the load

Kang et al. [3] and Izakian et al. [4] developed a DPSO algorithm of a particle representing in PVF. An example for a particle that corresponds to a task assignment that assigns five tasks to three processors is given in Figure 4 a). The internal representation of the particle in Figure 4 a) is shown in Figure 4 b), which always provides a single possible schedule in PVF.

In Figure 4, the tasks 2, 3 and 5 are assigned to processor 1. Because of independent nature of the tasks, the tasks can be executed in any order. So, it can make $3! = 6$ different possible schedules. The possible remaining schedules can be represented using PBF only. Two sample schedules are presented in Figure 5.

It is inferred from Figure 6 to Figure 8, the value of make span does not change if the order of the tasks assigned within a processor varies, but the value of flow time varies if the order of tasks assigned within a processor varies. The PVF is not

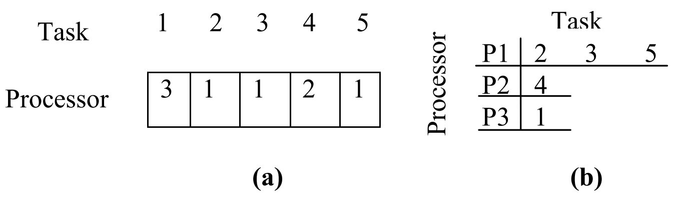


Figure 4. a) A particle in PVF, b) the internal representation of the particle in PVF

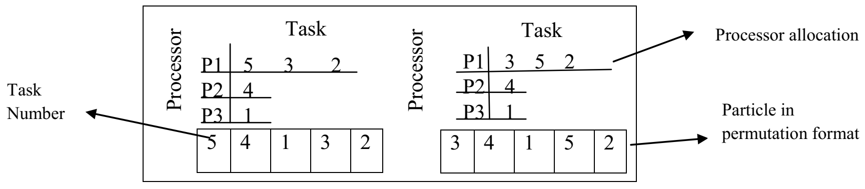
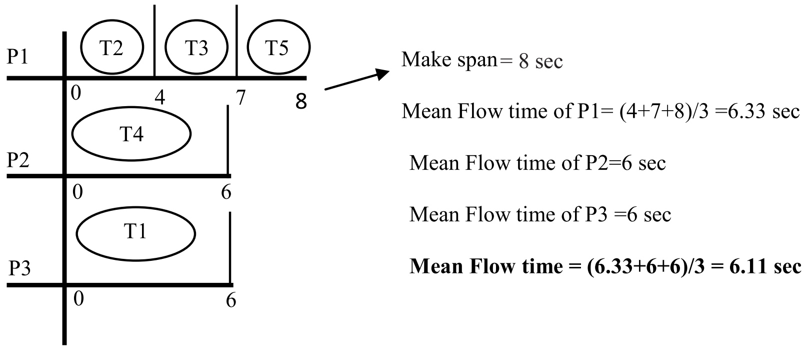


Figure 5. Two possible schedules represented in PBF



Make span= 8 sec

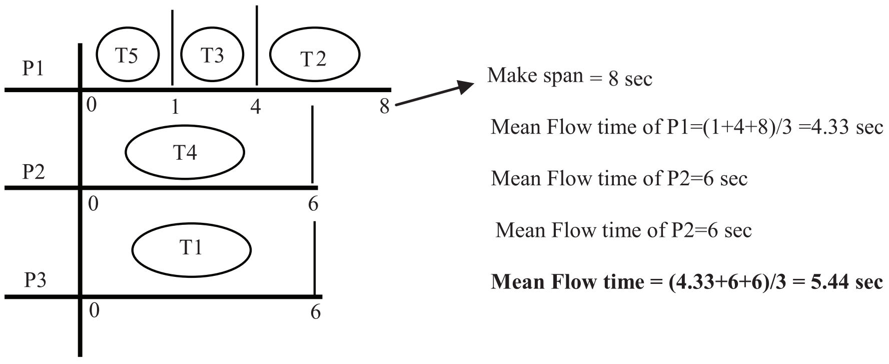
Mean Flow time of P1= (4+7+8)/3 =6.33 sec

Mean Flow time of P2=6 sec

Mean Flow time of P3 =6 sec

Mean Flow time = (6.33+6+6)/3 = 6.11 sec

Figure 6. Make span and mean flow time of a particle shown in Figure 4



Make span = 8 sec

Mean Flow time of P1=(1+4+8)/3 =4.33 sec

Mean Flow time of P2=6 sec

Mean Flow time of P2=6 sec

Mean Flow time = (4.33+6+6)/3 = 5.44 sec

Figure 7. Make span and mean flow time of a left side particle shown in Figure 5

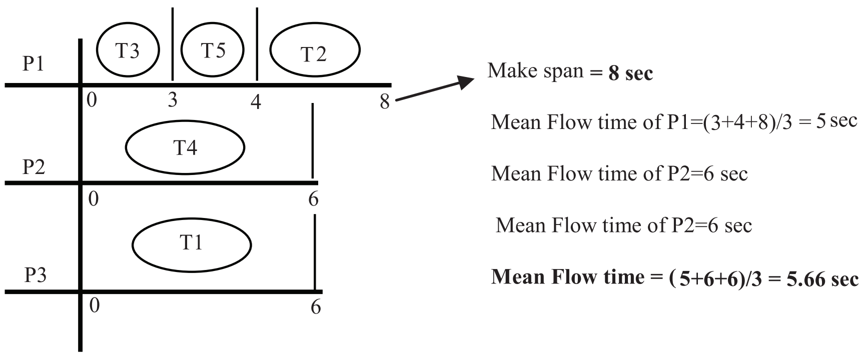

P1
T3
T5
T2
0
3
4
8
P2
T4
0
6
P3
T1
0
6
Make span = **8 sec**
Mean Flow time of P1=(3+4+8)/3 = 5 sec
Mean Flow time of P2=6 sec
Mean Flow time of P2=6 sec
Mean Flow time = (5+6+6)/3 = 5.66 sec

Figure 8. Make span and flow time of a right side particle shown in Figure 5

able to represent the particles in Figures 7 and 8. These can be represented only in PBF. Therefore, the proposed algorithm represents the particles in PBF to improve the performance for minimizing flow time.

3.2 Particle Evaluation

The three evaluation criteria such as make span [5], mean flow time and reliability cost [5] are used to assess the efficiency of the proposed algorithms.

The value of mean flow time [17] is used to evaluate flow time. Assume k is the total number of tasks assigned to processor P_i and F_{ji} is the finishing time of *task* T_j on a processor P_i. The calculation of mean flow time is given in Equation (9).

$$\text{Mean Flow time} = \frac{\sum_{i=1}^{m} M\_Flow_i}{m}, \tag{9}$$

$$M\_Flow_i = \frac{\sum_{j=1}^{k} F_{ji}}{k}. \tag{10}$$

The three objectives are used to evaluate the performance of the scheduler. The weighted single objective function called AWS [6] is used to calculate the fitness value of each solution. This can be estimated using Equation (11).

$$\text{Fitness} = \alpha_1\alpha_2\text{Makespan} + (1-\alpha_1)\alpha_2\text{Mean Flow time} + (1-\alpha_2)\text{Reliability Cost}, \quad \alpha_i \epsilon [0,1]. \tag{11}$$

3.3 Updating the Particle's Personal Best and Global Best Position

The Personal best position (Pbest) of each particle and Global best (Gbest) position of the swarm can be determined based on the fitness value. The Pbest and the Gbest should be determined before updating the position of the particle.

3.4 Updating the Particle's Velocity and Position

The particles in HDDPSO update their velocity and position using Equations (8) and (4), respectively.

3.5 VND Heuristic

Local search technique was not included in DPSO [5] algorithm. Therefore the algorithm may get struck in local optima. The HDDPSO algorithm applies VND [3] (Variable NeighborhooD) local search heuristic when there is no change in the Global best particle for consecutive iterations.

3.6 Stopping Condition

The above iterative processes on a swarm will continue until there is no change in the fitness value of the Global best particle after applying VND heuristic for consecutive iterations. The fitness value of the proposed algorithms and compared algorithms are calculated using AWS method. In this method, the weights of the objective values change generation to generation. The particles in different generation may have the same objective values, but different fitness values. Therefore, the algorithm may take time to converge. Hence, the stopping criterion of all the algorithms is set to predefined maximum number of iteration. In this paper, the maximum number of iteration is set to 1 000.

4 SIMULATION EXPERIMENTS AND ANALYSIS

The simulation results are attained using a set of benchmark ETC instances [19] for the distributed heterogeneous systems. All algorithms are coded in Java and executed in i5 processor.

4.1 Benchmark Instances Description

The simulation is performed on the benchmark ETC instances [19] which are categorized in 12 types of ETC's based on the 3 following metrics: task heterogeneity, machine heterogeneity and consistency. In this benchmark, quality of the ETC matrices are varied in an attempt to simulate various possible HC environments by setting the values of parameters $\text{mean}_{\text{task}}$, 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. The Coefficient of Variation Based (CVB) [3, 20] ETC generation method gives a larger control over the spread of execution time values than the Range Based (RB) method proposed by Braun [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.

All instances consisting of 512 tasks and 16 processors are classified into 12 different types of ETC matrices according to the 3 metrics.

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 – consistent, i – inconsistent, and s – semi-consistent.
- bb indicates the heterogeneity of the tasks; hi – high and lo – low.
- cc represents the heterogeneity of the machines; hi – high and lo – low.

4.2 Algorithms Comparison

The proposed algorithms in this paper are given below:

a) H-DPSO: Incorporate the proposed Hamming inertia in DPSO [3] instead of using LDI

b) LDPSO: Incorporate PBF in DPSO [3]

c) HDPSO: Incorporate Hamming inertia in LDPSO instead of LDI

d) HDDPSO: Incorporate Dependent random parameters in HDPSO

Simulations were carried out to compare the performance analysis of proposed algorithms with respect to:

a) Differential Evolution (DE) [20]. To make an effective comparison, DE [20] is extended to minimize reliability cost also.

b) DPSO (position vector representation) [3]. For an effective comparison, DPSO is extended to minimize mean flow time and reliability cost also.

c) Modified DPSO (MDPSO) [14]. This is a DPSO (position vector representation) with LDI and no VND. Swap mutation was applied to avoid premature convergence. For an effective comparison, mean flow time is used instead of flow time.

d) Multi-Objective DPSO (MODPSO) [5]. This is a DPSO (Permutation based representation) with LDI and no VND. For an effective comparison, mean flow time is used instead of flow time.

All the algorithms are stochastic based algorithms. Each independent run of the same algorithm on a particular problem instance may yield a different result. To make a better comparison of the algorithms each experiment was repeated 10 times with different random seeds and average of the results are tabulated and presented below.

4.3 Parameter Setup

The following parameters are initialized for simulating existing algorithms DE, DPSO, MDPSO, MODPSO and the proposed algorithms H-DPSO, LDPSO, HDPSO and HDDPSO algorithms.

- The population size of all the algorithms is set to 32 as recommended in [3].
- Number of iterations is set to 1 000 for all the algorithms.
- The Failure rate for each processor is uniformly distributed [21] in the range from 0.95×10^{-6}/h to 1.05×10^{-6}/h.

4.4 Performance Comparisons

In the initial population, one solution is generated using min-min heuristic and the others are generated randomly. The average results of compared algorithms and the proposed algorithm are shown in Table 1 for makespan, in Table 2 for mean flow time and in Table 3 for reliability cost.

From Table 1 to Table 3, the first column indicates the ETC instance name, the second, third, fourth, fifth, sixth, seventh, eighth and ninth columns indicate the value obtained by DE, DPSO, MDPSO, H-DPSO, MODPSO, LDPSO, HDPSO and HDDPSO, respectively. From Table 1 to Table 3, the values in bold indicate an optimal value.

From Table 1, the proposed HDDPSO algorithm substantially provides better makespan value in most of the ETC instances than the respective values obtained by other algorithms and LDPSO provides acceptable makespan only in one case.

Table 2 shows that the proposed HDDPSO algorithm can achieve better mean flow time than other algorithms. DPSO, MDPSO, H-DPSO attain admissible mean flow time only for few instances. With obtained reliability cost in Table 3, the proposed HDDPSO algorithm considerably gives better performance in most of the ETC instances than other algorithms. The DE, H-DPSO, LDPSO and HDPSO achieve acceptable results in few cases only.

The existing meta-heuristic algorithms DE, DPSO, MDPSO and proposed H-DPSO represent the particles in position vector format. The existing MODPSO and proposed LDPSO, HDPSO and HDDPSO algorithms represent the particles in permutation based method. The above results proved that the permutation based particle representation algorithms (MODPSO, LDPSO, HDPSO and HDDPSO) provide significant improvements compared to the position vector based particle representation algorithms (DE, DPSO, MDPSO and H-DPSO) for optimizing multiple

ETC Instance	DE	DPSO	MDPSO	Proposed H-DPSO	MODPSO	Proposed LDPSO	Proposed HDPSO	Proposed HDDPSO
c_lo_lo	451 107.72	441 108.14	431 008.43	568 200.82	49 368.42	46 154.02	55 214.35	**43 643.99**
c_lo_hi	178 413.11	188 413.43	187 473.93	70 649.59	99 730.68	90 482.87	82 255.11	**81 612.81**
c_hi_lo	179 996.63	177 927.26	176 323.66	436 224.21	58 249.97	49 637.11	57 680.64	**43 051.54**
c_hi_hi	76 838.66	86 838.73	85 388.39	99 854.43	107 577.11	77 396.29	93 516.83	**66 631.78**
i_lo_lo	1 606 821.45	1 506 821.25	1 513 281.22	605 141.31	36 936.14	36 121.21	33 948.45	**32 705.98**
i_lo_hi	2 232 555.56	2 232 529.67	2 332 100.16	609 668.95	41 757.22	44 309.75	42 768.94	**35 669.85**
i_hi_lo	504 768.32	484 768.45	484 918.35	1 204 419.23	41 317.72	43 776.15	44 000.51	**36 901.93**
i_hi_hi	877 159.62	977 159.28	997 019.18	452 256.82	54 514.32	43 518.04	66 065.56	**37 296.16**
s_lo_lo	954 417.64	924 417.39	900 492.94	195 943.47	42 991.11	38 812.83	41 607.35	**38 001.28**
s_li_hi	68 716.88	58 716.82	60 716.82	222 039.16	69 037.49	**49 788.27**	51 983.13	50 695.75
s_hi_lo	648 728.23	748 728.25	729 718.05	1 213 979.11	50 652.82	42 031.54	39 670.28	**37 104.56**
s_hi_hi	702 668.29	672 668.55	692 668.44	232 549.82	72 357.83	63 718.57	65 144.73	**53 539.03**

Table 1. Comparison of makespan (in seconds) of proposed algorithms with existing algorithms

ETC Instance	DE	DPSO	MDPSO	Proposed H-DPSO	MODPSO	Proposed LDPSO	Proposed HDPSO	Proposed HDDPSO
c_lo_lo	58 138.47	55 138.47	52 939.92	71 025.09	16 479.65	16 703.19	16 828.59	**16 307.47**
c_lo_hi	29 551.81	23 551.67	21 291.72	**8 831.19**	16 511.09	16 214.53	16 241.61	15 632.38
c_hi_lo	20 240.81	22 240.82	22 102.12	54 528.03	15 891.55	16 415.92	16 115.71	**15 872.59**
c_hi_hi	11 854.93	10 854.83	**10 729.18**	12 481.81	17 296.37	16 394.39	16 853.59	15 420.42
i_lo_lo	198 352.59	188 352.59	189 952.35	75 642.67	16 537.13	15 653.43	16 205.09	**15 196.03**
i_lo_hi	379 066.18	279 066.18	281 126.98	76 208.62	16 318.29	14 944.84	15 401.63	**14 912.74**
i_hi_lo	20 435.81	26 435.81	26 785.93	150 552.41	15 688.48	15 542.91	15 716.21	**14 992.95**
i_hi_hi	123 144.84	122 144.84	130 014.31	56 532.11	17 727.99	15 770.66	16 794.22	**14 204.53**
s_lo_lo	151 552.14	115 552.14	121 242.19	24 492.93	16 380.35	16 173.51	16 664.88	**13 433.23**
s_li_hi	10 339.61	**7 339.61**	7 399.69	27 754.89	16 294.42	15 107.32	15 959.73	15 269.09
s_hi_lo	90 591.03	93 591.03	94 191.43	151 747.31	15 858.55	15 342.13	15 584.69	**14 875.33**
s_hi_hi	80 083.53	84 083.53	84 993.63	29 068.73	17 548.32	15 579.29	16 392.15	**14 963.85**

Table 2. Comparison of mean flow time (in seconds) of proposed algorithms with existing algorithms

objectives. Comparison among the proposed algorithms, HDDPSO provides better results in majority of the ETC instances with respect to multiple objectives.

The following part describes the performance improvement of the proposed HDDPSO with other permutation based particle representation algorithms (MODPSO, LDPSO and HDPSO) in terms of average Relative Percentage Deviation (RPD) [3].

ETC Instance	DE	DPSO	MDPSO	Proposed H-DPSO	MODPSO	Proposed LDPSO	Proposed HDPSO	Proposed HDDPSO
c_lo_lo	1.050639	1.000639	1.030139	1.000961	0.509932	0.532011	0.515035	**0.504714**
c_lo_hi	0.839667	0.296866	**0.192816**	0.269642	0.519825	0.508718	0.503148	0.493887
c_hi_lo	**0.473802**	0.573804	0.551804	0.575274	0.504692	0.529443	**0.477007**	0.483374
c_hi_hi	**0.305931**	0.307031	0.306931	0.306731	0.524231	0.498573	0.515936	0.473123
i_lo_lo	1.940761	1.419476	1.310476	1.089812	0.512596	**0.488641**	0.539979	0.496405
i_lo_hi	1.731361	1.341873	1.243813	1.234524	0.505007	0.535615	**0.469199**	0.491156
i_hi_lo	0.725419	1.725419	1.800419	1.405323	0.497707	0.479278	0.499176	**0.478036**
i_hi_hi	1.100903	1.172903	1.181003	1.253371	0.534967	0.509245	0.518727	**0.439922**
s_lo_lo	1.007405	1.500741	1.199741	0.918074	0.507683	0.502852	0.529157	**0.476322**
s_li_hi	0.566386	0.460566	0.461066	0.909172	0.513329	0.474346	0.494333	**0.434523**
s_hi_lo	0.995581	0.981049	0.982009	1.479681	0.501462	0.492261	0.497563	**0.470336**
s_hi_hi	0.597713	0.935977	0.921077	**0.461891**	0.528311	0.483202	0.501902	0.473279

Table 3. Comparison of reliability cost of proposed algorithms with existing algorithms

RPD is calculated using Equation (12), where P is the average result of the HDDPSO algorithm and AC_i is the average result provided by MODPSO, LDPSO and HDPSO for each instance.

$$RPD = (AC_i - P)/P * 100. \tag{12}$$

The standard deviation of fitness values of MODPSO, LDPSO, HDPSO and HDDPSO algorithms for 10 different random seeds are calculated and presented in Table 4. The values in bold indicates the optimal results. In most of the ETC instances, HDDPSO provides optimal results than other algorithms.

ETC Instance	**MODPSO**	**Proposed HDDPSO**	**RPD (%)**	**Proposed LDPSO**	**Proposed HDDPSO**	**RPD (%)**	**Proposed DPSO**	**Proposed HDDPSO**	**RPD (%)**
c_lo_lo	2 516.61	**2 042.06**	23.24	2 203.03	**2 042.06**	7.88	2 657.69	**2 042.06**	30.15
c_lo_hi	1 527.53	**1 126.94**	35.55	1 527.53	**1 126.94**	35.55	2 516.61	**1 126.94**	123.31
c_hi_lo	**1 223.17**	2 393.56	−48.89	3 511.89	**2 393.56**	46.72	**2 000.01**	2 393.56	−16.44
c_hi_hi	3 908.79	**1 527.53**	155.89	4 163.33	**1 527.53**	172.55	5 800.01	**1 527.53**	279.69
i_lo_lo	4 041.45	**1 877.35**	115.27	**1 014.89**	1 877.35	−45.94	2 181.67	**1 877.35**	16.21
i_lo_hi	**2 377.98**	2 445.75	−2.77	**2 000.01**	2 445.75	−18.22	**1 527.53**	2 445.75	−37.54
i_hi_lo	3 514.55	**2 486.75**	41.33	**2 081.67**	2 486.75	−16.29	2 741.75	**2 486.75**	10.25
i_hi_hi	6 931.61	**2 001.67**	246.29	5 041.45	**2 001.67**	151.86	3 705.55	**2 001.67**	85.12
s_lo_lo	2 516.61	**2 167.49**	16.11	3 605.55	**2 167.49**	66.35	**1 539.48**	2 167.49	−28.97
s_li_hi	2 500.01	**1 527.53**	63.66	2 000.01	**1 527.53**	30.93	2 055.05	**1 527.53**	34.53
s_hi_lo	1 542.22	**1 000.01**	54.22	4 041.45	**1 000.01**	304.14	2 400.01	**1 000.01**	139.99
s_hi_hi	5 990.01	**1 501.88**	298.83	3 516.61	**1 501.88**	134.15	3 911.89	**1 501.88**	160.47
Average			**83.23**			**72.47**			**66.39**

Table 4. Comparison of standard deviation of fitness value of HDDPSO algorithm with other algorithms

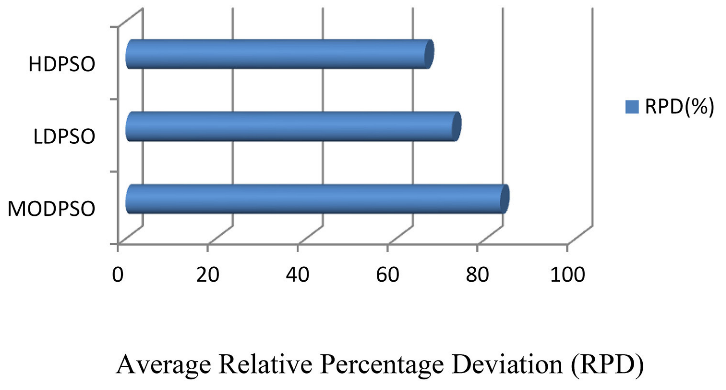


Figure 9. Performance improvement of HDDPSO compared to other algorithms in terms of average RPD (%)

Performance improvement of the proposed HDDPSO algorithm is presented in Figure 9. It shows that the HDDPSO algorithm considerably provides better performance than MODPSO, LDPSO, HDPSO in terms of fitness value by 83.23 %, 72.47 % and 66.39 % across all ETC instances respectively.

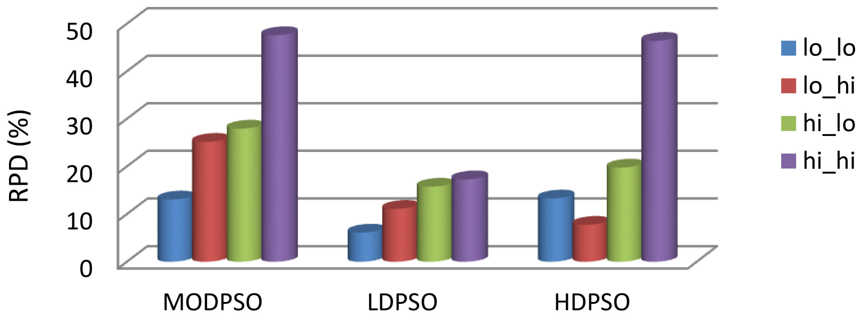


Figure 10. RPD comparison of HDDPSO algorithm with other algorithms with respect to makespan

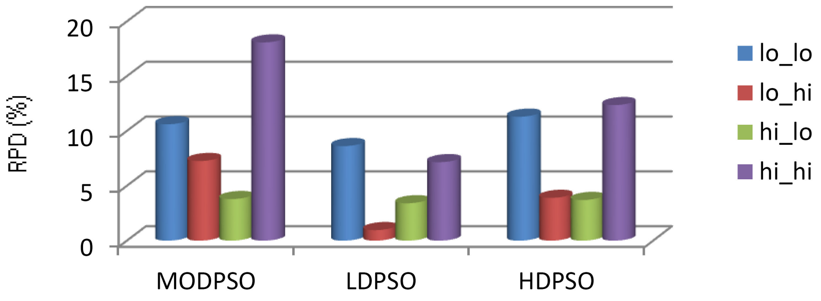


Figure 11. RPD comparison of HDDPSO algorithm with other algorithms with respect to mean flow time

From Table 5, it is inferred that HDDPSO is able to give better performance in terms of make span by 28.43 %, 12.56 % and 21.86 %, mean flow time by 9.94 %, 5.03 % and 7.82 % and reliability cost by 28.43 %, 12.56 % and 21.86 % compared to MODPSO, LDPSO and HDPSO across all ETC instances respectively. The negative RPD values in the Table 5 indicate that the algorithm does not provide the optimal results.

Table 6 presents the average RPD values obtained by the algorithms with respect to ETC instances such as lo_lo, lo_hi, hi_lo and hi_hi. The values in bold represent the highest RPD value of the algorithm.

The results obtained from Table 6 and Figure 10 to Figure 12, the proposed HDDPSO algorithm provides the highest RPD values in hi_hi ETC instances. Hence, it is found to be more suitable for high task and high machine heterogeneity.

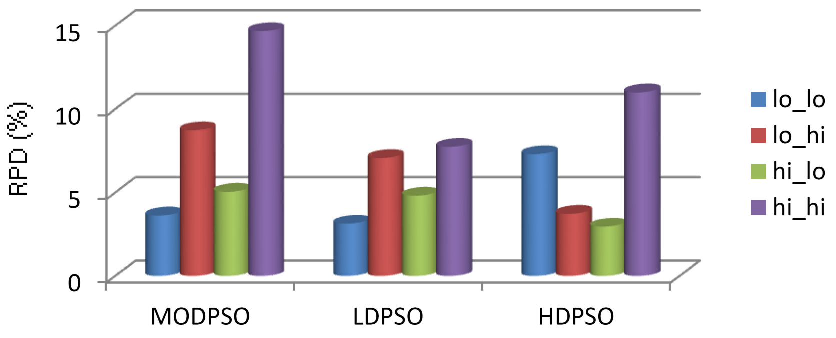


Figure 12. RPD comparison of HDDPSO algorithm with other algorithms with respect to reliability cost

Instance	**Make span**			**Mean Flow Time**			**Reliability Cost**		
	RPD (%) LDPSO-NVND, HDDPSO	**RPD (%) LDPSO-VND, HDDPSO**	**RPD (%) HDPSO-VND, HDDPSO**	**RPD (%) LDPSO-NVND, HDDPSO**	**RPD (%) LDPSO-VND, HDDPSO**	**RPD (%) HDPSO-VND, HDDPSO**	**RPD (%) LDPSO-NVND, HDDPSO**	**RPD (%) LDPSO-VND, HDDPSO**	**RPD (%) HDPSO-VND, HDDPSO**
c_lo_lo	13.12	5.75	26.51	1.06	2.43	3.20	1.03	5.41	2.04
c_lo_hi	22.20	10.87	0.79	5.62	3.72	3.90	5.25	3.00	1.88
c_hi_lo	35.30	15.30	33.10	0.12	3.42	1.53	4.41	9.53	−1.32
c_hi_hi	61.45	16.16	40.35	12.17	6.32	9.30	10.80	5.38	9.05
i_lo_lo	12.93	10.44	3.80	8.83	3.01	6.64	3.26	−1.56	8.78
i_lo_hi	17.07	24.22	19.90	9.43	0.22	3.28	2.82	9.05	−4.47
i_hi_lo	11.97	18.63	19.24	4.64	3.67	4.82	4.11	0.26	4.42
i_hi_hi	46.17	16.68	77.14	24.81	11.03	18.23	21.60	15.76	17.91
s_lo_lo	13.13	2.14	9.49	21.94	20.40	24.06	6.58	5.57	11.09
s_lo_hi	36.18	−1.80	2.54	6.72	−1.06	4.52	18.14	9.16	13.76
s_hi_lo	36.51	13.28	6.92	6.61	3.14	4.77	6.62	4.66	5.79
s_hi_hi	35.15	19.01	21.68	17.27	4.11	9.55	11.63	2.10	6.05
Average	**28.43**	**12.56**	**21.79**	**9.94**	**5.03**	**7.82**	**8.02**	**5.69**	**6.25**

Table 5. Performance comparison of RPD (%) of HDDPSO algorithm with other algorithms

ETC	**Make span**			**Mean Flow Time**			**Reliability Cost**		
Instance	**RPD (%) MODPSO, HDDPSO**	**RPD (%) LDPSO, HDDPSO**	**RPD (%) HDPSO, HDDPSO**	**RPD (%) MODPSO, HDDPSO**	**RPD (%) LDPSO, HDDPSO**	**RPD (%) HDPSO, HDDPSO**	**RPD (%) MODPSO, HDDPSO**	**RPD (%) LDPSO, HDDPSO**	**RPD (%) HDPSO, HDDPSO**
lo_lo	13.06	6.11	13.27	10.61	8.61	11.31	3.62	3.14	7.30
lo_hi	25.15	11.09	7.74	7.26	0.96	3.91	8.74	7.07	3.72
hi_lo	27.93	15.74	19.75	3.79	3.41	3.71	5.05	4.82	2.96
hi_hi	**47.59**	**17.28**	**46.39**	**18.08**	**7.15**	**12.36**	**14.68**	**7.75**	**11.01**

Table 6. Performance comparison of average RPD of HDDPSO algorithm with other algorithms

It can be seen from the results that the proposed HDDPSO algorithm remarkably gives better performance than exiting algorithms DE, DPSO, MDPSO and the proposed H-DPSO by representing the particles in permutation based method which optimizes the multiple objectives. Compared to the existing MODPSO and the proposed LDPSO, using Hamming inertia confirmed the particles are not moved away from the global best particle. The proposed HDDPSO algorithm has also ensured that the particles are not moved too far away from the local optimum due to make a dependency between the random parameters to control the balance of personal and social experiences. Therefore, the proposed HDDSPO is significantly better than the proposed HDPSO.

5 CONCLUSION

The proposed HDDPSO algorithm is efficient and adapts the inertia weight based on the Hamming distance of the particles from global best. In comparison with the position vector based algorithms, the permutation based algorithms provide significant results in majority of the ETC instances. In addition, an effective VND local search algorithm is embedded into the DPSO to perform exploitation. Simulation results and comparisons with existing algorithms demonstrated the effectiveness of the proposed HDDPSO algorithm and it also shows that the HDDPSO is more suitable for high task and high machine heterogeneity environment.

REFERENCES

[1] Kennedy, J.—Eberhart, R: Particle Swarm Optimization. Proceeding of the Fourth IEEE International Conference on Neural Networks, 1995, Vol. 4, pp. 1942–1948, doi: 10.1109/ICNN.1995.488968.

[2] Braun, T. D.—Siegel, H. J.—Bec, N.: 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, 2001, pp. 810–837, doi: 10.1006/jpdc.2000.1714.

[3] Kang, Q.—He, H.: A Novel Discrete Particle Swarm Optimization Algorithm for Meta-Task Assignment in Heterogeneous Computing Systems. Elsevier, Microprocessors and Microsystems, Vol. 35, 2011, No. 1, pp. 10–17, doi: 10.1016/j.micpro.2010.11.001.

[4] Izakian, H.—Ladani, B. T.—Abraham, A.—Snasel, V.: A Discrete Particle Swarm Optimization Approach for Grid Job Scheduling. International Journal of Innovative Computing, Information and Control, Vol. 6, 2010, No. 9, pp. 1–14.

[5] Sarathambekai, S.—Umamaheswari, K.: Task Scheduling in Distributed Systems Using Discrete Particle Swarm Optimization. International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 4, 2014, No. 2, ISSN 2277-128X.

[6] Kim, I. Y.—de Weck, O. L.: Adaptive Weighted Sum Method for Multi-Objective Optimization: A New Method for Pareto Front Generation. Springer-Structural and Multidisciplinary Optimization, Vol. 31, 2006, No. 2, pp. 105–116.

[7] Izakian, H.—Abraham, A.—Snasel, V.: Meta-Heuristic Based Scheduling Meta-Tasks in Distributed Heterogeneous Computing Systems. Sensors (Basel), Vol. 9, 2009, No. 7, pp. 5339–5350, doi: 10.3390/s90705339.

[8] Suresh, K.—Ghosh, S.—Kundu, D.—Sen, A.—Das, S.—Abraham, A.: Inertia-Adaptive Particle Swarm Optimizer for Improved Global Search. Proceeding of IEEE International Conference on Intelligent Systems and Design, Vol. 2, 2008, pp. 253–258, doi: 10.1109/ISDA.2008.199.

[9] Izakian, H.—Abraham, A.—Snasel, V.: Performance Comparison of Six Efficient Pure Heuristics for Scheduling Meta-Tasks on Heterogeneous Distributed Environments. Neural Network World, Vol. 19, 2009, No. 6, pp. 695–710.

[10] Abraham, A.—Liu, H.—Grosan, C.—Xhafa, F.: Nature Inspired Meta-Heuristics for Grid Scheduling: Single and Multi-Objective Optimization Approaches. Springer, Studies in Computational Intelligence, Vol. 146, 2008, pp. 247–272, doi: 10.1007/978-3-540-69277-5_9.

[11] Shi, Y.—Eberhart, R.: A Modified Particle Swarm Optimizer. Proceeding of IEEE International Conference on Evolutionary Computation, 2002, pp. 69–73, doi: 10.1109/ICEC.1998.699146.

[12] Xin, J.—Chen, G.—Hai, Y.: A Particle Swarm Optimizer with Multistage Linearly-Decreasing Inertia Weight. Proceeding of IEEE International Joint Conference on Computational Sciences and Optimization, Vol. 1, 2009, pp. 505–508.

[13] Bansal, J. C.—Singh, P. K.—Saraswat, M.—Verma, A.—Jadon, S. S.—Abraham, A.: Inertia Weight Strategies in Particle Swarm Optimization. IEEE World Congress on Nature and Biologically Inspired Computing (NaBIC), Vol. 3, 2011, pp. 633–640, doi: 10.1109/NaBIC.2011.6089659.

[14] Sarathambekai, S.—Umamaheswari, K.: Comparison among Four Modified Discrete Particle Swarm Optimization for Task Scheduling in Heterogeneous Computing Systems. International Journal of Soft Computing and Engineering (IJSCE), Vol. 3, 2013, No. 2, pp. 371–378.

[15] Uysal, O.—Bulkan, S.: Comparison of Genetic Algorithm and Particle Swarm Optimization for Bicriteria Permutation Flowshop Scheduling Problem. International Journal of Computational Intelligence Research, Vol. 4, 2008, No. 2, pp. 159–175, ISSN 0973-1873.

[16] Kaur, K.—Tiwari, S. P.: Grid Scheduling Using PSO with SPV Rule. International Journal of Advanced Research in Computer Science and Electronics Engineering, Vol. 1, 2012, No. 5, pp. 20–26.

[17] Mandal, D.—Ghoshal, S. P.—Bhattacharjee, A. K.: Radiation Pattern Optimization for Concentric Circular Antenna Array with Central Element Feeding Using Craziness Based Particle Swarm Optimization. International Journal of RF and Microwave Computer-Aided Engineering, Vol. 5, 2010, pp. 577–586, doi: 10.1002/mmce.20467.

[18] Lindeke, R.: Scheduling of Jobs. Spring, IE 3265 – POM, 2005, `http://www.d.umn.edu/~rlindek1/.../Scheduling%20of%20Jobs_Sset11.ppt`.

[19] Ali, S.—Siegel, H. J.: Representing Task and Machine Heterogeneities for Heterogeneous Computing Systems. Tamkang Journal of Science and Engineering. Vol. 3, 2000, No. 3, pp. 195–207.

[20] Kromer, P.—Snasel, V.—Platos, J.—Abraham, A.—Izakian, H.: Differential Evolution for Scheduling Independent Tasks on Heterogeneous Distributed Environments. Springer, Advances in Intelligent and Soft Computing, Vol. 67, 2010, pp. 127–134, doi: 10.1007/978-3-642-10687-3_12.

[21] Qin, X.—Jiang, H.: Dynamic, Reliability-Driven Scheduling of Parallel Real-Time Jobs in Heterogeneous Systems. IEEE International Conference on Parallel Processing, 2001, pp. 113–122, doi: 10.1109/ICPP.2001.952053.

Subramaniam Sarathambekai is working as Assistant Professor (Senior Grade) in the Department of IT, PSG College of Technology, Coimbatore, Tamilnadu, India. She received her MTech degree (IT) in 2010. She is currently pursuing her Ph.D. in the field of evolutionary computation. Her areas of interest include distributed systems, swarm intelligence, web technologies and algorithm design.

Kandaswamy Umamaheswari is working as Professor in the Department of IT, PSG College of Technology, Coimbatore, Tamilnadu, India. She received her Ph.D. degree in 2010. Her research areas include classification techniques in data mining and other areas of interest are information retrieval, software engineering, theory of computation and compiler design. She has published more than 50 papers in international, national journals and conferences. She serves as an editor for National Journal of Technology, PSG College of Technology, and as a reviewer for many national and international journals.