Engineering with Computers (2016) 32:35–47
DOI 10.1007/s00366-015-0396-z

ORIGINAL ARTICLE

A novel hybrid PSO–GA meta-heuristic for scheduling of DAG with communication on multiprocessor systems

Neetesh Kumar · Deo Prakash Vidyarthi

Received: 28 July 2014 / Accepted: 16 January 2015 / Published online: 1 February 2015

Abstract This work presents a novel hybrid meta-heuristic that combines particle swarm optimization and genetic algorithm (PSO–GA) for the job/tasks in the form of directed acyclic graph (DAG) exhibiting inter-task communication. The proposed meta-heuristic starts with PSO and enters into GA when local best result from PSO is obtained. Thus, the proposed PSO–GA meta-heuristic is different than other such hybrid meta-heuristics as it aims at improving the solution obtained by PSO using GA. In the proposed meta-heuristic, PSO is used to provide diversification while GA is used to provide intensification. The PSO–GA is tested for task scheduling on two standard well-known linear algebra problems: LU decomposition and Gauss–Jordan elimination. It is also compared with other states-of-the-art heuristics for known solutions. Furthermore, its effectiveness is evaluated on few large sizes of random task graphs. Comparative study of the proposed PSO-GA with other heuristics depicts that the PSO–GA performs quite effectively for multiprocessor DAG scheduling problem.

Keywords Multiprocessor DAG scheduling · Genetic algorithm · Particle swarm optimization · NP-hard · Critical path

N. Kumar · D. P. Vidyarthi (✉)
School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi 110067, India
e-mail: dpv@mail.jnu.ac.in

N. Kumar
e-mail: dgoldneetesh15@gmail.com

1 Introduction

Rapid enhancement in numbers and sizes of computational problems is radically motivating to design the modern parallel architecture. The applications that are computationally expensive are using parallel CPU architecture (multiprocessor) vastly [4].Task scheduling techniques play an effective role in efficient utilization of such multiprocessor systems. Task scheduling basically refers to allocate N number of tasks onto M number of available processing units (CPU) with the objectives to enhance the performance of the system. However, number of constraints exposed by both, the applications and the hardware infrastructure, makes the task scheduling an NP-hard problem [2, 3]. Many other factors such as task inter-dependency, discriminating nature of tasks, uniformity/diversity in task's execution time, processors topology etc. makes the scheduling problem further complex. Because of such complexities, scheduling continue to be an important research area with the possibility of applying various heuristics and meta-heuristics for a reasonably good solution [18].

Various heuristics and meta-heuristics techniques for multiprocessor task scheduling problem are available in literature [1, 2, 6–15, 25]. Jin et al. [2]. presents a comparative study on different heuristics; Min–Min heuristic by Ibarra and Kim [6], Chaining heuristics by Djordjevic and Tosic [7], A* search by Kcafil et al. [8], Simulated annealing by Monte Carlo [9], Tabu Search by Tian [10] and Porto et al. [11], highest level first with estimated times (HLFET) by Adam et al. [12], insertion scheduling heuristic (ISH) by Kruatrachue and Lewis [13, 14], duplication scheduling heuristic (DSH) by Kruatrachue and Lewis [13, 14] and by Ahmad and Kwok [15] and genetic algorithm (GA) by Hou et al. [25]. Jin et al. [2] presents a performance study number of heuristics/meta-heuristics for

homogeneous multiprocessor task scheduling and considering task precedence constraint and task inter-communication delays. They implemented different heuristics on two well-known problems of linear algebra: LU decomposition and Gauss–Jordan elimination. Furthermore, Ahmad and Ali [1] proposed a multiprocessor task scheduling that uses particle swarm optimization (PSO) and evaluated its performance on Gauss–Jordan elimination problem. They also compared its performance with DSH and GA heuristics and concluded that PSO performs better than GA.

Blum et al. [17] observed that complementary characteristics of different optimization heuristics benefits from hybridization. Inspired from this, an improved PSO and GA hybridization for multiprocessor task scheduling problem is proposed in this paper. According to [1], PSO provides better solutions over other heuristics/meta-heuristics for multiprocessor task scheduling problem. Also, the work of [16] infers that GA works effectively when used as a local search optimization technique in a hybrid meta-heuristic. Together, GA and PSO have been used widely to optimize solutions for different applications [26, 28–30, 33].

This paper hybridizes the PSO and the GA for multiprocessor DAG scheduling problem in which PSO works as a solution builder and GA as solution refiner to be used as a local search optimization technique. Initial population is produced by PSO in form of particles (solutions). Few good particles enter into the new GA generation. GA operators, crossover and mutation, generate some newer and better particles after few generation which updates global best solution obtained using PSO. This procedure is repeated until the termination condition of the PSO is met. The new generated particles in GA component are expected to be better as these are produced from the genetic characteristics of the best fitness particles. The novelty of the proposal is that in each iteration GA component builds the new solutions using their parent's features to act as local search technique and PSO component constructs diverse solution which provide a global search. Thus, the proposed PSO–GA hybrid meta-heuristic inculcates the advantages of both the meta-heuristics with positive feedback mechanism which is not so in other hybrid PSO, GA meta-heuristics [26, 28–30, 33]. Furthermore, to the best of author's knowledge, the hybrid PSO, GA has not been applied earlier for the multiprocessor DAG scheduling problem.

The outline of the paper is as follows. An illustration of the multiprocessor task scheduling problem is given in Sect. 2. The proposed meta-heuristic for multiprocessor task scheduling using PSO-GA is fully described in Sect. 3. The performance of the proposed meta-heuristic with comparative study with other heuristics is given in Sect. 4. Finally, the work is concluded in Sect. 5.

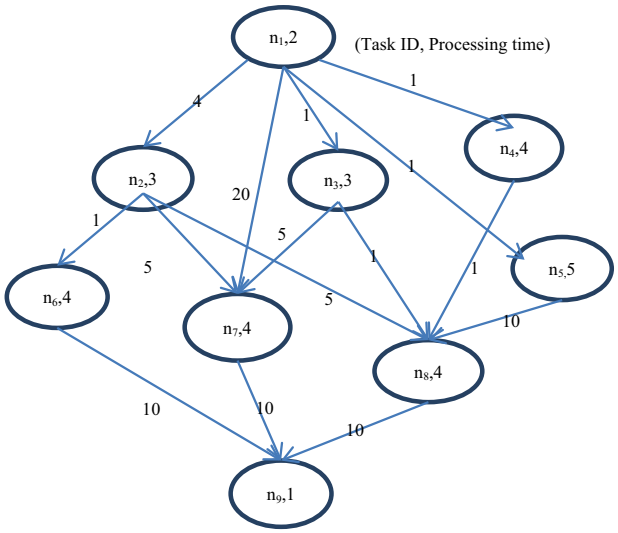

Fig. 1 Directed acyclic graph

2 Multiprocessor DAG scheduling problem

This section focuses on the multiprocessor task scheduling problem with precedence constraint and communication amongst the tasks [1, 18]. All the parameters, related to the problem are assumed to be deterministic. Also, it is assumed that there are M homogeneous processors in the multiprocessor system and N tasks. Job/tasks, can be represented in form of directed acyclic graph (DAG) as G (V, E), where V and E denote the set of nodes and the set of directed edges, respectively. A node $n_i \in V$ represents task number followed by some weight w (n_i) depicting processing time of the task n_i. A directed edge $(n_i, n_j) \in E$ represents the communication and the precedence between the two tasks n_i and n_j. Precedence (n_i, n_j) indicates that node n_j cannot start its execution before n_i. An edge (n_i, n_j) is assigned some weight w (n_i, n_j) which represents the communication between n_i and n_j. If tasks n_i and n_j are assigned to the same processor, communication becomes zero i.e., n_j can start its execution latest by finish time (n_i). Otherwise n_j will start its execution on some other processor latest by finish time $(n_i) + w(n_i, n_j)$. The objective is to assign N number of DAG tasks onto M number of homogeneous processors with the given precedence and communication constraints such that makespan of the DAG is minimized.

Figure 1 shows a DAG with 9 nodes (tasks) represented by oval inside which node number and processing time is shown. Each edge denotes the precedence relation between the nodes along with the communication cost. Figure 2 shows a feasible schedule with respect to the DAG of Fig. 1 on two homogeneous processors that gives makespan of 25 time units each.

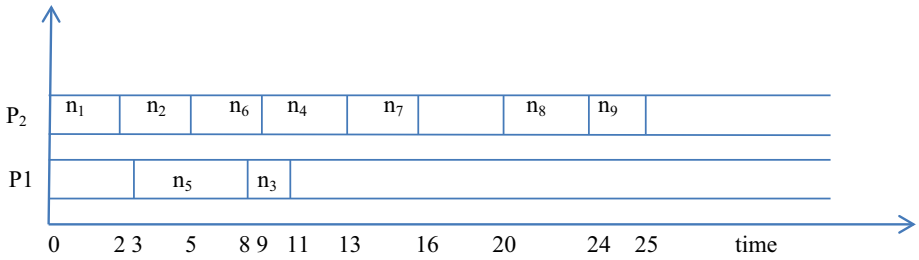

Fig. 2 A feasible schedule corresponding to Fig. 1

3 The proposed meta-heuristic

In this section, the proposed hybrid meta-heuristic (PSO-GA) with complete flowchart describing its various features is demonstrated. It begins by exploring discrete solutions using PSO which further are exploited using GA for a global best (G_{best}), instantly. The process is repeated until a termination condition of PSO (Sect. 3.2.1) is met.

3.1 Framework

The algorithmic flowchart of the proposed hybrid meta-heuristic (PSO-GA) for task scheduling in multiprocessor system is presented in Fig. 3. It starts with random initial population (particles). Smallest position value (SPV) rule [1, 5] is applied thereafter to produce sequence vector corresponding to each particle. The fitness of each particle is calculated and personal best (P_{best}) and global best (G_{best}) are updated. If at least one termination condition of PSO is met, the algorithm terminates otherwise it continues. Consequently, all the parameters including inertia weight, velocity vector, position vector and Pbest of each particle are updated. To update the G_{best}, t number of best particles (with minimum fitness) are extracted using a module min_fitness() and submitted to the GA part of the meta-heuristic as an initial population of GA. GA operators, crossover and mutation [18], are applied to exploit these submitted particles. Using SPV rule, each solution is scored and using a selection method [18] again t best solutions are forwarded to the next GA generation. The best solution of the GA (B_{sol}) out of them is updated. For each PSO iteration, same GA procedure is repeated until a termination condition of GA (Sect. 3.2.1) is met. When it exits from the GA part, G_{best} is updated. The PSO–GA hybrid meta-heuristic repeatedly continues until at least one predefined termination condition of PSO (Sect. 3.2.1) is met.

3.2 PSO-related terms

This section introduces PSO and its related terms and operators embedded into the proposed PSO–GA hybrid meta-heuristic.

3.2.1 Basic terms

Kennedy et al. [19] proposed PSO as an optimization technique that mimics the behavior of social creatures i.e., particles in food searching [1, 5]. In this technique, all particles search food in multidimensional search space based on their two important characteristics; position (suggested solution) and velocity (rate of change of particle position). If any particle finds optimal path to food location, it attracts other particles to follow its path. The optimal path is determined by fitness function. All particles move toward the optimal solution updating their personal best (Pbest) and global best (G_{best}) solution. Finally, all particles reach to the destination following the most optimal path.

Position vector $\vec{X}$ referred as position vector of length N, where N represents the number of dimensions or nodes in the task graph. It is represented by $\vec{X} = [x_1 \ldots x_i \ldots x_N]$, where x_i represents the position value in the ith dimension.

Velocity vector $\vec{V}$ referred as velocity vector of length N, where N represents the number of dimensions or nodes in the task graph. Velocity vector is represented by $\vec{V} = [v_1 \ldots v_i \ldots v_N]$, where, v_i represents velocity value in the ith dimension.

Inertia weight ω^k, an important parameter, used to control the impact of pervious velocity on the current velocity (kth iteration).

Personal best $P\text{best}_i^k$ represents the local best fitness position of ith particle until kth iteration.

Global best G_{best}^k represents globally best fitness position achieved by global best particle until kth iteration.

Termination condition Two different termination conditions are used; one is the given number of iterations and the other is the convergence of the solution. In PSO, convergence is checked as G_{best} of the PSO component equal to the CP length (critical path discussed in next subsection). In GA, convergence is checked as B_{sol} of the GA component (shown in Fig. 3) equal to the CP length. Number of iterations is different for both PSO and GA components and are decided empirically.

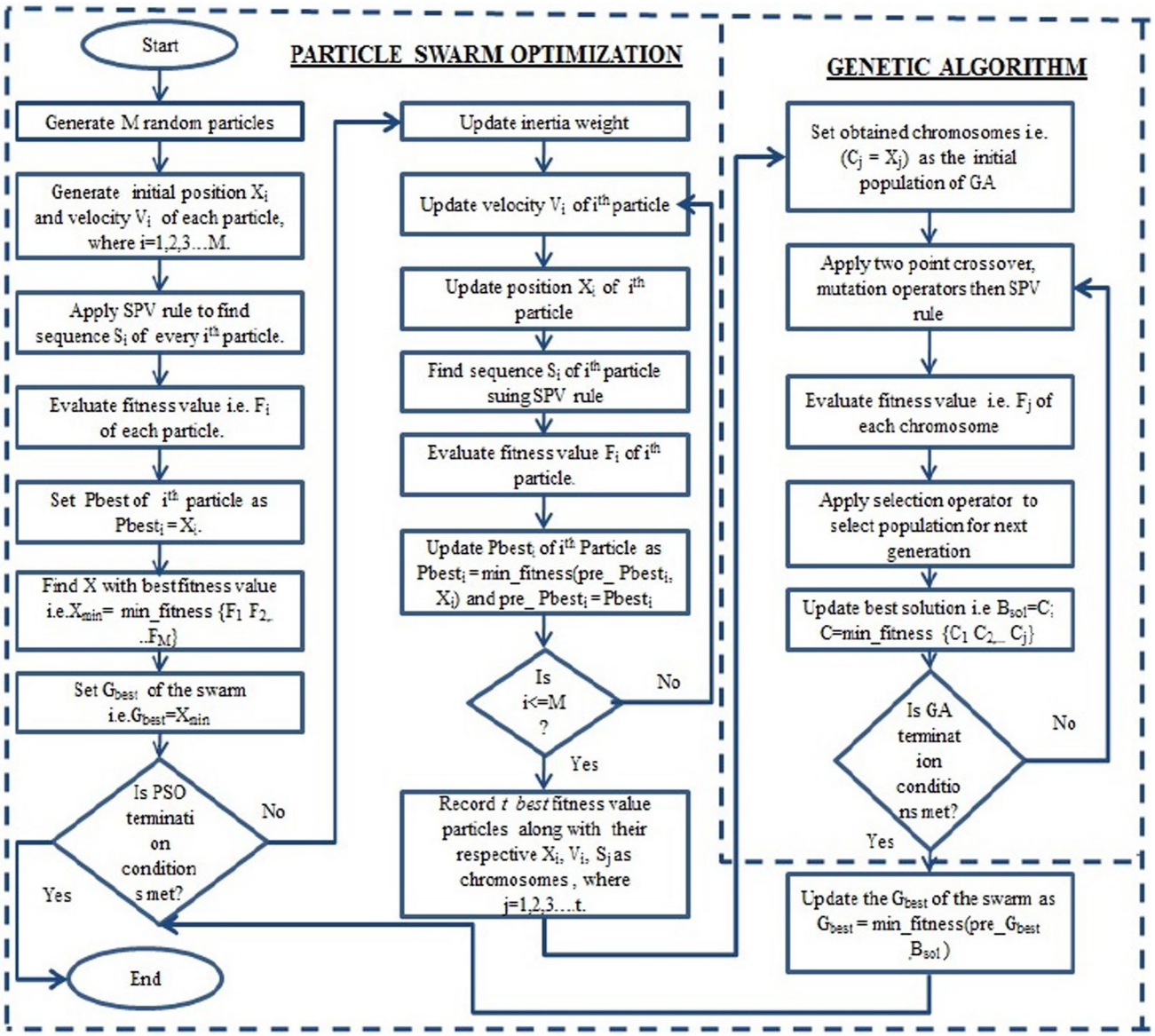

Fig. 3 Flowchart of the proposed hybrid PSO–GA

3.2.2 *Other related terms*

Few other important related terms that may not belong to general PSO but specially used in this meta-heuristic, are as follows.

Smallest position value (SVP), a heuristic proposed by Tasgetiren et al. [5], is used to convert continuous value vector of PSO into a discrete value vector so that it may apply to all sequencing class kind of problems. This concept is similar to the random keys concept proposed by Bean [31] for genetic algorithm. With this heuristic, it is easy to convert the continuous position value vector of wandering particles into discrete activity vector. Conclusively, this heuristic finds the discrete value sequence vector i.e. $\overrightarrow{S}$ by sorting particle's continuous value position vector $\vec{X}$ in ascending order. The detailed description for SPV is given in [1, 5] and the pseudo-code is as follows.

$\boldsymbol{SPV}(\boldsymbol{X_{ij}^k})$

{
Sort X_{ij}^k in ascending order
Enumerate S_{ij}^k with discrete values where $S_{ij}^k = \text{dimention}\left(X_{ij}^k\right)$
}

A demonstration of SPV rule is given in Table 1 in which the values corresponding to S_{ij}^k represent the ascending

Table 1 Particle encoding representation

J	1	2	3	4	5	6	7
X_{ij}^0	2.12	0.54	1.56	2.42	0.94	0.34	3.3
V_{ij}^0	0.34	−0.84	−1.83	1.34	3.83	−3.2	0.92
S_{ij}^0	6	2	5	3	1	4	7

order of the activities of the *i*th particle in *j*th dimension at *k*th iteration corresponding to their position values.

Critical path length CP length is a source to sink node path having highest makespan [1, 5] as represented in Eq. 1. The motive of CP length is to provide a bound to the optimal solution [1, 20].

$$\text{CP length} = \sum W_j \tag{1}$$

W_j is processing time of task j belonging to the critical path, $j \subset N$ and N is the number of tasks in the directed acyclic graph (DAG). To parallelize this DAG, at least M minimum number of processors is required which is obtained using Eq. 2.

$$M = \frac{\sum W_i}{CPlength} \quad 1 \le i \le N \tag{2}$$

According to Eqs. 1 and 2, CP length is equal to the optimal schedule if there is M minimum number of processors available and the communication cost is negligible.

3.2.3 Initial particle generation

Initially, position and velocity of the particles is generated randomly. X_{ij}^0 depicts the position vector for the *i*th particle corresponding to *j*th dimension at 0th iteration and is generated using Eq. 3.

$$X_{ij}^0 = X_{\min} + \left(X_{\max} - X_{\min}\right) * r \tag{3}$$

Where, $X_{\min}$ and $X_{\max}$ have values 0.0 and 4.0, respectively to make the procedure random and r takes uniform random values between 0 and 1 as given in literature [1, 5].

The velocity vector for *i*th particle corresponding to *j*th dimension at 0th iteration is generated using Eq. 4.

$$V_{ij}^0 = V_{\min} + \left(V_{\max} - V_{\min}\right) * r \tag{4}$$

Where, $V_{\min}$ and $V_{\max}$ have values −4.0 and 4.0, respectively and r is defined as uniform random value between 0 and 1. These values are taken for randomization purpose as given in [1, 5].

Sequencing vector $\overrightarrow{S_i^0}$ denotes the continuous position vector value of each particle i which is converted into discrete value permutation vector using SPV rule. Fitness, F_i of the *i*th particle, is evaluated using fitness function (Sect. 3.4). Personal best is initialized for each i particle i.e., $Pbest_i = X_i$. Global best is initialized as $G_{\text{best}} = X_b; b = \text{argmin}_i\{F_i\}$.

A representation of encoding of *i*th particle with *j*th dimensions is shown in Table 1. Where $1 \le j \le (N = 7)$. First row of the table represents the node numbers of the DAG. Second and third rows represent the position and velocity values generated randomly corresponding to the nodes of the DAG, respectively. Last row of the table represents the sequence position vector, produced using SPV rule. According to the SPV rule, position vector $\left(X_{i,j}^0\right)$ is sorted in ascending order corresponding to its values and SPV (S_{ij}^0) is the sequences of nodes with respect to sorted indexed (*j*th) nodes (tasks) as shown in Table 1. This sequence vector, as a solution, is evaluated using fitness function (Sect. 3.4) considering all problem constraints. The values in SPV (S_{ij}^0) represent the node numbers of the DAG.

3.2.4 PSO updating rules

After initialization, a predefined number of iterations are performed in which the particles evolve to achieve optimal solution. According to standard PSO procedure, updating rules for inertia weight and velocity, position of the particles are required. Updating rules used in this work, are as follows [1, 5].

Inertia weight updating rule: ω^k at *k*th iteration is updated using Eq. 5 as follows.

$$\omega^k = \omega^{k-1} * \alpha \tag{5}$$

Where, ω is predefined as $\omega = 0.9$ and α is a decrementing factor randomly generated between 0 and 1.

Velocity vector updating rule: The velocity at *k*th iteration is updated using Eq. 6.

$$\begin{aligned}\overrightarrow{V_k} = \omega * \overrightarrow{V_{k-1}} + c_1 r_1 \left(\overrightarrow{P\text{best}_{k-1}} - \overrightarrow{X_{k-1}}\right) \\ + c_2 r_2 \left(\overrightarrow{G_{\text{best}}} - \overrightarrow{X_{k-1}}\right)\end{aligned} \tag{6}$$

Where, c_1 and c_2 are self-recognition and social constant, respectively and r_1, r_2 are uniform random number between 0 and 1.

Position vector updating rule: At *k*th iteration, position of the particle is updated using Eq. 7.

$$\overrightarrow{X_k} = \overrightarrow{X_{k-1}} + \overrightarrow{V_k} \tag{7}$$

Table 2 GA encoding

Chromosome	0.6	2.4	3.6	0.4	0.65	0.86	0.77	0.78	0.09	0.1

3.3 GA-related terms and operators

This section briefly introduces GA, its related terms and operators specific to the proposed hybrid PSO-GA meta-heuristic.

GA is a well-known search technique applied for combinatorial problems in search of optimal solution [18, 21]. It is based on the principal of evolution and natural genetics. It works efficiently for large search space. It starts with a set of initial random generated solutions called initial population consisting of chromosomes. Each chromosome represents a potential solution to the specified problem and is composed of string of genes. Genes may be represented using binary {0, 1}, integer or real values depending upon the applications. Genetic operators such as crossover, mutation, and selection are iteratively applied to the population. Over the number of generation, GA converges to a near optimal solution.

Encoding Input for the GA component of the proposed PSO–GA meta-heuristic is the position vector (real values) of PSO-generated solution. This encoding is shown in Table 2. A chromosome for 10 real values is shown depicting the particle values at the corresponding position.

Crossover operator The work uses a random crossover which generates new 'individuals' by combining the portions of the parents' (solutions) genetic material [18]. A random two-point crossover is used and all the positions of string between randomly generated two points of both the parents are remapped. The overall procedure is shown in Fig. 4.

An illustration of the random crossover operator, which consists of three steps, is given in Fig. 5. In step 1, two cutting points are randomly selected for the substrings str1 and str2 from both the parents. Sorting Priorities (SP) corresponding to the values of substrings is generated in increasing order. The values of substring str1 are exchanged according to the sorted priority of str2 and vice versa as shown in step 2. In step 3, output offspring v1' and v2' are obtained.

Mutation Operator Simple swap mutation operator is used in this work. Two positions are randomly selected and their contents are swapped as shown in Fig. 7. This procedure guarantees to generate legal offspring. The swap mutation operator usually converts less effective solution to more effective solution. The overall processor is shown in Fig. 6.

Selection Roulette wheel selection [18, 22] is used in this meta-heuristic. All solutions are placed on roulette wheel where better solution has larger portion on the wheel. This gives a fair chance to each individual to be a potential parent in proportion to their fitness value. Best probability of selecting a parent is generated by a spinning roulette wheel along with the size of its slots for every parent. Obviously, parents with bigger fitness value (larger slot sizes) have higher probability to be chosen.

```
Procedure: Crossover
Input: parent v1,v2, number of tasks N
Output: offspring: v1', v2'
begin
        Point1 = random[1, N-1];
        Point2 = random[point1,N];
        Length = point2 – point1;
for  i = 1 to length
        str1[i] = v1[Point1+i];
        str2[i] = v2[Point2+i];
str1[.] = sorting(str1[.]);
str2[.] = sorting(str2[.]);
        v1' = v1[1: Point1]// v2[Point1: Point2]// v1[Point2+1:N];
        v2' = v2[1: Point1]// v1[Point1: Point2]// v2[Point2+1:N];
end for
 for  i = 1 to length
    for  j = 1 to length
        if v1'[Point1+i] = str2[j] then
            v1'[Point1+i] = str1[j];
        end if
    end for
    for  j = 1 to length
        if v2'[Point1+i] = str1[j] then
            v2'[Point1+i] = str2[j];
        end if
    end for
end for
end
```

Fig. 4 Random crossover procedure [18]

3.4 Fitness function

The motive of the work is to optimize the scheduling length of the given problem represented as DAG. Hence, the solution can be scored by makespan which is the complete scheduling length as defined in Eq. 8.

$$\text{makespan} = \max\left(\text{FT}(n)_{i,j}\right) \tag{8}$$

Where, $\text{FT}(n)_{i,j}$ represents the finish time of task n_i on best suitable processor p_j and $1 \leq i \leq N$, $1 \leq j \leq M$.

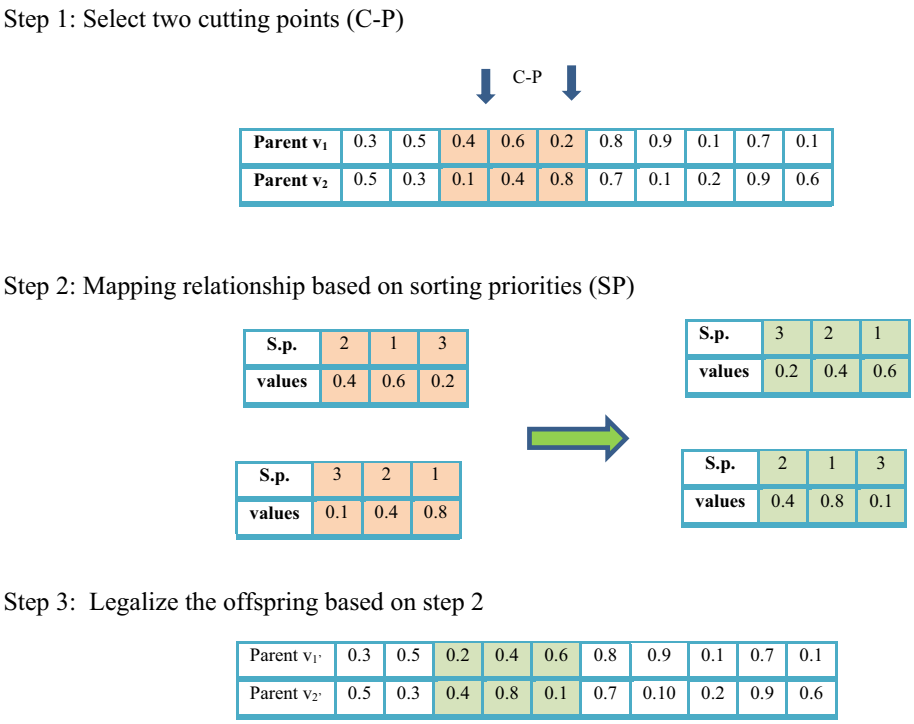

Fig. 5 An Illustration of random crossover operator

Procedure: Swap Mutation
Input: chromosome v, number of task N
Output: v'
begin
x = **random**[1,N-1];
y = **random**[x+1,N];
v' = v[1:x-1]//v[y]//v[x+1:y-1]//v[x]//v[y+1:N];
end

Fig. 6 Swap mutation operator [18]

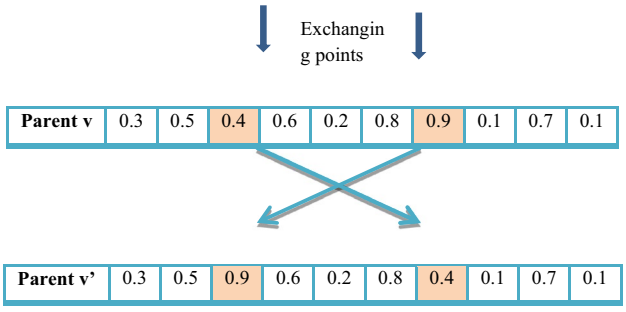

Fig. 7 Illustration of the swap mutation operator

This fitness function evaluates the solution using serial schedule scheme (SSS) [32]. The SSS is used when the generated solutions i.e., particles or chromosomes are invalid (not following dependency constraint). Basically, SSS schedules all the activities sequentially until a valid or feasible solution is achieved. The detailed description of the SSS procedure is given in [32].

4 Experimental studies

This section details the performance of the PSO–GA hybrid meta-heuristic applied for task scheduling in multiprocessor. It has been compared with some other heuristics as proposed in literature [2, 6–14, 25]. The compared nine heuristics/meta-heuristics are: Min–Min, Chaining, A*search, Simulated Annealing, Tabu Search, HLFET, ISH, GA and PSO. The DAG for job/task is generated for two standard linear algebra problems. Some random DAG is also used. The proposed PSO–GA meta-heuristic is simulated in Matlab.

4.1 Test bed

There are no commonly used and standard tasks set benchmarks available to study the performance of the heuristics/meta-heuristics on task scheduling problems [23]. Researchers have often used random task graphs of known optimal schedules [1, 23]. For effective comparative study, two well-known standard problems of linear algebra; LU decomposition [2] and Gauss–Jordan elimination (GJE) [2, 24] have been used. These are tested on

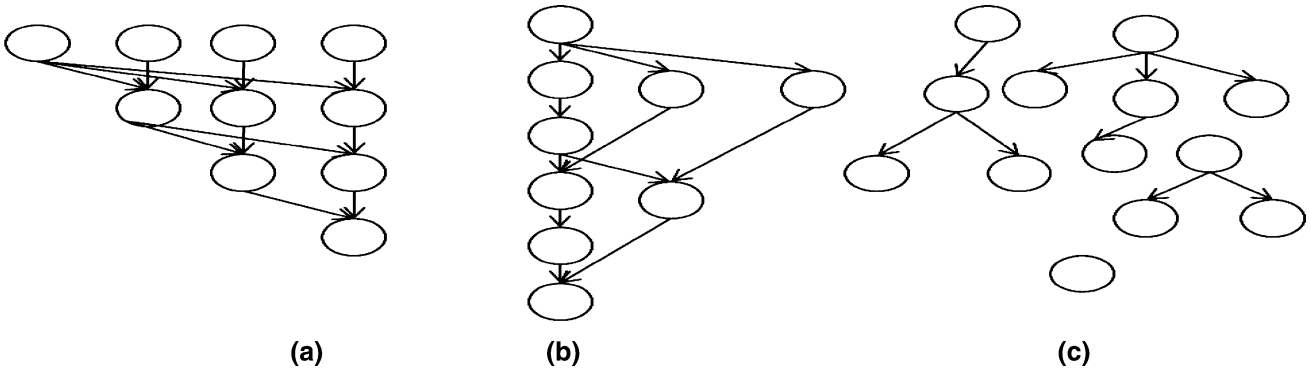

Fig. 8 Pictorial representation of GJE, LU decomposition and random graphs

Table 3 GJE task graphs experimental details

Number of tasks (N)	15	21	28	36
Processing time	40/task	40/task	40/task	40/task
Communication cost	100/edge	100/edge	100/edge	100/edge
Processors (M)	4	4	4	4
Population size	30	42	56	72
PSO-iterations	50	50	50	50
GA-iterations	10	10	10	10
C_1	2	2	2	2
C_2	2	2	2	2
ω	0.9	0.9	0.9	0.9
Crossover rate	0.7	0.7	0.7	0.7
Mutation rate	0.15	0.15	0.15	0.15

Table 4 LU decomposition task graphs experimental details

Number of tasks (N)	14	20	27	35
Processing time	10 s bottom layer task, plus 10 s for every layer			
Communication cost	80/edge	80/edge	80/edge	80/edge
Processors (M)	4	4	4	4
Population size	28	40	54	70
PSO-iterations	50	50	50	50
GA-iterations	10	10	10	10
C_1	2	2	2	2
C_2	2	2	2	2
ω	0.9	0.9	0.9	0.9
Crossover rate	0.7	0.7	0.7	0.7
Mutation rate	0.15	0.15	0.15	0.15

nine heuristics as defined in [1, 2]. Figure 8a, b depict the pictorial representation of LU decomposition and GJE problem.

Further, to study the behavior of the proposed PSO–GA on large sizes of task graphs, few experiments are done on random generated task graphs. Random task graphs are pictorially represented in Fig. 8c.

4.2 Experimental analysis

For the performance analysis of the PSO–GA, two set of experiments have been performed in this section. In the first subsection, the PSO-GA is evaluated on two well-known linear algebra problems i.e., GJE and LU decomposition graphs along with some known optimal solutions of different heuristics/meta-heuristics [1, 2]. Next subsection demonstrates the comparative results of the PSO–GA with PSO [1] and GA [25] on randomly generated graphs.

4.2.1 Benchmark task graphs experiments

This subsection depicts comparative results of the proposed PSO–GA meta-heuristic on GJE and LU decomposition graphs with other available heuristics. The parameters to generate task graphs corresponding to GJE and LU decomposition problems and input parameters of the PSO–GA are given in Table 3 and Table 4, respectively. All the parameters and number of iterations in both the tables are taken from other contemporary meta-heuristics [1, 2] for the purpose of fair comparative study. Inter-process communication cost on the edges of GJE and LU decomposition DAGs is taken same as in the models [1, 2, 23]. For random task graph experiments (Sect. 4.2.2), it is randomly generated between certain ranges as given in Table 5. The constants C_1 and C_2 (self-recognition and social constant, respectively) are taken as 2 against the suggested value of 2.05 in "standard" PSO [27]. The second and third rows details about the processing time (in time unit) taken by each node of the task graph and communication cost (in time unit) of the edges, respectively in both the tables.

Comparative result of all the heuristics with the PSO–GA meta-heuristic is shown in Figs. 9 and 10 corresponding to GJE and LU decomposition graphs, respectively.

Table 5 Input values for random task graphs experiment

Tasks (N)	100	200	500	800	1,000	2,000
Iterations for GA [25]	3,000	3,000	4,000	4,000	5,000	5,000
Iterations for PSO [1]	3,000	3,000	4,000	4,000	5,000	5,000
Iterations for proposed hybrid PSO–GA	PSO-150 GA-20	PSO-150 GA-20	PSO-200 GA-20	PSO-200 GA-20	PSO-250 GA-20	PSO-250 GA-20
Population	150	250	300	300	400	500
Communication cost	10–50	1–50	10–100	10–100	20–150	20–150
Processing time	10–30	10–30	10–60	10–60	20–90	20–90
Processors (M)	4	4	8	8	16	16
$C1$	2	2	2	2	2	2
$C2$	2	2	2	2	2	2
ω	0.9	0.9	0.9	0.9	0.9	0.9
Crossover rate	0.7	0.7	0.7	0.7	0.7	0.7
Mutation rate	0.15	0.15	0.15	0.15	0.15	0.15

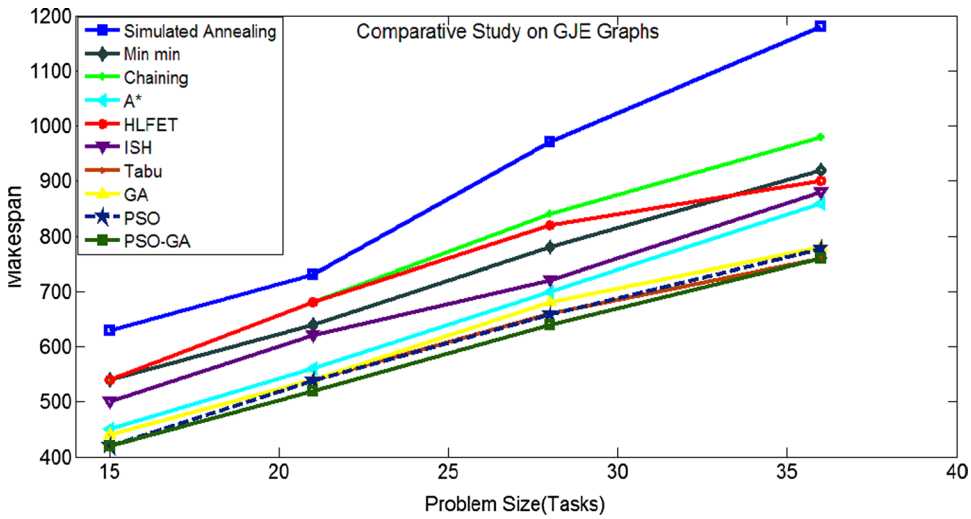

Fig. 9 Comparative study of 10 heuristics on GJE graphs

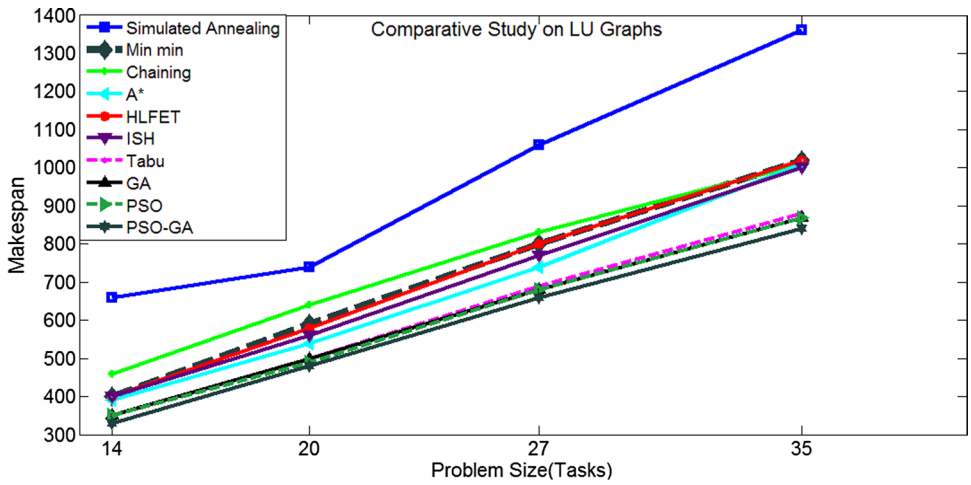

Fig. 10 Comparative study of 10 heuristics on LU decomposition graphs

From the experimental results of Figs. 9 and 10, it is observed that the proposed PSO–GA meta-heuristic outperforms all the other heuristics/meta-heuristics for GJE and LU decomposition problem. The experimental analysis also exhibit that GA and PSO are popular and most competing heuristics to solve large and complex problems. Hence, for

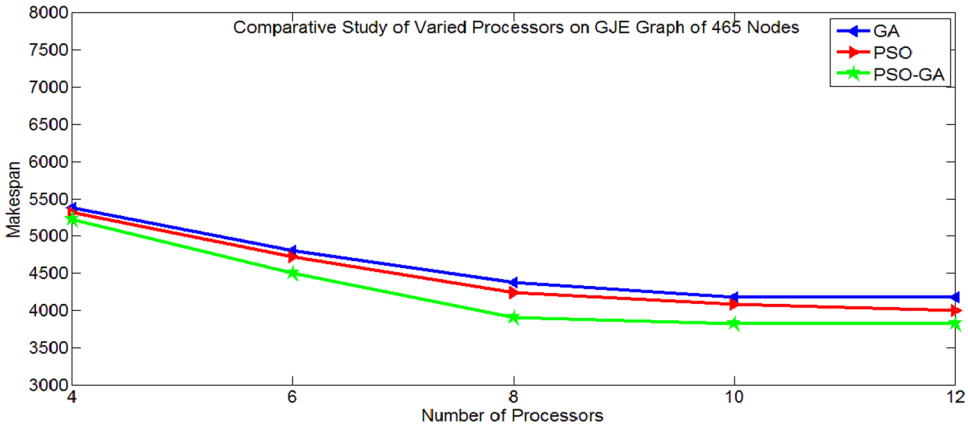

Fig. 11 Comparative study of three meta-heuristics using GJE taskgraph-465 with varied number of processors

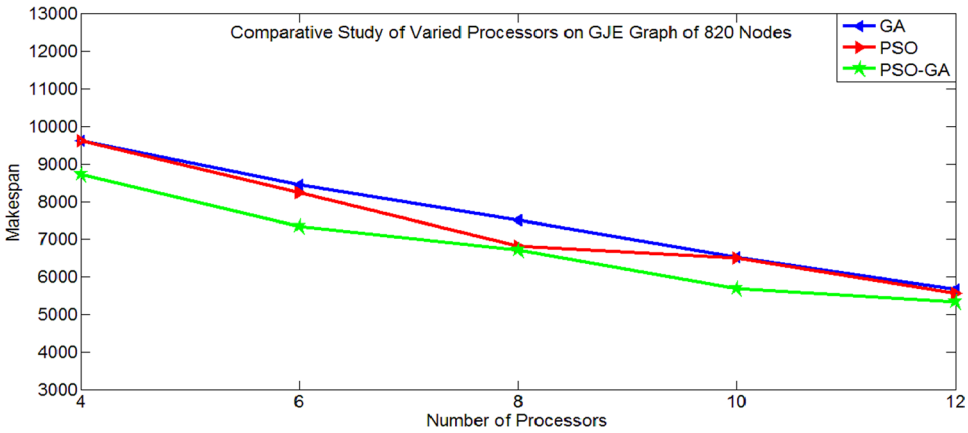

Fig. 12 Comparative study of three meta-heuristics using GJE taskgraph-820 with varied number of processors

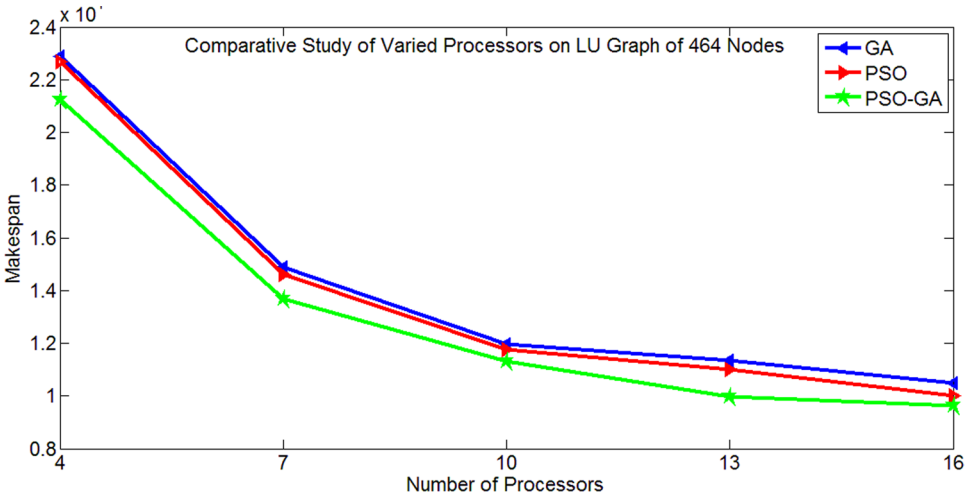

Fig. 13 Comparative study of three meta-heuristics using LU decomposition taskgraph-464 with varied number of processors

further experimentation, we select these two techniques for the comparison purpose.

The experimental results shown in Figs. 9 and 10 are limited to four numbers of processors and same size of problems (i.e. task graphs). This is to make a fair comparative study with other existing state of the arts heuristics [1, 2] as these have been applied for the same configuration. The results shown in the Figs. 9 and 10 are overlapping because of limited problem size and high precedence constraints among the nodes of DAGs. Also

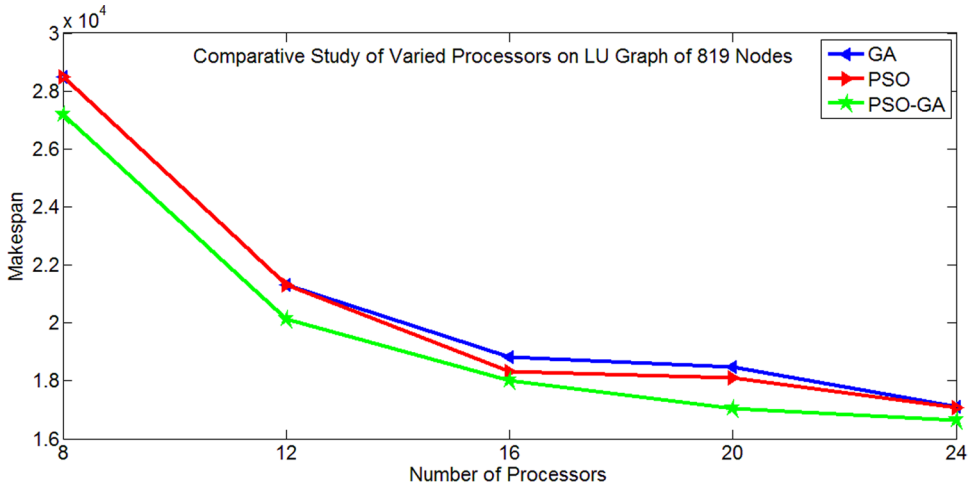

Fig. 14 Comparative study of three meta-heuristics using LU decomposition taskgraph-819 with varied number of processors

results cannot be further optimized increasing the number of processors because of limited problem size and precedence constraints among the nodes and the proposed PSO–GA has improved the quality of the solution to its max using the same configuration.

To further evaluate the performance of the PSO–GA in comparison to the dominating meta-heuristics i.e., GA and PSO, experiments have been performed by varying numbers of processors and using large sizes of JGE and LU decomposition graphs. Results are shown in in Figs. 11, 12, 13 and 14. To do this set of experiments, all the parameters except initial population and numbers of processors for GJE and LU decomposition task graphs are taken from Tables 3 and 4, respectively. For all the experiments, initial population is taken as 150 and numbers of processors are shown in the respective figures.

Figures 11 and 12 shows the performance of the PSO–GA in comparison to PSO and GA with varied number of processors on GJE task graphs of 645 and 820 nodes, respectively. From the results, it is observed that when numbers of processors are increased with respect to the large size of task graphs, the performance of the proposed PSO-GA is rapid improving in comparison to others.

To make it more clear, the impact of increasing numbers of processors on the proposed PSO–GA, an experimental analysis on large sizes of LU decomposition task graphs of 464 and 819 nodes is also done and results are shown in Figs. 13 and 14, respectively. The results infer that the PSO–GA again outperforms other heuristics with the varied number of processors corresponding to large sizes of LU decomposition task graphs.

The Figs. 13 and 14 exhibit the similar pattern as that of small sizes of DAGs. To observe the performance of the model on random task graphs, the next section presents an experimental analysis of the PSO-GA model using random task graphs.

4.2.2 Random task graphs experiment

In this experiment, the performance of the PSO–GA is studied by comparing it with PSO [1] and GA [25] over the large size random DAGs. The PSO-GA meta-heuristic is also tested by varying the number of processors. Table 5 represents the input parameters for random task graphs and meta-heuristic techniques used in random task graph experiments.

On the basis of the given input parameters (Table 5), six large sizes of random DAGs are generated with 100, 200, 500, 800, 1,000, and 2,000 tasks. Most of the parameters conforms to that of Tables 3 and 4 suitable for heuristics implementation and comparison with various state of the art. For a fair comparison, all the parameters, number of iterations and fitness function etc. are taken to be the same. The comparative results of GA, PSO and PSO–GA for random task graphs are shown in Fig. 15.

The parameters are taken for these experiments are shown in Table 5. From the Fig. 15, it is observed that the performance of the PSO–GA is far better than other meta-heuristics. This set of experiments also proves that the proposed hybrid PSO–GA meta-heuristic effectively outperforms PSO [1] and GA [25] in all the cases.

5 Conclusion

The paper proposes a hybrid PSO–GA meta-heuristic using PSO and GA meta-heuristics to solve the multiprocessor DAG scheduling problem using homogenous multiprocessor system. In the proposed PSO–GA meta-heuristic, PSO works as a solution builder and GA works as a solution refiner. Novelty of the proposal is the way GA is used inside the PSO i.e., good generated PSO solutions are further refined using GA. Moreover, at each iteration of the PSO, few best of these particles with their updated position

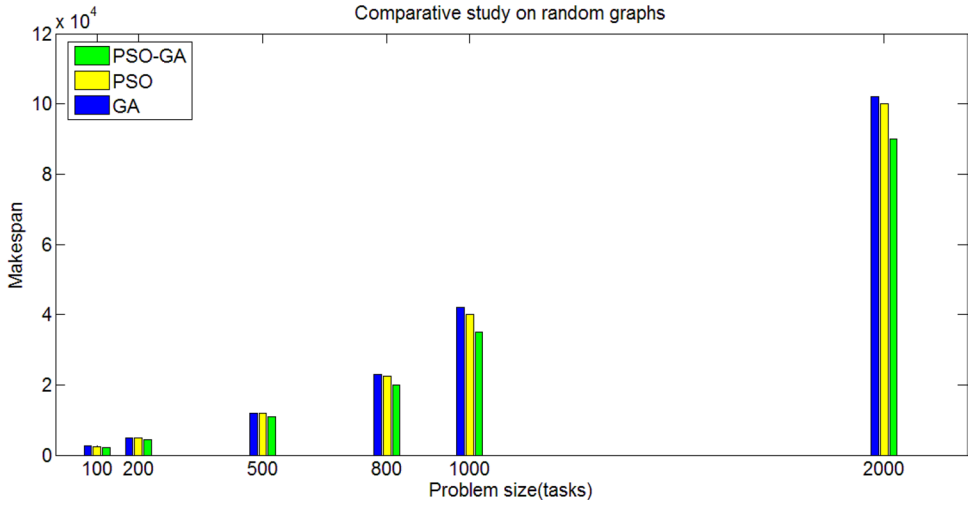

Fig. 15 Comparative Study of three meta-heuristics on random graphs

and velocity is fed to the GA component of the model and refined. Global best solution of PSO is updated using a best solution suggested by GA part of the model. The computational results of the PSO–GA are compared with other heuristics/meta-heuristics [1, 2] on two well-known linear algebra problems; GJE, LU decomposition and random-generated graphs. Initially, the results are compared with nine other states of the art using GJE and LU decomposition tasks graphs and few numbers of processors. From the results, it is observed that the PSO–GA outperforms nine states of the arts heuristics. Further, to test the behavior of the PSO–GA on larger sizes of GJE, LU tasks graphs on large number of processors, some experiments are done. The results are compared with two well-known and most effective meta-heuristics (for large search space problem) i.e., GA and PSO and it is found that the proposed PSO–GA meta-heuristic is effective and scalable to solve the large sizes of problems also. The performance of the PSO–GA is also tested on various large sizes of random tasks graphs with varying number of processors. Again, results proof the effectiveness of the proposed model.

Thus, it is concluded that PSO–GA hybrid meta-heuristic is a promising candidate for multiprocessor task scheduling problem. The proposed PSO–GA meta-heuristic can also be applied to solve other larger and complex combinatorial optimization problems effectively.

References

1. Al Badawi A, Shatnawi A (2013) Static scheduling of directed acyclic data flow graphs onto multiprocessors using particle swarm optimization. Comput Oper Res 40:2322–2328
2. Jin Shiyuan, Schiavone Guy, Turgut Damla (2008) A performance study of multiprocessor task scheduling algorithms. J Supercomput 43:77–97
3. Chow YC, Kohler WH (1979) Meta-heuristics for dynamic load balancing in a heterogeneous multiple processor system. IEEE Trans Comput c-28:354–361
4. Sinnen O (2007) Task scheduling for parallel systems, 1st edn. Wiley, Hoboken
5. Tasgetiren MF, Sevkli M, Liang YC, Gencyilmaz G (2004) Particle swarm optimization algorithm for single machine total weighted tardiness problem. Proc IEEE Congr Evol Comput 2:1412–1419
6. Ibarra O, Kim C (1977) Heuristic algorithms for scheduling independent tasks on non-identical processors. J Assoc Comput Mach 24(2):280–289
7. Djordjevic G, Tosic M (1996) A heuristic for scheduling task graphs with communication delays on to multi processors. Parallel Comput 22(9):1197–1214
8. Kafil M, Ahmad I (1997) Optimal task assignment in heterogeneous computing systems. In: Proceedings of 6th Heterogeneous Computing Workshop, HCW 97, pp 135–146
9. Eliasi R, Elperin T, Bar-Cohen A (1990) Monte Carlo thermal optimization of populated printed circuit board. IEEE Trans Compon 13:953–960
10. Tian Y, Sannomiya N, Xu Y, Tabu A (2000) Search with a new neighborhood search technique applied to flow shop scheduling problems. In: Proceedings of the 39th IEEE Conference On Decision And Control. 5: 4606–4611
11. Porto S, Ribeiro C (1995) A tabu search approach to task scheduling on heterogeneous processors under precedence constraints. Int J of High-Speed Comput 7(1):45–71
12. Adam T, Chandy K, Dickson J (1974) A comparison of list schedules for parallel processing systems. ACM Commun 17:685–690
13. Kruatrachue B, Lewis T (1987) duplication scheduling heuristic dsh: a new precedence task scheduler for parallel processing systems, Ph.D. thesis, Oregon State University, Corvallis, OR
14. Kruatrachue B, Lewis T (1998) Grain size determination for parallel processing. IEEE Softw 5(1):23–32
15. Ahmad I, Kwok Y (1998) On exploiting task duplication in parallel program scheduling. IEEE Trans Parallel Distrib Syst 9(9):872–892
16. Akpınara S, Mirac Bayhanb G, Baykasoglub A (2013) Hybridizing ant colony optimization via genetic algorithm for mixed-meta-heuristic assembly line balancing problem with sequence dependent setup times between tasks. Appl Soft Comput 13:574–589

17. Blum C, Puchinger J, Raidl GR, Roli A (2011) Hybrid meta-heuristics in combinatorial optimization: a survey. Appl Soft Comput 11(6):4135–4151
18. Hwang Reakook, Genb Mitsuo, Katayama Hiroshi (2008) A comparison of multiprocessor task scheduling algorithms with communication costs. Comput Oper Res 35:976–993
19. Kennedy J, Eberhart RC, Shi Y (2001) Swarm Intelligence, 1st Morgan Kaufmann, San Francisco, USA
20. Shatnawi A, Ahmad MO, Swamy MNS (2002) Optimal scheduling of digital signal processing data-flow graphs using shortest-path algorithm. Br Comput Soc 45(1):88–100
21. Vidyarthi DP, Sarker BK, Tripathi AK, Yang LT (2009) Scheduling in distributed computing systems, Springer, ISBN 978-0-387-74480-3, 2009
22. Holland J (1975) Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor
23. Kwok YK, Ahmad I (2003) Benchmarking the task graph scheduling algorithms. In: HKUST
24. Gerasoulis A, Yang T (1994) Performance bounds for column-block partitioning of parallel Gaussian-elimination and Gauss-Jordan methods. Appl Numer Math 16:283–297
25. Hou E, Ansari N, Ren H (1994) A genetic algorithm for multiprocessor scheduling. IEEE Trans Parallel Distrib Syst 5(2):113–120
26. Premalatha K, Natarajan AM (2009) Hybrid PSO and GA for global maximization. Int J Open Problems Comput Math 2(4):597–608
27. Bratton D, Kennedy J (2007) Defining a standard for particle swarm optimization. In: 2007 IEEE warm intelligence symposium, SIS, pp 120–127
28. Juang C-F (2004) A hybrid of genetic algorithm and particle swarm optimization for recurrent network design. IEEE Trans Syst Man Cybern: Part B Y Bernetics 34(2):997–1005
29. Kao Yi-Tung, Zahara Erwie (2008) A hybrid genetic algorithm and particle swarm optimization for multimodal functions. Appl Soft Comput 8:849–857
30. Shi XH, Liang YC, Lee HP, Lu C, Wang LM (2005) An improved GA and a novel PSO-GA-based hybrid algorithm. Inf Process Lett 93:255–261
31. Bean JC (1994) Genetic algorithms and random keys for sequencing and optimization. ORSA J Comput 6(2):154–180
32. Jia Qiong, Seo Yoonho (2013) An improved particle swarm optimization for the resource-constrained project scheduling problem. Int J Adv Manuf Technol 67:2627–2638
33. Premalatha K, Natrajan AM (2009) Hybrid PSO and GA for global maximization. Int J Open Problems Compt Math 2(4):597–608 ISSN 1998-6262