Expert Systems with Applications 38 (2011) 10787–10793

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Metaheuristic methods in hybrid flow shop scheduling problem

F. Choong [*], S. Phon-Amnuaisuk, M.Y. Alias

Taylor's University Lakeside Campus, School of Engineering, No. 1, Jalan Taylor's, 47500 Subang Jaya, Selangor, Malaysia

A R T I C L E I N F O

Keywords:
Memetic algorithms
Particle swarm optimization
Simulated annealing
Tabu search

A B S T R A C T

Memetic algorithms are hybrid evolutionary algorithms that combine global and local search by using an evolutionary algorithm to perform exploration while the local search method performs exploitation. This paper presents two hybrid heuristic algorithms that combine particle swarm optimization (PSO) with simulated annealing (SA) and tabu search (TS), respectively. The hybrid algorithms were applied on the hybrid flow shop scheduling problem. Experimental results reveal that these memetic techniques can effectively produce improved solutions over conventional methods with faster convergence.

1. Introduction

Multiprocessor task scheduling is a generalized form of classical machine scheduling where a task is processed by more than one processor. It is a challenging problem encountered in wide range of applications and it is vastly studied in the scheduling literature (Chan & Lee, 1999; Drozdowski, 1996) for a comprehensive introduction on this topic). However, Drozdowski (1996) showed that multiprocessor task scheduling is difficult to solve even in its simplest form. Hence, many heuristic algorithms were presented in the literature to tackle multiprocessor task scheduling problem. Jin, Schiavone, and Turgut (2008) presented a performance study of such algorithms. However, most of these studies are primarily concerned with a single stage setting of the processor environment. There are many practical problems where multiprocessor environment is a flow-shop made up of multiple stages and tasks have to go through one stage to another.

Flow-shop scheduling problem is also vastly studied in scheduling context though most of these studies concerned with single processor at each stage (Dauzère-Pérès & Paulli, 1997; Linn & Zhang, 1999). With the advances made in technology, in many practical applications, we encounter parallel processors at each stage instead of single processors such as parallel computing, power system simulations, operating system design for parallel computers, traffic control in restricted areas, manufacturing and many others (Caraffa, Ianes, Bagchi, & Sriskandarajah, 2001; Krawczyk & Kubale, 1985; Lee & Cai, 1999). This particular problem is defined as hybrid flow-show with multiprocessor tasks in scheduling terminology and minimizing the schedule length (makespan) is the typical scheduling problem addressed. However, Brucker and Kramer (1995) showed that multiprocessor flow-shop problem to minimize makespan is also NP-hard. Gupta (1988) showed that hybrid flow-shop even with two stages is NP-hard. Furthermore, the complexity of the problem increases with the increasing number of stages.

Multiprocessor task scheduling in a hybrid flow-shop environment has recently gained the attention of the research community. However, due to the complexity of the problem, in the early studies (Lee & Cai, 1999; Oðuz, Ercan, Cheng, & Fung, 2003) researchers targeted two layer flow-shops with multiprocessors. Simple list based heuristics as well as meta-heuristics were introduced for the solution (Jdrzêjowicz & Jdrzêjowicz, 2003; Oðuz et al., 2003, 2004). Apparently, a broader form of the problem will have arbitrary number of stages in the flow-shop environment. This is also studied recently and typically metaheuristic algorithms applied to minimize the makespan such as population learning algorithm (Jdrzêjowicz & Jdrzêjowicz, 2003), tabu search (Oðuz et al., 2004), genetic algorithm (Oðuz et al., 2003) and ant colony system (Ying & Lin, 2006). Minimizing the makespan is not the only scheduling problem tackled; recently Shiau, Cheng, and Huang (2008) focused on minimizing the weighted completion time in proportional flow shops.

These metaheuristic algorithms produce impressive results though they are sophisticated and require laborious programming effort. However, of late particle swarm optimization (PSO) is gaining popularity within the research community due to its simplicity. The algorithm is applied to various scheduling problems with notable performance. For instance, Sivanandam, Visalakshi, and Bhuvaneswari (2007) applied PSO to typical task allocation problem in multiprocessor scheduling. Chiang, Chang, and Huang (2006) and Tu, Hao, and Chen (2006) demonstrate application of PSO to well known job shop scheduling problem.

PSO, introduced by Kennedy and Eberhart (1995), is another evolutionary algorithm which mimics the behavior of flying birds and their communication mechanism to solve optimization

* Corresponding author.
E-mail address: florence.choong@gmail.com (F. Choong).

0957-4174/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2011.01.173

problems. It is based on a constructive cooperation between particles instead of survival of the fittest approach used in other evolutionary methods. PSO has many advantages therefore it is worth to study its performance for the scheduling problem presented here. The algorithm is simple, fast and very easy to code. It is not computationally intensive in terms of memory requirements and time. Furthermore, it has a few parameters to tune.

This paper presents the hybrid flow-shop with multiprocessor tasks scheduling problem and PSO algorithm proposed for the solution in details. Next, two hybrid heuristic algorithms that combines PSO with simulated annealing (SA) and tabu search (TS), respectively are presented. This paper also introduces other well known heuristics which are reported in literature for the solution of this problem. Finally, a performance comparison of these algorithms will be given.

2. Problem definition

The problem considered in this paper is formulated as follows: there is a set J of n independent and simultaneously available jobs where each job is made of Multi-Processor Tasks (MPT) to be processed in a multi-stage flow-shop environment, where stage j consists of m_j identical parallel processors ($j = 1, 2, \ldots, k$). Each $MPT_i \in J$ should be processed on $p_{i,j}$ identical processors simultaneously at stage j without interruption for a period of $t_{i,j}$ ($i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, k$). Hence, each $MPT_i \in J$ is characterized by its processing time, $t_{i,j}$, and its processor requirement, $p_{i,j}$. The scheduling problem is basically finding a sequence of jobs that can be processed on the system in the shortest possible time. The following assumptions are made when modeling the problem:

- All the processors are continuously available from time 0 onwards.
- Each processor can handle no more than one task at a time.
- The processing time and the number of processors required at each stage are known in advance.
- Set-up times and inter-processor communication time are all included in the processing time and it is independent of the job sequence.

3. Algorithms

3.1. The basic PSO algorithm

PSO is initialized with a population of random solutions which is similar in all the evolutionary algorithms. Each individual solution flies in the problem space with a velocity which is adjusted depending on the experiences of the individual and the population. As mentioned earlier, PSO and its hybrids are gaining popularity in solving scheduling problems. A few of these works have tackled the flow shop problem (Liu et al., 2005) though application to hybrid flow-shops with multiprocessor tasks is relatively new (Tseng & Liao, 2008).

In this study, we first employed the global model of the PSO. In the basic PSO algorithm, particle velocity and position are as shown in Eqs. (1) and (2):

$$V_{id} = WV_{id} + C_1R_1(P_{id} - X_{id}) + C_2R_2(P_{gd} - X_{id}) \qquad (1)$$

$$X_{id} = X_{id} + V_{id} \qquad (2)$$

In the above equations, V_{id} is the velocity of particle i and it represents the distance travelled from the current position. W is inertia weight. X_{id} represents particle position. P_{id} is the local best solution (also called as "pbest") and P_{gd} is global best solution (also called as "gbest"). C_1 and C_2 are acceleration constants which drive particles towards local and global best positions. R_1 and R_2 are two random numbers within the range of [0, 1]. This is the basic form of the PSO algorithm which follows the following steps in Fig. 1.

The initial swarm and particle velocity are generated randomly. A key issue is to establish a suitable way to encode a schedule (or solution) to PSO particle. We employed the method shown by Xia and Wu (2006). Each particle consists of a sequence of job numbers representing the n number of jobs on a machine with k number of stages where each stage has m_j identical processors ($j = 1, 2, \ldots, k$). The fitness of a particle is then measured with the maximum completion time of all jobs. A list scheduling algorithm is developed to map a given job sequence to the machine environment and to compute the maximum completion time (makespan). A particle with the lowest completion time is a good solution.

Fig. 2 shows an example to scheduling done by the list scheduling algorithm. In this example, number of jobs is $n = 5$ and a machine is made of two stages $k = 2$ where each stage contains four identical processors. Table 1 depicts the list of jobs and their processing times and processor requirements at each stage for this example.

For the schedule shown in Fig. 2, it is assumed that a job sequence is given as $S_1 = \{2, 3, 1, 4, 5\}$. At stage 1, jobs are iteratively allocated to processors from the list starting from time 0 onwards. As job 2 is the first in the list, it is scheduled at time 0. It is important to note that although there are enough available processors to schedule job 1 at time 0 this will violate the precedence relationship established in the list. Therefore, job 1 is scheduled to time instance 3 together with job 3 and this does not violate the precedence relationship given in S_1. Once all the jobs are scheduled at first stage, a new list is produced for the succeeding stage based on the completion of jobs at previous stage and the precedence relationships given in S_1. In the new list for stage 2, $S_2 = \{2, 1, 3, 4, 5\}$, job 1 is scheduled before job 3 since it is available earlier than job 3. At time instance 7, jobs 3, 4 and 5 are all available to be processed. Job 3 is scheduled first since its completion time is earlier at stage 1. Although, there is enough processor to schedule job 5 at time 8 this will again violate the order given in list S_1, hence it is scheduled together with job 4. In this particular

```
Initialize swarm with random positions and velocities;
begin
repeat
    For each particle evaluate the fitness i.e. makespan of the schedule;
    if current fitness of particle is better than Pid then set Pid to current value;
    if Pid is better than global best then set Pgd to current particle fitness value;
    Change the velocity and position of the particle;
until termination = True
end.
```

Fig. 1. The basic PSO.

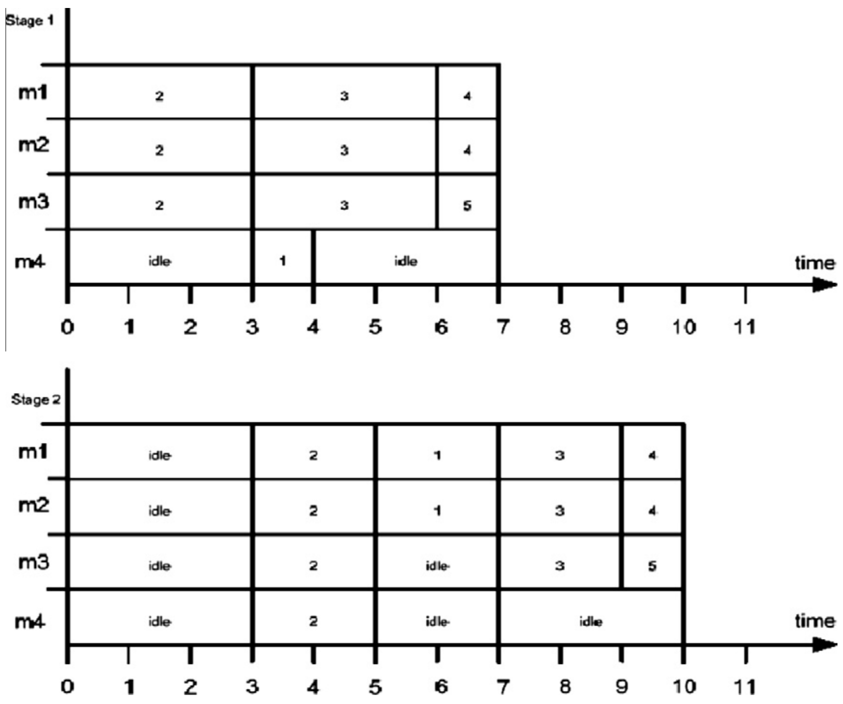

Fig. 2. The schedule of job sequence [2, 3, 1, 4, 5] after being allocated to processors of the multilayer system using the list scheduling algorithm. Idle periods of the processors are labeled as idle.

Table 1
Example jobs and their processing time and processor requirements at each stage.

Job #	Stage 1 ($j = 1$)		Stage 2 ($j = 2$)	
i	$p_{i,1}$	$t_{i,1}$	$p_{i,2}$	$t_{i,2}$
1	1	1	2	2
2	3	3	4	2
3	3	3	3	2
4	2	1	2	1
5	1	1	1	1

example, jobs 4 and 5 will be the last to be mapped to stage 2 and the over all completion time of tasks will be 10 units.

The parameters of PSO are set based on our empirical study as well as referring to the experiences of other researchers. The acceleration constants C_1 and C_2 are set to 2.0 and initial population of swarm is set to 100. Inertia weight, W, determines the search behavior of the algorithm. Large values for W facilitate searching new locations whereas small values provide a finer search in the current area. A balance can be established between global and local exploration by decreasing the inertia weight during the execution of the algorithm. This way PSO tends to have more global search ability at the beginning and more local search ability towards the end of the execution. In our PSO algorithm, an exponential function is used to set the inertia weight and it is defined as shown in Eq. (3):

$$W = W_{end} + (W_{start} - W_{end})e^{\frac{x\alpha}{x_{max}}} \tag{3}$$

where W_{start} is the starting, W_{end} is the ending inertia values. W_{start} are W_{end} are set as 1.5 and 0.3, respectively. In addition, x shows the current iteration number and x_{max} shows the maximum iteration number which is set to 10,000. An integer constant α is used to manipulate the gradient of the exponentially decreasing W value and it is set to 4.

In this application, X_{id} and V_{id} are used to generate and modify solutions therefore they are rounded off to the nearest integer and limited to a maximum value of n which is the maximum number of jobs. That is position coordinates are translated into job sequence in our algorithm and a move in search space is obtained by modifying the job sequence.

3.2. Hybrid PSO algorithm

Although, PSO is very robust and has a well global exploration capability, it has the tendency of being trapped in local minima and slow convergence. In order to improve its performance, many researchers experimented with hybrid PSO algorithms. Poli, Kennedy, and Blackwell (2007) give a review on the variations and the hybrids of PSO. Similarly, in scheduling problems, performance of PSO can be improved further by employing hybrid techniques. For instance, Xia and Wu (2006) applied PSO-SA hybrid to job shop scheduling problem and tested its performance with benchmark problems. The authors conclude that PSO-SA hybrid delivered equal solution quality as compared to other metaheuristic algorithms though PSO-SA offered easier modeling, simplicity and ease of implementation. These findings motivated us to apply PSO and its hybrids to this particular scheduling problem and study its performance.

The basic idea of the hybrid algorithms presented here is simply based on running PSO algorithm first and then improving the result by employing a SA or TS heuristics. SA and TS introduce a probability to avoid becoming trapped in a local minimum. In addition, by introducing a neighborhood formation and tuning the parameters, it is also possible to enhance the search process. The pseudo codes for the hybrid algorithms are shown in Figs. 3 and 4.

The initial temperature for PSO-SA hybrid is estimated after 50 randomly permuted neighborhood solutions of the initial solution. A ratio of average increase in the cost to acceptance ratio is used as initial temperature. Temperature is decreased using a simple cooling strategy $T_{current} = \lambda T_{current} - 1$. The best value for lambda is experimentally found and set as 0.998. The end temperature is set to 0.01. A neighbor of the current solution is obtained in various ways.

```
Initialize swarm with random positions and velocities;
  begin
  initialize PSO and SA;
    while (termination !=true)
    do{
     generate swarm;
     compute and find best Pgd;
     }
  set particle that gives best Pgd as initial solution to SA;
     while (Tcurrent>Temp_end)
      do{
      generate neighborhood;
      evaluate and update best solution and temperature;
      }
  end.
```

Fig. 3. Hybrid PSO with SA.

```
Initialize swarm with random positions and velocities;
  begin
  initialize PSO and TS;
     while (termination !=true)
     do{
     generate swarm;
     compute and find best Pgd;
     }
  set particle that gives best Pgd as initial solution to TS;
     while (termination!=true)
      do{
      generate sub set of neighborhoods;
      evaluate and update the best solution;
      update the tabu list;
      }
   end.
```

Fig. 4. Hybrid PSO with TS.

- *Interchange neighborhood*: Two randomly chosen jobs from the job list are exchanged.
- *Simple switch neighborhood*: It is a special case of interchange neighborhood where a randomly chosen job is exchanged with its predecessor.
- *Shift neighborhood*: A randomly selected job is removed from one position in the priority list and put it into another randomly chosen position.

It is experimentally found that interchange method performs the best amongst all three. The interchange strategy is also found to be the most effective one for generating the sub-neighborhoods for TS.

In the tabu list, a fixed number of last visited solutions are kept. Two methods for updating the tabu list are experimented; elimination of the farthest solution stored in the list, and removing the worst performing solution from the list. In PSO-TS hybrid removing the worst performing solution from the list method is used as it produced a slightly better result.

4. Other heuristic methods

The ant colony system (Dorigo & Gambardella, 1997) is another popular algorithm which is widely used in optimization problems. Recently, Ying and Lin (2006) applied ant colony system (ACS) to hybrid flow-shops with multiprocessors tasks. The authors determine the jobs-permutation at the first stage, by ACS approach. Other stages are scheduled using an ordered list which is obtained by referring to completion times of jobs at the previous stage. The authors also apply the same procedure to the inverse problem to obtain the backward schedules. After that they employed a local search approach to improve the best schedule obtained in current iteration. Their computational results show that ACS has better performance compared to TS or GA though their algorithm is not any simpler than that of TS or GA.

Recently, Tseng and Liao (2008) tackled the problem by PSO. Their algorithm differs in terms of encoding scheme to construct a particle, the velocity equation and local search mechanism when compared to the basic PSO and the hybrid PSO algorithms presented here. Based on their published experimental results, PSO algorithm developed by Tseng and Liao (2008) performs well in this scheduling problem. Lately, Ying (2008) applied iterated greedy (IG) heuristic in search of a simpler and more efficient solution. The IG heuristic also shows a notable performance as it is tailored to this particular problem.

5. Experimental results

The performance of all the meta-heuristics described above is tested using intensive computational experiments. Similarly, performance of the basic PSO and the hybrid PSO algorithms, in minimizing the overall completion time of all jobs, is also tested using the same computational experiments. The effects of various parameters such as number of jobs and processor configurations on the performance of the algorithm are also investigated. The results are presented in terms of Average Percentage Deviation (APD) of the solution from the lower bound as given in Eq. (4):

$$APD = \frac{C_{\max} - LB}{LB} \times 100 \tag{4}$$

Here, C_{max} indicates the completion time of the jobs and LB indicates the lower bound calculated for the problem instance. The lower bounds used in this performance study were developed by Oǒuz, Zinder, Do, Janiak, and Lichtenstein (2004) and it is given with the following Eq. (5):

$$LB = \left\{ \max_{j\epsilon M} \left\{ \min_{j\epsilon J} \sum_{l=1}^{i-1} t_{l,j} \right\} + \frac{1}{m_i} \sum_{j=\epsilon J} p_{i,j} \times t_{i,j} + \min_{j\epsilon J} \left\{ \sum_{l=i+1}^{l} t_{i,j} \right\} \right\} \tag{5}$$

In the above formula, M and J represent the set of stages and set of jobs consecutively. Data set contains instances for two types of processor configurations:

(i) *Random processor*: In this problem set, the number of processors in each stage is randomly selected from a set of $\{1, \ldots, 5\}$.
(ii) *Fixed processor*: In this case identical number of processors assigned at each stage which is fixed to five processors.

For both configurations, a set of 10 problem instances is randomly produced for various number of jobs (n = 5, 10, 20, 50, 100) and various number of stages (k = 2, 5, 8). For each n and k value, the average APD is taken over 10 problem instances. Tables 2 and 3 presents the APD results obtained for the basic PSO and the hybrid PSO algorithms. Furthermore, we compare the results with GA developed by Oǒuz (2005), tabu search by Oǒuz et al. (2004), ant colony system developed by Ying and Lin (2006), iterated greedy algorithm (IG) by Ying (2008) and PSO developed by Tseng and Liao (2008). The performance of GA (Oǒuz et al., 2003) is closely related to the control parameters and the cross over

Table 2
APD of the algorithms for 10 random instances. Random processors case ($m_j \sim [1,5]$).

k	n	TS (Oǧuz et al., 2004)	GA (Oǧuz et al., 2003)	ACS (Ying & Lin, 2006)	IG (Ying, 2008)	Basic PSO	PSO-SA	PSO-TS	PSO[a] (Tseng & Liao, 2008)
2	10	3.0	1.60	1.60	1.49	2.7	1.7	2.1	2.8[a]
	20	2.88	0.80	1.92	1.87	2.88	1.12	1.92	5.40
	50	2.23	0.69	2.37	2.21	2.38	2.4	2.4	2.30
	100	9.07	0.35	0.91	0.89	1.82	0.82	1.1	1.62
5	10	29.42	11.89	9.51	8.73	10.33	9.78	10.4	10.45
	20	24.40	5.54	3.07	2.97	8.6	3.19	4.53	6.04
	50	10.51	5.11	1.51	1.49	3.31	2.06	2.98	1.44
	100	11.81	3.06	1.05	1.03	2.11	1.05	1.77	2.80
8	10	46.53	16.14	16.50	13.91	18.23	16.14	17.5	19.01
	20	42.47	7.98	6.77	5.83	12.03	6.69	7.03	5.76
	50	21.04	6.03	2.59	2.47	5.98	3.0	4.19	2.91
	100	21.50	4.12	1.33	1.23	8.78	2.11	5.22	1.53

[a] Different problem set.

Table 3
APD of the algorithms for 10 random instances. Fixed processor case ($m_j = 5$).

k	n	TS (Oǧuz et al., 2004)	GA (Oǧuz et al., 2003)	ACS (Ying & Lin, 2006)	IG (Ying, 2008)	Basic PSO	PSO-SA	PSO-TS	PSO[a] (Tseng & Liao, 2008)
2	10	10.82	6.13	12.62	8.85	13.8	10.11	12.8	12.75[a]
	20	7.25	7.10	10.73	6.93	10.75	9.59	10.73	6.05
	50	5.80	3.34	8.17	5.66	10.32	7.02	8.82	5.69
	100	5.19	2.87	5.66	5.04	7.43	3.21	6.43	6.56
5	10	45.14	11.32	26.09	23.49	29.6	11.32	22.2	19.58
	20	35.13	10.78	15.11	12.64	19.4	10.77	16.5	12.33
	50	28.64	14.91	13.11	11.29	14.17	13.24	13.86	12.47
	100	26.49	11.02	12.45	10.53	12.45	12.45	12.45	11.49
8	10	77.21	25.98	25.14	22.17	30.81	25.83	25.83	33.92
	20	62.99	24.13	25.18	22.79	26.74	24.34	25.02	24.98
	50	54.25	21.87	22.23	20.71	27.01	23.07	25.11	19.41
	100	36.05	19.46	13.90	12.85	20.39	14.43	17.9	15.93

[a] Different problem set.

and mutation techniques used. Therefore, in Tables 2 and 3, we include the best results obtained from four different versions of GA reported. The performance comparison given in below tables is fair enough as most of the authors were employing the same problem set. Furthermore, all the algorithms use the same LB. However there are two exceptions. For the GA, authors use an improved version of the LB than the one given in Eq. (4). In addition, the PSO developed by Tseng and Liao (2008) is tested with different set of problems and with the same LB as in GA. However, these problems also have the same characteristic in terms of number of stage, generation methods for processor and processing time requirements, etc. From the presented results in Tables 2 and 3, it can be observed that TS delivers reasonably good results only in two stage case; whereas GA demonstrates a competitive performance for small to medium size problems. For large number of jobs (such as $n = 50$, 100) and large number of stages ($k = 8$), GA did not outperform ACS, IG or PSO. When we compare ACS with TS and GA, we can observe that it outperforms TS and GA in most of the cases. For instance, it outperforms GA in 8 out of 12 problems in random processor case (Table 2). Among those, the performance improvement was more than%50 in six cases. On the other hand, IG gives a better result when compared to ACS in all most all the cases. The IG heuristic shows notable performance improvement for large problems ($n = 50$ and 100). For example, in $n = 100$ and $k = 8$ case, IG result is 71% better as compared to GA, 95% compared to TS and 7% compared to ACS.

The basic PSO algorithm presented here approximates to GA and ACS results though it did not show a significant performance improvement. PSO outperformed GA 4 in 12 problems for random processors and 1 in 12 problems for fixed processors. The best performance improvement was 54%. On the other hand, PSO-SA hybrid outperformed GA 7 and ACS 3 in 12 problems. In most of the cases, PSO-SA and PSO-TS outperformed the basic PSO algorithm. Amongst the two hybrids experimented here, PSO-SA gave the best results. The best result obtained with PSO-SA was in 50-jobs, 5-stages case, where the improvement was about 59% when compared to GA but this was still not better than ACS or IG. However, PSO developed by Tseng and Liao (2008) gives much more competitive results. Although their results are for different set of problems, it can be seen that their algorithm performance improves when the problem size increases. Authors compared their algorithm with GA and ACS using the same set of data and reported that their PSO algorithm supersedes them, in particular for large problems. From the results, it can also be observed that when the number of processors are fixed, that is $m_j = 5$, the scheduling problem becomes more difficult to solve and APD results are relatively higher. This is evident in the given results of different metaheuristic algorithms as well as the basic PSO and the hybrid PSO algorithms presented here. In the fixed processor case, PSO-SA, which is the best performing algorithm among the three PSO algorithms, outperformed GA in 3 out of 12 problems and the best improvement achieved was 34%. The performance of ACS is better for large problems though IG is dominant in most of the problems. For the fixed problem case, PSO algorithm developed by Tseng & Liao (2008) did not show an exceptional performance when compared to GA or ACS for smaller problems though for large problems (that is $n = 50$ and 100) their PSO algorithm outperforms all.

The execution time of the algorithms is another indicator of the performance though it may not be a fair comparison as different processors and compilers used for each reported algorithm in literature. For instance, the basic PSO and the hybrid PSO algorithms presented here are implemented using Java language and run on a PC with 2 GHz Intel Pentium processor (with 1024 MB memory). GA (Oðuz et al., 2003) implemented with C++ and run on a PC with 2 GHz Pentium 4 processor (with 256 MB memory), IG (Ying, 2008) with visual C#.net and PC with 1.5 GHz CPU and ACS (Ying & Lin, 2006) with Visual C++ and PC with 1.5 GHz Pentium 4 CPU. However, for the sake of completeness we execute GA, the basic PSO and the hybrid PSO on the same computing platform using one easy ($k = 2$, $n = 10$) and one difficult problem ($k = 8$, $n = 100$) for the same termination criterion of 10,000 iterations for all the algorithms. Results are reported in Table 4, which illustrates the speed performance of PSO. It can be seen that PSO is approximately 48% to 35% faster than reported GA CPU timings. The fast execution time of PSO is also reported by Tseng and Liao (2008). However, in our case hybrid algorithms were as costly as GA due to computations in SA and TS steps.

Lastly, we run the basic PSO and the hybrid PSO algorithms together with GA and TS algorithm for 18 jobs and 12 jobs. As can be seen from Fig. 5, all the algorithms converge very rapidly. However, PSO algorithms converge faster than TS and GA. In addition, PSO-SA finds a slightly better solution for 12 job problem. All the algorithms find a reasonably good solution within 1000 iterations. Hence, for practical application of the scheduling algorithms a small value for termination criteria can be selected.

Table 4
Average CPU time (in seconds) of GA, TS and PSO. Processors setting is $m_j = 5$.

k	n	GA	PSO	PSO-SA	PSO-TS
2	10	12.14	6.3	13.5	12.94
8	100	2109.9	1388.67	4029.1	3816.3

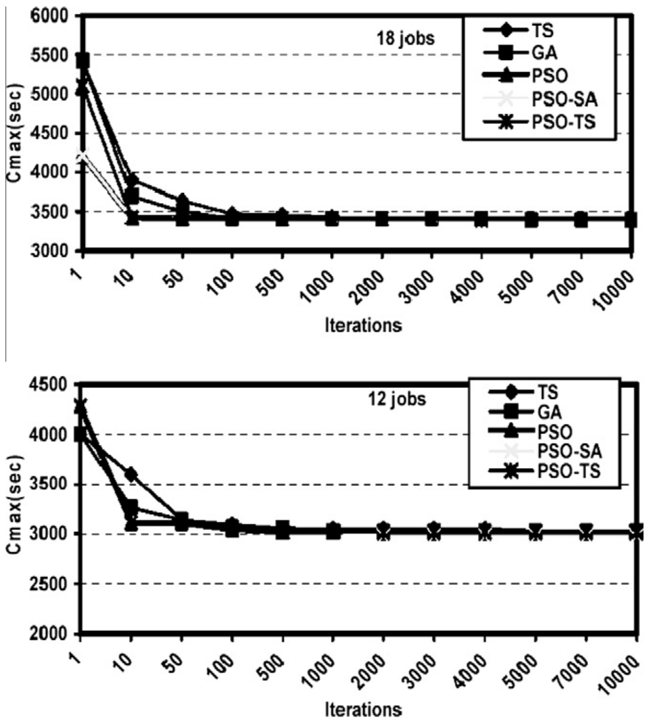

Fig. 5. Convergence of TS, GA and PSO algorithms for 18 jobs and 12 jobs, respectively.

6. Conclusion

In this chapter, a scheduling problem, defined as hybrid flow-shops with multiprocessor tasks, is presented together with various meta-heuristic algorithms reported for the solution in literature. As the solution to this scheduling problem has merits in practice, endeavor to find a good solution is worthy. The basic PSO and the hybrid PSO algorithms are employed to solve this scheduling problem, as PSO proven to be a simple and effective algorithm applied in various engineering problems. In this particular scheduling problem, a job is made up of interrelated multiprocessor tasks and each multiprocessor task is modeled with its processing requirement and processing time. The objective was to find a schedule in which completion time of all the tasks will be minimal. We observe that basic PSO has a competitive performance as compared to GA and ACS algorithms and superior performance when compared to TS. Considering the simplicity of the basic PSO algorithm, the performance achieved is in fact impressive. When experimented with the hybrids of PSO, it is observed that PSO-SA combination gave the best results. Hybrid methods improved the performance of PSO significantly though this is achieved at the expense of increased complexity. When compared to other published results on this problem, it can be concluded that IG algorithm (Ying, 2008) and PSO given by Tseng & Liao (2008) are the best performing algorithms on this problem so far. In terms of effort to develop an algorithm, execution time of algorithm and simplicity to tune it, PSO tops all the other meta-heuristics. As in many practical scheduling problems, it is likely to have precedence constraints among the jobs hence in future study hybrid flow-shops with precedence constraints will be investigated. In addition, PSO may be applied to other scheduling problems and its performance can be exploited in other engineering problems.

References

Brucker, P., & Kramer, B. (1995). Shop scheduling problems with multiprocessor tasks on dedicated processors. *Annals of Operations Research, 50*, 13–27.
Caraffa, V., Ianes, S., Bagchi, T. P., & Sriskandarajah, C. (2001). Minimizing makespan in blocking flow-shop using genetic algorithms. *International Journal of Production Economics, 70*, 101–115.
Chan, J., & Lee, C. Y. (1999). General multiprocessor task scheduling. *Naval Research Logistics, 46*, 57–74.
Chiang, T. C., Chang, P. Y., & Huang, Y. M. (2006). Multi-processor tasks with resource and timing constraints using particle swarm optimization. *International Journal of Computer Science and Network Security, 6*(4), 71–77.
Dauzère-Pérès, S., & Paulli, J. (1997). An integrated approach for modelling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research, 70*, 281–306.
Dorigo, M., & Gambardella, L. M. (1997). Ant colony system: A cooperative learning approach to the travelling sales man problem. *IEEE Transactions in Evolutionary Computing, 1*, 53–66.
Drozdowski, M. (1996). Scheduling multiprocessor tasks – An overview. *European Journal of Operational Research, 94*, 215–230.
Gupta, J. N. D. (1988). Two stage hybrid flow shop scheduling problem. *Journal of Operational Research Society, 39*(4), 359–364.
Jdrzêjowicz, J., & Jdrzêjowicz, P. (2003). Population-based approach to multiprocessor task scheduling in multistage hybrid flow shops. *Lecture Notes in Computer Science, 2773*, 279–286.
Jin, S., Schiavone, G., & Turgut, D. (2008). A performance study of multiprocessor task scheduling algorithms. *Journal of Supercomputing, 43*, 77–97.
Kennedy, J., & Eberhart R. (1995). Particle swarm optimization. In *Proceedings of IEEE international conference on neural network* (pp. 1942–1948).
Krawczyk, H., & Kubale, M. (1985). An approximation algorithm for diagnostic test scheduling in multi-computer systems. *IEEE Transactions on Computers, 2*, 869–877.
Lee, C. Y., & Cai, X. (1999). Scheduling one and two-processors tasks on two parallel processors. *IIE Transactions, 31*, 445–455.
Linn, R., & Zhang, W. (1999). Hybrid flow-shop schedule: A survey. *Computers and Industrial Engineering, 37*, 57–61.
Liu, B., Wang, L., & Jin, Y. H. (2005). Hybrid particle swarm optimization for flow shop scheduling with stochastic processing time. *Lecture Notes in Artificial Intelligence, 3801*, 630–637.

Oðuz, C., Ercan, M. F., Cheng, T. C. E., & Fung, Y. F. (2003). Heuristic algorithms for multiprocessor task scheduling in a two stage hybrid flow shop. *European Journal of Operations Research, 149*, 390–403.

Oðuz, C., Zinder, Y., Do, V., Janiak, A., & Lichtenstein, M. (2004). Hybrid flow-shop scheduling problems with multiprocessor task systems. *European Journal of Operations Research, 152*, 115–131.

Poli, R., Kennedy, J., & Blackwell, T. (2007). Particle swarm optimization an overview. *Swarm Intelligence, 1*(3), 33–57.

Shiau, D. F., Cheng, S. C., & Huang, Y. M. (2008). Proportionate flexible flow shop scheduling via a hybrid constructive genetic algorithm. *Expert Systems with Applications, 34*, 1133–1143.

Sivanandam, S. N., Visalakshi, P., & Bhuvaneswari, A. (2007). Multiprocessor scheduling using hybrid particle swarm optimization with dynamically varying inertia. *International Journal of Computer Science & Applications, 4*, 95–106.

Tseng, C. T., & Liao, C. J. (2008). A particle swarm optimization algorithm for hybrid flowshop scheduling with multiprocessor tasks. *International Journal of Production Research, 46*, 4655–4670.

Tu, K., Hao, Z., & Chen, M. (2006). PSO with improved strategy and topology for job shop scheduling. *Lecture Notes in Computer Science, 4222*, 146–155.

Xia, W. J., & Wu, Z. M. (2006). A hybrid particle swarm optimization approach for the jobshop scheduling problem. *International Journal of Advance Manufacturing Technology, 29*, 360–366.

Ying, K. C., & Lin, S. W. (2006). Multiprocessor task scheduling in multistage hybrid flowshops: An ant colony system approach. *International Journal of Production Research, 44*, 3161–3177.

Ying, K. C. (2008). Iterated greedy heuristic for multiprocessor task scheduling problems. *Journal of the Operations Research Society, 1*, 8.