Performance of Particle Swarm Optimization in Scheduling Hybrid Flow-Shops with Multiprocessor Tasks

M. Fikret Ercan[1] and Yu-Fai Fung[2]

[1] School of Electrical and Electronic Engineering, Singapore Polytechnic, 500 Dover Rd., S139651, Singapore
mfercan@sp.edu.sg
[2] Department of Electrical Engineering, Hong Kong Polytechnic University, Hung Hom Kowloon, Hong Kong SAR
eeyffung@inet.polyu.edu.hk

Abstract. In many industrial and computing applications, proper scheduling of tasks can determine the overall efficiency of the system. The algorithm, presented in this paper, tackles the scheduling problem in a multi-layer multiprocessor environment, which exists in many computing and industrial applications. Based on the scheduling terminology, the problem can be defined as multiprocessor task scheduling in hybrid flow-shops. This paper presents a particle swarm optimization algorithm for the solution and reports its performance. The results are compared with other well known meta-heuristic techniques proposed for the solution of the same problem. Our results show that particle swarm optimization has merits in solving multiprocessor task scheduling in a hybrid flow-shop environment.

1 Introduction

Scheduling concerns with allocating limited resources to optimize an objective and it is an immensely studied field of engineering. Multiprocessor task scheduling is a generalized form of classical machine scheduling problem where a task is processed by more than one processor [2]. Multiprocessor task scheduling is a challenging problem encountered in computer and manufacturing processes and it is difficult to solve even in its simplest form [3, 6]. Earlier studies in multiprocessor task scheduling are mainly concerned with a single stage setting of the processor environment. However, there are many practical problems that require jobs to go through more than one stage. This problem, known as multistage flow-shop, is also vastly studied in scheduling context though most of these studies consider single processor at each stage [13]. With the advances made in technology, in many practical flow-shop environments, we encounter parallel processors at each stage instead of single processor (see for instance [1, 4, 11, 12, 17]). This scheduling problem is defined as multiprocessor task scheduling in a hybrid flow-shop environment and it is gaining attention from the research community. As the complexity of the problem increases with the increasing number of layers, the early studies tackled a simple form of the problem, two-layer flow-shops with multiprocessors [14, 8]. Simple list based heuristics as well as meta-heuristics such as Simulated Annealing (SA), Tabu Search (TS) and Genetic Algorithm (GA) are introduced for the solution. A comparative performance study of these heuristics can be found in [5]. However, there are many practical applications where multiprocessor tasks are executed in a setting that includes more than two stages. Studies tackling this problem using meta-heuristics are quite recent. Such meta-heuristic solutions reported in literature include GA [16], TS [15], SA [18] and recently Ant Colony System [19]. Since solution to this problem finds applications in various fields in industry and in computing, finding a good solution is important. In this paper, Particle Swarm Optimization (PSO) approach for the solution is presented and results are compared with other algorithms published in the literature.

The PSO is another evolutionary algorithm [9, 10] gaining popularity within the research community. It mimics the behavior of flying birds and their communication mechanism to solve optimization problems. It is based on a constructive cooperation between particles, in contrast with survival of the fittest approach used in other evolutionary methods [9]. It has many advantages which made it worth to experiment and study its performance for the scheduling problem presented in this paper. The algorithm is simple, fast and easy to code. It is not computationally intensive in terms of memory requirements and time. Furthermore, it has only a few parameters to tune.

The problem considered in this paper is formulated as follows: There is a set J of n independent and simultaneously available Multi-Processor Tasks (MPTs) to be processed in a multi-stage flow-shop, where stage j consists of m_j identical parallel processors $(j = 1,2,..,k)$. Each $MPT_i \in J$ should be processed on $p_{i,j}$ identical processors simultaneously at stage j without interruption for a period of $t_{i,j}$ $(i = 1,2,...,n \textit{ and } j = 1,2,...,k)$. Hence, each $MPT_i \in J$ is characterized by its processing time, $t_{i,j}$, and its processor requirement, $p_{i,j}$ $(i = 1,2,...,n \textit{ and } j = 1,2,...,k)$. Tasks flow from one stage to another by utilizing any of the processors while satisfying the flow-shop and the MPT constraints. In the remainder of this paper, set of MPTs that are going to be processed on a multi stage system will be called as jobs and 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 and each processor can handle no more than one MPT at a time.
- The processing time and 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.

In the following section, we describe the design of the algorithm. Next, the computational study is presented. We then report and discuss the computational results. Conclusions are given in the last section.

2 The PSO Algorithm

The PSO algorithm is initialized with a population of random solutions which is similar to evolutionary algorithms. Each individual solution flies in the problem space with a velocity that is adjusted depending on the experiences of an individual and the population. There are various models of PSO. In this study, the global model is employed and it is defined as:

$$V_{id} = WV_{id} + C_1 R_1(P_{id} - X_{id}) + C_2 R_2(P_{gd} - X_{id}) \text{ and } X_{id} = X_{id} + V_{id}.$$

In the above equation, V_{id} is the velocity of particle i and it represents the distance traveled from its current position. W is the inertia weight. X_{id} represents the particle position. P_{id} is the local best solution (also called "*pbest*") and P_{gd} is the global best solution (also called "*qutgbest*"). C_1 and C_2 are acceleration constants which drive particles towards the local and the global best positions. R_1 and R_2 are two random numbers within the range $\{0,1\}$. The algorithm follows the following steps:

```
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 a particle is better than $P_{id}$
then set $P_{id}$ to the current value;
    if $P_{id}$ is better than the global best then set $P_{gd}$ to
current particle fitness value;
    Change the velocity and position of the particle;
    until termination = True
end.
```

The initial swarm and particle velocity are generated randomly. Each particle consists of a sequence of job numbers representing the n number of jobs on a machine with k number of layers where each layer has m_j identical processors $(j = 1,2,..,k)$. The fitness of a particle is measured with maximum completion time of all jobs. A function is developed to map a given job sequence to the multiprocessor system and compute the maximum completion time as shown in Figure 1. A particle with lowest completion time is a good solution and has to be kept in search process.

In the example given in Figure 1, it is assumed that a job sequence is generated as $S_1 = \{2,3,1,4,5\}$ and a machine is made up of two stages where each stage contains four identical processors. 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 list S_1. Once all the jobs are scheduled at first stage, a new list is produced for the succeeding stage based on completion of jobs at previous stage and precedence relationship given in list S_1. In the new list, $S_2 = \{2,1,3,4,5\}$, 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. 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.

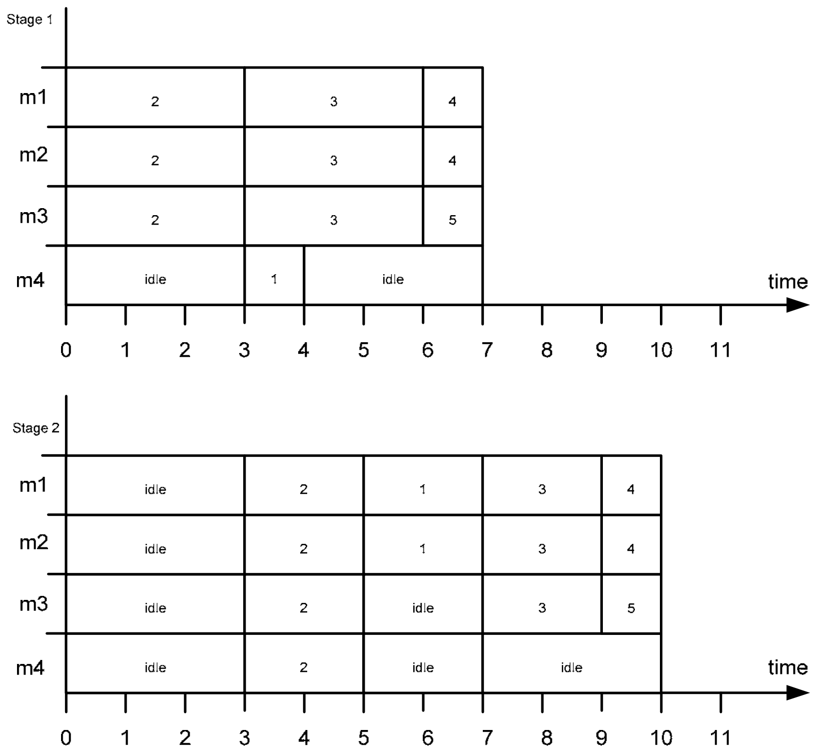


Fig. 1. The schedule of job sequence [2, 3, 1, 4, 5] after being decoded to processors with scheduling algorithm. Idle periods of the processors are labeled as idle.

The setting of parameters for PSO are based on our empirical studies as well as studies and 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 as a large value facilitates searching new locations whereas a small value provides 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 a 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 which is defined as:

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

where, W_{start} is the starting and W_{end} is the ending inertia values. W_{start} and W_{end} are set as 1.5 and 0.3 respectively. In addition, x shows current iteration number and $x_{\max}$ maximum iteration number which is set to 10000 iterations. An integer constant α is used to manipulate the gradient of the exponentially decreasing W and it is set to 4.

In our 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 the search space is obtained by modifying the job sequence.

3 Computational Results

An intensive computational study is conveyed to examine the performance of the PSO algorithm in minimizing the overall completion time of all jobs. Furthermore, the effects of parameters such as number of jobs and processor configurations on the performance of the algorithm are also investigated. The algorithm is implemented with the Java language and run on a PC, with 2GHz Pentium processor and 1024 MB memory. The results are presented in terms of Average Percentage Deviation (*APD*) of the solution from the lower bound which is expressed as $APD = \frac{C_{\max} - LB}{LB} \times 100$. Here, $C_{\max}$ indicates the completion time of the job and LB indicates the lower bound calculated for the problem instance. The lower bound used in this study was developed by Oğuz et. al. [15] and it is given with the following formula:

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

In the above formula, M and J represent the set of stages and set of jobs respectively. We used the benchmark data available at Oğuz's web-site (http://home.ku.edu.tr/~coguz/). Data set contains instances for two types of processor configurations: (i) random number of processors in each stage selected from a set of {1,..., 5} and (ii) identical number of processors at each stage which is fixed to 5 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 evaluated based on 10 problem instances. Tables 1 and 2 present the APD results obtained for PSO. Furthermore, we have compared the results of PSO algorithm with GA [16], TS [15] and ACS [19]. The efficiency of GA presented in [16] is closely related to control parameters, as well as cross over and mutation techniques. The authors introduce a new cross over technique named NXO, which aimed to capture the characteristics of the problem. In addition, they implemented the Partially Matched Crossover, PMX [7] for comparison and employed two types of mutation techniques namely Insertion mutation (iM) and Swap mutation (sM). In Tables 1 and 2, the best results reported by GA are listed. However, for the sake of completeness, crossover and mutation techniques used for which the best results were obtained are also indicated. Similarly, TS [15] and ACS [19] results obtained for the same problem sets are listed in the tables.

From the experimental studies, it can be observed that PSO outperformed the TS algorithm and approximated the results of GA and ACS algorithms. It outperformed TS in 10 problems out of a total of 12 for the random processor case and the best improvement rate was 82.13% when *k*=5 and *n*=100. Similarly, it outperformed 8 out of 12 problems for the fixed processor case though the best improvement rate was 60.09% when *k*=8 and *n*=10. When compared to another swarm based algorithm ACS, the results were inferior. ACS outperformed more than 50% in 6 of the 12 problems for the random processors case. On the other hand, the best improvement for ACS was 31.8% for the fixed processor case. PSO outperformed GA 4 in 12 problems for random processors with best improvement of 54% and 1 in 12 problems for fixed processors. It can be observed that GA is superior as compared to other heuristics methods when the number of stages is small.

For a given number of stages, increasing the number of jobs results in a decrease in *APD* value. This characteristic is observed for all the results presented in Tables 1 and 2. Apparently, the lower bound becomes more effective and close to the optimal solution. On the other hand, for a given number of jobs, increasing the number of stages results in an increase in *APD* as the lower bound becomes less effective. From the published results, it is observed that when the number of processors are fixed, that is $m_i = 5$, the scheduling problem becomes more difficult to solve and *APD* results are relatively higher. The same characteristic is also observed for PSO algorithm.

We also experimented the PSO algorithm on a set of data obtained from an actual machine vision system. This data can also be obtained from the web-site (http://home.ku.edu.tr/~coguz/). In this experiment, we compared the convergence of the algorithms. It can be seen from Figure 2, PSO converges much faster than TS and GA. In addition, PSO finds a slightly better solution for 12 job problem. All three algorithms demonstrate a fast convergence. In fact, search termination rule can be limited to 1000 iterations to complete scheduling in an acceptable time interval which is more suitable for practical applications.

CPU time spend for the experiments is another indicator of the algorithm performance though it won't be a fair comparison as different processors and compilers were used for each reported algorithm. However, in order to have a rough idea on the speed performance of PSO, the observations for one easy and one difficult problem case are collated in Table 3. Termination criterion is 10000 iterations for all the algorithms. It can be seen from Table 3 that PSO is approximately 48% to 35% faster than reported GA CPU timings and 55% to 57% faster than TS timings [16].

Table 1. *APD* of the algorithms for 10 random instances. Processors setting is $m_j \sim [1,5]$.

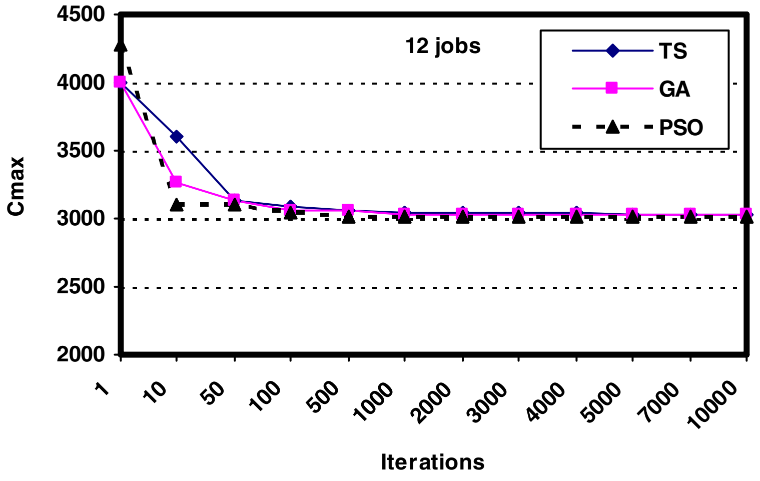
k	*n*	GA	*TS*	*ACS*	*PSO*
2	10	1.60 (NXO/iM)	3.00	1.60	2.7
2	20	0.80(NXO/iM)	2.88	1.92	2.88
2	50	0.69 (NXO/iM)	2.23	2.37	2.38
2	100	0.35 (NXO/iM)	9.07	0.91	1.82
5	10	11.89 (PMX/iM	29.42	9.51	10.33
5	20	5.54 (NXO/iM)	24.40	3.07	8.6
5	50	5.11 (NXO/iM)	10.51	1.51	3.31
5	100	3.06 (NXO/iM)	11.81	1.05	2.11
8	10	16.14 (NXO/sM)	46.53	16.50	18.23
8	20	7.98 (NXO/iM)	42.47	6.77	12.03
8	50	6.03 (NXO/iM)	11.81	2.59	5.98
8	100	4.12 (PMX/iM)	21.50	1.33	8.78

Table 2. *APD* of the algorithms for 10 random instances. Processors setting is $m_j = 5$.

k	*N*	GA	*TS*	*ACS*	*PSO*
2	10	6.13 (NXO/iM)	10.82	12.62	13.8
2	20	7.10 (NXO/iM)	7.25	10.73	10.75
2	50	3.34 (NXO/iM)	5.80	8.17	10.32
2	100	2.87 (NXO/iM)	5.19	5.66	7.43
5	10	11.32 (NXO/sM)	45.14	26.09	29.6
5	20	10.78(NXO/iM)	35.13	15.11	19.4
5	50	14.91 (NXO/iM)	28.64	13.11	14.17
5	100	11.02 (NXO/iM)	26.49	12.45	12.45
8	10	25.98 (NXO/iM)	77.21	25.14	30.81
8	20	24.13(NXO/iM)	62.99	25.18	26.74
8	50	21.87 (NXO/iM)	54.25	22.23	27.01
8	100	19.46 (NXO/iM)	36.05	13.90	20.39

Table 3. Average CPU time (in seconds) spend by TS, GA and PSO. Processors setting is $m_j = 5$.

k	*N*	GA	*TS*	*PSO*
2	10	12.14	14.11	6.3
8	100	2109.9	3181.32	1388.67



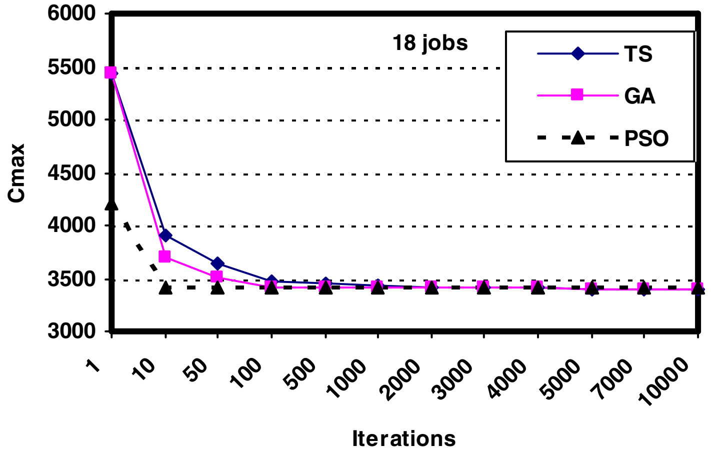


Fig. 2. Convergence of TS, GA and PSO algorithms for a machine vision system

4 Summary

In this paper, performance of a PSO algorithm for scheduling hybrid flow-shops with multiprocessors is presented. A job is made up of interrelated multiprocessor tasks where a multiprocessor task is modeled with its processing requirements and processing time. The performance of the PSO algorithm on its capacity to minimize maximum completion time of all jobs is evaluated. The algorithm outperforms the TS and approximates the performance of GA and ACS algorithms. However, considering that the simple form of the PSO algorithm is employed in this study, as compared to hybrid techniques employed in ACS [19] or problem specific cross over technique used in GA [16], results produced by PSO is satisfactory. In addition, in terms of convergence and CPU time, we observe that PSO outperforms TS and GA. Apparently, hybrid methods will improve the performance of PSO significantly and it is planned for our future study.

Acknowledgement

The first author gratefully acknowledges the contribution of his final year project students in coding and testing of the algorithm.

References

1. Caraffa, V., Ianes, S., Bagchi, T.P., Sriskandarajah, C.: Minimizing Make-Span in Blocking Flow-Shop Using Genetic Algorithms. International Journal of Production Economics 70, 101–115 (2001)
2. Chan, J., Lee, C.Y.: General Multiprocessor Task Scheduling. Naval Research Logistics 46, 57–74 (1999)
3. Drozdowski, M.: Scheduling Multiprocessor Tasks - An Overview. European Journal of Operational Research 94, 215–230 (1996)
4. Ercan, M.F., Fung, Y.F.: The Design and Evaluation of a Multiprocessor System for Computer Vision. Microprocessors and Microsystems 24, 365–377 (2000)
5. Ercan, M.F., Oğuz, C.P: Performance of Local Search Heuristics on Scheduling a Class of Pipelined Multiprocessor Tasks. Computers and Electrical Engineering 31, 537–555 (2005)
6. Garey, E.L., Johnson, D.S., Sethi, R.: The Complexity of Flow-shop and Job-shop Scheduling. Math. Operations Research 1, 117–129 (1976)
7. Goldberg, D., Lingle, R.: Alleles, Loci, and the Traveling Salesman Problem. In: Proceedings of the First International Conference on Genetic Algorithms and Their Applications, pp. 154–159 (1985)
8. Gupta, J.N.D, Hariri, A.M.A., Potts, C.N: Schedules for a Two-stage Hybrid Flow-shop with Parallel Machines at First Stage. Ann. Oper. Res. Soc. 69, 171–191 (1997)
9. Kennedy, J., Eberhart, R.: Particle Swarm Optimization. In: Proceedings of IEEE Int. Conf. on Neural Network, pp. 1942–1948 (1995)
10. Kennedy, J.: The Particle Swarm: Social Adaptation of Knowledge. In: Proceedings of IEEE Int. Conf. on Evolutionary Computation, pp. 303–308 (1997)
11. Krawczyk, H., Kubale, M.: An Approximation Algorithm for Diagnostic Test Scheduling in Multi-computer Systems. IEEE Trans. Computers 34(9), 869–872 (1985)
12. Lee, C.Y., Cai, X.: Scheduling One and Two-processors Tasks on Two Parallel Processors. IIE Transactions 31, 445–455 (1999)
13. Linn, R., Zhang, W.: Hybrid Flow-Shop Schedule: A Survey. Computers and Industrial Engineering. 37, 57–61 (1999)
14. Oğuz, C., Ercan, M.F., Cheng, T.C.E., Fung, Y.F.: Heuristic Algorithms for Multiprocessor Task Scheduling in a Two Stage Hybrid Flow Shop. European Journal of Operations Research 149, 390–403 (2003)
15. Oğuz, C., Zinder, Y., Do, V., Janiak, A., Lichtenstein, M.: Hybrid Flow-Shop Scheduling Problems with Multiprocessor Task Systems. European Journal of Operations Research 152, 115–131 (2004)

16. Oğuz, C., Ercan, M.F.: A Genetic Algorithm for Hybrid Flow-Shop Scheduling with Multiprocessor Tasks. Journal of Scheduling 8, 323–351 (2005)
17. Scala, M.L., Bose, A., Tylavsky, J., Chai, J.S.: A Highly Parallel Method for Transient Stability Analysis. IEEE Transactions on Power Systems 5, 1439–1446 (1990)
18. Sivrikaya-Serifoglu, F., Tiryaki, I.U.: Multiprocessor Task Scheduling in Multistage Hybrid Flow-Shops: A Simulated Annealing Approach. In: Proceedings of 2nd Int. Conf. on Responsive Manufacturing, pp. 270–274 (2002)
19. Ying, K.C, Lin, S.W.: Multiprocessor Task Scheduling in Multistage Hybrid Flow-Shops: an Ant Colony System Approach. International Journal of Production Research 44, 3161–3177 (2006)