Discrete Particle Swarm Optimization (DPSO) Algorithm for Permutation Flowshop Scheduling to Minimize Makespan

K. Rameshkumar[1], R.K. Suresh[1], and K.M. Mohanasundaram[2]

[1] Department of Production Engineering,
Amrita School of Engineering, Amrita Vishwa Vidyapeetham,
Ettimadai, Coimbatore 641105, India
k_rameshkumar@ettimadai.amrita.edu
[2] Department of Mechanical Engineering,
PSG College of Technology, Coimbatore, India

Abstract. In this paper a discrete particle swarm optimization (DPSO) algorithm is proposed to solve permutation flowshop scheduling problems with the objective of minimizing the makespan. A discussion on implementation details of DPSO algorithm is presented. The proposed algorithm has been applied to a set of benchmark problems and performance of the algorithm is evaluated by comparing the obtained results with the results published in the literature. Further, it is found that the proposed improvement heuristic algorithm performs better when local search is performed. The results are presented.

1 Introduction

Flowshop scheduling is one of the best-known production scheduling problem which has been proved to be strongly NP Complete [1]. It involves determination of the order of processing jobs on machines, arranged in series, to optimize the desired measure of performance such as makespan, total flowtime, etc. Makespan is the completion time of the last job. Minimization of the makespan time ensures increased utilization of the machines and thus leads to a high throughput [2,3]. The order of processing of jobs affects the performance measures.

Flowshop scheduling problem is a widely researched problem. Exact and constructive heuristic algorithms for solving static permutation flowshop scheduling problems (PFSPs) have been proposed by various researchers [4] over the years with the objective of minimizing makespan. The exact techniques are computationally effective for small size problems. But, for large size problems, such methods are computationally expensive and heuristics are resorted to. Heuristics for the flowshop scheduling problems can be a constructive heuristics or improvement heuristics. Johnson [5], Gupta [6], Palmer [7], Dannenbring [8], Koulamas [9] and Nawaz *et al.* [10] have proposed constructive heuristics. Among these, the NEH heuristics developed by Nawaz *et al.* performs better [11].

Unlike constructive heuristics, improvement heuristics take a solution and try to improve it. Dannenbring [8], Ho and Chang [12], Suliman [13] have proposed

L. Wang, K. Chen, and Y.S. Ong (Eds.): ICNC 2005, LNCS 3612, pp. 572 – 581, 2005.

improvement heuristics. General purpose metaheuristics such as genetic algorithm [14-19], simulated annealing [20,21] and tabu search [11,23,24] have also been applied successfully for PFSPs.

In this paper, a discrete particle swarm optimization algorithm has been proposed for solving a set of benchmark FSPs published in the literature [25,15]. The performance measure under study is the makespan value. The proposed approach is based on the PSO heuristic proposed in [26]. Performance of the proposed approach is evaluated by comparing the obtained results with the results published in the literature [18,19].

2 Notations and Conventions

n	Number of jobs to be scheduled
m	Number of machines in the flowshop
t_{ij}	Processing time of operation j of job i
C_i	Completion time of job i
C_{max}	Makespan
C^*	Optimal makespan value or lower bound value reported in the literature.
N	Swarm size
t	Iteration counter
t_{max}	Maximum number of Iterations
P_k^t	Sequence of the k^{th} particle during t^{th} iteration i.e, $P_{current}$
$Z\left(P_k^t\right)$	Makespan value of k^{th} particle during t^{th} iteration
$P_{k(best)}$	Particle best, i.e., the sequence with least makespan value, found by the particle up to 't' iterations, i.e, P_{best}
$Z\left(P_{k(best)}^t\right)$	Makespan value of the P_{best} sequence of the k^{th} particle
G	Global best sequence (G_{best}) = Min $\left\{ Z\left(P_{k(best)}^t\right); k = 1, N \right\}$
L	Length of the velocity list
v_k^t	Velocity of k^{th} particle during t^{th} iteration
c_1, c_2, c_3	Learning coefficients

3 Formulation of the Static Permutation Flowshop Scheduling Problem

The permutation flowshop scheduling problem under study consists of scheduling 'n' jobs, with known processing times t_{ij}, on 'm' machines with the sequence of processing as identical and unidirectional. A schedule of this type is called a permutation schedule. The objective is to find the job sequence that minimizes the

makespan. The makespan of the flowshop problem is defined by the time span necessary to complete processing of all jobs. The makespan is given by equation 1.

$$C_{\max} = \max\{C_i, i = 1,2,\cdots,n\} \tag{1}$$

It is assumed that machine can process only one job at a time and no pre-emption is allowed.

4 Proposed Discrete Particle Swarm Optimization (DPSO) Algorithm

Particle swarm optimization (PSO) is a recently developed population based optimization method inspired by the social behavior of organisms such as bird flocking and fish schooling. PSO was first applied to optimize various continuous nonlinear functions by Kennedy and Eberhart [27, 28]. Later, PSO has been successfully applied to a wide variety of problems such as power and voltage control [29], neural network training [30], mass-spring system [31], task assignment [32], supplier selection and ordering problem [33], optimal operational path finding for automated drilling operations [34], traveling salesman problems [26], etc. Application of DPSO algorithm for solving scheduling problem is not reported in the literature.

4.1 DPSO Algorithm for FSPs

The PSO starts with a population of randomly generated initial solutions and searches iteratively in the problem space for optimal solution. The potential solutions are called particles which fly through the problem space by following their best particles. PSO starts the search process with a group of randomly generated sequences called particles. After this, P_{best} and G_{best} sequences are identified. Now, the iterative search process starts with computing velocity of each particle v_k using equation 2.

$$v_k^{t+1} = c_1 \times v_k^t + c_2\,(P_{Best} - P_{Current}) + c_3\,(G_{Best} - P_{Current}) \tag{2}$$

New position of a particle at time (t+1) is determined by its current velocity v_k^t, and $P_{k(best)}$ and G_{best} sequences. c_1, c_2 *and* c_3 are learning co-efficients. Particle position i.e, sequence is updated after every iteration using equation 3.

New position = Current position + Particle velocity.

$$P_k^{t+1} = P_k^t + v_k^{t+1} \tag{3}$$

The velocity index of a particle k at time t is computed using equation 4.

$$v_k = \left(\left(i_q, j_q\right)\right); q = 1, L \tag{4}$$

Where i, j represents job positions, and L represents the length of the list or the maximum number of possible transpositions which is randomly generated between 1 and n.

Here v_k refers to exchange of job positions (i_1, j_1), then positions (i_2, j_2), etc. After generating the velocity index v_k, particle velocity v_k^t is computed using the

equation 2. Various operations performed for computing particle velocity and updating particle positions are explained below:

Subtraction (position – position) operator: Let x_1 and x_2 be two positions representing two different sequences. The difference x_2 - x_1 is a velocity v. In the equation 2, for example subtracting two positions i.e. (P_{best} – $P_{current}$) results in a velocity which is a set of transpositions.

Addition (position + velocity) operator: Let x be the position and v be the velocity. New position x_1 is found by applying the first transposition of v to p, i.e, $x_1 = x + v$ then the second one to the result etc.

Addition (velocity + velocity) operator: Let v_1 and v_2 be two velocities. In order to compute $v_1 + v_2$,we consider the list of transpositions which contains first the 'ones' of v_1, followed by the 'ones' of v_2.

Multiplication (Coefficient × velocity) operator: Let c be the learning coefficient and v be the velocity. $c \times v$ results in a new velocity.

A numerical illustration of the above operations used in the proposed DPSO algorithm is presented in the section 4.3. Pseudocode of the DPSO algorithm for the flowshop scheduling problem is presented in Fig. 1.

4.2 Pseudocode of the DPSO Algorithm

$t = 0$
For (k = 1, N)
  Generate P_k^t
  Evaluate $Z(P_k^t)$
  $P_{k(best)} = P_k^t$
  $G_{best} = best\ particle\ found\ in\ P_k^t$
do {
  For (k = 1, N)
    $v_k^{t+1} = c_1 \times v_k^t + c_2 (P_{best} - P_{current}) + c_3 (G_{best} - P_{current})$
    $P_k^{t+1} = P_k^t + v_k^{t+1}$
    If $(P_k^{t+1}\ is\ better\ than\ P_k^t)$
    $P_{k(best)} = P_k^{t+1}$
    If $(P_k^{t+1}\ is\ better\ than\ G_{best})$
    $G_{best} = P_k^{t+1}$
  $t = t+1$;
} $(while\ t < t_{max})$
Output G_{best}

Fig. 1. Pseudocode of the DPSO algorithm for FSP

4.3 Numerical Illustration of DPSO Algorithm

Let us assume the following data during t^{th} iteration.

P_k^t = {2,3,1,4} ($P_{Current}$ Sequence) ;

$P_{k(best)}^t$ = {2,3,1,4} (P_{Best} Sequence), and

G^t = {3,4,2,1} (G_{Best} Sequence)

Computation procedure for calculating particle velocity and particle movement is explained as follows:

4.3.1 Generating Velocity Index

Velocity index v_k for the particle *k* is computed as follows:

Let length of the velocity list L_k be equal to 2 with the randomly generated list {(2,4), (4,1)}. Similarly, velocity index is computed for other particles.

4.3.2 Calculation of Particle Velocity

For each particle in the population, velocity for moving the particle from one position to the other position is calculated using equation 2. The coefficient times the velocity operator is used to find out the number of velocity components to be applied over the position. For example, if the coefficient value is 0.5, then 50 percent of the velocity components are randomly selected from the velocity list and applied over the position.

$$v_k^{t+1} = 0.5\times[(2,4),(4,1)] + 0.5\times[(2,3,1,4) - (2,3,1,4)] + 0.5[(3,4,2,1) - (2,3,1,4)]$$

$$= [(2,4)] + 0.5\times[0] + 0.5[(1,2),(2,4),(3,4)] = [(2,4),(1,2),(2,4)]$$

4.3.3 Particle Movement

Moving the particles from their current position to the new position is done using equation (3).

$$\text{i.e. } P_k^{t+1} = P_k^t + v_k^{t+1} = \{2,3,1,4\} + [(2,4),(2,3),(2,4)]$$

$$= \{2,4,1,3\} + [(2,3),(2,4)] = \{2,1,4,3\} + [(2,4)] = \{2,3,4,1\}$$

$$P_k^{t+1} = \{2,3,4,1\}$$

Similarly for all the particles P_k^{t+1} is computed and $P_{k(best)}^t$ (P_{best}) and G (G_{best}) are updated.

5 Experimental Investigation

The DPSOA has been tested on 14 benchmark problems given by Carlier [25] and Reeves [15]. These test problems have number of jobs (n) varying from 7 to 20, and the number of machines (m) from 4 to 15. By trial and error, optimum swarm size is

found to be 10. In all our computational effort, the total number of sequences evaluated by the algorithm is limited to $50n^2$.

The problem of solving FSPs using DPSO has been done in two phases. In the first phase of our study, DPSO algorithm is applied for solving a set of Carlier FSPs [25]. It is found that the DPSO algorithm obtained the best results (C^*) for six out of the eight benchmark problems. The results presented in Table 1 indicate that DPSO algorithm performs better when problem size is comparatively small. Further, in order to improve the performance of DPSO algorithm, a local search is applied at the end of every iteration. During local search every job except the last one is interchanged with its successor. There will be (n-1) number of interchanges. It is to be noted that an interchange which does not improve the makespan value is ignored. The result obtained by the DPSO algorithm with local search is presented in column number 6 of Table 1. When DPSO algorithm assisted by local search it is able to find the best solutions for all the eight Carlier problems.

Table 1. Carlier benchmark results (makespan)

Problem	Problem Size		C^*	DPSO	DPSO + Local search
	n	m			
Car1	11	5	7038	7038	7038
Car2	13	4	7166	7166	7166
Car3	12	5	7312	7392	7312
Car4	14	4	8003	8003	8003
Car5	10	6	7720	7768	7720
Car6	8	9	8505	8505	8505
Car7	7	7	6590	6590	6590
Car8	8	8	8366	8366	8366

In the second phase our study both the DPSO algorithm and the hybrid approach, i.e, DPSO assisted by local search, are applied to solve large size problems. Results are presented in Table 2. It is found that the DPSO algorithm assisted by local search perform better even with the large size problems.

Table 2. Reeves benchmark results (makespan)

Problem	Problem Size		C^*	DPSO	DPSO + Local search
	n	m			
Rec01	20	5	1247	1264	1249
Rec03	20	5	1109	1115	1111
Rec05	20	5	1242	1254	1245
Rec07	20	10	566	1624	1584
Rec09	20	10	1537	1621	1574
Rec11	20	10	1431	1590	1446

In order to evaluate the performance of DPSO algorithms proposed in this paper, a comparative study is performed with the results obtained by the popular constructive heuristics such as NEH, CDS, RA, GUPTA and PALMER [18] and the results are presented in Tables 3 and 4.

Table 3. Carlier benchmark results (makespan)

Problem	Problem Size		Constructive Heuristics (Makespan value)					DPSO	
	n	m	Palmer	Gupta	CDS	RA	NEH	no local search	with local search
Car1	11	5	7472	7348	7202	7817	7038	7038	7038
Car2	13	4	7940	7534	7410	7509	7940	7166	7166
Car3	12	5	7725	7399	7399	7399	7503	7312	7312
Car4	14	4	8423	8423	8423	8357	8003	8003	8003
Car5	10	6	8520	8773	8627	8940	8190	7720	7720
Car6	8	9	9487	9441	9553	9514	9159	8505	8505
Car7	7	7	7639	7639	6819	6923	7668	6590	6590
Car8	8	8	9023	9224	8903	9062	9032	8366	8366

Table 4. Reeves benchmark results (makespan)

Problem	Problem Size		Constructive Heuristics (Makespan value)					DPSO	
	n	m	Palmer	Gupta	CDS	RA	NEH	no local search	with local search
Rec01	20	5	1391	1434	1399	1399	1334	1264	1249
Rec03	20	5	1223	1380	1273	1159	1136	1115	1111
Rec05	20	5	1290	1429	1338	1434	1294	1254	1245
Rec07	20	10	1715	1678	1697	1722	1637	1624	1584
Rec09	20	10	1915	1792	1639	1714	1692	1621	1574
Rec11	20	10	1685	1765	1597	1636	1635	1590	1446

The relative percentage increase in makespan yielded by the proposed DPSO algorithms, genetic algorithm [18] and a hybrid genetic algorithm [19] for these benchmark problems, are presented in Tables 5 and 6.

Table 5. Relative percentage increase in makespan (Carlier Problems)

Problem	Problem Size		Relative percentage increase in makespan			
	n	m	DPSO		Hybrid GA (Wang)	GA (Ponnambalam)
			no local search	with local search		
Car1	11	5	0.00	0.00	0.00	0.00
Car2	13	4	0.00	0.00	0.00	0.00
Car3	12	5	1.09	0.00	0.00	2.42
Car4	14	4	0.00	0.00	0.00	0.00
Car5	10	6	0.62	0.00	0.00	0.36
Car6	8	9	0.00	0.00	0.76	0.00
Car7	7	7	0.00	0.00	0.00	0.00
Car8	8	8	0.00	0.00	0.00	0.00

Table 6. Relative percentage increase in makespan (Reeves Problems)

Problem	Problem Size		Relative percentage increase in makespan			
	n	m	DPSO		Hybrid GA (Wang)	GA (Ponnambalam)
			no local search	with local search		
Rec01	20	5	1.36	0.16	0.14	8.26
Rec03	20	5	0.54	0.18	0.09	7.21
Rec05	20	5	0.97	0.24	0.29	5.23
Rec07	20	10	3.70	1.15	0.69	8.56
Rec09	20	10	5.47	2.41	0.64	5.14
Rec11	20	10	11.11	1.05	1.1	8.32

Since the data available in [19] is the relative percentage increase in makespan, the same has been computed and presented in Tables 4 and 5. The relative percentage increase in makespan is computed as follows:

$$\frac{\left(\textit{Makespan obtained by the heuristic} - \textit{optimal makespan value}\right)}{\textit{optimal makespan value}} \times 100 \tag{5}$$

The results presented in Tables 1 and 2 indicate that the performance of DPSO algorithm is better when local search is employed. It is clear from the results presented in Tables 3 and 4 that DPSO algorithms outperform constructive heuristics. From the Table 5, performance of the DPSO algorithm is found to be better than the genetic algorithms [18] and the hybrid GA [19] for the small size PFSPs [25]. The results of large size PFSPs [15] are presented in Table 6. It indicates that the results of the Hybrid GA are better than the DPSO algorithms proposed in our paper. Better performance of the hybrid GA is due to high computational effort applied in [19]. The computational complexity of the DPSO algorithm is $50(n^2)$ where as for the hybrid GA it is approximately 800 (n^2).

6 Conclusions

In this paper a discrete particle swarm optimization (DPSO) algorithm is proposed to solve permutation flowshop scheduling problems. Drawbacks of DPSO in solving FSP have been identified. Local search procedure has been found to help the DPSO algorithm to escape from the local optima. The obtained results are compared with the results published in the literature. DPSO algorithms proposed in this paper found to perform better in terms of quality of the solutions and computational complexity.

Acknowledgment

The authors are thankful to the three anonymous reviewers for their comments to improve the earlier version of this paper.

References

1. Garey, M.R., Johnson, D. S.:Computers and Intractability: a guide to theory of NP – Completeness, Freeman, SanFrancisco (1979).
2. Pinedo, M.:Scheduling: Theory, Algorithms and systems, Second ed. Prentice-Hall, Englewood Cliffs, NJ (2002).
3. Chandraseakaran Rajendran, Sudipta Lahiri, Narendran, T.T.:Evaluation of heuristics for scheduling in a flowshop : a case study, Production Planning and control, 4 (2), (1993), 153-158.
4. Dimopoulos, C., Ali. M. S. Zalza.:Recent developments in evolutionary computation for manufacturing optimization: problems, solutions, and Computations, IEEE Transactions on Evolutionary Computation, 4(2), (2000), 93-113.
5. Johnson, S.:Optimal two and three stage production schedules with setup times included, Naval Research Logistics Quarterly, 1, (1954), 61.
6. Gupta, J.N.:A functional heuristic algorithm for flowshop scheduling problem, Operational Research Quarterly , 22(1), (1971), 39-47.
7. Palmer, D.:Sequencing jobs through a multi stage process in the minimum total time- a quick method of obtaining near optimum, Operational Research Quarterly, 16(1), (1965), 101-107.
8. Dannenbring, D. G.:An evaluation of flowshop sequencing heuristic, Management science 23(11), (1977) 1174-1182.
9. Koulamas, C.:A new constructive heuristic for flow-shop scheduling problem, European Journal of Operational Research 105, (1998) 66-71.
10. Nawaz, M., Enscore Jr, E., Ham, I.: A heuristics algorithm for the m-machine, n-job flowshop sequencing problem, Omega, 11(1), (1983) 91-95 .
11. Taillard, E.:Some efficient heuristic methods for flowshop sequencing problem, European journal of Operational research 47, (1990), 67-74.
12. Ho, J.C., Chang, Y. L.:A new heuristic for n-job , m-machine flow-shop problem, European Journal of Operational Research, 52, (1991), 194-202.
13. Suliman, S.:A two-phase heuristic approach to the permutation flow-shop scheduling problem, International Journal of Production Economics, 64, (2000), 143-152.
14. Chen, C.L., Vempati, V.S., Aljaber, N.:An application of genetic algorithms for flowshop problems, European Journal of Operational Research, 80, (1995), 389-396.
15. Reeves, C. R.: A Genetic Algorithm for Flowshop Sequencing, Computers and Operations Research, 22(1), (1995) 5-13 .
16. Murata, T., Ishibuchi, H., Tanaka, H.:Genetic algorithms for flowshop scheduling problems, Computers and industrial Engineering, 30(4), (1996), 1061-1071.
17. Reeves, C., Yamada, T.:Genetic algorithms, path relinking, and the flowshop sequencing problem, Evolutionary Computation, 6(1), (1998),45-60.
18. Ponnambalam, S. G., Aravindan, P., Chandrasekaran , S.:Constructive and improvement flow shop scheduling heuristics: an extensive evaluation, Production Planning & Control, Vol. 12, No. 4, (2001) 335- 344.
19. Wang, L., Zheng, D. Z. :An Effective Hybrid heuristic for flowshop scheduling, International Journal of Advanced manufacturing technology, 21, (2003), 38-44.
20. Osman, I., Potts, C.:Simulated annealing for permutation flow-shop scheduling, OMEGA, The international Journal of Management Science, 17(6), (1989), 551-557.
21. Ogbu, F., Smith, D.:Simulated Annealing for the permutation flow-shop problem, OMEGA, The international Journal of Management Science, 19(1), (1990), 64-67.

22. Ishibuchi, H., Misaki, S., Tanaka, H.:Modified simulated annealing algorithms for the flowshop sequencing problem, European Journal of Operational Research, 81, (1995), 388-398.
23. Nowicki, E., Smutnicki, C.:A fast Tabu Search algorithm for the permutation flow-shop problem, European Journal of Operational Research, 91, (1996), 160-175.
24. Moccellin, J.A.V., Dos Santos, M.O.:A new heuristic method for the permutation flowshop scheduling problem, Journal of the Operational Research Siciety, 105, (2000), 883-886.
25. Carlier, J.:Ordonnancements a contraintes disjonctives, Rairo Recherehe operationelle / Operations Research, 12, (1978) 333-351.
26. Clerc, M.:Discrete particle swarm optimization, illustrated by the Traveling Salesman Problem, New Optimization Techniques in Engineering. Heidelberg, Germany: Springer, (2004) 219-239.
27. Kennedy, J., Eberhart, R.C.:Particle swarm optimization, Proceedings of IEEE International Conference on Neural Networks, Piscataway, NJ, USA, (1995), 1942-1948.
28. Eberhart, R. C., Kennedy, J.:A new optimizer using particle swarm theory, Proceedings of the Sixth International symposium on Micro machine and Human Science, Nagoya, Japan, IEEE Service center, Piscataway, NJ, (1995) 39-43.
29. Abido, M.A.:Optimal power flow using particle swarm optimization, Electrical Power and Energy Systems 24, (2002) 563-571
30. Van den Bergh, F., Engelbecht, A.P.: Cooperative learning in neural networks using particle swarm optimizers, South African Computer Journal 26 (2000) 84-90.
31. Brandstatter, B. , Baumgartner, U. :Particle swarm optimization: mass-spring system analogon, IEEE Transactions on Magnetics 38 (2002) 997- 1000.
32. Salman, A., Ahmad, I., Al-Madani, S.:Particle swarm optimization for task assignment problem", Microprocessors and Microsystems, 26, 2003.
33. Yeh, L. W.:Optimal Procurement Policies for Multi-product Multi-supplier with Capacity Constraint and Price Discount, Master thesis, Department of Industrial Engineering and Management, Yuan Ze University, Taiwan, R.O.C., (2003).
34. Onwubolu, G.C., Clerc, M.: Optimal operational path for automated drilling operations by a new heuristic approach using particle swarm optimization, International Journal of Production Research, 42(3), (2004), 473-491.