On the Influence of Parameters in Particle Swarm Optimisation Algorithm for Job Shop Scheduling

ANIL, B. and SIVAKUMAR, S.
Department of Mechanical Engineering,
College of Engineering, Trivandrum
INDIA

Abstract: - Particle Swarm Optimization (PSO) is one of the latest nature inspired meta-heuristic algorithms based on the metaphor of social interaction and communication such as bird flocking and fish schooling. PSO is a population based algorithm for finding optimal regions of complex search spaces through interaction of individuals in the population. In PSO, a set of randomly generated solutions (initial swarm) navigate in the design space towards the optimal solution over a number of iterations (moves) based on large amount of information about the design space that is assimilated and shared by the members of the swarm. The solution of PSO depend on the parameter setting such as swarm size, number of generations, inertia factor, self confidence factor and swarm confidence factor. This work describes the study on the influence of parameter settings in PSO algorithm in solving the Static Job Shop Scheduling Problem, which is a NP-hard combinatorial problem. Performance of the algorithm is tested on benchmark problems. The influence of each PSO parameter on the performance of algorithm is studied in detail.

Key-Words:- Combinatorial optimization, nature inspired meta-heuristic, PSO, Scheduling, Job shop, parameter tuning

1 Introduction

Particle Swarm Optimization (PSO) algorithm developed by Eberhart and Kennedy [1], is applied extensively for finding optimal solutions in a large number of problems. PSO, inspired by the behavior of fish schooling and bird flocking, employs a set of solutions (swarm) to search the solution space. Each member solution is called a particle, and it moves around in the multi-dimensional search space with a velocity constantly updated by the particle's inertia, the experience of the particle, and the experience of the whole swarm. PSO was first introduced to optimize various continuous nonlinear functions. The advantages of PSO algorithm such as simple structure, ease of implementation, speed to get the solutions, and robustness made it very popular in a short period of time. Fatih Tasgetiren et. al [2] proposed the Smallest Position Value (SPV) rule, to convert continuous position values to discrete operation sequence enabling to extend the scope of PSO to combinatorial problems. The PSO applied to permutation flowshop can be seen in [10] and an application of SPV rule based PSO algorithm in multi-objective flow shop scheduling is found in [8].

Job Shop Scheduling (JSS) problems have been studied for a long time and is perhaps the most general and complex of all types of shop scheduling problems. Since it is a constrained optimization problem, the possibility of getting a large number of feasible solutions is less as compared to flow shop scheduling problem[7]. It is difficult to reach the optimal solution in a short time, since the problems have a very wide solution space and there is no guarantee to reach a better state after a feasible state. Small size instances of the JSS problem can be solved within reasonable computational time by exact algorithms such as branch-and-bound and the time orientation approach. However, when the problem size increases, the computational time of exact methods grows exponentially. On the other hand heuristic algorithms require generally acceptable time and memory requirements, but do not guarantee optimality of the final solution. The most recent research on JSS problems has been focused on heuristic algorithms. The solution techniques of heuristic algorithms can be broadly classified into two groups: meta-heuristics and local search type heuristics[4]. In the first category, Simulated Annealing (SA), Genetic Algorithm (GA),Tabu Search (TS), Ant Colony Optimization (ACO), Hybrid SA and GA[5], Neural Network (NN), have provided abundant research. The latter group consists of shifting bottleneck procedure, Priority Dispatching rules, Randomized dispatching rules, Guided local search, constraint propagation and parallel Greedy Randomized Adaptive Search Procedure (GRASP). In this work, the JSS problem is solved by PSO algorithm and a detailed study on the influence of various PSO parameters on the performance of the algorithm in the Job shop context is conducted.

2 Problem Description

The JSS problem with the objective function of minimizing makespan can be stated as follows;

There are n jobs and each of the jobs is to be processed without preemption by m machines. Each job consists of m operations that own a predetermined processing order through machines. Each machine can handle no more than one job at a time and each job must visit each machine only once. The release time of all jobs is zero. Set-up and knock-down times on each machine are included in the processing time. An example of 3x3 JSSP is given in the Table 1. The data includes the routing of each job and the processing time for each operation (in parentheses) [9].

Table 1. A 3x3 JSS problem

Job	Operations routing (Processing time)		
1	1(3)	2(3)	3(3)
2	1(2)	3(3)	2(4)
3	2(3)	1(2)	3(1)

The overall objective is to minimize the completion time of whole schedule(overall completion time of all jobs).

The objective function is, Minimize $C_{\max}$
where $C_{\max} = \max\{C_j\}_{j=1,2..,n}$ (1)
C_j is the completion time of j^{th} job, and n is the total number of jobs.

3 PSO Algorithm in Job Shop Problem

3.1 Solution Representation

In order to construct a direct relationship between the problem domain (JSS problem) and the PSO particles, n x m number of dimensions for as much number of operations is used. The particle $Xti = [xti_1, xti_2,...,xt_{nm}]$ corresponds to the continuous position values for n x m operations of a particular solution i for t^{th} iteration. The particle itself does not present a sequence of operation or solution. Instead, the Shortest Position Value (SPV) rule[2] is used to determine the permutation implied by the position values x_{ij}^t of particle Xi^t. According to the SPV rule, the operation corresponding to the smallest position value is assigned to be the first operation in the permutation π_i^t; the operation corresponding to the next higher position value is assigned to be the second operation, and so on (operations are sorted according to the position values x_{ij}^t to construct the operation sequence π_i^t). Representation of a particle(Xi) for a 3x3 JSS problem, with position values and corresponding operation sequence(πi) generated using SPV rule are shown in Table 2. For a 3 job, 3 machine problem, there will be 9 operations to be scheduled. The second row represents the randomly generated position values for 9 operations. Third row sorts the position values in ascending order based on SPV rule. Fourth row (π_i) represents a single solution in the form of operation sequence.

Table 2. Solution representation using SPV rule

Dimensions n x m (No.of operation)	1	2	3	4	5	6	7	8	9
Position values (Xi)	2.1	3.5	0.5	1.2	-3.6	0.8	2.6	-0.3	-1.5
Applying SPV rule	-3.6	-1.5	-0.3	0.5	0.8	1.2	2.1	2.6	3.5
Operation order (π_i)	5	9	8	3	6	4	1	7	2

3.2 Initial Population Generation

Normally in PSO algorithm, the initial population of particles are generated randomly. In the JSS problem, the probability of getting feasible solutions in the random generation of initial population is very low. In this work, one feasible schedule is generated by applying a priority dispatching rule (Shortest Processing Time rule - SPT) and used as the first particle in the initial population. The remaining *(Ps-1)* particles are generated randomly, where *Ps* is the total number of particles in the population or swarm.
The following equation is used to construct the initial continuous position values of the particle:

$$X_{ij}^0 = x_{min} + (x_{max} - x_{min}) * r_1 \quad (2)$$

where $x_{min} = -4$, $x_{max} = 4.0$, and r_1 is a uniform random number between 0 and 1.

Initial velocities are generated for each particle by a similar formula as follows:

$$v_{ij}^0 = v_{min} + (v_{max} - v_{min}) * r_2 \quad (3)$$

where $v_{min} = -4.0$, $v_{max} = 4.0$, and r_2 is a uniform random number between 0 and 1.

3.3 The computational procedure

Step 1: Initialization

Set iteration count, $t = 0$, Ps = the number of particles in the swarm. Generate Ps particles with position vectors X and Velocity V randomly,

$\{X_i^0, i = 1,2,...,Ps\}$, where $X_i^0 = [x_{i1}^0, x_{i2}^0, .., x_{i(nxm)}^0]$

$\{V_i^0, i = 1,2,...,Ps\}$, where $V_i = [v_{i1}^0, v_{i2}^0, ..., v_{i(nxm)}^0]$

Apply the SPV rule to find the operation sequence except for the first particle.

$\pi^0_i = [\pi^0_{i1}, \pi^0_{i2}, .., \pi^0_{i(nxm)}]$ of particle X^0_i for $i = 2,3,...,Ps$.

Check the feasibility of each particle(operating sequence) by comparing it with the constraint.

Find the fitness value (f – makespan) of each feasible particle in the swarm. Infeasible particles will be assigned with a makespan of very high value until they get a feasible solution in subsequent iterations.

For each particle i in the swarm, set the *personal best* position values to the present position values ($P^0_i = X^0_i$) and assign the fitness as the best fitness value, f^{pb}_i for $i = 1,2,..., Ps$.

When an infeasible solution gets a feasible solution in a subsequent iteration, the position values in that iteration will be set as personal best of that particle.

Find the Global best, which is the best fitness value among the whole swarm such that,

$f_l = min\{ f^0_i \}$ *for* $i = 1,2,...,Ps$ with its corresponding positions X^0_l.

Set global best to $G = X^0_l$ such that

$G^0 = [g_1 = x_{l,1}, g_2 = x_{l,2}, ..., g_{nxm} = x_{l\,nxm}]$ with its fitness value

$f^{gb} = f_l$

Step 2: Update iteration counter ($t = t+1$). Update Inertia weight factor (w),

$$w^t = a + \{b \text{ x } (Mg-h)/(Mg-1)\} \quad (4)$$

Where Mg is the maximum number of generations, h is the current iteration number.
(Inertia weight is gradually reduced from a+b to a linearly as iteration proceeds)

Step 3: Update velocities and positions

$$v^t_{ij} = \{w^t v^{t-1}_{ij} + C_1 r_1 (p^{t-1}_{ij} - x^{t-1}_{ij}) + C_2 r_2 (g^{t-1}_j - x^{t-1}_{ij})\} \quad (5)$$

Where C_1 and C_2 are *Self confidence* and *Swarm confidence* factors, r_1 and r_2 are uniform random numbers between (0, 1).

$$x^t_{ij} = x^{t-1}_{ij} + v^t_{ij} \quad (6)$$

Step 4: Apply the SPV rule to find the new permutations and calculate fitness.

Step 5: Update personal best and global best
If $f^t_i < f^{pb}_i$ for $i=1,2,.,Ps,$ then personal best is updated as $P^t_i = X^t_i$ *and* $f^{pb}_i = f^t_i$

$f^t_l = min\{ f^{pb}_i \}$, $i = 1,2,..., Ps$; $l \in \{i; i = 1,2,..., Ps\}$.

If $f^t_l < f^{gb}$, then the global best is updated as $G^t = X^t_l$ and $f^{gb} = f^t_l$

Step 6: Termination criterion
If the iteration count exceeds the maximum number of iterations, then stop; otherwise go to step 2.

4 Parameter Study

The performance of the PSO algorithm depends heavily on the parameter values used and tuning of the parameters is essential for the fast convergence of the solution [6]. The major parameters in the PSO algorithm are the *swarm size, number of generations, inertia factor, self confidence factor* and *swarm confidence factor*. The details of the parameters used in this study are given in Table 3.

Table 3. Different parameters of PSO and its role

PSO Parameter	Role of parameter
Swarm size (*Ps*)	Number of particles in the population
Number of Generations (*Mg*)	Termination criteria
Inertia factor (*w*)	To control the effect of previous velocity on current velocity
Self confidence factor (*C1*)	To control the effect of Personal best on solution
Swarm confidence factor (*C2*)	To control the effect of Global best on solution

The algorithm is applied on the benchmark problems of Fischer Thompson[3] of size 6x6 (FT-06), and Lawrence problems of sizes 10x5 (LA-01), 15x5 (LA-06), 20x5 (LA-11, LA-10), 15x10 (LA-21), 20x10 (LA-26) and 30x10 (LA-31).

4.1 Effect of Swarm size (*Ps*)

A 10 job x 5 machine problem(LA-01), was considered for the study. The swarm size was varied between 50 to 700. The variation in makespan with generations at different swarm size is shown in Figure 1. The table 4 shows the converged makespan value for different swarm sizes. For a swarm size of 500 particles, the value of makespan is 704. There was no further improvement in the value at higher swarm sizes. The study on different problem sets reveal that a swarm size of 500 gives the best performance at reasonable computation time.

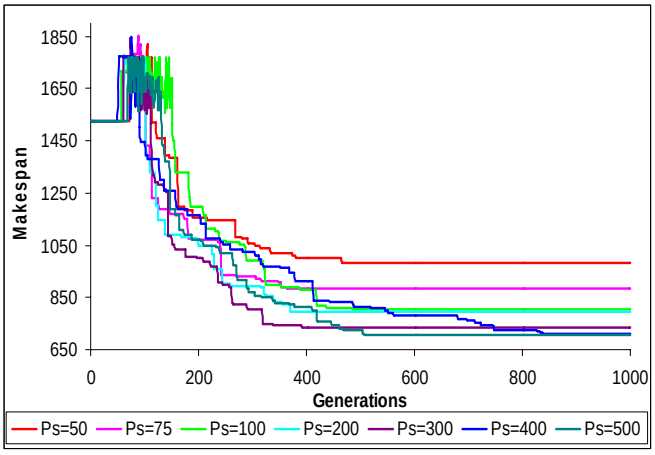

Fig 1. Convergence behavior with different swarm size

Table 4 Converged Makespan at different swarm sizes

Swarm-size (*Ps*)	Makespan
50	984
75	886
100	803
200	794
300	736
400	712
500	**704**
600	704
700	704

4.2 Effect of Number of Generations (*Mg*)

In order to fix a termination criteria, a study has been carried out to find the effect of *Number of generations* required on different sized problems. The swarm size has been fixed at 500. Table 5 summarises the results on different problem sizes. As the problem size increases, the number of generations required for convergence also increases. Therefore, the number of generations (termination criteria) has to be varied with the problem size. This also points to one of the inherent limitations of PSO.

Table 5 Convergence behavior with problem size

Problem code	Problem size	No. of operations in a schedule	Generation at which converges
FT-06	6 x 6	36	170
LA-01	10 x 5	50	538
LA-06	15 x 5	75	712
LA-11	20 x 5	100	1121
LA-21	15 x 10	150	1322
LA-26	20 x 10	200	1685
LA-31	30 x 10	300	1936

4.3 Effect of Inertia factor (*w*)

Inertia factor is used in the algorithm to control the effect of previous velocity of a particle on its current velocity. It is given a high value during the start of the search process and gradually decreased linearly with time to the lower limit. The algorithm is tested on a 6 job x 6 machine problem with different settings of inertia weight. The variation of makespan for a particular inertia factor over the generations is shown in Fig 2.

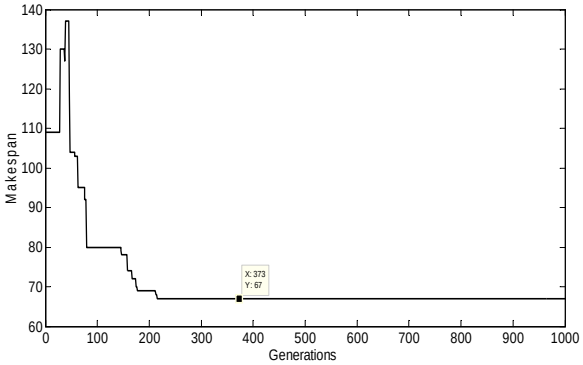

Fig 2 Variation of makespan with w =1.0 e^{-5} to 1.0 e^{-6} (for a 6 x 6 problem)

Table 6 summarises the convergence of the algorithm at different settings of inertia factor. The converged makespan value is shown in column 3. It has been observed that with a higher value of *w* (*w*=0.1 to 0.01), convergence rate is affected. Since the effect of previous velocity on current velocity is more in this case, it has higher tendency to keep the solution in the neighborhood of the original solution preventing it from seeking a global optimum. On the other hand a lower value of inertia factor (*w* =1e^{-5} to 1e^{-6}) results in a better solution within 200 generations. As can be seen from the table, an optimum value of makespan of 60 is obtained with an inertia factor setting of 1.0 e-6 to 1.0 e-7. There was no further improvement in solution at *w* values lower than this setting. It is evident that for encouraging exploration and for getting more feasible solutions a very low value of inertia factor shall be used.

Table 6. Convergence behavior with different inertia factors

Inertia factor (w)	Nature of Convergence	Makespan
0.1 to 0.01	Not converging	130
1.0 e^{-2} to 1.0 e^{-3}	Not converging	130
1.0 e^{-3} to 1.0 e^{-4}	Converging to a lower value	83
1.0 e^{-4} to 1.0 e^{-5}	Converging to a lower value	73
1.0 e^{-5} to 1.0 e^{-6}	Converging to a lower value	67
1.0 e^{-6} to 1.0 e^{-7}	**Converging to a lower value close to optimum**	**60**

4.3 Effect of *Self Confidence factor* (*C1*)

The importance to be given to the previous best position of each particle is determined by this factor. It is noted that this factor represents cognition, or the private thinking of the particle when comparing its current position to its own best. The effect of *C1* on algorithm is studied on a 15x5 problem (LA-10). *C1* value is varied from 2 to 1.0 e^{-8}. Fig 3 shows the convergence with *C1* = $1e^{-3}$. Table 7 summarises the nature of convergence. With high values of *C1*, no additional feasible solutions are generated after the first generation (The value 2182 is the makespan of the single feasible solution introduced in the initial population).

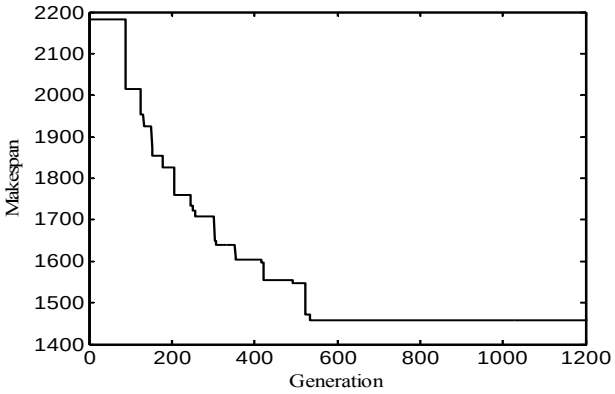

Fig 3 Variation of makespan for 15x5 problem (*C1*= $1e^{-3}$)

It is observed that, the probability of getting more feasible schedules is increased while the value of *C1*= 0.001 or higher. A converged makespan value close to the optimum value has been obtained for *C1*= 1.0 e^{-7}. The low value of *C1* helps the algorithm to explore new search areas with more feasible solutions. The study has shown that the solution quality has not improved with *C1* values lower than $1e^{-7}$.

Table 7 Convergence behavior with different values of *C1*

C1	Convergence Behavior	Makespan
2	Not getting other feasible solutions	2182
1	Not getting other feasible solutions	2182
0.1	Not getting other feasible solutions	2182
0.01	Not getting other feasible solutions	2182
$1e^{-3}$	Getting Converging solutions	1457
$1e^{-4}$	Getting Converging solutions	1257
$1e^{-5}$	Getting Converging solutions	1173
$1e^{-6}$	Getting Converging solutions	1113
$1e^{-7}$	Getting Converging solutions close to optimum	1020
$1e^{-8}$	Getting Converging solutions close to optimum	1020

4.5 Effect of Swarm confidence factor (*C2*)

This factor represents the influence of social collaboration among the particles, which compares a particle's current position to that of the best particle in the swarm. The effect of *C2* has been studied on a 6x6 problem (FT-06). The convergence behavior of makespan with generation for *C2=4* is shown in Fig 4.

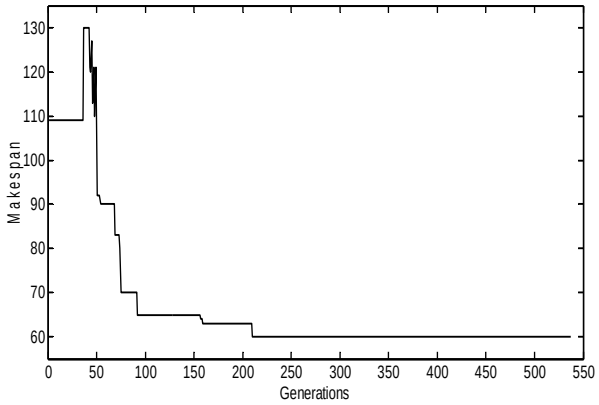

Fig 4 Variation of makespan for 6x6 problem (*C2*=4)

Table 8. summarizes the study on influence of swarm confidence factor on the performance of the algorithm. A very low value of *C2* (0.001) did not lead to generation of additional feasible schedules. As the value of *C2* is increased many feasible schedules with higher makespan values have been obtained. The best performance was obtained at *C2*=4, with a converged makespan value of 60, which is very close to optimum. Much higher values of *C2* resulted in solutions which are away from optimum.

Table 8 Convergence behavior at different values of C2

C2	Convergence behavior	Makespan
0.001	Not getting further feasible solutions	109
0.1	Getting feasible solutions, converges to a higher value	130
1	Getting feasible solutions but converges at higher value	130
2	Getting feasible solutions but converges at higher value	130
3	Getting feasible solutions but converges very slowly at higher value	130
3.5	Getting feasible solutions but converges very slowly at higher value	130
3.8	Getting feasible solutions and converges at lower value	73
4	**Getting feasible solutions and converges at lower value close to optimum**	60
4.5	Getting feasible solutions and converges at value away from optimum	75

The PSO algorithm for the JSS problem is coded in MATLAB and run on an Intel Pentium IV PC with 512 MB memory. This study has helped to identify the influence of different PSO parameters for a Job Shop Scheduling problem. The final setting of PSO parameters for JSSP is summarized in Table 9.

Table 9 Final parameter setting suggested in PSO algorithm for the JSSP

Parameter	Value
Swarm size (Ps)	500
Inertia factor (w)	1.0 e-6 to 1.0 e-7
Self confidence factor (C1)	1.0 e-7.
Swarm confidence factor (C2)	4
Number of generations (Mg)	Vary-with problem size

It is seen that large population leads to better solutions as it ensures more feasible solutions. However a swarm size of more than 500 is not recommended. The influence of inertia factor and self confidence factor are relatively less compared to swarm confidence factor in obtaining global optimum solutions. When the problem size is large, since no mechanism is employed in this study to screen the infeasible solutions, the algorithm took more number of generations to reach convergence. If a hybrid algorithm is employed with a tabu search the number of unwanted searches can be avoided and convergence behavior can be improved.

5. Conclusion

This paper described an application of PSO in a discrete optimization problem by incorporating SPV rule. The present study is a humble attempt to investigate the application of Particle Swarm Optimization algorithm in solving the Static Job Shop Scheduling problem which is one of the hardest combinatorial optimization problems. The effect of five PSO parameters (Swarm size, Number of generations, Inertia weight, Self confidence factor and Swarm confidence factor) on the performance of the algorithm in benchmark problems is studied and an optimum combination of the parameters is suggested. Being a recent technique, the parameter setting for PSO still needs tuning to get better performance in job shop environment. It is hoped that the findings from the investigations will help in obtaining a better insight into the algorithm and will lead to application of PSO in other areas.

References:

[1] Eberhart R.C., J.Kennedy, A new optimizer using particle swarm theory, *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, Japan, 1995, pp.39-43.

[2] Fatih Tasgetiren M, Yun-Chia Liang, Mehmet Sevkli, Gunes Gencyilmaz, *Particle Swarm Optimization Algorithm for Makespan and Maximum Lateness Minimization in Permutation Flowshop Sequencing Problem*, Department of Management, Fatih University, 2005.

[3] Fischer.H, Thompson.G.L, Probabalistic learning combinations of local job shop scheduling rules, *Industrial scheduling*, Prentice-hall, Engle Cliffs, NJ, 1963, pp.225-251.

[4] Jayamohan M.S, Dispatching rules for Dynamic Job shop scheduling with time based and cost based measures of performance. Thesis, Dept. of Humanities and Social sciences, IIT. Madras.1999'

[5] Jose Fernado Goncalves,Jorge Jose de Magalhades Mendes, Mauricio G.C.Resende, A Hybrid Genetic Algorithm for Job Shop Scheduling Problem, *AT&T Labs Research technical report*, 2002.

[6] Loan Cristian Trelea, The particle swarm optimization algorithm: convergence analysis and parameter selection", INA P-G, UMR Génie et Microbiologie des Procédés Alimentaires,Thiverval-Grignon, France,2003.

[7] Michael Pinedo, *Scheduling-Theory,Algorithms and systems*. Prentice- Hall, New Jersey,1995.

[8] Vipin.J.S., *Multi-objective flow shop scheduling using nature inspired algorithms*, Thesis, University of Kerala, 2005.

[9] Yamada, T. and R. Nakano. Genetic Algorithm for Job Shop Scheduling Problems. *Proceedings of Modern heuristics for decision support*, Unicom seminar, London, 1997, pp. 67-81.

[10] Zhigang Lian, Xingsheng, Bin Jiao, A similar particle swarm optimization algorithm for permutation flowshop scheduling to minimize makespan. *Journal of applied mathematics and computation*. 2005.