19

A Stochastically Perturbed Particle Swarm Optimization for Identical Parallel Machine Scheduling Problems

Mehmet Sevkli[1] and Aise Zulal Sevkli[2]
[1]*King Saud University, Faculty of Engineering, Department of Industrial Engineering, Riyadh*
[2]*King Saud University, College of Computer and Information Sciences, Department of Information Technology, Riyadh*
Kingdom of Saudi Arabia

1. Introduction

Identical parallel machine scheduling (PMS) problems with the objective of minimizing makespan (C_{max}) is one of the well known NP-hard [1] combinatorial optimization problems. It is unlikely to obtain optimal schedule through polynomial time-bounded algorithms. Small size instances of PMS problem can be solved with reasonable computational time by exact algorithms such as branch-and-bound [2, 3], and the cutting plane algorithm [4]. However, as the problem size increases, the computation time of exact methods increases exponentially. On the other hand, heuristic algorithms generally have acceptable time and memory requirements, but do not guarantee optimal solution. That is, a feasible solution is obtained which is likely to be either optimal or near optimal. The well-known longest processing time (LPT) rule of Graham [5] is a sort of so called list scheduling algorithm. It is known that the rule works very well when makespan is taken as the single criterion [6]. Later, Coffman et al. [7] proposed MULTIFIT algorithm that considers the relation between bin-packing and maximum completion time problems. Yue [8] showed that the MULTIFIT heuristic is not guaranteed to perform better than LPT for every problem. Gupta and Ruiz-Torres [9] developed a LISTFIT algorithm that combines the bin packing method of the MULTIFIT heuristic with multiple lists of jobs. Min and Cheng [10] introduced a genetic algorithm (GA) that outperformed simulated annealing (SA) algorithm. Lee et al. [11] proposed a SA algorithm for the PMS problems and compared their results with the LISTFIT algorithm. Tang and Luo [12] developed a new iterated local search (ILS) algorithm that is based on varying number of cyclic exchanges.

Particle swarm optimization (PSO) is based on the metaphor of social interaction and communication among different spaces in nature, such as bird flocking and fish schooling. It is different from other evolutionary methods in a way that it does not use the genetic operators (such as crossover and mutation), and the members of the entire population are maintained through out the search procedure. Thus, information is socially shared among

individuals to direct the search towards the best position in the search space. In a PSO algorithm, each member is called a particle, and each particle moves around in the multi-dimensional search space with a velocity constantly updated by the particle's experience, the experience of the particle's neighbours, and the experience of the whole swarm. PSO was first introduced to optimize various continuous nonlinear functions by Eberhart and Kennedy [13]. PSO has been successfully applied to a wide range of applications such as automated drilling [14], home care worker scheduling [15], neural network training [16], permutation flow shop sequencing problems [17], job shop scheduling problems [18], and task assignment [19]. More information about PSO can be found in Kennedy et al. [20].

The organization of this chapter is as follows: Section II introduces PMS problem, the way how to represent the problem, lower bound of the problem and overview of the classical PSO algorithm. The third section reveals the proposed heuristic algorithm. The computational results are reported and discussed in the fourth section, while the fifth section includes the concluding remarks.

2. Background

2.1 Problem description

The problem of identical parallel machine scheduling is about creating schedules for a set $J=\{J_1, J_2, J_3, ..., J_n\}$ of n independent jobs to be processed on a set $M=\{M_1, M_2, M_3, ..., M_m\}$ of m identical machines. Each job should be carried out on one of the machines, where the time required for processing job i on a machine is denoted by p_i. The subset of jobs assigned to machine M_i in a schedule is denoted by S_{M_i}. Once a job begins processing, it must be completed without interruption. Furthermore, each machine can process one job at a time, and there is no precedence relation between the jobs. The aim is to find a permutation for the n jobs to machines from set M so as to minimize the maximum completion time, in other words the makespan. The problem is denoted as $P||C_{max}$, where P represents identical parallel machines, the jobs are not constrained, and the objective is to obtain the minimum length schedule. An integer programming formulation of the problem that minimize the makespan is as follows: [5]

$$min\ y$$

subject to:

$$\sum_{j=1}^{m} x_{ij} = 1, \quad 1 \le i \le n, \tag{1}$$

$$y - \sum_{i=1}^{n} p_i x_{ij} \ge 0, \quad 1 \le j \le m \tag{2}$$

where the optimal value of y is C_{max} and x_{ij}=1 when job i is assigned to machine j, otherwise x_{ij}=0.

2.2 Solution representation and lower bound

The solution for the PMS problem is represented as a permutation of integers Π= *{1,..., n}* where Π defines the processing order of the jobs. As mentioned in the text above, three versions of the PSO algorithm are compared in terms of solution quality and CPU time.

In continuous based PSO by Tasgetiren et al. [17], PSO_{spv} , particles themselves do not present permutations. Instead, the SPV rule is used to derive a permutation from the position values of the particle. In discrete PSO by Pan et al.[21] and the proposed algorithm (SPPSO), on the other hand, the particles present permutations themselves.

Jobs	1	2	3	4	5	6	7	8	9
p_i	7	7	6	6	5	5	4	4	4

Table 1. An example of 9-job × 4-machine PMS problem

For all of the three algorithms, the process of finding makespan value for a particle can be illustrated by an example. Namely, let's assume a permutation vector of Π= *{1 8 3 4 5 6 7 2 9}*. By considering 4 parallel machines and 9 jobs, whose processing times are given in Table 1, the makespan value of the given vector is depicted in Figure 1.

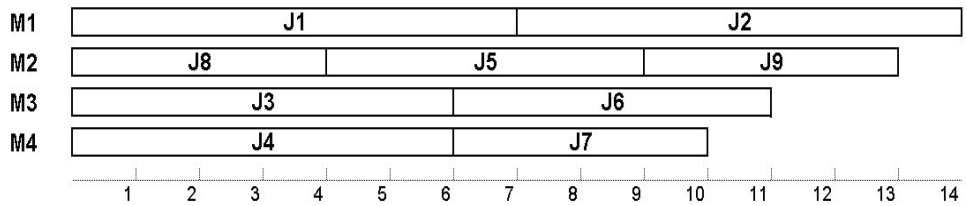


Fig. 1. Shedule generated from random sequence

According to the schedule, each value of the vector is iteratively assigned to the most available machine. First four elements of the permutation vector (1,8,3,4) are assigned to the four machines respectively. The remaining jobs are assigned one by one to the first machine available. For instance, 5 goes to second machine (M_2), since it is the first machine released. If there is more than one available machine at the time, the job will be assigned randomly (ties can be broken arbitrarily). The makespan value of the given sequence is $C_{max}(\Pi)=14$, as can easily be seen in figure 1.

The lower bound for $P||C_{max}$ is calculated as follows [22]:

$$LB(C_{\max}) = \max\left\{\left\lceil \frac{1}{m}\sum_{i=1}^{n} p_i \right\rceil \;\; ; \;\; \max_i\{p_i\}\right\} \tag{3}$$

It is obtained by assuming that preemption is not allowed. If $C_{max}(\Pi)=LB(C_{max})$, the current solution(Π) is optimum. So, lower bound will be used as one of the termination criteria

throughout this chapter. The lower bound of the example presented in Table 1 can be calculated as:

$$LB(C_{\max}) = \max\left\{\left\lceil \frac{1}{4}\sum_{i=1}^{9} p_i \right\rceil \;\; ; \;\; \max_i\{p_i\}\right\} = \max(12;7) = 12$$

2.3 Classic Particle Swarm Optimization

In PSO, each single solution, called a particle, is considered as an individual, the group becomes a swarm (population) and the search space is the area to explore. Each particle has a fitness value calculated by a fitness function, and a velocity to fly towards the optimum. All particles fly across the problem space following the particle that is nearest to the optimum. PSO starts with an initial population of solutions, which is updated iteration-by-iteration. The principles that govern PSO algorithm can be stated as follows:

- n dimensional position ($X_i = (x_{i1}, x_{i2}, ..., x_{in})$) and velocity vector ($V_i = (v_{i1}, v_{i2}, ..., v_{in})$ for i^{th} particle starts with a random position and velocity.
- Each particle knows its position and value of the objective function for that position. The best position of i^{th} particle is donated as $P_i = (p_{i1}, p_{i2}, ..., p_{in})$, and the best position of the whole swarm as, $G = (g_1, g_2, ..., g_n)$ respectively. The PSO algorithm is governed by the following main equations:

$$\begin{aligned} v_{in}^{t+1} &= w v_{in}^{t} + c_1 r_1 (p_{in}^{t} - x_{in}^{t}) + c_2 r_2 (g_i^{t} - x_{in}^{t}), \\ x_{in}^{t+1} &= v_{in}^{t+1} + x_{in}^{t} \end{aligned} \tag{4}$$

where t represents the iteration number, w is the inertia weight which is a coefficient to control the impact of the previous velocities on the current velocity. c_1 and c_2 are called learning factors. r_1 and r_2 are uniformly distributed random variables in [0,1].

The original PSO algorithm can optimize problems in which the elements of the solution space are continuous real numbers. The major obstacle for successfully applying PSO to combinatorial problems in the literature is due to its continuous nature. To remedy this drawback, Tasgetiren et al. [17] presented the smallest position value (SPV) rule. Another approach to tackle combinatorial problems with PSO is done by Pan et al. [21]. They generate a similar PSO equation to update the particle's velocity and position vectors using one and two cut genetic crossover operators.

3. The proposed Stochastically Perturbed Particle Swarm Optimization algorithm

In this chapter, a stochastically perturbed particle swarm optimization algorithm (SPPSO) is proposed for the PMS problems. The initial population is generated randomly. Initially, each individual with its position, and fitness value is assigned to its personal best (i.e., the best value of each individual found so far). The best individual in the whole swarm with its position and fitness value, on the other hand, is assigned to the global best (i.e., the best particle in the whole swarm). Then, the position of each particle is updated based on the personal best and the global best. These operations in SPPSO are similar to classical PSO

algorithm. However, the search strategy of SPPSO is different. That is, each particle in the swarm moves based on the following equations.

$$\begin{aligned} s_1 &= w^t \oplus \eta(X_i^t) \\ w^{t+1} &= w \cdot \beta \\ s_2 &= c_1 \oplus \eta(P_i^t) \\ s_3 &= c_2 \oplus \eta(G^t) \\ X_i^{t+1} &= best(s_1; s_2; s_3) \end{aligned} \tag{5}$$

At each iteration, the position vector of each particle, its personal best and the global best are considered. First of all, a random number of U(0,1) is generated to compare with the inertia weight to decide whether to apply *Insert* function(η) to the particle or not.

Insert function(η) implies the insertion of a randomly chosen job in front (or back sometimes) of another randomly chosen job. For instance, for the PMS problem, suppose a sequence of {3, 5, 6, 7, 8, 9, 1, 2, 4}. In order to apply *Insert* function, we also need to derive two random numbers; one is for determining the job to change place and the other is for the job in front of which the former job is to be inserted. Let's say those numbers are 3 and 5 (that is, the third job will move in front of the fifth. In other words, job no.6 will be inserted in front of job no.8 {3, 5, 6, 7, 8, 9, 1, 2, 4}). The new sequence will be {3, 5, 7, 8, 6, 9, 1, 2, 4}.

If the random number chosen is less than the inertia weight, the particle is manipulated with this *Insert* function, and the resulting solution, say s_1, is obtained. Meanwhile, the inertia weight is discounted by a constant factor at each iteration, in order to tighten the acceptability of the manipulated particle for the next generation, that is, to diminish the impact of the randomly operated solutions on the swarm evolution.

The next step is to generate another random number of U(0,1) to be compared with c_1, cognitive parameter, to make a decision whether to apply *Insert* function to personal best of the particle considered. If the random number is less than c_1, then the personal best of the particle undertaken is manipulated and the resulting solution is spared as s_2. Likewise, a third random number of U(0,1) is generated for making a decision whether to manipulate the global best with the *Insert* function. If the random number is less than c_2, social parameter, then *Insert* is applied to the global best to obtain a new solution of s_3. Unlike the case of inertia weight, the values of c_1 and c_2 factors are not increased or decreased iteratively, but are fixed at 0.5. That means the probability of applying *Insert* function to the personal and global bests remains the same. The new replacement solution is selected among s_1, s_2 and s_3, based on their fitness values. This solution may not always be better than the current solution. This is to keep the swarm diverse. The convergence is traced by checking the personal best of each new particle and the global best. As it is seen, proposed equations have all major characteristics of the classical PSO equations. The following pseudo-code describes in detail the steps of the SPPSO algorithm.

It can be seen from the pseudo-code of the algorithm that the algorithm has all major characteristics of the classical PSO, the search strategy of the algorithm is different in a way

that the new solution is selected among s_1, s_2 and s_3, based on their fitness values. The selected particle may be worse than the current solution that keep the swarm diverse. The convergence is obtained by changing the personal best of each new particle and the global best.

```
Begin
   Initialize particles (population) randomly
   For each particle
      Calculate fitness value
      Set to position vector and fitness value as personal best (P_i^t)
      Select the best particle and its position vector as global best(G_t)
   End
   Do{
      Update inertia weight
      For each particle
       Apply insert with the probability of inertia weight (s1)
       Apply insert to (P_i^t) with the probability of c1 (s2)
       Apply insert to (G_t) with the probability of c2 (s3)
       Select the best one among the s1,s2 and s3
       Update personal best (P_i^t)
      End
      Update global best (G_t)
   }While (Maximum Iteration is not reached)

   End
```

Fig. 2. Pseudo code of the proposed SPPSO algorithm for PMS problem

4. Computational results

In this section, a comparison study is carried out on the effectiveness of the proposed SPPSO algorithm. SPPSO was exclusively tested in comparison with two other recently introduced PSO algorithms: PSO_{spv} algorithm of Tasgetiren et al. [17] and DPSO algorithm of Pan et al. [21]. Two experimental frameworks, namely E1 and E2, are considered implying the type of discrete uniform distribution used to generate job-processing times. That is, the processing time of each job is generated by using uniform distribution of U[1,100] and U[100,800] for experiments E1 and E2 respectively. All SPPSO, PSO_{spv} and DPSO algorithms are coded in C and run on a PC with the configuration of 2.6 GHz CPU and 512MB memory. The size of the population considered by all algorithms is the number of jobs (n).

For SPPSO and DPSO, the social and cognitive parameters were taken as $c_1 = c_2 = 0.5$, initial inertia weight is set to 0.9 and never decreased below 0.40, and the decrement factor β is fixed at 0.999. For the PSO_{spv} algorithm, the social and cognitive parameters were fixed at $c_1 = c_2 = 2$, initial inertia weight is set to 0.9 and never decreased below 0.40, and the decrement factor β is selected as 0.999. The algorithms were run for $20000/n$ iterations. All the there algorithms were applied without embedding any kind of local search.

The instances of problems were generated for 3, 4, 5, 10, 20, 30, 40, 50 machines and 20, 50, 100, 200, and 500 jobs. In order to allow for the variations, 10 instances are generated for each problem size. Hence, the overall number of instances added up to 350. The measures considered in this chapter are mainly about the solution quality. The performance measure

is a relative quality measure, C/LB, where C is the result achieved (makespan) by the algorithm and LB is the lower bound of the instance which is calculated in Eq.(3). Once C catches LB, the index results 1.0, otherwise remains larger.

		PSO_{spv}			DPSO			SPPSO		
m	**n**	**min**	**avg**	**max**	**min**	**avg**	**max**	**min**	**avg**	**max**
3	20	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	50	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	100	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	200	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	500	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
4	20	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	50	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	100	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	200	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	500	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
5	20	1.000	1.001	1.005	1.000	1.001	1.005	1.000	1.001	1.005
	50	1.000	1.000	1.002	1.000	1.000	1.000	1.000	1.000	1.000
	100	1.000	1.000	1.001	1.000	1.000	1.000	1.000	1.000	1.000
	200	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	500	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
10	20	1.050	1.091	1.168	1.050	1.091	1.168	1.050	1.091	1.168
	50	1.000	1.002	1.004	1.004	1.005	1.008	1.000	1.001	1.004
	100	1.000	1.001	1.002	1.002	1.003	1.005	1.000	1.000	1.002
	200	1.001	1.002	1.002	1.001	1.002	1.002	1.000	1.001	1.001
	500	1.001	1.001	1.002	1.001	1.001	1.001	1.000	1.000	1.001
20	50	1.015	1.026	1.050	1.033	1.043	1.053	1.009	1.024	1.050
	100	1.007	1.009	1.013	1.025	1.029	1.037	1.004	1.009	1.013
	200	1.006	1.007	1.010	1.013	1.015	1.018	1.004	1.006	1.008
	500	1.004	1.006	1.007	1.006	1.007	1.009	1.002	1.003	1.005
30	50	1.066	1.154	1.266	1.076	1.161	1.266	1.066	1.154	1.266
	100	1.013	1.022	1.028	1.043	1.061	1.072	1.019	1.029	1.039
	200	1.009	1.017	1.021	1.032	1.037	1.043	1.014	1.017	1.020
	500	1.009	1.011	1.015	1.011	1.016	1.021	1.008	1.009	1.011
40	50	1.282	1.538	1.707	1.282	1.538	1.707	1.282	1.538	1.707
	100	1.033	1.047	1.067	1.084	1.115	1.142	1.042	1.055	1.061
	200	1.021	1.028	1.034	1.054	1.067	1.075	1.028	1.035	1.042
	500	1.016	1.019	1.022	1.025	1.030	1.031	1.016	1.020	1.026
50	100	1.070	1.088	1.114	1.156	1.184	1.220	1.070	1.097	1.140
	200	1.036	1.044	1.053	1.081	1.096	1.106	1.049	1.057	1.065
	500	1.023	1.027	1.030	1.034	1.043	1.046	1.028	1.032	1.035
Average		**1.019**	**1.033**	**1.046**	**1.029**	**1.044**	**1.058**	**1.020**	**1.034**	**1.048**

Table 2. Results for experiment E1:p~U(1,100)

		PSO_{spv}			DPSO			SPPSO		
m	**n**	**min**	**avg**	**max**	**min**	**avg**	**max**	**min**	**avg**	**max**
3	20	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	50	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	100	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	200	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	500	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
4	20	1.000	1.001	1.001	1.000	1.000	1.001	1.000	1.000	1.001
	50	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	100	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	200	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	500	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
5	20	1.001	1.002	1.003	1.001	1.002	1.003	1.001	1.001	1.002
	50	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	100	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	200	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	500	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
10	20	1.046	1.071	1.128	1.040	1.068	1.128	1.040	1.068	1.128
	50	1.001	1.003	1.005	1.003	1.006	1.010	1.001	1.002	1.003
	100	1.000	1.001	1.001	1.003	1.004	1.004	1.001	1.001	1.001
	200	1.000	1.000	1.001	1.001	1.002	1.003	1.000	1.001	1.001
	500	1.000	1.000	1.000	1.000	1.001	1.002	1.000	1.000	1.001
20	50	1.022	1.067	1.113	1.026	1.037	1.054	1.011	1.019	1.025
	100	1.012	1.016	1.021	1.012	1.023	1.029	1.006	1.006	1.007
	200	1.002	1.005	1.010	1.011	1.014	1.017	1.003	1.003	1.004
	500	1.000	1.001	1.002	1.005	1.007	1.009	1.001	1.002	1.003
30	50	1.080	1.122	1.195	1.096	1.128	1.195	1.080	1.123	1.195
	100	1.016	1.029	1.043	1.038	1.055	1.065	1.012	1.015	1.016
	200	1.012	1.017	1.022	1.027	1.033	1.037	1.008	1.010	1.012
	500	1.005	1.006	1.007	1.012	1.015	1.017	1.005	1.007	1.008
40	50	1.268	1.378	1.534	1.268	1.378	1.534	1.268	1.378	1.534
	100	1.024	1.069	1.095	1.077	1.093	1.102	1.022	1.029	1.036
	200	1.016	1.022	1.028	1.046	1.057	1.066	1.015	1.019	1.021
	500	1.009	1.010	1.011	1.022	1.025	1.027	1.011	1.012	1.014
50	100	1.034	1.052	1.084	1.121	1.154	1.166	1.047	1.060	1.084
	200	1.007	1.011	1.022	1.076	1.086	1.099	1.026	1.032	1.035
	500	1.001	1.003	1.007	1.034	1.039	1.044	1.015	1.019	1.022
Average		**1.016**	**1.025**	**1.038**	**1.026**	**1.035**	**1.046**	**1.016**	**1.023**	**1.033**

Table 3. Results for experiment E2:p~U(100,800)

		PSO_{spv}				DPSO				SPPSO			
		p~U(1,100)		p~U(100,800)		p~U(1,100)		p~U(100,800)		p~U(1,100)		p~U(100,800)	
m	**n**	**nopt**	**CPU**	**nopt**	**CPU**	**nopt**	**CPU**	**nopt**	**CPU**	**nopt**	**CPU**	**nopt**	**CPU**
3	20	10	0.008	10	0.266	10	0.014	10	0.308	10	0.005	10	0.241
	50	10	0.015	10	0.571	10	0.008	10	0.077	10	0.003	10	0.029
	100	10	0.038	9	2.020	10	0.010	10	0.091	10	0.005	10	0.023
	200	10	0.310	9	8.054	10	0.044	10	0.239	10	0.019	10	0.062
	500	10	3.172	10	57.143	10	0.259	10	1.437	10	0.083	10	0.180
4	20	10	0.112	1	1.007	10	0.201	3	0.383	10	0.096	4	0.406
	50	10	0.013	2	0.836	10	0.055	7	0.294	10	0.024	10	0.202
	100	10	0.027	9	1.676	10	0.059	8	0.355	10	0.019	10	0.126
	200	10	0.202	9	4.391	10	0.115	7	0.865	10	0.053	10	0.239
	500	10	3.169	10	11.438	10	1.085	10	3.635	10	0.234	10	0.485
5	20	7	0.206	0	0.603	8	0.218	0	0.363	9	0.233	0	0.430
	50	9	0.084	8	0.678	10	0.134	1	0.274	10	0.052	5	0.286
	100	8	0.028	5	2.308	10	0.199	3	0.424	10	0.072	9	0.255
	200	9	0.408	9	4.877	10	0.397	2	1.023	10	0.127	9	0.357
	500	6	3.177	9	15.739	10	2.502	3	4.576	10	0.453	9	0.720
10	20	0	0.414	0	0.429	0	0.374	0	0.401	0	0.559	0	0.449
	50	5	0.799	0	0.922	0	0.322	0	0.329	8	0.344	0	0.399
	100	4	0.778	1	2.853	0	0.512	0	0.542	8	0.354	0	0.435
	200	0	0.208	1	10.314	0	1.189	0	1.259	5	0.630	0	0.673
	500	0	3.194	5	52.414	0	4.869	0	5.207	2	1.347	0	1.439
20	50	0	0.960	0	1.514	0	0.438	0	0.446	0	0.450	0	0.471
	100	0	2.840	0	2.883	0	0.627	0	0.650	0	0.510	0	0.551
	200	0	10.385	0	10.671	0	1.397	0	1.451	0	0.806	0	0.862
	500	0	52.525	0	67.284	0	5.334	0	5.643	0	1.750	0	1.853
30	50	0	1.636	0	1.631	0	0.459	0	0.469	0	0.485	0	0.504
	100	0	2.842	0	2.898	0	0.643	0	0.674	0	0.561	0	0.607
	200	0	10.495	0	11.330	0	1.455	0	1.532	0	0.906	0	0.972
	500	0	59.247	0	66.154	0	5.550	0	5.940	0	1.978	0	2.324
40	50	0	1.684	0	1.636	0	0.497	0	0.522	0	0.518	0	0.590
	100	0	2.984	0	2.873	0	0.699	0	0.742	0	0.620	0	0.726
	200	0	10.625	0	10.531	0	1.568	0	1.667	0	1.022	0	1.164
	500	0	59.573	0	65.551	0	5.829	0	6.292	0	2.244	0	2.548
50	100	0	3.658	0	3.626	0	0.813	0	0.861	0	0.697	0	0.745
	200	0	10.702	0	10.556	0	1.680	0	1.763	0	1.140	0	1.247
	500	0	65.759	0	65.793	0	6.117	0	6.465	0	2.521	0	2.844
Total		**148**		**117**		**148**		**94**		**172**		**126**	
Average			**8.922**		**14.385**		**1.305**		**1.634**		**0.598**		**0.727**

Table 4. Results for both experiments

The results for the instances with different sizes are shown in Table 3 and Table 4, where the minimum, average and maximum of the C/LB ratio are presented. Each line summarizes the values for the 10 instances of each problem size, where 10 replications are performed for each instance.

The result for the experiment E1, in which processing times are generated by using U(1,100) are summarized in Table 2. In this experiment, it is found that the minimum, average and maximum values of the ratios are quite similar for SPPSO and PSO_{spv}. On the other hand, SPPSO and PSO_{spv} performed better than DPSO.

The result for the experiment E2 in which processing times are generated by using U(100,800) are summarized in Table 3. In this experiment, there is also no significant difference between SPPSO and PSO_{spv}. However, in terms of max ratio performance SPPSO performed slightly better than PSO_{spv}. In addition, PSO_{spv} and SPPSO are also better than DPSO for all the three ratios in this experiment.

Table 4 shows the number of times the optimum is reached within the group (nopt) for each algorithm and their average CPU times in seconds for each experiment. Total number of optimum solutions obtained by PSO_{spv}, DPSO and SPPSO for the both experiment are summarized as (148,148,172) and (117, 94,126) respectively. Here, the superiority of SPPSO over PSO_{spv} and DPSO is more pronounced in terms of number of total optimum solutions obtained.

In terms of the average CPU, SPPSO shows better performance than PSO_{spv} and DSPO. SPPSO (0.598, 0.727) is about 15 times faster than PSO_{spv} (8.922, 14,395) and about 2 times faster than DPSO (1.305, 1.634) in both experiments.

5. Conclusion

In this chapter, a stochastically perturbed particle swarm optimization algorithm (SPPSO) is proposed for identical parallel machine scheduling (PMS) problems. The SPPSO has all major characteristics of the classical PSO. However, the search strategy of SPPSO is different. The algorithm is applied to (PMS) problem and compared with two recent PSO algorithms. The algorithms are kept standard and not extended by embedding any local search. It is concluded that SPPSO produced better results than DPSO and PSO_{spv} in terms of number of optimum solutions obtained. In terms of average relative percent deviation, there is no significant difference between SPPSO and PSO_{spv}. However, they are better than DPSO.

It also should be noted that, since PSO_{spv} considers each particle based on three key vectors; position (X_i), velocity (V_i), and permutation (Π_i), it consumes more memory than SPPSO. In addition, since DPSO uses one and two cut crossover operators in every iteration, implementation of DPSO to combinatorial optimization problems is rather cumbersome. The proposed algorithm can be applied to other combinatorial optimization problems such as flow shop scheduling, job shop scheduling etc. as future work.

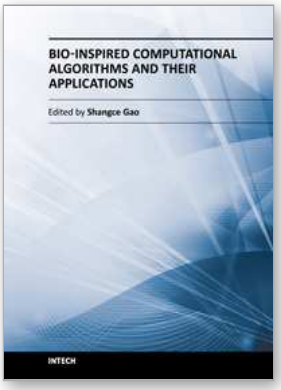
6. References

[1] Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP completeness. Freeman, San Francisco, California

[2] Van deVelde, S. L. (1993) "Duality-based algorithms for scheduling unrelated parallel machines". ORSA Journal on Computing, 5, 192–205.
[3] Dell Amico, M., Martello, S. (1995) "Optimal scheduling of tasks on identical parallel processors", ORSA Journal on Computing 7, 191-200.
[4] Mokotoff, E. (2004). "An exact algorithm for the identical parallel machine scheduling problem", European Journal of Operational Research, 152, 758–769.
[5] Graham, R. L., (1969). "Bounds on multiprocessor timing anomalies. SIAM", Journal of Applied Mathematics, 17, 416-429.
[6] Blazewicz, J ., Ecker, K., Pesch, E., Schmidt, G., and Weglarz, J., (1996), "Scheduling Computer and Manufacturing Systems". (Berlin: Springer).
[7] Coffman EG, Garey MR, Johnson DS, (1978). "An application of bin-packing to multi-processor scheduling". SIAM Journal of Computing 7, 1–17.
[8] Yue, M., (1990) "On the exact upper bound for the MULTIFIT processor algorithm", Annals of Operations Research, 24, 233-259
[9] Gupta JND, Ruiz-Torres AJ (2001) "A LISTFIT heuristic for minimizing makespan on identical parallel machines". Prod Plan Control 12:28–36
[10] Min, L.,Cheng, W.(1999) "A genetic algorithm for the minimizing the makespan in case of scheduling identical parallel machines", Artificial Intelligence in Engineering 13, 399-403
[11] Lee WC, Wu CC, Chen P (2006) "A simulated annealing approach to makespan minimization on identical parallel machines". Intelligent Journal of Advanced Manufacturing Technology 31, 328–334.
[12] Tang L,, Luo J. (2006) "A New ILS Algorithm for Parallel Machine Scheduling Problems", Journal of Intelligent Manufacturing 17 (5), 609-619
[13] Eberhart, R.C., and Kennedy, J., (1995) "A new optimizer using particle swarm theory, Proceedings of the Sixth International Symposium on Micro Machine and Human Science", Nagoya, Japan, 1995, 39-43.
[14] Onwubolu, G.C. and M. Clerc. (2004). "Optimal Operating Path for Automated Drilling Operations by a New Heuristic Approach Using Particle Swarm Optimisation." International Journal of Production Research 42(3), 473-491
[15] Akjiratikarl,C., Yenradee,P., Drake,P.R. (2007), "PSO-based algorithm for home care worker scheduling in the UK", Computers & Industrial Engineering 53(4), 559-583
[16] Van den Bergh, F. and A.P. Engelbecht. (2000). "Cooperative Learning in Neural Networks Using Particle Swarm Optimizers." South African Computer Journal 26, 84-90.
[17] Tasgetiren, M.F., Liang, Y.C., Sevkli, M. and Gencyilmaz, G, (2007) "Particle Swarm Optimization Algorithm for Makespan and Total Flowtime Minimization in Permutation Flowshop Sequencing Problem", European Journal of Operational Research 177 (3), 1930-1947
[18] Sha,D.Y., Hsu, C-Y, (2006) "A hybrid particle swarm optimization for job shop scheduling problem", Computers & Industrial Engineering, 51(4),791-808
[19] Salman, A., I. Ahmad, and S. Al-Madani. (2003). "Particle Swarm Optimization for Task Assignment Problem." Microprocessors and Microsystems 26, 363-371.
[20] Kennedy, J., R.C. Eberhart, and Y. Shi. (2001). Swarm Intelligence, San Mateo, Morgan Kaufmann, CA, USA.

[21] Pan, Q-K, Tasgetiren, M.F., and Liang,Y-C, (2008) A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem, Computers & Operations Research, Vol.35(9), 2807-2839.

[22] Pinedo, M. (1995) Scheduling: theory, algorithm, and systems, Prentice hall, Englewood cliffs, New Jersey



Bio-Inspired Computational Algorithms and Their Applications

Edited by Dr. Shangce Gao

ISBN 978-953-51-0214-4
Hard cover, 420 pages
Publisher InTech
Published online 07, March, 2012
Published in print edition March, 2012

Bio-inspired computational algorithms are always hot research topics in artificial intelligence communities. Biology is a bewildering source of inspiration for the design of intelligent artifacts that are capable of efficient and autonomous operation in unknown and changing environments. It is difficult to resist the fascination of creating artifacts that display elements of lifelike intelligence, thus needing techniques for control, optimization, prediction, security, design, and so on. Bio-Inspired Computational Algorithms and Their Applications is a compendium that addresses this need. It integrates contrasting techniques of genetic algorithms, artificial immune systems, particle swarm optimization, and hybrid models to solve many real-world problems. The works presented in this book give insights into the creation of innovative improvements over algorithm performance, potential applications on various practical tasks, and combination of different techniques. The book provides a reference to researchers, practitioners, and students in both artificial intelligence and engineering communities, forming a foundation for the development of the field.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821