Desheng LI[1], Qian HE[2]

Anhui Science and Technology University (1), Beijing University of Posts & Telecommunications (2)

A version of cooperative multi-swarm PSO using electoral mechanism to solve hybrid flow shop scheduling problem

Abstract. *Hybrid flow shop (HFS) scheduling problem is a kind of scheduling consisted of a series of stages, in which there exist more than one parallel machine. In this paper, we propose a meta-heuristics using a version of cooperative multi-swarm PSO algorithm for the HFS with minimum makespan objective. The main contribution of this algorithm is to import an electoral mechanism to accelerate the converging and a disturbance approach to help escape from local optima. Finally, experiments show that the algorithm outperforms all the compared in the HFS problem.*

Streszczenie. *W artykule zaproponowano nowy rojowy algorytm do rozwiązywania problemu w harmonogramie dostępu typu HFS. Nowy mechanizm wyboru pozwala na przyśpieszenie konwergencji. (**Kooperacyjny algorytm rojowy PSO wykorzystujący mechanizm wyboru do rozwiązywania problemów harmonogramu w systemie HFS**)*

Keywords: Hybrid Flow Shop; Particle Swarm Optimization; Electoral Mechanism.
Słowa kluczowe: algorytm rojowy, harmonogram dostępu do sieci.

1. Introduction

Production scheduling problems are often decision-making process which plays an important role with certain guiding functions in most manufacturing and production system. In such problems, the *n*-job and *k*-stage hybrid flow shop (HFS) scheduling problem is a kind of scheduling consisted of a series of stages, in which there exist more than one parallel machine. But there also exist at least two or more than two machines in a certain phrase. HFS systems are very common in real industries, such as in glass production, who's manufacturing technique includes ingredients, melting, forming, annealing stages and each stage also owns several machines. As HFS being a collection of assignment and sorting, it is more complicated than other flow shop scheduling problems. Gupta [1] has been proved that it is a NP-hard problem for the minimum makespan objective in a two-stage problem or only one machine in a stage. Hence, probe of optimal solution with small-scale is considerable hard, much less large/huge-scale.

Recently, meta-heuristics, such as SA, TS, GA, and PSO have been used more often to solve the HFS scheduling problem. Janiak and Kozan [2] merged simulated annealing (SA) and tabu search (TS) together to solve the HFS with a cost related objective. Genetic algorithm (GA) was applied by many authors for solving the HFS with the objective of minimizing the makespan, such as in the research of Engin et al.[3] In the past few years, PSO has been successfully applied to the scheduling problems such as flow shop and open shop scheduling [4, 5].

In this paper, we will introduce a meta-heuristics using a new version of cooperative multi-swarm PSO algorithm employed with an electoral mechanism for the HFS with minimum makespan objective. Then the proposed algorithm will be tested on the benchmark problems of Carlier and Né ron.[6]

The rest of the paper is organized as follows. Section 2 formalizes the HFS scheduling problem mathematically. The proposed version PSO with some techniques is presented in Section 3. The experimental results performed on the benchmark are provided in Section 4. Finally, conclusions and future research are given in Section 5.

2. Mathematical formulation of HFS

Hybrid flow shop (HFS) scheduling problem is a kind of scheduling consisted of a series of stages, in which there exist more than one parallel machine. But there also exist at least two or more than two machines in a certain phrase.

Note that, we obey some hypotheses: Jobs are independent and available at time zero. The handover time between consecutive stages and the machine setup time are in the processing time. The processing time of jobs at each stage is fixed. Preemption is not allowed when processing a job. The intermediate storage is unlimited between two successive stages. Based on the above notation and hypotheses, we can formalize the HFS problem as an integer program according to Néron et al.[7] as the following formulae (1) and (2):

(1) Object: Minimize C_{max}

(2) Constraints: *s.t.*

$$\begin{cases} C_{max} \geq F_{js}, \ s = 1,...,k, j = 1,...,n \\ F_{js} = S_{js} + P_{js}, \ s = 1,...,k, j = 1,...,n \\ \sum_{i=1}^{m_s} X_{jis} = 1, \ s = 1,...,k, j = 1,...,n \\ F_{js} \leq S_{j(s+1)}, \ s = 1,...,k-1 \\ S_{xs} \geq F_{ys} - LY_{xys}, \ \forall (x,y), s = 1,...,k \\ X_{jis} \in \{0,1\}, Y_{jis} \in \{0,1\}, \ j = 1,...,n, i = 1,...,m_s, s = 1,...,k \end{cases}$$

where: j – job index, i – machine index, S_{js} – starting time of job j at stage s, P_{js} – processing time of job j at stage s, F_{js} – finishing time of job j at stage s, m_s – number of machines at stage s, L – a large constant, X_{jis} – a binary variable equal to 1 when job j is assigned to machine i at stage s; 0 otherwise, Y_{jis} – a binary variable equal to 1 when job j precedes job i at stage s; 0 otherwise.

3. The proposed algorithm

3.1 PSO

PSO algorithm was first introduced by Kennedy and Eberhart [8] as a simulation of this behavior, but quickly evolved into one of the most powerful optimization algorithms in the computational intelligence field. The algorithm consists of a population of particles that are flown through an n-dimensional search space.

The position of each particle represents a potential solution to the optimization problem and is used in

determining the fitness (or performance) of a particle. In each generation of iteration, particle in swarm can be updated by the values of the best solution found by it and the one found by the whole swarm by far according to the following equation set (3):

$$(3)\qquad \begin{cases} V_{id}^{new} = \omega \times V_{id} + C_1 \times Rand() \times (P_{id}^{best} - P_{id}) \\ \qquad\qquad + C_2 \times Rand() \times (P_{gid}^{best} - P_{id}) \\ P_{id}^{new} = P_{id} + V_{id}^{new} \end{cases}$$

where: V_{id}^{new} – particle's new movement distance in a step, limited to *[vmin, vmax]*, V_{id} – particle's current movement distance in a step, P_{id}^{new} – particle's new position, P_{id} – particle's current position, P_{id}^{best} – P_{id}*'s* best experience f, P_{gid}^{best} – *gid*-th particle's best experience, V_{id} – particle's current movement distance in a step, ω – inertial weight factor, C_1 – cognition learning factor, C_2 – social learning factor.

3.2 Cooperative multi-swarm PSO (CMPSO)

Another variation of PSO, Cooperative Multi-Swarm Particle Swarm Optimization (CMPSO) proposed by Van den Bergh F.[9] could be seen as a improvement to the single swarm PSO, in which the high-dimension search space can be decompose into small scale ones similar to the idea of RELAX/CLEAN algorithm. However, its difference to it is that due to the imported information exchange mechanism among particles, the more accurate estimates did not need reduplicative iterations any more. Compared to basic single swarm PSO, both robustness and precision are improved and guarantied.

In key idea of CPSO is to divide all the *n*-dimension vectors into *k* sub-swarms. So the front *n/k* swarms are *⌈n/k⌉*-dimensional, and the $k-(n/k)$ swarms behind have *⌊n/k⌋*-dimensional vectors. In each pass of iteration, the solution is updated based on *k* sub-swarms rather than the original one. When the particles in one sub-swarm complete a search along some component, their latest best position will be combined with other sub-swarms to generate a whole solution. The function *b* performs exactly this: it takes the best particle from each of the other sub-swarms, concatenates them, splicing in the current particle from the current sub-swarm *j* in the appropriate position. Particles in each sub-swarm update their latest best positions according to Formula (5), while the latest best positions of each sub-swarm are renovated by Equation (6), where S_i denotes the *i*-th sub-swarm. Note that Equation (3) is the composition function of position with the global best fitness of all sub-swarms which is also illustrated in Fig. 1.

$$(4)\qquad b(u,Z) = (S_1.P_{gid}^{best},...,S_{u-1}.P_{gid}^{best},Z,S_{u+1}.P_{gid}^{best},...,S_k.P_{gid}^{best}), \quad 1 \le u \le k$$

$$(5)\qquad b(u,S_u.P_{id}^{best}) = argmin\ fitness(b(u,S_u.P_{id}^{best}),b(u,S_u.P_{id})), \quad 1 \le u \le k$$

$$(6)\qquad b(u,S_u.P_{gid}^{best}) = argmin\ fitness(b(u,S_u.P_{id})), \quad 1 \le id \le s, 1 \le u \le k$$

3.3 Cooperative multi-swarm PSO using electoral mechanism (CMPSO-EM)

In this paper, we present a new cooperative swarm optimization algorithm named CMPSO-EM. Firstly, we will discuss the dynamics of particles in the swarm, which is different with plain PSO and conventional cooperative PSO algorithms. The movement equation can be formalized as following equation set (7):

$$(7)\qquad \begin{cases} V_{id}^{new} = \omega \times V_{id} + C_1 \times Rand() \times (P_{id}^{best} - P_{id}) + \\ C_2 \times Rand() \times (P_{gid}^{best} - P_{id}) + C_3 \times Rand() \times (\hat{P}_{egid}^{best} \uparrow_{id} - P_{id}) \\ P_{id}^{new} = P_{id} + V_{id}^{new} \end{cases}$$

The principle of electoral cooperative mechanism is depicted in Fig. 1, in which it can clearly seen that three parts: the local best position (particles with orange color), the global best position in sub-swarm (particles with blue color), and that of electoral swarm (particles with purple color) both take participate in the evaluation of fitness function with its own position. Note that the members of electoral swarm are voted from the primitive sub-swarms with dynamic population during generation of iteration.

To import this electoral mechanism into PSO, we introduce two components of it. One is $\hat{P}_{egid}^{best}$, which denote the *egid*-th particle's best experience, i.e., the best experience of electoral swarm. However, as the position is the one of each dimension, this component could not be used directly. So another operation $\uparrow_{id}$ is also employed to calculate the projection of $\hat{P}_{egid}^{best}$, i.e., $\hat{P}_{egid}^{best} \uparrow_{id}$.

The function *b* shown in Equation (3) performs exactly this: it takes the best particle from each of the other sub-swarms, concatenates them, splicing in the current particle from the current sub-swarm *j* in the appropriate position. According to this function, the composition of P_{id}^{best}, P_{gid}^{best} and $\hat{P}_{egid}^{best}$ can be calculated based on Equation (8).

$$(8)\qquad b(u,S_u.\hat{P}_{gid}^{best}) = argmin\ fitness(b(u,S_u.P_{id}^{best}),b(u,ES.P_{gid}^{best})), \quad 1 \le id \le s, 1 \le u \le k$$

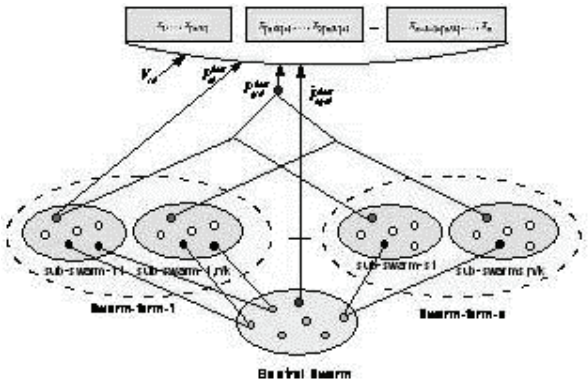


Fig.1. Principle of electoral cooperative mechanism

3.4 Solution representation

For the HFS problem, the nature number coding method is used rather than that of real number. Concretely, a solution is simply represented by a string of numbers consisting of a permutation of *n* jobs denoted by (1, 2, ..., *n*). To decode the solution, the jobs are arranged into machine by priority rules to the first available machine. Consider a simple example of a HFS problem with 5 jobs and 2 stages. The solution (2, 3, 1, 4, 5) schedules the two machines in stage 1, where jobs are arranged into machine by the sequence to the first available machine; While the schedule in stage 2 is arranged when jobs are completed by the preceding stage.

3.5 Fitness calculation

Given a sequence $S = (a_1, a_2, \ldots, a_n)$, whose priority is handled by job. That is, to assign each job on the processors which have earliest release time in each stage.

Performing the jobs' assignment on stage *s* to illustrate the no-idle machine distributed policy:

Step 1: (Assignment). Assign the M_s jobs ahead to M_s processors in stage s; Then calculate the finishing time of assigned jobs.

Step 2: (Release). Compare the release time of the *Ms* processors, and let the M_{s+1}-th job assigned at the processor with minimum release time. Finally, compute the finishing time of assigned jobs and update the release time of processor.

Step 3: (Iteration). Repeat the step 2 and assign the rest jobs.

For the minimum makespan, in a given priority of job processing, the no-idle job assignment policy is an optimal distributed pattern. So the step after sequence coding is to perform the assignment with the priority represented by the code onto the machines in each stage. The fitness value is the makespan after the assignment.

3.6 Disturbance approach

In literature [10], in order to prevent trapping into local optimization, a disturbance factor mechanism is imported to provide that if the global optimum fitness found by a sub-swarm has not updated for n iterations, then velocities of all particles should be reset, where t_n denotes the generation number of iteration with latest global best fitness; the natural number n is the disturbance factor.

In our algorithm, how to design the cooperation amongst electoral swarm and sub-swarms is the crucial problem. Here we import another policy to renew the electoral swarm dynamically based on the degree of individual fitness.

The disturbance approach can be divided into the following steps:

The disturbance approach can be divided into the following steps:

Step 1: (Judgment). Taking the judgment of $t - t_n \le n$.

Step 2: (Adjust). When the judgment is true, adjust the sub-swarms' votes to electoral swarm according to an exponential penalty factor;

Step 3: (Reset). Otherwise, reset the all sub-swarms;

Step 4: (Wiederwahl). Wiederwahl the electoral swarm again.

3.7 Main procedure

In this subsection, we will give the main phrases in the algorithm as follows:

Step 1: (Initialization). The phrase of initialization includes the following steps:

(Division). Divide the population into *s* swarm-farms, range from swarm-farm-1 to swarm-farm-s;

(Parameterization). Set the related parameters;

(Sub-swarms setup). Construct the corresponding sub-swarms of every swarm-farm, which take responsibility for optimizes related vector component;

(Initialization). Initialize the representative positions by NEH and others randomly, velocities of each particles in sub-swarms;

(Best positions selection). Let the current position as its local best position, and select a random particle as the global best positions in the sub-swarms.

Step 2: (Evaluation in sub-swarms). Evaluate the objective values of all individuals, and determine the best individual best with the best objective value in the sub-swarms.

Step 3: (Electoral procedure). Get the votes of primitive sub-swarms, and elect the best (first time randomly) particles in respective primitive sub-swarms into a new electoral swarm.

Step 4: (Evaluation in electoral swarm). Evaluate the objective values of all individuals in the electoral swarm, and determine the best individual best with the best objective value in the swarm.

Step 5: (Fitness calculation). Check the fitness hold-on generations of iteration. If true, then rest the velocities of all particles.

Step 6: (Updating). Based on the results of optimizations of primitive sub-swarms and electoral swarm, update the positions and velocities of all particles.

Step 7: (Termination). Update the generation of iterations, if it not reaches the limit, then repeat the Step 6, otherwise stop the procedure and output the local best fitness value.

4. Experiments on Carlier and Néron's benchmark

4.1 Design of experiments

In Carlier and Néron's benchmark [6], there exist 77 problems all with 3 characteristics, which define the problem are the number of jobs, number of stages and number of identical machines at each stage. Among them, there exist 53 easy problems and 24 hard problems. The problem sizes vary from 10 jobs×5 stages to 15 jobs×10 stages. For an instance, the notation j15c10b1 means a 15-job, 5-stage problem. The number 1 is the problem index for a specific type. j and c are abbreviations for job and stage, respectively, and the b denotes the structure of the machine layout at the stages, which is explained Table 1:

Table 1. Structure of the machine layout

Layout	Machine	Stage	Bottleneck
a	1	Mid.	Y
	3	Others	
b	1	1st.	Y
	3	Others	
c	2	Mid.	Y
	3	Others	
d	3	All	N

For example, layout "*a*" denotes that there is one machine at the middle stage with bottleneck and three machines at the other stages; while "*d*" expresses there are three machines at each stage without bottleneck. According to the different layouts, the total 77 problems can be divided into 13 classes. Moreover, The problems with a and b machine layouts are easier to solve, while the problems with *c* and *d* layouts are relatively harder so they are mostly grouped as hard problems.

4.2 Experiment results

In our experiments, we first propose to compare our solutions of lower bounds (LBs) [11] and those of other algorithms, such as B&B [12]. Then the CPU time, deviation and convergence are also in our consideration.

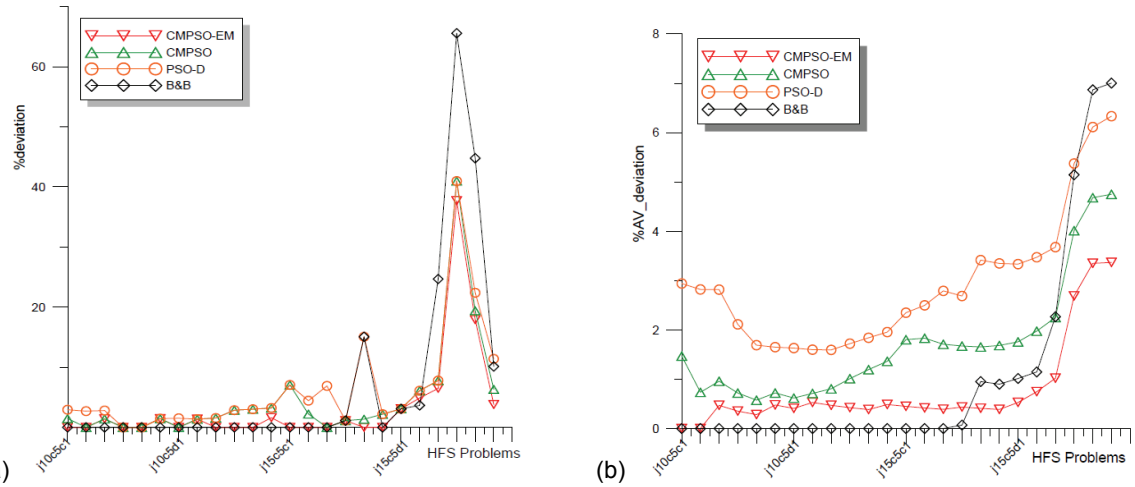
Firstly, the comparison is made based on the solution quality, measured by the percentage deviation (9) and percentage average deviation (10) between the solution and the LBs.

$$\%\ deviation = \frac{C_{max}^{bset} - \mathrm{LB}}{\mathrm{LB}} \tag{9}$$

$$\%\ AV\_deviation = \sum_{1}^{i} (\%\ deviation_i) \Big/ i \tag{10}$$

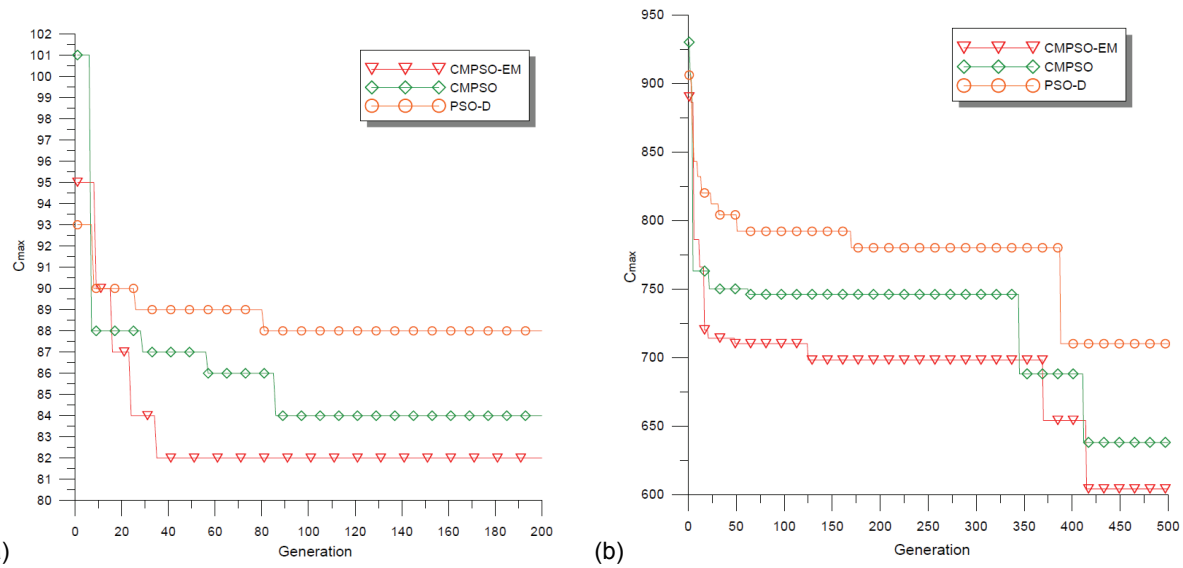
Table.2. Comparison results on the hard problems in benchmark

Problem	CMPSO-EM		CMPSO	PSO-D	B&B		LB	CMPSO-EM	CMPSO	PSO-D	B&B
	C_{max}	CPU	C_{max}	C_{max}	C_{max}	CPU		% Deviation			
j10c5c1	68	9.96	69	70	68	28	68	0	1.47	2.94	0
j10c5c2	74	22.6	74	76	74	19	74	0	0	2.70	0
j10c5c3	72	219.9	72	73	71	240	71	1.41	1.41	2.81	0
j10c5c4	66	15.5	66	66	66	1017	66	0	0	0	0
j10c5c5	78	8.2	78	78	78	42	78	0	0	0	0
j10c5c6	70	24.6	70	70	69	4865(b)	69	1.45	1.45	1.45	0
j10c5d1	66	4.71	66	67	66	6490(b)	66	0	0	1.52	0
j10c5d2	74	51.46	74	74	73	2617(b)	73	1.37	1.37	1.37	0
j10c5d3	64	36.42	65	65	64	481	64	0	1.56	1.56	0
j10c5d4	70	23.82	72	72	70	393	70	0	2.86	2.86	0
j10c5d5	66	17.14	68	68	66	1627(b)	66	0	3.03	3.03	0
j10c5d6	63	12.97	64	64	62	6821(b)	62	1.61	3.22	3.23	0
j15c5c1	85	154	91	91	85	2131(b)	85	0	7.06	7.06	0
j15c5c2	90	799	92	94	90	184	90	0	2.22	4.44	0
j15c5c3	87	169.8	87	93	87	202	87	0	0	6.90	0
j15c5c4	90	84.1	90	90	90	c	89	1.12	1.12	1.12	1.12
j15c5c5	73	610	74	84	84	c	73	0	1.36	15.07	15.07
j15c5c6	91	17.4	93	93	91	57	91	0	2.19	2.20	0
j15c5d1	167	1	167	167	167	24	162	3.09	3.09	3.09	3.09
j15c5d2	86	a	87	87	85	c	82	4.88	6.10	6.10	3.66
j15c5d3	82	a	83	83	96	c	77	6.49	7.79	7.79	24.68
j15c5d4	84	a	86	86	101	c	61	37.70	40.98	40.98	65.57
j15c5d5	79	a	80	82	97	c	67	17.91	19.40	22.39	44.78
j15c5d6	82	a	84	88	87	c	79	3.80	6.33	11.39	10.13



(a) (b)

Fig.2. (a) Percentage deviation of hard problems; (b) Percentage average deviation of hard problems;



(a) (b)

Fig.3. (a) Evolution curves of *j15c5d6*; (b) Evolution curves of *j30c5e10*

The computational results are summarized in Table 2, in which the "% deviation" columns show the performance comparison among different algorithms for the 4 group hard problems. As noticed from the table, CMPSO-EM method obtains better solutions compared to other versions of PSO, and also to the B&B when it reaches 15 jobs. On the other hand, the CPU time of CMPSO-EM is obviously shorter than that of B&B. With regard to the computational environment, CMPSO-EM, CMPSO and PSO-D was coded in Matlab 2010a and run on Intel Pentium Dual-Core 3.2 GHz 2GB RAM, and B&B [7] came from its original papers

In Fig. 2, we get the percentage deviation (a) and average deviation (b) of the 4 group hard problems. Although B&B performs well in the small-scale problems, it increases faster than the PSO algorithms in the large-scale ones, where that of CMPSO-EM is smallest.

Also, we investigate the generations of convergence of three PSO algorithms. As shown in Fig. 3, we can draw a conclusion that CMPSO-EM can converge faster into better solutions than CMPSO and PSO-D. On the other hand, due to the approach of escaping from local optima, CMPSO-EM could always reach the best solution than CMPSO and PSO-D.

Conclusions

HFS, a collection of assignment and sorting, it is more complicated and overloaded than other flow shop scheduling problems. To solve this problem, we develop a variant of PSO importing an electoral mechanism to accelerate the converging speed and a disturbance approach to help escape from local optima. Based on the experiments with the benchmark problems provided by Carlier and Néron, the results show that the proposed algorithm outperforms all the compared ones in the HFS problem.

Future research may include a further investigation of the algorithm to solve other scheduling problems. It is also worthwhile to tuning the electoral mechanism and finding new voting and disturbance approaches. The parameter setting optimally in a dynamic environment is also one of our focuses.

Acknowledgments

This work was supported by the Open Project of Key laboratory of Networking and Switching Technology under Grant No. SKLNST-2011-1-01, and the Talent Introduction Special Fund of Anhui Science and Technology University under Grant No. ZRC2011304. This is gratefully acknowledged.

REFERENCES

[1] Gupta J.N.D., Two-stage hybrid flow shop scheduling problem, *Journal of the Operational Research Society*, 39 (1988), 359-364

[2] Janiak A., Kozan E., Lichtenstein M., Oğuz C., Metaheuristic approaches to hybrid flow shop scheduling problem with a cost-related criterion, *International Journal of Production Economics*, 105 (2007), 407-424

[3] Engin O., G. Ceran, M.K. Yilmaz, An efficient genetic algorithm for hybrid flow shop scheduling with multiprocessor task problems, *Applied Soft Computing*, (2011), doi:10.1016/j.asoc.-2010.12.006

[4] Liao C.J., Tseng C.T., P. Luarn, A discrete version of particle swarm optimization for flowshop scheduling problems, *Computers and Operations Research*, 34 (2007), 3099-3111

[5] Sha D.Y., Hsu C.Y., A new particle swarm optimization for the open shop scheduling problem, *Computers and Operations Research*, 35 (2008), 3243-3261

[6] Carlier J., Néron E., An exact method for solving the multiprocessor flowshop, *R.A.I.R.O: Operations Research*, 34 (2000), 1-25

[7] Néron E., Baptise P., Gupta J.N.D., Solving hybrid flow shop problem using energetic reasoning and global operations, *Omega*, 29 (2001), 501-511

[8] Kennedy J., Eberhart R.C., Particle swarm optimization, in: *Proceedings of IEEE International Conference on Neural Network*, (1995), 1942–1948

[9] Van den Bergh F., Engelbrecht A.P., Effects of swarm size on cooperative particle swarm optimizers. *Proceedings of the GECCO*. (2001), 892-899

[10] Yu B., Jiao B., Gu X., An Improved Cooperative Particle Swarm Optimization and Its Application to Flow Shop Scheduling Problem, *Journal of East China University of Science and Technology (Natural Science Edition)*, 35 (2009), 468-474

[11] Kahraman C., Engin O., Kaya I., Yilmaz M.K.. An application of effective genetic algorithms for Solving Hybrid Flow Shop Scheduling Problems, *International Journal of Computational Intelligence Systems*, 35 (2008), 134-147

[12] .Néron E., Baptise P., Gupta J.N.D., Solving hybrid flow shop problem using energetic reasoning and global operations, *Omega,* 29 (2001), 501-511

***Authors**: Desheng LI ,Anhui Science and Technology University, School of Science, ul. Donghua Road 9, 233100, Fengyang, Anhui, China, E-mail: ldsyy2006@126.com; Qian HE Beijing University of Posts & Telecommunications, Key laboratory of Networking and Switching Technology, ul. Xitucheng 10, 100876 Being, China, E-mail: treeqian@gmail.com.*