ATLANTIS PRESS
Advances in Intelligent Systems Research, volume 159
International Conference on Mathematics, Modelling, Simulation and Algorithms (MMSA 2018)

Solving the Problem of Multi-objective Flexible Job Shop Based on Hybrid Genetic Algorithm and Particle Swarm Optimization

Xiabao Huang[1,2]
[1]School of Mechanical Science & Engineering, Huzhong University of Science &Technology, Wuhan, 420074, China
[2]School of Business Administration, Fujian Jiangxia University, Fuzhou, 350108, China

***Abstract*—A teaching-learning-based hybrid genetic-particle swarm optimization algorithm is proposed for multi-objective flexible job shop scheduling problem. It includes three modules: genetic algorithm (GA), bi-memory learning (BL) and particle swarm optimization (PSO). Firstly, in the BL module, a learning mechanism is introduced into GA to generate chromosomes which have a self-learning characteristic. During the process of evolution, the offspring in GA learn the characteristics of good chromosomes in the BL. Then, a discretization PSO algorithm which iterates the genetic population and particle population simultaneously is proposed. Finally, experiments are conducted to compare the rationality and validity of the proposed algorithm with others.**

Keywords—flexible job shop scheduling problem; genetic algorithm; particle swarm optimization; hybrid algorithm

I. INTRODUCTION

Flexible Job-shop Scheduling Problem (FJSP) is an extension of traditional Job-shop Scheduling Problem (JSP). Usually we need to optimize several objects simultaneously, so the multi-objective flexible Job-shop Scheduling Problem (MOFJSP) has become hot and important research topic. Currently, many researchers have tried to apply evolutionary algorithm to solve MOFJSP. Brandimarte combined the tabu search (TS) with dispatching rules to handle MOFJSP[1]. He used TS to solve the sub-problem of optimizing the job sequences and then used the dispatching rules to find the optimal solution of the assignment problem for each machine. Kacem et al. embed the local optimization algorithm within evolution algorithm[2]. Later they added fuzzy logic into their scheme and used Pareto method to tradeoff between makespan and machine workload[3]. Gao et al. [4] developed an adapted genetic algorithm (GA) hybridized with a shifting bottleneck strategy for MOFJSP. Li et al.[5] developed a Pareto-based discrete artificial bee colony algorithm for MOFJSP. Ho and Tay[6, 7] proposed a hybrid evolutionary algorithm with local search, and meanwhile introduced an elitism memory to store all non-dominated solutions that have been found. Yuan and Xu proposed a new memetic algorithms (MAs) by incorporating a local search algorithm into the adapted NSGA-II for MOFJSP [8]. Xia and Wu[9] proposed a hybrid method to solve MOFJSP. They used particle swarm optimization (PSO) to assign operations and used simulated annealing (SA) algorithm to sequence operations on each machine.

In this paper, we propose a Teaching-Learning-Based Hybrid Genetic-Particle Swarm Optimization Algorithms (TL-HGAPSO) in which we introduce the machine learning techniques into GA and combine GA and PSO algorithms to solve the FJSP.

II. MULTI-OBJECTIVE FJSP

FJSP can be described as: producing *n* jobs on *m* machines. Each job has one or more operations (O_{jh} denotes the *h-th* operation of the *j-th* job) and each operation can be processed by different machine. Operation's processing time is related to the machine. So FJSP contains two sub-problems: machine allocation and operations sorting. The objective of MOFSP is to allocate the operations to machines and sort the operations to minimize the maximum completion time(C_m), bottleneck workload (W_m) and total machine workload (W_t). Equation (1) - (4) describe MOFSP.

$$C_{\mathrm{m}} = \min(\max_{1\le j\le n}(\mathrm{C}_j)) \cdot \tag{1}$$

$$W_{\mathrm{m}} = \min(\max_{1\le i\le m}\sum_{j=1}^{n}\sum_{h=1}^{h_j} P_{ijh}x_{ijh}) \cdot \tag{2}$$

$$W_t = \min(\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{h=1}^{h_j} P_{ijh}x_{ijh}) \quad . \tag{3}$$

$$x_{ijh} = \begin{cases} 1, \textit{if operation } O_{jh} \textit{ is processed by machine } i \\ 0, \textit{otherwise} \end{cases} \quad . \tag{4}$$

Where, C_j is the completion time for the *j-th* job, P_{ijh} is the processing time of the *h-th* operation of the *j-th* job by machine *i* and { x_{ijh} } are the decision variables.

III. ALGORITHM FOR MOFJSP

Based on the successful experience by applying GA to FJSP, we introduce the machine learning algorithms into GA and set two thresholds to dynamically adjust the time for crossover and mutation. In TL-HGAPSO, we design multi-

parent crossover scheme to increase the rate of generating good offspring. At the same time, we update the particle swarm and exchange the information between two populations. The structure of TL-HGAPSO contain 3 modules: Genetic Algorithms (GA), Bi-Memory Learning(BL) and Particle Swarm Optimization (PSO). Figure 1 shows the relationship among those three modules.

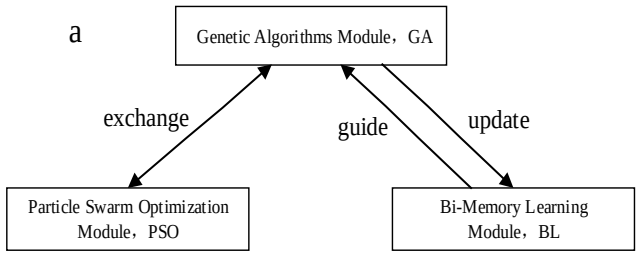


FIGURE I. STRUCTURE OF TL-HGAPSO

In TL-HGAPSO, we use GA to initialize the Bi-Memory in BL. To ensure the distribution of populations, the GA module generates the initial populations with one half generated by minimizing waiting time method and the other half by random choosing. During the process of evolution, the children in GA will continuously learn from the excellent chromosome in the BL to improve their fitness and the BL will also be updated by the elite chromosome in GA simultaneously. In TL-HGAPSO, PSO and GA will also exchange information to improve the population quality.

A. GA Module

1) Coding

To solve FJSP, we not only need to determine the sequence of operations but also to allocate the machines to operations. In TL-HGAPSO, the coding consists of by two parts: operation sorting and machine allocation. For example, see Figure 2.

The length of two chromosomes are both equal to the number of all operations T_0. The gene of operation sorting is represented by the job's number. The gene of machine allocation stands for the machine allocated to the operation. For example, in Figure 2, [1 2 1 3 2 1 3 2 3] is the operation sorting code. The first '3' means the first operation of J3, i.e., O_{31} and the second '3' means O_{32}. So the sequence is: O_{11},O_{21},O_{12},O_{31},O_{22},O_{13},O_{32},O_{23},O_{33}. [1 3 5 3 2 3 1 4 3] is the machine allocation code, which indicates that O_{11},O_{12},O_{13},O_{21},O_{22},O_{23},O_{31},O_{32} and O_{33} is processed on machine1,3,5,3,2,3,1,4,3 respectively.

2) Crossover Operation

In order to improve crossover operation performance, we introduce the third parent to make offspring have a better chance to inherit the good genes from parents. Multi-parents Precedence Preserving Order-based Crossover (MPOX) is used for operations sorting, and Multiple Point Preservation Crossover (MPX) for machine allocation.

The procedures of MPOX are as follows:

Step 1: Separate the population according to the average fitness. For the individuals with fitness will be the Parent1 and Parent2; others will be the Parent3.

Step 2: Divide the set of jobs into two non-empty subset G1 and G2 randomly.

Step 3: Put the genes both belong to Parent1 and G2 into Children1 and put the genes both belong to Parent2 and G1 into Children2; The sequence of genes remains unchanged.

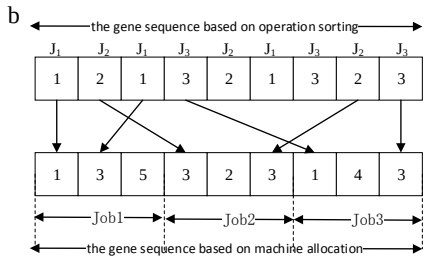


FIGURE II. CODING IN TL-HGAPSO

Step 4: Put the genes both belong to Partent3 and G1 into Children1 and put the genes both belongs to Parent3 and G2 into Children2. The sequence of genes remains unchanged. Crossover operation finish.

The procedures of MPX are as follows:

Step 1: Randomly initialize a 0,1 sequence rand0_1 which length is equal to the length of machine allocation sequence.

Step 2: Pick the '1' position in rand0_1, put the machine in Parent1 into Children2 and the machine in Parent2 into Children.

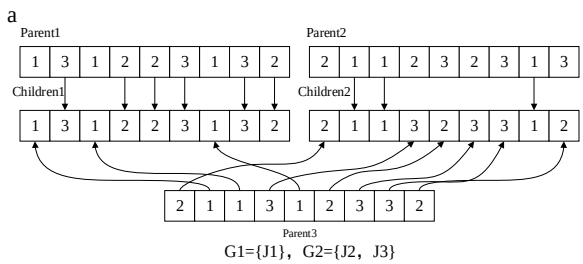
Step 3: Pick the '0' position in rand0_1, put the machine in Parent1 into Children1 and the machine in Parent2 into Children; Crossover finish.

An example of Crossover operation MPOX and MPX is shown in Figure 3(a) and Figure 3 (b).

3) Mutation operation

We modify traditional mutation operation in GA by combining crossover operation.

Firstly, a similarity threshold is used to decide whether an individual need to be mutated or not. If an individual has high similarity, it will be mutated before crossover. Otherwise, it will skip the mutation operation and go to crossover directly. We also set a convergence threshold to control the mutation rate.



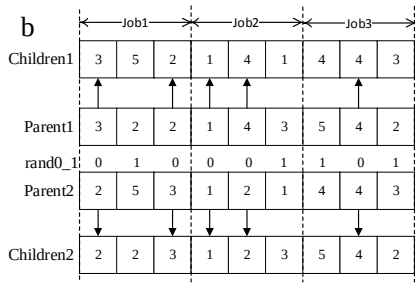


FIGURE III. (A) MPOX CROSSOVER; (B) MPX CROSSOVER.

In addition, in TL-HGAPSO, the initial population is generated by using minimum waiting time of the operation to reduce the workload as well as the completion time, fast and elitist non-dominated Pareto Policy in NSGA-II Algorithm is used for Pareto ranking[10], and tournament selection is used.

B. BL Module

In order to ensure the offspring can efficiently learn good characteristic from parents, two outer storages: chromosome storage and operation storage, are introduced in BL module. The chromosome storage is initialized by using the best *m* chromosomes in initial population. The length of operation storage equals to the number of operations, and each element is just the machine number which has the minimum processing time for corresponding operation. The k-Nearest Neighbor (K-NN) classification algorithm is used to dynamically learn the elite pattern in the chromosome in this paper. Compare the best *n* chromosomes in current population with the chromosome storage, then pick out the most similar *k* chromosomes in the chromosome storage to replace the worst *k* chromosomes in current population.

Since in TL-HGAPSO, GA module's mutation is guided by the operation storage of BL module, it will help the algorithm get the optimal solution faster. In addition, considering computation resource, for every *q* generations, the outer storages in BL module will be updated by using elite selection policy.

C. PSO Module

We re-define the position update equation of PSO for FJSP as (5).

$$X_i^{t+1}=c_2 \otimes f_4(c_1 \otimes f_3(c_1 \otimes f_2(w \otimes f_1(X_i^t), pbest_i^t), pbest_i^t), gbest_i^t) . \quad (5)$$

Where X_i^t is the particle position at t-th generation, w is the inertia weights, c_1 is self-awareness coefficient, c_2 is the society-awareness coefficient, $pbest_i^t$ is the best position for particle i at t-th generation, $gbest_i^t$ is position of the best particle in swarm at t-th generation and $w, c_1, c_2 \in [0,1]$.

$$A = w \otimes f_1(X_i^t) = \begin{cases} f_1(X_i^t) & , r < w \\ X_i^t & , else \end{cases} . \quad (6)$$

Where $r \in (0,1)$, (6) describes that by the influence of particle historical information, mutation is used to complete particle random search. In operation sorting, an operation is chosen randomly and inserted before another operation; in machine allocation, for an operation, a machine is chosen randomly from the machine set which can be used for the operation to replace the original machine.

$$B = c_1 \otimes f_2(A, pbest_i^t) = \begin{cases} f_2(A, pbest_i^t) & , r < c_1 \\ A & , else \end{cases} . \quad (7)$$

$$C = c_1 \otimes f_3(B, pbest_i^t) = \begin{cases} f_3(B, pbest_i^t) & , r < c_1 \\ B & , else \end{cases} . \quad (8)$$

Equation (7) and Equation (8) show that the particle adjusts the position guided by its best position. Where $f_2(A, pbest_i^t)$ is the POX crossover and $f_3(B, pbest_i^t)$ is the MPX crossover.

$$D = c_2 \otimes f_4(C, gbest_i^t) = \begin{cases} f_4(C, gbest_i^t) & , r < c_2 \\ C & , else \end{cases} . \quad (9)$$

Equation (9) represents the particle learn from the best particle in swarm. $f_4(C, gbest_i^t)$ is the MPX Crossover.

In TL-HGAPSO, by discretizing the way of updating the position of particle, we combine the way of information exchange in PSO with GA operations. Crossover operation is used to implement information exchange between particle and itself or between particle and the best particle in swarm. Mutation operation can be viewed as the way of particle's random search by which the local search performance will be improved. c_1 and c_2 here can be treated as the crossover rate and w can be treated as the mutation rate.

IV. EXPERIMENT ANALYSIS

TL-HGAPSO algorithms is programmed and implemented in Matlab, and it run on a pc with P4 CPU, basic frequency 3.3GHz and 8G memory. We pick 3 standard cases proposed by Kacem. Each case is run 20 times. Our benchmark methods are AL+CGA by Kacem et al.[3] and SM[11]. For fairness, we set the parameters same for all algorithm as follows: population size=200, maximum generation Gen=200, the initial rate between random case and minimum waiting time is 1:1, Crossover rate P_c=0.7, initial mutation rate P_0=0.1, the update frequency of BL module q=5, the size of chromosome storage h=5, the number of chromosomes used to update chromosome storage is 5, inertia weights w=0.1, self-awareness c_1=1.9, c_2=1.1 and random number $r_1, r_2 \in (0,1)$. Part of the experiment results are shown in Table 1.

TABLE I. MK01 TO MK10 EXPERIMENT RESULTS COMPARISON

Problem	Obj	SM	Time(Min)	TL-HGAPSO						Time(Min)
Mk01	C_M	42		40	42	42	43			
	W_M	42	4.78	40	42	41	40			1.78
	W_T	162		162	162	158	156			
Mk02	C_M	28		27	28	29	29	30		
	W_M	28	3.02	27	27	27	29	30		2.12
	W_T	155		148	151	146	143	141		
Mk03	C_M	204		204	204	212	213	214	220	
	W_M	204	26.14	199	199	199	202	210	204	4.75
	W_T	852		850	855	850	853	849	847	
Mk04	C_M	68		67	69	72	73	74	74	
	W_M	67	17.74	67	68	55	54	54	55	4.08
	W_T	352		346	345	350	349	350	348	
Mk05	C_M	177		175	177	179	181			
	W_M	177	8.26	175	177	179	181			4.34
	W_T	702		682	697	679	677			
Mk06	C_M	75		68	72	73	76	76	78	
	W_M	67	18.79	66	60	60	74	60	62	3.62
	W_T	431		421	378	375	356	362	361	
Mk07	C_M	150		144	148	151	154	157		
	W_M	150	5.68	144	148	151	154	157		5.06
	W_T	717		673	670	669	663	662		
Mk08	C_M	523		523	524	526	528			
	W_M	532	67.67	523	524	524	524			31.54
	W_T	2524		2524	2519	2521	2507			
Mk09	C_M	311		310	311	311	314			
	W_M	299	77.76	299	299	303	299			27.87
	W_T	2374		2369	2295	2291	2283			
Mk10	C_M	227		224	232	239	243	245	252	
	W_M	221	122.	220	216	217	223	205	227	
	W_T	1989	52	1978	1919	1879	1923	1896	1902	
Mk10	C_M	227		268	275	276	280			37.65
	W_M	221	122.	164	255	256	269			
	W_T	1989	52	1858	1866	1857	1862			

From the experiments, we can see that TL-HGAPSO outperforms the benchmarks based on these cases. In terms of solution quality, TL-HGAPSO yields no worse solution than benchmarks, and in some cases TL-HGAPSO's solution is much better than the benchmark. As for computation time, TL-HGAPSO use much less runtime than the benchmark methods.

V. CONCLUSION

We put forwards TL-HGAPSO algorithm for MOFJSP, which consists of 3 modules. The BL module is construct with elite selection policy. In the GA module, we modify operation sorting and machine allocation coding method, initialize population by using minimum waiting time, and put forwards MPOX crossover operation. We also combine the mutation with crossover to increase the crossover performance by introducing two thresholds to control mutation timing and mutation rate. The PSO module is based on the discrete position updating formulations. The information between the GA population and PSO population can be updated and exchanged simultaneously. Numerical experiments show that in most cases, TL-HGAPSO outperformance the benchmark methods both in terms with solution quality and computation time.Mk01 to Mk10 experiment results comparison

REFERENCES

[1] P. Brandimarte, "Routing and scheduling in a flexible job shop by tabu search," Annals of Operations Research, vol. 41, pp. 157-183, 1993.

[2] I. Kacem, S. Hammadi, and P. Borne, "Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems," IEEE Transactions on Systems Man & Cybernetics Part C, vol. 32, pp. 1-13, 2002.

[3] I. Kacem, S. Hammadi, and P. Borne, "Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic," Mathematics & Computers in Simulation, vol. 60, pp. 245-276, 2002.

[4] J. Gao, M. Gen, L. Sun, and X. Zhao, "A hybrid of genetic algorithm and bottleneck shifting for multiobjective flexible job shop scheduling problems," Computers & Industrial Engineering, vol. 53, pp. 149-162, 2007.

[5] J. Q. Li, Q. K. Pan, and K. Z. Gao, "Pareto-based discrete artificial bee colony algorithm for multi-objective flexible job shop scheduling problems," International Journal of Advanced Manufacturing Technology, vol. 55, pp. 1159-1169, 2011.

[6] N. B. Ho and J. C. Tay, "Solving Multiple-Objective Flexible Job Shop Problems by Evolution and Local Search," IEEE Transactions on Systems Man & Cybernetics Part C, vol. 38, pp. 674-685, 2008.

[7] N. B. Ho, J. C. Tay, and M. K. Lai, "An effective architecture for learning and evolving flexible job-shop schedules," European Journal of Operational Research, vol. 179, pp. 316-333, 2007.

[8] Y. Yuan and H. Xu, "Multiobjective Flexible Job Shop Scheduling Using Memetic Algorithms," IEEE Transactions on Automation Science & Engineering, vol. 12, pp. 336-353, 2014.

[9] W. Xia and Z. Wu, "An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems," Computers & Industrial Engineering, vol. 48, pp. 409-425, 3// 2005.

[10] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II," in International Conference on Parallel Problem Solving From Nature, 2000, pp. 849-858.

[11] M. Sakawa and R. Kubota, "Fuzzy programming for multiobjective job shop scheduling with fuzzy processing time and fuzzy duedate through genetic algorithms," European Journal of Operational Research, vol. 120, pp. 393-407, 2000.