Int j simul model 19 (2020) 3, 529-539
ISSN 1726-4529 Original scientific paper

# DYNAMIC SCHEDULING OF BLOCKING FLOW-SHOP BASED ON MULTI-POPULATION ACO ALGORITHM

Zhang, Y. Q.[#] & Zhang, H.
School of Information Engineering, Yulin University, Yulin 719000, China
E-Mail: zhangyaqiong@yulinu.edu.cn, zhanghui@yulinu.edu.cn ([#] Corresponding author)

**Abstract**
Drawing on the coevolution between populations, this paper proposes a dynamic multi-population ant colony optimization (ACO) algorithm to solve the blocking flow-shop scheduling problem (FSP). In our algorithm, the ant colony is divided into an elite population, multiple search populations, and a mutation population. In the initial stage, only the elite population and the search populations participate in optimization. After a certain number of iterations, a mutation population is dynamically generated from the worst solution in each search population and that in the elite population. The mutation population is reinitialized before entering the optimization process. The mutated population can jump out of the original search space for another search. Finally, the superiority of our algorithm in solving blocking FSP was proved through comparative simulations.
(Received in June 2020, accepted in August 2020. This paper was with the authors 5 weeks for 1 revision.)

**Key Words:** Flow-Shop Scheduling Problem (FSP), Dynamic Job-Shop Scheduling, Multi-Population Ant Colony Optimization (ACO) Algorithm, Discrete Event Simulation

## 1. INTRODUCTION

Inspired by natural phenomena, swarm intelligence algorithms [1, 2] provide good solutions to optimization problems like the JSP. The typical swarm intelligence algorithms include genetic algorithm (GA) [3, 4], ant colony optimization (ACO) algorithm [5, 6], simulated annealing (SA) algorithm [7, 8], and particle swarm optimization (PSO) [9-11]. These algorithms rely on the interaction between populations to obtain the optimal or suboptimal solutions of the problem.

Flow-shop scheduling problem (FSP) [12] is a classic JSP simplified from the actual production line. The FSP can be described as a task to process *n* jobs on *m* machines under the following constraints:

- each job has a fixed sequence of *m* operations,
- each job must pass through each machine in turn,
- each machine can process one job at a time,
- each job can be processed by one machine at a time,
- once begins, the processing of a job cannot be interrupted.

The blocking FSP is the most important subproblem of the FSP. To solve the blocking FSP, this paper designs a dynamic multi-population ACO algorithm, which divides the ant colony into three co-competitive populations: an elite population (with a local search algorithm based on exchange neighbourhood), several search populations, and a mutation population. The three kinds of populations search together until the algorithm converges to the optimal solution.

## 2. LITERATURE REVIEW

As a kind of metaheuristics, swarm intelligence algorithms solve problems mimicking natural phenomena or the habits of some social living being. The problem-solving process can be summarized as follows: generating one or several random scheduling solutions, optimizing the solutions iteratively by the defined rules, and converging to the optimal or suboptimal solution. Below is a brief review of the relevant literature on typical swarm intelligence algorithms.

Drawing on the principle of survival of the fittest, the GA represents each solution as a chromosome, and update the population through selection, crossover, and mutation. The population in each generation is better than that of the previous generation. In this way, the GA gradually approximates the best solution of the problem. Iyer and Saxena [13] introduced the GA to solve the FSP. Fan and Winley [14] proposed a heuristic search GA and applied it to solve the hybrid FSP. Pan and Ruiz and Giannopoulos et al. presented a hybrid GA to maximize the makespan of the FSP [15, 16]. Soltani and Karimi [17] created a novel GA algorithm to solve FSP constrained by intermediate storage. Li et al. [18] developed a multi-objective GA to solve hybrid FSP.

The ACO algorithm models and solves problems by simulating the foraging behaviour of ants [19]. With advantages like self-organization, positive feedback, parallelism, and diversity, the ACO algorithm is an effective way to solve various optimization problems. Chávez et al. [20] designed an ACO algorithm to minimize the makespan of the permutation FSP (PFSP). Khalouli et al. [21] adopted the ACO algorithm to solve hybrid batch FSP. To solve hybrid FSP, Marichelvam et al. [22] put forward a super heuristic algorithm based on ACO algorithm and GA. Qiu and Lau [23] proposed a multi-scenario ACO algorithm for dynamic JSP. Ghanavati [24] relied on the ACO algorithm to solve distributed scheduling problem.

The PSO [25] is an evolutionary algorithm inspired by the social behaviour of bird flocking. The main strategies of the PSO include evolutionary strategy and evolutionary planning. Cao et al. [26] improved the PSO for dynamic flexible JSP. Kamble et al. [27] proposed a hybrid PSO to solve the multi-objective flexible JSP. Zhang et al. [28] designed a two-stage PSO to detect stochastic faults. Lei et al. [29] invented a hybrid PSO for hybrid FSP. Alfaro-Fernández et al. [30] applied the combined PSO to solve the hybrid FSP.

The SA algorithm is an intelligent optimization algorithm that simulates the gradual annealing process of physical matter. The algorithm can effectively approximate the optimal solution through local search and avoid the local optimum trap by traversing the solution space. Dugardin et al. [31] developed a SA algorithm to maximize the makespan and total delay of hybrid FSP. Khan and Govindan [32] proposed a SA algorithm to optimize the makespan of hybrid FSP. Defersha [33] presented a SA algorithm with multiple search paths and parallel computing, aiming to solve an integrated scheduling problem. Shivasankaran et al. [34] designed a hybrid SA algorithm to solve the flexible JSP.

## 3. BLOCKING FSP

The FSP is an important problem in many fields, namely, manufacturing, assemblage, and information services. Traditionally, the FSP assumes that the intermediate buffer between two adjacent machines has infinite capacity; after being processed on the current machine, a job can be stored in the buffer and wait until the next machine is available.

In actual production, however, there is often no intermediate buffer, due to the requirements of the processing technology or the process features of the machine. This gives rise to the blocking FSP, an important branch of the FSP. In this problem, only the jobs processed on the current machine remain on that machine, before the next machine is available.

The blocking FSP can be described as follows: $n$ jobs $W = \{1, 2, \dots, n\}$ need to be processed on $m$ machines $M = \{1, 2, \dots, m\}$ in turn; there is no buffer between any two adjacent machines; if a job has been processed on the current machine, and if the machine of the next operation of the job is occupied, then the job can only wait on the current machine; the current machine is not available during the waiting, that is, cannot process subsequent jobs.

In the blocking FSP, the makespan refers to the time for the last job to leave the last machine. The makespan should be optimized by properly arranging the jobs $w = \{w(1), w(2), \dots, w(n)\}$ through the machines:

$$CT_{1,0} = 0 \tag{1}$$

$$CT_{1,k} = CT_{1,k-1} + PT_{1,k} \;\; k \in \{1, \dots, m-1\} \tag{2}$$

$$CT_{i,0} = CT_{i-1,1} \;\; i \in \{2, \dots, n\} \tag{3}$$

$$CT_{i,k} = \max\{CT_{i,k-1} + PT_{i,k}, CT_{i-1,k-1}\} \tag{4}$$

$$CT_{i,m} = CT_{i,m-1} + PT_{1,k} \;\; i \in \{1, \dots, n\} \tag{5}$$

where, $CT_{i,k}$ is the completion time of job $i$ on machine $k$; $PT_{i,k}$ is the processing time of job $i$ on machine $k$.

To solve the blocking FSP, it is necessary to search for a job sequence $w^*$ that makes $CT_{max}(w^*) \leq CT_{max}(w)$. Through iterative searches, the completion time of each job on each machine is counted, and the makespan can be obtained by $CT_{max}(w) = CT_{n,m}$.

The blocking FSP can be solved by coevolutionary algorithm for interspecific competition. In ecology, coevolution means two species evolve together through interactions, and reflects the adaptive features of interspecific interaction. In engineering, the evolution of a complex system is also an adaptive evolution process enabled by the interaction of its subsystems.

In nature, the survival and growth of a population depend on many factors, such as self-adaptability or competition from other populations. The earliest model of population growth focuses on a single population. As defined by Malthus, the population growth rate $\rho$ is a constant:

$$\mathrm{d}N/\mathrm{d}t = N\rho \tag{6}$$

Then, the population growth can be expressed as:

$$N(t) = N_0 e^{\rho(t-t_0)} \tag{7}$$

where, $N_0 = N(t_0)$ is the population size at the initial time $t_0$.

One notable feature of Malthus model is that the time $T$ for the population to double its size remains fixed:

$$2N_0 = N_0 e^{\rho t} \tag{8}$$

$$T = ln2/\rho \tag{9}$$

Malthus model is only suitable for small populations. If a population is too large, the individuals will compete for living space, causing changes to the population size.

Logistic model [35] provides another tool to define population growth. By this model, the net growth rate of a population can be defined a function of population size $\rho = \rho(N)$:

$$\mathrm{d}N/\mathrm{d}t = N\rho(N) \tag{10}$$

where, $\rho(N) = \rho - \varepsilon N$ is the competition term. Hence, the differential equation can be obtained:

$$\mathrm{d}N/\mathrm{d}t = (\rho - \varepsilon N)N \tag{11}$$

The competition term has a negative coefficient, because the growing population size stirs up competition among individuals and suppresses the growth rate.

Then, Eq. (11) can also be written as:

$$\mathrm{d}N/\mathrm{d}t = Q(Q-N)N \tag{12}$$

where, $Q$ is the maximum population size allowed in the habitat; $Q - N$ is the population size allowed in the habitat. Then, the population growth rate is proportional to the product of $N$ and $Q - N$.

In this paper, the coevolution between multiple populations is explored by the logistic model. Suppose there are populations A and B that compete for living space. Then, the growth rate of each population can be described by the logistic model as:

$$\begin{cases} dA/dt = \rho_1 A(1 - A/Q_1 - s_{21}B/Q_1) \\ dB/dt = \rho_2 B(1 - B/Q_2 - s_{12}A/Q_2) \end{cases} \tag{13}$$

where, $Q_1$ and $Q_2$ are the environmental loads of the two populations without competition, respectively; $\rho_1$ and $\rho_2$ are the maximum instantaneous growth rates of the two populations, respectively; $A$ and $B$ are the population sizes, respectively; $s_{21}$ and $s_{12}$ are competition factors for the competitive inhibition of individuals in the two populations, respectively.

From Eq. (13), the competition between *n* populations can be expressed as:

$$dN_i/dt = \rho_i N_i(1 - N_i/Q_i - \sum_{j=1,j\neq i}^{n} s_{ji}N_j/Q_i \tag{14}$$

In our model, the whole population is divided into several co-competitive sub-populations.

## 4. ALGORITHM DESIGN

Drawing on the idea of coevolution, this section improves the performance of the ACO algorithm by dividing the ant colony into three kinds of co-competitive populations: an elite population, $k$ search populations, and a mutation population. The populations evolve together through co-competition, promoting the evolution of the entire colony.

The elite population consists of the optimal solution of each population and tries to further optimize the optimal solutions. Each search population explores the solution space of the problem. During the search, the $k$ search populations regularly communicates with each other, and with the elite population, in order to pass the optimal solution to the elite population. The mutation population is made up of the worse solutions of elite and $k$ search populations, making the solutions more diverse and less duplicate. In the initial stage, only elite and $k$ search populations are present, while the mutation population is generated dynamically after the preset number of iterations is reached.

At the beginning of the improved ACO, an elite population and $k$ randomly numbered search populations are generated for optimization. The $k$ search populations evolve every $\rho$ iterations: firstly, the search populations exchange information with each other, and the optimal solution of the previous population is transferred to the next population every $\rho$ iterations; next, the optimal solution of the last population is transferred to the first population; after *n* iterations, the optimal solution is selected from the $k$ search populations and passed to the elite population. In exchange, the elite population passes the worst solution to the corresponding search population.

After that, the worst solutions of the $k$ search populations and the worst solution of the elite population are combined into a mutation population, containing $k + 1$ solutions. The mutation population is reinitialized before participating in the optimization. After each iteration, the optimal solution of the mutation population is compared with that of the elite population. If it is better, that solution will be transferred to the elite population in exchange for the worst solution of the elite population. The above process continues iteratively until the termination condition is reached. The general structure of the proposed algorithm is illustrated in Fig. 1.

In our algorithm, the elite population is responsible for further optimizing the optimal and suboptimal solutions of the ant colony. For this purpose, a local search algorithm based on exchange neighbourhood search was added to the elite population. In each iteration, the local search algorithm looks for an optimal solution and a suboptimal solution, making the solving process more efficient. The state transition formula can be expressed as:

$$s = \begin{cases} argmax\left\{\left[\tau_{ij}(t)\right]^{\sigma} \cdot \left[\pi_{ij}\right]^{\varphi}\right\} & \text{if } q < q_0 \\ p_{ij}^{k}(t) \end{cases} \tag{15}$$

$$p_{ij}^{k}(t) = \left[\tau_{ij}(t)\right]^{\sigma} \cdot \left[\pi_{ij}\right]^{\varphi} / \sum_{j\in N^k}\left[\tau_{ij}(t)\right]^{\sigma} \cdot \left[\pi_{ij}\right]^{\varphi} \tag{16}$$

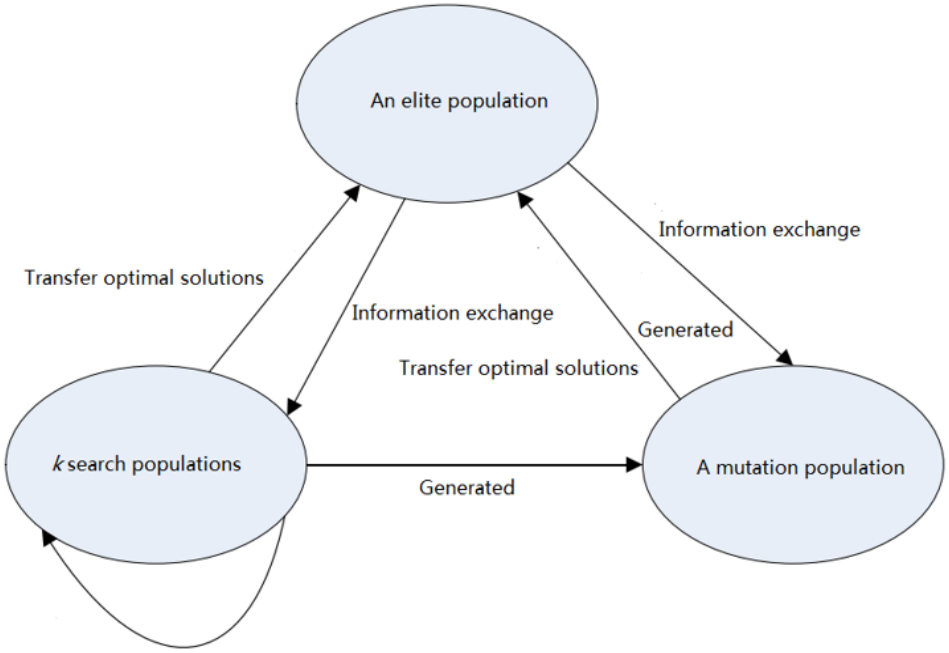


Figure 1: The general structure of our algorithm.

To increase randomness, ant $k$ chooses the next node from node $i$ by roulette wheel selection. The elite population adopts the global pheromone update rules. When all ants complete an iteration, the pheromone update is performed:

$$\tau_{ij}(t+1) = (1-\vartheta)\cdot\tau_{ij}(t) + \vartheta\cdot\Delta\tau_{ij}^{b}(t),\ 0<\vartheta<1 \tag{17}$$

$$\Delta\tau_{ij}^{b}(t) = 1/C_{max}(t) \tag{18}$$

where, $\tau_{ij}$ is the pheromone concentration between nodes $i$ and $j$; $C_{max}$ is the optimal solution found by all ants.

As shown in Fig. 2, the basic steps of the exchange neighbourhood local search algorithm are as follows:

**Step 1**. From the first to the last job, swap the positions of each two adjacent jobs are swapped in turn, and calculate the objective function value of each change.

**Step 2**. Save the optimal objective function values found in all swap operations.

**Step 3**. Update the selected solution by the optimal objective function value.

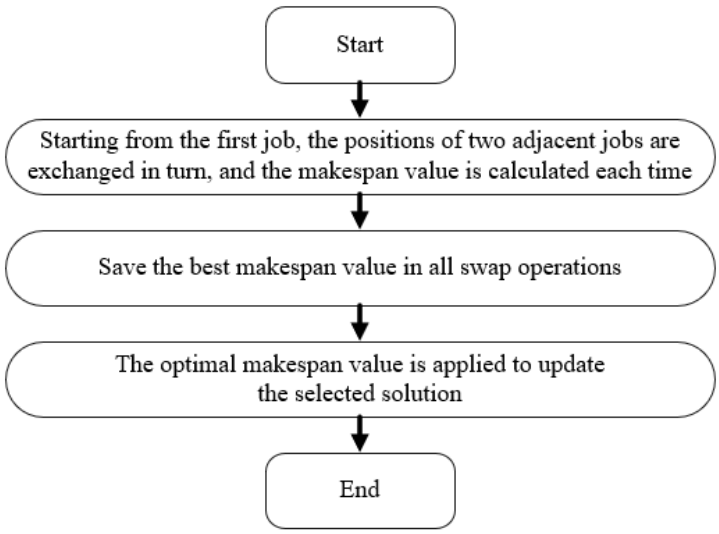


Figure 2: The workflow of the exchange neighbourhood local search algorithm.

The search populations mainly explore the solution space. Every certain number of iterations, the search populations must communicate with each other, and exchange information with elite and mutation populations, aiming to guide the convergence to high-quality solution. Each search population needs to complete three tasks: communication with other search populations, communication with elite population, and communication with mutation population.

(1) Communication with other search populations

First, all search populations are randomly numbered, and the optimal solution of the previous population is transferred to the next population every certain number of iterations. In the meantime, the last population transfers the optimal solution to the first population. In this way, each search space gets a larger search space, increasing the probability of finding the optimal solution.

(2) Communication with elite population

After a certain number of iterations, a search population will communicate with the elite population. Through the communication, the best solution of each search population is passed to the elite population, while the elite population returns the then worst solution to each search population.

(3) Communication with mutation population

The search populations work with the elite population to generate a dynamic mutation population, which consists of the worst individuals of $k$ search populations and the worst individual of elite population. That is, the size of the mutation population is $k + 1$.

The mutation population is a dynamic collection of the worst solutions of the elite and search populations, with the aim to increase the diversity of algorithm solution. Once generated, the mutation population is reinitialized to create a new search space, and to prevent the local optimum trap. After each iteration, the mutation population communicates with the elite population, and transfers its optimal solution to the elite population. Then, the elite population further optimizes the optimal solution, and passes its worst solution to the mutation population.

As shown in Fig. 3, the proposed algorithm can be implemented in the following steps:

**Step 1**. Population initialization

Initialize one elite population and $k$ search populations.

**Step 2**. Communication between search populations and elite population

The elite population and the search populations separately look for the optimal solution. Each search population exchanges information with its peers, and then with the elite population.

**Step 3**. Generation of mutation population

After a certain number of iterations, combine the worst solutions of each search population and the elite population into a mutation population.

**Step 4**. Communication between mutation population and elite population

After each iteration, the mutation population exchanges information with the elite population: the two populations compare their optimal solutions; if it is better, the optimal solution of the mutation population will be transferred to the elite population, and the elite population will return its worst solution.

**Step 5**. Termination

The three kinds of populations search together until the termination condition of the is reached.

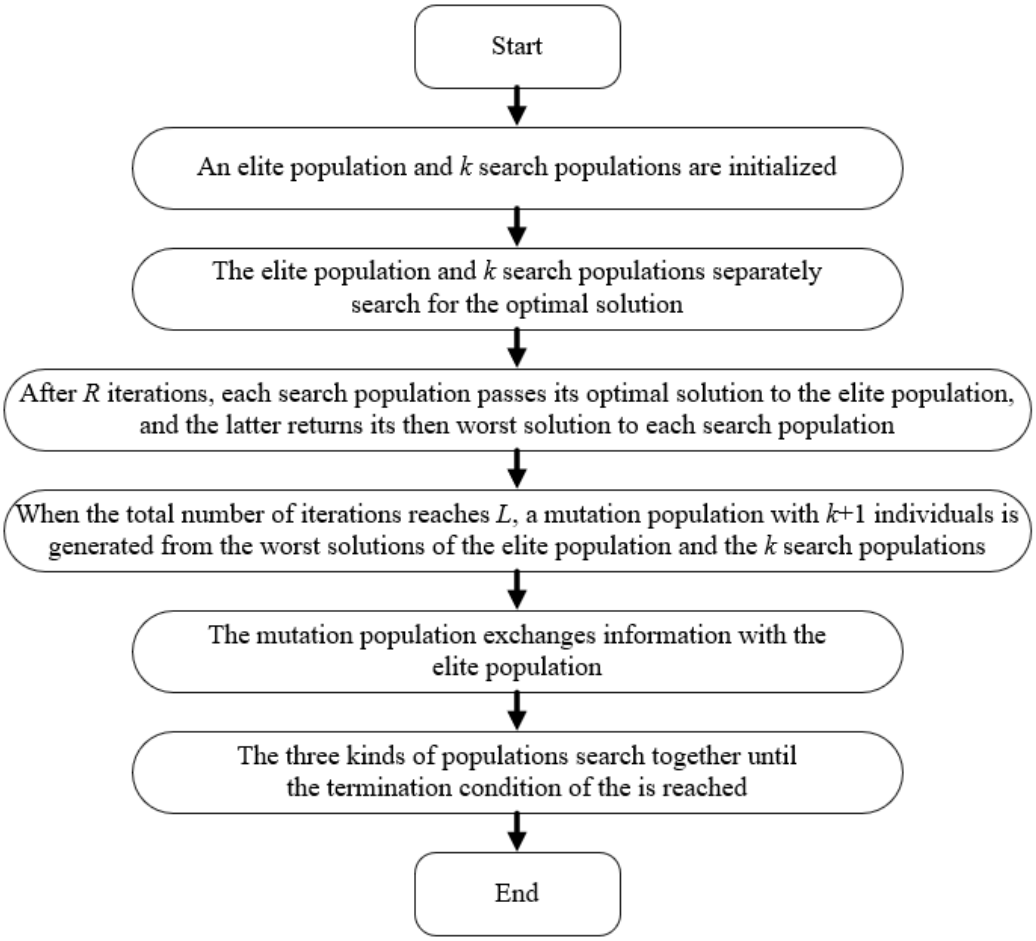


Figure 3: The workflow of our algorithm.

## 5. SIMULATION AND RESULTS ANALYSIS

To verify its performance, our algorithm, especially the module of exchange neighbourhood local search algorithm, was compared with a common dynamic multi-population ACO (DMACO) algorithm through simulation. Each algorithm ran each example of blocking FSP 30 times, and the minimum value was taken as the final result. The optimal solutions of the two algorithms are compared in Table I.

Table I: The optimal solution of the two algorithms.

| $n \times m$ | Our algorithm | Common DMACO algorithm |
|---|---|---|
| 20 × 5 | 1,310 | 1,310 |
| 20 × 10 | 1,556 | 1,556 |
| 20 × 20 | 2,236 | 2,259 |
| 50 × 5 | 3,316 | 3,350 |
| 50 × 10 | 3,750 | 3,815 |
| 50 × 20 | 4,345 | 4,420 |
| 100 × 5 | 6,125 | 6,156 |
| 100 × 10 | 7,232 | 7,262 |
| 100 × 20 | 8,250 | 8,316 |

As shown in Table I, the two algorithms achieved the same results on the $20 \times 5$ and $20 \times 10$ examples, but our algorithm obtained superior optimal solutions than the common DMACO algorithm on all the other examples.

From the optimal solutions on all the examples, the average relative percentage deviation (*ARPD*) and the relative percent deviation (*RPD*) of the two algorithms were calculated and compared in Table II.

Table II: The ARPD values of the two algorithms.

| $n \times m$ | Common DMACO algorithm | | | Our algorithm | | |
|---|---|---|---|---|---|---|
| | *ARPD* | *MinRPD* | *MaxRPD* | *ARPD* | *MinRPD* | *MaxRPD* |
| 20 × 5 | 0.93 | 0.52 | 1.51 | 0.00 | 0.00 | 0.00 |
| 20 × 10 | 0.87 | 0.28 | 1.26 | 0.00 | 0.00 | 0.00 |
| 20 × 20 | 0.68 | 0.31 | 1.12 | 0.00 | 0.00 | 0.00 |
| 50 × 5 | 0.62 | 0.29 | 0.98 | 0.00 | 0.00 | 0.00 |
| 50 × 10 | 0.58 | 0.22 | 0.79 | 0.00 | 0.00 | 0.00 |
| 50 × 20 | 0.46 | 0.17 | 0.72 | 0.00 | 0.00 | 0.00 |
| 100 × 5 | 0.39 | 0.22 | 0.69 | 0.00 | 0.00 | 0.00 |
| 100 × 10 | 0.36 | 0.17 | 0.57 | 0.00 | 0.00 | 0.00 |
| 100 × 20 | 0.37 | 0.11 | 0.61 | 0.00 | 0.00 | 0.00 |

As shown in Table II, the average *ARPD*, *MinRPD*, and *MaxRPD* of common DMACO algorithm were 0.58, 0.25 and 0.91 respectively; meanwhile, the average *ARPD*, *MinRPD*, and *MaxRPD* of our algorithm were all 0.00. It shows that our algorithm finds all the optimal values of the two algorithms.

Further, the convergence curves of the two algorithms on examples E12 and E33 are compared in Figs. 4 and 5.

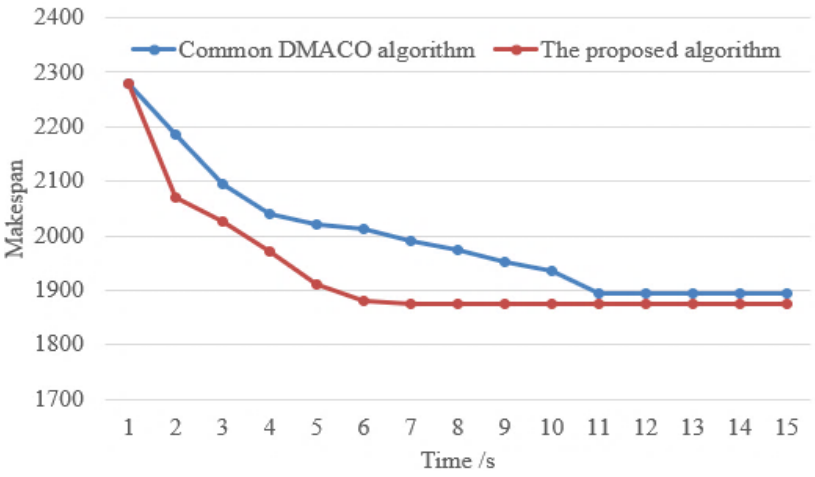


Figure 4: The convergence curves of the two algorithms on example E12.

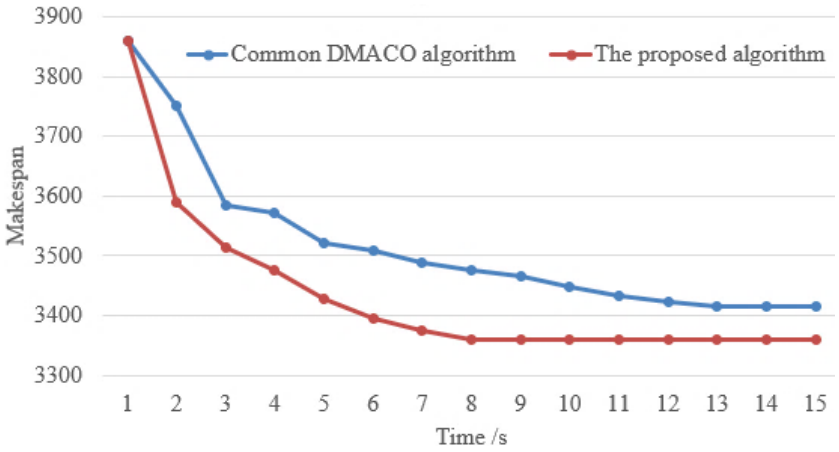


Figure 5: The convergence curves of the two algorithms on example E33.

As shown in Figs. 4 and 5, our algorithm converged to the optimal solutions of the two examples much faster than the common DMACO algorithm, thanks to the use of local search algorithm. The convergence curves of the two algorithms were similar on the other examples. From the point of convergence accuracy, our algorithm is better than common DMACO algorithm.

## 6. CONCLUSIONS

This paper probes into the blocking FSP from the angle of swarm intelligence optimization. Inspired by multi-population coevolution, a dynamic multi-population ACO algorithm was developed to solve the blocking FSP, an important branch of the FSP. The algorithm divides the ant colony into three kinds of population, which enhance solution quality and convergence speed through information exchange. The effectiveness of our algorithm in solving the blocking FSP was demonstrated through comparative simulations.

## ACKNOWLEDGEMENTS

This work was supported by Shaanxi Province Science and Technology plan projects (2020NY-170, 2020NY-163) and Yunlin Municipal Science and Technology Bureau research project (2019-77-1). Thanks for their help.

## REFERENCES

[1] Deng, W.; Chen, R.; He, B.; Liu, Y.; Yin, L.; Guo, J. (2012). A novel two-stage hybrid swarm intelligence optimization algorithm and application, *Soft Computing*, Vol. 16, No. 10, 1707-1722, doi:10.1007/s00500-012-0855-z

[2] Lu, X. M.; Wu, Q.; Zhou, Y.; Ma, Y.; Song, C. C.; Ma, C. (2019). A dynamic swarm firefly algorithm based on chaos theory and Max-Min distance algorithm, *Traitement du Signal*, Vol. 36, No. 3, 227-231, doi:10.18280/ts.360304

[3] Lim, K.; Lee, B. M.; Kang, U.; Lee, Y. H. (2018). An optimized DBN-based coronary heart disease risk prediction, *International Journal of Computers Communications & Control*, Vol. 13, No. 4, 492-502, doi:10.15837/ijccc.2018.4.3269

[4] Cao, Q. K.; Qin, M. N.; Ren, X. Y. (2018). Bi-level programming model and genetic simulated annealing algorithm for inland collection and distribution system optimization under uncertain demand, *Advances in Production Engineering & Management*, Vol. 13, No. 2, 147-157, doi:10.14743/apem2018.2.280

[5] Zhao, D. M.; Yang, T. T.; Ou, W.; Zhou, H. (2018). Autopilot design for unmanned surface vehicle based on CNN and ACO, *International Journal of Computers Communications & Control*, Vol. 13, No. 3, 429-439, doi:10.15837/ijccc.2018.3.3236

[6] Bai, L.; Du, C. L. (2019). Design and simulation of a collision-free path planning algorithm for mobile robots based on improved ant colony optimization, *Ingénierie des Systèmes d'Information*, Vol. 24, No. 3, 331-336, doi:10.18280/isi.240313

[7] Diaz Cazanas, R.; Delgado Sobrino, D. R.; Caganova, D.; Kostal, P.; Velisek, K. (2019). Joint programming of production-maintenance tasks: a simulated annealing-based method, *International Journal of Simulation Modelling*, Vol. 18, No. 4, 666-677, doi:10.2507/IJSIMM18(4)503

[8] Pang, J.; Zhou, H.; Tsai, Y.-C.; Chou, F.-D. (2018). A scatter simulated annealing algorithm for the bi-objective scheduling problem for the wet station of semiconductor manufacturing, *Computers & Industrial Engineering*, Vol. 123, 54-66, doi:10.1016/j.cie.2018.06.017

[9] Zarrouk, R.; Bennour, I. E.; Jemai, A. (2019). A two-level particle swarm optimization algorithm for the flexible job shop scheduling problem, *Swarm Intelligence*, Vol. 13, No. 2, 145-168, doi:10.1007/s11721-019-00167-w

[10] Li, Z. L.; Zhou, Y.; Bao, R. (2019). An image classification method based on optimized fuzzy bag-of-words model, *Traitement du Signal*, Vol. 36, No. 3, 239-244, doi:10.18280/ts.360306

[11] Zhang, Z.; Guan, Z. L.; Zhang, J.; Xie, X. (2019). A novel job-shop scheduling strategy based on particle swarm optimization and neural network, *International Journal of Simulation Modelling*, Vol. 18, No. 4, 699-707, doi:10.2507/IJSIMM18(4)CO18
[12] Ignall, E.; Schrage, L. (1965). Application of the branch and bound technique to some flow-shop scheduling problems, *Operations Research*, Vol. 13, No. 3, 400-412, doi:10.1287/opre.13.3.400
[13] Iyer, S. K.; Saxena, B. (2004). Improved genetic algorithm for the permutation flowshop scheduling problem, *Computers & Operations Research*, Vol. 31, No. 4, 593-606, doi:10.1016/S0305-0548(03)00016-9
[14] Fan, J. P.; Winley, G. K. (2008). A heuristic search algorithm for flow-shop scheduling, *Informatica: Journal of Computing and Informatics*, Vol. 32, No. 4, 453-464
[15] Pan, Q.-K.; Ruiz, R. (2012). An estimation of distribution algorithm for lot-streaming flow shop problems with setup times, *Omega*, Vol. 40, No. 2, 166-180, doi:10.1016/j.omega.2011.05.002
[16] Giannopoulos, N.; Moulianitis, V. C.; Nearchou, A. C. (2012). Multi-objective optimization with fuzzy measures and its application to flow-shop scheduling, *Engineering Applications of Artificial Intelligence*, Vol. 25, No. 7, 1381-1394, doi:10.1016/j.engappai.2012.06.011
[17] Soltani, S. A.; Karimi, B. (2015). Cyclic hybrid flow shop scheduling problem with limited buffers and machine eligibility constraints, *The International Journal of Advanced Manufacturing Technology*, Vol. 76, No. 9-12, 1739-1755, doi:10.1007/s00170-014-6343-0
[18] Li, J.-Q.; Sang, H.-Y.; Han, Y.-Y.; Wang, C.-G.; Gao, K.-Z. (2018). Efficient multi-objective optimization algorithm for hybrid flow shop scheduling problems with setup energy consumptions, *Journal of Cleaner Production*, Vol. 181, 584-598, doi:10.1016/j.jclepro.2018.02.004
[19] Lin, Z. S.; Chen, X. (2019). Intelligent loading of scattered cargoes based on improved ant colony optimization, *Revue d'Intelligence Artificielle*, Vol. 33, No. 2, 119-125, doi:10.18280/ria.330206
[20] Chávez, J. J. S.; Escobar, J. W.; Echeverri, M. G. (2016). A multi-objective pareto ant colony algorithm for the multi-depot vehicle routing problem with backhauls, *International Journal of Industrial Engineering Computations*, Vol. 7, No. 1, 35-48, doi:10.5267/j.ijiec.2015.8.003
[21] Khalouli, S.; Ghedjati, F.; Hamzaoui, A. (2010). A meta-heuristic approach to solve a JIT scheduling problem in hybrid flow shop, *Engineering Applications of Artificial Intelligence*, Vol. 23, No. 5, 765-771, doi:10.1016/j.engappai.2010.01.008
[22] Marichelvam, M. K.; Geetha, M.; Tosun, Ö. (2020). An improved particle swarm optimization algorithm to solve hybrid flowshop scheduling problems with the effect of human factors – a case study, *Computers & Operations Research*, Vol. 114, Paper 104812, 9 pages, doi:10.1016/j.cor.2019.104812
[23] Qiu, X.; Lau, H. Y. K. (2013). An AIS-based hybrid algorithm with PDRs for multi-objective dynamic online job shop scheduling problem, *Applied Soft Computing*, Vol. 13, No. 3, 1340-1351, doi:10.1016/j.asoc.2012.07.033
[24] Ghanavati, N. (2015). An intelligent method for static task scheduling in heterogeneous distributed systems using ant colony algorithm, *Journal of Selcuk University Natural and Applied Science*, Vol. 4, Special issue: ICASE Conference 2015, 104-115
[25] Liu, Y.; Yang, H.; Sun, G. X.; Bin, S. (2020). Collaborative filtering recommendation algorithm based on multi-relationship social network, *Ingénierie des Systèmes d'Information*, Vol. 25, No. 3, 359-364, doi:10.18280/isi.250310
[26] Cao, Z.; Zhou, L.; Hu, B.; Lin, C. (2019). An adaptive scheduling algorithm for dynamic jobs for dealing with the flexible job shop scheduling problem, *Business & Information Systems Engineering*, Vol. 61, No. 3, 299-309, doi:10.1007/s12599-019-00590-7
[27] Kamble, S. V.; Mane, S. U.; Umbarkar, A. J. (2015). Hybrid multi-objective particle swarm optimization for flexible job shop scheduling problem, *International Journal of Intelligent Systems and Applications*, Vol. 7, No. 4, 54-61, doi:10.5815/ijisa.2015.04.08
[28] Zhang, R.; Song, S.; Wu, C. (2012). A two-stage hybrid particle swarm optimization algorithm for the stochastic job shop scheduling problem, *Knowledge-Based Systems*, Vol. 27, 393-406, doi:10.1016/j.knosys.2011.11.018
[29] Lei, D.; Gao, L.; Zheng, Y. (2017). A novel teaching-learning-based optimization algorithm for energy-efficient scheduling in hybrid flow shop, *IEEE Transactions on Engineering Management*, Vol. 65, No. 2, 330-340, doi:10.1109/TEM.2017.2774281

[30] Alfaro-Fernández, P.; Ruiz, R.; Pagnozzi, F.; Stützle, T. (2020). Automatic algorithm design for hybrid flowshop scheduling problems, *European Journal of Operational Research*, Vol. 282, No. 3, 835-845, doi:10.1016/j.ejor.2019.10.004

[31] Dugardin, F.; Yalaoui, F.; Amodeo, L. (2010). New multi-objective method to solve reentrant hybrid flow shop scheduling problem, *European Journal of Operational Research*, Vol. 203, No. 1, 22-31, doi:10.1016/j.ejor.2009.06.031

[32] Khan, B. S. H.; Govindan, K. (2011). A multi-objective simulated annealing algorithm for permutation flow shop scheduling problem, *International Journal of Advanced Operations Management*, Vol. 3, No. 1, 88-100, doi:10.1504/IJAOM.2011.040661

[33] Defersha, F. M. (2015). A simulated annealing with multiple-search paths and parallel computation for a comprehensive flowshop scheduling problem, *International Transactions in Operational Research*, Vol. 22, No. 4, 669-691, doi:10.1111/itor.12105

[34] Shivasankaran, N.; Kumar, P. S.; Raja, K. V. (2015). Hybrid sorting immune simulated annealing algorithm for flexible job shop scheduling, *International Journal of Computational Intelligence Systems*, Vol. 8, No. 3, 455-466, doi:10.1080/18756891.2015.1017383

[35] Nazifa, T. H.; Mohaed, S. F.; Amin, A. B. (2019). A brief discussion on supply chain management in construction industry, *Journal of System and Management Sciences*, Vol. 9, No. 1, 69-86, doi:10.33168/JSMS.2019.0104