Int j simul model 19 (2020) 2, 313-322
ISSN 1726-4529 Original scientific paper

BLOCKING FLOW SHOP SCHEDULING BASED ON HYBRID ANT COLONY OPTIMIZATION

Shen, C. & Chen, Y. L.[#]
School of Business, Shandong Agriculture and Engineering University, Jinan 250100, China
E-Mail: bgs@sdaeu.edu.cn ([#] Corresponding author)

Abstract
This paper attempts to solve blocking flow shop scheduling problems (BFSSPs) with the aid of swarm intelligence. After briefly introducing the BFSSPs, two single population growth models were compared. Between them, the logistic model was selected to derive the co-evolution model among multiple populations. Then, a dynamic hybrid ant colony optimization (ACO) strategy was proposed based on the competition among populations. The hybrid ACO divides the ant colony into an elite population, k search populations and a mutant population. The three populations, with the help of a swap local search algorithm, evolve and interact with each other interactively until the algorithm converge to the optimal solution. The feasibility of the hybrid ACO was verified through simulations on Taillard's classic examples. This research provides a good reference for applying swarm intelligence in job-shop scheduling.
(Received in November 2019, accepted in February 2020. This paper was with the authors 2 months for 1 revision.)

Key Words: Blocking Flow Shop Scheduling Problem (BFSSP), Ant Colony Optimization (ACO), Swarm Intelligence Algorithm, Swap Local Search Algorithm

1. INTRODUCTION

Through resource allocation, the scheduling problem aims to find the optimal or near-optimal solutions to optimize one or more aspects of the tasks. Since the 1950s, many scholars have explored the job-shop scheduling problem (JSP), creating quite a few scheduling methods [1, 2]. With the booming economy, however, product processing is growing in terms of scale and complexity. The traditional methods cannot adapt to the increasingly large and complex scheduling problems.

This gives rise to new scheduling algorithms like swarm intelligence algorithms [3], which fully consider the rapid growth of enterprise production. The typical examples include genetic algorithm (GA) [4], ant colony optimization (ACO) [5], simulated annealing (SA) algorithm [6], and particle swarm optimization (PSO) algorithm [7]. Inspired by natural phenomena, these algorithms rely on the interaction between individuals in the swarm to solve problems. With simple constraints and good universality, swarm intelligence algorithms can converge to the optimal solution in a short time.

In the basic JSP [8], a production task needs to be completed with limited resources (e.g. a limited number of machines). To optimize the performance indices, the operations of the jobs contained in the task and the limited resources be allocated reasonably. Mathematically, the JSP is to model a given production task, and make the objective function value(s) optimal or sub-optimal under specific constraints.

This paper aims to develop an effective solver for blocking flow shop scheduling problems (BFSSPs), an important branch of the flow shop scheduling problem (FSSP). Firstly, the co-evolution model among multiple populations was derived from the logistic model. Then, a hybrid ant colony optimization (ACO) strategy was proposed based on the competition among populations, and the swap local search algorithm. The proposed strategy was verified through simulation on Taillard's classic examples.

2. LITERATURE REVIEW

The existing methods for JSPs fall into three categories: traditional methods, heuristic methods, and swarm intelligence algorithms.

The traditional methods solve JSPs with a mathematical model, namely, brand and bound (BAB) method [9], Lagrange relaxation method [10], and dynamic programming [11]. Tozkapan et al. [12] used the BAB algorithm to maximize the completion time of BFSSP. Aitzai et al. [13] combined BAB algorithm with PSO to solve the JSP under the blocking constraint. Prins et al. [14] applied Lagrange relaxation to solve the mixed scheduling and rescheduling of the steel-making process. Ariani et al. [15] solved the hybrid scheduling problem through Lagrange relaxation. Bautista et al. [16] solved BFSSP through dynamic programming. The traditional methods are generally suitable for small-scale scheduling problems.

The heuristic methods design the scheduling plan of jobs based on predefined rules. Common heuristic rules [17-19] include shortest processing time (SPT) rule, earliest due date (EDD) rule, Johnson's rule, Campbell-Dudek-Smith (CDS) rule, Palmer's rule, and Nawaz-Enscore-Ham (NEH). Based on Johnson's rule, Guinet [20] minimized the makespan of the two-machine flow shop scheduling problem (FSSP). Wang et al. [21] proposed a flow shop scheduling algorithm that prioritizes the jobs with the shortest total makespan on all machines.

Swarm intelligence algorithms are metaheuristic optimizers mimicking the behaviours of social animals. In generally, swarm intelligence algorithms randomly initialize one or more solutions, and then iteratively optimize the solution(s) under predefined rules, such as to quickly converge to the optimal solution. Wang and Liu [22] developed a heuristic search GA and applied it to hybrid FSSP. Zandieh and Karimi [23] created a hybrid GA to maximize the completion time of FSSP. Gajpal and Rajendran [24] proposed a new ACO to minimize the makespan of flexible JSP. Based on ACO and GA, Khalouli et al. [25] designed a hyper-heuristic algorithm for hybrid FSSP. Zhang et al. [26] improved the PSO for dynamic flexible JSP. Li et al. [27] designed a hybrid PSO algorithm to solve the BFSSP. Pempera et al. [28] put forward a SA algorithm to maximize the completion time and total delay of the mixed flow JSP. Wang et al. [29] presented a SA algorithm to minimize the makespan of the multi-process hybrid FSSP.

3. METHODOLOGY

3.1 The BFSSP

In recent years, the BFSSP has attracted much attention, owing to its significance in manufacturing and information services. As mentioned before, the BFSSP is an important branch of the FSSP. Traditionally, the FSSP assumes that the buffer between two consecutive machines has unlimited capacity. After its operation is complete on the current machine, a job can be stored in the buffer until the next machine is available. However, there is no buffer in many real-world scenarios, due to the requirements of the processing technology or machine operations. The FSSP without any buffer becomes a BFSSP, in which the job, whose operation on the current machine has completed, must remain on the current machine until the next machine is available.

The BFSSP can be described as follows: A total of n jobs $J = \{1, 2, \ldots, n\}$ need to be processed in turn on m machines $M = \{1, 2, \ldots, m\}$. There is no buffer between any two adjacent machines. Although its operation on the current machine is complete, a job can only wait on the current machine until the machine of the next operation is no longer occupied. During the waiting, the current machine is occupied and cannot process the subsequent jobs.

The objective of the BFSSP is to minimize the makespan, i.e. to minimize the time before the last job leaves the last machine by properly arranging the processing sequence of jobs on the machines.

Let $T_{i,k}$ be the time when job i leaves machine k, $P_{i,k}$, be the processing time of job i on machine k, and $\tau = (\tau(1), \tau(2), \ldots, \tau(n))$ be job sequence. Then, the makespan of the job sequence τ can be calculated by:

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

$$T_{1,k} = T_{1,k-1} + P_{1,k}, k \in \{1, 2, \ldots, m-1\} \tag{2}$$

$$T_{i,0} = T_{i-1,1}, i \in \{2, 3, \ldots, n\} \tag{3}$$

$$T_{i,k} = \max\left\{T_{i,k-1} + P_{i,k}, T_{i-1,k-1}\right\} \quad k \in \{1, 2, \ldots, m-1\} \quad i \in \{2, 3, \ldots, n\} \tag{4}$$

$$T_{i,m} = T_{i,m-1} + P_{i,m}, \quad i \in \{1, 2, \ldots, n\} \tag{5}$$

The BFSSP aims to find the job sequence τ^* making $T_{max}(\tau^*) \leq T_{max}(\tau)$. By the above formulas, the time of each job leaving each machine can be computed sequentially, and thus the makespan can be obtained by $T_{max}(\tau) = T_{n,m}$.

3.2 Co-evolutionary algorithm based on competition among populations

Co-evolution is a biological term indicating that several species evolve together through constant interactions. It reflects the self-adaptive features of the interactions between species. In engineering, a complex system evolves through the co-evolution, i.e. adaptive interactions, between its subsystems.

Co-evolutionary algorithms, as an emerging type of evolutionary algorithms, investigate how populations change and grow, and interact with each other. With the development of the evolution of system theory, many co-evolutionary algorithms have been developed and applied successfully. This subsection will focus on co-evolutionary algorithm based on competition among populations.

In nature, the survival and growth of populations depend on many factors, namely, adaptability and competition among populations. Firstly, two single population growth models are introduced below.

(1) Malthusian model

After analysing the law of population growth, Malthus held that population grows at a constant rate r.

$$\frac{1}{N}\frac{dN}{dt} = r$$

That is:

$$\frac{dN}{dt} = rN$$

The solution is:

$$N(t) = N_0 e^{r(t-t_0)} \tag{6}$$

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

Besides, it takes a fixed time T for the population to double itself:

$$2N_0 = N_0 e^{rt} \tag{7}$$

The Malthusian model only applies to small populations. If the population is too large, the individuals will compete for living space, causing changes to the population. Therefore, the constant net growth rate as assumed by Malthus is not realistic, but changes with the population size.

(2) Logistic model

Regarding the net growth rate of population as a function of population size, $r = r(N)$, we have:

$$\frac{dN}{dt} = r(N)N \tag{8}$$

Substituting the first term (competitive term) to the Malthusian model and assuming that $r(N) = r - aN$, the differential equation can be obtained:

$$\frac{dN}{dt} = (r - aN)N \tag{9}$$

Eq. (9) is the famous Logistic model. The linear coefficient in the model is negative, because the competition among individuals will occur with the gradual increase in the population size. The competition will suppress the growth rate. Eq. (9) can be rewritten as:

$$\frac{dN}{dt} = K(K - N)N \tag{10}$$

where, K is the maximum number of individuals that can be accommodated in the habitat (K is nearly constant); N is the current number of individuals; $K–N$ is the number of future individuals that can be accommodated in the habitat.

Eq. (10) shows that the population growth rate is proportional to the product of N and $K–N$. This agrees with the statistical law and many experiments. Hence, this equation is also known as the statistical calculation rate of the growth of the total number of individuals.

Here, the co-evolution among multiple populations is discussed based on the logistic model. Let P_1 and P_2 be two populations competing for the same habitat. Then, the growth rates of the two populations can be derived from the logistic model:

$$\begin{cases} \dfrac{dN_1}{dt} = r_1 N_1 \left(1 - \dfrac{N_1}{K_1} - \dfrac{a_{21} N_2}{K_1}\right) \\ \dfrac{dN_2}{dt} = r_2 N_2 \left(1 - \dfrac{N_2}{K_2} - \dfrac{a_{12} N_1}{K_2}\right) \end{cases} \tag{11}$$

where, K_1 and K_2 are the environmental loads of the two populations in the absence of competition, respectively; r_1 and r_2 are the maximum instantaneous growth rates of the two populations, respectively; N_1 and N_2 are the sizes of the two populations, respectively; a_{21} and a_{12} are competition factors about the inhibitory effects of the sizes of the two populations on competition, respectively.

From Eq. (11), the competition between n populations can be derived as:

$$\frac{dN_i}{dt} = r_i N_i \left(1 - \frac{N_i}{K_i} - \sum_{j=1, j\neq i}^{n} \frac{a_{ji} N_j}{K_i}\right) \tag{12}$$

Eq. (12) is the so-called co-evolution algorithm based on competition among populations. This algorithm breaks down a large population into several sub-populations that competing and cooperating with each other at the same time.

3.3 Hybrid ACO based on co-evolution algorithm

Drawing on the co-evolutionary algorithm of competition among populations, the competitive evolution was simulated within the same population and among different populations. The ACO was divided into multiple co-evolutionary systems. Through the cooperation and competition among the subsystems, the entire algorithm and its search ability constantly evolve.

The ant colony was split into three parts, namely, an elite population, k search populations, and a mutant population. The elite population contains the optimal solutions in each

population, which are further optimized. The k search populations traverse the solution space, communicate with each other and the elite population on a regular basis, and pass the optimal solution to the elite population. The mutant population consists of the worst solutions of the elite population and the *k* search populations, serving to increase the solution diversity and minimize the number of duplicate solutions. The mutant population does not appear until the algorithm reaches a certain number of iterations.

The three kinds of populations evolve and interact with each other interactively until the termination condition is reached.

(1) The operations of the elite population

The elite population needs to further optimize the optimal solution and sub-optimal solution of the entire colony. The merits of ant colony system (ACS) and the maximum minimum ant colony system (MMAS) were combined with the strength of the swap local search algorithm. In each iteration, an optimal solution and a sub-optimal solution are selected by swap local search, aiming to improve the solution quality.

The state transition rules can be defined as:

$$t = \begin{cases} \operatorname{argmax}\left\{\left[\tau_{ij}(t)\right]^{a} \cdot \left[n_{ij}\right]^{b}\right\} & \text{if } q \le q_0 \\ p_{ij}^{k}(t) \end{cases} \tag{13}$$

$$p_{ij}^{k}(t) = \frac{\left[\tau_{ij}(t)\right]^{\alpha} \cdot \left[n_{ij}\right]^{\beta}}{\sum_{j \in N^k} \left[\tau_{ij}(t)\right]^{\alpha} \cdot \left[n_{ij}\right]^{\beta}} \quad \left(j \notin N^k\right) \tag{14}$$

To increase the randomness, the roulette selection was introduced for ant *k* to select the next node from node *i*.

Once all ants complete an iteration, the elite population update pheromone levels by global pheromone update rules:

$$\tau_{ij}(t+1) = (1-\delta) \cdot \tau_{ij}(t) + \delta \cdot \Delta\tau_{ij}^{gb}(t), 0 \le \delta \le 1 \tag{15}$$

$$\Delta\tau_{ij}^{gb}(t) = 1 / T_{\max}(t) \tag{16}$$

where, τ_{ij} represents the pheromone level between node *i* and node *j*; T_{max} is the optimal solution found by all ants.

The basic steps of the swap local search algorithm are as follows:

Step 1. Perform swaps on the selected solution: From the first job to the last job, swap the positions of two adjacent jobs in turn, and compute the objective function values after each swap.

Step 2. Save the optimal objective function value after all swaps.

Step 3. Update the selected solution based on the optimal objective function value.

(2) The operations of the *k* search populations

The *k* search populations traverse the solution space, and exchange information at fixed intervals. To speed up the convergence to high-quality solutions, each of them also exchange information with the elite population and the mutant population. The interactions within and outside the k search populations are detailed below:

1) Interactions within the *k* search populations

Each search population is initialized with a random size. For each search population, the optimal solution of the previous population is passed to the next population every certain number of iterations. Among the *k* search populations, the last population passes the optimal solution to the first population. In this way, the search space of each search population is increased, making it more likely to find the optimal solution.

2) Interactions with the elite population

After a number of iterations, each search population interacts with the elite population. The optimal individual of the k search populations is passed to the elite population, which returns the worst individual to the search population contributing the optimal individual.

3) Interactions with the mutant population

The worst individual of the k search populations and the worst individual of the elite population are combined into a mutant population containing $k+1$ individuals.

(3) The operations of the mutant population

The mutant population mainly serves to increase the diversity of solutions. During the optimization, the worst individual of the k search populations and that of the elite population are dynamically grouped into a mutant population. Once the mutant population is generated, the ant colony is reinitialized to create a new search space, thus avoiding the local optimum trap.

After each iteration, the mutant population interacts with the elite population, and passes the found high-quality solution to the elite population. The elite population further optimizes the found high-quality solution, and returns its worst individual to the mutant population.

The mutant population is critical to the search process of the entire colony: this population enables the algorithm to jump out of the local optimum trap, and greatly enhances the diversity of solutions.

(4) The basic steps of the hybrid ACO

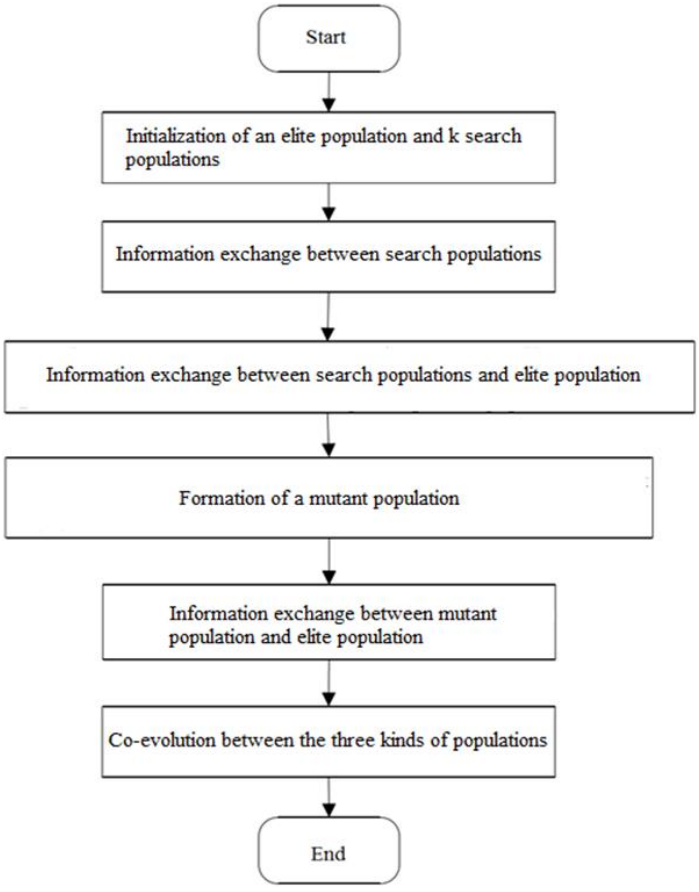

Figure 1: The workflow of the hybrid ACO.

As shown in Fig. 1, the hybrid ACO solves the BFFSP in the following steps:

Step 1. Initialize one elite population and k search populations.

Step 2. The k search populations exchange information with each other and then with the elite population. Both types of populations are optimized through the information exchange.

Step 3. After a certain number of iterations, the worst individual of the k search populations and that of the elite population are combined into a mutant population.

Step 4. In each iteration, the optimal solution of the mutant population is compared with the optimal solution of the elite population. If it is better, the optimal solution is passed to the elite population, while the elite population returns its worst individual to the mutant population.

Step 5. The three kinds of populations evolve and interact with each other interactively until the termination condition is reached.

4. SIMULATION AND RESULTS ANALYSIS

To verify its performance in solving BFSSPs, the proposed hybrid ACO was compared with several algorithms through simulations on Taillard's classic examples [30]. The examples range from 5 machines with 20 jobs to 20 machines with 100 jobs.

Table I: Optimal values of the two algorithms on Taillard's classic examples.

n×m	**ACO_nL**	**Hybrid ACO**
20×5	1,397	1,380
20×10	1,692	1,683
20×20	2,266	2,253
50×5	3,197	3,178
50×10	3,891	3,869
50×20	4,542	4,517
100×5	6,431	6,392
100×10	7,226	7,205
100×20	8,467	8,427

Table II: The ARPDs of the two algorithms on Taillard's classic examples.

n×m	**ACO_nL**			**Hybrid ACO**		
	ARPD	*MinRPD*	*MaxRPD*	*ARPD*	*MinRPD*	*MaxRPD*
20×5	0.94	0.51	1.53	0.00	0.00	0.00
20×10	0.83	0.28	1.26	0.00	0.00	0.00
20×20	0.66	0.31	1.13	0.00	0.00	0.00
50×5	0.61	0.29	0.97	0.00	0.00	0.00
50×10	0.58	0.22	0.78	0.00	0.00	0.00
50×20	0.49	0.18	0.72	0.00	0.00	0.00
100×5	0.40	0.22	0.69	0.00	0.00	0.00
100×10	0.36	0.17	0.53	0.00	0.00	0.00
100×20	0.38	0.11	0.59	0.00	0.00	0.00
Mean	0.58	0.25	0.91	0.00	0.00	0.00

First, the swap local search was removed from the hybrid ACO. After the removal, the algorithm is denoted as ACO_nL. Then, the ACO_nL and the hybrid ACO were separately applied 20 times to the mentioned examples. The minimum makespan of the 20 simulations on each example was taken as the final result. Table I compares the final results of the two algorithms on these examples. As shown in Table I, our hybrid ACO output better optimal makespans on all examples than the ACO_nL. The superiority demonstrates the importance of the swap local search.

After that, the average relative percentage deviation (*ARPD*) of the optimal values obtained by each algorithm on the examples was calculated and compared in Table II. It can be seen that the average *ARPD*, average *MinRPD*, and average *MaxRPD* of the ACO_nL were

0.58, 0.25 and 0.91, while those of our hybrid ACO were all 0.00. This means our algorithm has found the optimal values on all examples.

Furthermore, the convergence curves of the two algorithms on example tail12 were compared (Fig. 2). The comparison shows that our algorithm converged much faster to the optimal value of the example, thanks to the swap local search.

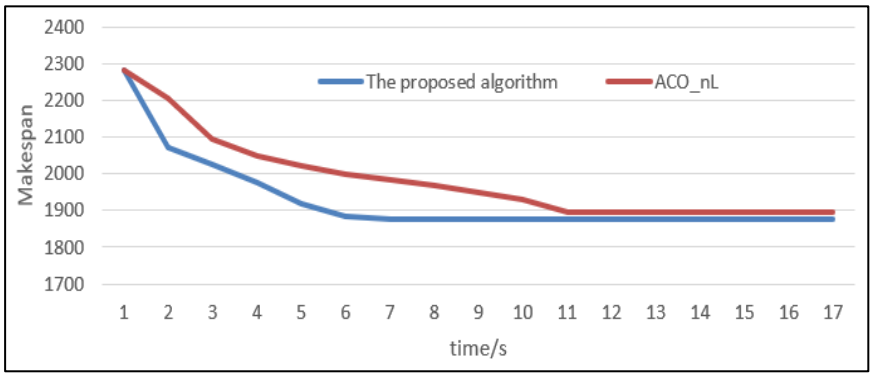

Figure 2: Convergence curves of the two algorithms on example tail12.

To sum up, our algorithm greatly outperforms the ACO_nL, highlighting the necessity of adding the swap local search to the multi-population ACO.

Finally, our algorithm was compared on the said examples with ACO with neighbourhood structure (ACONS) [31] and ACO with global pheromone evaluation (ACO_GPE) [32]. The *ARPD*s of the three algorithms are compared in Table III. The average *ARPD*, average *MinRPD*, and average *MaxRPD* of ACONS were 1.22, 0.88 and 1.73, respectively; the average *ARPD*, average *MinRPD*, and average *MaxRPD* of ACO_GPE were 1.02, 0.70 and 1.38, respectively; the average *ARPD*, average *MinRPD*, and average *MaxRPD* of our algorithm were 0.00 on all examples. The above results demonstrate that our algorithm has found the optimal values on all examples.

Table III: The ARPDs of the three algorithms on Taillard's classic examples.

n×m	**ACONS**			**ACO_GPE**			**Hybrid ACO**		
	ARPD	*MinRPD*	*MaxRPD*	*ARPD*	*MinRPD*	*MaxRPD*	*ARPD*	*MinRPD*	*MaxRPD*
20×5	2.16	2.01	3.57	1.66	1.02	2.31	0.00	0.00	0.00
20×10	2.03	1.15	2.88	1.61	0.96	2.28	0.00	0.00	0.00
20×20	1.21	0.82	2.08	1.06	0.65	1.71	0.00	0.00	0.00
50×5	1.28	0.81	1.69	1.10	0.77	1.46	0.00	0.00	0.00
50×10	1.13	0.49	1.36	1.03	0.51	1.32	0.00	0.00	0.00
50×20	1.12	0.95	1.53	0.85	0.60	1.02	0.00	0.00	0.00
100×5	0.75	0.64	0.92	0.68	0.59	0.86	0.00	0.00	0.00
100×10	0.67	0.51	0.82	0.62	0.60	0.88	0.00	0.00	0.00
100×20	0.65	0.55	0.71	0.58	0.56	0.62	0.00	0.00	0.00
Mean	1.22	0.88	1.73	1.02	0.70	1.38	0.00	0.00	0.00

The above simulation results fully testify that our algorithm can effectively solve the BFSSPs.

5. CONCLUSIONS

Based on competition among populations, this paper proposes a dynamic hybrid ACO for the BFSSP. The proposed algorithm divides the ant colony into an elite population, k search populations, and a mutant population. Initially, there are only an elite population and k

search populations. During the iterations, information is exchanged among the search populations, and between them and the elite population. To further improve the optimal solution, the elite population generates an optimal solution and a suboptimal solution in each iteration by the swap local search algorithm. Using Taillard's classic examples, the proposed algorithm was proved effective in solving the BFSSPs through repeated simulations.

REFERENCES

[1] Lopez, L.; Carter, M. W.; Gendreau, M. (1998). The hot strip mill production scheduling problem: a tabu search approach, *European Journal of Operational Research*, Vol. 106, No. 2-3, 317-335, doi:10.1016/S0377-2217(97)00277-4

[2] Zhu, J.; Shao, Z. H.; Chen, C. (2019). An improved whale optimization algorithm for job-shop scheduling based on quantum computing, *International Journal of Simulation Modelling*, Vol. 18, No. 3, 521-530, doi:10.2507/IJSIMM18(3)CO13

[3] Kim, J. B. (2019). Implementation of artificial intelligence system and traditional system: a comparative study, *Journal of System and Management Sciences*, Vol. 9, No. 3, 135-146

[4] Özdemir, H.; Sever, R.; Polat, Ö. (2019). GA-based optimization of SURF algorithm and realization based on Vivado-HLS, *Traitement du Signal*, Vol. 36, No. 5, 377-382, doi:10.18280/ts.360501

[5] Subekti, R.; Sari, E. R.; Kusumawati, R. (2018). Ant colony algorithm for clustering in portfolio optimization, *Journal of Physics: Conference Series*, Vol. 983, No. 1, Paper 012096, 6 pages, doi:10.1088/1742-6596/983/1/012096

[6] Selim, S. Z.; Alsultan, K. (1991). A simulated annealing algorithm for the clustering problem, *Pattern Recognition*, Vol. 24, No. 10, 1003-1008, doi:10.1016/0031-3203(91)90097-O

[7] Yang, L. L. (2019). An attitude motion planning algorithm for one-legged hopping robot based on spline approximation and particle swarm optimization, *Revue d'Intelligence Artificielle*, Vol. 33, No. 1, 49-52, doi:10.18280/ria.330109

[8] Shafaei, R.; Brunn, P. (1999). Workshop scheduling using practical (inaccurate) data Part 2: an investigation of the robustness of scheduling rules in a dynamic and stochastic environment, *International Journal of Production Research*, Vol. 37, No. 18, 4105-4117, doi:10.1080/002075499189682

[9] Norkin, V. I.; Pflug, G. C.; Andrzej, R. (1998). A branch and bound method for stochastic global optimization, *Mathematical Programming*, Vol. 83, No. 1-3, 425-450, doi:10.1007/BF02680569

[10] Li, S. Z.; Soh, W. Y. C.; Teoh, E. K. (1998). Relaxation labeling using augmented Lagrange-Hopfield method, *Pattern Recognition*, Vol. 31, No. 1, 73-81, doi:10.1016/S0031-3203(97)00024-1

[11] Chen, H.; Chu, C.; Proth, J.-M. (1998). An improvement of the Lagrangean relaxation approach for job shop scheduling: a dynamic programming method, *IEEE Transactions on Robotics and Automation*, Vol. 14, No. 5, 786-795, doi:10.1109/70.720354

[12] Tozkapan, A.; Kirca, O.; Chung, C.-S. (2003). A branch and bound algorithm to minimize the total weighted flowtime for the two-stage assembly scheduling problem, *Computers & Operations Research*, Vol. 30, No. 2, 309-320, doi:10.1016/S0305-0548(01)00098-3

[13] Aitzai, A.; Benmedjdoub, B.; Boudhar, M. (2016). Branch-and-bound and PSO algorithms for no-wait job shop scheduling, *Journal of Intelligent Manufacturing*, Vol. 27, No. 3, 679-688, doi:10.1007/s10845-014-0906-7

[14] Prins, C.; Prodhon, C.; Calvo, R. W. (2006). Two-phase method and Lagrangian relaxation to solve the bi-objective set covering problem, *Annals of Operations Research*, Vol. 147, No. 1, 23-41, doi:10.1007/s10479-006-0060-5

[15] Arani, T.; Karwan, M.; Lofti, V. (1988). A Lagrangian relaxation approach to solve the second phase of the exam scheduling problem, *European Journal of Operational Research*, Vol. 34, No. 3, 372-383, doi:10.1016/0377-2217(88)90158-0

[16] Bautista, J.; Cano, A.; Companys, R.; Ribas, I. (2012). Solving the $Fm|block|C_{max}$ problem using bounded dynamic programming, *Engineering Applications of Artificial Intelligence*, Vol. 25, No. 6, 1235-1245, doi:10.1016/j.engappai.2011.09.001

[17] Ye, N.; Yang, Z.; Lai, Y.-C.; Farley, T. (2005). Enhancing router QoS through job scheduling with weighted shortest processing time-adjusted, *Computers & Operations Research*, Vol. 32, No. 9, 2255-2269, doi:10.1016/j.cor.2004.03.001

[18] Bin, S.; Sun, G.; Cao, N.; Qiu, J.; Zheng, Z.; Yang, G.; Zhao, H.; Jiang, M.; Xu, L. (2019). Collaborative filtering recommendation algorithm based on multi-relationship social network, *CMC-Computers, Materials & Continua*, Vol. 60, No. 2, 659-674, doi:10.32604/cmc.2019.05858

[19] Kurniawati, D. A.; Nugroho, Y. I. (2017). Computational study of *n*-job *m*-machine flow shop scheduling problems: SPT, EDD, NEH, NEH-EDD, and modified-NEH algorithms, *Journal of Advanced Manufacturing Systems*, Vol. 16, No. 4, 375-384, doi:10.1142/S0219686717500226

[20] Guinet, A. (2000). Efficiency of reductions of job-shop to flow-shop problems, *European Journal of Operational Research*, Vol. 125, No. 3, 469-485, doi:10.1016/S0377-2217(99)00389-6

[21] Wang, K.; Choi, S. H.; Qin, H.; Huang, Y. (2013). A cluster-based scheduling model using SPT and SA for dynamic hybrid flow shop problems, *International Journal of Advanced Manufacturing Technology*, Vol. 67, No. 9-12, 2243-2258, doi:10.1007/s00170-012-4645-7

[22] Wang, S.; Liu, M. (2013). A genetic algorithm for two-stage no-wait hybrid flow shop scheduling problem, *Computers & Operations Research*, Vol. 40, No. 4, 1064-1075, doi:10.1016/j.cor.2012.10.015

[23] Zandieh, M.; Karimi, N. (2011). An adaptive multi-population genetic algorithm to solve the multi-objective group scheduling problem in hybrid flexible flowshop with sequence-dependent setup times, *Journal of Intelligent Manufacturing*, Vol. 22, No. 6, 979-989, doi:10.1007/s10845-009-0374-7

[24] Gajpal, Y.; Rajendran, C. (2006). An ant-colony optimization algorithm for minimizing the completion-time variance of jobs in flowshops, *International Journal of Production Economics*, Vol. 101, No. 2, 259-272, doi:10.1016/j.ijpe.2005.01.003

[25] Khalouli, S.; Ghedjati, F.; Hamzaoui, A. (2011). An ant colony system algorithm for the hybrid flow-shop scheduling problem, *International Journal of Applied Metaheuristic Computing*, Vol. 2, No. 1, 29-43, doi:10.4018/jamc.2011010103

[26] Zhang, J.; Wang, W.-L.; Xu, X.-L.; Wang, H.-Y. (2012). Improved particle swarm algorithm for batch splitting flexible job shop scheduling, *Control and Decision*, Vol. 2012, No. 4, 513-518

[27] Li, J.-Q.; Pan, Q.-K.; Liang, Y.-C. (2010). An effective hybrid tabu search algorithm for multi-objective flexible job-shop scheduling problems, *Computers & Industrial Engineering*, Vol. 59, No. 4, 647-662, doi:10.1016/j.cie.2010.07.014

[28] Pempera, J.; Smutnicki, C.; Zelazny, D. (2013). Optimizing bicriteria flow shop scheduling problem by simulated annealing algorithm, *Procedia Computer Science*, Vol. 18, 936-945, doi:10.1016/j.procs.2013.05.259

[29] Wang, H.-M.; Chou, F.-D.; Wu, F.-C. (2011). A simulated annealing for hybrid flow shop scheduling with multiprocessor tasks to minimize makespan, *The International Journal of Advanced Manufacturing Technology*, Vol. 53, No. 5-8, 761-776, doi:10.1007/s00170-010-2868-z

[30] Sun, G.; Bin, S. (2017). Router-level internet topology evolution model based on multi-subnet composited complex network model, *Journal of Internet Technology*, Vol. 18, No. 6, 1275-1283, doi:10.6138/JIT.2017.18.6.20140617

[31] Blum, C.; Sampels, M. (2004). An ant colony optimization algorithm for shop scheduling problems, *Journal of Mathematical Modelling and Algorithms*, Vol. 3, No. 3, 285-308, doi:10.1023/B:JMMA.0000038614.39977.6f

[32] Merkle, D.; Middendorf, M. (2003). Ant colony optimization with global pheromone evaluation for scheduling a single machine, *Applied Intelligence*, Vol. 18, No. 1, 105-111, doi:10.1023/A:1020999407672