Int J Adv Manuf Technol (2008) 39:151–157
DOI 10.1007/s00170-007-1203-9

ORIGINAL ARTICLE

Ant colony system for job shop scheduling with time windows

Rong-Hwa Huang · Chang-Lin Yang

Received: 2 October 2006 / Accepted: 14 August 2007 / Published online: 15 September 2007

Abstract Scheduling for a job shop production system is an integral aspect of production management. Scheduling operations must minimize stock, waste, and idle time and ensure on-time delivery of goods in a time window problem. In this study, due date is considered as an interval instead of a time point. This study addresses scheduling with a time window of job shop scheduling problem (JSP) and yields a solution that is close to the time window. The total penalty due to earliness and tardiness is minimized. As the problem is NP-hard, a mathematical model of the JSP with a time window is initially constructed, and data are then simulated. Solutions are obtained by ant colony optimization (ACO) programs written in C-language and are compared with the best solution obtained using LINGO 7.0 to determine the efficiency and robustness. Test results indicate that ACO is extremely efficient. Solution time using ACO is less than that using LINGO. Hence, ACO is both effective and efficient, which are two qualities stressed in business management.

Keywords Scheduling · Job shop · Time window · Ant colony optimization

1 Introduction

In production systems, scheduling problems in production systems are extremely important, both to theoretical research and in practical applications. An optimal scheduling system enhances business operating efficiency, reduces idle capacity, and delivers products and services on time to downstream or final consumers. Delivering goods on time reduces process inventory, and generates favorable relationships with customers. Two primary approaches can be identified in scheduling problem literature. The first approach identifies an exact solution via integer programming, dynamic programming, branch and bound (B&B), enumeration or other approaches [1, 2]. These approaches take too much time and become inefficient as the number of jobs increases. Furthermore, not all conditions in a real situation can be considered due to limitations in problem size, thereby reducing the applicability of these approaches. The second approach is meta-heuristic, which yields an approximate solution in a very short time; the methods involved are very diverse, including a genetic algorithm (GA), simulated annealing (SA), tabu search (TS), ant colony optimization (ACO), and neural networks [3–7]. In the literature that measures scheduling performance, the most popular scheme is to minimize the makespan [8]. If delivery time is limited, the tardiness or number of tardy deliveries is then minimized [9].

As the business environment has become increasingly competitive, customization and Just-In-Time production systems have attracted increased interest. Scheduling problems can be grouped into two classes. In the first type, due date is a particular time point; when a job is completed before this point, no cost is incurred. The second type, proposed by Chen and Lee [10], considered machine scheduling problems in which all jobs share a single time window, indicating that the due date of all jobs are in an interval. When jobs are finished in this interval, performance is acceptable to customers; other-

R.-H. Huang · C.-L. Yang (✉)
Business Administration, Fu Jen Catholic University,
No. 510 Chung-Cheng Rd.,
Hsinchuang, Taipei Hsien, Taiwan 24205,
People's Republic of China
e-mail: badm2063@mails.fju.edu.tw

wise, additional costs are incurred. When jobs are finished before the due date, then there is an inventory cost; when jobs are finished after the due date, a late cost is incurred. Baker [11] proposed the notion of a time window when studying the traveling salesman problem (TSP). Solomon and Desrosiers [12] proposed dividing time windows into hard time windows and soft time windows based on customer responses to questions. Hard time windows are those in which cars, for example, must be delivered, early arrival necessitates waiting until the prescribed time, and no delivery can be made after the time window. A soft time window is more flexible than a hard window: delivery is allowed outside the time window, but at the expense of reducing customer satisfaction and an associated penalty.

This study analyzes the job shop scheduling problem (JSP), and relaxes the delivery time limit from a time point to an interval, focusing on the plan and arrangement of job-shop production scheduling. When goods are delivered before the earliest time, they will be rejected, and a manufacturer must then bear an inventory cost. When goods are delivered later than the latest time, a manufacturer must bear a delay cost. In summary, completing shipments too early or too late is disadvantageous to upstream manufacturers and downstream customers. This study utilizes ACO to search for efficient approximate solutions to the JSP. Notably, ACO is based on experience, exploration of new solutions, and improvement of existing solutions in a potential solution space. Optimal solutions are generated by iterative self-improvement. Ant colony optimization has been applied to numerous problems including the traveling salesman problem (TSP), the quadratic assignment problem (QAP), the partitioning problem, the vehicle routing problem (VRP), the network routing problem (NRP), and scheduling problems [13–18].

This study uses the ACO algorithm to solve the JSP with time windows. Solutions are compared with optimal solutions obtained using LINGO 7.0 to determine the efficiency and robustness of the ACO algorithm. The algorithm is applied herein efficiently to solve the JSP and is highly effective and robust.

2 Developing the scheduling model

This study attempts to obtain a schedule that minimizes early and tardy costs. The programming model is given by

$$Min \sum_{j=1}^{n} \omega_1 E_j + \omega_2 T_j \tag{1}$$

subject to

$$E_j = \max\left\{L_j - C_j, 0\right\} \quad j = 1, 2, \ldots, n \tag{2}$$

$$T_j = \max\left\{C_j - U_j, 0\right\} \quad j = 1, 2, \ldots, n \tag{3}$$

$$C_j = \max_{1 \le k \le m}\left\{C_{jk}\right\} \quad j = 1, 2, \ldots n \tag{4}$$

$$\begin{gathered} C_{jk} - C_{ik} + M\left(1 - X_{ijk}\right) \ge P_{jk} \\ i = 1, 2 \ldots, n;\ j = 1, 2, \ldots, n;\ i \ne j; \\ k = 1, 2, \ldots, m \end{gathered} \tag{5}$$

$$\begin{gathered} C_{jk} - C_{jh} + M\left(1 - Z_{jhk}\right) \ge P_{jk}\ j = 1, 2, \ldots, n; \\ h = 1, 2, \ldots, m;\ k = 1, 2 \ldots, m;\ h \ne k \end{gathered} \tag{6}$$

$$X_{ijk} = \begin{cases} 1, & \text{job } i \text{ is performed before job } j \text{ on machine } k \\ 0, & \text{otherwise} \end{cases}$$

$$i = 1, 2 \ldots, n;\ j = 1, 2, \ldots, n;\ i \ne j;\ k = 1, 2, \ldots, m \tag{7}$$

$$Z_{jhk} = \begin{cases} 1, & \text{job } j \text{ proceeds on machine } h \text{ and then machine } k \\ 0, & \text{otherwise} \end{cases}$$

$$j = 1, 2, \ldots, n;\ h = 1, 2, \ldots, m;\ k = 1, 2 \ldots, m;\ h \ne k \tag{8}$$

where:

N	number of jobs
M	number of machines
P_{jk}	process time of job j on machine k
L_j	lower bound of due date for job j
U_j	upper bound of due date for job j
C_{jk}	completion time of job j on machine k
C_j	completion time of job j
E_j	earliness of job j
T_j	tardiness of job j
ω_1	unit stock cost
ω_2	tardy unit cost
M	a very large number

The objective function (Eq. 1) minimizes the sum of inventory costs due to early completion and penalty cost due to late completion. The constraint equation (Eq. 2)

Table 1 The process information of each job

Job	J_1		J_2		J_3	
Machine	Process precedence	Process time	Process precedence	Process time	Process precedence	Process time
M_1	1	2	1	2	3	3
M_2	2	3	3	4	1	2
M_3	3	4	2	3	2	4
	Lower bound	Upper bound	Lower bound	Upper bound	Lower bound	Upper bound
Time windows	13	15	9	12	7	9
Earliness cost	1		1		1	
Lateness cost	2		2		2	

represents job earliness. The restriction equation (Eq. 3) represents job tardiness. The restriction equation (Eq. 4) indicates the time when job *j* leaves the last machine *k*. The LINGO 7.0 software package requires that the nonlinear restriction Eq. 2 is first formulated as $E_j \geq L_j - C_j$ and $E_j \geq 0$, restriction Eq. 3 is formulated as $T_j \geq C_j - U_j$ and $T_j \geq 0$, and restriction Eq. 4 is formulated as $C_j \geq C_{jk}$. Restriction Eq. 5 represents the condition in which job *i* precedes job *j* on machine *k*. Restriction Eq. 6 represents the condition in which job *j* on machine *h* precedes that on machine *k*. Restriction Eq. 7 indicates that X_{ijk} is a decision variable, either 1 or 0. For a job preceding job *j* on machine *k*, X_{ijk} is 1; otherwise, X_{ijk} is 0. Restriction Eq. 8 indicates that Z_{jhk} is a decision variable, either 1 or 0; when job *j* must be processed on machine *h* before machine *k*, then Z_{jhk} is 1; otherwise, Z_{jhk} is 0.

3 ACO-based approach

In the ACO algorithm, artificial ants simulate the foraging behavior of real ants. The artificial ants deposit pheromones that guide other ants between food and their nest. Initially, no pheromone trail exists between food and the nest. The ant pheromone trail forms iteratively. At this point, the probability that ants will turn left or right is equal. Over time, ants on a short route leave more pheromone than ants on a long route. This pheromone trail acts as positive feedback attracting other ants to the short route. This natural principle has been applied to solve optimization problems. The ant system simulates the search approach used by ants and then formulates and systematizes the scheme to solve optimization problems.

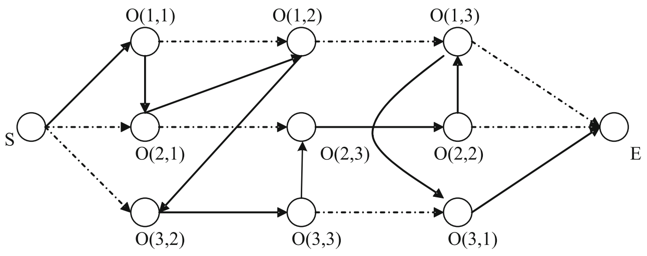

Fig. 1 The separate network figure of JSP

This study uses ACO and generates a heuristic algorithm for rapidly solving the JSP with time windows. The ACO was first proposed by Dorigo [19]. Dorigo then extended the ant system and generated ACO [13].

3.1 The ACO algorithm

Artificial ants are created using a program that simulates the natural behavior of ants. These artificial ants solve problems using three principal characteristics: (1) ants prefer routes with the most pheromone; (2) pheromone accumulates faster on short routes than on long routes; and (3) ants communicate indirectly through the pheromone. Moreover, artificial ants identify superior routes they pass (exploitation) and unsearched combinations of routes (exploration) to obtain enhanced solutions.

3.2 The ACO procedure

The search procedures of the ACO algorithm for the JSP can be divided into four steps.

Step 1 Initialize ACO parameters.

The ACO parameters are as follows: *N* is the number of ants; $\tau(r,u)$ is the pheromone trail of edge (r,u); ρ is the proportion of local pheromone evaporated; α is the proportion of global pheromone evaporated; β is the parameter determining the relative importance of heuristic information; $\eta(r,u)$ is the greedy heuristic; and q_0 is a comparative proportion ratio of controlled exploitation and exploration.

Table 2 Completion times of each job

Data on process time			
M_1	$J_1(2)$	$J_2(4)$	$J_3(10)$
M_2	$J_1(5)$	$J_3(7)$	$J_2(11)$
M_3	$J_3(4)$	$J_2(7)$	$J_1(11)$

Table 3 The parameter of time windows

Problem set	Due date range (DR)	Tardy factor (TF)
Type 1	0.60	0.20
Type 2	1.00	0.20
Type 3	0.60	0.50
Type 4	1.00	0.50

When an ant k in node r selects the next node s in the ACO algorithm, the following transition rule is followed:

$$s = \begin{cases} arg \max\limits_{u \in J_k(r)} \left\{ [\tau(r,u)]\, [\eta(r,u)]^{\beta} \right\} & \text{when } q \le q_0 \text{ (exploitation)} \\ S & \text{others (exploration)} \end{cases} \tag{9}$$

where q is a random variable between 0 and 1, q_0 is a parameter from the uniform distribution between 0 and 1, and S is a random variable derived by Eq. 10. When an ant selects the node after node r, random variable q is selected. When $q \le q_0$, then the best node is selected for moving from $J_k(r)$, thereby creating a significantly improved solution using Eq. 9; otherwise, a node is selected with a random probability distribution using Eq. 10 (explore new solutions).

$$p_k(r,s) = \begin{cases} \dfrac{[\tau(r,s)]^{\alpha}[\eta(r,s)]^{\beta}}{\sum\limits_{u = J_k(r)} [\tau(r,s)]^{\alpha}[\eta(r,s)]^{\beta}} & \text{when } s \in J_k(r) \\ 0 & \text{others} \end{cases} \tag{10}$$

Step 2 Local update of the pheromone trail.

The transition rule for choosing the next node in a search until all nodes are chosen is called a completion solution. When an ant establishes a complete solution, the pheromone on the route must be updated via a local updating rule. Local updating rules are designed to follow ants when they identify a different route and to calculate earliness and tardiness. The local updating rule is demonstrated by Eq. 11:

$$\tau(r,s) = (1-\rho)\cdot\tau(r,s) + \rho\cdot\Delta\tau(r,s) \tag{11}$$

where ρ is a local evaporating parameter with a value of 0–1. Furthermore, $\Delta\tau(r,s)=\tau_0$ and τ_0 represents the initial quantity of pheromone. When N ants generate a complete solution during an iteration, the best solution is selected via a local search and replaces the original solution. This procedure helps locate ideal solutions other ants can learn.

Step 3 Globally update the pheromone trail.

Following the local search, the best solution chosen from the routes of N ants is treated with the global updating rule to update the pheromone trail. The global updating rule is Eq. 12:

$$\tau(r,s) = (1-\alpha)\cdot\tau(r,s) + \alpha\cdot\Delta\tau(r,s) \tag{12}$$

where

$$\Delta\tau(r,s) = \begin{cases} (L_{gb})^{-1} & \textit{if } (r,s) \in \textit{ best route presently} \\ 0 & \textit{others} \end{cases}$$

In Eq. 12, α is a global parameter for pheromone evaporation and is in the range of 0–1. Moreover, L_{gb} is the best solution to date. Therefore, only the best route is determined by the amount of pheromone in an iteration. When an iteration does not reach the previously set value, then searching continues; otherwise, searching ends.

Table 4 Comparison of effectivity test of average solution values

Type	n	m	LINGO average solution	LINGO CPU time (s)	ACO average solution	ACO CPU time (s)
Type 1	3	3	7.267	3.133	8.867	1.507
	4	4	14.667	20.800	17.000	3.151
	5	5	34.700	61.933	41.133	4.385
	6	6	41.333	125.667	48.233	7.012
	7	7	69.667	495.167	75.333	15.774
	8	8	77.333	2,828.033	85.500	19.360
	9	9	92.167	24,995.900	98.233	24.521
	10	10	108.233	360,225.000	116.00	31.199
Type 2	3	3	2.533	2.333	3.200	1.599
	4	4	20.467	16.533	23.733	5.010
	5	5	43.267	55.000	46.067	5.715
	6	6	57.000	94.167	64.167	7.837
	7	7	78.133	378.367	82.333	10.556
	8	8	90.677	3,104.159	98.667	12.198
	9	9	112.067	37,543.17	120.133	16.338
	10	10	131.267	404,217.100	154.467	21.006
Type 3	3	3	10.267	2.00	11.00	1.378
	4	4	5.533	20.333	6.533	3.276
	5	5	32.333	66.733	36.967	3.630
	6	6	52.467	214.867	61.067	4.687
	7	7	65.100	709.010	69.133	6.191
	8	8	83.967	3,173.160	91.677	9.040
	9	9	89.133	45,502.330	97.767	12.529
	10	10	113.533	622,239.300	121.400	16.869
Type 4	3	3	10.833	2.00	13.267	2.714
	4	4	31.567	14.667	34.333	3.646
	5	5	37.733	63.600	40.467	5.980
	6	6	43.100	151.233	49.133	5.111
	7	7	59.900	411.200	67.333	6.207
	8	8	72.667	2,342.033	80.433	5.009
	9	9	84.833	32,127.970	89.967	11.541
	10	10	93.130	572,828.300	101.300	19.803

Step 4 Stop and generate a best sequence.

3.3 Illustrative example

The JSP is basically the same as most scheduling problems that can be conventionally described using a disjunctive graph. In the JSP problem, every job has constraints, which represent the largest bottleneck in applying ACO to the JSP. When using the algorithm, due date windows of each job must be considered; late and early delivery costs are minimized.

This study illustrates the JSP for three jobs and three machines. Table 1 presents process time, process precedence, time windows, and late and early costs of each job.

Figure 1 plots the disjunctive graph of the JSP. The disjunctive graph has 11 nodes, including nine operating nodes *O(i, j)*, where *i* is the job, *j* is the machine, and the two virtual nodes represent the start (*S*) and end (*E*). When an ant wants to develop a complete solution, it must start from node *S* and pass through nine nodes. When the ant reaches the final node, a complete solution has been acquired. As ants visit nodes, as well as selecting the next node according to the state transition rule, they must select the operation whose preceding operation has been completed; that is, jobs must be processed in order.

First, let point *P* represent the set of nodes to be visited; *Q* is the lead operation, and *O* is the set of all nodes. Initially, $P=O-\{S\}$ and $Q=\{O_{1,1}, O_{2,1}, O_{3,2}\}$, which comprise the set of first operations for all jobs. A node is selected according to the state transition rule and stored in the memory list. The node is excluded from *P* and *Q*. Therefore, a node that is already chosen cannot be chosen again. Following this procedure, one node is visited during each iteration until $P=\Phi$. Eventually, the order of operations recorded in the memory list determines the order of operations on every machine and yields a reasonable solution, enabling calculation of maximum completion time for every job and determination of whether delivery constraints will be satisfied.

Table 5 Results of robust test

Type	*n*	*m*	LINGO best	ACO best	ACO worst	ACO average	ACO CPU (s)	LINGO CPU (s)
Type 1	3	3	0	0	2	0.800	1.837	2
	4	4	5	8	12	9.570	3.167	20
	5	5	57	58	78	66.250	5.479	64
	6	6	34	42	61	54.600	6.091	209
	7	7	53	60	76	68.333	9.425	787
	8	8	77	81	102	88.133	13.917	3963
	9	9	86	94	117	106.967	17.521	18,535
	10	10	102	109	129	118.000	20.749	213,982
Type 2	3	3	0	1	4	2.833	1.930	2
	4	4	20	20	31	24.467	2.112	26
	5	5	0	0	13	6.433	5.311	68
	6	6	46	52	73	60.333	8.015	253
	7	7	71	83	107	96.100	11.126	1560
	8	8	104	112	126	117.667	16.527	8255
	9	9	98	103	125	115.967	20.741	88,196
	10	10	131	147	173	164.833	24.987	343,416
Type 3	3	3	3	3	6	4.900	1.306	2
	4	4	9	11	19	13.233	4.082	22
	5	5	11	13	22	17.967	6.734	66
	6	6	31	35	48	37.467	9.548	319
	7	7	47	54	71	59.000	13.247	2028
	8	8	63	65	80	72.333	18.846	14,308
	9	9	80	88	107	96.667	21.055	106,698
	10	10	109	118	135	125.400	36.748	233,584
Type 4	3	3	2	2	5	2.667	2.248	2
	4	4	9	9	25	16.000	4.353	12
	5	5	24	26	42	32.566	5.278	58
	6	6	27	37	48	41.833	5.774	423
	7	7	59	61	78	64.133	7.648	3,006
	8	8	65	69	84	74.167	14.984	11,607
	9	9	87	97	121	104.000	20.737	58,659
	10	10	129	136	162	142.367	31.165	293,295

Figure 1 shows the separate network of JSP as an example. When an ant completes a set of solutions, the array in the memory list is ($O_{1,1}$, $O_{2,1}$, $O_{1,2}$, $O_{3,2}$, $O_{3,3}$, $O_{2,3}$, $O_{2,2}$, $O_{1,3}$, $O_{3,1}$). This solution is only a reasonable solution. In Fig. 1, the solid line is the path traveled by the ant, and the dotted line is the path not traveled by the ant. Therefore, the process order in M_1 is (J_1, J_2, J_3); the process order in M_2 is (J_1, J_3, J_2); the process order in M_3 is (J_3, J_2, J_1). However, J_1 cannot be processed in M_1 and M_2 simultaneously, so either M_1 or M_2 must be initially idle.

According this information for process time and process order, Table 2 can be used to derive the time required to complete J_1, which is 11; the time needed to complete J_2 is 11, and the time needed to complete J_3 is 10. The delivery time window of each job can then be assessed to determine the time required to complete J_1, which does not fall in the time window but precedes the time window. Hence, an earliness cost is incurred. The approach for determining earliness cost is $(13-11)*1=2$. The time required to complete J_2 is 11, and falls within time window J_2; thus, no earliness cost is incurred. The time required to complete J_3 is 10, which does not fall in the time window, but is after the time window; thus, a tardiness cost is incurred. Tardiness cost is given by $(10-9)*2=2$. Therefore, total cost is the sum of costs for J_1, J_2 and J_3, which is $2+0+2=4$.

4 Experiment

Simulated data are obtained as evidence of the effectiveness and robustness of the proposed algorithm. The four parts of the data test are building the test data, effectiveness analysis, robustness analysis, and the simulation for a large JSP.

1. Generation of test data.

First, job processing time is randomly generated using a program written in C language, and a simulated test is performed on a PC with a Pentium 4 1.4 GHz CPU. The data were uniformly distributed in the interval [1, 10] based on the framework in Table 3 and presented by Zheng et al. [20]. The due dates are uniformly distributed in the interval $\left[(1-TF-DR/2)\sum_{i=1}^{m} p_i*(1+(\#M-1)*0.3),(1-TF+DR/2)\sum_{i=1}^{m} p_i*(1+(\#M-1)*0.3)\right]$. The experiments were conducted on four types of problems, both small and large problems, which were generated with n=3–10, m=3–10, n=20–100, and m=20–100, respectively. The order of the machines is generated using a random number. The ACO parameters were set to α=0.1, β=3, Q=25, ρ=0.1, q_0=0.95, and τ_0=5. Moreover, 100 generations are permitted for every example, with each generation involving ten ants. The parameter set only affects CPU time.

2. Analysis of the effectiveness.

In small sized problems, effective analysis is to compare the ACO solution and optimal solution in order to verify whether the ACO solution is good enough. During analysis of effectiveness, a set of 30 randomly generated data are tested using LINGO 7.0. The weights of the early and tardy costs in the objective function are both set to 1. Test results obtained using LINGO are compared with those obtained using ACO. Table 4 shows test results. Results obtained for the average solution of ACO are worse than that obtained using LINGO;

Table 6 Results of large problem test

Type	n	m	Best solution	Worst solution	Average solution	CPU time(s)
Type 1	20	20	579	731	644.100	37.902
	40	40	986	1,548	1,150.000	46.188
	60	60	1,424	1,783	1,548.967	50.254
	80	80	4,875	5,827	5,263.166	75.487
	100	100	10,567	11,781	11,025.033	84.110
Type 2	20	20	819	954	878.676	27.852
	40	40	1,087	1,345	1,266.500	30.874
	60	60	2,548	3,250	2,888.167	41.755
	80	80	5,750	6,953	6,053.333	50.878
	100	100	9,456	10,785	9,013.967	66.874
Type 3	20	20	537	743	616.733	38.013
	40	40	848	1,478	1,233.133	47.210
	60	60	2,178	2,701	2,544.667	69.454
	80	80	4,780	5,477	4,985.833	82.525
	100	100	10,174	11,647	10,984.167	89.149
Type 4	20	20	432	527	494.533	28.764
	40	40	793	1,478	910.667	50.003
	60	60	3,798	4,717	4,258.967	61.599
	80	80	6,794	7,884	7,298.800	70.564
	100	100	10,478	11,989	10,954.467	84.143

however, the error values between them are smaller than 10. When the number of jobs and machines increases, the solution time via LINGO increases very rapidly. It exceeds thousands of seconds to solve when the number of jobs and machines are more than eight. Conversely, the solving time using the proposed algorithm does not increase rapidly as the number of jobs and machines increases, that is, solution time increases only a little. The ACO is effective in solving small-scale problems.

3. Analysis of robustness.

The robust analysis is to verify the steadiness of the ACO solutions. Table 5 lists the processing time results for different numbers of machines and jobs for 30 replications. The best and worst objective values for these 30 replications are examined. For robustness, the difference between best and worst values is less than 30. Consequently, ACO is a very good tool for deriving an optimum robustness schedule.

4. Simulation of large problems.

The job and machine number influence the complexity of the JSP. Calculating an optimum solution using LINGO is extremely difficult and becomes unreasonable. Table 6 lists the ACO heuristic results for a large-scale problem and the time required to solve the problem. In robust analysis, the CPU time increases slowly and the variations between best and worst solutions are acceptable. As stated, LINGO is inefficient for large-scale problems. Thus, ACO is better at solving large JSPs with time windows than LINGO.

5 Conclusion

This study considers JSPs and relaxes the delivery time limitation to an interval. The four parts of the data test in this study are construction of test data, analysis of effectiveness, analysis of robustness, and simulation of a large problem. Ant colony optimization is applied to identify approximate solutions rapidly. Simulation is performed to provide evidence of effectiveness and robustness of the proposed algorithm. Test results indicate that ACO is both effective and efficient, which are two qualities favored in business management.

References

1. Carlier J, Pinson E (1989) An algorithm for solving the job-shop problem. Manage Sci 5:146–176
2. Webster S, Azizoglu M (2001) Dynamic programming algorithms for scheduling parallel machines with family setup times. Comput Oper Res 28:127–137
3. Kim JU, Kim YD (1996) Simulated annealing and genetic algorithms for scheduling products with multi-level product structure. Comput Oper Res 23:857–868
4. Chen H, Ihlow J, Lehmann C (1999) A genetic algorithm for flexible job-shop scheduling. Proceedings of the 1999 IEEE international conference on robotics and automation, Detroit, Michigan, pp 1120–1125
5. Lee I, Shaw MJ (2000) A neural-net approach to real time flow-shop sequencing. Comput Ind Eng 38:125–147
6. Ferland JA, Ichoua S, Lavoie A, Gagne E (2001) Scheduling using tabu search methods with intensification and diversification. Comput Oper Res 28:1075–1092
7. Laarhoven V, Aarts PJM, Lenstra JK (1992) Job shop scheduling by simulated annealing. Oper Res 40:113–125
8. Aggoune R (2003) Minimizing the makespan for the flow shop scheduling problem with availability constraints. Eur J Oper Res 153:534–543
9. Armentano VA, Ronconi DP (1999) Tabu search for total tardiness minimization in flowshop scheduling problems. Comput Oper Res 26:219–235
10. Chen ZL, Lee CY (2002) Parallel machine scheduling with a common due window. Eur J Oper Res 136:512–527
11. Baker EK (1983) An exact algorithm for the time-constrained traveling salesman problem. Oper Res 31:938–945
12. Solomon MM, Desrosiers J (1988) Time windows constrained routing and scheduling problem. Transp Sci 22:1–13
13. Dorigo M, Gambardella LM (1997) Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE T Evolut Comput 1:53–66
14. Gambardella L, Taillard E, Dorigo M (1999) Ant colonies for the quadratic assignment problem. J Oper Res Soc 50:167–176
15. McMullen PR (2001) An ant colony optimization approach to addressing a JIT sequencing problem with multiple objectives. Artif Intell Eng 15:309–317
16. Ying KC, Liao CJ (2004) Ant colony system for permutation flow-shop sequencing. Comput Oper Res 31:791–801
17. Dorigo Colorni A, Dorigo M, Maniezzo V, Trubian M (1994) Ant system for job-shop scheduling. Belg J Oper Res 34:39–53
18. Zwaan SVD, Marques C (1999) Ant colony optimization for job shop scheduling. Proceedings of the 1999 workshop on genetic algorithms and artificial life (GAAL'99), Lisbon, Portugal
19. Dorigo M (1992) Learning and natural algorithm. PhD thesis, DEI, Politecnico di Milano, Italy
20. Zheng WX, Nagasawa H, Nishiyama N (1993) Single-machine scheduling for minimizing total cost with identical, asymmetrical earliness and tardiness penalties. Int J Prod Res 31:1611–1620