Implementation of an Ant Colony Optimization technique for job shop scheduling problem

Jun Zhang, Xiaomin Hu, X. Tan, J.H. Zhong and Q. Huang

Department of Computer Science, Sun Yat-sen University, P.R. China

Research on optimization of the job shop scheduling problem (JSP) is one of the most significant and promising areas of optimization. Instead of the traditional optimization method, this paper presents an investigation into the use of an Ant Colony System (ACS) to optimize the JSP. The main characteristics of this system are positive feedback, distributed computation, robustness and the use of a constructive greedy heuristic. In this paper, an improvement of the performance of ACS will be discussed. The numerical experiments of ACS were implemented in a small JSP. The optimized results of the ACS are favourably compared with the traditional optimization methods.

Key words: job shop scheduling problem (JSP); Ant Colony System (ACS); Ant Colony Optimization (ACO); natural computation (NC).

1. Introduction

The job shop scheduling problem (JSP) is one of the most difficult combinatorial problems in classical scheduling theory (Garey *et al*., 1976) and it is NP-hard (non-deterministic polynomial time-hard). Only some special cases with a small number of jobs and machines can be solved in polynomial times (Ling, 2003). The standard model

Address for correspondence: Jun Zhang, Department of Computer Science, Sun Yat-sen University, P.R. China. E-mail: junzhang@ieee.org

of a JSP is the $J/M/G/C_{\max}$ model, where J and M stand for the number of jobs and machines, and the G and $C_{\max}$ represent the precedence rules and the minimum makespan, respectively. Makespan is the time used in completing operations of all jobs. $C_{\max}$ is the maximum earliest completion time of the last operation of any job. In 1989, Carlier and Pinson used a branch-and-bound algorithm to solve $10/10/G/C_{\max}$ exactly. Many methods have been applied to the JSP, including linear programming, Lagrangian relaxation, branch-and-bound, constraint satisfaction, local search, neural networks, (Huang and Ma, 1999; Huang *et al*., 2005) expert systems (Jain and Meeran, 1999), genetic algorithm (Candido *et al*., 1998; Della Croce *et al*., 1995; Ling, 2003; Mati *et al*., 2001), greedy heuristic (Mati *et al*., 2001), Taboo Search (Bames and Chambers, 1995; Dell Amico and Trubian, 1993; Wan and Feng, 2003), simulated annealing (van Laarhoven *et al*., 1992; Matsuo *et al*., 1988) and the Ant System (Colorni *et al*., 1993a; Zhou *et al*., 2004).

The Ant Colony System (ACS) algorithm is a distributed algorithm which is extensively used to solve NP-hard combinatorial optimization problems. Its original model is based on the foraging behaviour of real ants who find an approximately shortest way to the food by detecting the density of pheromone deposited on the route. Pheromone for the real ants is a chemical substance deposited by ants as they walk, but here it acts as something that lures the artificial ants. Deneubourg has shown that ants create pheromone paths from their nests to a food source and that the path with the highest pheromone concentration is the shortest path between the nest and the food source. In the computer-based algorithm – ACS algorithm – its objective is also to find a 'shortest route', ie, a solution with the optimal cost over all feasible ones.

In this paper, the ACS algorithm was applied in a JSP. The parameters in ACS are tuned to make the results better. The ACS in this paper used $\alpha = 0.1$, $\beta = 2.0$, $\rho = 0.01$ and $q_0 = 0.8$ in solving the JSP. The case when $\beta = 0$ is studied and a conclusion is drawn.

This paper is structured as follows. In Section 2, the JSP is explained and is formally defined. In Section 3, a modified ACS algorithm by Dorigo is described. In Section 4, the transformations of ACS applied to JSP are given. In Section 5, there are case studies with the ACS algorithm described in this paper and a comparison is made with optimal values in seven classic JSP problems. The summary and conclusions are given in Section 6.

2. Jop shop scheduling problem

2.1 Definition of a JSP

A JSP may be formulated as follows: given an $n \times m$ static JSP, in which n jobs must be processed exactly once on each of m machines, the set of n jobs can be defined as $J = \{J_1, \ldots, J_n\}$, while the set of m machines is $M = \{M_1, \ldots, M_m\}$. Each job is routed through the m machines in a pre-defined order, which is also known as operation precedence constraints. The processing of a job on one machine is called an operation, and the processing of job i on machine j is denoted by u_{ij}. So the set of operations can be defined as $\boldsymbol{O} = \{u_{ij} | i \in [1, n], j \in [1, m]\}$, in which n denotes the number of jobs and m denotes the number of machines. Once processing is initiated, an operation cannot be interrupted, and concurrency is not allowed. That is, processing u_{ij} cannot begin processing until $u_{i,j-1}$ has completed.

Each operation of $u_{ij} \in \boldsymbol{O}$ on machine j of job i must have an integral processing time p_{ij} ($p_{ij} > 0$), which is also known as the machine processing constraints. The set $\boldsymbol{O}$ is decomposed into chains corresponding to the jobs: if the relation $u_{ip} \to u_{iq}$ is existed in a chain, then both operations u_{ip} and u_{iq} belong to job J_i and there is no machine k, which is not p or q, and relations such as $u_{ip} \to u_{ik}$ or $u_{ik} \to u_{iq}$. This means, in relation $u_{ip} \to u_{iq}$, u_{iq} is directly immediate to u_{ip}.

The value $C_{ij} = C_{ik} + p_{ij}$ is a completion time in operation u_{ij} in relation $u_{ik} \to u_{ij}$. p_{ij} is pre-set and the problem is only to find out the completion time C_{ij} ($\forall u_{ij} \in \boldsymbol{O}$) which minimizes:

$$C_{\max} = \max_{all\ u_{ij} \in O}(C_{ij}) = \max_{u_{ik} \to u_{ij}}(C_{ik} + p_{ij}) \tag{1}$$

which is goal of our JSP.

2.2 Constraints for the JSP

The JSP subjects to two constraints, known as the operation precedence constraint and machine processing constraint:

1) The operation precedence constraint on the job is that the order of operations of job is fixed and the processing of an operation cannot be interrupted and concurrent.

$$\neg \exists k(u_{ip} \to u_{ik} \vee u_{ik} \to u_{iq}), \text{ when } u_{ip} \to u_{iq},\ k \neq p \wedge k \neq q \tag{2}$$

Considering delays in a job such as waiting time for a machine during operations, we can obtain

$$C_{ij} \geq C_{ik} + p_{ij} \text{ when } u_{ik} \to u_{ij} \tag{3}$$

2) The machine processing constraint is that only a single job can be processed at the same time on the same machine

$$C_{ij} \geq C_{kj} + p_{ij}, \quad \text{operation } u_{kj} \text{ is finished before } u_{ij} \tag{4}$$

The operations must be assigned to the time intervals in such a way that once an operation is started it must be completed.

2.3 Representation of the JSP

The disjunctive graph $D = (\boldsymbol{N}, \boldsymbol{A}, \boldsymbol{B})$ is a useful tool for visualizing the problem. N is the set of nodes, which correspond to all of the operations. $\boldsymbol{A}$ is the set of conjunctive directed arcs that are based on the precedence rules, and $\boldsymbol{B}$ is the set of disjunctive, undirected edges that connect two operations from two different jobs that are processed on the same machine. On a disjunctive graph associated with an instance of JSP, $\boldsymbol{N}$, $\boldsymbol{A}$ and $\boldsymbol{B}$ are defined as:

$\boldsymbol{N} = O \cup \{u_0\} \cup \{u_{N+1}\}$; $\{u_0\}$ and $\{u_{N+1}\}$ are the special dummy nodes which identify the start and the completion of the overall job shop. $\boldsymbol{N}$ is the total number of valid operations that $\boldsymbol{N} = n*m$, here n is the number of jobs while m is the number of machines.

$\boldsymbol{A} = \{(u_{ij}, u_{i,j+1}) | u_{ij} \rightarrow u_{i,j+1}$ is the operation order for job J_i, $1 \leq i \leq n, 1 \leq j \leq m\} \cup \{(u_0, u_{i1})$: u_{i1} is the first operation for job J_i, $1 \leq i \leq n\} \cup \{(u_{im}, u_{N+1})$: u_{im} is the last operation for job J_i, $1 \leq i \leq n\}$.

$\boldsymbol{B} = \{(u_{ij}, u_{hj}) | 1 \leq i \leq n,\ 1 \leq j \leq m,\ h \neq i\}$.

The edge weight is associated with the processing time p_{ij} with each operations $u_{ij} \in \boldsymbol{O}$ ($p_0 = p_{N+1} = 0$). The length of a path is defined as the sum of the weight within the constraints of jobs and machines of all operations in this path from the start point to the completion point of a job.

An example of $3/3/G/C_{\max}$ JSP (FT03) with three jobs and three machines is shown in Figure 1.

Here, each node represents one operation which is assigned with a unique number based on the job and the use of the machine. The tuples (*job, machine*) *time* beside the operation node stand for the number of job and machine and the processing time of this operation. For example, (1, 2) 5 beside node 1 shows us operation 1 belongs to job 1 and will be handled on machine 2, with the processing time of 5. We can also use u_{11} to stand for node 1, while the latter 1 stands for the first operation in job 1. The machine that job i used in its jth operation is defined in $u_{ij}.machine$. Hence, we know that all of the nodes in Figure 1 form the node set $\boldsymbol{N}$, while nodes 0 and 10 are the special dummy nodes which identify the start and the completion of the job shop. In a JSP of n jobs and m machines, the maximum number of nodes is $|\boldsymbol{N}| = n*m + 2$. Besides node 0 and node 10, every line of nodes in Figure 1 makes up of one specific job, eg, nodes 1, 2 and 3 make up job 1, while nodes 4, 5 and 6 is job 2, and so on. Every job must use m machines once and only once, so there are m operations in each job. With a node of number *index* ($0 < index < |\boldsymbol{N}|$), we can get it as in job $(index - 1)/m$ with the operation $(index - 1)\%m + 1$ in the job. The arrowheads in Figure 1 show us the operation order of each operation. Based on the operation precedence constraint, we know that each node has exactly one pre-operation node and exactly one immediate one, excluding the start node and the completion node.

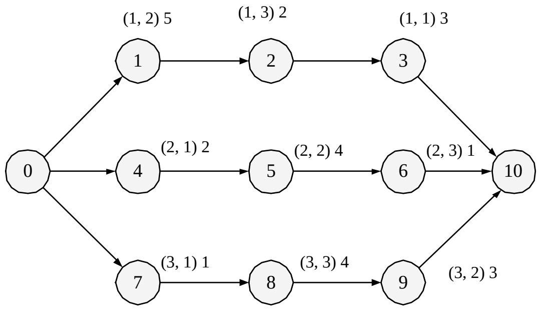

Figure 1 Graphic representation of $3/3/G/C_{\max}$ JSP (FT03)

	m1	m2	m3
j1	12	5	9
j2	2	9	10
j3	3	12	7

Figure 2 Matrix description of Figure 1

There is a solution of the problem in Figure 1, with a processing chain as {4, 1, 7, 8, 2, 3, 5, 9, 6}, in which the numbers of the chain stand for the node number. The procedure of the solution is described in Figure 2.

The values C_{ij} in the matrix are the completion times of job i on machine j. All of the operations are started at time 0. First, node 4 is executed, which is from job 2 and operates on machine 1. During this time, machine 1 is busy until it finishes the operation in node 4. When the operation on node 4 finishes, the time is 2, so $C_{21} = 2$. When node 4 is operating, machines 2 and 3 are free, so any operation demand within the job precedence constraint on these two machines could be satisfied. Node 1 is executed, which is operated on machine 2 of job 1. When this finishes, the time is 5, so $C_{12} = 5$. Node 7 is executed, which is operated on machine 1 of job 3. At time 2, the operation of job 2 on machine 1 has completed, and the operation in node 7 is the first operation in job 3, so the execution in node 7 can start at time 2 and complete at time 3, so $C_{31} = 3$. To node 8, its precedent is node 7, which has been finished at time 3. The immediate operation of node 7 is node 8, so the next operation in job 3 is node 8. At time 3, machine 3 is free so that node 8 can be started and it completes at time 7, so $C_{33} = 7$. The rest of the operations can be analysed in the same way. When all the operations have been completed, the time we get is 12, which is the makespan of the operation chain. Note that the procedure in the processing chain does not violate the constraints of the JSP. Our aim is to find the feasible operation chain of the minimum makespan.

3. Ant Colony System

3.1 The fundamentals of Ant Colony Optimization (ACO)

Ant colonies exhibit very interesting behaviours: one ant has limited capabilities, but the behaviour of a whole ant colony is highly structured. They are capable of finding the shortest path from their nest to a food source, without using visual cues but by exploiting pheromone information (Beckers *et al.*, 1992; Hölldobler and Wilson, 1990; Huang and Zhao, 2005). While walking, ants can deposit some pheromone on the path. The probability that the ants coming later choose the path is proportional to the amount of pheromone on the path, previously deposited by other ants.

Figure 3 (a)–(d) show the ants' miraculous behaviour. Considering Figure 3(a), ants want to find food, so they set off from their nest and arrive at a decision point at which they have to decide which path to go on, for there are three different paths. Since they have no clue about which is the best choice, they choose the path just randomly, and

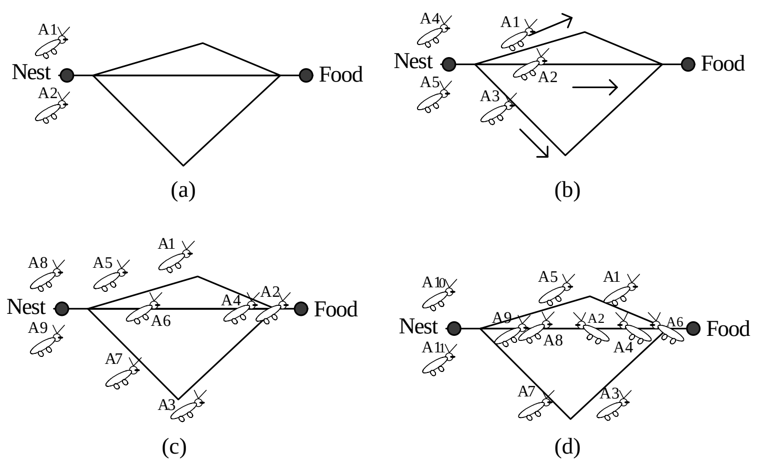

Figure 3 How real ants find a shortest road. (a) Ants arrive at a decision point. (b) Ants choose three roads randomly. (c) Since ants move at a constant speed, the ants which choose the shortest road in the middle reach the food faster than those who choose the other two roads. (d) Pheromone accumulates at a higher rate in the middle road, and more ants will choose this road.

on average the numbers of ants on every path are the same. Figure 3(b) shows what happens in the instant immediately following, supposing that all ants walk at the same speed and deposit the same amount of pheromone. In Figure 3(c), since the middle path is the shortest one, ants following this path reach the food point first. Therefore more ants will complete their tour through the middle path in the same period of time, and more pheromone will be deposited in this road correspondingly. Figure 3(d) shows what happens when ants return to their nest after they find the food; since there is more pheromone in the middle path, ants will prefer in probability to choose the middle path. This in turn increases the number of ants choosing the middle and shortest path. This is a positive feedback effect with which very soon all ants will follow the shortest path.

Real ants are not only capable of finding the shortest path from a food source to the nest (Colorni *et al*., 1993; Dorigo and Gambardella, 1997; Hölldobler and Wilson, 1990) without using visual cues, but also they are capable of adapting to changes in the environment. For example, they will find a new shortest path once the old one is no longer feasible because of an obstacle placed in the way (Beckers *et al*., 1992). The notions and motivations of the ant algorithm were originally introduced in Colorni and Dorigo (1991) and Colorni *et al*. (1993). Studies of ants' behaviour on problem solving and optimization were explored in those papers.

The first algorithm inspired by the behaviour of real ants was the Ant System (AS), which was first derived by Dorigo (1992). AS has been tested on some small TSP (Travelling Salesman Problem) instances and compared with other general heuristics. Some initial results were promising and have shown the viability of the approach. However, for larger size instances, AS gives a very poor solution quality compared with other heuristic algorithms. To improve the performance, a new ant algorithm, called Ant Colony System (ACS), was presented in 1997 (Dorigo and Gambardella, 1997).

3.2 State transition rule

The basic model of the ACS is that: m ants are initially placed at a start point with n decision points to the destination, and then let go to find routes according to some rules.

At the very beginning, we assume that the pheromone in every possible tour is at a quite low level τ_0. Dorigo *et al.* (1996) has made some experiments on $\tau_0 = 0$, $\tau_0 = 1/(n^*L_{nn})$, τ_0 is inspired by Q-learning. L_{nn} is the tour length produced by the nearest neighbour heuristic (Rosenkrantz *et al.*, 1977) and n is the number of decision points. He found that the result on $\tau_0 = 0$ is worse than the other ones and the latter ones perform quite well. In this paper, we choose to use $\tau_0 = 1/(n^*L_{nn})$ for the reason that it is simpler and the performance is adequate.

To accelerate the convergent speed of the ants, which is also known as the speed of finding a best way, once at a decision point, the ants make their choices based on the pheromone on the routes and the length of the selective routes. The more ants select the specific route, the more pheromone is dropped and the route must be a short one. To fully utilize the guiding information and avoid early convergence of finding a route, two selection methods are applied. Early convergence happens when too many ants gather in a wrong path and the pheromone becomes so dense that a better route cannot be discovered.

Each ant builds a tour by repeatedly applying a stochastic greedy rule, which is called the state transition rule.

$$s = \begin{cases} \arg\max_{u \in J(r)} \{[\tau(r,u)] \cdot [\eta(r,u)^{\beta}]\}, & \text{if } q \le q_0 \text{ (exploitation)} \\ S, & \text{otherwise (biased exploitation)} \end{cases} \tag{5}$$

(r, u) represents an edge between point r and u, and $\tau(r, u)$ stands for the pheromone on edge (r, u). $\eta(r, u)$ is the desirability of edge (r, u), which is usually defined as the inverse of the length of edge (r, u). q is a random number uniformly distributed in $[0,1]$, q_0 is a user-defined parameter with $(0 \le q_0 \le 1)$, β is the parameter controlling the relative importance of the desirability. $J(r)$ is the set of edges available at decision point r. S is a random variable selected according to the probability distribution given below.

$$P(r,s) = \begin{cases} \dfrac{[\tau(r,s)] \cdot [\eta(r,s)^{\beta}]}{\sum\limits_{u \in J(r)} [\tau(r,u)] \cdot [\eta(r,u)^{\beta}]}, & \text{if } s \in J(r) \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

The selection strategy used above is also called ‘roulette wheel’ selection since its mechanism is a simulation of the operation of a roulette wheel. Every city has its percentage in the roulette wheel and the bigger this percentage is, the larger the width of slot in the wheel so that the probability of choosing that city becomes larger. After a random spinning of the wheel, which is performed by generating a random number, a slot is chosen and the next route the ant will go on is determined.

3.3 Pheromone updating rule

A certain amount of pheromone is dropped when an ant goes by. It is a continuous process, but we can regard it as a discrete release by some rules. Here we introduce two kinds of pheromone update strategies, called local updating rule and the global updating rule.

3.3.1 Local updating rule: While constructing its tour, an ant will modify the amount of pheromone on the passed edges by applying the local updating rule.

$$\tau(r,s) \leftarrow (1-\rho)\cdot\tau(r,s) + \rho\cdot\tau_0 \tag{7}$$

where ρ is the coefficient representing pheromone evaporation (note: $0 < \rho < 1$).

3.3.2 Global updating rule: Once all ants have arrived at their destination, the amount of pheromone on the edge is modified again by applying the global updating rule.

$$\tau(r,s) \leftarrow (1-\alpha)\cdot\tau(r,s) + \alpha\cdot\Delta\tau(r,s) \tag{8}$$

where

$$\Delta\tau(r,s)\begin{cases}(L_{gb})^{-1}, & \textit{if } (r,s)\in \text{global-best-tour}\\ 0, & \text{otherwise}\end{cases} \tag{9}$$

Here $0 < \alpha < 1$ is the pheromone decay parameter, and L_{gb} is the length of the globally best tour from the beginning of the trial. $\Delta\tau(r,s)$ is the pheromone addition on edge (r, s). We can see that only the ant that finds the global best tour can achieve the pheromone increase.

3.4 The performance of the ACS

The flowchart for the ACS algorithm can be defined in Figure 4.

At the beginning of the algorithm, all parameters are initialized. There are m ants in the colonies that are placed at the starting points, and one iteration is completed when all ants arrive at the destination. When an ant chooses an edge to go on, it performs a local pheromone update. When the whole ant colony has reached the destination, global pheromone updating is performed. It influences the next iteration of ant colonies finding routes. When the maximum number of iterations reached, the algorithm stopped and the shortest route from the start point to the destination is found.

4. Integration of ACS and JSP

4.1 Conversion from the ACS problem formulation to the JSP

From the ACS model described above, we can convert it to solve the JSP. In a JSP of n jobs and m machines, the modified ACS is described as follows.

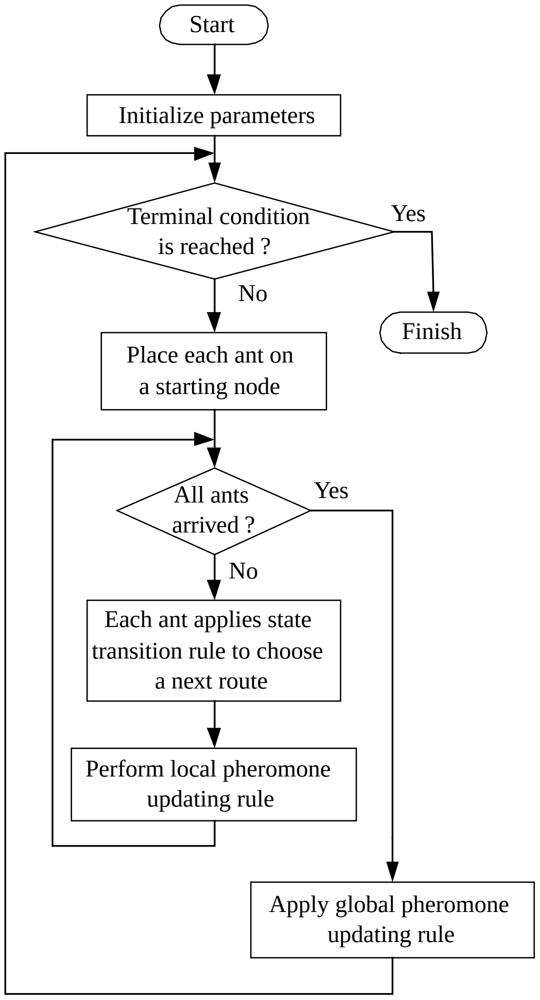
Start
Initialize parameters
Terminal condition is reached ?
Yes
No
Finish
Place each ant on a starting node
All ants arrived ?
Yes
No
Each ant applies state transition rule to choose a next route
Perform local pheromone updating rule
Apply global pheromone updating rule

Figure 4 Flowchart for ACS

By adding two dummy nodes which identify the start and the completion of the overall job shop, we now have the start point and the destination in the ACS. At first, ants are all allocated in operation u_0 (as in node 0), whose aim is u_{N+1} (as in node $n*m+1$).

The set of next operations for an ant in node i to visit is not all those not visited, as in the general ACS, but the nodes that follow the operation precedence constraints (machine constraints do not need to be considered here, for the ants' tour is linear). The set ant k can visit in the next operation is S_k, which is formed dynamically according to the order constraint of jobs and the working states (busy or free) of the machines. At the beginning of the iteration, S_k is initialized to be the first operation of all the jobs, ie, $S_k = \{u_{i1} | i \in [1, n]\}$, where n is the number of jobs. We can determine which machine the operation is using by $u_{ij}.machine$. When ant k moves from u_{ip} to u_{iq}, u_{ip} is deleted but u_{iq} is added to S_k if machine $u_{iq}.machine$ is free. All the next operations in the other jobs whose pre-operations have completed but where the new one has not yet been started are also added to S_k.

The transition rule of ant k in ACO can be changed to:

$$s = \begin{cases} \arg\max_{u \in S_k} \{[\tau(r,u)] \cdot [\eta(u)^{\beta}]\}, & \text{if } q \le q_0 \text{ (exploitation)} \\ S, & \text{otherwise (biased exploitation)} \end{cases} \tag{10}$$

and

$$P_k(r,s) = \begin{cases} \dfrac{[\tau(r,s)] \cdot [\eta(s)^{\beta}]}{\sum_{u \in S_k} [\tau(r,u)] \cdot [\eta(u)]^{\beta}}, & \text{if } s \in S_k \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

$\tau(r, s)$ is the pheromone on the edge (r, s), which is from node r to node s. $\eta(s) = 1/p(s)$, which is the inverse of the processing time of operation s. S_k is the set of operations that ant k can visit in the next step.

Correspondingly, $\Delta\tau(r,s)$ in ACO is changed to

$$\Delta\tau(r,s) = \begin{cases} (T_{gb})^{-1}, & \textit{if } (r,s) \in \text{global-best-tour} \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

where T_{gb} is the global best scheduling time in a complete job shop iteration.

The remaining formulas in ACO are used as their original types.

4.2 The performance of ACS in JSP

After completing all operations, we have to compute the maximum scheduling time of this tour. The algorithm of computing the scheduling time of a tour is shown in Figure 5.

The algorithm of the ACS for the JSP mainly includes two loops, which is the same as the original ACS in Figure 4. A complete pseudocode for ACS in JSP is presented in the Appendix.

5. Case study

Seven classic JSP problems (Ling, 2003) have been tested by using the ACS algorithm in this paper, which are FT03, FT06, ABZ6, LA06, LA07, LA11 and LA36. Table 1 shows the average results and the best results computed by the ACS algorithm in 100 runs of the program. The known optimal values of these problems are also listed in Table 1 and an error violation is made by comparing the ACS_Best with the optimal values (Kim and Lee, 1994). The '−' in the grid stands for no optimal value discovered until now.

In the table, we can see the results made by the ACO algorithm. In the FT03 problem, ACO obtained the optimal value. In FT06, ABZ6 and LA06 problems, ACO achieved modest results. However, in LA07, LA11 and LA36 problems, ACO could not reach the optimal value, but it has approached the optimal results.

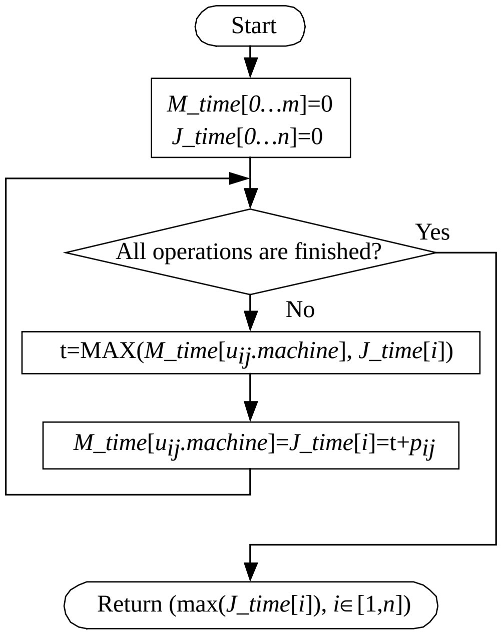
Start
M_time[0…m]=0
J_time[0…n]=0
All operations are finished?
Yes
No
t=MAX(M_time[u_ij.machine], J_time[i])
M_time[u_ij.machine]=J_time[i]=t+p_ij
Return (max(J_time[i]), i∈[1,n])

Figure 5 Flowchart for computing the maximum scheduling time

In the experiments, the setting of parameters is very important. Experimental observation has shown that $\alpha = 0.1$, $\beta = 2.0$, $\rho = 0.01$ and $q_0 = 0.8$ are the best choices of these parameters in general, which are also the choices of tests used in Table 1. τ_0 is the initial pheromone on the edges of the ACS algorithm. The setting of τ_0 is T_{nn}^{-1}, where T_{nn}^{-1} is the total time of all operations by the nearest neighbour heuristic.

Figures 6–8 show the convergent speed of the ACS for the JSP (for problems LA06, ABZ6 and LA07) in 1000 iterations, from which we can see that the convergent speed is quite fast, especially in the first 500 iterations.

It is interesting that $\beta = 0$ is a better choice for many given problems such as LA06, FT03 and FT06, and Figure 9 shows this strange instance (ACS for LA06 and FT06),

Table 1 The average and best results of seven problems of 100 runs using ACS for JSP

JSP name	*n*	*m*	Average	ACS_Best	Optimal value	Error
FT03	3	3	12	12	12	0%
FT06	6	6	59.1	55	–	–
ABZ6	10	10	1245	1154	–	–
LA06	15	5	1024	934	926	0.8%
LA07	15	5	1020	917	890	3%
LA11	20	5	1379	1254	1222	2.6%
LA36	15	15	1612	1461	1268	15.2%

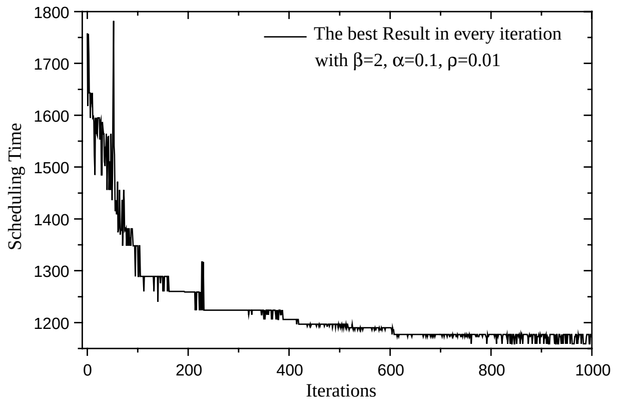

Figure 6 The result of ACS for LA06

from which it is very clear that the ACS with $\beta = 0$ finds a good solution and the convergent speed is also quicker than the ACS with $\beta = 2$.

There is an explanation for this instance. For some given problem, LA06 for example, constraints (the operation precedence constraint and the machine constraint) are very strict. For example, consider the ant located in operation u_{11}, and a processing time of u_{11} is 6, the *next_set* is $\{u_{12}, u_{21}\}$, the processing time for u_{12} and u_{21} are 1 and 5, the pheromones of these two operations are 0.1 and 0.2, respectively. According to (3), the probability transfer to u_{12} is larger than that of u_{21}, so ant will probably choose u_{12} as its next-visit operation. However, where u_{12} belongs to the same job as u_{11}, if the ant choose u_{12} as its next-visit operation, it has to wait 6 units of processing time until u_{11} is processed, so u_{12} is a poorer choice than u_{21}. If $\beta = 0$, the ant has a bigger probability to choose the right next-visit operation u_{21} only according to pheromone. However, this is not true for all JSP problems, it is necessary to test different values for parameters in solving specific problems.

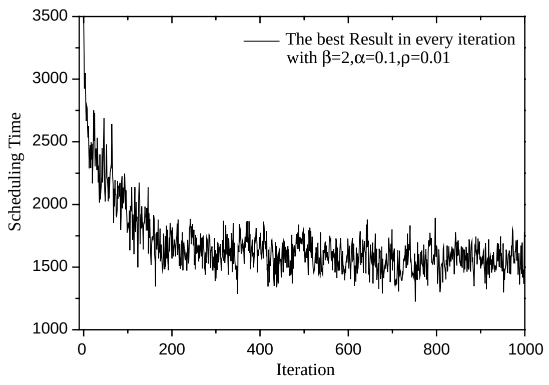

Figure 7 The result of ACS for ABZ6

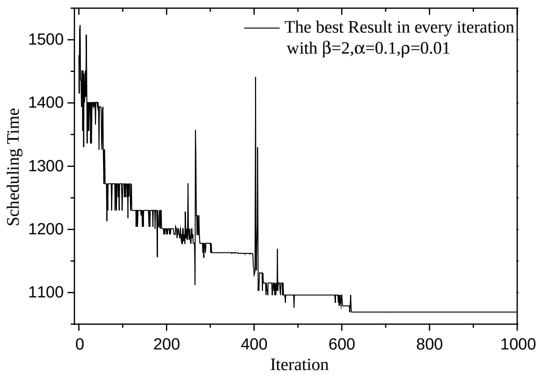

Figure 8 The result of ACS for LA07

6. Summary and conclusion

This paper has applied the ACS to a JSP problem and analysed the experiment results. The results have shown that the ACS is an effective method for the JSP and it can find a good solution. However, the performance of the ACS for the JSP largely depends on

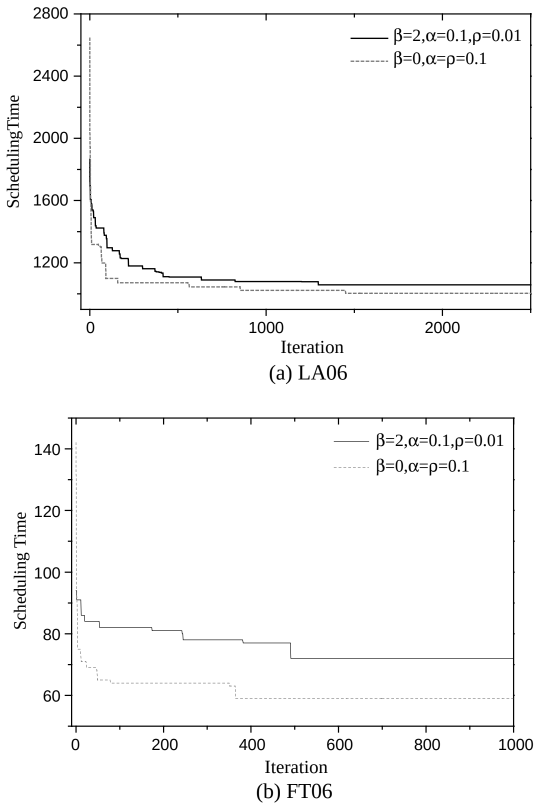

Figure 9 Comparison of $\beta = 0$ and 2.0 (ACS for LA06 and FT06)

the parameter values and the number of the ants. Adjusting these parameter values takes a great deal of time, for the optimal parameter values depend on the problem to solve, and it is difficult to find an all-purpose setting of parameters for all problems.

Acknowledgements

This work was supported by the National Natural Science Foundation of China project No. 60573066 and Guangdong Natural Science Foundation project No. 5003346.

References

Bames, J.W. and **Chambers, J.B.** 1995: Solving the job-shop scheduling problem using tabu search. *IIE Transactions* 27, 257–63.

Beckers, R., Deneubourg, J.L. and **Goss, S.** 1992: Trails and U-turns in the selection of the shortest path by the ant Lasius niger. *Journal of Theoretical Biology* 397–415.

Candido, M.A.B., Khator, S.K. and **Barcia, R.M.** 1998: A genetic algorithm based procedure for more realistic job shop scheduling problem. *International Journal of Production Research* 36, 3437–57.

Colorni, A., Dorigo, M. and **Maniezzo, V.** 1991: Distributed optimization by ant colonies. In *Proceedings of ECAL91 – European Conference on Artificial Life*. Elsevier, 134–42.

Colorni, A., Dorigo, M., Maniezzo, V. and **Trubian, M.** 1993: Ant system for Job-Shop scheduling. *Belgian Journal of Operations Research, Statistics and Computer Science* 34, 39–54.

Dell Amico, M. and **Trubian, M.** 1993: Applying tabu search to the job shop scheduling problem. *Annals of Operations Research* 41, 231–52.

Della Croce, F., Tadei, R. and **Volta, G.** 1995: A genetic algorithm for the job shop problem. *Computers & Operations Research* 22, 15–24.

Dorigo, M. 1992: Optimization, learning, and natural algorithms. PhD thesis, Politecnico di Milano.

Dorigo, M. and **Gambardella, L.M.** 1997: Ant Colony System: a cooperative learning approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation* 1, 53–66.

Dorigo, M., Maniezzo, V. and **Colorni, A.** 1996: Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics* 26, 29–41.

Garey, M.R., Johnson, D.S. and **Sethi, R.** 1976: The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research* 1, 117–29.

Hölldobler, B. and **Wilson, E.O.** 1990: *The ants*. Springer-Verlag.

Huang, D.S. and **Ma, S.D.** 1999: Linear and nonlinear feedforward neural network classifiers: a comprehensive understanding. *Journal of Intelligent Systems* 9, 1–38.

Huang, D.S. and **Wen-Bo Zhao**, 2005: Determining the centers of radial basis probabilities neural networks by recursive orthogonal least square algorithms. *Applied Mathematics and Computation* 162, 461–73.

Huang, D.S., Ip, H.S., Law, K.C.K. and **Zheru Chi**. 2005: Zeroing polynomials using modified constrained neural network approach. *IEEE Transanction on Neural Networks* 16, 721–32.

Jain, A. and **Meeran, S.** 1999: Deterministic job-shop scheduling: past, present and future. *European Journal of Operational Research* 113, 390–434.

Kim, G.H. and **Lee, C.S.G.** 1994: An evolutionary approach to the job-shop scheduling problem. *Robotics and Automation. Proceedings 1994 IEEE International Conference*, 8–13 May 1994, Volume 1, 501–506.

Mati, Y., Rezg, N. and **Xiaolan Xie.** 2001: An integrated greedy heuristic for a flexible job shop scheduling problem. *Systems, Man, and Cybernetics, 2001 IEEE International Conference* 7–10 October 2001, Volume 4, 2534–39.

Matsuo, H., Suth, C.J. and **Sullivan, R.S.** 1988: *A controlled search simulated annealing method for the general Job-Shop scheduling problem*.

Graduate School of Business, University of Texas.

Rosenkrantz, D.J., Stearns, R.E. and **Lewis, P.M.** 1977: An analysis of several heuristics for the traveling salesman problem. *SIAM Journal of Computing* 6, 563–81.

van Laarhoven, P.J.M., Aarts, E.H.L. and **Lenstra, J.K.** 1992: Job shop scheduling by simulated annealing. *Operations Research* 40, 113–25.

Wan, G. and **Wan, F.** 2003: Job shop scheduling by taboo search with fuzzy reasoning. *Systems, Man and Cybernetics, 2003. IEEE International Conference* 5–8 October 2003, Volume 2, 1566–70.

Wang Ling. 2003: *Shop scheduling with genetic algorithm*. TsingHua University Press, 59–67.

Zhou Pin, Li Xiao-ping and **Zhang Hong-fang.** 2004: An ant colony algorithm for Job Shop scheduling problem. *Proceedings of the 5th World Congress on Intelligent Control and Automation*, 15–19 June.

Appendix: The algorithm of ACS for JSP

```
1)/* Initialization phase*/
  For each pair(r, s), $\tau(r, s) = \tau_0$ End-for
  For $i = 1$ to ANTS do
    Let visited to be empty for ant i;/*visited is the set ant $i$ has visited*/
    Let next_set = {$\text{Operation}_{11}$, $\text{Operation}_{21}$, ..., $\text{Operation}_{\text{JOBS},1}$}
      /*next_set is the set can be visited by ant $i$ in the next step*/
      Let time = 0, Tour[0] = 0 and current_operation = 0 for ant $i$;
      /*time is the time used by ant $i$, Tour is the array records the tour visited by ant $i$*/
      /*current_operation is the operation where ant $i$ is located*/
  End-for
2)/*This is the phase in which ants build their tours. The tour of ant $i$ is stored in $\text{Tour}_k$*/
  For $i = 1$ to OPERATIONS-1 do
    If $i <$ OPERATIONS-1 then
       For $k = 1$ to ANTS do
          IF is the first iteration AND ant $k$ is in the first operation Then
          /*In this program, the number of ants is as same as that of Jobs, */
              /*and at the beginning of the program each Job has a ant*/
             Let the first operation of Job $k$ to the ant $k$'s first visited operation
             Let visited[$k$][1] = 1 for ant $k$ /*mark the visited operation for ant $k$*/
             Put the $\text{Operation}_{k,2}$ into next_set
             Let Tour[1] = the first operation of Job $k$ for ant $k$;
          Else
             Choose the next operation according to function choose_next_operation
             Let Tour[$i+1$] = next_operaion for ant $k$
             /*next_operation is the [(next_operation-1)%MACHINES + 1]th operation*/
                   /* of Job [(next_operation-1)/MACHINES] */
             Let visited [(next_operation-1)/MACHINES]
                         [(next_operation-1)%MACHINES + 1] = 1
           If next_operation is not the last operation of some job Then
               Put the operation $u_{ij}$ after next_operation (next_operation $\to u_{ij}$)
```

```
                              into the next_set of ant $k$;
                    End-if
                End-if
            End-for
        Else
                For $k = 1$ to ANTS do
                Let OPERATIONS-1 is the next operation for ant $k$
                Let Tour[$i + 1$] = OPERATIONS-1
                End-for
        End-if
            /*In this phase local updating occurs and pheromone is updating*/
            For $k = 1$ to ANTS do
                $\tau$(current_operation, next_operation) = $(1 - \rho)\tau$(current_operation, next_operation)
                $+ \rho\tau_0$
                current_operation = next_operation for ant
            End-for
        End-for /*end a iteration* /
3)/*In this phase global updating occurs and pheromone is updated*/
        For $k = 1$ to ANTS do
            Computer the $time_k$ for ant $k$ /* $time_k$ is the time used by ant $k$*/
        End-for
        Find the shortest time from all $time_k$ and $tour_{best}$ as well
        /*Update edge belong to $tour_{best}$ */
        For each edge($r$, $s$) belong to $tour_{best}$ do
                $\tau(r, s) = (1 - \alpha)\ \tau(r, s) + \alpha(shortest\_time)^{-1}$
        End-for
4) If (End_condition = True) Then
            Print the result
        Else goto 2
```