Research Journal of Applied Sciences, Engineering and Technology 4(20): 3999-4006, 2012
ISSN: 2040-7467

Submitted: December 20, 2011 Accepted: April 23, 2012 Published: October 15, 2012

Object-Guided Ant Colony Optimization Algorithm with Enhanced Memory for Traveling Salesman Problem

Lijin Wang, Rongying Cai, Lin Jing and Hui Zhang
College of Computer and Information Science, Fujian Agriculture and Forestry University, Fuzhou, 350002, China

Abstract: In this study, we presents an object-guided ACO algorithm which is consisted of ants with enhanced memory. In the process of solution construction, each ant stores a complete solution in its enhanced memory. Each time ant selects a solution component probabilistically, it will calculate the difference between current solution and the new solution after adding the selected component and then Metropolis accepting rule, which has been used in simulated annealing algorithm successfully, is used to decide whether to accept the component or discard it. The simulation results, which were carried on benchmark traveling salesman problems, show that the improvement of individual intelligence can improve the performance of ACO algorithm remarkably.

Keywords: Ant colony optimization, individual intelligence, object-guided, traveling salesman problem

INTRODUCTION

Ant Colony Optimization (ACO) algorithms are a class of most famous swarm intelligence optimization algorithm. The original idea of ACO algorithms comes from observing the exploitation of food resources among ants, in which ants individually limited intelligence have collectively been able to find the shortest path between a food source and the nest. In the natural world, ants (initially) wander randomly and upon finding food return to their nest while laying down pheromone trails. If other ants find such a path, they are likely not to keep travelling at random, but to instead follow the trail, returning and reinforcing it if they eventually find food. Over time, however, the pheromone trail starts to evaporate, thus reducing its attractive strength. The more time it takes for an ant to travel down the path and back again, the more time the pheromones have to evaporate. A short path, by comparison, gets marched over more frequently and thus the pheromone density becomes higher on shorter paths than longer ones. Pheromone evaporation also has the advantage of avoiding the convergence to a locally optimal solution. If there were no evaporation at all, the paths chosen by the first ants would tend to be excessively attractive to the following ones. In that case, the exploration of the solution space would be constrained. Inspired by this phenomenon, (Dorigo *et al*., 1996; Colorni *et al*., 1992a, b) presented the first ACO algorithm and used it to Traveling Salesman Problem (TSP). Since then, ACO algorithms have been studied widely and used in diverse fields.

Memory is one of the basics for an organism to have intelligent behaviors and it is considered to be a main mechanism for intelligent optimization algorithms to have good performance. Many intelligent optimization algorithms have explicitly or (and) implicitly memory mechanism (Taillard *et al*., 2001; Blum and Roli, 2003). We can distinct those memory mechanism as long-term memory and short-term memory. Generally, short-term memory is used to store the recently performed moves, visited solutions, or decision taken etc. and long-term memory is usually used to store an accumulation of synthetic parameters about the search (Blum and Roli, 2003). In ACO algorithms, the pheromone keeps the history of searching, it is an implicit long-term memory; both the inner state, which is used for the construction of new solution and the solution, which is used to update the pheromone, are explicit short-term memory.

Unlike other intelligent optimization algorithms, such as Genetic Algorithm (GA) and Particle Swarm Optimization algorithm (PSO), ACO algorithm is a constructive algorithm, it doesn use the solution constructed in last iteration directly and constructs a new solution from scratch in each iteration. From this aspect, we can say that ant is forgetful; it cannot remember the solution it constructed in last iteration and doesn use it to guide the construction of new solution directly. Moreover, individuals in swarm intelligence are low level agents; they have no explicit desire to reach the object. For example, after ant selects a solution component, it will accept it blindly even if the component will deteriorate the solution and drag it away from the goal. On the other

Corresponding Author: Lijin Wang, College of Computer and Information Science, Fujian Agriculture and Forestry University, Fuzhou, 350002, China

hand, studies in the fields of Simulated Annealing (SA) algorithm and Tabu Search (TS) have showed that object-guided search strategy can improve the intensification ability of intelligent optimization algorithm remarkably. So, if we can improve the intelligence of individual ant by integrating the store of solution constructed in last iteration and Object-Guided strategy, the performance of ACO algorithm may be improved.

Based on the above analysis, this study presents an Object-Guided Ant Colony Optimization (OG-ACO) algorithm. Each ant has an extended short-term memory which is used to store the solution constructed in last iteration. Object-Guided strategy is introduced in the algorithm to decide whether to accept a selected component. In order to prevent premature convergence, metropolis rule of SA algorithm is used to decide whether to accept the selected component. Specifically, it will accept the component when an improvement is confirmed; otherwise, those components, which will lead to worse solutions, will be accepted in some probability. This strategy can improve the intensification ability of ACO algorithm.

In this study, we present a new way to improve the performance of ACO algorithm. It enhances the short-term memory of ant to store a complete solution always and combines the idea of SA algorithm to improve ant's individual intelligence. Moreover, when ant selects a component probabilistically, metropolis rule is used to decide whether to accept it or discard it. Simulation results show this strategy can improve the performance of ACO algorithm effectively.

METHODOLOGY

Ant colony optimization algorithm for traveling salesman problem:

Traveling salesman problem: TSP problem is one of the most famous hard combinatorial optimization problems. It belongs to the class of NP-hard optimization problems. This means that no polynomial time algorithm is known for its solution. Consider a salesman who has to visit *n* cities. The TSP problem consists of finding a shortest tour through all the cities such that no city is visited twice and the salesman returns back to the starting city at the end of the tour. It can be defined as follows. For a *n* cites problem, we can use a distance matrix $D = (d_{i,j})_{n\times n}$ to store distances between all the pair of cites, where,,, each element $d_{i,j}$ of matrix *D* represents the distance between city *i* and *j*. And we use a set of permutations Π of the integers from 1 to *n*, which contains all the possible tours of the problem. The goal is to find a permutation $\pi = (\pi(1), \pi(2), ..., \pi(n))$ that minimizes:

$$f(\pi)=\sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)} + d_{\pi(n),\pi(1)} \quad (1)$$

TSP problem may be symmetric or asymmetric. In the symmetric TSP, the distance between two cities is the same in each opposite direction, forming an undirected graph. This symmetry halves the number of possible solutions. In the asymmetric TSP, paths may not exist in both directions or the distances might be different, forming a directed graph. Traffic collisions, one-way streets and airfares for cities with different departure and arrival fees are examples of how this symmetry could break down.

Ant colony optimization algorithm: We will use the Ant System (AS) (Dorigo *et al*., 1996) for TSP problem as an example to introduce the idea of ACO algorithm. ACO algorithms are based on a parameterized probabilistic model (the pheromone model) that is used to model the chemical pheromone trails. Artificial ants incrementally construct tours by adding opportunely selected paths to a partial tour under consideration. For doing that, artificial ants perform randomized walks on a completely connected graph $G = (V, E)$ whose vertices are the cities *V* and the set *E* are the paths. This graph is commonly called construction graph. Each path $e_{i,j} \in E$ has associated pheromone trail parameter $\tau_{i,j}$. The set of all pheromone trail parameters is denoted by *T*. Furthermore, each path can have associated a heuristic value $\eta_{i,j}$, representing a priori information extracted from the problem instance. The set of all heuristic values is denoted by *H*. These values are used by the ants to make probabilistic decisions on how to move on the construction graph as follows. Suppose at some moment ant *k* is in city v_i, it will select path $e_{i,j}$ as a solution component according to the following probability:

$$p_{i,j}^{k}=\begin{cases}\dfrac{\left[\tau_{i,j}\right]^{\alpha}\left[\eta_{i,j}\right]^{\beta}}{\sum_{\mu\in J_i^k}\left[\tau_{i,u}\right]^{\alpha}\left[\eta_{i,u}\right]^{\beta}}, & if\ \ j\in J_i^k \\ 0, & otherwise\end{cases} \quad (2)$$

where, J_i^k is the set of cities which ant *k* can visit from city v_i at this moment, α and β are parameters used to adjust the relative importance of heuristic information and pheromone values.

Once all ants have constructed their tours, pheromone will be updated as following:

$$\tau_{i,j}(t) = (1-\rho)\tau_{i,j}(t)+\Delta\tau_{i,j}(t) \quad (3)$$

where, $\rho\in(0, 1)$ is the pheromone evaporation rate, it means the pheromone trail will evaporate over time, thus reducing its attractive strength; $\Delta\tau_{i,j}(t)$ is the quantity of pheromone laid on path $e_{i,j}$ by all ants, it is given by:

$$\Delta\tau_{i,j}(t)=\sum_{k=1}^{m}\Delta\tau_{i,j}^{k}(t) \quad (4)$$

where, $\Delta\tau^{k}_{i,j}(t)$ is the pheromone laid on path $e_{i,j}$ by ant k, in AS algorithm it is given by:

$$\Delta\tau^{k}_{i,j}(t)=\begin{cases} Q/L^{k} & if\ e_{i,j}\in Tour^{k}(t) \\ 0 & otherwise \end{cases} \quad (5)$$

where, Q is a constant, L^k is the tour length of ant k and $Tour^k(t)$ is the tour of ant k.

Since the first introduction of AS by Dorigo *et al*. (1996) ACO algorithms have been applied to many combinatorial optimization problems, ranging from TSP problem to protein structure prediction or routing vehicles and a lot of derived methods have been adapted to dynamic problems in real variables, stochastic problems, multi-targets and parallel implementations. Recently, they have been used for some continuous optimization problems also. There are many variants which aim to improve the performance of ACO algorithm. In the early years of ACO research, the focus was in developing ACO variants that modify the pheromone update or the solution generation mechanism to improve the algorithmic performance. More recently, researchers have started to explore combinations of ACO with other algorithmic techniques (Marco and Thomas, 2010).

Dorigo proposed the elitist strategy (Dorigo *et al*., 1996), which consists in giving the best tour since the start of the algorithm a strong additional weight. Bullnheimer *et al*. (1999) proposed the rank-based Ant System (ASrank), which is in a sense an extension of the elitist strategy: it sorts the ants according to the lengths of the tours they generated and, after each tour construction phase, only top some of the best ants and the global-best ant are allowed to deposit pheromone. Stützle and Hoos (2000) proposed the MAX IN Ant System (MMAS), which introduces upper and lower bounds to the values of the pheromone trails, as well as a different initialization of their values. Taillard *et al*. (2001) proposed the Ant Colony System (ACS), which improves over AS by increasing the importance of exploitation of information collected by previous ants with respect to exploration of the search space. Zhu and Yang *et al*. (2004) proposed an ACO algorithm based on the nearest neighbor node choosing, dynamic pheromone updating and mutation. Liangjun *et al*. (2008) proposed a Finite Grade pheromone ACO (FGACO), in which pheromone is classified into finite grades, pheromone updating is realized by changing the grades and the updated quantity of pheromone is independent of the objective function values. Chen *et al*. (2003) proposed an adaptive ant colony algorithm based on equilibrium of distribution, which can dynamically adjust the influence of each ant to the trail information updating and the selected probabilities of the paths according to the equilibrium of the ant distribution. Chang *et al*. (1997) proposed an adaptive MMAS, which improves over MMAS using adaptive adjustment of parameters. More detail review can be found in Marco and Thomas (2010).

The above variants mainly focus on the pheromone update strategies, which aims to improve the cooperative intelligence of ants. Recently, some researchers have tried to improve ACO algorithms through another direction, which uses more short-term memory to improve individual ant intelligence. Acan A proposed ACO algorithm with external memory, which stores partial solution extracted from best solution of each iteration and then ants start the solution construction from partial solutions. Wiesemann *et al*. (2010) proposed iterated ant, which store the complete solution of last iteration and then ants start the solution construction from partial solutions obtained by removing solution components from the complete solution. Tsutsui S proposed cunning ant system which also constructs solution from partial solution obtained from complete solution. Two important advantages of starting the solution construction from partial solutions are that

- The solution construction process is much faster
- Good parts of solutions may be exploited directly (Marco and Thomas, 2010).

The store complete solution and use it reasonably may improve the performance of ACO algorithms. In this study, we will use the stored complete solution in a different way. We will use it to guide the construction of solution directly. Specifically, after an ant has selected a component probabilistically, it will use the stored complete solution and object of optimization to guide him whether to accept it or discard it. We call the proposed algorithm as Object-Guided Ant Colony Optimization (OG-ACO) algorithm.

Object-guided ant colony optimization algorithm:

Basic ideal: Each ant has an enhanced short-term memory, which is used to store a complete solution π. Suppose at some moment ant is in city v_i and it select city v_j (edge $e_{i,j}$) as next visiting city (edge). Canonical ACO algorithms will accept $e_{i,j}$ blindly. Because OG-ACO stores a complete solution, it can accept $e_{i,j}$ intelligently. Specifically, it can decide to accept $e_{i,j}$ or discard it according to the difference of tour length between the stored solution and the new solution gained by adding $e_{i,j}$. If it discards $e_{i,j}$, it will visit the next city of v_i in the original solution and continue the construction from that city. For OG-ACO to work, we must decide two strategies:

- Accepting strategy, which is used to decide whether to accept $e_{i,j}$ or discard it
- Adding edge strategy, which is used to add the selected $e_{i,j}$ into solution
- Metropolis rule of SA algorithm is used to decide whether to accept or discard a selected edge $e_{i,j}$.

Canonical ACO algorithms use a blind accepting strategy which accept the selected edge $e_{i,j}$ unconditionally. This blind accepting is one of the reasons which deteriorate the intensification ability and convergence speed of ACO algorithms. In OG-ACO algorithm, because each ant has a complete solution, it can use it to calculate the difference of solution between current solution and the new solution gained by adding the selected edge $e_{i,j}$ and then metropolis rule of SA algorithm can be used to decide whether to accept edge $e_{i,j}$ or discard it. Suppose the new solution after accepting $e_{i,j}$ is π' and the difference of solutions is:

$$\Delta = f(\pi') - f(\pi) \tag{6}$$

Suppose the current temperature is t and rand ϵ[0, 1] is random. Using metropolis rule, the new solution π will be:

$$\pi = \begin{cases} \pi' \ if \ \Delta < 0 \\ \pi' \ if \ rand < \exp(-\Delta / t) \\ \pi \quad otherwise \end{cases} \tag{7}$$

In order to use metropolis rule, we must set the two parameters, initial temperature t_0 and cooling coefficient λ. In order to balance solution quality and convergence speed, the accepting probability p_0 in the beginning must be high enough and the accepting probability p_{end} in the end must be low enough. Using p_0 and p_{end} as two parameters, we use sampling based strategy to calculate the t_0 and λ. First, we generate a solution randomly and then generate a number of perturbations randomly. Suppose the average absolute difference among solutions is Δ_{avr}, the probability p to accept a new solution is:

$$p = e^{-\Delta_{avr} / t_0} \tag{8}$$

Suppose the parameter p in the beginning is p_0, we have:

$$t_0 = {}^{-\Delta_{avr}} \Big/ \ln(p_0) \tag{9}$$

Although the more slowly the temperature is lowered, the better the solution is. But the theoretical optimal cooling schedule requires too much computational time in practice. Therefore, we use the widely used cooling scheme:

$$t_{k+1} = \lambda t_k \tag{10}$$

As t_0, λ is decided by parameter p_{end}. Suppose the total iteration times is k, then the temperature t_{end} at the end is:

$$t_{end} = \lambda^k t_0 \tag{11}$$

Suppose we hope the accepting probability at the end of iteration is p_{end}, then we have:

$$p_{end} = e^{-\Delta_{avr} / t_{end}} = e^{-\Delta_{avr} / (\lambda^k t_o)} \tag{12}$$

Then the cooling coefficient λ can be calculated by:

$$\lambda = \left({}^{-\Delta_{avr}} \Big/ (\ln(p_{end}) t_0) \right)^{1/k} \tag{13}$$

- Seven different adding edge strategies are used in our algorithm. We use three operators to define the seven adding edge strategies.

Define 1: Inverse operator *inverse* (π, *i*, *j*). It means to inverse the cities between position *i* and *j*. The *inverse* (π, *i*, *j*) will generate a new solution π' such that $\pi'(i) = \pi(j)$, $\pi'(i+1) = \pi(j-1)$, … , $\pi'(j) = \pi(i)$, where,,,, $1 \le i, j \le n \wedge 1 \le j - i < n - 1$; in addition, if $j - i = n - 1$, it means $i = 1$ and $j = n$, then $\pi'(i) = \pi(j)$ and $\pi'(j) = \pi(i)$.

Define 2: Insert operator *insert* (π, *i*, *j*). It means to move the city in position *j* to position *i*. The *insert* (π, *i*, *j*) operator will generate a new solution π' such that $\pi'(i) = \pi(j)$, $\pi'(i+1) = \pi(i)$, … , $\pi'(j) = \pi(j-1)$, in the case of $i<j$; or $\pi'(j) = \pi'(j+1)$, … , $\pi'(i-1) = \pi(i)$, $\pi'(i) = \pi(j)$, in the case of $i>j$.

Define 3: Swap operator *swap* (π, *i*, *j*). It means to swap the city in position *j* and city in position *i*. The *swap* (π, *i*, *j*) operator will generate a new solution π' such that $\pi'(i) = \pi(j)$ and $\pi'(j) = \pi(i)$.

Using the above three operators, we define the seven adding edge strategies to add edge $e_{i,j}$ into solution π:

- Inverse strategy (*iv*)
  $\pi' = inverse(\pi, i-1, j)$
- Insert strategy (*is*)
  $\pi' = insert(\pi, i-1, j)$
- Swap strategy (*sw*)
  $\pi' = swap(\pi, i-1, j)$
- Inverse and insert strategy (*iv* + *is*)
  $\pi' = min(inverse(\pi, i-1, j), insert(\pi, i-1, j))$
- Inverse and swap strategy (*iv* + *sw*)
  $\pi' = min(inverse(\pi, i-1, j), swap(\pi, i-1, j))$
- Insert and swap strategy (*is* + *sw*)
  $\pi' = min(insert(\pi, i-1, j), swap(\pi, i-1, j))$
- Inverse, insert and swap strategy (*iv* + *is* + *sw*)

$\pi' = min(inverse(\pi, i\text{-}1, j), insert(\pi, i\text{-}1, j), swap(\pi, i\text{-}1, j))$

Method to improve diversity: Compare with the blindly accepting strategy used by canonical ACO algorithms, the metropolis rule used by OG-ACO will speed up its convergence speed. In order to prevent premature convergence, we use the MMAS as basic algorithm framework, so each edge will have a chance to be selected. Furthermore, if the solution doesn change in a generation, we will perturb the solution using the above defined three operators randomly.

The main idea of MMAS is to combine an improved exploitation of the best solutions found during the search with an effective mechanism for avoiding early search stagnation. To achieve this goal, it uses three strategies:

- To exploit the best solutions found during an iteration or during the run of the algorithm, after each iteration only one single ant adds pheromone. This ant may be the one which found the best solution in the current iteration (iteration-best ant) or the one which found the best solution from the beginning of the trial (global-best ant)
- To avoid stagnation of the search the range of possible pheromone trails on each solution component is limited to an interval $[\tau_{min}, \tau_{max}]$
- To smooth pheromone trail. When MMAS has converged or is very close to convergence, it increases the pheromone trails proportionally to their difference to the maximum pheromone limit.

So, after updating $\tau_{i,j}$ using Eq. (3), MMAS will adjust $\tau_{i,j}$ as following:

$$\tau_{i,j}(t)=\begin{cases}\tau_{\min} \; if \; \tau_{i,j}(t) < \tau_{\min} \\ \tau_{\max} \; if \; \tau_{i,j}(t) > \tau_{\max} \\ \tau_{i,j}(t) \;\; otherwise\end{cases} \quad (14)$$

The Pheromone Trail Smooth (PTS) strategy is:

$$\tau_{i,j} = \tau_{i,j} + \delta\left(\tau_{\max} - \tau_{i,j}\right), with\, 0 < \delta < 1 \quad (15)$$

Descript ion of OG-ACO algorithm: Following is the pseudo code of OG-ACO algorithm. The main difference of OG-ACO algorithm and canonical ACO algorithms lies in line 10 and 13 to line 15. In line 10, metropolis rule of SA algorithm is used to decide whether to accept the selected city v_j or not. From line 13 to 15, a perturbation, which is produced by one of the three operators defined in last subsection, is added to the current ant solution in case no change happened in the complete construction process. To do so, an operator and two positions *i* and *j* will be selected randomly first, then the corresponding operator and parameters are used on the solution to produce a new perturbed solution.

Algorithm OG-ACO:

- Initialize $\tau_{i,j}$, $\eta_{i,j}$ and other parameters
- Construct solution π for each ant
- Calculate t_0 and λ using Eq. (9) and (13)
- While end condition is not met, repeat
- For each ant, repeat
- Initialize its inner state, tabu list and current city v_i
- While construction is not completed, repeat
- Calculate $p_{i,j}$ using Eq. (2)
- Select next visiting city v_j using $p_{i,j}$
- Update current solution using metroplis rule
- Update ant inner state and tabu list
- End While
- If the solution of ant has not changed
- Perturb it using a randomly selected operator
- End If
- End For
- Update each $\tau_{i,j}$ using Eq. (3), (4), (5) and (14)
- Smooth pheromone trail using (15) if necessary
- Decrease temperature using Eq. (10)
- End While

SIMULATION AND RESULTS

In order to observe and analyze the performance of OG-ACO algorithm, two experiments were carried on typical TSP problems. The first experiment was used to analyze the effect of the seven adding edge strategies and the second was carried to compare performance with other ACO algorithms. Suppose the number of ant is *M*and the city number is *n*, we use the following parameters: $M = n$, $\alpha = 1$, $\beta = 5$, $\rho = 0.01$, $Q = 100*M$, $\delta = 0.01$, $p_0 = 0.9$, $p_{end} = 10^{-100}$, $\tau_{min} = 1/(\bar{d}_{i,j} \times n)$ and $\tau_{max} = 1/\bar{d}_{i,j}$, where, $\bar{d}_{i,j}$ is the average distance off all paths in the TSP problem. We use pheromone trail smooth strategy once in 50 iterations.

The effect of different adding edge strategies: For symmetric TSP problems, we run our algorithm with different adding edge strategies on chn31 and chn144 problem respectively. The chn31 problem, whose best known integer solution is 15381, is consisted of 31 main cities of china. The chn144 problem, whose best known integer solution is 30348, is consisted of 144 cities of china. We repeat the experiments 100 times and the iteration times are 200 for chn31 and 500 for chn144. We compare the best solutions, average solutions, worst solutions, Standard Differences (SD) and the times to reach the best known solution of different strategies.

Table 1: Simuation results of different adding edge strategy on chn31 problem

Strategy	Best	Worst	Average	Times	SD
iv	15381	15381	153.81	100	0.00
is	15381	15556	15408.87	41	39.33
sw	15602	16681	16132.92	0	248.51
iv+is	15381	15381	15381.00	100	0.00
iv+sw	15381	15455	15389.44	65	16.74
is+sw	15381	15612	15389.74	20	52.01
iv+is+sw	15381	15381	15381.00	100	0.00

Table 2: Simuation results of different adding edge strategy on chn144 problem

Strategy	Best	Worst	Average	Times	SD
iv	30348	30389	30355.38	36	9.83
is	31425	32389	31830.96	0	224.13
sw	35259	39362	37098.13	0	691.31
iv+is	30348	30349	30348.17	83	0.38
iv+sw	30379	30610	30472.57	0	58.15
is+sw	31380	32388	31907.90	0	218.08
iv+is+sw	30348	30381	30354.361	11	0.92

Table 1 is the simulation results on chn31 problem. It shows that among the *iv*, *is* and *sw* strategies, *iv* is far better than *is* and *sw*, it can reach the best known solution always and *sw* is extremely bad, it can never find the best known solution. Among the four combined strategies, both *iv* + *is* and *iv* + *is* + *sw* can find the best known solution always. For chn31 problem, which has only 31 cities, there is no significant difference among *iv*, *iv* + *is* and *iv* + *is* + *sw*.

In order to compare the performance of *iv*, *iv* + *is* and *iv* + *is* + *sw*, we run our algorithm on chn144 problem, which is much more difficult than chn31 problem. Table 2 is the simulation results on chn144 problem. From Table 2, *iv* + *is* is better than *iv* and *iv* + *is* + *sw* in terms of average solution and times to reach the best known solution. There is no significant difference between *iv* and *iv* + *is* + *sw* in terms of average solution, but *iv* is better than *iv* + *is* + *sw* in terms of times to reach the best known solution. Table 1 and 2 show that *iv* + *is* is best for symmetric TSP problem.

For asymmetric TSP problems, we run our algorithm with different adding edge strategies on kro124p problem from TSPLIB. The kro124p problem, whose best known integer solution is 36230, is consisted of 100 cities. We repeat the experiments 100 times and the iteration times are 200. Because asymmetric problem is much difficult than symmetric problem, we let each ant construct solution 2 times in each temperature. We compare the best solutions, average solutions, worst solutions, Standard Differences (SD) and the times to reach the best known solution of different strategies.

Table 3 is the simulation results on kro124p problem. It shows those strategies with *is* strategy are better than those without it and *iv* + *is* + *sw* strategy has the best results. Table 1 and 2 show that those strategies with *iv* strategy are better than those without it on symmetric problem. For symmetric problem, *iv* strategy changes the solution least, it changes two edges only. But for asymmetric problem, *is* strategy changes the solution least, it changes three edges only. That is because the less changes a strategy produce, the more finely it can search the solution space, so it can have better performance.

Table 3: Simuation results of different adding edge strategy on kro124p problem

Strategy	Best	Worst	Average	Times	SD
iv	37196	40474	39341.98	0	656.90
is	36894	38595	37882.72	0	404.39
sw	39252	43232	42033.92	0	766.90
iv+is	36436	37797	36968.09	0	239.36
iv+sw	36763	39447	38474.73	0	538.65
is+sw	36676	38485	37429.10	0	355.89
iv+is+sw	36435	37412	36918.26	0	230.88

Table 4: Comparison with other algorithms on eil51 problem

Algorithm	Best	Average	SD	Iteration times
OG-ACO	426	426.00	0.00	200
cAS	426	426.20	0.50	10000
ACS	426	428.06	2.48	10000
MMAS+PTS	426	427.10	-	10000
FGACO	-	426.80	-	10000

Table 5: Comparison with other algorithms on ry48p problem

Algorithm	Best	Average	SD	Iteration times
OG-ACO	14422	14448.60	30.25	400
cAS	14422	14465.40	34.90	20000
ACS	14422	14565.45	115.23	20000
MMAS+PTS	14422	14523.40	-	20000
FGACO	-	14498.20	-	20000

Compare with other ACO algorithm: We compare OG-ACO algorithm with ACS, MMAS, cAS and FGACO on symmetric eil51problem and asymmetric ry48p problem. For eil51problem, OG-ASO algorithm uses *iv* + *is* strategy. For ry48pproblem, OG-ACO algorithm uses *iv* + *is* + *sw* strategy. The number of ants is set to city number and we repeat the experiment 25 times.

Table 4 is the simulation results on eil51 problem. In the table, means the original literature had not corresponding data. MMAS+PTS means MMAS with pheromone smooth strategy. Table 4 shows that even though the maximum iteration times of OG-ACO are only 200, which are far less than the maximum iteration times of other algorithms, OG-ACO can find the best known solution always. It means OG-ACO can reach better solution in less generation than other ACO algorithms on symmetric problem.

Table 5 is the simulation results on ry48 problem. Table 5 shows that even though the maximum iteration times of OG-ACO are only 400, which are far less than the maximum iteration times of other algorithms, OG-ACO has best average tour length. It means OG-ACO can reach better solution in less generation than other ACO algorithms on asymmetric problem also.

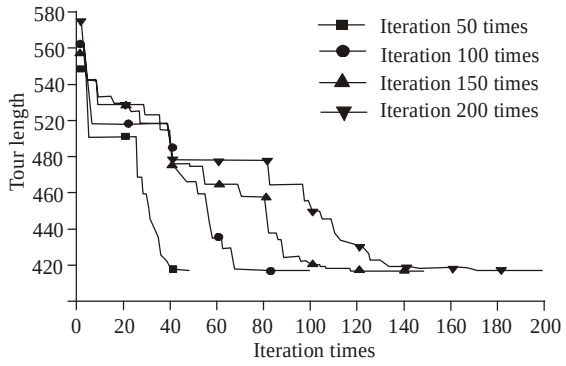

Fig. 1: The iteration process on eil51 problem

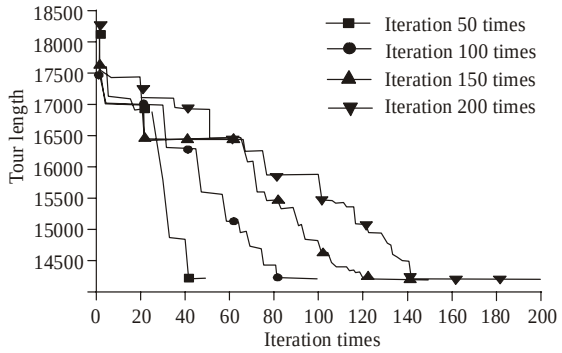

Fig. 2: The iteration process on eil51 problem

Iteration process with equal generations: In order to observe the convergence process of OG-ACO algorithms with different iteration times, we draw the typical iteration process of best solution when the maximum iteration times are 50, 100, 150 and 200 respectively. Figure 1 is the graph of best solution for each generation in one run on eil51 problem and Fig. 2 is the graph of best solution for each generation in one run on ry48 problem. Those graphs show clearly that OG-ACO converge quickly when the maximum iteration times is low. As the maximum iteration times increase, the convergence speed will slow down and it is less likely to trap in local minimum. The strategy we used to set the cooling coefficient, which adjust the cooling coefficient according the maximum iteration times allowed, can control the search process elegantly.

CONCLUSION

This study presents a new way to improve the performance of ACO algorithm. It enhances the short-term memory of ant to store a complete solution always and combines the idea of SA algorithm to improve ant individual intelligence. Specifically, when ant selects a component probabilistically, metropolis rule is used to decide whether to accept it or discard it. Simulation results show this strategy can improve the performance of ACO algorithm effectively.

ACKNOWLEDGMENT

This study was supported by Nature Science Foundation of Fujian Province of P. R. China (No. 2009J05043, No. 2011J05044 and No. 2008J0316).

REFERENCES

Blum, C. and A. Roli, 2003. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Comput. Surv., 35(3): 68-308.

Bullnheimer, B., R.F. Hartl and C. Strauss, 1999. A new rank based version of the ant system: A computational study. Cent. Eur. J. Operat. Res. Econ., 7(1): 25-38.

Chang, S.G., B. Yu and M. Vetterli, 1997. Bridging compression to wavelet thresholding as a denoising method. Proceeding of Conference of Information Sciences Systems, Baltimore, MD, pp: 568-573.

Chen, L., S. Jie, Q. Ling, C. Hong-jian, L. Chen, J. Shen and L. Qin, 2003. An adaptive ant colony algorithm based on equilibrium of distribution. J. Softw., 14(6): 1418-1425.

Colorni, A., M. Dorigo and V. Maniezzo, 1992a. Distributed optimization by ant colonies. Proceedings of the 1st European Conference on Artificial Life, Elsevier Publishing, Paris, France, pp: 34-142.

Colorni, A., M. Dorigo and V. Maniezzo, 1992b. An investigation of some properties of an ant algorithm. Proceedings of the Parallel Problem Solving from Nature Conference, Elsevier Publishing, Brussels, Belgium, pp: 509-520.

Dorigo, M., V. Maniezzo and A. Colorni, 1996. The ant system: Optimization by a colony of cooperating agents. IEEE T. Syst. Man Cyb., 6(1): 29-41.

Marco, D. and S. Thomas, 2010. Ant colony optimization: Overview and recent advances. Handb. Metaheurist Int. Series Oper. Res. Manage. Sci., 146: 227-263.

Stützle, T. and H.H. Hoos, 2000. Max-min ant system. Future Gener. Comp. Sy., 16: 889-914.

Taillard, E.D., L.M. Gambardella, M. Gendreau and J.Y. Potvinc, 2001. Adaptive memory programming: A unified view of metaheuristics. Eur. J. Oper. Res., 135(1): 1-16.

Wiesemann, C., S. Ude-koeller, G.H. Sinnecker and U. Thyen, 2010. Ethical principles and recommendations for the medical management of differences of sex development (DSD)/intersex in children and adolescents. Eur. J. Pediatr., 169: 671-679.

Liangjun K., F. Yuanjing and Y. Li, 2008. Finite grade pheromone ant colony optimization for image segmentation. Optoelectron. Rev., 16(2): 163-171.

Zhu, Q.B. and Z.J. Yang, 2004. An ant colony optimization algorithm based on mutation and dynamic pheromone updating. J. Softw., 15(2): 185-192.