A Novel Ant Colony Genetic Hybrid Algorithm

Shang Gao
School of Computer Science and Technology, Jiangsu University of Science and Technology, Zhenjiang 212003, China
Email: gao_shang@hotmail.com

Zaiyue Zhang and Cungen Cao
Key Laboratory of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080, China
Email: yzzjzzy@sina.com, cgcao@ict.ac.cn

***Abstract*—By use of the properties of ant colony algorithm and genetic algorithm, a novel ant colony genetic hybrid algorithm, whose framework of hybrid algorithm is genetic algorithm, is proposed to solve the traveling salesman problems. The selection operator is an artificial version of natural selection, and chromosomes with better length of tour have higher probabilities of being selected in the next generation. Based on the properties of pheromone in ant colony algorithm the ant colony crossover operation is given. Four mutation strategies are put forward using the characteristic of traveling salesman problems. The hybrid algorithm with 2-opt local search can effectively find better minimum beyond premature convergence. Ants choose several tours based on trail, and these tours will replace the worse solution. Compare with the simulated annealing algorithm, the standard genetic algorithm and the standard ant colony algorithm, all the 4 hybrid algorithms are proved effective. Especially the hybrid algorithm with strategy D is a simple and effective better algorithm than others.**

***Index Terms*—ant colony algorithm, genetic algorithm, traveling salesman problem**

I. INTRODUCTION

Inspired by the behavior of real ants, Marco Dorigo first introduced the colony optimization approach in his Ph.D. thesis in 1992 and expanded it in his further work. The characteristics of artificial ant colony include a method to construct solutions that balances pheromone trails and a problem-specific heuristic, a method to both reinforce and evaporate pheromone, and local search to improve the constructed solutions. The ACO[1] methods have been successfully applied to diverse combinatorial optimization problems including traveling salesman, quadratic assignment, vehicle routing[2], telecommunication networks[3], graph coloring, constraint satisfaction, Hamitonian graphs and scheduling. Genetic algorithms (GAs) or more generally, evolutionary algorithms [4] have been touted as a class of general-purpose search strategies for optimization problems. GAs use a population of solutions, from which, using crossover, mutation and selection strategies, better and better solutions can be produced. GAs can handle any kind of objective functions and any kind of constraints without much mathematical requirements about the optimization problems, and have been widely used as search algorithms in various applications. Various GAs have been proposed in the literature [5,6] and shown superior performances over other methods. As a consequence, GAs seemed to be nice approaches for solving TSP. However, GAs may cause certain degeneracy in search performance if their operators are not carefully designed [6]. A genetic algorithm (GA) is a metaheuristic inspired by the efficiency of natural selection in biological evolution. Genetic algorithms have been applied successfully to a wide variety of combinatorial optimization problems and are the subject of numerous recent books [7-8] and conference proceedings. Unlike traditional heuristics (and some metaheuristics like tabu search) that generate a single solution and work hard to improve it, GAs maintain a large number of solutions and perform comparatively little work on each one. Several researchers (see [9] and the references contained within) have implemented GAs for the standard TSP, with mixed results. The GA in [9] found new best solutions for some well studied benchmark problems. Recently, there are many search activities over artificial ants, which are agents with the capability of mimicking the behavior of real ants [10,11]. The agents are sufficiently intelligent to exploit pheromone information that has been left on the traversed ground. Agents can then choose a route according to the amount of pheromone. The larger amount of pheromone is on a route, the larger is the probability of selecting the route by agents. With such concept, a population-based algorithm, Ant colony optimization (ACO), has been widely used as a new cooperative search algorithm [10]. In this paper, a novel algorithm of ant colony genetic algorithm for traveling salesman problem is proposed.

II. THE BASIC ACO ALGORITHM

In this section we introduce the basic ACO algorithm. We decided to use the well-known traveling

Supported by the National Natural Science Foundation of China under Grant No.60773059; National Basic Research Program of Jiangsu Province University (08KJB520003)

doi:10.4304/jsw.5.11.1179-1186

salesman problem as benchmark, in order to make the comparison with other heuristic approaches easier. Given a set of n towns, the TSP can be stated as the problem of finding a minimal length closed tour that visits each town once. We call d_{ij} the length of the path between towns i and j. In the case of Euclidean TSP, d_{ij} is the Euclidean distance between i and j (i.e., $d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. An instance of the TSP is given by a graph (N, E), where N is the set of towns and E is the set of edges between towns (a fully connected graph in the Euclidean TSP).

Let $b_i(t)$ ($i = 1,2,\cdots,n$) be the number of ants in town i at time t and let $m = \sum_{i=1}^{n} b_i(t)$ be the total number of ants. Each ant is a simple agent with the following characteristics:

(1) It chooses the town to go to with a probability that is a function of the town distance and of the amount of trail present on the connecting edge.

(2) To force the ant to make legal tours, transitions to already visited towns are disallowed until a tour is completed (this is controlled by a *tabu* list).

(3) When it completes a tour, it lays a substance called trail on each edge (i, j) visited.

Let $\tau_{ij}(t)$ be the intensity of trail on edge (i, j) at time t. Each ant at time t chooses the next town, where it will be at time $t+1$. Therefore, if we call an iteration of the ACO algorithm the m moves carried out by the m ants in the interval $(t, t+1)$, then every n iterations of the algorithm (which we call a cycle) each ant has completed a tour. At this point the trail intensity is updated according to the following formula

$$\tau_{ij}(t+n) = \rho\tau_{ij}(t) + \Delta\tau_{ij} \tag{1}$$

where ρ is a coefficient such that (1- ρ) represents the evaporation of trail between time t and $t+n$,

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{2}$$

where $\Delta\tau_{ij}^{k}$ is the quantity per unit of length of trail substance (pheromone in real ants) laid on edge (i, j) by the k-th ant between time t and $t+n$. It is given by

$$\Delta\tau_{ij}^{k} = \begin{cases} \dfrac{Q}{L_k} & \text{if k - th ant uses edge (i, j) in its tour} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where Q is a constant and L_k is the tour length of the k-th ant. The coefficient ρ must be set to a value ρ <1 to avoid unlimited accumulation of trail. In our experiments, we set the intensity of trail at time 0, $\tau_{ij}(0)$, to a small positive constant *c*.

In order to satisfy the constraint that an ant visits all the n different towns, we associate with each ant a data structure called the *tabu list*, that saves the towns already visited up to time t and forbids the ant to visit them again before n iterations (a tour) have been completed. When a tour is completed, the *tabu* list is used to compute the ant's current solution (i.e., the distance of the path followed by the ant). The *tabu* list is then emptied and the ant is free again to choose. We define $tabu_k$ the dynamically growing vector, which contains the *tabu* list of the kth ant, $tabu_k$ the set obtained from the elements of $tabu_k$, and $tabu_k(s)$ the s-th element of the list (i.e., the s-th town visited by the k-th ant in the current tour).

We call *visibility* η_{ij} the quantity $1/d_{ij}$. This quantity is not modified during the run of the ACO algorithm, as opposed to the trail, which instead changes according to the previous formula (1).

We define the transition probability from town i to town j for the k-th ant as

$$p_{ij}^{k}(t) = \begin{cases} \dfrac{\tau_{ij}^{\alpha}(t) \cdot \eta_{ij}^{\beta}}{\sum_{s \in allowed_k} \tau_{is}^{\alpha}(t) \cdot \eta_{is}^{\beta}} & if \ \ j \in allowed_k \\ 0 & otherwise \end{cases} \tag{4}$$

where $allowed_k$ = { N - $tabu_k$ } and where α and β are parameters that control the relative importance of trail versus visibility. Therefore the transition probability is a trade-off between visibility (which says that close towns should be chosen with high probability, thus implementing a greedy constructive heuristic) and trail intensity at time t (that says that if on edge (i, j) there has been a lot of traffic then it is highly desirable, thus implementing the autocatalytic process).

Given the definitions of the preceding section, the so-called ant-cycle algorithm is simply stated as follows. Formally the ant-cycle algorithm is:

1. Initialize:
   Set t:=0 {t is the time counter}
   Set NC:=0 {NC is the cycles counter}
   For every edge (i,j) set an initial value $\tau_{ij} = c$ for trail intensity and $\Delta\tau_{ij} = 0$
   Place the m ants on the n nodes
2. Set s:=1 {s is the tabu list index}
   For k:=1 to m do
   Place the starting town of the k-th ant in $tabu_k(s)$
3. Repeat until tabu list is full {this step will be repeated (n-1) times}
   Set s:=s+1
   For k:=1 to m do
   Choose the town j to move to, with probability $p_{ij}^{k}(t)$ given by equation (4) {at time t the k-th ant is on town $i = tabu_k(s-1)$ }

Move the k-th ant to the town j

Insert town j in $tabu_k(s)$

4. For k:=1 to m do

Move the k-th ant from $tabu_k(n)$ to $tabu_k(1)$

Compute the length L_k of the tour described by the k-th ant

Update the shortest tour found

For every edge (i,j)

For k:=1 to m do

$$\Delta\tau_{ij}^{k} = \begin{cases} \dfrac{Q}{L_k} & \text{if k - th ant uses edge (i, j) in its tour} \\ 0 & \text{otherwise} \end{cases}$$

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$$

5. For every edge (i,j) compute $\tau_{ij}(t+n)$ according to equation $\tau_{ij}(t+n) = \rho\tau_{ij}(t) + \Delta\tau_{ij}$

Set t:=t+n

Set NC:=NC+1

For every edge (i,j) set $\Delta\tau_{ij} := 0$

6. If (NC < NC_{MAX}) and (not stagnation behavior)

then

Empty all tabu lists

Goto step 2

else

Print shortest tour

Stop

The complexity of the *ant-cycle* algorithm is $O(NC \cdot n^2 \cdot m)$ if we stop the algorithm after NC cycles. In fact step 1 is $O(n^2 + m)$, step 2 is $O(m)$, step 3 is $O(n^2 \cdot m)$, step 4 is $O(n^2 \cdot m)$, step 5 is $O(n^2)$, step 6 is $O(n \cdot m)$. Since we have experimentally found a linear relation between the number of towns and the best number of ants, the complexity of the algorithm is $O(NC \cdot n^3)$.

We also experimented with two other algorithms of the AS, which we called *ant-density* and *ant-quantity* algorithms. They differ in the way the trail is updated. In these two models each ant lays its trail at each step, without waiting for the end of the tour. In the *ant-density* model a quantity Q of trail is left on edge (i,j) every time an ant goes from i to j; in the *ant quantity* model an ant going from i to j leaves a quantity Q/d_{ij} of trail on edge (i,j) every time it goes from i to j. Therefore, in the *ant-density* model we have

$$\Delta\tau_{ij}^{k} = \begin{cases} Q & \text{if k - th ant uses edge (i, j) in its tour} \\ 0 & \text{otherwise} \end{cases}$$

and in the *ant-quantity* model we have

$$\Delta\tau_{ij}^{k} = \begin{cases} Q/d_{ij} & \text{if k - th ant uses edge (i, j) in its tour} \\ 0 & \text{otherwise} \end{cases}$$

From these definitions it is clear that the increase in trail on edge (i,j) when an ant goes from i to j is independent of dij in the *ant-density* model, while it is inversely proportional to dij in the *ant-quantity* model (i.e., shorter edges are made more desirable by ants in the *ant-quantity* model).

III. THE BASIC GENETIC ALGORITHM

The evolutionary theory attributes the process of the natural evolution of populations to the Darwin`s principle of natural selection "survival of the fittest". Genetic Algorithms (GA) were developed by Holland (1975), and are based on the principles of natural selection and genetic modification. GA are optimization methods, which operate on a population of points, designated as individuals. Each individual of the population represents a possible solution of the optimization problem. Individuals are evaluated depending upon their fitness. The fitness indicates how well an individual of the population solves the optimization problem.

GA begin with random initialization of the population. The transition of a population to the next takes place via the application of the genetic operators: Selection, crossover, and mutation. Through the selection process, the fittest individuals will be chosen to go to the next population. Crossover exchanges the genetic material of two individuals creating two new individuals. Mutation arbitrarily changes the genetic material of an individual. The application of the genetic operators upon the individuals of the population continues until a sufficiently good solution of the optimization problem is found. The solution is usually achieved when a pre-defined stop condition, i.e., a certain number of generations is reached. GA has the following general features:

(1) GA operates with a population of possible solutions (individuals) instead of a single individual. Thus, the search is carried out in a parallel form.

(2) GA is able to find optimal or sub-optimal solutions in complex and large search spaces. Moreover, GA are applicable to nonlinear optimization problems with constraints, that can be defined in discrete or continuous search spaces.

(3) GA examines many possible solutions at the same time. So there is a higher probability that the search converges to an optimal solution.

In the classical GA developed by Holland (1975), the individuals are represented by binary numbers, i.e., bit strings. In the meantime, new representations for individuals and appropriate genetic operators have been developed. For optimization problems with variables within the continuous domain the real representation has shown to be more suitable. With this type of the representation, individuals are represented directly as real numbers. For this case, it is no necessary to transform real numbers into binary. In the following, some terms and definitions are described.

After several generations, GA can converge to the best solution. Let P(t) and C(t) are parents and offspring in generation t. A usual form of general GA is shown in the following:

Procedure: General GA
Begin
t ← 0;
Initialize P(t); {Generate random population of n chromosomes (suitable solutions for the problem)}
Evaluate P(t); {Evaluate the fitness f(x) of each chromosome x in the population}
While (not match the termination conditions) do
Recombine P(t) to yield C(t);
[Selection] Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)
[Crossover] With a crossover probability cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.
[Mutation] With a mutation probability mutate new offspring at each locus (position in chromosome).
[Accepting] Place new offspring in a new population
[Replace] Use new generated population for a further run of algorithm
Evaluate C(t);
Select P(t+1) form P(t) and C(t);
t ← t+1;
End;
End;

Recently, genetic algorithms with local search have also been considered as good alternatives for solving optimization problems. The local search for TSP, 2-opt approach, can be implemented after crossover and mutation operators.

IV. ANT COLONY GENTIC ALGORITHM

A. Ant Colony Trail Update

After all ants have constructed a solution, we can update pheromone trail according to populations. The pheromone-updating rule is performed as

$$\tau(r,s) = \rho\tau(r,s) + \Delta\tau(r,s) \qquad (5)$$

Where $\tau(r,s)$ is pheromone trail between city r and s, $\Delta\tau(r,s) = \begin{cases} Q/L_{gb} & \text{if} \quad (r,s) \in \text{best} \quad \text{tour} \\ 0 & \text{otherwise} \end{cases}$, and L_{gb} is the optimal tour length for the TSP problem.

To avoid search stagnation, the allowed range of the pheromone trail strengths is limited to the interval $[\tau_{\min}, \tau_{\max}]$, that is, $\forall \tau_{ij}, \tau_{\min} \le \tau_{ij} \le \tau_{\max}$.

B. Selection Operators

This operator is designed by a common method of natural selection in GA called the Roulette Wheel method. The Roulette Wheel method simply chooses the strings in a statistical fashion based solely upon their relative (i.e., percentage) cost or fitness values. So, the natural selection operator in this GA randomly chooses strings from the current population with probability inversely proportional to their cost.

The i th chromosome is selected based on the probability

$$P_i = \frac{f_i}{\sum_{i=1}^{n} f_i} \qquad (6)$$

where f_i is fitness of i th chromosome. f_i is the reciprocal of length of tour. The shorter the route the higher the fitness value is.

C. Ant Colony Crossover Operators

There are many different types of crossover operators, but we discuss ant colony crossover operator as following. Let's suppose we have two parent tours given by

P1=1 2 3 4 5 6 7 8
P2=1 2 8 3 4 5 7.6

They are shown in figure 1. The coarse linear represent that the pheromone trail of these two city is higher. We can sort the 8 sides pheromone trail of two parent tours. We note the coarse linear from the maximum to k th ($k = 5$) maximum according the pheromone trail. For example, the pheromone trail of e_{12}、e_{23}、e_{34}、e_{45}, e_{67} in P_1 are higher, and e_{12}、e_{34}、e_{45}、e_{16}, e_{67} in P_2 are higher. Then we use these sides make up a new offspring solution. We can choose the side which pheromone trail is the higher, as the solution is illegal. And we choose the other side by nearest rule. The process is shown in figure2.

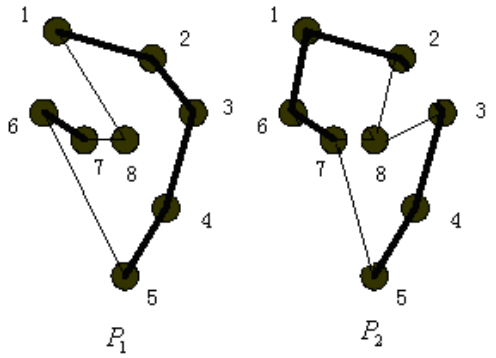

Figure 1. Two parent solutions

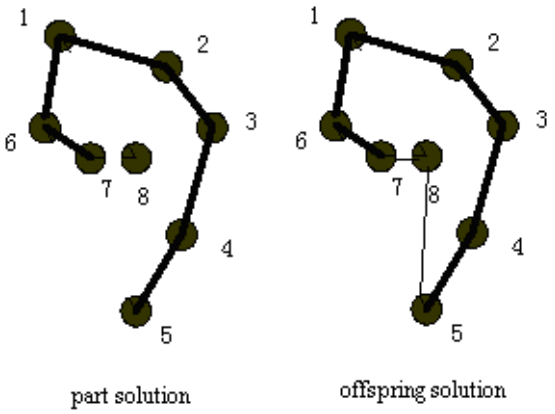

Figure2. The part solution and offspring solution

D. Mutation Operators

There are following methods to generate a new tour C_1 from the tour C_0.

Mutation operator A Choose two cities j_1 and j_2 from the tour C_0 by randomly, and then swap j_1 with j_2 in the tour C_0, so the new tour is C_1. For example, suppose C_0 =2 3 4 1 5 7 9 8 6, $j_1 = 3$ and $j_2 = 9$, so C_1=2 9 4 1 5 7 3 8 6.

Mutation operator B Choose a city j_1 from the tour C_0 by randomly, and then swap j_1 with the next visited city. For example, suppose C_0 =2 3 4 1 5 7 9 8 6, $j_1 = 3$, so C_1=2 4 3 1 5 7 9 8 6.

Mutation operator C A modified solution C_1 is generated from C_0 by randomly choose two cities j_1 and j_2 and reversing the sequence in which the cities in between cities j_1 and j_2 are traversed, i.e. the 2-change generation mechanism. For example, suppose C_0 =2 3 4 1 5 7 9 8 6, $j_1 = 3$ and $j_2 = 9$, so C_1=2 9 7 5 1 4 3 8 6.

Mutation operator D Choose two cities j_1 and j_2 from the tour C_0 by randomly, and then insert city j_1 into the latter of j_2 city. For example, suppose C_0 =2 3 4 1 5 7 9 8 6, $j_1 = 3$ and $j_2 = 9$, so C_1=2 4 1 5 7 9 3 8 6.

E. 2-Option Llocal Search

Each individual has a 2-option local search minimization applied to their tour. The 2-option local search is not the most effective local search method. A 3-option local search or the "champion" LK search has been proven much more effective and efficient. While this may be true, they also require much more computation time and perhaps parallel processing to obtain an adequate number of generations in a reasonable timeframe. For the city sets examine here a 2-option local search is sufficient, and for larger city sets employing a LK search will continue the success of the algorithm. The framework for a 2-option local search is shown in Figure 3.

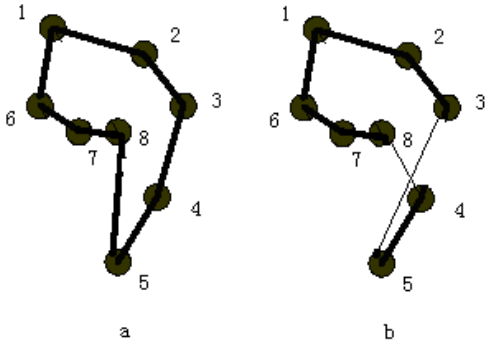

Figure3. 2-option local search

The local search takes an initial solution and makes incremental modifications in order to find a better tour. The operation makes 2 breaks in the tour, recombines in the only possible other option for a complete tour, and then compares the new tour distance to the previous tour distance. This process results in a local optimum due to small rearrangements which work out short nonoptimized sections in the tour.

F. Ant Colony Gentic Hybrid Algorithm

The ant colony genetic hybrid algorithm solving TSP can be expressed as follows:

1. Initialize:
   Set t:=0 {t is the time counter}
   Set NC:=0 {NC is the cycles counter}
   Generate N tours, and choose the better m tours from these N tours, and pheromone laid on edge of these m better tours.
   Set $\Delta\tau_{ij} = 0$
   Place the m ants on the n nodes
2. Set s:=1 {s is the tabu list index}
   For k:=1 to m do
   Place the starting town of the k-th ant in $tabu_k(s)$
3. Repeat until tabu list is full {this step will be repeated (n-1) times}
   Set s:=s+1
   For k:=1 to m do
   Choose the town j to move to, with probability $p_{ij}^k(t)$ given by equation (4) {at time t the k-th ant is on town $i = tabu_k(s-1)$ }
   Move the k-th ant to the town j
   Insert town j in $tabu_k(s)$
4. For k:=1 to m do
   Move the k-th ant from $tabu_k(n)$ to $tabu_k(1)$
   Compute the length L_k of the tour described by the k-th ant
   Update the shortest tour found
   For every edge (i,j)

For k:=1 to m do

$$\Delta\tau_{ij}^{k} = \begin{cases} \dfrac{Q}{L_k} & \text{if k - th ant uses edge (i, j) in its tour} \\ 0 & \text{otherwise} \end{cases}$$

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$$

5. Selection operators
6. Ant colony crossover operators
7. Mutation operator
8. 2-option local search
9. Ants choose several tours based on trail, and these tours will replace the worse solution
10. Save the current best tour
11. For every edge (i,j) compute $\tau_{ij}(t+n)$ according to equation $\tau_{ij}(t+n) = \rho\tau_{ij}(t) + \Delta\tau_{ij}$

    Set t:=t+n
    Set NC:=NC+1
    For every edge (i,j) set $\Delta\tau_{ij} := 0$
12. If (NC < NC_{MAX}) and (not stagnation behavior)
    then
      Empty all tabu lists
      Goto step 2
    else
      Print shortest tour
      Stop

V..EXPERIMENTAL RESULTS

A. Traveling Salesman Problem

Almost all ACO algorithms have initially been tested on the traveling salesman problem. The traveling salesman problem (TSP) can be represented by a complete graph $G = (N, A)$ with N being the set of nodes, also called cities, and A being the set of arcs fully connecting the nodes. Each arc $(i, j) \in A$ is assigned a value d_{ij} which represents the distance between cities i and j. The TSP then is the problem of finding a shortest closed tour visiting each of the $n = |N|$ nodes of G exactly once. For symmetric TSPs, the distances between the cities are independent of the direction of traversing the arcs, that is, $d_{ij} = d_{ji}$ for every pair of nodes. In the asymmetric TSP(ATSP) at least for one pair of nodes i and j we have $d_{ij} \neq d_{ji}$.All the TSP instances used in the empirical studies presented in this article are taken from the TSPLIB benchmark library accessible at http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/. These instances have been used in many other studies and partly stem from practical applications of the TSP.

The exact algorithms are designed to find the optimal solution to the TSP, that is, the tour of minimal length. They are computationally expensive because they must (implicitly) consider all solutions in order to identify the optimum. These exact algorithms are typically derived from the integer linear programming (ILP) formulation of the TSP

$$\begin{aligned} &\min \sum_{i=1}^{N}\sum_{j=1}^{N} d_{ij}x_{ij} \\ &s.t. \quad \sum_{j=1}^{N} x_{ij} = 1 \qquad (i = 1,2,\cdots,n) \\ &\sum_{i=1}^{N} x_{ij} = 1 \qquad (j = 1,2,\cdots,n) \\ &x_{ij} = 0,1 \end{aligned} \tag{7}$$

where n is the number of vertices, the x_{ij}'s are the decision variables: x_{ij} is set to 1 when arc (i,j) is included in the tour, and 0 otherwise. Branch and bound algorithms are commonly used to find an optimal solution to the TSP, and the above AP-relaxation is useful to generate good lower bounds on the optimal value.

B. Simulated Annealing Algorithms

Simulated annealing (SA) is a Monte Carlo approach to minimizing multivariate functions, and meanwhile a numerical optimization model based on the principles of thermo dynamics, which is motivated by an analogy to annealing in solids. The concept of SA derives from a paper published by Metropolis *et al.* in 1953. The algorithm in this paper simulated the cooling of materials in a heated bath, which is also known as an annealing process. If you heat a solid past melting point and then cool it, the structural properties of the solid will primarily depend on the rate of cooling. If the liquid is cooled slowly enough, the large crystals will be formed. However, if the liquid is cooled quickly (quenched), the crystals will take shape with some imperfections. Metropolis's algorithm simulated the material as a system of particles. The algorithm simulates the cooling process by gradually lowering the temperature of the system until it converges to a steady, frozen state. Simulated annealing (SA) takes advantage of search strategies in which cost-deteriorating neighborhood solution may be accepted to search the optimal solutions. In SA, in addition to better-fitness neighbors are always accepted, worse-fitness neighbors may also be accepted according to a probability that is gradually decreased in the cooling process. With the stochastic nature, SA enables asymptotic convergence to optimal solution and has been widely used for solving optimization problems. In SA, if a modified solution is found to have better fitness than its ancestor then the modified solution is retained and the previous solution is discarded. If the modified solution is found to have less fitness than its ancestor, the modified solution may be still retained with a probability related to the current temperature. As the process continues and the temperature decreases, unsatisfactory solutions are less likely accepted. By using this approach, it is possible for the SA algorithm to move out of local minima, and more likely that good solutions will not be discarded.

The Simulated annealing algorithm can be described as follows:

1. Set the initial temperature $T = 100000$, the final temperature $T_0 = 1$, and annealing velocity $\alpha = 0.9$. Generate a random the tour C_0, and calculate the overall sum of the squared errors f_0.

2. If $T > T_0$, go to Step 3. Otherwise print output C_0 and stop.

3. Generate a new tour C_1 from the tour C_0.

4. Calculate the overall sum of the squared errors f_1, and set $\Delta E = f_1 - f_0$ .If $\Delta E \le 0$, accept the new solution, $C_0 \leftarrow C_1$, $T \leftarrow \alpha T$, go to Step 2. Otherwise if $\exp(-\Delta E / T) > rand(0,1)$, accept the new solution also, $C_0 \leftarrow C_1$, $T \leftarrow \alpha T$, go to Step 2. Else go to Step 3.

C. Computational Results

This section compares the results of simulated annealing algorithm, genetic algorithm, ACO algorithm and hybrid algorithms on traveling salesman problem of Oliver30 problem and att48 problem. The parameters of simulated annealing algorithm are set as follows: the initial temperature $T = 100000$, the final temperature $T_0 = 1$, and annealing velocity $\alpha = 0.99$. The parameters of the genetic algorithm optimization toolbox (GAOT) used to solving TSP are set as follows: the population $N = 30$, the cross probability $P_c = 0.2$, and the mutation probability $P_m = 0.5$. The parameters of the hybrid algorithms are set as follows: $\alpha = 1.5$, $m = 30$, $\beta = 2$, and $\rho = 0.9$. 100 rounds of computer simulation are conducted for each algorithm, and the results are shown in Table 1. The optimal tour of Oliver30 by hybrid algorithm is shown in Fig. 4. The optimal tour of att48 by hybrid algorithm is shown in Fig. 5. All the 4 hybrid algorithms are proved effective. Especially the hybrid algorithm with mutation strategy D is a simple and effective better algorithm than others.

TABLE I.
TESTING RESULT OF ALGORITHMS

Algorithms	Oliver30			att48		
	Average solutions	Best solutions	Worst solutions	Average solutions	Best solutions	Worst solutions
Simulated annealing algorithm	438.5223	424.6918	479.8312	34958	35176	40536
Genetic algorithm	483.4572	467.6844	502.5742	38541	38732	42458
Basic ACO algorithm	450.0346	441.9581	499.9331	35876	36532	42234
Crossover operator + Mutation operator A+2-Opt+ACO	438.9323	424.6257	457.9002	34893	35134	38573
Crossover operator + Mutation operator B+2-Opt+ACO	439.4758	426.1782	465.9797	3553	35300	39357
Crossover operator + Mutation operator C+2-Opt+ACO	435.4134	424.9003	447.3198	34689	35185	37698
Crossover operator + Mutation operator D+2-Opt+ACO	431.4876	423.7406	447.6734	33764	33522	36789

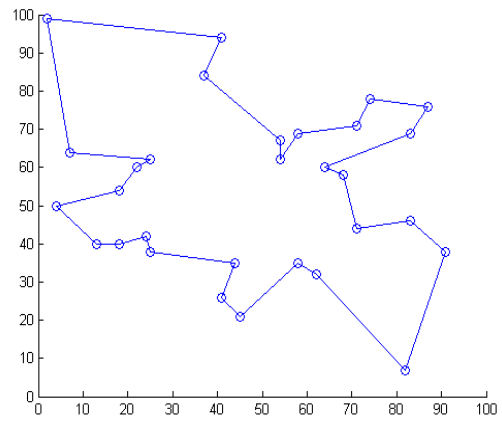

Figure 4 The optimal tour of Oliver30 by hybrid algorithm

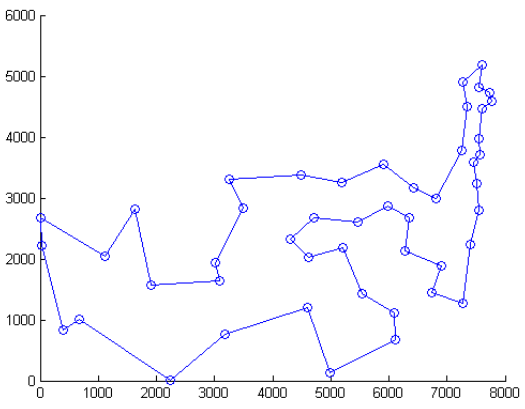

Figure 5 The optimal tour of att48 by hybrid algorithm

V..CONCLUSIONS

In this paper, we presented a novel ant colony genetic hybrid algorithm for traveling salesman problem. It keeps the advantages of ant colony optimization and GAs. From our simulation for those test problems, the proposed algorithm indeed can find the best solutions or optimal solutions. In other words, the proposed algorithm seems to have admirable performance. Experiments for benchmark problems show the hybrid algorithm better than other algorithms.

The following problems need to be considered. 1. The parameters and their affect on the performance of the optimization should be studied in more detail. 2. How to explore hybrid algorithm application to continuous space problem should be investigated. 3. The hybrid algorithm's convergent speed, or the efficiency, should be worth further investigating. 4. How to evaluate the quality of hybrid algorithm and other algorithms is still a problem.

ACKNOWLEDGMENT

This work was partially supported by National Basic Research Program of Jiangsu Province University (08KJB520003) and the National Natural Science Foundation of China under Grant No.60773059.

REFERENCES

[1] A. Colorni, M. Dorigo, and V. Maniezzo, "An investigation of some properties of an ant algorithm", Proc. Of the Parallel Problem Solving from Nature Conference (PPSN'92). Brussels, Belgium: Elsevier Publishing,1992, pp.509-520.

[2] Bernd Bullnheimer, F. Richard. Hartl, and Christine Strauss, "Applying the ant System to the vehicle routing problem". 2nd Metaheuristics International Conference (MIC-97). Sophia-Antipolis, France, 1997, pp.21-24.

[3] Gianni Di Caro, and M. Dorigo, "Mobile agents for adaptive routing", Proceedings of the 31th Hawaii International Conference on system Sciences. Big Island of Hawaii,1998, pp.74-83.

[4] T. Bäck, U. Hammel, and H. P. Schwefel, "Evolutionary computation: Comments on the history and current state," IEEE Trans. On Evolutionary Computation, Vol. 1, No. 1, 1997,pp.3-17.

[5] M. Gen, and R. Cheng, Genetic Algorithms and Engineering Design, John Wiley & Sons Inc., 1997.

[6] L. Jiao, and L. Wang, "Novel genetic algorithm based on immunity," IEEE Transactions on Systems, Man and Cybernetics, Part A, Vol. 30, No. 5, 2000, pp.552 –561.

[7] K. F. Man, K. S. Tang, and S. Kwong, Genetic Algorithms: Concepts and Designs. Springer, New York, 1999.

[8] G. Winter, J. Periaux, M. Galan, and P. Cuesta, editors. Genetic Algorithms in Engineering and Computer Science. Wiley, New York, 1995.

[9] G. Laporte, A. Asef-Vaziri, and C. Sriskandarajah, "Some applications of the generalized travelling salesman problem," Journal of the Operational Research Society,Vol 47, No12,1996,pp.1461–1467.

[10] M. Dorigo, and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," IEEE Trans. On Evolutionary Computation, Vol. 1,1997, pp.53-66.

[11] R. Beckers, J. L. Deneubourg, and S. Goss, "Trails and u-turns in the selection of the shortest path by the ant Lasius Niger," Journal of Theoretical Biology,Vol. 159, 1992, pp.397-415.

Shang Gao was born in 1972, and received his M.S. degree in 1996 and Ph.D degree in 2006. He now works in school of computer science and technology, Jiangsu University of Science and Technology. He is a professor and He is engage mainly in systems engineering and soft computing.

Zaiyue Zhang was born in 1961, and received his M.S. degree in mathematics in 1991 from the Department of Mathematics, Yangzhou Teaching College, and the Ph.D. degree in mathematics in 1995 from the Institute of Software, the Chinese Academy of Sciences. His current research areas are recursion theory, knowledge representation and knowledge reasoning.

Cungen Cao was born in 1964, and received his M.S. degree in 1989 and Ph.D. degree in 1993 both in mathematics from the Institute of Mathematics, the Chinese Academy of Sciences. Now he is a professor of the Institute of Computing Technology, the Chinese Academy of Sciences. His research area is large scale knowledge processing.