# 2

# Bio-inspired Algorithms for TSP and Generalized TSP

Zhifeng Hao, Han Huang and Ruichu Cai
*South China University of Technology*
*China*

## 1. Introduction

The Traveling Salesman Problem (TSP) is to find a Hamiltonian tour of minimal length on a fully connected graph. The TSP is a NP-Complete, and there is no polynomial algorithm to find the optimal result. Many bio-inspired algorithms has been proposed to address this problem. Generally, generic algorithm (GA), ant colony optimization (ACO) and particle swarm optimization (PSO) are three typical bio-inspired algorithm for TSP. In this section we will give a brief introduction to the above three bio-inspired algorithms and their application to the TSP.

### 1.1 GAs for TSP

GAs were introduced by Holland in the 1970s [1]. These algorithms are adaptive search techniques based on the mechanisms of natural selection and the survival of the fittest concept of biological evolution. By simulating biological evolution, GAs can solve searching problem domains effectively and easily apply to many of the current engineering problems. GAs have been widely used in many applications of TSP and its extensions throughout the literature [2-4].

A particularly nice introduction to GAs is given in Goldberg's book [5]. The main idea behind GAs is to start with randomly generating initial solutions and implements the "survival of the fittest" strategy to increasing better solutions through generations. A traditional GA process includes initial population generation, fitness evaluation, chromosome selection, applying genetic operators for reproduction, and suspension condition.

In designing a GA, how to encode a search solution is a primary and key issue [6]. Many optimization operators for TSP were proposed by Goldberg [5]. A commonly used encoding strategy is transposition expression [7]. In the transposition expression strategy, each city of the TSP is encoded as a gene of the chromosome with the constraint that each city appears once and only once in the chromosome. Transposition expression is the most nature expression for TSP which based on the order of tour. While such method may leads to infeasible tour after traditional crossover operator [7]. This is a common occurrence for TSP. Although feasibility can be maintained in many ways named 'repair algorithms', such algorithms can consume a considerable amount of time and can inhibit convergence.



Source: Travelling Salesman Problem, Book edited by: Federico Greco, ISBN 978-953-7619-10-7, pp. 202, September 2008, I-Tech, Vienna, Austria

| 3 | 2 | 1 | 5 | 4 | 6 | 8 | 7 |
|---|---|---|---|---|---|---|---|

Figure 1. Transposition expression encoding method for TSP

Another typical encoding method is Random Keys encoding [8] which is introduced by Bean. In Random Keys encoding a random numbers encode the structure of the solution. Such representation ensures that feasible tours are maintained during the application of genetic operators.

In the GA, the crossover and mutation are two of most important method for the success of the algorithm. A crossover operator generates new individuals through recombining the current population hopefully to retain good features from the parents. Numbers of different crossovers have been proposed in the literatures to solve the TSP using a GA. The partially mapped crossover [5,11-12], linear order crossover [13] and order based crossover [5,8,11,12,14] are the commonly used crossover strategy in the TSP context. Expect the commonly used crossover strategy, many different crossover strategy are proposed for the TSP problem, for example: sub-tour crossover[15,16], edge recombination [17-20], distance preserving crossover [21-22], generic crossover [23], NGA [24], EAX [25-26], GSX [27-28], heuristic based crossover [29-35].

A mutation operator is used to enhance the diversity and provide a chance to escape from local optima. Many mutation operators were proposed such as inverse, insert, displace, swap, hybrid mutation [34], and heuristic mutation. The former five are realized by small alterations of genes. Heuristic mutation was proposed by Cheng and Gen [37-38], which adopts a neighborhood strategy to improve the solution.

At present, the genetic algorithm to solve the TSP has been to promote large-scale TSP, as well as a multiple TSP (MTSP) and generalized TSP. A lot of progress was made recently. Arthur E. Carter and Cliff T. Ragsdale propose a new GA chromosome and related operators for the MTSP [39]. H. D. Nguyen, et al described a hybrid GA based on a parallel implementation of a multi population steady-state GA involving local search heuristics [40]. Samanlioglu et.al proposes a methodology uses a "target-vector approach" in which the evaluation function is a weighted Tchebycheff metric with an ideal point for a symmetric multi-objective traveling salesman problem [41-43].

### 1.2 Ant colony optimization (ACO) for TSP

Ant Colony Optimization (ACO), first proposed by M. Dorigo et al. [44-46], is a population-based, general-purpose heuristic approach to combinational optimization problems. The earliest ACO algorithm [44-45], Ant System (AS), was applied to the TSP (mainly because the TSP is "a shortest path problem to which the ant colony metaphor is easily adapted and that it is a didactic problem" [4]. After that, most improved ACO algorithms also used the TSP as a test problem and the result is promising.

As the name suggests, ACO took inspiration from the foraging behavior of real ant colonies. Ants deposit pheromone on the ground they cover while working, forming a pheromone trail. Other ants tend to follow the pheromone trail. Consider an ant colony exploring the paths between their nest to a food source. At the beginning, the ants choose paths randomly in equal rate since there's no pheromone on the paths help them make the decision. Suppose that every ant walk in the same speed, shorter paths accumulate pheromone faster than longer paths because ants on those paths return earlier. A moment later, the difference in the

amount of pheromone on the paths becomes sufficient large so that the ants' decision are influenced and more ants select the shorter paths. Experiments show that this behavior can lead the ant colony to the shortest path.

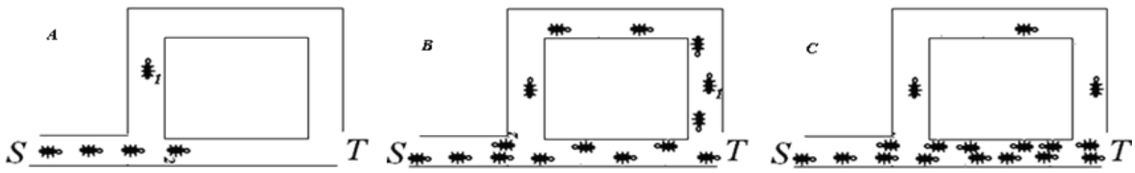


Figure 2. an ant colony exploits the paths between ***S*** and ***T***. (A) The two paths are selected with the same probability at first. (B) Ant 2 choosing the lower path returns to ***S*** earlier. Thus pheromone on the lower path rises faster. (C) Most ants walk on the lower path after a minute.

Typical ant algorithms stimulate the above foraging behavior of ant colonies using a set of agents (artificial ants) and an indirect communication mechanism employing (artificial) pheromone. A simple framework may look like this:

**Loop** /* **at this level each loop is called iteration** */
  Each ant is positioned on a starting node.
  **Loop** /* **at this level each loop is called a step** */
    Each ant applies a stochastic state transition rule to incrementally build a solution
  **Until** all ants have built a complete solution
  A pheromone updating rule is applied
**Until** End condition

The stochastic state transition rule and the pheromone updating rule are two factor to the success of the ACO. And many strategies have been proposed for these two operators. The Ant Colony System (ACS) [48-50] and MAX-MIN Ant System (MMAS) [51] are among the most successful algorithms [52]. Recent researches focus most on extending the applications of ACO algorithms to more challenge problems. There're also some studies on the convergence theory of ACO algorithms too [47, 53-58].

### 1.3 PSO for TSP

The particle swarm optimization (PSO) was originally presented by Kennedy and Eberhart in 1995 [59]. It is an algorithm based stochastic optimization technique which inspired by social behavior among individuals. In the PSO system, individuals (we call them particles) move around a multidimensional search space. Each particle represents a potential solution of the problem, and can remember the best position (so1ution) it has reached. All the particles can share their information about the search space, so there is a global best solution.

In each iteration, every particle adjusts its velocity $v_i$ and position $x_i$ according to the following formulas:

$$\begin{aligned} v_i &= v_i + c_1 * r_1 * (x_{i,pbest} - x_i) + c_2 * r_2 * (x_{i,gbest} - x_i) \\ x_i &= x_i + v_i \end{aligned} \tag{1}$$

Where $w$ is inertia weight, $c_1$ and $c_2$ are acceleration coefficients, $r_1$ and $r_2$ are two independent random values between 0 and 1. $x_{i,pbest}$ is the best solution this particle has reached; $x_{i,gbest}$ is the global best solution of all the particles until now.

Due to the continuous characters of the position of particles in PSO, the standard encoding scheme of PSO can not be directly adopted for TSP. Much work was published to avoid such problem. Clerc adopted discrete particle swarm optimization (DPSO)[60] to make PSO suitable for solving TSP. Bo Liu, et al. proposes a PSO-based MA [61](PSOMA) for TSP, which combined evolutionary searching mechanism of PSO and adaptive local search. Yong-Qin Tao, et al. proposed a GRPSAC algorithm [62] combined ACO with PSO organically and adds gene regulation operator at the same time, which make solution of TSP problem more efficiency. Other recently work such as heuristic information method based on improved fuzzy discrete PSO [63] and chaotic PSO algorithm [64] were proved to be effective for TSP.

## 2. Ant colony optimization (ACO) for TSP

### 2.1 The method of ant colony optimization solving TSP

Among the bio-inspired algorithms, the ant colony optimization (ACO) is a popular approach for TSP since it's proposed by M.Dorigo in early nineties [65-66]. Ant colony optimization (ACO) takes inspiration from the foraging behavior of some ant species. These ants deposit pheromone on the ground in order to mark some favorable path that should be followed by other ants of the colony. Ant colony optimization exploits a similar mechanism for solving optimization problems.

In TSP, a set of cities is given and the distance between each of them is known. The goal is to find a Hamiltonian tour of minimal length on a fully connected graph. This goal is very similar with the ants to find the shortest path between the nest and the food source. In ant colony optimization, the problem is tackled by simulating a number of artificial ants moving on a graph that encodes the problem itself. A variable called pheromone is associated with each edge and can be read and modified by ants. The artificial ants explore the pheromone to find the most favorable path which is the shortest Hamiltonian Tour in TSP.

Ant colony optimization is an iterative algorithm. In an iterative step, each ant of the colony builds a solution by walking from vertex to vertex on the graph with the constraint of not visiting any vertex that has been visited before. The solution construction and the pheromone updating are two main steps for the ACO. In the solution construction step, an ant selects the next vertex to be visited according to a stochastic mechanism that is biased by the pheromone. After the solution construction step, the pheromone is updated on the basis of the quality of the solutions.

Under the above framework, many different version of the algorithm are proposed. According to the M.Dorigo's work [46,67], the Ant System (AS), MAX−MIN Ant System (MMAS) and Ant Colony System (ACS) are three of most popular ant algorithms. Following, we will give a short brief of those three algorithms on TSP.

#### 2.1.1 Ant System (AS)

Ant System is the first ACO algorithm proposed in the literature [44,65-66]. Its main characteristic is that, at each iteration, the pheromone values are updated by all the $m$ ants that have built a solution in the iteration itself. The pheromone $\tau_{ij}$, associated with the edge joining cities $i$ and $j$, is updated as follows:

$$\tau_{ij} \leftarrow (1-\rho)\cdot\tau_{ij} + \sum_{k=1}^{m}\Delta\tau_{ij}^{k} \quad (1)$$

where $\rho$ is the evaporation rate, $m$ is the number of ants, and $\Delta\tau_{ij}^{k}$ is the quantity of pheromone laid on edge $(i, j)$ by ant $k$:

$$\Delta\tau_{ij}^{k} = \begin{cases} Q/L_k & if\ ant\ k\ used\ edge(i,j)\ in\ its\ tour, \\ 0 & otherwise, \end{cases} \quad (2)$$

where $Q$ is a constant, and $L_k$ is the length of the tour constructed by ant $k$.

In the construction of a solution, ants select the following city to be visited through a stochastic mechanism. When ant $k$ is in city $i$ and has so far constructed the partial solution $S^p$, the probability of going to city $j$ is given by:

$$P_{gs}^{\ k}(t) = \begin{cases} \dfrac{[\tau_{gs}(t)]^{\alpha}[\eta_{gs}]^{\beta}}{\sum_{r\in J_k(g)}[\tau_{gr}(t)]^{\alpha}[\eta_{gr}]^{\beta}} & if \quad s\in J_k(g) \\ 0 & otherwise \end{cases} \quad (3)$$

where $J_k(g)$ is the set of cities not visited yet by ant $k$ when at city $g$. The parameters $\alpha$ and $\beta$ control the relative importance of the pheromone versus the heuristic information $\eta_{ij}$, which is given by $\eta_{ij} = \dfrac{1}{d_{ij}}$, where $d_{ij}$ is the distance between cities $i$ and $j$.

### 2.1.2 MAX −MIN Ant System (MMAS)

The MAX −MIN Ant System [51] is an improvement over the original Ant System. In the MMAS, only the best ant updates the pheromone trails and the value of the pheromone is bound. The pheromone update is implemented as follows:

$$\tau_{ij} = \begin{cases} \tau_{\max} & if\ (1-\rho)\cdot\tau_{ij} + \Delta\tau_{ij}^{best} > \tau_{\max}, \\ \tau_{\min} & if\ (1-\rho)\cdot\tau_{ij} + \Delta\tau_{ij}^{best} < \tau_{\min}, \\ (1-\rho)\cdot\tau_{ij} + \Delta\tau_{ij}^{best} & otherwise; \end{cases} \quad (4)$$

where $\tau_{\max}$ and $\tau_{\min}$ min are respectively the upper and lower bounds imposed on the pheromone and $\Delta\tau_{ij}^{best}$ is:

$$\Delta\tau_{ij}^{best} = \begin{cases} 1/L_{best} & if\ (i,j)\ belongs\ to\ the\ best\ tour \\ 0 & otherwise; \end{cases} \quad (5)$$

where $L_{best}$ is the length of the tour of the best ant.

For the $\tau_{\max}$ and $\tau_{\min}$, they are typically obtained empirically and tuned on the specific problem considered [68]. And some guidelines have been provided for defining $\tau_{\min}$ and $\tau_{\max}$ on the basis of analytical considerations [51].

### 2.1.3 Ant Colony System (ACS)

The ACS was considered the most efficient algorithm on the TSP problem. The main contribution of ACS [48, 50, 69] is introducing a novel local pheromone update in addition to the global pheromone.

The local pheromone update is performed by all the ants after each construction step. Each ant applies it only to the last edge traversed:

$$\tau_{ij} = (1-\varphi)\cdot\tau_{ij} + \varphi\cdot\tau_0 \tag{6}$$

where $\varphi \in (0,1]$ is the pheromone decay coefficient, and $\tau_0$ is the initial value of the pheromone.

Using the local update strategy, the pheromone concentration on the traversed edges is decreased. So, the subsequent ants are encouraged to choose other edges and to produce different solutions. This makes it less likely that several ants produce identical solutions during one iteration.

## 2.2 An adaptive strategy for weight parameter

Many strategies for ACO have been studied, but little theoretical work has been done on ACO's parameters $\alpha$ and $\beta$, which control the relative weight of pheromone trail and heuristic value. In this part, we will theoretical show the importance and functioning of $\alpha$ and $\beta$. The theoretical analysis show that a fixed $\beta$ may not enable ACO to use both heuristic and pheromone information for solution when $\alpha$ = 1. An adaptive $\beta$ strategy and a new ACO called adaptive weight ant colony system (AWACS) with the adaptive $\beta$ and $\alpha$ = 1 is introduced. The numerical experiment results show that the AWACS is more effective and steady than traditional ACS.

### 2.2.1 Theoretical analysis of the weight parameter

Given $a, b \in J_k(g)$, if ${P_{ga}}^k(t) > {P_{gb}}^k(t)$, which means that city *a* may be chosen by the ant *k* as the next city to city *g* with higher probability than city *b*, then $\alpha$ and $\beta$ satisfies the following formula: ${P_{ga}}^k(t) > {P_{gb}}^k(t) \Leftrightarrow [\tau_{ga}(t)]^{\alpha}[\eta_{ga}]^{\beta} > [\tau_{gb}(t)]^{\alpha}[\eta_{gb}]^{\beta}$.

When $\tau_{ga}(t) = \tau_{gb}(t)$ or $\eta_{ga} = \eta_{gb}$, for $\forall \alpha, \beta > 0$, the formula above holds, so we have:

$${P_{ga}}^k(t) > {P_{gb}}^k(t) \Leftrightarrow \begin{cases} \eta_{ga} > \eta_{gb} & \tau_{ga} = \tau_{gb} \\ \tau_{ga} > \tau_{gb} & \eta_{ga} = \eta_{gb} \end{cases} \tag{8}$$

However, when $\tau_{ga}(t) \neq \tau_{gb}(t), \tau_{ga}(t) > 0, \tau_{gb}(t) > 0$ and $\eta_{ga} \neq \eta_{gb}$ ($\eta_{ga}(t) > 0, \eta_{gb}(t) > 0$), one has: $P_{ga}{}^{k}(t) > P_{gb}{}^{k}(t) \Leftrightarrow [\tau_{ga}(t)]^{\alpha}[\eta_{ga}]^{\beta} > [\tau_{gb}(t)]^{\alpha}[\eta_{gb}]^{\beta} \Leftrightarrow \log \tau_{ga}(t) - \log \tau_{gb}(t) > \frac{\beta}{\alpha}(\log \eta_{gb} - \log \eta_{ga})$.
And we have:

$$\begin{cases} \frac{\beta}{\alpha} < \frac{\log \tau_{ga}(t) - \log \tau_{gb}(t)}{\log \eta_{gb} - \log \eta_{ga}} & \eta_{gb} > \eta_{ga} \\ \frac{\beta}{\alpha} > \frac{\log \tau_{ga}(t) - \log \tau_{gb}(t)}{\log \eta_{gb} - \log \eta_{ga}} & \eta_{gb} < \eta_{ga} \end{cases} \tag{7}$$

Particularly, when $\alpha$=1, which exists in ACO algorithms like ACS, a conclusion can be drawn:

$$P_{ga}{}^{k}(t) > P_{gb}{}^{k}(t) \Leftrightarrow \begin{cases} \beta < \frac{\log \tau_{ga}(t) - \log \tau_{gb}(t)}{\log \eta_{gb} - \log \eta_{ga}} & \eta_{gb} > \eta_{ga} \\ \beta > \frac{\log \tau_{ga}(t) - \log \tau_{gb}(t)}{\log \eta_{gb} - \log \eta_{ga}} & \eta_{gb} < \eta_{ga} \end{cases} \tag{8}$$

For the sake of convenience, some symbols about the pheromone trail are defined as follows: $\tau_{g}{}^{\max}(t) = \max_{r \in J_k(g)} \{\tau_{gr}(t)\}$ is the highest pheromone trail among all the cities feasible to be selected as next stop to city $g$. $\tau_{g}{}^{\min}(t) = \min_{r \in J_k(g)} \{\tau_{gr}(t)\}$ is the lowest one, and $\tau_{g}{}^{ave}(t) = |J_k(g)|^{-1} \sum_{r \in J_k(g)} \tau_{gr}(t)$ is the average pheromone trail, where $|J_k(g)|$ is the number of elements in the set $J_k(g)$. $\eta_{g}{}^{\max} = \max_{r \in J_k(g)} \{d_{gr}{}^{-1}\}$ is the highest heuristic value of elements in the set $J_k(g)$. $\eta_{g}{}^{\min} = \min_{r \in J_k(g)} \{d_{gr}{}^{-1}\}$ stands for the lowest heuristic value, and $\eta_{g}{}^{ave} = |J_k(g)|^{-1} \sum_{r \in J_k(g)} \eta_{gr} = |J_k(g)|^{-1} \sum_{r \in J_k(g)} d_{gr}{}^{-1}$ is the average heuristic value.

Let $\alpha = 1$, two cases are discussed in the following:

① $[\tau_{g}{}^{\max}(t)]^{\alpha}[\eta_{g}{}^{\min}]^{\beta} > [\tau_{g}{}^{ave}(t)]^{\alpha}[\eta_{g}{}^{\max}]^{\beta}$. It means that the ants will select the paths with the maximum pheromone trail with a very high probability ACS. According to Formula (3), one has $\beta < \frac{\log \tau_{g}{}^{\max}(t) - \log \tau_{g}{}^{ave}(t)}{\log \eta_{g}{}^{\max} - \log \eta_{g}{}^{\min}} = M_1(g,t)$, because it is obvious that $\eta_{g}{}^{\max} > \eta_{g}{}^{\min}$ holds in TSPLIB problems.

② $[\tau_{g}{}^{\min}(t)]^{\alpha}[\eta_{g}{}^{\max}]^{\beta} > [\tau_{g}{}^{\max}(t)]^{\alpha}[\eta_{g}{}^{ave}]^{\beta}$. It means that the ants will select the paths with the maximum heuristic value with a very high probability in ACS. It is

obvious that $\beta > \dfrac{\log \tau_g^{\min}(t) - \log \tau_g^{\max}(t)}{\log \eta_g^{ave} - \log \eta_g^{\max}} = M_2(g,t)$ holds, when $\eta_g^{\max} > \eta_g^{ave}$ and $\tau_g^{\min}(t) > 0$.

According to the analysis of case ① and ②, ACO may work as a non-heuristic searching when $\beta < M_1(g,t)$, and as a greedy searching without using pheromone trail when $\beta > M_2(g,t)$. Therefore, a fixed $\beta$ may not enable ACO to find optimal solution by using both heuristic and pheromone information. However, the process of ACO will not be in the extreme as non-heuristic or greedy searching when $M_1(g,t) \le \beta \le M_2(g,t)$. So a new adaptive parameter $\beta$ is designed as follows:

$$\alpha = 1, \beta(g,t) = \frac{\log \tau_g^{ave}(t) - \log \tau_g^{\max}(t)}{\log \eta_g^{ave} - \log \eta_g^{\max}} \quad (\tau_g^{\min}(t) > 0) \tag{9}$$

where $M_1(g,t) \le \beta(g,t) \le M_2(g,t)$ can be proved.

Based on the adaptive parameter $\beta(g,t)$ strategy shown in Formula (4), a novel ACO algorithm, which is called adaptive weight ant colony system (AWACS) can be described as follows.

Initialize /* $\beta$ is chosen in [0, 5] randomly, $q_0$=0.6 */
**Loop /* at this level each loop is called iteration */**
  Each ant is positioned on a starting node.
  **Loop /* at this level each loop is called a step */**
    Each ant applies a stochastic state transition rule to incrementally build a solution and a local pheromone updating rule
  **Until** all ants have built a complete solution
  A global pheromone updating rule is applied
  $\beta(g,t)$ **is updated (**$g$ = 1,…,$n$**) following Formula (11)**
**Until** End condition

The proof of its convergence ($g$ = 1,…,$n$) is the same as the one in Ref. [54]. According to the work of Ref. [54], it still holds that $\tau_g^{\min}(t) > 0$ and $\tau_g^{\max}(t) < +\infty$ ($g$ = 1,…,$n$) when the adaptive parameter $\beta(g,t)$ strategy in Formula (4) is applied. Then, AWACS can be proved to find the optimal solution with probability one following the conclusion given by T. Stützle and M. Dorigo [54,69].

### 2.2.2 Numerical results and analyses

A comparison of the performance of ACS and AWACS is given in this section. In our experiments, the parameters are set as follows: $m$ = 10, $\alpha$ = $\rho$= 0.1, $\tau_0 = (nL_{nn})^{-1}$. $q_0$ is set $q_0$ =0.9 in ACS, and $q_0$ =0.6 in AWACS , respectively. The initial value of $\beta$ in AWACS is a

random figure changing in the interval [1,5]. The initial feasible solutions of TSP are generated in the way from Ref [49]. What's more, no local search strategy is used in experiment.
The experiments are conducted on two set of TSP problems. In the first set of 10 TSP, the distances between cities are measured by integers and in the left 10 TSP, and the distances are measured by real values. The datasets can be found in TSPLIB: http://www.iwr.uni-heidelberg.de/ iwr/comopt/soft/TSPLIB95/TSPLIB.html. The detail of the experiment result is given at table 1, table 2 and table 3.

| Instance | Optimal | Best (ACS) | Best (AWACS) | Average (ACS) | Average (AWACS) | $T_{avg}$(s) (ACS) | $T_{avg}$ (s) (AWACS) | Best $\beta$ (ACS) |
|---|---|---|---|---|---|---|---|---|
| st70 | 654 | 657 | 657 | 675.9 | 675.5 | 16.9 | 27.4 | 4 |
| rat99 | unknown | 1188 | 1188 | 1211.7 | 1199.4 | 53.2 | 59.7 | 3 |
| pr107 | unknown | 44539 | 44398 | 44906.3 | 44783.9 | 39.5 | 55.4 | 4 |
| pr124 | unknown | 59205 | 59067 | 59819.9 | 59646.6 | 59.2 | 42.3 | 4 |
| eil101 | 612 | 614 | 613 | 634.6 | 631.4 | 22.4 | 76.3 | 5 |
| rd100 | 7858 | 7909 | 7861 | 8100.4 | 8066.2 | 59.5 | 54.1 | 3 |
| eil51 | 415 | 415 | 415 | 423.9 | 423.7 | 6.7 | 7.8 | 3 |
| lin105 | 14345 | 14376 | 14354 | 14509.3 | 14465.6 | 73.7 | 50.8 | 4, 5 |
| kroD100 | 21249 | 21486 | 21265 | 21893 | 21628.2 | 25.8 | 60 | 5 |
| kroC100 | 20703 | 20703 | 20703 | 21165.3 | 20914.9 | 29.5 | 67.7 | 4 |

Table 1. Comparison I of the results obtained by ACS and AWACS

| Instance | Optimal | Best (ACS) | Best (AWACS) | Average (ACS) | Average (AWACS) | $T_{avg}$(s) (ACS) | $T_{avg}$ (s) (AWACS) | Best $\beta$ (ACS) |
|---|---|---|---|---|---|---|---|---|
| kroA100 | 21282 | 21285.44 | 21285.44 | 21345.78 | 21286.33 | 51.3 | 51.8 | 2, 3 |
| kroE100 | 22068 | 22078.66 | 22068.75 | 22206.62 | 22117.16 | 56.3 | 64.5 | 5 |
| berlin52 | 7542 | 7544.36 | 7544.36 | 7544.36 | 7544.36 | 8.7 | 9.8 | 5 |
| kroB150 | 26130 | 26127.35 | 26127.71 | 26332.75 | 26214.10 | 177.8 | 164.8 | 5 |
| ch150 | 6528 | 6530.90 | 6530.90 | 6594.94 | 6559.66 | 373.6 | 118.1 | 2 |
| kroB100 | 22141 | 22139.07 | 22139.07 | 22335.72 | 22177.47 | 55.5 | 68.6 | 4 |
| kroA150 | 26524 | 26618.33 | 26524.86 | 26809.08 | 26685.73 | 204.5 | 242.9 | 5 |
| u159 | 42080 | 42075.67 | 42075.67 | 42472.04 | 42168.54 | 356.7 | 80.2 | 1 |
| pr76 | 108159 | 108159.4 | 108159.4 | 108610.6 | 108581.7 | 50.5 | 42.8 | 1 |
| pr136 | 96772 | 96870.89 | 96785.86 | 97854.16 | 97236.61 | 344.3 | 158.9 | 14,5 |

Table 2. Comparison II of the results obtained by ACS and AWACS

| Instance \Standard deviation | AWACS | ACS $\beta = 1$ | ACS $\beta = 2$ | ACS $\beta = 3$ | ACS $\beta = 4$ | ACS $\beta = 5$ |
|---|---|---|---|---|---|---|
| kroA100 | 8.49 | 460.04 | 338.84 | 183.55 | 625.81 | 447.03 |
| kroE100 | 123.82 | 327.01 | 467.75 | 529.73 | 330.12 | 366.49 |
| berlin52 | 0.00 | 376.98 | 376.19 | 357.76 | 548.34 | 0.00 |
| kroB150 | 221.98 | 447.98 | 652.57 | 821.48 | 664.54 | 486.91 |
| ch150 | 54.50 | 114.76 | 153.71 | 109.36 | 171.66 | 54.81 |
| kroB100 | 132.37 | 554.43 | 579.73 | 1091.25 | 558.86 | 233.01 |
| kroA150 | 384.39 | 522.81 | 942.11 | 974.79 | 640.34 | 432.72 |
| u159 | 623.16 | 726.99 | 3531.45 | 2458.43 | 1509.09 | 1661.63 |
| pr76 | 1158.43 | 1180.56 | 5058.92 | 2088.68 | 1677.73 | 1411.15 |
| pr136 | 1300.78 | 2386.53 | 5303.40 | 4572.69 | 3304.40 | 2173.27 |

Table 3. Comparison of standard deviations of the tour lengths obtained by AWACS and ACS

As shown in the above tables, there might be something like precision and time cost in the result of our experiments different from those in the former research because of the different program tools, systems and computing machines. Another possible reason is that the distances between cities in the first 10 instances are measured by integer numbers. But ACS and AWACS are running in the same setting, so the result remains helpful to compare the performance of these two algorithms.

From Table 1-2, it could be seen that AWACS performs better than ACS with the fixed $\beta$. The shortest lengths and the average lengths obtained by AWACS are shorter than those found by ACS in all of the TSP instances. As Table 3 shows, it can be concluded that the standard deviations of the tour lengths obtained by AWACS are smaller than those of ACS with the fixed$\beta$. Therefore, we can conclude that AWACS is proved to be more effective and steady than ACS.

ACS has to change the best integer value of parameter$\beta$ with respect to different instances in the experiments. AWACS can avoid the difficulty about how to choose the experimental value of$\beta$, because its adaptive strategy can be considered as a function trying to find the best setting for each path search via meeting the request of Formula 4. Though, the time cost $t_{avg}$ of AWACS is more than ACS in some case, it is less than the sum of time ACS costs with $\beta$ =*1,2,3,4,5* in all of the instances. As a result, the adaptive setting can save much time in choosing the experimental$\beta$. Item $t_{avg}$ of AWACS is not less than ACS in all of the instances because it needs to compute the value of $\beta^{n}$ (number of cities) times in each iteration. However, the adaptive function of AWACS is feasible to use because of its acceptable time cost.

### 2.3 Bi-directional searching ant colony system

In 2.2, an adaptive strategy for the weight parameter is proposed by exploring the function of the parameter in the stochastic mechanism. In this section, we will further explore the stochastic mechanism and a bi-directional searching ant colony system is proposed.

#### 2.3.1 Bi-directional searching strategy using adaptive weight parameter

In the proposed ACO algorithms, the state transition rule of the artificial ants is given as follows:

$$P_{gs}^{\ k}(t)=\begin{cases}\dfrac{[\tau_{gs}(t)]^{\alpha}[\eta_{gs}]^{\beta(g,t)}}{\sum_{r\in J_k(g)}[\tau_{gr}(t)]^{\alpha}[\eta_{gr}]^{\beta(g,t)}} & if \quad s\in J_k(g)\\ 0 & otherwise\end{cases} \tag{10}$$

The only difference between the (10) and (3) is the setting of the parameter $\beta(g,t)$. In the Bi-directional case, the parameter is set with one of the following two methods by probability 0.5

1). $\beta(g,t)=\dfrac{\log \tau_g^{\max}(t)-\log \tau_g^{ave}(t)}{\log \eta_g^{\max}-\log \eta_g^{\min}}-\varepsilon_0$ 2). $\beta(g,t)=\dfrac{\log \tau_g^{\min}(t)-\log \tau_g^{\max}(t)}{\log \eta_g^{ave}-\log \eta_g^{\max}}+\varepsilon_0$.

where $\tau_g^{\max}(t)=\max_{i\in J_k(g)}\{\tau_{gi}(t)\}$ is the highest pheromone trail among all the cities feasible to be selected as next stop to city $g$. $\tau_g^{\min}(t)=\min_{i\in J_k(g)}\{\tau_{gi}(t)\}$ is the lowest one, and $\tau_g^{ave}(t)=|J_k(g)|^{-1}\sum_{i\in J_k(g)}\tau_{gi}(t)$ is the average pheromone trail, where $|J_k(g)|$ is the number of elements in the set $J_k(g)$. $\eta_g^{\max}=\max_{i\in J_k(g)}\{d_{gi}^{-1}\}$ is the highest heuristic value of elements in the set $J_k(g)$. $\eta_g^{\min}=\min_{i\in J_k(g)}\{d_{gi}^{-1}\}$ stands for the lowest heuristic value, $\eta_g^{ave}=|J_k(g)|^{-1}\sum_{i\in J_k(g)}\eta_{gi}=|J_k(g)|^{-1}\sum_{i\in J_k(g)}d_{gi}^{-1}$ is the average heuristic value, and $\varepsilon_0$>0..

For the first method,

$$\beta(g,t)<\frac{\log \tau_g^{\max}(t)-\log \tau_g^{ave}(t)}{\log \eta_g^{\max}-\log \eta_g^{\min}}\Leftrightarrow\frac{[\tau_g^{\max}(t)]^{\alpha}[\eta_g^{\min}]^{\beta(g,t)}}{\sum_{r\in J_k(g)}[\tau_{gr}(t)]^{\alpha}[\eta_{gr}]^{\beta(g,t)}}>\frac{[\tau_g^{ave}(t)]^{\alpha}[\eta_g^{\max}]^{\beta(g,t)}}{\sum_{r\in J_k(g)}[\tau_{gr}(t)]^{\alpha}[\eta_{gr}]^{\beta(g,t)}}.$$

It means that the ants will select the paths with the maximum pheromone trail by the higher probability than most of the other feasible paths, even if they are paths with the highest heuristic value.

For the second one,

$$\beta(g,t)>\frac{\log \tau_g^{\min}(t)-\log \tau_g^{\max}(t)}{\log \eta_g^{ave}-\log \eta_g^{\max}}\Leftrightarrow\frac{[\tau_g^{\min}(t)]^{\alpha}[\eta_g^{\max}]^{\beta(g,t)}}{\sum_{r\in J_k(g)}[\tau_{gr}(t)]^{\alpha}[\eta_{gr}]^{\beta(g,t)}}>\frac{[\tau_g^{\max}(t)]^{\alpha}[\eta_g^{ave}]^{\beta(g,t)}}{\sum_{r\in J_k(g)}[\tau_{gr}(t)]^{\alpha}[\eta_{gr}]^{\beta(g,t)}}$$

It means that the ants will select the paths with the maximum heuristic value by the higher probability than most of the other feasible paths, even if they are paths with the highest pheromone trail.
Combing the above two methods of the parameter setting, the new ACO algorithm BSACS is designed as:

Initialize
/*$\beta$ is chosen between 0 and 5 randomly, $q_0$=0.6 */
**Loop /* at this level each loop is called iteration */**
Each ant is positioned on a starting node
**Loop /* at this level each loop is called a step */**
Each ant applies a state transition rule to incrementally build a solution and a local pheromone updating rule is applied
**Until** all ants have built a complete solution
A global pheromone updating rule is applied
$\beta(g,t)$is updated by either of the two methods by probability 0.5 (g=1,…,$n$)
**Until** End_condition

### 2.3.2 Numerical results and analyses

In this section, several large TSP instances of TSPIB [70] are tested by BSACS and ACS to show the efficiency of the BSACS. The parameters are set as follows: $m$ = 10, $a$ = $\rho$= 0.1, $\tau_0 = (nL_{nn})^{-1}$, and $\alpha$=1. $q_0$=0.9 in ACS, $\varepsilon_0$=0.001 and $q_0$=0.6 in BSACS , respectively. All the instances are computed by BSACS 10 times, and so does ACS with each $\beta(\beta$ =1,2,3,4,5). As shown in Table 4 and Table 5, Item (1) is the length of the best tour obtained by ACS and BSACS. Item (6) is the length of optimal solution published in the TSPLIB: http://www.iwr.uniheidelberg.de/iwr/comopt/soft/TSPLIB95/TSPLIB.html. Item (2) is the relative error which can be computed by $(2) = ((1)-(3))\times(3)^{-1}\times 100\%$. Item (1) and (2) show that BSACS can obtain better solution than ACS in all of the instances. Item (4) is the average length of the solutions found by both ACS and BSACS. Item (5) is the best value of $\beta$ which can make ACS perform the best according to Item (1) or Item (4).
The experiment result shows that BSACS can perform better than ACS in every computation. What's more, ACS has to change the selection of $\beta$ in different instances and cannot solve different large size TSP problems steadily with a fixed value of $\beta$. The reason is that ACS is not able to effectively use the pheromone trail and heuristic value in searching when $\beta$ of the transition rule is fixed and unchanged in iterations. This disadvantage could be avoided by using BSACS because the new rule of BSACS (Formula 1) functions based on both pheromone trail and heuristic value adaptively. For the computational complexity, the BSACS need more time than ACS, because $\beta(g,t)$(g=1,…, $n$) has to be updated at each iteration. However, it doesn't mean that the cost of BSACS is more than ACS in the application, because the cost of ACS for the best parameter selection (Item (5) in Table 2) has not been calculated here. Therefore, BSACS can save the time in choosing the experimental value of the parameter. Generally, the BSACS improves the performance of ACS in solving large size TSP problems.

| Instance | Algorithm | (1)Best | (2)%Error | (3)Optimal |
|---|---|---|---|---|
| **bier127** | ACS | 118773 | 0.423 | 118282 |
| | BSACS | 118372 | 0.076 | |
| **d198** | ACS | 15888 | 0.68 | 15780 |
| | BSACS | 15780 | 0.00 | |
| **ts225** | ACS | 128829 | 1.734 | 126643 |
| | BSACS | 126905 | 0.206 | |
| **pcb442** | ACS | 51286 | 0.96 | 50799 |
| | BSACS | 51271 | 0.93 | |
| **att532** | ACS | 28147 | 1.67 | 27686 |
| | BSACS | 27939 | 0.91 | |
| **u574** | ACS | 38318 | 3.829 | 36905 |
| | BSACS | 37662 | 2.052 | |
| **rat783** | ACS | 9015 | 2.37 | 8806 |
| | BSACS | 8819 | 0.14 | |
| **fl1577** | ACS | 22977 | 3.27 | 222490. |
| | BSACS | 22611 | 1.63 | |

Table 4. Comparison of the best solution obtained by ACS and BSACS

| Instance | Algorithm | (4) Average | (5)Best $\beta$ of ACS | (6) $t_{avg}$ (second) |
|---|---|---|---|---|
| **bier127** | ACS | 119185.3 | 2, 3 | 45.6 |
| | BSACS | 118826.8 | - | 91.0 |
| **d198** | ACS | 16054.8 | 2 | 97.8 |
| | BSACS | 15842.1 | - | 124.5 |
| **ts225** | ACS | 129102.5 | 4, 5 | 32.0 |
| | BSACS | 127262.8 | - | 67.0 |
| **pcb442** | ACS | 51690.2 | 2 | 281.6 |
| | BSACS | 51642.8 | - | 461.2 |
| **att532** | ACS | 28532.0 | 2 | 401.5 |
| | BSACS | 28163.7 | - | 539.7 |
| **u574** | ACS | 38657.8 | 1, 5 | 305.3 |
| | BSACS | 38291.9 | - | 504.3 |
| **rat783** | ACS | 9066.0 | 2 | 1185.4 |
| | BSACS | 8985.8 | - | 1559.8 |
| **fl1577** | ACS | 23163.5 | 2 | 3884.0 |
| | BSACS | 22680.3 | - | 6290.2 |

Table 5. Comparison of the average solution obtained by ACS and BSAC

### 2.4 An adaptive volatility rate of pheromone trail

The following presents a trial work of setting the parameters of ACO adaptively. First, a tuning rule for $\rho$ is designed based on the quality of the solution constructed by artificial ants. Then, we introduce the adaptive $\rho$ to form a new ACO algorithm, which is tested to compute several benchmark instances of traveling sales-man problem and film-copy deliverer problem.

### 2.4.1 An adaptive volatility rate setting strategy

After constructing its tour, an artificial ant also modifies the amount of pheromone on the visited edges by applying the pheromone updating rule. The rule is designed so that it tends to give more pheromone to the edges which should be visited by ants. The classical pheromone updating rule is as (1). And the optimal $\rho$ was set $\rho$ =0.1 experimentally [46, 49, 55], which means that 90 per cent of the original pheromone trail remains and its 10 per cent is replaced by the increment.

In order to update the pheromone according to the quality of solutions found by ants, an adaptive rule for volatility of the pheromone trail is designed as follows:

$$\rho_m = L_m^{-1} / (L_m^{-1} + L_P^{-1}) \tag{11}$$

where$L_m$ is the length of the solution $S_m$ found by ant $m$, and $L_p$ is the length of the solution $S_p$ built based on the pheromone matrix, shown as $s = \arg\max_{u \in J_m(r)} \{[\tau(r,u)\}$ where $s$ is the city selected as the next one to city $r$ for any $(r,s) \in S_p$.

The motivation of the proposed rule is: better solutions should contribute more pheromone, and the worse ones contribute less. And a new ACO algorithm with the adaptive rule (shown as Equation 3) is introduced as follows:

Initialize
/*$\beta$ is chosen between 0 and 5 randomly, $q_0$=0.6 */
**Loop** /* at this level each loop is called iteration */
Each ant is positioned on a starting node
**Loop** /* at this level each loop is called a step */
Each ant applies a state transition rule to incrementally build a solution and a local pheromone updating rule is applied
**Each ant the calculate the $\rho_i$ is based on its solution's length**
**Until** all ants have built a complete solution
**$\rho_{best}$ is calculated based on the best solution $S_{best}$.**
**Carry out the pheromone updating rule with $\rho_i$ ($i$=1,…,$k$) and $\rho_{best}$.**
**Until** End_condition

### 2.4.2 Numerical results

This section indicates the numerical results in the experiment that the proposed ACO algorithm is used to solve TSP problems [69]. Several TSP instances are computed by ACS [49], ACO [71] and the proposed ACO on a PC with an Intel Pentium 550MBHz Processor and 256MB SDR Memory, and the results are shown in Table 1.

It should be noted that every instance is computed 20 times. The algorithms are both programmed in Visual C++6.0 for Windows System. They would not stop until a better solution could be found in 500 iterations, which is considered as a virtual convergence of the algorithms. Table 6 shows that the proposed ACO algorithm (PACO) performs better than ACS [49] and ACO [71]. The shortest lengths and the average lengths obtained by PACO are shorter than those found by ACS and ACO in all of the TSP instances. Furthermore, it can be concluded that the standard deviations of the tour lengths obtained by PACO are smaller than those of another algorithms. Therefore, we can conclude that PACO is proved to be

more effective and steady than ACS [49] and ACO [71]. Computation time cost of PACO is not less than ACS and ACO in all of the instances because it needs to compute the value of volatility rate $k$+1 times per iteration. Although all optimal tours of TSP problems cannot be found by the tested algorithms, all of the errors for PACO are much less than that for another two ACO approaches. The algorithms may make improvement in solving TSP when reinforcing heuristic strategies like local search like ACS-3opt [49] and MMAS+rs [51] are used.

| Problem | Algorithm | best | ave | time(s) | standard deviation |
|---|---|---|---|---|---|
| kroA100 | ACS | 21958 | 22088.8 | 65 | 1142.77 |
| | ACO | 21863 | 22082.5 | 94.6 | 1265.30 |
| | PACO | 21682 | 22076.2 | 117.2 | 549.85 |
| ts225 | ACS | 130577 | 133195 | 430.6 | 7038.30 |
| | ACO | 130568 | 132984 | 439.3 | 7652.80 |
| | PACO | 130507 | 131560 | 419.4 | 1434.98 |
| pr226 | ACS | 84534 | 86913.8 | 378.4 | 4065.25 |
| | ACO | 83659 | 87215.6 | 523.8 | 5206.70 |
| | PACO | 81967 | 83462.2 | 762.2 | 3103.41 |
| lin105 | ACS | 14883 | 15125.4 | 88.8 | 475.37 |
| | ACO | 14795 | 15038.4 | 106.6 | 526.43 |
| | PACO | 14736 | 14888 | 112.2 | 211.34 |
| kroB100 | ACS | 23014 | 23353.8 | 56.2 | 685.79 |
| | ACO | 22691 | 23468.1 | 102.9 | 702.46 |
| | PACO | 22289 | 22728 | 169.6 | 668.26 |
| kroC100 | ACS | 21594 | 21942.6 | 54.8 | 509.77 |
| | ACO | 21236 | 21909.8 | 78.1 | 814.53 |
| | PACO | 20775 | 21598.4 | 114.8 | 414.62 |
| lin318 | ACS | 48554 | 49224.4 | 849.2 | 1785.21 |
| | ACO | 48282 | 49196.7 | 902.7 | 2459.16 |
| | PACO | 47885 | 49172.8 | 866.8 | 1108.34 |

Table 6. Comparison of the ACS [49], ACO [51] and the proposed ACO (PACO) in TSP instances

## 3. Genetic algorithm for generalized TSP

### 3.1 Generalized TSP (GTSP)

The generalized traveling salesman problem (GTSP) is a kind of combinatorial optimization problem, which has been introduced by Henry-Labordere [72] and Saksena [73] in the context of computer record balancing and of visit sequencing through welfare agencies since 1960s. The GTSP can be described as the problem of seeking a special Hamiltonian cycle with lowest cost in a complete weighted graph.

Let *G*=(V, E, M) be a complete weighted graph where $V=\{v_1, v_2, \cdots, v_n\}\ (n>3)$, $E=\{e_{ij} \mid v_i, v_j \in V\}$ and $W=\{w_{ij} \mid w_{ij} \geq 0\ and\ w_{ii}=0, \forall i, j \in N(n)\}$ are vertex set, edge set, and cost set, respectively. The vertex set *V* is partitioned into *m* possibly intersecting groups $V_1, V_2, \cdots V_m$ with $|V_j| \geq 1$ and $V=\bigcup_{j=1}^{m} V_j$. The GTSP is required to pass through all of the groups, but not all of the vertices differing from that of TSP. For convenience, we also call ***W*** as the cost matrix and take it as ***W***=$(w_{ij})_{n\times n}$. There are two different kinds of GTSP under the abovementioned framework of the special Hamiltonian cycle [75-76]: (1) the cycle passes exactly one vertex in each group (refer to Fig. 1) and (2) the cycle passes at least one vertex in each group (refer to Fig. 2). The first kind of GTSP is also known as E-GTSP, where E stands for equality [76]. In this paper we only discuss the GTSP for the first case and will still call it as GTSP for convenience.

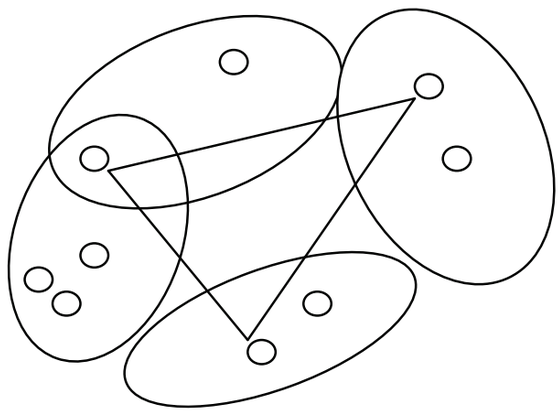

Figure 3. Exactly one vertex is visited in a GTSP cycle.

GTSP has extensive application fields. Laport et al. [75], Lien et al. [77], and Castelino et al. [78] reported the applications of GTSP. Just as mentioned in Ref. [77], "for many real-world problems that are inherently hierarchical, the GTSP offers a more accurate model than the TSP." Generally, GTSP provides a more ideal modeling tool for many real problems. Furthermore, GTSP can include the grouped and isolated vertices at the same time according to our present extension. Therefore, GTSP includes TSP theoretically (see Fig. 3) and application fields of GTSP are wider than those of TSP.

Although since late 1960s GTSP has been proposed [72-74], the related reported works are very limited compared with those on TSP [79–82] and the existing algorithms for GTSP are mainly based on dynamic programming techniques [72-74,76,83-84]. However, because of its NP-hard quality, only a few solutions of modest-size problems are supported by the current hardware technology and most of them fail to obtain the results due to the huge memory required in dynamic programming algorithms and the problem of lengthy computational time.

The main methodology of the dynamic programming algorithms is to transform the GTSP into TSP and then to solve the TSP using existing algorithms [76, 84–86]. The shortcomings of these methods are that the transformation increases the problem dimension dramatically and in some cases the dimension would expand up to more than three times of the original [77, 87-89]. Therefore, although theoretically the GTSP could be solved using the

corresponding transformed TSP, the technological limitation ruins its practical feasibility. Some studies have been performed to discuss and solve the problem [90–92]. This study, we will show some bio-inspired method on the GTSP problem.

### 3.2 Genetic algorithm for generalized TSP

Genetic algorithm (GA) is one of the powerful tools to deal with NP-hard combinatorial optimization problems and has been widely applied for finding the solution of TSP due to its high efficiency and strong searching ability. However, theoretical and application studies related to using GA methods to solve GTSP are very few. The [90] and [93] are two of most interesting work on this problem. In [90], a hybrid GTSP solving algorithm is proposed based on random-key GA and local search method, the main difficult of the method it is hard to handle large scale problems. In [93], a generalized chromosome is used and a generalized chromosome- based GA (GCGA) is proposed accordingly. The advantages of the GCGA are that it does not require the transformation from GTSP to TSP and remove the limitation of triangle inequality of the cost matrix, which enables the GCGA to be able to run with high efficiency.

#### 3.2.1 Generalized chromosome

The solution of GTSP is a special Hamiltonian cycle, which passes through all of the groups. The encoding for solution of GTSP is designed similarly to the one proposed by Huang et al. [94]. A hybrid encoding, which includes a head encoded with binary number and a body encoded with integer number, is given for the solution as figure 1 shows.

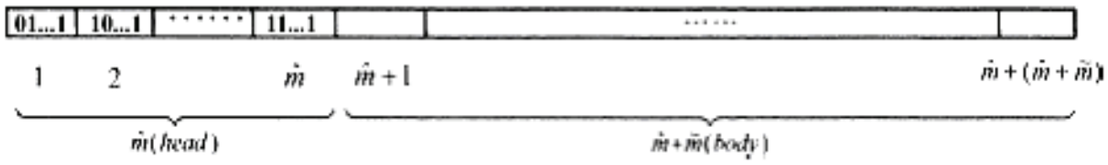


Figure 4. Hybrid Encoding for Solution of GTSP

In the body, there are $m$ integer elements representing $m$ groups ($m = \hat{m} + \tilde{m}$, $\hat{m}$ supper vertexes and $\tilde{m}$ scattering vertexes[93]. In the head, there are $\hat{m}$ binary elements representing vertexes in groups.

Let $[e_1,...,e_i,...,e_{\hat{m}},g_1,...,g_j,...,g_m]$ be a GTSP solution, where $e_i$ (encoded in binary number) is the sequence number of the vertex in Group i, and $g_j$ (encoded in integer number) is the sequence number of the jth group in the cycle. The set of hybrid encoding can be denoted by $D = \{x | x = h \oplus b, h \in H, b \in B\}$, and each solution for GTSP can be encoded as $x \in D$, where $H = \{h \mid h = e_1,...,e_{\hat{m}}; e_i \le |V_i|, i \le \hat{m}\}$ represents the head and $B = \{b \mid b = g_1,...,g_m\}$) represents the body.

#### 3.2.2 The framework of the GCGA

The special designed operators are needed to conduct random search on the generalized chromosome. The GCGA contains the following four operators: Initializing operator **P**, Generalized crossover operator **C**, Generalized crossover operator **C** and Generalized

reversion operator **R**. We give a brief introduction to these five steps in this section. More information about the GCGA can refer to [93].

**1. Initializing operator P**

Initializing operator **P** is used to generate an initial population. It is a two-element random operator. Its two variables are H and B, and its result is a subset of D. Denoting P as a population, then the initialization of P can be represented as $P_N$=($H$,$B$), where PN is an operator to randomly generate an initial population with size $N$.

**2. Generalized crossover operator C**

To implement the crossover operation and generate new chromosomes, a generalized crossover operator is defined as $C$:$D$×$D$→ $D$×$D$. It is a two-element random operator. Its variables are the elements of D. The behavior of the operator is somewhat similar to the two-point crossover in the standard GA. Let the two crossover points selected randomly be $i_1$ and $i_2$ (assume $i_1 < i_2$), where $i_1 = random(\hat{2m} + \tilde{m})$, and $i_2 = random(\hat{2m} + \tilde{m})$. If $i_1 > \hat{m}$ then the crossover takes place in the body parts. In this case, the effect of crossover operator is equal to the conventional crossover in some extent, because the body parts of GC are equivalent to two normal chromosomes. If $i_2 \le \hat{m}$, then the crossover takes place in the head parts. In this case, it is only needed to exchange the genes within the crossover segments. If $i_1 \le \hat{m} < i_2$, then the generalized crossover can be treated as the combination of the above cases.

**3. Generalized mutation operator M**

To increase the diversity of the gene segments, the generalized mutation operator M is designed based on the insertion mutation used in standard GA. Preliminary gene $i_2 = random(\hat{2m} + \tilde{m})$ is randomly selected, which is taken as the gene to be mutated. The difference between GCGA and standard GA is that if $i < \hat{m}$ then the preliminary gene lies in the head part and its corresponding body part also need to be generated.

**4. Generalized reversion operator R**

To enhance the convergent speed of the GCGA, the generalized reversion operator is designed which is similar to the conventional reversion operation. Operator **R** can be used to select two reversion points $i_1$ and $i_2$ according to $i_1 = random(\hat{m} + \tilde{m})$, and $i_2 = random(\hat{m} + \tilde{m})$. If the solution generated after the reversion operator, then the operator R is taken, otherwise the operator won't taken.

### 3.3 Improved Evolutionary Algorithm (EA) for GTSP

#### 3.3.1 The framework of EA for GTSP

In this section, an improved EA for the GTSP (EA-GTSP) has been proposed. In the EA-GTSP, the generalized chromosome described in 3.2 is used to encode the problem. And the following three operators are specially designed to improve the efficiency of the algorithm on the GTSP: crossover operator, mutation operator and local optimization strategy.

**a. Crossover**

At Step 3, pairs of solutions may be selected to carry out the crossover operator by the crossover probability $P_c$. Given two solutions $S_x = h_x \oplus b_x$ and $S_y = h_y \oplus b_y$ selected at Step 3 ( $h_x$, $h_y \in H$.$b_x$,$b_y \in B$), the process of crossover can be shown as follows:

Two integer numbers $i_1, i_2$ ($i_1, i_2 \le \hat{m} + (\hat{m} + \tilde{m})$, $i_1 < i_2$) are generated randomly to set the crossing position. If $i_1 > \hat{m}$, then, $b_x \otimes b_y \to (b'_x, b'_y)$, which is the same operator as the GCGA, $x' = h_x \oplus b'_x$, $y' = h_y \oplus b'_y$. If $i_2 \le \hat{m}$, $h_x \otimes h_y \to (h'_x, h'_y)$, $x' = h'_x \oplus b_x$, $y' = h'_y \oplus b_y$. If $i_1 < \hat{m}$ and $i_2 \ge \hat{m}$, $h_x \otimes h_y \to (h'_x, h'_y)$ and $b_x \otimes b_y \to (b'_x, b'_y)$, $x' = h'_x \oplus b'_x$ and $y' = h'_y \oplus b'_y$.

If the GTSP solution $S_x$ costs less than $S_y$ ($i_2 \le \hat{m}$),

$$h'_x = \{e_{x1}, ..., e_{xi_1-1}, e_{xi_1}.\text{AND}.e_{yi_1}, ..., e_{xi_2}.\text{AND}.e_{yi_2}, e_{xi_2+1}, ..., e_{x\hat{m}}\}$$

$$h'_y = \{e_{x1}, ..., e_{xi_1-1}, e_{xi_1}.\text{OR}.e_{yi_1}, ..., e_{xi_2}.\text{OR}.e_{yi_2}, e_{xi_2+1}, ..., e_{x\hat{m}}\};$$

otherwise,

$$h'_x = \{e_{y1}, ..., e_{yi_1-1}, e_{xi_1}.\text{AND}.e_{yi_1}, ..., e_{xi_2}.\text{AND}.e_{yi_2}, e_{yi_2+1}, ..., e_{y\hat{m}}\}$$

$$h'_y = \{e_{y1}, ..., e_{yi_1-1}, e_{xi_1}.\text{OR}.e_{yi_1}, ..., e_{xi_2}.\text{OR}.e_{yi_2}, e_{yi_2+1}, ..., e_{y\hat{m}}\}.$$

If the GTSP solution $S_x$ costs less than $S_y$ ($i_1 < \hat{m}$ and $i_2 \ge \hat{m}$),

$$h'_x = \{e_{x1}, ..., e_{xi_1-1}, e_{xi_1}.\text{AND}.e_{yi_1}, ..., e_{x\hat{m}}.\text{AND}.e_{y\hat{m}}\}$$

$$h'_y = \{e_{x1}, ..., e_{xi_1-1}, e_{xi_1}.\text{OR}.e_{yi_1}, ..., e_{x\hat{m}}.\text{OR}.e_{y\hat{m}}\};$$

otherwise,

$$h'_x = \{e_{y1}, ..., e_{yi_1-1}, e_{xi_1}.\text{AND}.e_{yi_1}, ..., e_{x\hat{m}}.\text{AND}.e_{y\hat{m}}\}$$

$$h'_y = \{e_{y1}, ..., e_{yi_1-1}, e_{xi_1}.\text{OR}.e_{yi_1}, ..., e_{x\hat{m}}.\text{OR}.e_{y\hat{m}}\}.$$

**b. Mutation**

The mutation operator is added to help EA-GTSP converge to the global optimal solution. Each solution is affected by the mutation operator by probability $P_m$. There are two procedures called head mutation and body mutation in the operator.

In the head mutation, given a head of a solution, the procedure of head mutation is:

Head mutation: $h_z \to h'_z$, $h'_z = \{e_{z1}, ..., e_{zi_3-1}, rebuild(e_{zi_3}), e_{zi_3+1}, ..., e_{z\hat{m}}\}$

where $h_z = \{e_{z1}, ..., e_{zi_3-1}, e_{zi_3}, e_{zi_3+1}, ..., e_{z\hat{m}}\}$. $rebuild(e_{zi_3})$ will generate a segment of binary bits randomly. Every binary element of solution $S_Z$ may be affected by $rebuild(e_{zi_3})$ when $r_{mh} < P_{mh}$ ($r_{mh}$ is generated randomly in [0,1] for each binary element of the solution obtained at Steps 3 and 4).

In the body mutation, the procedure is described as follows:

Body mutation: $b_z \rightarrow b'_z, b'_z = \{g'_{z1}, ..., g'_{zi-1}, g'_{zi}, g'_{zi+1}, ..., g'_{z\hat{m}}\}$

where $b_z = \{g_{z1}, ..., g_{zi}, ..., g_{z\hat{m}}\}$ and $r_{mb} < P_{mb}$ ($r_{mb}$ is generated randomly in [0,1] for each solution obtained at Steps 3 and 4).

So the mutation operator of the EA-GTSP is defined as follows:

Mutation of EA-GTSP: $S_z = h_z \oplus b_z \xrightarrow{mutation} S_z' = h'_z \oplus b'_z$.

**c. Local Optimal Strategy**

The local optimal strategy is helpful to find the best solution in a local searching space. Each solutions of the population are optimized according to a heuristic algorithm as follows:

Input: GTSP solution $S_q$
For $i$ =1 to $\hat{m}$ do //optimization for head
  Choose a vertex in Group i to make $S_q$ cost the lest
End for
For $j$ =1 to $\hat{m} + \tilde{m}$ -1 do //optimization for body
  Choose an order for $g_{qj}$and $g_{qj+1}$ to make $S_q$ cost the least.
End for
Output: a new solution $S'_q$ ($S_q$ is changed into $S'_q$.)

**d. Decoding for solution of GTSP**

Because the head encoding is designed as binary number, it needs to be decoded in the following function.

$$h = [e_1, \ldots, e_{\hat{m}}] \xrightarrow{decoding} [e_1 . \text{MOD} . |V_1|, \ldots, e_i . \text{MOD} . |V_i|, ..., e_{\hat{m}} . \text{MOD} . |V_{\hat{m}}|]$$

where $|V_i|$ is the number of vertexes in Group i ($V_i$).

Until now, we can summarize the algorithm of the improved EA for the GTSP as follows.

Initialize parameters.
Encode and initialize a population of solutions.
/*$\beta$ is chosen between 0 and 5 randomly, $q_0$=0.6 */
**Loop** /* at this level each loop is called iteration */
  **Crossover Operator:** select pairs of solutions and change them into pairs of new Local solutions with the crossover operator by the crossover probability.
  **Optimal Strategy:** optimize all of the solutions with a heuristic algorithm locally.
  **Mutation Operator:** select several solutions by the mutation probability and change
End_condition
Decoding for solution of GTSP

### 3.3.2 Numerical result

In this section, the efficiency of the EA-GTSP and other algorithms are compared on some benchmark problems [93].

| Problem \five runs | EA-GTSP Best | EA-GTSP Average | GCGA Best | GCGA Average | HCGA Best | HCGA Average |
|---|---|---|---|---|---|---|
| 30KROA150 | 11018 | 11018 | 11018 | 11022 | 11018 | 11018 |
| 30KROB150 | 12195 | 12195 | 12196 | 12314 | 12195 | 12195 |
| 31PR152 | 51573 | 51573 | 51586 | 53376 | 51573 | 51573 |
| 32U159 | 22664 | 22664 | 22664 | 22685 | 22664 | 22664 |
| 40KROA200 | 13408 | 13408 | 13408 | 13617 | 13408 | 13408 |
| 40KROB200 | 13113 | 13114 | 13120 | 13352 | 13113 | 13119 |
| 45TS225 | 68340 | 68403 | 68340 | 68789 | 68340 | 68432 |
| 46PR226 | 64007 | 64007 | 64007 | 64574 | 64007 | 64007 |
| 53GIL262 | 1011 | 1011 | 1011 | 1057 | 1011 | 1011 |
| 53PR264 | 29546 | 29546 | 29549 | 29791 | 29546 | 29546 |
| 60PR299 | 22617 | 22631 | 22638 | 22996 | 22631 | 22638 |
| 64LIN318 | 20769 | 20799 | 20977 | 22115 | 20788 | 20914 |
| 80RD400 | 6446 | 6480 | 6465 | 6604 | 6456 | 6498 |
| 84FL417 | 9663 | 9663 | 9663 | 9725 | 9663 | 9663 |
| 88PR439 | 60099 | 60249 | 61273 | 62674 | 60184 | 60558 |
| 89PCB442 | 21695 | 21735 | 21978 | 22634 | 21768 | 21860 |

Table 7. Comparison of solution among EA-GTSP, GCGA and HCGA

The instances can be obtained from TSPLIB library which were originally generated for testing standard TSP algorithms. To test GTSP algorithms, Fischetti et al. [95] provided a partition algorithm to convert the TSP instances to GTSP instances.

In our experiments, we set the population size as 100 (*pop_size*=100), crossover probability as 0.5 ($P_c = 0.5$), and mutation probability as 0.09 ($P_m$=0.09, $P_{mh}$=0.001, $P_{mb}$=0.005). The algorithms would stop when no better solution could be found in 500 iterations. All of the instances are computed by EA-GTSP, HCGA [94] and GCGA [93] twenty times on a PC with 2.0 GHz processor and 256 MB SDR memory, and the results are shown in Table 1.

In Table 7, not only the best solution obtained by EA-GTSP is shorter than the one obtained by HCGA and GCGA does, but also the one on average, in all of the examples. It can show global optimal function of EA-GTSP. In order to show the performance of EA-GTSP, there is a comparison between it and several heuristic algorithms [96] by computing the same GTSP instances. As Table 2 shows, EA-GTSP is more efficient and steady than all of the test algorithms because it can get the best solution in most of the instances.

| Problem\five runs | EA-GTSP | NN (G-opt) | NN (G2-opt) | CI (G-opt) | CI (G2-opt) | MO (G-opt) | MO (G2-opt) | CI[2] | GI[3] |
|---|---|---|---|---|---|---|---|---|---|
| 30KROA150 | 11018 | 11018 | 11018 | 11018 | 11018 | 11018 | 11018 | 11018 | 11018 |
| 30KROB150 | 12195 | 12196 | 12196 | 12196 | 12196 | 12196 | 12196 | 12196 | 12196 |
| 31PR152 | 51573 | 52506 | 52506 | 51915 | 51915 | 51820 | 51820 | 51820 | 51820 |
| 32U159 | 22664 | 23296 | 23296 | 22664 | 22664 | 22923 | 22923 | 23254 | 23254 |
| 40KROA200 | 13408 | 14110 | 14110 | 14059 | 14059 | 13887 | 13887 | 13406 | 13406 |
| 40KROB200 | 13113 | 13932 | 13111 | 13117 | 13117 | 13117 | 13117 | 13111 | 13111 |
| 45TS225 | 68340 | 68340 | 68340 | 69279 | 69279 | 68756 | 68756 | 68756 | 68756 |
| 46PR226 | 64007 | 65811 | 65395 | 65395 | 65395 | 64007 | 64007 | 64007 | 64007 |
| 53GIL262 | 1011 | 1077 | 1032 | 1036 | 1036 | 1021 | 1021 | 1064 | 1064 |
| 53PR264 | 29546 | 31241 | 31241 | 31056 | 31056 | 30779 | 30779 | 29655 | 29655 |
| 60PR299 | 22617 | 24163 | 23069 | 23119 | 23119 | 23129 | 23129 | 23119 | 23119 |
| 64LIN318 | 20769 | 22233 | 21787 | 21858 | 21858 | 22403 | 22403 | 21719 | 21719 |
| 80RD400 | 6446 | 7083 | 6614 | 6550 | 6550 | 6546 | 6546 | 6439 | 6439 |
| 84FL417 | 9663 | 9754 | 9754 | 9662 | 9662 | 9697 | 9697 | 9932 | 9697 |
| 88PR439 | 60099 | 63736 | 62514 | 61126 | 61126 | 62091 | 62091 | 62215 | 62215 |
| 89PCB442 | 21695 | 23364 | 21704 | 23307 | 23307 | 22697 | 22697 | 22936 | 22936 |

Table 8. Comparison of solution among EA-GTSP, GCGA and HCG

## 4. Conclusion and discussions

The chapter introduces two examples of bio-inspired algorithm for traveling sales-man problems and its extended version. The first algorithm, named ant colony optimization (ACO) which is designed inspired by the natural ants' behavior, is a novel method to deal with TSPs. The experimental results prove the performance of ACO approach, which is feasible to solve TSP instances as well as the traditional method. The research results about the self-adaptive parameters of ACO are presented in the chapter, which indicates how to set an optimal ACO algorithm for different TSPs. Another algorithm is genetic algorithm, which is used to solve generalized traveling sales-man problem (GTSP) that is one extended style of TSPs. The best-so-far genetic algorithm for GTSP is introduced in the final sub-section. Bio-inspired algorithms are the feasible methods for TSPs, and can attain better performance with the modified setting like self-adaptive parameters and hybrid coding, which are described in the chapter.

## 5. References

J.H. Holland, Adaptation in Natural and Artificial Systems. MIT Press, Cambridge, MA. 1975

R. Cheng, & M. Gen, Crossover on intensive search and traveling salesman problem. Computers & Industrial Engineering, 27(1–4), pp. 485–488, 1994.

R. Cheng, M. Gen, & M. Sasaki, Film-copy deliverer problem using genetic algorithms. Computers & Industrial Engineering, 29(1–4):pp. 549–553, 1995.

N. Kubota, T. Fukuda, & K. Shimojima, Virus-evolutionary genetic algorithm for a self-organizing manufacturing system. Computers and Engineering, 30(4), 1015–1026, 1996.

D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison–Wesley, Reading, MA.1989.

R. Cheng, M. Gen and Y. Tsujimura, "A tutorial survey of jobshop scheduling problems using genetic algorithms – I. Representation", Computers and Industrial Engineering, pp. 983–997, 1996.

C. R. Reeves, "A genetic algorithm for flowshop sequencing", Computers and Operations Research, 22(1), pp. 5–13, 1995

J. C. Bean, Genetic algorithms and random keys for sequencing and optimization. ORSA Journal on Computing, 6(2), pp.154–160,1994.

L. Davis, Job shop scheduling with genetic algorithms. In: Proceedings of the International Conference on Genetic Algorithms, London, pp. 136–140, 1985.

D.E. Goldberg, R.J. Lingle, Alleles, loci and the traveling salesman problem. In: Proceedings of the International Conference on Genetic Algorithms, London, pp.154–159, 1985.

I. Oliver, D. Smith, J. Holland, A study of permutation crossover operators on the traveling salesman problem. In: Proceedings of the Second International Conference on Genetic Algorithms, London, pp. 224–230, 1987.

T. Starkweather, et al., A comparison of genetic sequencing operators. In: Proceedings of the Fourth International Conference on Genetic Algorithms, Los Altos, CA, pp. 69–76, 1991.

F. D. Croce, R. Tadei and G. Volta, "A genetic algorithm for the job shop problem", Computers and Operations Research, 22(1), pp. 15–24, 1995.

G. Syswerda, Uniform crossover in genetic algorithms.In: Proceedings of the Third International Conference on Genetic Algorithms, Los Altos, pp. 502–508, 1989.

M. Yamamura, , T Ono, and S. Kobayashi, Character-preserving genetic algorithms for traveling salesman problem, Journal of Japan Society for Artificial Intelligence, vol. 6.pp. 1049-1059, 1992.

M. Yamamura, T Ono, and S. Kobayashi, Emergent search on double circle TSPs using subtour exchange crossover, in: Proceedings of the Third IEEE Conference on Evolutionary Computation, IEEE Press, Nagoya, Japan, pp. 535-540, 1996.

J. Dzubera and D. Whitley, "Advanced correlation analysis of operators for the traveling salesman problem," in Parallel Problem Solving From Nature—PPSN III, Y. Davidor, H.-P. Schwefel, and R. Männer, Eds. New York: Springer-Verlag, pp. 68–77, 1994.

K. Mathias and D. Whitley, "Genetic operators, the fitness landscape, and the traveling salesman problem," in Parallel Problem Solving From Nature. Amsterdam, The Netherlands: Elsevier, pp. 219–228, 1992.

H. D. Nguyen, I. Yoshihara, and M. Yasunaga, "Modified edge recombination operators of genetic algorithms for the traveling salesman problem," in Proc. 3rd Asia-Pacific Conf. Simul. Evol. and Learn., Nagoya, Japan, pp. 2815–2820, 2000.

T. Starkweather, S. McDaniel, K. Mathias, C. Whitley, and D. Whitley, "A comparison of genetic sequencing operators," in Proc. 4th Int. Conf. Genetic Algorithms, pp. 69–76, 1991.

B. Freisleben and P. Merz, "A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems," in Proc. IEEE Int. Conf. Evol. Comput., pp. 616–621, 1996.

P. Merz and B. Freisleben, "Genetic local search for the TSP: New results,"in Proc. IEEE Int. Conf. Evol. Comput., pp. 159–164, 1997.

——, "Memetic algorithms for the traveling salesman problem," Complex Syst., 13(4), pp. 297–345, 2001.

S. Jung and B. Moon, "The natural crossover for the 2D Euclidean TSP," in Proc. Genetic and Evol. Comput. Conf, pp. 1003–1010, 2001.

Y. Nagata and S. Kobayashi, "Edge assembly crossover: A high-power genetic algorithm for the traveling salesman problem," in Proc. 7th Int. Conf. Genetic Algorithms, pp. 450–457, 1997.

Y. Nagata, "The EAX algorithm considering diversity loss," in Parallel Problem Solving From Nature—PPSN VIII. New York: Springer-Verlag, pp. 332–341, 2004.

H. D. Nguyen, I. Yoshihara, K. Yamamori, and M. Yasunaga, "Greedy genetic algorithms for symmetric and asymmetric TSPs," IPSJ Trans. Math. Modeling and Appl., 43, SIG10 (TOM7), pp. 165–175, 2002.

H. Sengoku and I. Yoshihara, "A fast TSP solver using GA on JAVA," in Proc. 3rd Int. Symp. Artif. Life and Robot , pp. 283–288, 1998.

J. Grefenstette, et al. Genetic algorithms for the traveling salesman problem, in: Proceedings of the First International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Hillsdale, pp. 160-168,1985.

G. Liepins, et al. Greedy genetics, in: Proceedings of the First International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Hillsdale, pp.90-99, 1985.

P.W. Poon, J.N. Carter. Genetic algorithm crossover operators for ordering applications. Computers and Operations Research 22 (1), pp. 135–147,1995.

L. Qu, R. Sun, A synergetic approach to genetic algorithms for solving traveling salesman problem. Information Sciences 117 (3–4), pp. 267–283, 1999.

C. Liaw. A hybrid genetic algorithm for the open shop scheduling problem. European Journal of Operational Research 124 (1), pp. 28–42, 2000.

K. Katayama, H. Sakamoto, H. Narihisa. The efficiency of hybrid mutation genetic algorithm for the traveling salesman problem. Mathematical and Computer Modeling 31 (10–12), pp.197–203, 2000.

R. Knosala, T. Wal. A production scheduling problem using genetic algorithm. Journal of Materials Processing Technology 109 (1–2), 90–95, 2001.

C. Moon, et al., An efficient genetic algorithm for the traveling salesman problem with precedence constraints. European Journal of Operational Research 140 (3), pp. 606–617, 2002.

R. Cheng, M. Gen, Resource constrained project scheduling problem using genetic algorithms. International Journal of Intelligent Automation and Soft Computing, 1996.

K. Katayama and H. Sakamoto. The Efficiency of Hybrid Mutation Genetic Algorithm for the Travelling Salesman Problem. Mathematical and Computer Modelling 31,pp. 197-203,1990.

Arthur E. Carter, Cliff T. Ragsdale. A new approach to solving the multiple traveling salesperson problem using genetic algorithms. European Journal of Operational Research 175, pp. 246–257 , 2006.

H. D. Nguyen, et al. Implementation of an Effective Hybrid GA for Large-Scale Traveling Salesman Problems: IEEE Transactions on Systems, Man and Cybernetics-Part B: Cybernetics, 37(1):92-99,2007

F. Samanlioglu, M.E. Kurz, , & W.G. Ferrell Jr.. A genetic algorithm with random-keys representation for a symmetric multiobjective traveling salesman problem. In: Proceedings of the Institute of Industrial Engineers Annual Conference. Orlando: Florida, 2006.

F. Samanlioglu, M. E. Kurz, W. G. Ferrell Jr., & Tangudu, S. A hybrid random-key genetic algorithm for a symmetric traveling salesman problem. International Journal of Operations Research, 2(1), 47–63, 2007.

F.Samanlioglu et al. A memetic random-key genetic algorithm for a symmetric multi-objective traveling salesman problem. Computers & Industrial Engineering. www.sciencedirect.com, 2008.

M. Dorigo, V. Maniezzo, and A. Colorni. Ant System: Optimization by a Colony of Cooperating Agents. IEEE Transactions on Systems, Man, and Cybernetics-Part B: CYBERNETIC, 26(1), FEBRUARY, 1996.

M. Dorigo. Optimization, Learning and Natural Algorithms (in Italian). PhD thesis, Dipartimento di Elettronica e Informazione, Politecnico di Milano, IT, 1992.

M. Dorigo, G.D. Caro, L.M. Gambardella. Ant algorithms for Discrete Optimization. Massachusetts Institute of Technology, Artificial Life 5, pp. 137-172, 1999.

W.J. Gutjahr. ACO algorithms with guaranteed convergence to the optimal solution. Information Processing Letters 82, pp.145-153, 2002.

M. Dorigo and L.M. Gambardella. Ant colonies for the traveling salesman problem. BioSystems, 43, pp. 73-81, 1997.

M. Dorigo and L.M. Gambardella. Ant Colony System: A cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation, 1(1), pp. 53-66, 1997.

L.M. Gambardella and M. Dorigo. Solving symmetric and asymmetric TSPs by ant colonies. Proceedings of IEEE International Conference on Evolutionary Computation. 1996.

T. Stutzle and H.H. Hoos. MAX-MIN Ant System. Future Generation Computer Systems,16(8), pp. 889-914, 2000.

M. Dorigo, M. Birattari, T. Stutzle. Ant colony optimization. Computational Intelligence Magazine, IEEE, Vol. 1, Nov., pp.28-39, 2006.

W.J. Gutjahr. A graph-based ant system and its convergence. Future Generation Computer Systems 16(9), pp. 873-888, 2000.

T. Stutzle, M. Dorigo. A Short Convergence Proof for a Class of Ant colony Optimization Algorithms. IEEE Transactions on Evolutionary Computation, 6(4), 2002.

M. Dorigo, C. Blum. Ant colony optimization theory: A survey. Theoretical Computer Science, 344, pp. 243-278, 2005.

A. Badr, A. Fahmy. A proof of convergence for Ant algorithms. Information Sciences 160, pp. 267-279, 2004.

S. Fidanova. ACO Algorithm with Additional Reinforcement. M. Dorigo et al. (Eds): ANTS 2002, LNCS 2463, pp. 292-293, 2002.

S. Fidanova. Convergence Proof for a Monte Carlo Method for Combinatorial Optimization Problems. M. Bubak et al. (Eds.): ICCS 2004, LNCS 3039, pp. 523-530, 2004

J. Kennedy and R.C. Eberhart. Particle Swarm Optimisation. In Proceedings of the International Conference on Neural Networks, pp.1942–1948, 1995.

Clerc, M.. Discrete Particle Swarm Optimization, Illustrated by Traveling Salesman Problem. In New Optimization Techniques in Engineering. Springer-Verlag, Berlin , 2004.

Bo Liu, Ling Wang, Yi-hui Jin, and De-xian Huang, An Effective PSO-Based Memetic Algorithm for TSP, In: D.-S. Huang, K. Li, and G.W. Irwin (Eds.): ICIC 2006, LNCIS 345, pp. 1151 – 1156, 2006.

Yong-Qin Tao, Du-Wu Cui, Xiang-Lin Miao, and Hao Chen, An Improved Swarm Intelligence Algorithm for Solving TSP Problem. In D.-S. Huang, L. Heutte, and M. Loog (Eds.): ICIC 2007, LNAI 4682, pp. 813–822, 2007.

Bin Shen, Min Yao, and Wensheng Yi., Heuristic Information Based Improved Fuzzy Discrete PSO Method for Solving TSP. In Computer Science, PRICAI 2006, LNAI 4099, pp. 859 – 863, 2006.

Yuan, Zhenglei, et al. Chaotic Particle Swarm Optimization Algorithm for Traveling Salesman Problem.Automation and Logistics, 2007 IEEE International Conference on 18-21. pp. 1121 – 1124, 2007.

M. Dorigo, V. Maniezzo, and A. Colorni, “Positive feedback as a search strategy,” Dipartimento di Elettronica, Politecnico di Milano, Italy, Tech. Rep. 91-016, 1991.

M. Dorigo, “Optimization, learning and natural algorithms (in italian),” Ph.D. dissertation, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1992.

M. Dorigo and G. Di Caro, “The Ant Colony Optimization meta-heuristic,” in New Ideas in Optimization, D. Corne et al., Eds., McGraw Hill, London, UK, pp. 11–32, 1999.

Sun, J., Xiong, S. W., Guo, F. M.: A new pheromone updating strategy in ant colony optimization, Proceedings of 2004 International Conference on Machine Learning and Cybernetics, 1, pp. 620-625, 2004

Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press, Cambridge, MA. 2004.

G. Reinelt, "TSPLIB. A traveling salesman problem library," ORSA Journal on Computing, 3(4), pp. 376-384, 1991.

K. Socha, J. Knowles, and M. Sampels, "A MAX–MIN ant system for the university timetabling problem," in Proc. ANTS 2002, ser. LNCS, M. Dorigo et al., Eds., vol. 2463, p. 1, Berlin, Germany: Springer Verlag, 2002.
A. L. Henry-Labordere, RIRO B 2, 43, 1969.
J. P. Saskena, CORS J. 8, 185, 1970.
S. S. Srivastava, S. Kumar, R. C. Garg, and P. Sen, CORS J. 7, 7 ,1969.
G. Laporte, A. Asef-vaziri, and C. Sriskandarajah, J. Oper.Res. Soc. 47, 1461, 1996.
M. Fischetti, J. J. Salazar, and P. Toth, Oper. Res. 45, 378, 1997.
Y. N. Lien, E. Ma, and B. W.-S. Wah, J. Chem. Inf. Comput. Sci. 74, 177, 1993.
K. Castelino, R. D'Souza, and P. K. Wright, http://kingkong.me.berkeley.edu/_kenneth/
N. E. Bowler, T. M. A. Fink, and R. C. Ball, Phys. Rev. E 68,036703, 2003.
M. Andrecut and M. K. Ali, Phys. Rev. E 63, 047103, 2001.
T. Munakata and Y. Nakamura, Phys. Rev. E 64, 046127, 2001.
J. Bentner, G. Bauer, G. M. Obermair, I. Morgenstern, and J.Schneider, Phys. Rev. E 64, 036701 2001.
G. Laporte and Y. Nobert, INFOR 21, 61, 1983.
C. E. Noon and J. C. Bean, Oper. Res. 39, 623, 1991.
D. Ben-Arieh, G. Gutin, M. Penn, A. Yeo, and A. Zverovitch,Int. J. Prod. Res. 41, 2581, 2003.
D. Ben-Arieh, G. Gutin, M. Penn, A. Yeo, and A. Zverovitch,Oper. Res. Lett. 31, 357, 2003.
V. Dimitrijevic and Z. Saric, J. Chem. Inf. Comput. Sci. 102,105, 1997.
G. Laporte and F. Semet, INFOR 37, 114, 1999.
C. E. Noon and J. C. Bean, INFOR 31, 39, 1993.
L. V. Snyder and M. S. Daskin, A Random-key genetic algorithm for the generalized traveling salesman problem (Northwestern University, see, l-snyder3@northweatern.edu, m-daskin@northwestern.edu)
O. Jellouli, in IEEE International Conference on Systems, Man, and Cybernetics, 4, pp. 2765–2768, 2001.
Y. Matsuyama, Trans. Inst. Electron., Inf. Commun. Eng. D-II J74D-II, 416, 1991.
C. G., Wu, Y. C., Liang, H. P., Lee, and C., Lu, Generalized chromosome genetic algorithm for generalized traveling salesman problems and its applications for machining, Physical Review E. 69, 1, 2004
Huang H, Yang XW, Hao ZF, Liang YC, Wu CG, Zhao X. Hybrid chromosome genetic algorithm for generalized traveling salesman problems, Lecture Notes in Computer Science 3612: 137-140 2005.
M. Fischetti, J. J. Salazar, and P. Toth, Branch-and-cut algorithm for the symmetric generalized traveling salesman problem, Operations Research. 45(3), pp.378-394, 1997.
J., Renaud, F. F., Boctor, An efficient composite heuristic for the symmetric generalized traveling salesman problem, European Journal of Operational Research 108, pp.571-584, 1998.
Huang H, Yang XW, Hao ZF, Cai RC. A novel ACO algorithm with adaptive parameter, Lecture Notes in Bioinformatics 4115: 12-21 2006.
Huang H, Hao ZF. ACO for continuous optimization based on discrete encoding. Lecture Notes in Computer Science 4150: 504-505 2006.

Huang H, Hao ZF. An ACO algorithm with bi-directional searching rule. Dynamics of Continuous Discrete and Impulsive Systems-Series B-Applications & Algorithms 13: 71-75, 2006.

Hao ZF, Huang H, Zhang XL, Tu K. A time complexity analysis of ACO for linear functions, Lecture Notes in Computer Science 4247: 513-520 2006.

Zhifeng Hao, Han Huang, Yong Qin, Ruichu Cai. An ACO Algorithm with Adaptive Volatility Rate of Pheromone Trail, Lecture Notes in Computer Science 4490: 1167–1170, 2007.


**Traveling Salesman Problem**

Edited by Federico Greco

ISBN 978-953-7619-10-7
Hard cover, 202 pages
**Publisher** InTech
**Published online** 01, September, 2008
**Published in print edition** September, 2008

The idea behind TSP was conceived by Austrian mathematician Karl Menger in mid 1930s who invited the research community to consider a problem from the everyday life from a mathematical point of view. A traveling salesman has to visit exactly once each one of a list of m cities and then return to the home city. He knows the cost of traveling from any city i to any other city j. Thus, which is the tour of least possible cost the salesman can take? In this book the problem of finding algorithmic technique leading to good/optimal solutions for TSP (or for some other strictly related problems) is considered. TSP is a very attractive problem for the research community because it arises as a natural subproblem in many applications concerning the every day life. Indeed, each application, in which an optimal ordering of a number of items has to be chosen in a way that the total cost of a solution is determined by adding up the costs arising from two successively items, can be modelled as a TSP instance. Thus, studying TSP can never be considered as an abstract research with no real importance.



**InTech Europe**
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

**InTech China**
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen