Hindawi Publishing Corporation
Abstract and Applied Analysis
Volume 2014, Article ID 545391, 8 pages
http://dx.doi.org/10.1155/2014/545391

Research Article

Applying Data Clustering Feature to Speed Up Ant Colony Optimization

Chao-Yang Pang,[1] Ben-Qiong Hu,[2] Jie Zhang,[3] Wei Hu,[4] and Zheng-Chao Shan[5]

[1] *College of Computer Science, Sichuan Normal University, Chengdu 610101, China*
[2] *College of Management Science, Chengdu University of Technology, Chengdu 610059, China*
[3] *Department of Control Engineering, Chengdu University of Information Technology, Chengdu 610225, China*
[4] *North Sichuan Preschool Educators College, Guangyuan 628000, China*
[5] *The Personnel Department of Sichuan Normal University, Chengdu 610068, China*

Correspondence should be addressed to Ben-Qiong Hu; hbq402@126.com

Received 23 January 2014; Accepted 4 April 2014; Published 5 May 2014

Academic Editor: Zhiwu Liao

Ant colony optimization (ACO) is often used to solve optimization problems, such as traveling salesman problem (TSP). When it is applied to TSP, its runtime is proportional to the squared size of problem N so as to look less efficient. The following statistical feature is observed during the authors' long-term gene data analysis using ACO: when the data size N becomes big, local clustering appears frequently. That is, some data cluster tightly in a small area and form a class, and the correlation between different classes is weak. And this feature makes the idea of divide and rule feasible for the estimate of solution of TSP. In this paper an improved ACO algorithm is presented, which firstly divided all data into local clusters and calculated small TSP routes and then assembled a big TSP route with them. Simulation shows that the presented method improves the running speed of ACO by 200 factors under the condition that data set holds feature of local clustering.

1. Introduction

1.1. Introduction of Ant Colony Optimization (ACO). In 1991, ant colony optimization (ACO) was presented firstly by Colorni et al. [1] and applied to solve TSP firstly by Dorigo et al. [1–3]. Dorigo et al. created a new research topic which is studied by many scholars now.

ACO is essentially a system based on agents that simulate the natural behavior of ants, in which real ants are able to find the shortest route from a food source to their nest, without using visual cues by exploiting pheromone information [2]. Pheromone is deposited when ants are walking on a route. It provides heuristic information for other ants to choose their routes. The more dense the pheromone trail of a route is, the more possibly the route is selected by ants. At last, nearly all ants select the route that has the most dense pheromone trail, and it is the shortest route potentially.

ACO has been applied to solve optimization problems widely and successfully, such as TSP [1–4], quadratic assignment problem [5], image processing [6], data mining [7], classification or clustering analysis [8], and biology [9]. The application of ACO leads the theoretic study of ACO. Gutijahr firstly analyzes the convergence property of ACO [10]. Stutzle and Dorigo prove the important conclusion that if the running time of ACO is long enough, ACO can find optimal solution possibly [11]. The other interesting property is revealed currently by Birattari et al. that the sequence of solutions of some algorithms does not depend on the scale of problem instance [12].

ACO is especially well suited for solving difficult optimization problems, where traditional optimization methods are less efficient. However, ACO is not very efficient in solving large problems because running time is too long and the quality of solution is still low. To solve the two main problems, the configuration of the parameters is discussed [2, 3]. To further improve ACO, many approaches have been proposed. Among these approaches, parallel computation and other methods are used to accelerate ACO [13]. In this study, we design a novel clustering algorithm named special local clustering algorithm (SLC), which is applied to classify and find the solution for TSP problem. Moreover, a colony of ants acts on each class to get a local TSP path. And we use the convergence of route length as termination criterion of ACO. The experimental results indicated that the improved ACO speeds up and its quality becomes higher for testing problems. It is more robust than comparative approaches.

1.2. Clustering Correlates to the Running Time of ACO. One of study focuses of ACO is to cut down running time. The running time of ACO is $O(t_{\max}MN^2)$, and $M = [N/1.5]$ in general, where $t_{\max}$, M, and N denote the iteration number, number of ants, and number of cities, respectively [4]. The running time is proportional to N^2. Cutting down the number of cities N is the key to reduce running time. Therefore, classifying all cities into different classes and letting ACO act on each class will reduce running time heavily. Hu and Huang used this method to improve the running speed of ACO [14], which is named ACO-K-Means. It is faster than ACO by factors of 5–15 approximately. Simulations show that ACO-K-Means algorithm is valid only to the set of cities that has evident clustering feature and invalid to more general situation. ACO-K-Means implies that using clustering method to improve the running speed of ACO is possible.

1.3. Introduction of Local Clustering Algorithm. Clustering is classifying objects of a set (named training set) into different clusters (or groups), so that the data in each class (ideally) share some common traits. One of the most popular clustering algorithms is K-Means clustering algorithm [15, 16]. K-Means clustering algorithm assigns each point to the cluster whose center (i.e., centroid) is nearest to it and then updates the centroid. Repeat this process until termination criterion is satisfied [16].

During the tth iteration of K-Means algorithm, the ith class has distortion that is defined as the average distance of each point and the class centroid, which is denoted by $D_i^{(t)}$ $(1 \le i \le m)$, where m is the number of classes. Pang proves that for each i the distortion sequence $\{D_i^{(t)}\}$ is convergent if the ith class is separated from other classes evidently [16]. That is, distortion sequence is convergent locally. According to this property, an algorithm named local clustering algorithm (LC) is presented [17], and its essential idea is introduced as below.

Step 1. K-Means is applied to a given training set to generate classes.

Step 2. The class whose distortion $D_i^{(t)}$ is convergent first is deleted from training set. Then, update training set such that it is comprised of residual points. Go to Step 1.

Repeat the process of Steps 1 and 2 until all data is classified.

LC algorithm is faster than K-Means algorithm by factors of 4–13 approximately.

Suppose that the ith class is $R_i^{(t)}$ during the tth iteration of K-Means algorithm. Set $R_i^{(t)}$ has entropy $H(R_i^{(t)})$, where $H(R_i^{(t)}) = -\sum_{a\in R_i^{(t)}} p(a)\log_2 p(a)$ and $p(a)$ is the probability of data a. It is proved that entropy sequence $\{H(R_i^{(0)}), H(R_i^{(1)}), \ldots, H(R_i^{(t)}), \ldots\}$ is convergent [16]. That is, the convergent criterion of K-Means algorithm can be replaced by the convergence of entropy sequence [18]. The K-Means with convergent criterion of entropy convergence is fast by factors of 2 at least [18, 19].

2. Improve Local Clustering Algorithm to Generate Compact Class

2.1. Compact Set and the Method of Generation. For any subset of Euclidean space R^n, every sequence in this subset has a convergent subsequence, the limit point of which belongs to the set. This subset is called *compact set*. The conception of compact set (or compactness) is a topology conception. To understand it easily, compactness can be described visually as the phenomenon where many points cluster tightly in a small region, while noncompact set is the set of which most of points cluster loosely in a big region.

K-Means clustering, LC, or other algorithms aim to partition a training set into classes. Some classes are compact and some are not. The most common situation is that a class contains a compact subset and some loose points, and points of the compact subset are around the center of the class. That is, the central part of class is compact possibly. To extract compact subset from a class, the following 3δ-principle is introduced.

For Gauss distribution, suppose that δ denotes the deviation of random data. It is the 3δ-principle that there is more than 99% probability that a random point falls into the central region of data set whose radius is 3δ [16]. The central region contains more than 99% points. Thus, if radius 3δ is small enough and the number of points is big enough, the central region is compact. If the central region with radius 3δ is not compact, shortening the radius of central region to $3\delta/4$, $3\delta/16$, and so on will make it compact. For Gauss distribution which is comprised of enough points, the compact central region always exists. In general, for a class generated by clustering algorithm, all distances of points from class centroid comprise a similar gauss distribution. Therefore, the central region of a class is compact possibly.

Suppose that the ith class is $R_i^{(t)}$ at the tth iteration of K-Means or LC algorithm. With the increase of iteration, class sequence $\{R_i^{(0)}, R_i^{(1)}, \ldots, R_i^{(t)}, R_i^{(t+1)}, \ldots\}$ $(1 \le i \le m)$ appears, where m denotes the number of classes. Let

$$D_i^{(t)} = \frac{1}{\left|R_i^{(t)}\right|} \sum_{x\in R_i^{(t)}} d\left(x, c_i^{(t)}\right), \tag{1}$$

where $|R_i^{(t)}|$ denotes the number of elements in $R_i^{(t)}$ and $d(x, c_i^{(t)})$ denotes distance.

Consider

$$\delta_i^{(t)} = \frac{1}{\left|R_i^{(t)}\right|} \sum_{x\in R_i^{(t)}} \left|d\left(x, c_i^{(t)}\right) - D_i^t\right|. \tag{2}$$

Clearly, $D_i^{(t)}$ is the distortion of class $R_i^{(t)}$ and $\delta_i^{(t)}$ is the approximation of deviation of $D_i^{(t)}$.

Consider

$$K_i^{(t)} = \left\{ x \mid d\left(x, c_i^{(t)}\right) \leq \frac{1}{4p}\left(D_i^{(t)} + 3\delta_i^{(t)}\right), \right. \\ \left. x \in R_i^{(t)} \; (p \geq 0) \right\}. \tag{3}$$

$K_i^{(t)}$ is the central region of class $R_i^{(t)}$. Parameter p is used to shorten the radius of central region $K_i^{(t)}$ and makes it compact. Figure 1 illustrates the 3δ-principle and compact subset $K_i^{(t)}$.

2.2. Subroutine 1: Local Clustering Algorithm with 3δ-Principle. The local clustering algorithm with 3δ-principle is used to classify points into classes and to extract compact central region of classes. Its essential idea is described as below.

Firstly, apply LC algorithm to cluster data. And apply the criterion of entropy convergence (i.e., $|H(R_i^{(t)}) - H(R_{i+1}^{(t+1)})|/H(R_i^{(t)}) \to 0$) to mark the stable class $R_i^{(t)}$.

Secondly, extract compact central region $K_i^{(t)}$ from class $R_i^{(t)}$ and preserve it as a genuine class. Remove $R_i^{(t)}$ from training set and update it. Repeat the above two steps until all compact central regions are extracted. The details are described in Algorithm 1.

2.3. Special LC Algorithm to Generate Compact Classes (SLC). Note that above subroutine 1 is not a partition of training set. Subroutine 1 extracts only compact central regions of all classes and the residual points are unclassified. The residual points comprise a new training set. And it is possible that some of residual points cluster together tightly and comprise some small compact subsets again. These small compact subsets are new classes. To obtain these new classes and classify all points, SLC algorithm is described in Algorithm 2.

2.4. The Clustering for Mixture Distribution (SLC-Mixture). The clustering algorithm SLC presented above generates spherical classes only. However, for a general distribution, some classes are of spherical shape, some classes are of chain shape in which points cluster closely around a curve (or a line), and some classes contain isolated points. This common distribution is called mixture distribution. For a large-scale TSP, the distribution of cities is mixture distribution in general. The clustering method for mixture distribution is proposed as below.

2.4.1. The Simple Maker to Distinguish Spherical Class from Chain-Shaped Class. The position of city on a map is two-dimensional point. A given class can be divided into 8 areas along the 4 directions of the north-south and west-east and two diagonal lines through the centroid of the class. If the class is spherical, the percentage of points in each area is close to 1/8 and is the same approximately. If the class is chain-shaped class (or part of chain-shaped class), it is impossible that the percentage of every area is close to 1/8 at the same time. Therefore, the percentage of points in each area is the maker of spherical class. Figure 2 illustrates the marker.

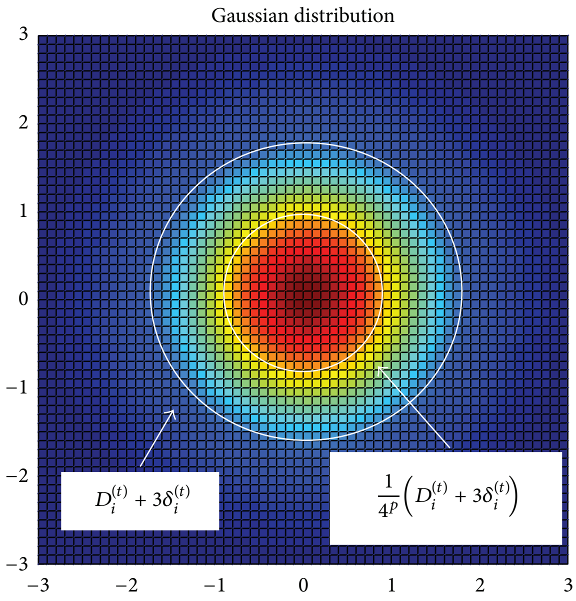

Figure 1: The illustration of compact central region of a class. In a class $R_i^{(t)}$, most of points cluster around their centroid and few points are far away from the centroid. Subset $K_i^{(t)}$ (i.e., the shadow part) is the central region of class $R_i^{(t)}$. Compact set is the set where many points cluster in a small region tightly. Increasing parameter p will shorten radius and make $K_i^{(t)}$ compact.

2.4.2. Applying SLC to Process Mixture Distribution (SLC-Mixture). At first, apply SLC to classify all data of training set. Secondly, apply the marker presented above to distinguish spherical classes and extract them from the training set. Then all residual points comprise a new set named residual set. The residual set contains only chain-shaped classes and isolated points. Thirdly, apply the method presented in [20] to classify all residual points of residual set into different chain-shaped classes or marked as isolated points. The method presented in [20] is named chain-shaped clustering algorithm.

The clustering method presented in this section is called SLC-Mixture algorithm, which processes the mixture distribution of spherical classes, chain-shaped classes, and isolated points.

3. Apply SLC to ACO

3.1. The Termination Criterion of ACO. Suppose ACO acts on a compact class and let L_t denote the minimum route length that is generated at the tth iteration of computation. There is sequence $\{L_1, L_2, \ldots, L_t, L_{t+1}, \ldots\}$ and it is convergent under ideal condition. The convergent criterion $|L_t - L_{t+1}|/L_t \leq \varepsilon$ is proposed as the termination criterion of ACO in this paper.

In the following discussion, ACO refers to the algorithm whose termination criterion is $(|L_t - L_{t+1}|/L_t) \leq \varepsilon$.

3.2. Apply SLC to Improve the Running Speed of ACO (ACO-SLC). In this section, the clustering algorithm SLC will be

```
Input parameters:
$T$: Training Set
$m$: The number of classes
$\varepsilon$: The stop threshold for clustering.
$C^{(0)} = \{c_i^0 (1 \le i \le m)\}$: Initial centroids set.
$p$: A parameter to adjust the size of compact subs-ets $K_i^{(t)} (p \ge 0)$.
Output:
$\varphi(T) = \{K_1^{(t)}, K_2^{(t)}, \dots, K_i^{(t)}, \dots, K_m^{(t)}\}$ (i.e., the set of co-mpact subset, see Figure 1)
$\sigma(T) = \{B_1^{(t)}, B_2^{(t)}, \dots, B_i^{(t)}, \dots, B_m^{(t)}\}$, where $B_i^{(t)} = R_i^{(t)} - K_i^{(t)}$, and it is comprised by dispersive points
$(1 \le i \le m$, see Figure 1)
Void Subroutine 1 $(T, m, \varepsilon, C^{(0)}, p, \varphi(T), \sigma(T))$
{
Step 1. Initialization: Let iteration number $t = 0$. Let $Counter = m$. Let $\varphi(T) = \phi$ and $\sigma(T) = \phi$, where $\phi$
denotes empty set. According to initial centroids set $C^{(0)}$, generate initial partition of training set
$\varphi^0 = \{R_i^0 \mid R_i^0 \subset T, 1 \le i \le m\}$.
Step 2. While ($Counter > 0$) {
  Step 2.1. Generate new centroids set $C^{(t+1)} = \{c_i^{(t+1)} \mid 1 \le i \le m\}$ and new partition $\varphi^{(t+1)} = \{R_i^{(t+1)} \mid 1 \le i \le m\}$

/* Note: Check whether entropy sequence $H_i^{(0)}, H_i^{(1)}, \dots, H_i^{(t)}, H_i^{(t+1)}, \dots$ is convergent. If it is convergent,
let the convergent marker StableMarker $(R_i^{(t+1)}) = True$ */
  Step 2.2. For ($i = 1$; $i \le Counter$; $i++$){
    Estimate the entropy of class $R_i^{(t+1)}$, that is, $H(R_i^{(t+1)}) = \log_2 |R_i^{(t+1)}|$.
    If $\left( \frac{|H(R_i^{(t)}) - H(R_i^{(t+1)})|}{H(R_i^{(t)})} < \varepsilon \right)$ $\{$StableMarker$(R_i^{(t+1)}) = True;\}$
    Else $\{$StableMarker$(R_i^{(t+1)}) = False\}$
  }
/* Note: Extract the data around the centroid of class as a genuine class */
  Step 2.3. For ($i = 1$; $i \le Counter$; $i++$){
    If (StableMarker $(R_i^{(t+1)}) = True$) {
    Calculate compact central region $K_i^{(t)}$ according to formula (3)
    Calculate $B_i^{(t)}$: $B_i^{(t)} = R_i^{(t+1)} - K_i^{(t)}$
    Let $\varphi(T) = \varphi(T) \cup K_i^{(t)}$
    Let $\sigma(T) = \sigma(T) \cup B_i^{(t)}$
    Update Training Set: $T = T - R_i^{(t)}$
    Update centroids set: $C^{(t+1)} = C^{(t+1)} - \{c_i^{(t+1)}\}$
    $Counter = Counter - 1$
  }
 }
 $t = t + 1$
 }
}
```

Algorithm 1

applied to improve the running speed of ACO. The method is named ACO-SLC and it is described as below.

Input parameter:

T: set of cities.

Output: the shortest TSP route obtained by the algorithm.

ACO-SLC Algorithm.

Step 1. Apply SLC algorithm to partition set T. The classes are $B_1, B_2, \dots, B_i, \dots$, and B_{Num}, and their centroids are $b_1, b_2, \dots, b_i, \dots$, and b_{Num}, respectively.

Step 2. Construct graph G': centroids $b_1, b_2, \dots, b_i, \dots$, and b_{Num} are regarded as virtual cities, respectively, and the virtual cities are regarded as the vertices of graph G'. For a pair of classes B_i and B_j, if there exist two cities that belong to B_i and B_j, respectively, and they join each other, use an edge to join the two corresponding vertices b_i and b_j. The weight of edge is the minimum distance between two classes; that is,

$$d(B_i, B_j) = \min \{ d(x_i, x_j) \mid x_i \in B_i, x_j \in B_j \}. \quad (4)$$

Step 3. Calculate a TSP route of graph G' to generate the traveling order of all classes: let ACO algorithm act on graph G' to find a TSP route denoted by $b_{j1}, b_{j2}, \dots b_{j\text{Num}}$, where

```
Input parameters:
$T_0$: Training Set
$m_0$: The initial number of classes.
$\varepsilon$: The stop threshold for clustering.
Output:
Num: The final number of classes.
CLS: The partition of $T_0$, in which each class is com-pact.
SLC Algorithm:
Step 1. Initialization: Let $T = T_0$, $m = m_0$, $\mathrm{CLS} = \phi$, and $p = 0$.
Step 2. For ($i = 0$; $i < [\log_2 m]$; $i$ + +) /*Note: $[\log_2 m]$ denotes the integer */
{ Step 2.1. Generate initial centroids set $C^0 = \{c_i^{(0)} \mid 1 \le i \le m\}$.
  Step 2.2. Call Subroutine1 $(T, m, C^{(0)}, \varepsilon, p, \varphi(T), \sigma(T))$
  Step 2.3. $\mathrm{CLS} = \mathrm{CLS} \cup \varphi(T)$;
  Step 2.4. $T = \sigma(T)$;
  /* Note: Increase $p$ to get smaller compact class */
  Step 2.5. $m = \left[\frac{m}{2}\right]$; $p = p + 1$
}
Step 3. Every residual point $x$ in the last set $\sigma(T)$ is regarded as a class $\{x\}$. And let $\mathrm{CLS} = \mathrm{CLS} \cup \{x\}$.
Let Num denote the number of classes contained in CLS. The two outputs are CLS and Num.
```

Algorithm 2

$j_1, j_2, \ldots j_{\mathrm{Num}}$, is a permutation of sequence $1, 2, \ldots$ Num. The pair of classes B_{ji} and $B_{j(i+1)}$ is called neighbor class.

Step 4. Choose an edge as the bridge to join a pair of neighbor classes, and this edge is named bridge edge. Assume that the two neighbor classes are B_{j1} and B_{j2}. If there exists an edge such that

$$d(x_u, x_v) = \min \left\{ d(a,b) \mid a \in B_{j1}, b \in B_{j2} \right\}, \qquad (5)$$

edge (x_u, x_v) is the bridge edge, x_u and x_v are called border cities, where vertices a and b should be not used to join other neighbor classes.

Step 5. Calculate a local TSP route for every class B_i ($1 \le i \le$ Num): add a new edge to join the two border cities in the class and mark the edge as necessary edge of the local TSP route. This edge is named pseudoedge. Let the ACO algorithm with convergence criterion $(|L_t - L_{t+1}|/L_t) \le \varepsilon$ act on the class to generate a local TSP route.

Step 6. Construct a TSP route: walk along the traveling order obtained at Step 3; for every pair of neighbor classes, delete the pseudoedge of each class such that the local route is not close. Then let the local route of each class and the bridge edge between these two classes be joined.

Figure 3 illustrates the processing of ACO-SLC algorithm.

3.3. Using the Method of Little-Window and Removing Cross-Edge to Improve ACO-SLC (ACO-SLC-LWCR). Clustering may cause the error of solution although it improves the running speed of ACO heavily. If all classes are compact and separated clearly, the quality of solution of ACO-SLC should be very good. However, in fact, the border between two neighbor classes is fuzzy. The fuzzy border will cause the inaccuracy of solution, and much longer route will appear. And recognizing the longer part and removing it will generate better solution possibly. It is well known that the shortest route is always at the surface of a convex hull. Thus, the longer part should be at the inner of a convex hull and two longer edges intersect. In other words, intersection of two edges is a marker of longer part of a route possibly. According to the marker, removing longer edges is called *removing cross-edge* or *removing intersection edges*, which is similar to the method in [4]. (Notice: in [4], before executing ACO, the long and crossed edges are removed to improve the running speed of ACO, not to improve the solution quality.)

Figure 4 illustrates the method of removing cross-edge.

In addition, a simple method named *little-window strategy* is proposed to improve the running speed of ACO in [21]. Construct a set S_i that is comprised by w accessible and short edges which join the ith city, where w is a preassigned constant. The ant which has arrived at ith city will select an edge from window set S_i only to arrive at its next city and not select an edge from all neighbor edges of this vertex. So, this method improves the running speed of ACO.

The ACO-SLC with little-window strategy and cross-edges removing is called *ACO-SLC-LWCR*.

3.4. The ACO-SLC for Mixture Distribution (ACO-SLC-Mixture). ACO-SLC is suitable for the spherical shape distribution only, and the low quality of solution will appear possibly when ACO-SLC is applied to process mixture distribution. To process mixture distribution, the following method named ACO-SLC-Mixture is proposed in this paper.

Firstly, apply SLC-Mixture at Section 2.4.2 to partition the set of cities into spherical classes, chain-shaped classes, or isolated points. Secondly, apply ACO-SL-C-LWCR to each class and generate a TSP route.

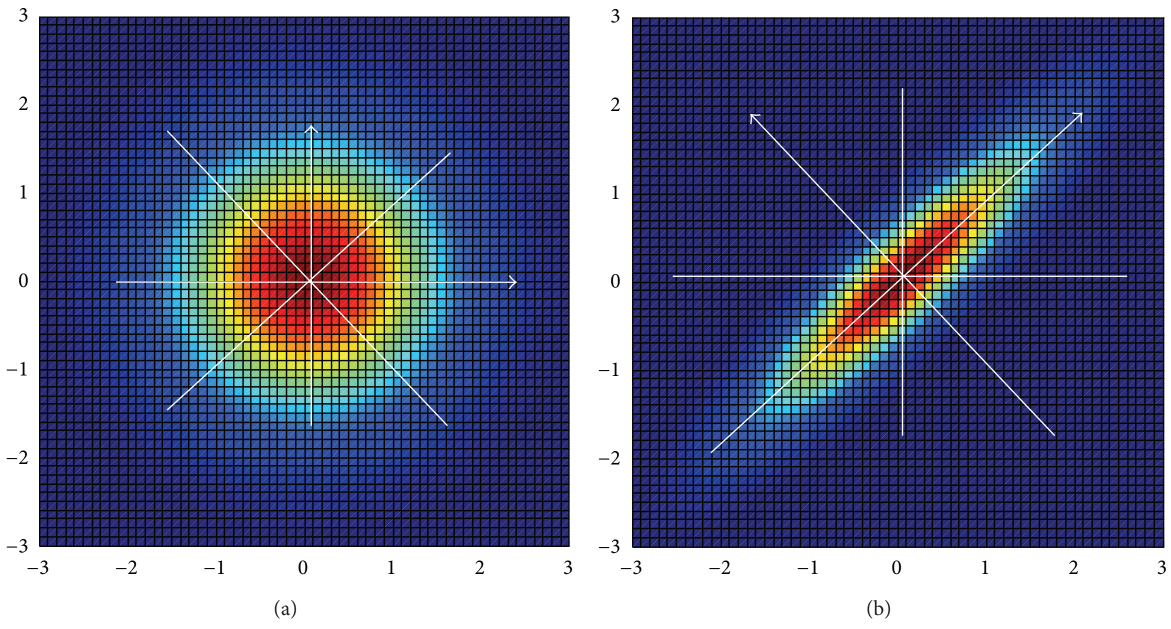

Figure 2: The illustration of distinguishing spherical class from chain-shaped class. A given class is divided into 8 parts along the 8 lines through the centroid of the class. If the class is spherical, the percentage of each part is close to threshold $\varepsilon = 1/8$. If the class is chain-shaped class (or part of chain-shaped class), there are 2–4 parts whose percentage is far less than 1/8. Therefore, the percentage of each part is the marker of spherical class. $\varepsilon = 0.058$ in this paper because some classes are elliptical.

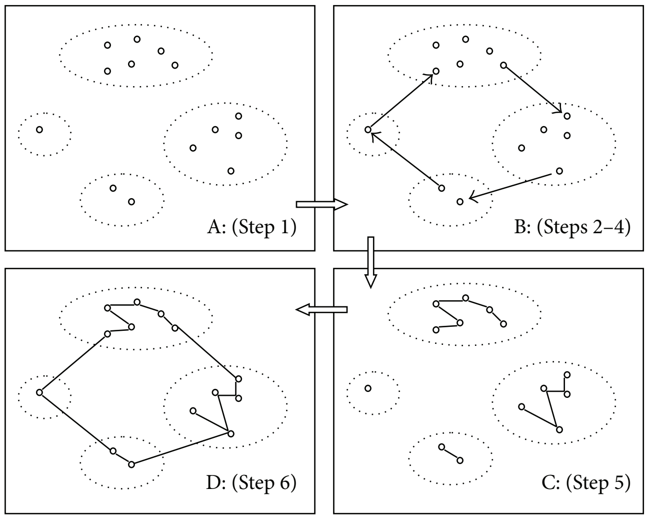

Figure 3: The schematic diagram of ACO-SLC. Firstly, classify all points into compact classes. Secondly, the centroid of each class is regarded as a virtual city; calculate a virtual TSP route. Then along the virtual route, join all classes. Thirdly, let ACO act on each class to get a local TSP route. Fourthly, join all local TSP routes along the virtual route to form the last route.

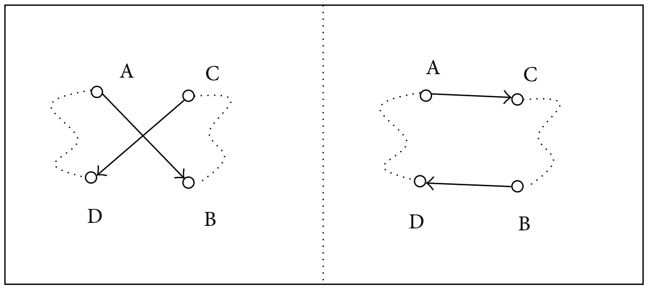

Figure 4: The schematic diagram of ACO-SLC. Firstly, classify all points into compact classes. Secondly, the centroid of each class is regarded as a virtual city. And calculate a virtual TSP route. Then along the virtual route, join all classes. Thirdly, let ACO act on each class to get a local TSP route. Fourthly, join all local TSP routes along the virtual route to form the last route. The illustration of removing cross-edges from TSP route: at the left figure, AB and CD intersect each other. There is a principle that the shortest route is at the surface of a convex hull. Thus, edges AB and CD are the longer part of route and should be removed. Removing these two edges will generate shorter route (see right figure).

4. Simulation

In this section, five related algorithms ACO, ACO-K-Means, ACO-SLC, ACO-SLC-LWCR, and ACO-SLC-Mixture are tested and compared. In the following simulation, ACO refers to ant-cycle presented by Colorni et al., which is very typical [1].

All test data in this paper is downloaded from http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/. All algorithms in this paper run on personal computer (CPU: 1.80 GHz; memory: 480 M; software: Matlab). The parameters are listed as below. Initialize pheromone trails $\tau_{ij}(0) = 1$, iteration number 1000, $\varepsilon = 0.001$, $\alpha = 1$, $\beta = 10$, $\rho = 0.4$, $Q = 300$, and $m = [N/1.5]$. Two performance items are tested. One item is the running time, which is defined as *Ratio = Time (ACO)/Time (Algorithm)*. The bigger the ratio is, the faster

The ratio of running speed
Ratio = Time (ACO)/Time (Algorithm)

	ACO	ACO-K -Means	ACO-SLC	ACO-SLC -LWCR	ACA-SLC -Mixture
Pr107	1.00	11.86	575.15	577.43	539.69
Ch130	1.00	13.38	754.98	765.72	465.26
Pr136	1.00	7.12	466.42	465.56	437.81
D198	1.00	5.05	415.32	390.08	257.72
Pr226	1.00	8.00	1347.06	1152.37	1346.84
Lin318	1.00	15.24	1298.11	1671.64	1630.29
P654	1.00	109.52	10735.77	10191.92	9419.74

Solution error (%)
Error = (Solution−Optimum)/Optimum

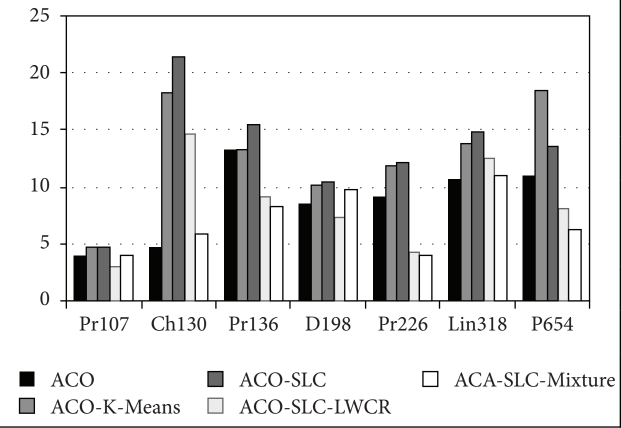

Figure 5: The performance comparison with five algorithms. The figure shows that ACO-SLC algorithm, ACO-SLC, ACO-SLC-LWCR, and ACO-SLC-Mixture are faster than ACO by 415 ~ 10736, 390 ~ 10192, and 257–9419 of factors, respectively! However, some solutions of ACO-K-Means and ACO-SLC have low quality. ACO-SLC-Mixture can process mixture distribution and its inaccuracy ratio is less than ACO in most cases and is bigger than ACO by 2% at most. It should be noted that under the condition where the data set holds feature of local clustering significantly, the quality of solution is good.

the algorithm is. In addition, the advantage of the ratio is that the subtle infection of other processes to runtime is evaded as possible, and it is more accurate than raw measured runtime because the value caused by other processes gives little contribution to the ratio. The other item is the quality of solution, which is defined as the percentage of error *Error = (Solution-Optimum)/Optimum*, where *Optimum* denotes the best solution known currently. The smaller the error is, the better the quality of solution is.

The performances of the five algorithms are listed in Figure 5. It shows that ACO-SLC, ACO-SLC-LWCR, and ACO-SLC-Mixture are faster than ACO by 415 ~ 10736, 390 ~ 101–92, and 257–9419 of factors, respectively! However, some solutions of ACO-K-Means and ACO-SLC have low quality. The inaccuracy ratio of ACO-SLC-Mixture is less than ACO in most cases and is bigger than ACO by 2% at most.

The Defect of ACO-SLC. (1) From Figure 5, it should be noted that only under the condition that the data set holds feature of local clustering significantly, the quality of solution is good. (2) The simulations of this paper show that the quality of ACO-SLC solution depends on the quality of clustering and clustering quality of SLC is sensitive to the initial centroids just like K-Mean algorithm. This is the main defect of ACO-SLC.

5. Conclusion

Time Complexity of ACO. ACO is the algorithm that is inspired by the foraging behavior of ant colonies and has been applied to solve many optimization problems. The typical application of ACO is the application at traveling salesman problem (TSP). The running time of ACO is $O(t_{\max}MN^2)$, where $t_{\max}$, M, and N denote the iteration number, number of ants, and number of cities, respectively. Parameter m is an experiential value and is set to $[N/1.5]$ in general. Parameter N is the key factor of running time because running time is proportional to its square. Parameter $t_{\max}$ and N are available, and decreasing parameter $t_{\max}$ and N will cut down running time.

Focus of ACO Study. ACO can generate solution with high quality in general. But its shortage is that running time is too long. Cutting down running time is one of study focuses of ACO, and one way is to decrease parameters $t_{\max}$ and N, especially N.

Basic Idea for this Study Focus. For this study focus, the following basic idea is presented in this paper.

Firstly, all cities are classified into compact classes, where compact class is the class where all cities in this class cluster tightly in a small region. Secondly, let ACO act on every class to get a local TSP route. Thirdly, all local TSP routes are joined to form solution. Fourthly, the inaccuracy of solution caused by clustering is eliminated.

Realization of Basic Idea. The realization of above idea is based on a novel clustering algorithm presented in this paper, which is named special local clustering algorithm (SLC). The running time of SLC is far less than the time of ACO. SLC generates compact classes, while current popular clustering algorithm such as K-Means does not generate compact classes in general. The compactness of class makes the length of TSP route L_t at every iteration convergent; the convergence of L_t (i.e., $(|L_t - L_{t+1}|/L_t) \rightarrow 0$) is proposed as the termination criterion of ACO in this paper. Thus, parameter $t_{\max}$ is cut down to improve the running speed of ACO. In addition, every class has small size; ACO acting on small class makes parameter N cut down, and running speed is improved.

According to this analysis, ACO-SLC algorithm is presented in this paper. Simulation shows that ACO-SLC is faster than ACO by 415 ~ 10736 of factors!

Elimination of the Solution Inaccuracy Caused by Clustering. Although the running speed is improved in this paper, the inaccuracy of solution is heavy. Two factors causing the inaccuracy are found in this paper. One is the cross-edges (see Section 3.3) and the other factor is the unmatching between ACO-SLC and mixture distribution (see Section 3.4). According to these two factors, ACO-SLC-LWCR and ACO-SLC-Mixture are presented in this paper, which is the improvement of ACO-SLC. Simulation shows that ACO-SLC-LWCR and ACO-SLC-Mixture are faster than ACO by 390 ~ 101–92 and 257–9419 of factors, respectively! The inaccuracy ratio of ACO-SLC-Mixture is less than ACO in most cases and is bigger than ACO by 2% at most.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The work is supported by Aeronautical Science Foundation of China (no. 2012ZD11) and partially by the Education Department of Sichuan Province (no. 12zA134 and no. 09zz028).

References

[1] A. Colorni, M. Dorigo, and V. Maniezzo, "Distributed optimization by ant colonies," in *Proceedings of the 1st European Conference on Artificial Life*, pp. 134–142, Paris, France, 1991.

[2] M. Dorigo and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, 1997.

[3] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics B: Cybernetics*, vol. 26, no. 1, pp. 29–41, 1996.

[4] H. B. Duan, *Ant Colony Algorithms: Theory and Applications*, Science Publisher, Beijin, China, 2005.

[5] V. Maniezzo and A. Colorni, "The ant system applied to the quadratic assignment problem," *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, no. 5, pp. 769–778, 1999.

[6] S. Meshoul and M. Batouche, "Ant colony system with extremal dynamics for point matching and pose estimation," in *Proceeding of the 16th International Conference on Pattern Recognition*, pp. 823–826, Quebec, Canada, 2002.

[7] R. S. Parpinelli, H. S. Lopes, and A. A. Freitas, "Data mining with an ant colony optimization algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 4, pp. 321–332, 2002.

[8] X. Li, X. H. Luo, and J. H. Zhang, "Codebook design by a hybridization of ant colony with improved LBG algorithm," in *Proceedings of the International Conference on Neural Networks and Signal Processing (ICNNSP '03)*, pp. 469–472, Nanjing, China, December 2003.

[9] P. Meksangsouy and N. Chaiyaratana, "DNA fragment assembly using an ant colony system algorithm," in *Proceedings of the Congress on Evolutionary Computation*, pp. 1756–1763, Canberra, Australia, 2003.

[10] W. J. Gutjahr, "Graph-based ant system and its convergence," *Future Generation Computer Systems*, vol. 16, no. 8, pp. 873–888, 2000.

[11] T. Stutzle and M. Dorigo, "A short convergence proof for a class of ant colony optimization algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 4, pp. 358–365, 2002.

[12] M. Birattari, P. Pellegrini, and M. Dorigo, "On the invariance of ant colony optimization," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 732–742, 2007.

[13] B. Bullnheimer, G. Kotsis, and C. Strauß, "Parallelization strategies for the ant system," in *High Performance and Algorithms and Software in Nonlinear Optimization*, vol. 24 of *Applied Optimization*, pp. 87–100, 1998.

[14] X.-B. Hu and X.-Y. Huang, "Solving TSP with characteristic of clustering by ant colony algorithm," *Journal of System Simulation*, vol. 16, no. 12, pp. 55–58, 2004.

[15] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantization design," *IEEE transactions on communications systems*, vol. 28, no. 1, pp. 84–95, 1980.

[16] C. Y. Pang, *Quantization and image compression [Ph.D. thesis]*, University of Electronic Science and Technology of China, Chengdu, China, 2002.

[17] P. Chaoyang, S. Shixin, P. Ye, and G. Haiying, "A fast codebook training algorithm using local clustering," *Journal of Electronics and Information Technology*, vol. 9, pp. 1282–1286, 2002.

[18] C.-Y. Pang and S.-X. Sun, "Codebook training algorithm by the convergence of entropy sequence for vector quantization," *Systems Engineering and Electronics*, vol. 24, no. 1, pp. 83–85, 2002.

[19] X. Li, X. H. Luo, and J. H. Zhang, "Modeling of vector quantization image coding in an Ant colony system," *Chinese Journal of Electronics*, vol. 13, no. 2, pp. 305–307, 2004.

[20] X. Huang, J. Wang, and Y. Zhang, "Adaptive K near neighbor clustering algorithm for data with non-spherical-shape distribution," *Computer Engineering*, vol. 29, pp. 21–22, 2003.

[21] Y. Xiao and B. Li, "Ant colony algorithm based on little window," *Computer Engineering*, vol. 29, pp. 143–145, 2003.