Journal of Systems Engineering and Electronics
Vol. 31, No. 4, August 2020, pp.751 – 760

Parallel discrete lion swarm optimization algorithm for solving traveling salesman problem

ZHANG Daoqing and JIANG Mingyan*

School of Information Science and Engineering, Shandong University, Qingdao 266237, China

Abstract: As a typical representative of the NP-complete problem, the traveling salesman problem (TSP) is widely utilized in computer networks, logistics distribution, and other fields. In this paper, a discrete lion swarm optimization (DLSO) algorithm is proposed to solve the TSP. Firstly, we introduce discrete coding and order crossover operators in DLSO. Secondly, we use the complete 2-opt (C2-opt) algorithm to enhance the local search ability. Then in order to enhance the efficiency of the algorithm, a parallel discrete lion swarm optimization (PDLSO) algorithm is proposed. The PDLSO has multiple populations, and each sub-population independently runs the DLSO algorithm in parallel. We use the ring topology to transfer information between sub-populations. Experiments on some benchmarks TSP problems show that the DLSO algorithm has a better accuracy than other algorithms, and the PDLSO algorithm can effectively shorten the running time.

Keywords: discrete lion swarm optimization (DLSO) algorithm, complete 2-opt (C2-opt) algorithm, parallel discrete lion swarm optimization (PDLSO) algorithm, traveling salesman problem (TSP).

DOI: 10.23919/JSEE.2020.000050

1. Introduction

The classical traveling salesman problem (TSP) has a very intuitive explanation: for a given city group, the traveling salesman needs to find the shortest route through all cities, requiring to pass each city only once and finally return to the starting city.

In the early stage, the exact method was mainly utilized to solve the TSP, including the cutting plane method [1], the dynamic programming method [2], the branch definition method [3]. These methods are characterized by traversing the entire solution space at the cost of high complexity to obtain the theoretical optimal solution, not suitable for large-scale TSP. In order to overcome the shortcomings of the exact algorithm, approximation algorithm becomes the mainstream algorithm in the TSP field, such as the Clarke & Wright's savings (CWS) algorithm [4], the local search algorithm (2-opt, 3-opt, Lin and Kernighan's (LK) algorithm) [5], the Lin-Kernighan heuristic (LKH) algorithm [6] and the heuristic algorithm represented by ant colony (ACO) algorithm [7], the artificial bee colony (ABC) algorithm [8], and the artificial fish swarm algorithm [9]. The heuristic algorithm can find the approximate global optimal solution in the nonlinear sophisticated solution space, so it can be utilized in various optimization problems, which is not limited by the convexity and derivability of the optimization problem. In the TSP, a hybrid algorithm combining the local search algorithm with strong local optimization ability and the heuristic algorithm with global optimization ability can obtain a practical solution within an acceptable time, which becomes a new direction to solve the TSP. Kan et al. [10] proposed an improved ACO algorithm, by reducing the processing cost of ant routing and using the individual variation and routing strategy (IVRS). Experiments show that the IVRS has achieved good results. Kefi et al. [11] proposed a hybrid algorithm of ant heuristic-particle swarm optimization-2opt (AS-PSO-2opt). The PSO algorithm optimized the parameters of the ACO. The ACO algorithm searched for the route, then was optimized by the 2-opt operator. Experiments showed this method could achieve better results. Zhou et al. [12] proposed the discrete weed algorithm. Ouaarab et al. [13] proposed the discrete cuckoo search (DCS) algorithm. They are all hybrid algorithms, and the results show the combination of the heuristic algorithm and the local search algorithm is an effective way to solve the TSP. In recent years, in some NP-hard combinatorial optimization problems related to TSP, the discretized heuristic optimization algorithm also performs well. Han [14] et al. proposed the discrete ABC (DABC) algorithm to solve the flow shop scheduling problem; at the same time, the distributed heuristic optimization algorithm has an excellent performance in solving urban search and rescue prob-

Manuscript received September 29, 2019.
*Corresponding author.
This work was supported by the National Natural Science Foundation of China (61771293) and the Key Project of Shangdong Province (2019JZZY010111).

lems [15], localizing odor sources [16,17]. Inspired by the above papers, we extend the original lion swarm optimization (LSO) algorithms to parallel and discrete forms.

Noted that it is still a complicated problem to get the exact solution of the TSP, and how to get an effective solution within a reasonable time has become the focus of the research. In this paper, a discrete LSO (DLSO) algorithm and a parallel DLSO algorithm (PDLSO) for solving the TSPs are proposed. We propose these two algorithms because they have two main advantages: (i) the DLSO algorithm obtains a better solution in a reasonable time; (ii) the PDLSO algorithm significantly shortens the running time.

The article is organized as follows: Section 2 introduces the LSO algorithm and the complete 2-opt (C2-opt) algorithm; Section 3 introduces DLSO algorithms and PDLSO algorithms; Section 4 utilizes symmetric benchmarks TSPs in TSP library (TSPLIB) to validate the effectiveness of the algorithm and analyze the experimental results; Section 5 concludes this paper and proposes future work.

2. Fundamental knowledge

2.1 LSO algorithm

The LSO algorithm [18] is a new swarm intelligence algorithm proposed by Liu et al. in 2018. By simulating the lion hunting behavior, the LSO algorithm performs better than the traditional genetic algorithm (GA) and PSO algorithm in terms of convergence speed and solution accuracy. The main idea of the LSO algorithm is as follows: according to the fitness value, the lion swarm is divided into three categories: lion king, lioness, and lion cubs. Among them, lion king and lioness are adult lions, the proportion of adult lions in a population is β.

In the D dimensional search space, SN lions form a population, and the position of the ith lion in the tth iteration is $\boldsymbol{x}_i^t$, $\boldsymbol{x}_i^t = (x_{i1}, x_{i2}, \dots, x_{iD})$ $(1 \leqslant i \leqslant SN)$.

The lion king moves around the global optimal in a small range:

$$\boldsymbol{x}_i^{t+1} = (1 + \gamma \boldsymbol{p}_i^t - \boldsymbol{g}^t)\boldsymbol{g}^t \tag{1}$$

where γ is a random number generated according to the standard normal distribution $N(0,1)$; $\boldsymbol{p}_i^t$ is the optimal historical position of the ith lion in the population at the tth generation; $\boldsymbol{g}^t$ is the optimal position of the tth generation.

Lioness moves in pair as follows:

$$\boldsymbol{x}_i^{t+1} = \frac{(1+\gamma\alpha_f)(\boldsymbol{p}_i^t + \boldsymbol{p}_c^t)}{2} \tag{2}$$

where $\alpha_f = \text{step} \cdot \exp\left(-\dfrac{30t}{T}\right)^{10}$ is the disturbance factor of the lioness movement, taking the step length $\text{step} = 0.1(\overline{\text{high}} - \overline{\text{low}})$, where $\overline{\text{low}}$ and $\overline{\text{high}}$ are the minimum mean and maximum mean values of each dimension in the lion activity space; $\boldsymbol{p}_c^t$ is the optimal historical position of a lioness randomly selected from the tth generation lioness population.

To increase the global optimization ability, lion cubs have three movement modes: follow the lion king to move, follow the lioness to move, and move by elite opposition-based learning [19]:

$$\boldsymbol{x}_i^{t+1} = \begin{cases} \dfrac{(1+\gamma\alpha_c)(\boldsymbol{p}_i^t + \boldsymbol{g}^t)}{2}, & q \leqslant \dfrac{1}{3} \\ \dfrac{(1+\gamma\alpha_c)(\boldsymbol{p}_i^t + \boldsymbol{p}_m^t)}{2}, & \dfrac{1}{3} \leqslant q \leqslant \dfrac{2}{3} \\ \dfrac{(1+\gamma\alpha_c)(\boldsymbol{p}_i^t + \overline{\boldsymbol{g}}^t)}{2}, & \dfrac{2}{3} \leqslant q \leqslant 1 \end{cases} \tag{3}$$

where q is a random fraction generated according to the uniform distribution $U(0,1)$; $\alpha_c = \text{step} \cdot \dfrac{T-t}{T}$ is the disturbance factor of the cub movement, where T is the maximum number of iterations; $\boldsymbol{p}_m^t$ is the optimal historical position of the lioness at the tth generation; $\overline{\boldsymbol{g}}^t = \overline{\text{high}} + \overline{\text{low}} - \boldsymbol{g}^t$ indicates the position where the cubs are expelled.

From the above introduction, we know that the LSO algorithm has three main control parameters: the disturbance factor of the lioness movement α_f, the disturbance factor of the cub movement α_c, and the percentage of adult lions β. The values of α_f and α_c are based on step which is determined by the optimization problem itself, and the value of β in the original paper is 0.3.

The algorithm flow is as follows:

Step 1 Initialization: Initialize parameters T and β. Randomly generate SN lion swarm individuals $\boldsymbol{x}_i^t$ $(i = 1, 2, \dots, SN)$. First, sort according to the fitness value of the individual, and then determine the initial positions of the lion king, lioness, and cubs.

Step 2 Lions move: According to (1), (2) and (3), complete the movement of the lion king, lioness and lion cubs.

Step 3 Lions update: Update the fitness value; update $\boldsymbol{p}_i^t$ and $\boldsymbol{g}^t$.

Step 4 Algorithm termination: Repeat steps 2 to 3 until $t = T$; record the optimal fitness value and the optimal individual of the population.

2.2 C2-opt algorithm

The 2-opt algorithm is a branch of the local search K-opt algorithm, and it is the most basic method in the local search algorithm. The basic idea is to delete two edges in the original route and replace them with two other shorter edges. As shown in Fig. 1, the left route is the original route. Two edges AB and CD are selected randomly. If the sum of the edge length of AB and CD is larger than that of

AC and BD, edges AB and CD are deleted and replaced by shorter edges AC and BD.

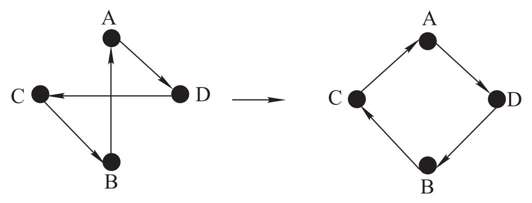

Fig. 1 The 2-opt schematic algorithm

The convergence of the 2-opt algorithm using the random selection method requires a long computation time, so the C2-opt algorithm is utilized in this paper. The C2-opt performs a C2-opt operation on a route to obtain local optimum so that it can converge in a short time [12].

3. Proposed methods (DLSO and PDLSO)

3.1 DLSO algorithm

The original LSO algorithm is only suitable for the continuous functions. In order to make the LSO applicable to the TSP, a DLSO algorithm is proposed, the specific improvement methods are as follows:

(i) Discrete coding: The lion population changes from the traditional real value coding to integer discrete coding. For the TSP with n cities, the lion individual is represented as an array of length n, and the value of each element in the array is a positive integer between 1 and n, representing the corresponding city; in order to satisfy the Hamiltonian loop condition, there are no elements with the same value in the array.

(ii) Individual crossover: The lion movement of the LSO algorithm is completed by (1), (2) and (3), and the core of formulas is to adjust the numerical value of elements in the individual. If a discretely coded individual is directly substituted into formulas, a non-integer will be obtained, although it can be converted to an integer by rounding, modulo, etc. However, the TSP focuses on the sequential structure of the elements in the individual rather than the numerical value of the element, these methods cannot guarantee that the newly generated individual will inherit the structure of the parent individual. The crossover operator introduced in the GA can enable the offspring to retain the structural information of the parent, making the movement of individuals more effective. Standard crossover operators in GAs include partially-mapped crossover (PMX), shuffle crossover, order crossover (OX), cycle crossover (CX) and uniform crossover [20]. According to the experiment, we can see the OX operator will achieve a better solution in GA [21], so we utilize the OX operator in DLSO.

The OX was proposed by Oliver et al.. It generates the offspring by choosing a sub-tour from one parent and preserving the relative order of cities of the other parent. The OX works as follows:

Randomly select two cut points marked with "|" on parents PAR_1 and PAR_2:

$$PAR_1 = (1\ \ 2|3\ \ 4\ \ 5|6\ \ 7),$$

$$PAR_2 = (2\ \ 5|6\ \ 1\ \ 3|4\ \ 7).$$

Generate offspring in the following way: first, the subtour between the two cut points are coppied into offsprings:

$$O_1 = (-\ \ -|3\ \ 4\ \ 5|-\ \ -),$$

$$O_2 = (-\ \ -|6\ \ 1\ \ 3|-\ \ -).$$

Then, starting from the second cut point of a parent, the bits from another parent are coppied in the same order, and the existing bits are omitted. Take generating O_1 as an example. The sequence of bits in PAR_2 from the second cut point is $(4\ \ 7\ \ 2\ \ 5\ \ 6\ \ 1\ \ 3)$, removing bits $(3\ \ 4\ \ 5)$ that already exist in O_1, we get a new sequence $(7\ \ 2\ \ 6\ \ 1)$. Starting from the second cut point, place this sequence in O_1, we get

$$O_1 = (6\ \ 1|3\ \ 4\ \ 5|7\ \ 2).$$

Again, we can get

$$O_2 = (4\ \ 5|6\ \ 1\ \ 3|7\ \ 2).$$

In summary, the movements of the lion king, lioness and cubs are completed according to

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{p}_i^t \text{ OX } \boldsymbol{g}^t, \tag{4}$$

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{p}_i^t \text{ OX } \boldsymbol{p}_c^t, \tag{5}$$

$$\boldsymbol{x}_i^{t+1} = \begin{cases} \boldsymbol{p}_i^t \text{ OX } \boldsymbol{g}^t, & q \leqslant \dfrac{1}{3} \\ \boldsymbol{p}_i^t \text{ OX } \boldsymbol{p}_m^t, & \dfrac{1}{3} < q \leqslant \dfrac{2}{3} \\ \boldsymbol{p}_i^t \text{ OX } \overline{\boldsymbol{g}}^t, & \dfrac{2}{3} < q \leqslant 1 \end{cases} \tag{6}$$

where the meanings of the symbol $\boldsymbol{p}_i^t, \boldsymbol{g}^t, \boldsymbol{p}_c^t, \boldsymbol{p}_m^t, \overline{\boldsymbol{g}}^t$ are the same as those in (1), (2) and (3); OX indicates the OX operation.

(iii) Introduction of C2-opt operator: The lion swarm algorithm guarantees strong global optimization ability by dividing the population into three categories — lion king, lioness and lion cub; but it converges slowly in the TSP and has poor local optimization ability. However, by combining with the C2-opt operator with strong local optimization ability, the ideal solution can be obtained. The specific method is that in each iteration, C2-opt operation is performed on the lion king, the two optimal lionesses, and the optimal lion cub.

(iv) Fitness function: The graph theory gives an accurate mathematical description of the TSP — giving a weighted directed graph $G = (V, E, W)$, vertex set $V = \{v_1, v_2, v_3, \dots, v_n\}$, where v_i represents the ith city; the edge set $E = \{\langle v_i, v_j\rangle | i, j \in \mathbf{N}^+; i, j \leqslant n\}$; the weight set $W = \{d_{i,j} | i, j \in \mathbf{N}^+; i, j \leqslant n; d_{i,j} \in \mathbf{R}^+\}$, where $d_{i,j}$ represents the length corresponding to the edge $\langle v_i, v_j\rangle$. The loop that passes through each vertex once and only once in the graph G is called a Hamiltonian loop. As we mentioned in discrete coding, the individual coding in the population satisfies the Hamiltonian loop condition, so the individual $\boldsymbol{x}$ can be decoded into a Hamiltonian loop $\langle v_{x_1}, v_{x_2}, v_{x_3}, \dots, v_{x_n}, v_{x_1}\rangle$. The sum of the weights of this loop is the fitness value, so the fitness function can be expressed by

$$f(\boldsymbol{x}) = d_{x_1,x_n} + \sum_{i=1}^{n-1} d_{x_i,x_{i+1}}. \tag{7}$$

The pseudo-code of DLSO is as follows:

Algorithm 1 DLSO algorithm

Input Population size SN; percentage of adult lions β; maximum iteration number T.

```
1. $t = 0$
2. **repeat**
3.   **if** $t = 0$ **then**
4.     Randomly generate discrete coding lion swarm $\boldsymbol{x}_i^t$ $(i = 1, 2, \dots, SN)$.
5.     Sort by fitness value $f(\boldsymbol{x}_i^t)$, divide swarm into lion king, lioness and lion cubs.
6.     $\boldsymbol{p}_i^t = \boldsymbol{x}_i^t; \boldsymbol{g}^t = \arg\min_{\boldsymbol{x}}(f(\boldsymbol{x})), \boldsymbol{x} \in \{\boldsymbol{p}_1^t, \dots, \boldsymbol{p}_{SN}^t\}$.
7.   **end if**
8.   **for** $i = 1 : SN$ **do**
9.     **if** $\boldsymbol{x}_i^t$ is the lion king **then**
10.      Generate $\boldsymbol{x}_i^{t+1}$ according to (4)
11.    **else if** $\boldsymbol{x}_i^t$ is the lioness **then**
12.      Generate $\boldsymbol{x}_i^{t+1}$ according to (5)
13.    **else** $\boldsymbol{x}_i^t$ is lion cub **then**
14.      Generate $\boldsymbol{x}_i^{t+1}$ according to (6)
15.    **end if**
16.    $\boldsymbol{p}_i^{t+1} = \arg\min_{\boldsymbol{x}}(f(\boldsymbol{x})), \boldsymbol{x} \in \{\boldsymbol{x}_i^{t+1}, \boldsymbol{p}_i^t\}$
17.  **end for**
18.  $\boldsymbol{g}^{t+1} = \arg\min_{\boldsymbol{x}}(f(\boldsymbol{x})), \boldsymbol{x} \in \{\boldsymbol{p}_1^{t+1}, \dots, \boldsymbol{p}_{SN}^{t+1}\}$
19.  Run C2-opt in the lion king, two optimal lionesses and the optimal lion cub
20.  $t := t + 1$
21. **until** $t = T$
```

Output Optimal individual of the population $\boldsymbol{g}^t$.

3.2 PDLSO algorithm

The DLSO algorithm consumes a lot of computing time in solving large-scale TSPs. In order to obtain a solution with the same quality as the DLSO algorithm in a shorter time, the PDLSO algorithm is proposed in this paper.

The idea of the PDLSO algorithm is as follows: Set SN as the total population size, the population is divided into N sub-populations. Therefore, the population size of each sub-population SN_{sub} can be derived from

$$SN_{\text{sub}} = \frac{SN}{N}. \tag{8}$$

Then, each sub-population runs the DLSO algorithm independently, and exchanges information with each other until the algorithm terminates, the optimal solution of all sub-populations is the final solution of the algorithm. The topological structure and the parallel strategy have a great influence on the PDLSO algorithm; common topologies are shown in Fig. 2, including the star topology, ring topology, and fully connected topology [22].

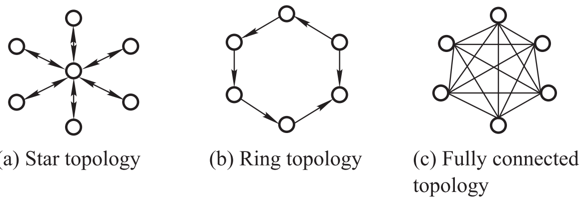

Fig. 2 Three topologies

Nalepa and Blocho's [23] experiments show the star topology distributes the best individuals between sub-populations, so it converges relatively fast; the ring topology converges slower than the star topology, but it can benefit from broader search space exploration and give the best asymptotic results. Frahnow et al. [24] proved that the convergence rate of the ring topology is faster than the fully connected topology, and the ring topology can be preferable over the fully connected topology in terms of maintaining diversity. Because the DLSO algorithm has a strong local optimization ability, the ring topology with the global optimization ability is more suitable for the PDLSO algorithm, so we choose the ring topology.

The parallel strategy which is mainly determined by the migration rate, the migration period R, and the migration rules, determines the information exchange between sub-populations and increases the global information of the algorithm. The migration rate determines the number of individuals that undergo migration in every exchange; the migration period R represents the number of iterations between two migrations; the migration rule determines which individuals are migrated and which individuals in the target sub-population are replaced. Combined with the ring topology, the parallel strategy utilized in this paper is as follows: when the migration period R is satisfied, the lion king $\boldsymbol{g}_k$ in the kth sub-population is migrated to its neighboring $(k+1)$th sub-population, replacing the worst individual in the $(k+1)$th sub-population. With the above pa-

Captions within figure: (a) Star topology (b) Ring topology (c) Fully connected topology rallel strategy, the number of communication times between sub-populations comm can be derived from (9), where T is the maximum number of iterations.

$$\mathrm{comm} = N \cdot \frac{T}{R}. \tag{9}$$

The pseudo-code of PDLSO is as follows:

Algorithm 2 PDLSO algorithm

Input Total population size SN; percentage of adult lions β; maximum iteration number T; sub-population number N; migration interval R.

```
1. $t = 0$
2. repeat
3.   parallel for $k = 1 : N$ do
4.     if $t = 0$ then
5.       Randomly generate discrete coding lion swarm $\boldsymbol{x}_i^t$ $(i = 1, \dots, SN_{\mathrm{sub}})$ in sub-population $k$.
6. Sort by fitness value $f(\boldsymbol{x}_i^t)$, divide swarm into king, lioness, lion cubs in sub-population $k$.
7. $\boldsymbol{p}_i^t = \boldsymbol{x}_i^t; \boldsymbol{g}^t = \arg\min_{\boldsymbol{x}}(f(\boldsymbol{x})), \boldsymbol{x} \in \{\boldsymbol{p}_1^t, \dots, \boldsymbol{p}_{SN}^t\}$ in sub-population $k$.
8.     end if
9.     for $i = 1 : SN$ do
10. Generate $\boldsymbol{x}_i^{t+1}$ in sub-population $k$ according to (4), (5) and (6).
11. $\boldsymbol{p}_i^{t+1} = \arg\min_{\boldsymbol{x}}(f(\boldsymbol{x})), \boldsymbol{x} \in \{\boldsymbol{x}_i^{t+1}, \boldsymbol{p}_i^t\}$ in sub-population $k$.
12.     end for
13. $\boldsymbol{g}^{t+1} = \arg\min_{\boldsymbol{x}}(f(\boldsymbol{x})), \boldsymbol{x} \in \{\boldsymbol{p}_1^{t+1}, \dots, \boldsymbol{p}_{SN}^{t+1}\}$ in sub-population $k$.
14. if $t\%S = 0$ then
15. Run C2-opt in the lion king, two optimal lionesses and the optimal lion cub in sub-population $k$.
16. end if
17.     if $t\%R = 0$ then
18.       Find the optimal position $\boldsymbol{g}_k^t$ of the $k$th sub-population.
19.       Send $\boldsymbol{g}_k^t$ to the $k^*$th sub-population, where $k^* = (k+1)\%S$.
20. $\boldsymbol{g}_k^t$ replace the worst individual in the $k^*$th sub-population.
21.     end if
22. $t := t + 1$
23. until $t = T$ in all sub-populations
```

Output The optimal individual of all sub-populations $\boldsymbol{g}^t = \min(\boldsymbol{g}_1^t, \dots, \boldsymbol{g}_N^t)$.

4. Experiments and results

The specifications of the testing platform utilized in the experiments are shown in Table 1. Seventeen symmetric TSPs (Eil51, Berlin52, St70, Pr76, KroA100, KroB100, Lin105, Ch130, Ch150, D198, KroA200, KroB200, Tsp225, A280, Lin318, Pcb442, Pr1002) in TSPLIB [25] are selected as cases to test the DLSO algorithm; ten TSPs of different scales (Eil51, Berlin 52, Eit70, KroA100, Ch150, D198, Dp225, A280, Lincb318, Pcb442) are selected as cases to test the PDLSO algorithm. Each instance runs 20 times independently.

Table 1 Specifications of the experimental platform

Specification	Value
Operating system	Ubuntu 18.04LTS
CPU	Core i7 4 GHz
Memory	30 GB
Python version	Python 3.6
MPI version	Mpi4py

4.1 Experimental analysis of DLSO algorithm

In this section, we first analyze the results of DLSO in the TSP problem and then verify the effectiveness of the DLSO algorithm by comparing it with other algorithms. The parameters of the DLSO algorithm are shown in Table 2. Table 3 show the experimental results. In Table 3, the first column contains 16 specific TSPs, the column OPT represents the known optimal value, the column Best represents the optimal value obtained by the DLSO algorithm in 20 independent runs, the column Average represents the mean value of solutions, the column Std represents the standard deviation, the column Time represents the average running time of the DLSO algorithm, and the column Error represents the relative deviation between the average value of the DLSO algorithm and the known optimal value. Relative deviation is a general index in the TSP [26], which is calculated by

$$\mathrm{error} = \frac{\mathrm{average} - \mathrm{opt}}{\mathrm{opt}}. \tag{10}$$

Table 2 DLSO parameters setting

Variable	Value	Meaning
SN	96	Population size
β	0.2	Percentage of adult lions
T	150	Maximum iteration number

As can be seen from Table 3, for the problem of the city size within 100, the DLSO algorithm can get the error of [0.03%, 0.9%] in 1 min, and the deviation of solutions is small, for example, the Berlin52 problem's standard deviation is 0, indicating the algorithm is robust. For the problem of 100 to 300 cities, the error of solution is between [0.4%, 2.7%], and the running time is between 1 min and 12 min, which does not increase rapidly. For the problem of 300 to 500 cities, the running time can be limited within 50 min, and the error is less than 4%. For the problem of over 1 000 cities, although the running time increases rapidly, the error is about 6%, which has not increased rapidly. With the increase of the city scale, the error of the algorithm becomes more massive, but in this experiment, the maximum number of iterations T is 150, and a more accurate solution can be obtained by increasing the value of T appropriately.

Table 3 Experimental results of DLSO algorithm for 17 TSP problems of TSPLIB

Problem	OPT	Best	Average	Error/%	Std	Time/s
Eil51	426	428.87	429.70	0.87	1.61	8.51
Berlin52	7 542	7 544.37	7544.37	0.031	0	8.92
St70	675	677.11	678.78	0.56	3.38	17.89
Pr76	108 159	108 159.43	108 572.35	0.38	341.96	22.12
KroA100	21 282	21 285.44	21 370.09	0.41	44.66	61.02
KroB100	22 141	22 142.07	22 270.58	0.58	95.52	45.08
Lin105	14 379	14 383.0	14 433.33	0.38	34.23	50.55
Ch130	6 110	6 158.08	6 201.98	1.50	30.96	93.33
Ch150	6 528	6 530.90	6 597.83	1.07	38.83	164.09
D198	15 780	15 808.93	15 896.48	0.74	35.21	284.53
KroA200	29 368	29 519.83	29 766.27	1.37	118.37	288.19
KroB200	29 437	29 652.94	29 994.08	1.89	226.62	285.06
Tsp225	3 916	3 929.51	3 977.53	1.57	21.05	397.92
A280	2 579	2 609.54	2 650.49	2.77	33.83	768.44
Lin318	42 029	42 744.96	43 172.51	2.72	235.18	1 113.98
Pcb442	50 778	52 330.24	52 841.26	4.06	230.75	2 883.50
Pr1002	259 045	273 696.03	275 825	6.47	1 189.80	34 456.31

Fig. 3 shows the fitness curve, and it can be seen from the figure that the algorithm can converge in shorter iterations, indicating the DLSO algorithm has a strong local optimization ability; while the algorithm can still further optimize in the late iterations, indicating the DLSO algorithm has a strong global optimization ability, not easy to fall into local extremes. Fig. 4 shows the route obtained by the DLSO algorithm in the Pcb442 problem.

The IVRS+2opt [10], ACO+2opt and ACO+ABC [27] algorithms with good results in recent years were selected as comparisons to verify the effectiveness of the DLSO algorithm. Table 4 gives the comparison results on six TSPs. "—" means that there is no data in the literature, and boldface numbers mean the best value among the four algorithms. It can be seen from Table 4 that the DLSO algorithm is superior to other three algorithms in the optimal value and the average value. For the ACO+ABC algorithm, the DLSO algorithm can improve the accuracy of solution in the range of [1%, 5%]; for the ACO+2opt algorithm, DLSO can significantly improve the accuracy of solution on Eil51, KroA100 and D198 problems, especially for KroA100, the DLSO algorithm can improve about 10%; for the improved AC algorithm IVRS+2opt, DLSO improves the accuracy of solution about 0.5% on Eil51, KroA100 and D198.

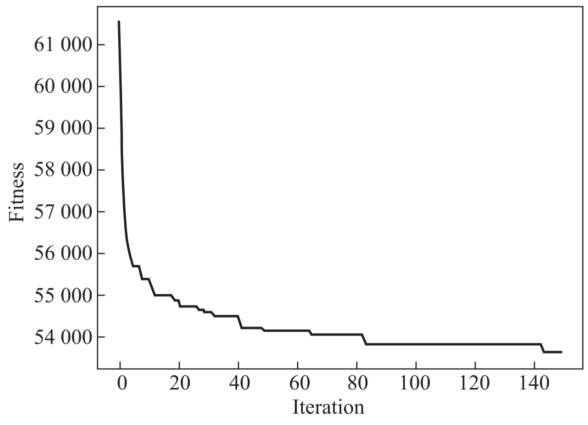

Fig. 3 Curve evolution diagram of DPLO in Pcb442

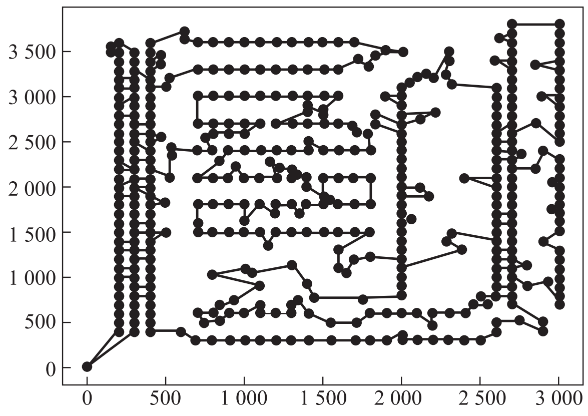

Fig. 4 Tours obtained by DLSO in Pcb442

Table 4 Comparison of the DLSO algorithm with other algorithms for six TSPs

Problem	OPT	IVRS+2opt			ACO+2opt			ACO+ABC			DLSO+2opt		
		Best	Average	Error/%	Best	Average	Error/%	Best	Average	Error/%	Best	Average	Error/%
Eil51	426	429.12	431.11	1.20	431.26	439.26	3.11	431.74	443.39	4.08	**428.98**	**429.70**	**0.87**
Berlin52	7 542	7 544.37	7 547.24	0.07	7 549.53	7 556.59	0.19	7 544.37	7 544.37	3.14	**7 544.37**	**7 544.37**	**0.031**
St70	675	—	—	—	—	—	—	687.24	700.58	3.79	**677.11**	**678.78**	**0.56**
KroA100	21 282	21 309.43	21 498.62	1.02	22 006.40	23 441.80	10.15	22 122.75	22 435.31	5.42	**21 285.44**	**21 370.09**	**0.41**
Eil101	629	**631.29**	**648.67**	**3.13**	654.50	672.38	6.90	672.71	683.39	8.65	642.53	649.05	3.19
Ch150	6 528	—	—	—	—	—	—	6 641.69	6 677.12	2.28	**6 530.90**	**6 597.83**	**1.07**
D198	15 780	15 842.52	15 972.47	1.22	16 054.47	16 252.93	2.30	—	—	—	**15 808.93**	**15 896.48**	**0.74**

4.2 Experimental analysis of PDLSO algorithm

Because different computing devices will affect the acceleration performance of parallel algorithms, we divide the experiment into two cases according to different computing devices. Case 1: single computer with multi-process; Case 2: double computers of the same configuration with multi-process. Case 2 can be divided into two situations: wired connection and wireless connection according to the way two computers establish local area network (LAN). We first compare the results of the PDLSO and DLSO algorithms to verify the effectiveness of the PDLSO algorithm, then analyze the acceleration performance of the PDLSO algorithms in Case 1 and Case 2. The parameters of the PDLSO algorithm are shown in Table 5.

Table 5 PDLSO parameters setting

Variable	Value	Meaning
SN	96	Total population size
β	0.2	Percentage of adult lions
T	150	Maximum iteration number
N	2/3/4	Sub-population number
R	10	Migration interval

Case 1 Single computer with multi-process

Table 6 shows the experimental results of a single computer, gives the results of serial DLSO, 2-process PDLSO, 3-process PDLSO and 4-process PDLSO, including the time of algorithms, mean value and standard deviation of the solution. In terms of time of the algorithm, PDLSO can significantly reduce the running time of the DLSO algorithm and achieve an approximately linear acceleration ratio. The mean and standard deviations of the solution are important criteria for measuring the quality of the solution, it can be seen from Table 6 that the PDLSO algorithm of 2-process and 3-process is basically consistent with the serial DLSO algorithm in terms of the quality of the solution, the accuracy of 4-process PDLSO solutions decreases due to fewer individuals in the sub-population, but the difference between 4-process PDLSO solutions and DLSO solutions is less than 0.5%.

Table 7 gives a more detailed description of the parallel performance of PDLSO in a single computer, where S represents the parallel speedup ratio, which is obtained by (11); EFF represents the parallel efficiency, which is obtained by (12) [28]. Among them, T_{serial} represents the running time of the serial algorithm, T_{parallel} represents the running time of the parallel algorithm, and PN represents the process number of the parallel algorithm.

$$S = \frac{T_{\text{serial}}}{T_{\text{parallel}}} \tag{11}$$

$$\text{EFF} = \frac{T_{\text{serial}}}{PN \cdot T_{\text{parallel}}} \tag{12}$$

Table 6 Comparison of the PDLSO in a single computer with serial DLSO for nine TSP problems

Problem	OPT	Serial DLSO			2-process PLSO			3-process PLSO			4-process PLSO		
		Time/s	Average	Std	Time/s	Average	Std	Time/s	Average	Std	Time/s	Average	Std
Eil51	426	8.64	429.70	1.61	4.74	430.64	1.68	3.14	430.65	1.51	2.52	430.46	1.13
Berlin52	7 542	8.92	7 544.37	0	4.90	7 544.37	0	3.41	7 544.37	0	2.59	7 544.37	0
St70	675	17.90	678.78	3.38	9.55	679.07	2.31	6.58	680.09	4.12	5.27	680.58	3.48
KroA100	21 282	44.66	21 370.09	61.02	23.67	21 367.71	74.85	16.05	21 355.29	64.71	12.81	21 344.47	66.47
Ch150	6 528	134.44	6 597.83	38.83	69.00	6 603.26	36.87	47.09	6 627.82	40.16	38.17	6 623.73	42.47
D198	15 780	283.14	15 896.48	35.21	144.97	15 991.86	64.46	99.25	15 914.60	38.44	78.85	15 937.27	53.58
Tsp225	3 916	397.92	3 977.53	21.05	207.77	3 978.96	23.06	142.46	3 986.98	25.41	113.98	3 994.11	21.07
A280	2 579	768.44	2 650.49	33.83	393.56	2 636.71	20.51	273.71	2 659.07	26.37	217.52	2 663.63	33.00
Lin318	42 029	1 113.98	43 172.51	235.18	564.55	43 204.69	420.07	384.45	43 258.12	228.34	305.92	43 271.40	310.14
Pcb442	50 778	2 883.50	52 841.26	230.75	1 460.11	53 057.05	363.56	992.77	53 238.61	408.95	782.37	53 591.85	474.61

Taking line St70 in Table 7 as an example, the efficiency of the 2-process is 0.94, the 3-process is 0.91, and the 4-process is 0.84. It can be seen that for a fixed problem, as the number of processes PN increases, the parallel efficiency EFF decreases. This is because the larger the processes PN, the greater the number of communication times comm, that is, the number of non-parallel parts of the algorithm increases, and T_{parallel} increases accordingly, however, the increase of processes has not brought about drastic changes in parallel efficiency EFF, because for T_{parallel}, the time consumed by communication is relatively small. Taking the column 2-process PDLSO in Table 7 as an example, it can be seen that as the scale of the TSP increases, the speedup ratio and efficiency increase. Gustafson's law [29] can give an intuitive explanation: as the size of the problem increases, the proportion of the communication part of the DLSO algorithm decreases. Fig. 5 shows the parallel efficiency comparison of three TSPs with different city sizes. The above conclusion can be seen intuitively from Fig. 5. It can be seen from the above analysis that PDLSO algorithm has a good parallel performance on the TSPs.

Table 7 Performance of PDLSO in a single computer of nine TSP problems

Problem	2-process PDLSO		3-process PDLSO		4-process PDLSO	
	S	EFF	S	EFF	S	EFF
Eil51	1.82	0.91	2.75	0.91	3.43	0.86
Berlin52	1.82	0.91	2.62	0.87	3.44	0.86
St70	1.87	0.94	2.72	0.91	3.40	0.84
KroA100	1.87	0.94	2.78	0.93	3.49	0.87
Ch150	1.95	0.97	2.85	0.95	3.52	0.88
D198	1.95	0.97	2.85	0.95	3.59	0.89
Tsp225	1.92	0.96	2.79	0.93	3.49	0.87
A280	1.95	0.97	2.81	0.94	3.53	0.88
Lin318	1.97	0.98	2.89	0.96	3.64	0.91
Pcb442	1.97	0.98	2.90	0.96	3.68	0.92

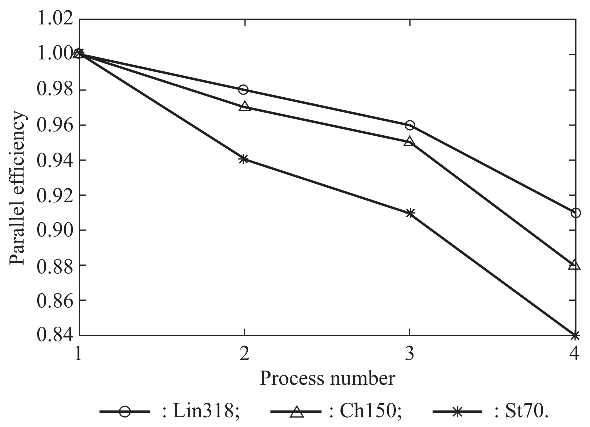

Fig. 5 Parallel efficiency comparison of St70, Ch150 and Lin318

Case 2 Double computers of the same configuration with multi-process

In Case 1, we have proved that the results of PDLSO and DLSO are basically the same. Since different computing devices do not change the results of the PDLSO algorithm, we mainly analyze the impact of different computing devices on the acceleration performance of the PDLSO algorithm in Case 2. We utilize a router to establish an LAN between two computers. According to the connection mode, it can be divided into wired connection and wireless connection. The wired connection utilizes IEEE 802.3 protocols, and the wireless connection utilizes IEEE 802.11g protocols. The maximum transmission rate is 54 Mbps. The results of the PDLSO algorithm have been given in Case 1, here we mainly focus on the parallel acceleration of PDLSO on the double computers platform. The performance of double computers wired connection is shown in Table 8, the performance of double computers wireless connection is shown in Table 9. From Table 8 and Table 9, we can draw the same conclusion as that in Case 1: for a fixed problem, as the number of processes PN increases, the parallel efficiency EFF decreases; as the scale of the TSP increases, the speedup ratio and efficiency increase. Comparing Table 8 with Table 9, we find the acceleration performance of wireless connection and wired connection is basically the same. Comparing Table 7, Table 8, and Table 9, and taking the acceleration ratio S of the row D198 in the tables as an example, we obtain the parallel speedup ratio comparison in Fig. 6. From Fig. 6, it can be seen intuitively that the acceleration ratio of the double computers is lower than that of the single computer, but still achieves effective acceleration.

Table 8 Performance of PDLSO in double computers wired connection of nine TSP problems

Problem	2-process PDLSO		3-process PDLSO		4-process PDLSO	
	S	EFF	S	EFF	S	EFF
Eil51	1.63	0.82	2.45	0.81	3.09	0.77
Berlin52	1.66	0.83	2.47	0.82	3.10	0.77
St70	1.67	0.83	2.51	0.83	3.25	0.81
KroA100	1.69	0.84	2.55	0.85	3.30	0.82
Ch150	1.73	0.86	2.58	0.85	3.38	0.84
D198	1.75	0.87	2.60	0.86	3.40	0.85
Tsp225	1.79	0.89	2.65	0.88	3.47	0.86
A280	1.80	0.90	2.67	0.89	3.45	0.86
Lin318	1.80	0.90	2.68	0.89	3.47	0.87
Pcb442	1.82	0.91	2.69	0.89	3.49	0.87

Table 9 Performance of PDLSO in double computers wireless connection of nine TSP problems

Problem	2-process PDLSO		3-process PDLSO		4-process PDLSO	
	S	EFF	S	EFF	S	EFF
Eil51	1.62	0.81	2.45	0.81	3.17	0.79
Berlin52	1.65	0.82	2.51	0.83	3.06	0.76
St70	1.68	0.84	2.51	0.83	3.25	0.81
KroA100	1.68	0.84	2.57	0.85	3.29	0.82
Ch150	1.72	0.86	2.58	0.86	3.34	0.83
D198	1.73	0.86	2.60	0.86	3.36	0.84
Tsp225	1.75	0.87	2.65	0.88	3.37	0.84
A280	1.79	0.89	2.67	0.89	3.44	0.86
Lin318	1.79	0.89	2.68	0.89	3.46	0.86
Pcb442	1.81	0.90	2.45	0.81	3.49	0.87

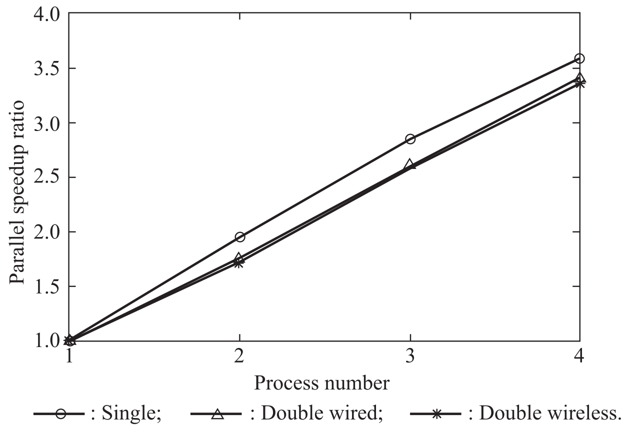

Fig. 6 Parallel speedup ratio comparison of D198 in different platforms

According to the experiments in Case 1 and Case 2, we can see that in different computing devices, the PDLSO algorithm can significantly reduce the running time while ensuring the same accuracy as the DLSO algorithm.

5. Conclusions

This paper proposes the DLSO algorithm and the PDLSO algorithm for solving the TSP. The LSO algorithm is transformed into the DLSO algorithm that can solve the TSP by discrete coding, order crossover operator and C2-opt operator. The PDLSO algorithm divides the population into multiple sub-populations. Sub-populations run the DLSO algorithm in parallel through message passing interface (MPI), and connected by ring topological structure. Sub-populations transfer the optimal individuals between neighborhoods. Experiments on several benchmarks TSPs of different scales are carried out to test DLSO's performance. The statistical results indicate that the DLSO algorithm can find the route close to the theoretical optimal in a reasonable time. The performance of the PDLSO algorithm is tested on a single computer with multi-process and double computers with multi-process platforms respectively. By comparing the results with DLSO, we find the PDLSO algorithm can reduce the running time of the DLSO algorithm in an approximate linear acceleration way while ensuring that the quality of the solution is consistent with the DLSO algorithm. How to apply DLSO and PDLSO to larger scale TSPs and other combinatorial optimization problems is our future work.

References

[1] HUANG Z, ZHENG Q P, PASILIAO E, et al. A cutting plane method for risk-constrained traveling salesman problem with random arc costs. Journal of Global Optimization, 2019, 74(4): 839 – 859.

[2] BELLMAN R E, DREYFUS S E. Applied dynamic programming. Princeton, New Jersey: Princeton University Press, 2015.

[3] CARRABS F, CERULLI R, SPERANZA M G. A branch-and-bound algorithm for the double travelling salesman problem with two stacks. Networks, 2013, 61(1): 58 – 75.

[4] PICHPIBUL T, KAWTUMMACHAI R. An improved Clarke and Wright savings algorithm for the capacitated vehicle routing problem. ScienceAsia, 2012, 38(3): 307 – 318.

[5] KARAPETYAN D, GUTIN G. Lin-Kernighan heuristic adaptations for the generalized traveling salesman problem. European Journal of Operational Research, 2011, 208(3): 221 – 232.

[6] HELSGAUN K. An effective implementation of the Lin-Kernighan traveling salesman heuristic. European Journal of Operational Research, 2000, 126(1): 106 – 130.

[7] UCHIDA A, ITO Y, NAKANO K. An efficient GPU implementation of ant colony optimization for the traveling salesman problem. Proc. of the 3rd IEEE International Conference on Networking and Computing, 2012: 94 – 102.

[8] JIANG M Y, YUAN D F. Artificial bee colony algorithm and its application. Beijing: Science Press, 2014. (in Chinese)

[9] JIANG M Y, YUAN D F. Artificial fish swarm algorithm and its application. Beijing: Science Press, 2012. (in Chinese)

[10] KAN J M, ZHANG Y. Application of an improved ant colony optimization on generalized traveling salesman problem. Energy Procedia, 2012, 17: 319 – 325.

[11] KEFI S, ROKBANI N, KROMER P, et al. Ant supervised by PSO and 2-opt algorithm, AS-PSO-2Opt, applied to traveling salesman problem. Proc. of the IEEE International Conference on Systems, Man, and Cybernetics, 2016, DOI: 10.1109/SMC.2016.7844999.

[12] ZHOU Y, LUO Q, CHEN H, et al. A discrete invasive weed optimization algorithm for solving traveling salesman problem. Neurocomputing, 2015, 151: 1227 – 1236.

[13] OUAARAB A, AHIOD B, YANG X S. Discrete cuckoo search algorithm for the travelling salesman problem. Neural Computing and Applications, 2014, 24(7/8): 1659 – 1669.

[14] HAN Y Y, GONG D W, SUN X. A discrete artificial bee colony algorithm incorporating differential evolution for the flow-shop scheduling problem with blocking. Engineering Optimization, 2015, 47(7): 927 – 946.

[15] GENG N, MENG Q, GONG D W, et al. How good are distributed allocation algorithms for solving urban search and rescue problems? A comparative study with centralized algorithms. IEEE Trans. on Automation Science and Engineering, 2018, 16(1): 478 – 485.

[16] ZHANG J H, GONG D W, ZHANG Y. A niching PSO-based multi-robot cooperation method for localizing odor sources. Neurocomputing, 2014, 123: 308 – 317.

[17] GONG D W, ZHANG Y, QI C L. Localising odour source using multi-robot and anemotaxis-based particle swarm optimisation. IET Control Theory & Applications, 2012, 6(11): 1661 – 1670.

[18] LIU S J, YANG Y, ZHOU Y Q. A swarm intelligence algorithm — lion swarm optimization. Pattern Recognition and Artificial Intelligence, 2018, 31(5): 431 – 441.

[19] MAHDAVI S, RAHNAMAYAN S, DEB K. Opposition based learning: a literature review. Swarm and Evolutionary Computation, 2018, 39: 1 – 23.

[20] UMBARKAR A J, SHETH P D. Crossover operations in genetic algorithms: a review. ICTACT Journal on Soft Computing, 2015, 6(1): 1083 – 1092.

[21] ABDOUN O, ABOUCHABAKA J. A comparative study of adaptive crossover operators for genetic algorithms to resolve the traveling salesman problem. International Journal of Computer Applications, 2011, 31(11): 49 – 57.

[22] LALWANI S, SHARMA H, SATAPATHY S C, et al. A survey on parallel particle swarm optimization algorithms. Arabian Journal for Science and Engineering, 2019, 44(4): 2899 – 2923.

[23] NALEPA J, BLOCHO M. Co-operation in the parallel memetic algorithm. International Journal of Parallel Programming, 2015, 43(5): 812 – 839.

[24] FRAHNOW C, KOTZING T. Ring migration topology helps bypassing local optima. Proc. of the International Conference on Parallel Problem Solving from Nature, 2018: 129 – 140.

[25] REINELT G. TSPLIB — a traveling salesman problem library. ORSA Journal on Computing, 1991, 3(4): 376 – 384.

[26] GUTIN G, PUNNEN A. The traveling salesman problem and its variations. Dordrecht: Kluwer Academic Publishers, 2002.

[27] GUNDUZ M, KIRAN M S, OZCEYLAN E. A hierarchic approach based on swarm intelligence to solve the traveling salesman problem. Turkish Journal of Electrical Engineering & Computer Sciences, 2015, 23(1): 103 – 117.

[28] AMDAHL G M. Validity of the single processor approach

to achieving large scale computing capabilities. Proc. of the Spring Joint Computer Conference, 1967: 483 – 485.

[29] GUSTAFSON J L. Reevaluating Amdahl's law. Communications of the ACM, 1988, 31(5): 532 – 533.

Biographies

ZHANG Daoqing was born in 1994. He is currently working toward his M.E. degree in the School of Information Science and Engineering, Shandong University, China. His research interests include evolutionary computation and machine learning.
E-mail: zhang_dao_qing@163.com

JIANG Mingyan was born in 1964. He received his M.S. degree in 1992 and his Ph.D. degree in 2005 from Shandong University. He finished his postdoctoral research in communication signal and system at Spain Telecommunications Technology Center of Catalonia in 2007. Now he is a full professor and a doctoral supervisor in the School of Information Science and Engineering, Shandong University, China. He has published more than 200 professional papers and six books. His research interests include soft computing, signal and image processing, computer network, artificial intelligence and data mining.
E-mail: jiangmingyan@sdu.edu.cn