IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 8, 2019, accepted September 29, 2019. Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2019.2945570

Enhanced List-Based Simulated Annealing Algorithm for Large-Scale Traveling Salesman Problem

LIJIN WANG[1,2,3], RONGYING CAI[1], MIN LIN[1,2,3], AND YIWEN ZHONG[1,2,3]

[1]College of Computer and Information Science, Fujian Agriculture and Forestry University, Fuzhou 350002, China
[2]Key Laboratory of Smart Agriculture and Forestry (Fujian Agriculture and Forestry University), Fujian Province University, Fuzhou 350002, China
[3]Digital Fujian Research Institute of Big Data for Agriculture and Forestry, Fujian Agriculture and Forestry University, Fuzhou 350002, China

Corresponding author: Yiwen Zhong (yiwzhong@fafu.edu.cn)

This work was supported in part by the Nature Science Foundation of Fujian Province of China under Grant 2019J01401, and in part by the Special Fund for Scientific and Technological Innovation of Fujian Agriculture and Forestry University of China under Grant CXZX2016026 and Grant CXZX2016031.

ABSTRACT List-based simulated annealing (LBSA) algorithm is a novel simulated annealing algorithm where list-based cooling scheme is used to control the change of parameter temperature. Aiming to improve the efficiency of the LBSA algorithm for large-scale optimization problems, this paper proposes an enhanced LBSA (ELBSA) algorithm for solving large-scale traveling salesman problem (TSP). The ELBSA algorithm can drive more sampling at more suitable temperatures and from more promising neighborhoods. Specifically, heuristic augmented sampling strategy is used to ensure that more neighbors are from promising neighborhoods, systematic selection strategy is proposed to guarantee that each component of the current solution has a chance to be improved, and variable Markov chain length (VMCL), based on arithmetic sequence, is used to sample more neighbors at more suitable temperatures. Extensive experiments were performed to show the contribution of the heuristic augmented sampling strategy, and to verify the advantage of using systematic selection and VMCL. Comparative experiments, which were conducted on a wide range of large-scale TSP instances, show that the ELBSA algorithm is better than or competitive with most other state-of-the-art metaheuristics.

INDEX TERMS Simulated annealing, traveling salesman problem, list-based cooling scheme, heuristic augmented sampling, systematic selection, variable Markov chain length.

I. INTRODUCTION

Simulated annealing (SA) algorithm [1], [2] is a typical iterative metaheuristic with an explicit strategy to escape from local optima by allowing hill-climbing moves. Due to the randomness of selecting candidate solutions from neighborhoods of the current solution, its efficiency is not high, and its convergence process can be extremely slow for large-scale optimization problems. In the field of traveling salesman problem (TSP), two strategies, instance-based sampling and knowledge-based sampling, have been proposed to tackle this low efficiency of SA algorithm. Instance-based sampling was proposed by Wang et al. in multi-agent SA (MSA) algorithm [3] and was also used in list-based SA (LBSA) algorithm [4]. In instance-based sampling, solution components of other solutions are used to guide the generation of candidate solution. Knowledge-based sampling was proposed by Wang et al. in swarm SA (SSA) algorithm [5]. In knowledge-based sampling, knowledge from searching history is stored in pheromone matrix and is used to guide the generation of candidate solution. Both instance-based sampling and knowledge-based sampling use the experiences learned from its searching history to guide the selection of neighbors. As a result, the efficiency of sampling can be improved remarkably.

Although instance-based sampling has shown promising performance, it does not use heuristic information of TSP instance. Furthermore, the random selection strategy, which is used by the LBSA algorithm to select the solution component to be replaced, may also deteriorate its efficiency. Aiming to tackle those shortcomings, this paper proposes an enhanced list-based SA (ELBSA) algorithm

The associate editor coordinating the review of this manuscript and approving it for publication was Sudipta Roy.

with heuristic augmented instance-based sampling strategy for the TSP. Inspired by the systematic selection strategy used in SA algorithm [6] for quadratic assignment problem (QAP), the ELBSA algorithm uses a systematic way to select the solution component for producing a candidate solution. Finally, the ELBSA algorithm adopts a variable Markov chain length (VMCL) to conduct more trials at more profitable temperatures. Extensive experiments were carried out to show the advantage of heuristic augmented sampling, systematic selection, and VMCL. Comparative experiments, which were carried out on a wide range of large-scale benchmark TSP instances, show that the proposed algorithm is better than or competitive with most other state-of-the-art metaheuristics.

The rest of this paper is organized as follows. Section 2 provides a short description of the TSP, the LBSA algorithm, and metaheuristics for the large-scale TSP. Section 3 presents the three strategies used by the ELBSA algorithm and the pseudocode of the ELBSA algorithm. Section 4 fine-tunes the parameters and analyzes the behavior of the ELBSA algorithm. Section 5 compares the performance of the ELBSA algorithm with that of other state-of-the-art metaheuristics on a large number of TSP instances. Finally, in section 6 we summarize our study and possible future research directions.

II. RELATED WORK

A. TRAVELING SALESMAN PROBLEM

The TSP is one of the classical NP-hard problems in combinatorial optimization. The objective of the TSP is to find a shortest route that visits each city once and returns to the origin city. Consider a salesman who has to visit n cities, the TSP can be defined as follows. Suppose a matrix $D = (d_{i,j})_{n*n}$ is used to store the distances between all pairs of cities, where each element $d_{i,j}$ represents the length of the edge from city i to city j. We can use a linked list x to represent a solution. Each element x_j in x represents an edge from city j to city x_j. To guarantee that x is a valid solution, x must be a permutation of cities and $x_j \neq j$ for each $j \in \{1, 2, 3, ..., n\}$. Using a TSP instance with four cities as an example, suppose $x = (3, 1, 4, 2)$, then, x represents a solution with four edges $(e_{1,3}, e_{2,1}, e_{3,4}, e_{4,2})$. Therefore, the route of solution x is $1 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 1$. The goal of the TSP is to find a solution x that minimizes

$$f(x) = \sum_{j=1}^{n} d_{j,x_j} \tag{1}$$

B. LIST-BASED SIMULATED ANNEALING

The basic idea of SA algorithm is to allow accepting worse solutions in order to escape from local minima. The probability of accepting worse solution is decreased during the search under the control of the parameter temperature. SA algorithm uses the Metropolis acceptance criterion to decide whether to accept a candidate solution. Suppose x is the current solution and y is the candidate solution selected from x's neighbors. Their objective function values are $f(x)$ and $f(y)$. For a minimization problem, the candidate solution y is accepted as the new current solution x based on the acceptance probability

$$p = \begin{cases} 1, & if\ f(y) \leqslant f(x) \\ e^{-(f(y)-f(x))/t}, & otherwise \end{cases} \tag{2}$$

where $t > 0$ is the parameter temperature.

To use SA algorithm for a specific optimization problem, one must specify the cooling schedule for the parameter temperature. The LBSA [4] algorithm is a novel SA algorithm which uses list-based cooling scheme to control this parameter. In the list-based cooling scheme, all temperatures are stored in a priority list, where a larger value has a higher priority. In each generation, the maximum value in the list is used as the current temperature to calculate the acceptance probability for a candidate solution, and the temperature value in the list is updated adaptively according to the effectiveness of sampling. In the following subsections, we introduce the method for producing the initial temperature list and the temperature updating strategy used by the LBSA algorithm.

1) PRODUCTION OF THE INITIAL TEMPERATURE LIST

The LBSA algorithm uses parameter p_0 (initial acceptance probability) to produce the initial temperature values as follows. Suppose x is the current solution, y is the candidate solution, and $f(x)$ and $f(y)$ are their objective function values. According to the Metropolis acceptance criterion, the acceptance probability p of y can be calculated using Eq. 2. Conversely, if the acceptance probability p_0 is known, then we can calculate a corresponding temperature t_0 as int Eq. 3.

$$t_0 = \frac{-(f(y) - f(x))}{ln(p_0)} \tag{3}$$

The LBSA algorithm uses parameter p_0 and Eq. 3 to produce the initial temperature values for the temperature list.

2) TEMPERATURE UPDATING STRATEGY

In each generation, LBSA uses the maximum value t_{max} in the temperature list as the current temperature. For each t_{max}, suppose there are c times when the worse solution is accepted; we use d_i and p_i to represent the difference between the objective function values and the acceptance probability calculated by Eq. 2, where $i \in \{1, 2, \ldots, c\}$. In SA algorithm, whenever a worse solution is met, a random number r is created and is compared with p to decide whether to accept this worse solution. If r is less than p, then the worse solution is accepted. Therefore, for the ith accepted worse solution, we can use Eq. 3 to calculate a new temperature t_i as $t_i = -d_i/ln(r_i)$. The LBSA algorithm uses the average of all t_i $(\sum_{i=1}^{c} t_i/c)$ to replace the t_{max} in the temperature list. Because r_i is less than p_i and $t_{max} = -d_i/ln(p_i)$, t_i is always less than t_{max}. Therefore, $\sum_{i=1}^{c} t_i/c$ is also less than t_{max}. In this way, the temperature will become lower and lower as the search progresses.

C. METAHEURISTICS FOR THE LARGE-SCALE TSP

In recent years, many metaheuristics have been proposed to solve the TSP. Among these metaheuristics, some have been applied to large-scale TSP instances, e.g., ant algorithm [7]–[13], genetic algorithm [14], [15], particle swarm optimization [16], [17], cuckoo search [18], [19], bee-inspired algorithm [20]–[22], pigeon-inspired optimization algorithm [23], bat algorithm [24], [25], immune algorithm [26], invasive weed optimization [27], African buffalo optimization [28], symbiotic organisms search [29], [30], harmony search [31], evolutionary algorithm [32], neural network [33], hybrid algorithms [34], [35], and those based on local search [36]–[38].

Several versions of SA algorithm and hybrid SA algorithm have been proposed for solving the TSP [3]–[5], [16], [23], [30], [31], [36], [38]–[44]. Some of them have been applied to large-scale TSP instances. Wang *et al.* [3] proposed the multi-agent SA (MSA) algorithm with instance-based sampling that produces the candidate solution by perturbing the current solution with the solution component of other solutions. Zhan *et al.* [4] proposed the LBSA algorithm that uses list-based cooling scheme to control the change of temperature. Wang *et al.* [5] proposed the swarm SA (SSA) algorithm with knowledge-based sampling. In SSA, knowledge about the search experience is stored in pheromone matrix and the algorithm produces candidate solution by perturbing the current solution with solution component selected according to the pheromone matrix and heuristic information. Lin *et al.* [36] proposed an adaptive hybrid simulated annealing-tabu search algorithm (AHSA-TS) that combines the ideas of SA algorithm and tabu search algorithm. In AHSA-TS, the Metropolis acceptance criterion of SA algorithm is used to decide whether to accept the solution generated by tabu search. Geng *et al.* [38] proposed an adaptive simulated annealing algorithm with greedy search (ASA-GS), in which greedy search technique is embedded in SA algorithm to speed up the convergence rate. In ASA-GS, the Metropolis acceptance criterion is only used for the best solution of a number of bad solutions. Wang *et al.* [31] proposed an evolutionary harmony search algorithm, in which SA algorithm is used to improve its intensification ability. In the discrete comprehensive learning PSO (D-CLPSO) algorithm [16], the discrete pigeon-inspired optimization (D-PIO) algorithm [23], and the SA-based symbiotic organisms search (SOS-SA) algorithm [30], the Metropolis acceptance criterion is used to decide whether to accept the newly produced solutions. These studies have shown that SA algorithm can not only be independently used to solve TSP with promising performance but also be hybridized with population-based metaheuristics to improve the intensification ability of these algorithms.

III. ENHANCED LIST-BASED SA ALGORITHM

A. METHOD FOR CREATING THE INITIAL TEMPERATURE LIST

The LBSA algorithm needs a parameter p_0 to produce initial temperature values. It is very tedious to fine-tune parameter values, and those parameter values found by trial and error are quite often very poor. Therefore, an algorithm with fewer parameters is very attractive for practical users. Zhong *et al.* [16] suggested that parameter p_0 can be removed and that the difference between objective function values can be directly used as initial temperature values. To enhance the robustness of the initial temperature values, we delete some extreme values to reduce the effect of noise. Alg. 1 is the method for creating the initial temperature list, where parameter *len* is the length of the temperature list. As shown in line 4 of Alg. 1, the absolute value of $f(y)-f(x)$ is directly used as the initial temperature value. To reduce the impact of noise, $2\times len$ temperatures are first inserted into *lst*, and then, the top $len/2$ temperatures and the bottom $len/2$ temperatures are deleted from *lst* in line 9 and line 10 of Alg. 1.

Algorithm 1 Method for Creating the Initial Temperature List

Input: *len* The length of the initial temperature list
Output: A priority list of temperatures

```
1:  Create an initial solution x and an empty priority list lst
2:  while the length of lst is less than 2 × len do
3:      Produce candidate solution y from neighbors of x
4:      Insert |f(y) − f(x)| into lst
5:      if y is better than x then
6:          x = y
7:      end if
8:  end while
9:  Remove the top len/2 elements from lst
10: Remove the bottom len/2 elements from lst
11: Return lst.
```

B. SYSTEMATIC SELECTION AND HEURISTIC AUGMENTED SAMPLING STRATEGY

The original instance-based sampling [3], [4] strategy randomly selects city i and uses only history knowledge (represented by the solution of each agent) to guide the selection of the next visiting city of city i. We can improve the instance-based sampling strategy in two aspects: (1) use a systematic way to select city i and, (2) in addition to using history knowledge, also use heuristic information to guide the sampling. To use systematic selection strategy for city i, we can use the city index one-by-one as city i to create the candidate solution. In this way, each solution component has a chance to be improved. Conversely, if random selection is used, potential improvements might be missed at low temperatures because of the random nature of the search.

To use heuristic information to guide the sampling, we can construct a nearest neighbor list for each city i. If the selected edge $e_{i,j}$ is already in solution x, then we use the nearest neighbor list of city i to select an edge to disturb solution x. Heuristic information can not only speed up the convergence speed but also provide extra diversity. Alg. 2 is the pseudocode of the heuristic augmented instance-based sampling method, where parameter i is used to control whether to use systematic selection or random selection. A calling method

Algorithm 2 Heuristic Augmented Instance-Based Sampling Method

Input: i A city number
Input: x A solution represented as a linked list
Output: A new solution

1: Select another solution y randomly
2: $j = y_i$ //City j is the city following i in y
3: **if** $e_{i,j}$ is already in x **then**
4: $j = k$ where $y_k = i$ //City k is the city leading i in y
5: **while** $e_{i,j}$ is already in x **do**
6: Randomly select city j from the nearest city list of i
7: **end while**
8: **end if**
9: $x_1 = inverse(x, e_{i,j})$
10: $x_2 = swap(x, e_{i,j})$
11: $x_3 = blockInsert(x, e_{i,j})$
12: **Return** The best one among x_1, x_2, and x_3

may use a random way or a systematic way to create parameter i for Alg. 2. In line 6 of Alg. 2, heuristic information is used to select another edge in case the selected edge $e_{i,j}$ is already in solution x. After the new edge $e_{i,j}$ is selected, as in [16], [23], the best solution among the solutions produced by *inverse*, *swap*, and *blockInsert* is selected as the candidate solution. In Alg. 2, the $inverse(x, e_{i,j})$ operator produces a new solution by inverting the visiting sequence of cities between x_i and j. The $swap(x, e_{i,j})$ operator produces a new solution by swapping the positions of x_i and j. The $blockInsert(x, e_{i,j})$ operator produces a new solution by moving a block of cities led by j to the front of x_i. We use a TSP instance with six cities as an example to explain these operators in detail. Suppose the current solution x is (2,3,4,5,6,1), and the selected edge is $e_{1,5}$. The current solution x, *inverse*, *swap*, and *blockInsert* operators are depicted in Fig.1. Fig.1 (a) depicts the current solution x, the edges in the current solution, and its tour. Because the selected edge is $e_{1,5}$, we must put city 5 into the element x_1 whose current value is city 2. Fig.1 (b) depicts the *inverse* operator where the visiting sequence of cities between city 2 and city 5 has been inverted. Fig.1 (c) depicts the *swap* operator where the positions of city 2 and city 5 have been exchanged. Fig.1 (d) depicts the *blockInsert* operator where a block of two cities led by city 5 has been moved to the front of city 2. In this paper, the block size is a randomly selected number in the range 1 to 10, and the length of the nearest city list is 20.

C. VARIABLE Markov CHAIN LENGTH BASED ON ARITHMETIC SEQUENCE

In the classical homogeneous SA algorithm, the Markov chain length (MCL) at each temperature is a fixed number. For large-scale optimization problems, this may not be the most suitable strategy. Several studies have shown that trying more neighbors at optimal temperatures can improve SA's performance. In the field of QAP, Bölte and Thonemann [45]

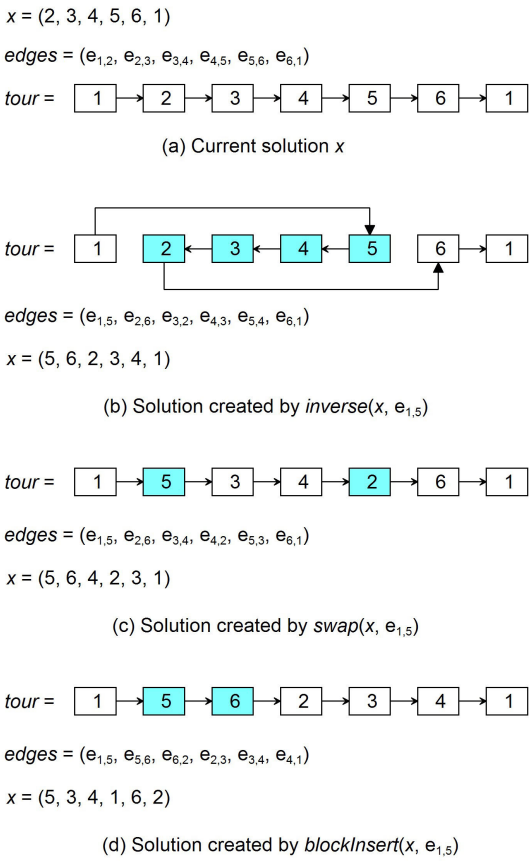

FIGURE 1. Neighbor operators used to produce candidate solutions.

proposed an annealing schedule with cosine-based oscillation; Misevičius [6] presented an annealing schedule with Lundy-Mees-function-based oscillation. In the field of the uncapacitated exam scheduling problem, Dowsland and Thompson [46] recommended that less time be spent exploring the neighborhood at high temperatures where most moves are accepted; Cheraitia and Haddadi [47] presented an annealing schedule with increasing MCL based on geometry sequence.

A typical search process of SA algorithm has the following features: (1) In the early stage with high temperature, most candidate solutions are accepted, even though those solutions are worse than the current solution. (2) In the late stage when the temperature is low, almost no candidate solution is accepted. (3) In the middle stage, there exist some profitable temperatures where the largest improvement is obtained. Inspired by the above features of SA algorithm, we propose a VMCL strategy based on arithmetic sequence such that more trials may be tried at more profitable temperatures. Suppose the fixed MCL for each temperature in the LBSA algorithm is M. To obtain the objective of trying more trials at profitable temperatures, we propose the VMCL sequence as follows. (1) Both the initial MCL and the final MCL are $M/2$. (2) The MCL in the generation $G \times pos$ has the largest value $3M/2$, where G is the maximum generation and pos is a parameter

to determine the position with the maximum MCL. (3) In the generations before $M \times pos$, MCL increases from $M/2$ to $3M/2$ according arithmetic sequence. (4) In the generations after $G \times pos$, MCL decreases from $3M/2$ to $M/2$ according arithmetic sequence. Suppose M is equal to 100 and G is equal to 1000, then, the effect of parameter *pos* for the VMCL is as shown in Fig.2.

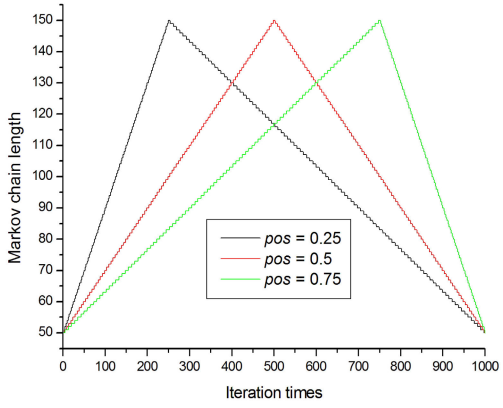

FIGURE 2. **Comparison of MCLs with different parameters *pos*.**

D. Pseudocode OF THE PROPOSED ELBSA ALGORITHM

The ELBSA algorithm uses a simple SA framework based on fixed iteration times for the outer loop and VMCL at each temperature. The detailed pseudocode of the ELBSA algorithm is listed in Algo. 3. In Algo. 3, parameters P, G, M, and *pos* represent the population size, maximum iteration times, fixed MCL, and relative position of the generation with the maximum MCL, respectively. Array *aSol* is used to store the current solution of each agent, and *idx* is the identifier of each agent. Array *aCity* is used to store the index of the selected city for each agent whose following city will be changed to produce the candidate solution. Array *aMCL* is used to store the MCLs for each generation. In line 12, the index of the selected city is changed to the next one in sequence, and line 13 uses this city as the parameter to call Algo. 2 to produce candidate solution. These two lines implement the systematic selection for the ELBSA algorithm. Variable c is used to record the number of times worse solution is accepted at each temperature, and variable s is used to store the sum of temperatures calculated by Eq. 3. In line 27, the average temperature s/c is used to update the temperature list. To have good initial solutions, the ELBSA algorithm uses the greedy random construction method to produce initial solutions. The time complexity of the ELBSA algorithm is analyzed as follows. In total, the ELBSA algorithm calls Algo. 2 to produce $G \times P \times M$ candidate solutions. In Algo.2, the *inverse*, *swap*, and *blockInsert* operators are used to created candidate solutions. Because we use linked list to represent a solution, the time complexity of *swap* and *blockInsert* is $O(1)$, and the time complexity of *inverse* is $O(n)$. Therefore, the time complexity of Algo.2 is $O(n)$. As a result, the time complexity of the ELBSA algorithm is $O(G \times P \times M \times n)$. In our simulation, the parameter M is set to the city number n of the TSP instance, so the time complexity of the ELBSA algorithm is $O(G \times P \times n^2)$.

Algorithm 3 Enhanced List-Based SA Algorithm

Input: P, G, M, and *pos*
Output: *bx* Best solution found

```
1:  Use the greedy random construction method to produce solutions aSol
2:  Use Alg. 1 to produce the initial temperature list for each agent
3:  Create an array aCity of size P that stores the selected city for each agent
4:  Use parameters pos and M to create an array aMCL of size G that stores the MCLs for each generation
5:  bx = best solution in aSol
6:  for g = 0 to G − 1 do
7:    for idx = 0 to P − 1 do
8:      x = aSol[idx]
9:      t = The maximum value in x's temperature list
10:     s = 0, c = 0, k = 0.
11:     while k++ < aMCL[g] do
12:       aCity[idx] = (aCity[idx] + 1)%n
13:       Produce solution y using Alg. 2 with parameters aCity[idx] and x
14:       Calculate the acceptance probability p of y
15:       Produce a random number r in the range [0, 1)
16:       if r < p then
17:         if f(y) − f(x) > 0 then
18:           s+ = −(f(y) − f(x))/ln(r)
19:           c++
20:         else if f(x) < f(bx) then
21:           bx = x
22:         end if
23:         x = y
24:       end if
25:     end while //end of inner loop
26:     if c > 0 then
27:       Replace the maximum value in x's temperature list with s/c
28:     end if
29:   end for //end of for each agent
30: end for //end of outer loop
31: Return bx
```

IV. BEHAVIOR ANALYSIS OF THE ELBSA ALGORITHM

To fine-tune the parameters and analyze the behavior of the ELBSA algorithm, six experiments have been carried out on benchmark TSP instances from TSPLIB [48]. The details of these benchmark TSP instances can be obtained from 'comopt.ifi.uni-heidelberg.de/software/TSPLIB95/'. In our study, we only consider symmetric TSP instances where the distance from city i to city j is the same as that from city j to city i. The first experiment uses full factorial experiment to find a suitable combination of the list length for the initial

temperature list and the position of the maximum MCL. The other five experiments use the one-variable-at-a-time approach to analyze the behavior of the ELBSA algorithm. The second experiment is used to analyze the effect of the list length. The third experiment is used to analyze the effect of the position of the maximum MCL. The fourth experiment is used to analyze the contribution of the heuristic augmented sampling. The fifth experiment is used to analyze the convergence behavior of the ELBSA algorithm. These four experiments are performed on Pr1002, D2103, Fnl4461, and Pla7397 instances. The sixth experiment is used to verify the effectiveness of the systematic selection and the VMCL. The last experiment is carried out on 33 large-scale TSP instances with city number ranging from 1000 to 85900. In all of the experiments, the termination condition is 1000 generations. In the first five experiments, the population size P is 10. To control the running time, the population size P of the last experiment is set according to Eq.(4). The percentage error of the best tour length (PE) and the percentage error of the average tour length (PEav) are used to compare different variants of the ELBSA algorithm. The following experiments were run on an Intel Core i5-3570 CPU, with 3.4 GHz and 8 GB of RAM. Java was used as the programming language.

$$P = \begin{cases} 50, & \text{if } n < 1000 \\ 30, & \text{else if } n < 2000 \\ 20, & \text{else if } n < 4000 \\ 10, & \text{else if } n < 50000 \\ 3, & \text{otherwise} \end{cases} \tag{4}$$

where n is the city number of TSP instance.

A. PARAMETER TUNING

Parameter tuning is important for metaheuristics to achieve a good performance. The ELBSA algorithm has two numerical parameters that need tuning, i.e., the list length (*len*) of the temperature list and the position (*pos*) of the maximum MCL. Full factorial experiment is used to find a reasonable combination of *len* and *pos*. After some pilot experiments, we set five levels for $len \in \{130, 140, 150, 160, 170\}$ and three levels for $pos \in \{0.25, 0.375, 0.5\}$. Experiments were carried out on the 26 large-scale TSP instances with city number less than 10000. For each combination of *len* and *pos*, we run the ELBSA algorithm 25 times for each instance. The average of PEav on these 26 TSP instances is used to compare the performance. Fig.3 presents the histogram of the simulation results. Fig.3 shows the following findings: (1) the best combination of *len* and *pos* is 150 and 0.375; (2) the parameter *len* is more robust when the parameter *pos* is smaller; (3) the parameter *pos* is more robust when the parameter *len* is larger; and (4) the parameter *len* and the parameter *pos* are relevant. In general, the smaller *len* is, the smaller *pos* should be because optimal *pos* is dependent on the most suitable temperature; when *len* is smaller, the temperature will decrease more quickly. As a result, the optimal *pos* should also be smaller.

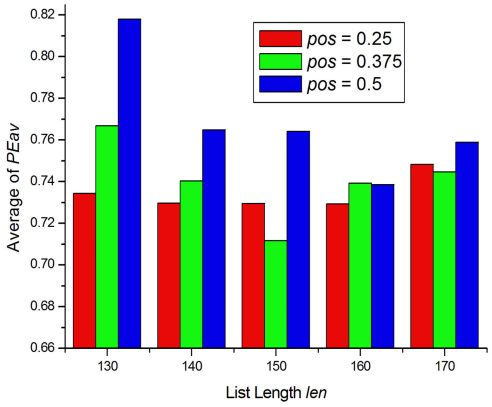

FIGURE 3. Comparison of the ELBSA algorithm with different combinations of *len* and *pos*.

B. EFFECT OF THE LIST LENGTH OF THE INITIAL TEMPERATURE LIST

To analyze the effect of the list length *len* on the behavior of the ELBSA algorithm, we test 11 different *len* from 100 to 200 with a step 10. For each *len*, we run the ELBSA algorithm 50 times for each instance and calculate the percentage error of the average tour lengths relative to the best known tour length. Fig. 4 presents the relation between the percentage error and *len*. Similar to those results obtained by [4] and [16], Fig. 4 shows the following: (1) The optimal list length is instance-dependent; for example, the optimal *len* is 130 for the Pr1002 instance, but 150 for the Pla7397 instance. (2) The list length is more robust on small instances than on large instances; for example, both a small and a large list length will notably deteriorate the performance of the ELBSA algorithm on the Pla7397 instance.

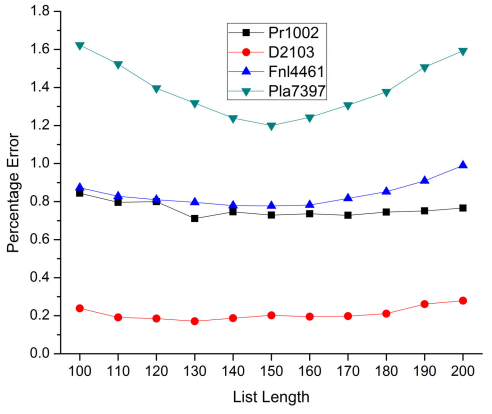

FIGURE 4. Comparison of the ELBSA algorithm with different list lengths *len*.

C. EFFECT OF THE POSITION OF THE MAXIMUM VMCL

To analyze the effect of the position *pos* of the maximum VMCL on the behavior of the ELBSA algorithm, we test 8 different *pos* from 0 to 1 with a step 1/8. For each *pos*, we run the ELBSA algorithm 50 times for each instance and calculate the percentage error of the average tour length

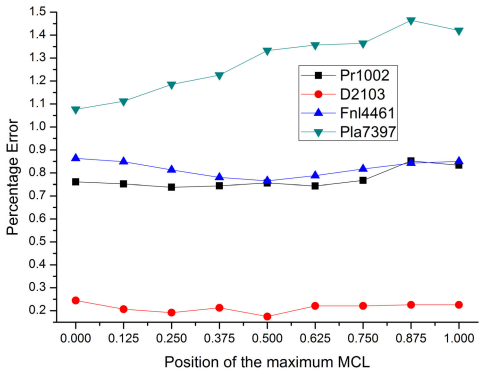

FIGURE 5. Comparison of the ELBSA algorithm with different positions *pos* of the maximum VMCL.

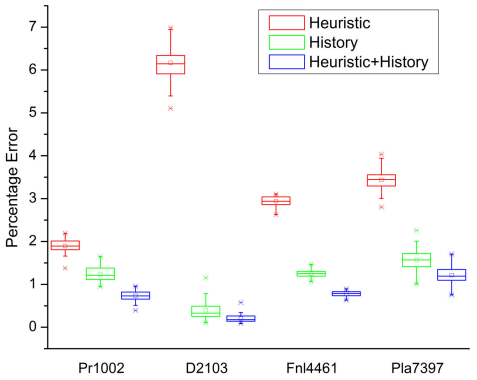

FIGURE 6. Comparison of the ELBSA algorithm with different sampling strategies.

relative to the best known tour length. Fig. 5 presents the relation between the percentage error and the *pos*. Fig. 5 shows the following: (1) The optimal *pos* is instance-dependent; for example, the optimal *pos* is 0.5 for the Fnl4461 instance, but 0 for the Pla7397 instance. (2) The *pos* is more robust on small instances than on large instances.

D. CONTRIBUTION OF THE HEURISTIC AUGMENTED SAMPLING STRATEGY

In the ELBSA algorithm, heuristic information is used to enhance the instance-based sampling. To analyze the effectiveness of the heuristic information, we compare the performances of the ELBSA algorithm with heuristic information and the ELBSA algorithm without heuristic information. To accomplish this, we implemented three variants of the ELBSA algorithm: ELBSA with heuristic-based sampling only, ELBSA with instance-based sampling only, and ELBSA with both heuristic-based sampling and instance-based sampling. Fig. 6 compares the contributions of the heuristic-based sampling and the instance-based sampling. Fig. 6 shows that using instance-based sampling only is better than using heuristic-based sampling only. Fig. 6 also shows than ELBSA with both heuristic-based sampling and instance-based sampling has best performance among these three variants. These results verify that the heuristic information has a positive effect on the performance of the ELBSA algorithm.

E. CONVERGENCE ANALYSIS

To analyze the convergence behavior of the ELBSA algorithm, we compare the temperature decrease process and PEav convergence process with different list lengths of the temperature list. The used list lengths of the temperature list include 100, 150, and 200. The four figures in Fig. 7 present the temperature decrease process of the ELBSA algorithm on the four TSP instances. The four figures in Fig. 8 present the PEav convergence process of the ELBSA algorithm. These figures clearly show that the list length of the temperature list determines the convergence speed of the ELBSA algorithm. If the list length is too small, then the ELBSA algorithm will converge quickly and may be trapped in the local optima more easily. Conversely, if the list length is too large, then the ELBSA algorithm may spend too much time at high temperatures and has insufficient intensification ability in the late stage.

F. ADVANTAGE OF SYSTEMATIC SELECTION AND VARIABLE MCL

The ELBSA algorithm uses the systematic selection strategy and the variable MCL strategy to improve its performance. To analyze the effect of the systematic selection and the variable MCL, we compare the performances of ELBSA algorithm with two variants of it, i.e., ELBSA with random selection and fixed MCL (ELBSA-RF) and ELBSA with systematic selection and fixed MCL (ELBSA-SF). Experiments were carried out on 33 large-scale TSP instances with city numbers from 1000 to 85900. Tab. 1 shows the simulation results. In Tab.1, the PEav of ELBSA-SF is highlighted in bold if it is better than the PEav of ELBSA-RF, and the PEav of ELBSA is highlighted in bold if it is better than the PEav of ELBSA-SF. The average PE of ELBSA-RF, ELBSA-SF, and ELBSA is 0.53, 0.463, and 0.459 respectively. The average PEav of ELBSA-RF, ELBSA-SF, and ELBSA is 0.73, 0.703, and 0.673 respectively. Among the 33 instances, ELBSA-SF obtains better PEav than ELBSA-RF on 25 instances, and ELBSA obtains better PEav than ELBSA-SF on 26 instances. The Wilcoxon signed ranks test [49] is used to compare the PEav of ELBSA-RF and ELBSA-SF, where R^+ denotes the sum of ranks for the instances in which ELBSA-SF outperforms ELBSA-RF, and R^- indicates the sum of ranks for the opposite. The computed R^+, R^-, and p-value are 406.5, 121.5, and 4.49e-03, respectively. This result means that the systematic selection can significantly improve the performance of the ELBSA algorithm. The Wilcoxon signed ranks test is also used to compare the PEav of ELBSA-SF and ELBSA, where R^+ denotes the sum of ranks for the instances in which ELBSA outperforms ELBSA-SF, and R^- indicates the sum of ranks for the opposite. The computed R^+, R^-, and p-value are 399.5, 128.5, and 6.60e-03, respectively. This result means that the variable MCL can significantly improve the performance of the ELBSA algorithm.

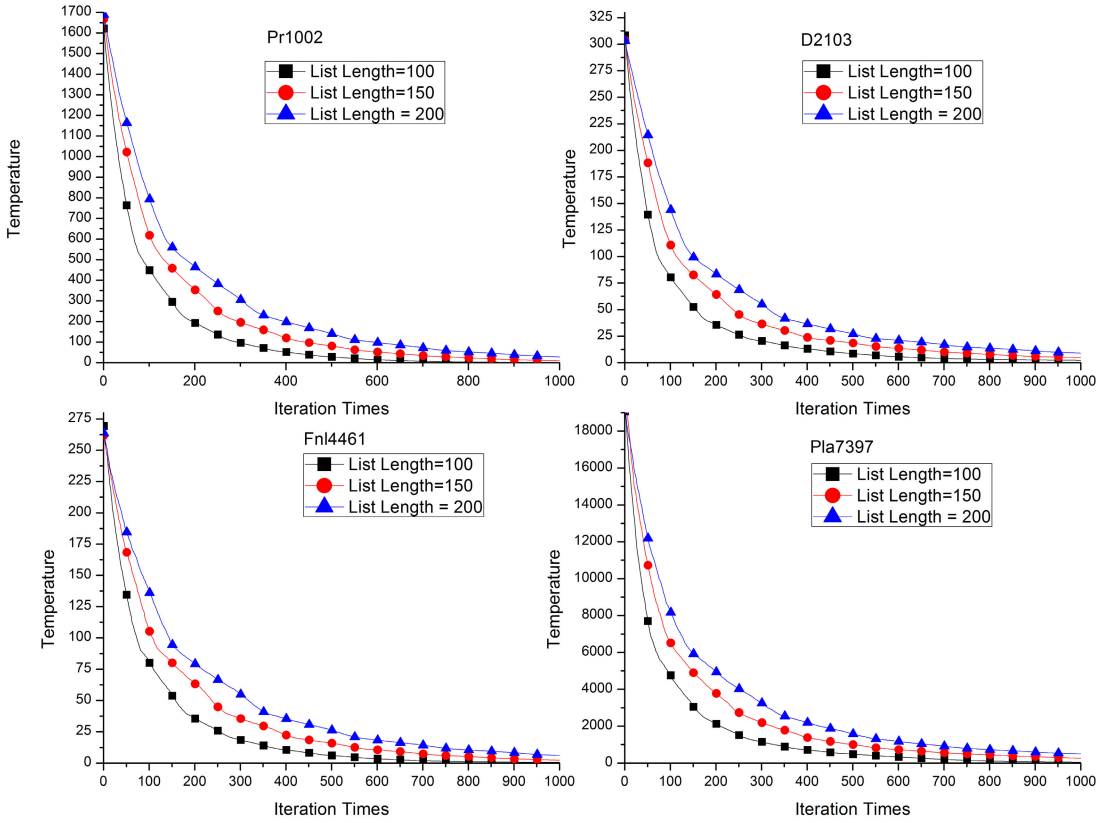

FIGURE 7. Comparison of the temperature decrease processes of the ELBSA algorithm with different list lengths.

V. COMPETITIVENESS OF THE ELBSA ALGORITHM

To observe the competitiveness of the ELBSA algorithm, ELBSA's performance was compared with that of 32 state-of-the-art algorithms on a large number of large-scale benchmark TSP instances. In all of the following experiments, the length of the temperature list is 150, the position parameter of the VMCL is 0.375, the maximum generation is 1000, and the total sampling times is $1000 * P * n$ where n is the city number. The population size P is set according to Eq. 4 such that the running time of the ELBSA algorithm is less than the running time of the competitor as far as possible. In the following subsections, Wilcoxon rank-sum test or Wilcoxon signed ranks test is used to compare the performance of the ELBSA algorithm with that of other state-of-the-art algorithms. In the Wilcoxon signed ranks test, R^+ denotes the sum of ranks for the instances in which the ELBSA algorithm outperforms the competitive one, and R^- indicates the sum of ranks for the opposite.

A. COMPARISON WITH THE LBSA ALGORITHM

The ELBSA algorithm is an enhanced LBSA algorithm. To observe the advantage of the ELBSA algorithm over the LBSA algorithm. The ELBSA algorithm was compared with the LBSA algorithm in detail on 33 large-scale TSP instances with integer distance. We run the ELBSA algorithm and the LBSA algorithm 25 times on each instance and calculate the statistical results. Tab.2 shows the simulation results, where the better one is highlighted in bold. In Tab.2, PEb, PEw, PEav, PEme, and Std represent the percentage error of the best solution, the percentage error of the worst solution, the percentage error of the average solution, the percentage error of the median solution, and the standard deviation, respectively. The Wilcoxon rank-sum test [49] was used to test whether the two algorithms have significant difference on each instance. If the p-value is less than 0.05, then the two algorithms have a significant difference and the corresponding p-value is highlighted in bold. Among the 33 instances, the ELBSA algorithm achieves a better performance on 29 instances and is significantly better than the LBSA algorithm on 25 instances, whereas the LBSA algorithm achieves a better performance on 4 instances and is significantly better than the ELBSA algorithm on 2 instances. The Wilcoxon signed ranks test is used to compare the PEav of the ELBSA algorithm and the LBSA algorithm. The computed R^+, R^-, and p-value are 493, 35, and 1.15e-05, respectively. This result means that the ELBSA algorithm is significantly better than the LBSA algorithm.

B. COMPARISON WITH OTHER SA-RELATED ALGORITHMS

Among the SA-related algorithms, we compare the ELBSA algorithm with ASA-GS [38] and SOS-SA [30] on 17 benchmark instances with float distance. Tab. 3 presents the

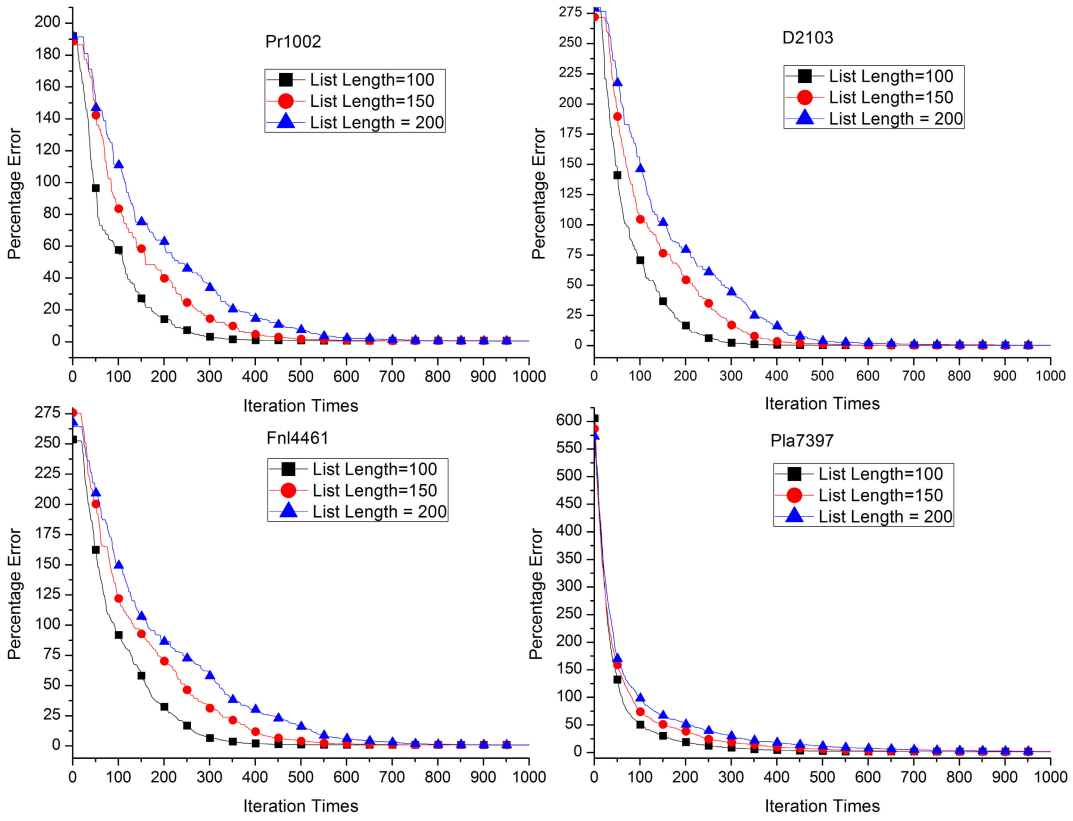

FIGURE 8. Comparison of the convergence processes of the ELBSA algorithm with different list lengths.

simulation results, where the best result is highlighted in bold. The average PEav values of ASA-GS, SOS-SA, and ELBSA are 3.32, 1.9, and 0.93, respectively. Among the 17 instances, ELBSA obtains the best PEav on 13 instances, and SOS-SA obtains the best PEav on 4 instances. The Wilcoxon signed ranks test is used to compare the PEav of ASA-GS, SOS-SA, and ELBSA. For ELBSA and ASA-GS, the computed R^+, R^-, and p-value are 136, 0, and 2.93e-4, respectively. For ELBSA and SOS-SA, the computed R^+, R^-, and p-value are 120, 16, and 4.18e-03, respectively. This result means that the ELBSA algorithm is significantly better than ASA-GS and SOS-SA.

The ELBSA algorithm is also compared with AHSA-TS [36] and D-CLPSO [16] on 20 benchmark instances with integer distance. Tab. 4 shows the simulation results, where the best result is highlighted in bold. The average PEav values of AHSA-TS, D-CLPSO, and ELBSA are 1.402, 0.828, and 0.598, respectively. Among the 20 instances, the ELBSA algorithm obtains the best PEav on 18 instances, and the D-CLPSO algorithm obtains the best PEav on 2 instances. The Wilcoxon signed ranks test is used to compare the PEav of AHSA-TS, D-CLPSO, and ELBSA. For ELBSA and AHSA-TS, the computed R^+, R^-, and p-value are 189, 1, and 1.03e-04, respectively. For ELBSA and D-CLPSO, the computed R^+, R^-, and p-value are 176, 14, and 6.81e-04, respectively. This result means that the ELBSA algorithm is significantly better than AHSA-TS and D-CLPSO.

C. COMPARISON WITH OTHER STATE-OF-THE-ART ALGORITHMS

To further observe the competitiveness of the ELBSA algorithm among state-of-the-art metaheuristics, the ELBSA algorithm is also compared with 27 newly published metaheuristics. Three of these metaheuristics are compared in detail, and the other 24 metaheuristics are compared in brief.

The ELBSA algorithm is compared in detail with honey bees mating optimization (HBMO) [21], massively parallel neural network (MPNN) [33], and D-PIO [23] on 33 benchmark instances with integer distance. Tab. 5 shows the simulation results, where the best result is highlighted in bold. The average PEav values of HBMO, MPNN, D-PIO, and ELBSA are 0.076, 8.826, 0.739, and 0.673, respectively. In terms of the average PEav for all of the instances, the HBMO algorithm, which is augmented by the 2-opt, 2.5-opt and 3-opt operators, achieves the best performance. The ELBSA algorithm outperforms the other two algorithms. The Wilcoxon signed ranks test is used to compare the PEav of MPNN, D-PIO, and ELBSA. For ELBSA and MPNN, the computed R^+, R^-, and p-value are 528, 0, and 5.4e-7, respectively. For ELBSA and D-PIO, the computed R^+, R^-, and p-value are

TABLE 1. Comparison of the ELBSA algorithm with ELBSA-RF and ELBSA-SF.

No.	Instance	Optimal	ELBSA-RF			ELBSA-SF			ELBSA		
			PE	PEav	Time(s)	PE	PEav	Time(s)	PE	PEav	Time(s)
1	Dsj1000	18659688	0.089	0.336	17.5	0.104	0.349	16.7	0.137	**0.304**	16.3
2	Pr1002	259045	0.518	0.655	14.6	0.414	**0.606**	13.0	0.257	**0.586**	12.8
3	U1060	224094	0.148	0.35	15.9	0.232	0.404	14.2	0.225	**0.363**	14.1
4	Vm1084	239297	0.071	0.318	18.6	0.059	**0.287**	17.1	0.097	**0.213**	16.8
5	Pcb1173	56892	0.237	0.534	18.8	0.148	**0.432**	16.7	0.128	**0.424**	16.5
6	D1291	50801	0.311	0.508	20.8	0.264	0.55	18.1	0.232	0.572	17.7
7	Rl1304	252948	0.039	0.315	23.1	0.057	0.329	21.1	0.02	**0.195**	20.7
8	Rl1323	270199	0.12	0.353	23.7	0.053	0.393	21.6	0.125	**0.354**	21.2
9	Nrw1379	56638	0.339	0.457	24.9	0.325	**0.441**	22.5	0.318	**0.418**	22.3
10	Fl1400	20127	0.005	0.223	26.3	0.005	0.24	23.8	0.015	0.395	23.7
11	U1432	152270	0.265	0.375	25.7	0.282	0.402	23.0	0.295	**0.386**	23.0
12	Fl1577	22249	0.139	0.237	28.3	0.13	**0.218**	24.5	0.139	**0.211**	24.2
13	D1655	62128	0.46	0.696	30.0	0.246	**0.49**	26.2	0.171	**0.45**	25.8
14	Vm1748	336556	0.278	0.477	37.7	0.21	**0.388**	35.0	0.251	0.41	34.7
15	U1817	57201	0.49	0.63	33.8	0.404	**0.613**	30.2	0.372	**0.548**	29.5
16	Rl1889	316536	0.333	0.647	41.0	0.053	**0.572**	37.8	0.171	**0.516**	37.2
17	D2103	80450	0.027	0.153	27.1	0.058	0.155	24.1	0.092	**0.148**	23.3
18	U2152	64253	0.689	0.857	29.2	0.529	**0.815**	26.1	0.509	**0.754**	25.7
19	U2319	234256	0.47	0.58	37.3	0.47	**0.566**	34.0	0.495	0.631	34.1
20	Pr2392	378032	0.644	0.854	34.2	0.561	**0.784**	30.3	0.515	**0.723**	29.7
21	Pcb3038	137694	0.491	0.708	51.9	0.394	**0.658**	46.5	0.444	**0.597**	45.7
22	Fl3795	28772	0.754	1.45	83.1	0.33	1.505	76.7	0.775	1.572	75.4
23	Fnl4461	182566	0.805	0.888	53.5	0.73	**0.847**	50.0	0.566	**0.784**	49.4
24	Rl5915	565530	0.752	1.088	85.0	0.631	**1.033**	82.6	0.523	**0.929**	80.1
25	Rl5934	556045	0.781	1.023	91.5	0.59	**0.986**	87.5	0.705	**0.974**	85.2
26	Pla7397	23260728	0.879	1.226	132.6	0.876	**1.186**	127.5	0.831	1.268	126.6
27	Rl11849	923288	0.867	1.025	453.1	0.793	**1.002**	458.7	0.817	**0.943**	438.4
28	Usa13509	19982859	0.903	1.017	450.3	0.873	**0.98**	431.7	0.821	**0.939**	422.2
29	Brd14051	469388	0.936	1	493.6	0.908	**0.986**	474.2	0.815	**0.902**	459.1
30	D15112	1573084	0.84	0.939	734.4	0.818	**0.907**	735.6	0.739	**0.813**	715.7
31	D18512	645244	0.928	0.99	824.0	0.904	**0.965**	790.2	0.846	**0.886**	769.7
32	Pla33810	66050535	1.524	1.71	2038.2	1.515	**1.661**	1930.4	1.355	**1.539**	1903.0
33	Pla85900	142383704	1.365	1.458	4060.6	1.313	**1.441**	3795.738	1.342	1.456	3827.2
		Average	0.53	0.73	305.463	0.463	**0.703**	289.81	0.459	**0.673**	286.9

435.5, 92.5, and 7.8*e*-4, respectively. This result means that the ELBSA algorithm is significantly better than MPNN and D-PIO.

The ELBSA algorithm is also compared in brief with 24 metaheuristics published in recent years. These metaheuristics include MSA [3], SSA [5], modified ant system (MAS) [8], two-stage hybrid swarm intelligence optimization (TSHACO) [35], ant colony extended (ACE) [11], ant colony optimization algorithm with two-stage updating pheromone (TSACO) [12], effective heuristics for ant colony optimization (ESACO) [7] where 2-opt local search is used to improve solution constructed by each ant, hybrid Max-Min ant system (HMMA) [10], permutation-coded genetic algorithm (PCGA) [15], genetic simulated annealing ant colony system with particle swarm optimization techniques (HGA) [34], set-based PSO where the global best solution is further improved by 3-opt local search operator (S-PSOg) [17], set-based PSO where iteration best positions are further improved by 3-opt local search operator (S-PSOi) [17], bee colony optimization with local search (BCO) [22], hybrid discrete artificial bee colony algorithm with threshold acceptance criterion (HDABC) [20], improved discrete cuckoo search algorithm (IDCS) [18], discrete cuckoo search algorithm (DCS) [19], discrete bat algorithm (DBA) [25], improved discrete bat algorithm (IDBA) [24], evolutionary harmony search algorithm (EHS) [31], immune algorithm combined with estimation of distribution (IA-EDA) [26], discrete invasive weed optimization algorithm (DIWO) [27], African buffalo optimization (ABO) [28], discrete symbiotic organisms search (DSOS) [29], and dynamic multiscale region search algorithm (DMRSA) [37]. Tab. 6 highlights the comparison of the ELBSA algorithm with these 24 metaheuristics. The column Distance Type denotes the type of distance between cities, which may be float or integer. The column N denotes the number of total instances. The column Dims denotes the city numbers of the smallest and the largest instances. The columns PEav and PEav1 denote the PEav values of the compared algorithm and the ELBSA algorithm, respectively. Because many of the metaheuristics were tested on a small number of large-scale TSP instances, instances with a city number of more than 100 are also included for comparison for these metaheuristics which were tested only on a small number of large-scale TSP instances. We run the ELBSA algorithm 25 times for each TSP instance to obtain the statistical results, and the results of other algorithms are directly obtained from the corresponding papers. The PEav values of the TSP instances were used with the Wilcoxon signed ranks test to compare the ELBSA algorithm with the other algorithms. Except for ESACO and S-PSOi, both of which are heavily enhanced by local search method,

TABLE 2. Comparison of the ELBSA algorithm with the LBSA algorithm in detail.

No.	Instance	LBSA					ELBSA					*p*-value
		PEb	PEw	PEav	PEme	Std	PEb	PEw	PEav	PEme	Std	
1	Dsj1000	0.233	0.883	0.535	0.52	0.143	0.137	0.499	**0.304**	0.292	0.1	**7.14e-07**
2	Pr1002	0.436	0.926	0.696	0.702	0.101	0.257	0.762	**0.586**	0.603	0.11	**5.73e-04**
3	U1060	0.171	0.598	0.409	0.413	0.117	0.225	0.578	**0.363**	0.342	0.087	1.60e-01
4	Vm1084	0.153	0.647	0.383	0.361	0.145	0.097	0.499	**0.213**	0.194	0.089	**4.61e-05**
5	Pcb1173	0.264	0.826	0.54	0.524	0.142	0.128	0.684	**0.424**	0.469	0.146	**2.76e-02**
6	D1291	0.366	1.276	0.801	0.762	0.219	0.232	1.161	**0.572**	0.502	0.239	**6.16e-04**
7	Rl1304	0.064	0.723	0.331	0.311	0.17	0.02	0.442	**0.195**	0.145	0.136	**4.21e-03**
8	Rl1323	0.246	0.752	0.471	0.443	0.144	0.125	0.522	**0.355**	0.379	0.123	**2.15e-02**
9	Nrw1379	0.381	0.77	0.495	0.482	0.089	0.318	0.517	**0.418**	0.418	0.057	**1.24e-03**
10	Fl1400	0.03	0.86	**0.331**	0.298	0.255	0.015	1.406	0.395	0.303	0.386	9.92e-01
11	U1432	0.214	0.457	**0.328**	0.328	0.065	0.295	0.558	0.386	0.366	0.061	**5.36e-03**
12	Fl1577	0.121	1.685	0.324	0.207	0.323	0.139	0.445	**0.211**	0.202	0.061	2.77e-01
13	D1655	0.237	0.776	0.521	0.486	0.166	0.171	0.697	**0.45**	0.464	0.116	2.48e-01
14	Vm1748	0.323	0.701	0.494	0.496	0.079	0.251	0.572	**0.41**	0.425	0.082	**1.24e-03**
15	U1817	0.505	1.143	0.781	0.801	0.141	0.372	0.745	**0.548**	0.561	0.097	**4.32e-07**
16	Rl1889	0.363	1.268	0.771	0.794	0.22	0.171	0.889	**0.516**	0.511	0.171	**1.49e-04**
17	D2103	0.04	0.288	0.164	0.15	0.068	0.092	0.305	**0.148**	0.124	0.052	4.55e-01
18	U2152	0.803	1.212	0.99	0.977	0.116	0.509	0.996	**0.754**	0.756	0.114	**1.99e-07**
19	U2319	0.295	0.4	**0.346**	0.33	0.028	0.495	0.758	0.632	0.625	0.062	**1.33e-09**
20	Pr2392	0.701	1.286	0.927	0.925	0.131	0.515	0.919	**0.723**	0.723	0.096	**5.29e-07**
21	Pcb3038	0.581	0.936	0.775	0.792	0.098	0.444	0.755	**0.597**	0.584	0.074	**3.34e-07**
22	Fl3795	0.514	2.568	**1.552**	1.616	0.488	0.775	2.471	1.572	1.571	0.552	8.16e-01
23	Fnl4461	1.093	1.393	1.25	1.271	0.078	0.566	0.88	**0.784**	0.795	0.064	**1.33e-09**
24	Rl5915	1.022	1.968	1.491	1.442	0.227	0.523	1.1	**0.929**	0.939	0.127	**2.43e-09**
25	Rl5934	1.18	1.985	1.573	1.554	0.22	0.705	1.221	**0.974**	0.924	0.148	**2.43e-09**
26	Pla7397	1.062	1.958	1.432	1.475	0.215	0.831	1.692	**1.269**	1.209	0.212	**1.57e-02**
27	Rl11849	1.355	1.83	1.664	1.661	0.105	0.817	1.082	**0.943**	0.941	0.071	**1.33e-09**
28	Usa13509	1.4	1.727	1.563	1.566	0.08	0.821	1.079	**0.939**	0.936	0.068	**1.33e-09**
29	Brd14051	1.305	1.61	1.475	1.488	0.069	0.815	1.099	**0.902**	0.893	0.061	**1.33e-09**
30	D15112	1.327	1.527	1.405	1.389	0.049	0.739	0.902	**0.813**	0.804	0.042	**1.33e-09**
31	D18512	1.356	1.535	1.455	1.459	0.042	0.846	0.924	**0.886**	0.888	0.022	**1.33e-09**
32	Pla33810	1.499	1.979	1.821	1.82	0.104	1.355	1.773	**1.539**	1.545	0.094	**2.71e-08**
33	Pla85900	3.062	3.359	3.216	3.227	0.075	1.342	1.553	**1.456**	1.478	0.052	**1.33e-09**
	Average	0.688	1.268	0.949	0.942	0.143	0.459	0.924	**0.673**	0.664	0.120	

TABLE 3. Comparison of ELBSA with ASA-GS and SOS-SA on 17 benchmark instances with float distance.

No.	Instance	Optimal	ASA-GS		SOS-SA		ELBSA	
			PEav	Time(s)	PEav	Time(s)	PEav	Time(s)
1	Pr1002	259045	2.01	164.42	1.06	12.81	**0.585**	9.44
2	Pcb1173	56892	1.63	193.08	1.19	8.73	**0.531**	13.25
3	D1291	50801	2.85	214.64	**0.96**	12.08	1.248	14.26
4	Rl1323	270199	1.2	210.16	0.56	11.02	**0.329**	17.33
5	Fl1400	20127	3.25	232.02	**0.52**	14.74	1.3	19.24
6	D1655	62128	3.26	281.88	3.19	16.19	**1.058**	22.13
7	Vm1748	336556	2.18	276.98	**0.05**	18.27	0.405	30.05
8	U2319	234256	1.06	410.97	**0.46**	18.11	0.632	33.26
9	Pcb3038	137694	2.57	554.28	1.46	25.67	**0.672**	43.32
10	Fnl4461	182566	2.65	830.9	1.63	32.74	**0.907**	46.78
11	Rl5934	556045	3.48	1043.95	1.83	49.99	**0.953**	78.28
12	Pla7397	23260728	3.89	1245.22	2.32	98.72	**1.242**	119.48
13	Usa13509	19982859	4.14	2016.05	7.09	313.11	**0.926**	399.60
14	Brd14051	469385	3.58	2080.5	1.8	370.88	**0.983**	457.05
15	D18512	645238	3.75	2593.97	2.2	601.85	**0.979**	746.93
16	Pla33810	66048945	5.27	4199.88	3.07	1899.99	**1.61**	1888.49
17	Pla85900	142382641	9.62	8855.13	2.84	7591.83	**1.448**	4030.31
		Average	3.32	1494.35	1.90	652.75	**0.93**	468.78

the ELBSA algorithm outperforms the other 22 metaheuristics. The results of the Wilcoxon signed ranks test with a 0.05 significance level show that, except for the ESACO, S-PSOi, and MAS algorithms, the ELBSA algorithm is significantly better than the other metaheuristics, where the *p*-value is highlighted in bold. It is worth noting that although the ELBSA algorithm is worse than ESACO in terms of the average PEav, the ELBSA algorithm outperforms ESACO on the four largest instances.

When comparing the ELBSA algorithm with other metaheuristics, we found that it is very difficult to fairly compare existing metaheuristics. There are several reasons,

TABLE 4. Comparison of ELBSA with AHSA-TS and D-CLPSO on 20 benchmark instances with integer distance.

No.	Instance	Optimal	AHSA-TS		D-CLPSO		ELBSA	
			PEav	Time(s)	PEav	Time(s)	PEav	Time(s)
1	Pr1002	259045	0.998	47.886	0.76	13.842	**0.586**	12.844
2	Pcb1173	56892	0.649	58.969	0.737	17.222	**0.424**	16.524
3	D1291	50801	1.08	197.472	0.645	19.149	**0.572**	17.659
4	Rl1304	252948	1.186	64.243	0.431	20.804	**0.195**	20.739
5	Rl1323	270199	0.671	68.257	0.41	20.964	**0.354**	21.16
6	Fl1400	20127	1.76	527.347	**0.328**	24.208	0.395	23.656
7	Fl1577	22249	1.187	81.542	0.219	27.248	**0.211**	24.22
8	D1655	62128	1.324	235.178	0.819	28.334	**0.45**	25.75
9	Vm1748	336556	1.165	99.985	0.678	33.21	**0.41**	34.714
10	Rl1889	316536	1.768	101.979	0.716	36.321	**0.516**	37.194
11	D2103	80450	1.171	299.784	0.381	39.257	**0.148**	23.318
12	U2319	234256	0.485	132.388	**0.385**	48.055	0.631	34.095
13	Pr2392	378032	1.479	133.704	1.042	30.211	**0.723**	29.704
14	Pcb3038	137694	1.083	168.122	0.998	43.985	**0.597**	45.68
15	Fnl4461	182566	1.128	275.655	1.222	43.685	**0.784**	49.386
16	Rl5934	556045	1.915	379.243	1.205	75.414	**0.974**	85.205
17	Pla7397	23260728	2.902	661.815	1.427	129.106	**1.268**	126.588
18	Usa13509	19982859	1.886	1119.91	1.466	488.165	**0.939**	422.184
19	Brd14051	469385	2.021	1230.27	1.328	543.184	**0.902**	459.056
20	D18512	645238	2.189	3328.67	1.362	837.399	**0.886**	769.659
		Average	1.402	460.621	0.828	125.988	**0.598**	113.967

TABLE 5. Comparison of ELBSA with HBMO, MPNN, and MAS algorithms on 33 benchmark instances with integer distance.

No.	Instance	HBMO		MPNN		D-PIO		ELBSA	
		PEav	Time(s)	PEav	Time(s)	PEav	Time(s)	PEav	Time(s)
1	Dsj1000	**0.012**	80.29	6.46	61.31	0.388	16.62	0.304	16.28
2	Pr1002	**0.001**	80.57	4.78	55.76	0.51	14.12	0.586	12.84
3	U1060	**0**	80.68	5.12	67.94	0.374	15.30	0.363	14.11
4	Vm1084	**0.005**	85.21	5.86	68.53	0.327	17.43	0.213	16.83
5	Pcb1173	**0.003**	89.28	7.5	70.53	0.392	17.76	0.424	16.52
6	D1291	**0**	91.14	9.66	81.58	0.668	19.29	0.572	17.66
7	Rl1304	**0**	103.29	10	78.64	0.313	21.48	0.195	20.74
8	Rl1323	**0**	113.43	9.45	82.52	0.408	21.98	0.354	21.16
9	Nrw1379	**0.009**	121.89	4.61	80.42	0.519	23.23	0.418	22.30
10	Fl1400	**0.011**	198.67	4.32	234.54	0.419	24.54	0.395	23.66
11	U1432	**0.016**	200.01	5.02	96.78	0.388	23.90	0.386	22.99
12	Fl1577	**0.022**	227.28	17.46	107.54	0.178	25.32	0.211	24.22
13	D1655	**0.122**	241.67	9.6	99.4	0.369	27.23	0.45	25.75
14	Vm1748	**0.189**	257.81	6.68	132.86	0.454	33.77	0.41	34.71
15	U1817	**0.028**	289.12	9.68	125.03	0.561	30.27	0.548	29.52
16	Rl1889	**0.017**	291.57	9.54	129.02	0.688	36.59	0.516	37.19
17	D2103	**0.041**	350.78	19.15	132.5	0.145	23.78	0.148	23.32
18	U2152	**0.39**	357.23	10.43	144.56	0.838	25.86	0.754	25.70
19	U2319	**0.028**	391.08	1.72	191.37	0.838	34.14	0.631	34.10
20	Pr2392	**0.026**	401.28	7.04	161.54	0.612	29.68	0.723	29.70
21	Pcb3038	**0.002**	457.29	7.88	195.26	0.624	43.72	0.597	45.68
22	Fl3795	**0.37**	461.81	16.13	389.22	1.52	67.33	1.572	75.43
23	Fnl4461	**0.35**	498.01	5.62	330.24	0.961	44.10	0.784	49.39
24	Rl5915	**0.012**	557.87	12.94	465.58	1.005	63.09	0.929	80.14
25	Rl5934	**0.0127**	561.21	13.02	478.35	1.041	68.71	0.974	85.21
26	Pla7397	**0.0086**	780.31	10.19	682.31	1.441	113.59	1.268	126.59
27	Rl11849	**0.098**	890.05	11.49	1234.15	1.062	299.36	0.943	438.39
28	Usa13509	**0.087**	897.09	7.62	1579.27	1.168	318.28	0.939	422.18
29	Brd14051	**0**	902.35	6.18	1459.43	1.051	347.70	0.902	459.06
30	D15112	**0.005**	928.34	5.95	1802.36	0.984	522.12	0.813	715.72
31	D18512	**0.25**	995.17	6	2084.02	1.049	569.24	0.886	769.66
32	Pla33810	**0.18**	1095.45	13.23	4788.57	1.726	1385.08	1.539	1903.00
33	Pla85900	**0.21**	1198.21	10.94	25034.37	1.378	5279.07	1.456	3827.17
	Average	**0.076**	432.59	8.826	1294.7	0.739	291.02	0.673	286.88

such as the following: (1) The time complexity to produce a new solution is quite different for different operators; for example, the construction method used in the ant-based system is far more complex than the *inverse*, *swap*, and *blockInsert* operators used by the ELBSA algorithm. (2) Different implementations may have quite different time complexities for same operator; using the *blockInsert* operator as an example, it has an $O(1)$ time complexity if the

TABLE 6. Comparison of ELBSA with 24 algorithms published in recent years.

No.	Algorithm	Distance Type	N	Dims	PEav	PEav1	$R+$	$R-$	p-value
1	MSA [3]	Float	17	[1002, 85900]	1.59	**0.93**	128	8	**1.18e-03**
2	SSA [5]	Integer	26	[101, 33810]	0.55	**0.27**	282.5	42.5	**1.24e-03**
3	MAS [8]	Integer	25	[1000, 15112]	1.12	**0.54**	208	92	5.11e-02
4	TSHACO [35]	Integer	26	[101, 33810]	2.63	**0.27**	325	0	**8.30e-06**
5	ACE [11]	Integer	15	[101, 1400]	0.77	**0.24**	92	13	**7.60e-03**
6	TSACO [12]	Integer	9	[105, 2392]	1.05	**0.24**	36	0	**7.69e-03**
7	ESACO [7]	Integer	11	[1002, 18512]	**0.70**	0.87	19	36	2.13e-01
8	HMMA [10]	Float	12	[105, 1400]	7.7	**0.29**	66	0	**2.22e-03**
9	PCGA [15]	Integer	14	[225, 18512]	4.33	**0.52**	91	0	**9.82e-04**
10	HGA [34]	Integer	15	[101, 1655]	2.22	**0.16**	105	0	**6.55e-04**
11	S-PSOg [17]	Integer	9	[198, 1577]	2.73	**0.25**	36	0	**7.69e-03**
12	S-PSOi [17]	Integer	9	[198, 1577]	**0.12**	0.25	8	28	5.81e-02
13	BCO [22]	Integer	16	[574, 1379]	1.40	**0.29**	119	1	**5.31e-04**
14	HDABC [20]	Float	17	[1002, 85900]	1.27	**0.93**	125	11	**1.93e-03**
15	IDCS [18]	Integer	21	[107, 1379]	1.03	**0.16**	192.5	17.5	**1.09e-03**
16	DCS [19]	Float	30	[127, 3795]	1.32	**0.24**	423	12	**5.75e-06**
17	DBA [25]	Integer	21	[107, 1379]	0.72	**0.16**	196	14	**6.81e-04**
18	IDBA [24]	Integer	10	[101, 1002]	2.93	**0.09**	45	0	**5.06e-03**
19	EHS [31]	Integer	17	[1002, 85900]	1.73	**0.77**	135	1	**3.52e-04**
20	IA-EDA [26]	Integer	14	[101, 100000]	1.84	**0.27**	91	0	**1.47e-03**
21	DIWO [27]	Float	13	[107, 2392]	1.45	**0.10**	78	0	**1.47e-03**
22	ABO [28]	Integer	15	[101, 1655]	1.34	**0.16**	104	1	**8.05e-04**
23	DSOS [29]	Integer	13	[101, 2392]	3.84	**0.16**	78	0	**1.47e-03**
24	DMRSA [37]	Integer	15	[101, 1655]	0.98	**0.16**	101	4	**2.33e-03**

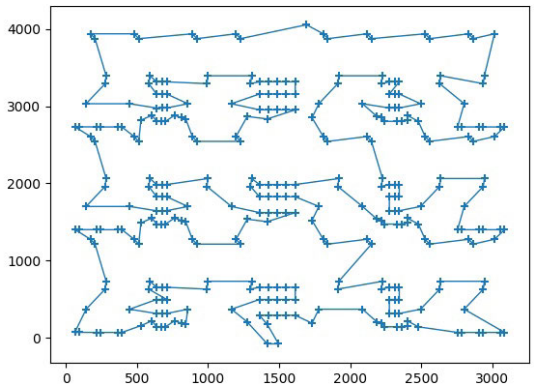

FIGURE 9. The tour of the best solution for the Lin318 instance.

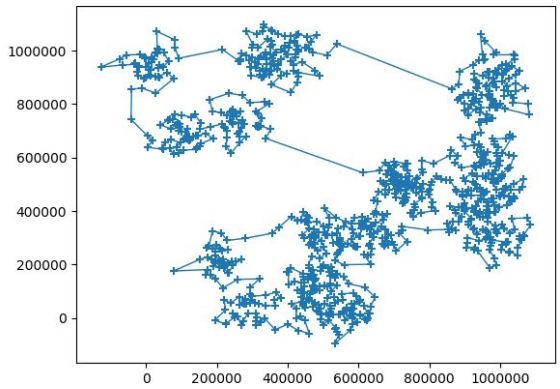

FIGURE 10. The tour of the best solution for the Dsj1000 instance.

linked list is used to represent the solution, but it has an $O(n)$ time complexity if a simple array is used to represent the solution. (3) For some algorithms, the solution evaluation times are not deterministic. (4) The software or (and) hardware platforms are different. To facilitate the comparative study of metaheuristics for the TSP, the source code implementing the ELBSA algorithm is made publicly available in GitHub (https://github.com/yiwzhong/ELBSA4TSP). To help researchers observe the results, we also provide a python script to draw the tour of a solution. Using the script, we depict the tours of the best results obtained by the ELBSA algorithm for the Lin318 and Dsj1000 instances as shown in Fig.9 and Fig.10. For the Lin318 instance, the tour length of the solution depicted in Fig.9 is also the optimal tour length. For the Dsj1000 instance, the percentage error of the tour length for the solution depicted in Fig.10 is 0.124.

VI. CONCLUSION

Aiming to improve the sampling efficiency of SA algorithm for the TSP, this paper presents an enhanced list-based SA algorithm that can sample more neighbors at more suitable temperatures and from more promising neighborhoods. Specifically, heuristic augmented sampling, systematic selection, and variable Markov chain length are adopted to improve the sampling efficiency. Extensive experiments have verified the effectiveness of the proposed strategies. The proposed ELBSA algorithm was compared with 32 state-of-the-art metaheuristics on a wide range of large-scale TSP instances. The simulation results show that except for the three hybrid algorithms, i.e., ESACO [7], S-PSOi [17], and HBMO [21], which are intensively augmented by local search methods, the ELBSA algorithm achieves better performance than the other algorithms. The three strategies used by the ELBSA algorithm may be used to enhance SA algorithm for other large-scale optimization problems. One shortage of the ELBSA algorithm is that its parameters are not only relevant, but are also instance-dependent. An interesting research

direction is to study how to use adaptive parameter control to improve the performance of the ELBSA algorithm. The ELBSA algorithm uses current population as an approximate representation of knowledge obtained in the search process, and uses the knowledge to guide its sampling. In the field of metaheuristics, more advanced knowledge representations exist, such as external archive and probabilistic models. A potential research direction is to explore whether these advanced knowledge representations can further improve the efficiency of SA's sampling.

REFERENCES

[1] V. Černý, "Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm," *J. Optim. Theory Appl.*, vol. 45, no. 1, pp. 41–51, 1985.
[2] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.
[3] C. Wang, M. Lin, Y. Zhong, and H. Zhang, "Solving travelling salesman problem using multiagent simulated annealing algorithm with instance-based sampling," *Int. J. Comput. Sci. Math.*, vol. 6, no. 4, pp. 336–353, Sep. 2015.
[4] S.-H. Zhan, J. Lin, Z.-J. Zhang, and Y.-W. Zhong, "List-based simulated annealing algorithm for traveling salesman problem," *Comput. Intell. Neurosci.*, vol. 2016, Feb. 2016, Art. no. 1712630.
[5] C. Wang, M. Lin, Y. Zhong, and H. Zhang, "Swarm simulated annealing algorithm with knowledge-based sampling for travelling salesman problem," *Int. J. Intell. Syst. Technol. Appl.*, vol. 15, no. 1, pp. 74–94, 2016.
[6] A. Misevičius, "A modified simulated annealing algorithm for the quadratic assignment problem," *Informatica*, vol. 14, no. 4, pp. 497–514, 2003.
[7] H. Ismkhan, "Effective heuristics for ant colony optimization to handle large-scale problems," *Swarm Evol. Comput.*, vol. 32, pp. 140–149, Feb. 2017.
[8] Y. Yan, H.-S. Sohn, and G. Reyes, "A modified ant system to achieve better balance between intensification and diversification for the traveling salesman problem," *Appl. Soft Comput.*, vol. 60, pp. 256–267, Nov. 2017.
[9] A. M. Mohsen, "Annealing ant colony optimization with mutation operator for solving TSP," *Comput. Intell. Neurosci.*, vol. 2016, Oct. 2016, Art. no. 8932896.
[10] W. Yong, "Hybrid Max—Min ant system with four vertices and three lines inequality for traveling salesman problem," *Soft Comput.*, vol. 19, no. 3, pp. 585–596, Mar. 2015.
[11] J. B. Escario, J. F. Jimenez, and J. M. Giron-Sierra, "Ant colony extended: Experiments on the travelling salesman problem," *Expert Syst. with Appl.*, vol. 42, no. 1, pp. 390–410, Jan. 2015.
[12] Z. Zhang and Z. Feng, "Two-stage updating pheromone for invariant ant colony optimization algorithm," *Expert Syst. Appl.*, vol. 39, no. 1, pp. 706–712, Jan. 2012.
[13] J. Ning, Q. Zhang, C. Zhang, and B. Zhang, "A best-path-updating information-guided ant colony optimization algorithm," *Inf. Sci.*, vols. 433–434, pp. 142–162, Apr. 2018.
[14] J. Wang, O. Ersoy, M. He, and F. Wang, "Multi-offspring genetic algorithm and its application to the traveling salesman problem," *Appl. Soft Comput.*, vol. 43, pp. 415–423, Jun. 2016.
[15] P. V. Paul, N. Moganarangan, S. S. Kumar, R. Raju, T. Vengattaraman, and P. Dhavachelvan, "Performance analyses over population seeding techniques of the permutation-coded genetic algorithm: An empirical study based on traveling salesman problems," *Appl. Soft Comput.*, vol. 32, pp. 383–402, Jul. 2015.
[16] Y. Zhong, J. Lin, L. Wang, and H. Zhang, "Discrete comprehensive learning particle swarm optimization algorithm with metropolis acceptance criterion for traveling salesman problem," *Swarm Evol. Comput.*, vol. 42, pp. 77–88, Oct. 2018.
[17] W.-N. Chen, J. Zhang, H. S. H. Chung, W.-L. Zhong, W.-G. Wu, and Y.-H. Shi, "A novel set-based particle swarm optimization method for discrete optimization problems," *IEEE Trans. Evol. Comput.*, vol. 14, no. 2, pp. 278–300, Apr. 2010.
[18] A. Ouaarab, B. Ahiod, and X.-S. Yang, "Discrete cuckoo search algorithm for the travelling salesman problem," *Neural Comput. Appl.*, vol. 24, nos. 7–8, pp. 1659–1669, 2014.
[19] Y. Zhou, X. Ouyang, and J. Xie, "A discrete cuckoo search algorithm for travelling salesman problem," *Int. J. Collaborative Intell.*, vol. 1, no. 1, pp. 68–84, 2014.
[20] Y. Zhong, J. Lin, L. Wang, and H. Zhang, "Hybrid discrete artificial bee colony algorithm with threshold acceptance criterion for traveling salesman problem," *Inf. Sci.*, vol. 421, pp. 70–84, Dec. 2017.
[21] Y. Marinakis, M. Marinaki, and G. Dounias, "Honey bees mating optimization algorithm for the Euclidean traveling salesman problem," *Inf. Sci.*, vol. 181, no. 20, pp. 4684–4698, Oct. 2011.
[22] L.-P. Wong, M. Y. H. Low, and C. S. Chong, "Bee colony optimization with local search for traveling salesman problem," *Int. J. Artif. Intell. Tools*, vol. 19, no. 3, pp. 305–334, 2010.
[23] Y. Zhong, L. Wang, M. Lin, and H. Zhang, "Discrete pigeon-inspired optimization algorithm with metropolis acceptance criterion for large-scale traveling salesman problem," *Swarm Evol. Comput.*, vol. 48, pp. 134–144, Aug. 2019.
[24] E. Osaba, X.-S. Yang, F. Diaz, P. Lopez-Garcia, and R. Carballedo, "An improved discrete bat algorithm for symmetric and asymmetric traveling Salesman problems," *Eng. Appl. Artif. Intel.*, vol. 48, pp. 59–71, Feb. 2016.
[25] Y. Saji and M. E. Riffi, "A novel discrete bat algorithm for solving the travelling salesman problem," *Neural Comput. Appl.*, vol. 27, no. 7, pp. 1853–1866, 2016.
[26] Z. Xu, Y. Wang, S. Li, Y. Liu, Y. Todo, and S. Gao, "Immune algorithm combined with estimation of distribution for traveling salesman problem," *IEEJ Trans. Elect. Electron. Eng.*, vol. 11, no. S1, pp. S142–S154, 2016.
[27] Y. Zhou, Q. Luo, H. Chen, A. He, and J. Wu, "A discrete invasive weed optimization algorithm for solving traveling salesman problem," *Neurocomputing*, vol. 151, no. 3, pp. 1227–1236, Mar. 2015.
[28] J. B. Odili and M. N. M. Kahar, "Solving the traveling salesman's problem using the African Buffalo optimization," *Comput. Intell. Neurosci.*, vol. 2016, Aug. 2015, Art. no. 1510256.
[29] A. E.-S. Ezugwu and A. O. Adewumi, "Discrete symbiotic organisms search algorithm for travelling salesman problem," *Expert Syst. Appl.*, vol. 87, pp. 70–78, Nov. 2017.
[30] A. E.-S. Ezugwu, A. O. Adewumi, and M. E. Frîncu, "Simulated annealing based symbiotic organisms search optimization algorithm for traveling salesman problem," *Expert Syst. Appl.*, vol. 77, pp. 189–210, Jul. 2017.
[31] C. Wang, J. Lin, M. Lin, and Y. Zhong, "Evolutionary harmony search algorithm with metropolis acceptance criterion for travelling salesman problem," *Int. J. Wireless Mobile Comput.*, vol. 10, no. 2, pp. 166–173, 2016.
[32] L. T. Kóczy, P. Földesi, and B. Tüu-Szabó, "Enhanced discrete bacterial memetic evolutionary algorithm—An efficacious metaheuristic for the traveling salesman optimization," *Inf. Sci.*, vols. 460–461, pp. 389–400, Sep. 2018.
[33] H. Wang, N. Zhang, and J.-C. Créput, "A massively parallel neural network approach to large-scale Euclidean traveling salesman problems," *Neurocomputing*, vol. 240, pp. 137–151, May 2017.
[34] S.-M. Chen and C.-Y. Chien, "Solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques," *Expert Syst. Appl.*, vol. 38, no. 12, pp. 14439–14450, 2011.
[35] W. Deng, R. Chen, B. He, Y. Q. Liu, L. F. Yin, and J. H. Guo, "A novel two-stage hybrid swarm intelligence optimization algorithm and application," *Soft Comput.*, vol. 16, no. 10, pp. 1707–1722, 2012.
[36] Y. Lin, Z. Bian, and X. Liu, "Developing a dynamic neighborhood structure for an adaptive hybrid simulated annealing tabu search algorithm to solve the symmetrical traveling salesman problem," *Appl. Soft Comput.*, vol. 49, pp. 937–952, Dec. 2016.
[37] H. G. Zhang and J. Zhou, "Dynamic multiscale region search algorithm using vitality selection for traveling salesman problem," *Expert Syst. Appl.*, vol. 60, pp. 81–95, Oct. 2016.
[38] X. Geng, Z. Chen, W. Yang, D. Shi, and K. Zhao, "Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search," *Appl. Soft Comput.*, vol. 11, no. 4, pp. 3680–3689, 2011.
[39] E. H. L. Aarts, J. H. M. Korst, and P. J. M. van Laarhoven, "A quantitative analysis of the simulated annealing algorithm: A case study for the traveling salesman problem," *J. Stat. Phys.*, vol. 50, nos. 1–2, pp. 187–206, 1988.
[40] J. R. A. Allwright and D. B. Carpenter, "A distributed implementation of simulated annealing for the travelling salesman problem," *Parallel Comput.*, vol. 10, no. 3, pp. 335–338, May 1989.

[41] C.-S. Jeong and M.-H. Kim, "Fast parallel simulated annealing for traveling salesman problem on SIMD machines with linear interconnections," *Parallel Comput.*, vol. 17, nos. 2–3, pp. 221–228, Jun. 1991.
[42] M. Hasegawa, "Verification and rectification of the physical analogy of simulated annealing for the solution of the traveling salesman problem," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 83, no. 3, 2011, Art. no. 036708.
[43] P. Tian and Z. Yang, "An improved simulated annealing algorithm with genetic characteristics and the traveling salesman problem," *J. Inf. Optim. Sci.*, vol. 14, no. 3, pp. 241–255, 1993.
[44] Z. Wang, X. Geng, and Z. Shao, "An effective simulated annealing algorithm for solving the traveling salesman problem," *J. Comput. Theor. Nanosci.*, vol. 6, no. 7, pp. 1680–1686, 2009.
[45] A. Bölte and U. W. Thonemann, "Optimizing simulated annealing schedules with genetic programming," *Eur. J. Oper. Res.*, vol. 92, no. 2, pp. 402–416, 1996.
[46] K. A. Dowsland and J. M. Thompson, "Simulated annealing," in *Handbook of Natural Computing*, vol. 43, G. Rozenberg, T. Bäck, and J. N. Kok, Eds. Berlin, Germany: Springer, 2012, no. 1, pp. 1623–1655.
[47] M. Cheraitia and S. Haddadi, "Simulated annealing for the uncapacitated exam scheduling problem," *Int. J. Metaheuristics*, vol. 5, no. 2, pp. 156–170, 2016.
[48] G. Reinelt, "TSPLIB-A traveling salesman problem library," *ORSA J. Comput.*, vol. 3, no. 4, pp. 376–384, 1991.
[49] S. García, D. Molina, M. Lozano, and F. Herrera, "A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: A case study on the CEC'2005 special session on real parameter optimization," *J. Heuristics*, vol. 15, no. 6, pp. 617–644, 2009.

LIJIN WANG received the Ph.D. degree from Beijing Forestry University, Beijing, China, in 2008. He is currently a Professor with the College of Computer and Information Science, Fujian Agriculture and Forestry University, Fuzhou, China. His current research interests include nature-inspired algorithm and intelligent information processing.

RONGYING CAI is currently a Senior Experimentalist with the College of Computer and Information Science, Fujian Agriculture and Forestry University. Her current research interests include intelligent computing and combinatorial optimization.

MIN LIN is currently an Associate Professor with the College of Computer and Information Science, Fujian Agriculture and Forestry University. His current research interests include intelligent computing, bioinformatics, and parallel computing.

YIWEN ZHONG received the M.S. and Ph.D. degrees in computer science and technology from Zhejiang University, Hangzhou, China, in 2002 and 2005, respectively. He is currently a Professor with the College of Computer and Information Science, Fujian Agriculture and Forestry University, Fuzhou, China. His current research interests include computational intelligence, data visualization, and bioinformatics.