Swarm and Evolutionary Computation 48 (2019) 134–144

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

journal homepage: www.elsevier.com/locate/swevo

Discrete pigeon-inspired optimization algorithm with Metropolis acceptance criterion for large-scale traveling salesman problem

Yiwen Zhong [a,b,*], Lijin Wang [a,b], Min Lin [a,b], Hui Zhang [c]

[a] College of Computer and Information Science, Fujian Agriculture and Forestry University, Fuzhou, Fujian Province 350002, PR China
[b] Key Laboratory of Smart Agriculture and Forestry (Fujian Agriculture and Forestry University), Fujian Province University, Fuzhou, Fujian Province 350002, PR China
[c] J.B. Speed School of Engineering, University of Louisville, Louisville, KY 40292, USA

ARTICLE INFO

Keywords:
Pigeon-inspired optimization
Traveling salesman problem
Comprehensive learning
Cooperative learning
Heuristic information
Metropolis acceptance criterion

ABSTRACT

Pigeon-inspired optimization (PIO) algorithm, which is a newly proposed swarm intelligence algorithm, has been mainly applied to continuous optimization problems. In this paper, a discrete PIO (DPIO) algorithm, which uses the Metropolis acceptance criterion of simulated annealing algorithm, is proposed for Traveling Salesman Problems (TSPs). A new map and compass operator with comprehensive learning ability is designed to enhance DPIO's exploration ability. A new landmark operator, which has cooperative learning ability and can learn from the heuristic information of TSP instance, is designed to improve DPIO's exploitation ability. Aim to enhance its ability to escape from premature convergence, the Metropolis acceptance criterion is used to decide whether to accept newly produced solutions. Systematic experiments were performed to analyze the behaviours of the map and compass operator and the landmark operator. The performance of DPIO algorithm was tested on 33 large-scale TSP instances from TSPLIB with city number from 1000 to 85900. Simulation results show that the proposed algorithm is effective and is competitive with most other state-of-the-art meta-heuristic algorithms.

1. Introduction

Swarm intelligence (SI) is the collective behaviour of a population of simple agents interacting locally with one another and with their environment. Well-known examples in natural systems of SI include bird flocks, fish schools, ant colonies, and animal herding etc. Pigeon-inspired optimization (PIO) algorithm, which was first proposed by Duan and Qiao [1], is a novel SI algorithm. In PIO algorithm, each pigeon of the swarm has a position, a velocity, and a personal best historical position, according to which it moves in the search space. The search process of pigeon consists of two stages. In the first stage, pigeon searches in solution space being guided by the best position found by the swarm and the flying experience of itself. In the second stage, pigeon is guided by the center of the positions of remaining successful pigeons. To date, PIO algorithm and its variants are mainly used for continuous problems, as we know, few paper has studied PIO algorithm for classical combinatorial optimization problems, such as traveling salesman problems (TSPs), quadratic assignment problems, knapsack problems, and job-shop scheduling problems, etc.

TSP is one of the classical NP-hard problems in combinatorial optimization. The objective of TSP is to find a shortest route that visits each city once and returns to the origin city. As an NP-hard problem, the computational complexity of exact algorithms for TSP is exponential with the number of cities. As for most NP-hard problems, it may be enough to find workable solutions with limited computation resource. As a result, a lot of interests have been focused on using efficient heuristics and meta-heuristics to solve the TSP. In recent years, many meta-heuristics have been proposed to solve large-scale TSP instances with at least 1000 cities. Among all those meta-heuristics, SI algorithms are most intensively studied. Those SI algorithms include ant-inspired algorithms [2–7], bird-inspired algorithms [8–11], bat-inspired algorithms [12,13], bee-inspired algorithms [14–16], African buffalo-inspired algorithm [17], symbiotic organisms-inspired algorithms [18,19], and hybrid SI algorithms [20,21], etc. Other meta-heuristics include genetic algorithms [22–24], evolutionary algorithm [25], immune algorithm [26], invasive weed optimization [27], artificial neural network [28], and those algorithms based on local search [29–36], etc.

* Corresponding author. College of Computer and Information Science, Fujian Agriculture and Forestry University, Fuzhou, Fujian Province 350002, PR China.
E-mail address: yiwzhong@fafu.edu.cn (Y. Zhong).

https://doi.org/10.1016/j.swevo.2019.04.002
Received 16 August 2018; Received in revised form 22 February 2019; Accepted 3 April 2019
Available online 9 April 2019

The learning strategy of PIO algorithm is similar to that of particle swarm optimization (PSO) algorithm. Compared with PSO algorithm, PIO algorithm can use different learning strategy in different stage and use different learning strategy for different group of pigeons. This flexibility may enhance PIO's ability to balance exploration and exploitation. Many studies have shown that PSO algorithm has good performance on solving discrete and combinatorial optimization problems. To study how to adjust PIO algorithm for discrete and combinatorial optimization problems, this paper presents a discrete PIO (DPIO) algorithm for TSPs. In DPIO, besides *minus* and *plus* operators are specifically redesigned for TSP, a new *selection* operator is designed to select a 'center' for the second stage. Using those basic operators, map and compass operator and landmark operator are redesigned for pigeon to produce new position. In map and compass operator, comprehensive learning, which is first proposed by Liang et al. [37] for PSO algorithm, is used to replace the global best based learning to improve the explorative ability. In landmark operator, cooperative learning and heuristic information are combined to improve the exploitative ability. To further balance intensification and diversification, DPIO uses the Metropolis acceptance criterion of simulated annealing (SA) algorithm to decide whether to accept the new solutions. Systematic experiments were performed to analyze the behaviours of DPIO algorithm. The performance of DPIO algorithm was compared with other state-of-the-art meta-heuristics on a wide range of large-scale benchmark TSP instances. The simulation results confirm the effectiveness and competitiveness of the proposed algorithm.

The rest of this paper is organized as follows. Section 2 provides a short description of basic PIO algorithm, TSP, and SI algorithms for large-scale TSP. Section 3 presents our proposed DPIO Algorithm. Section 4 analyzes the behaviors of DPIO Algorithm. Section 5 compares the performance of DPIO algorithm with other state-of-the-art meta-heuristics published in recent two years. Finally, in section 6 we summarize our study.

2. Related work

2.1. Pigeon-inspired optimization algorithm

PIO algorithm, which mimics some of the homing characteristics of pigeons, is a newly proposed SI algorithm. Specifically, two operators are designed by using some idealized rules of the homing characteristics of pigeons: (1) map and compass operator which mimics the homing behaviour guided by sun and magnetic particles; (2) landmark operator which mimics the homing behaviour guided by landmark of familiar region. Magnetoreception is a sense which allows an organism to detect a magnetic field to perceive direction, altitude or location. Pigeons can sense the earth field by using magnetoreception to shape the map in their brains. They regard the altitude of the sun as compass to adjust the direction. As they are in unfamiliar areas which are far from their home, they rely mainly on magnetoreception and sun. Map and compass operator is designed to mimic this homing behaviour. Suppose that the search space is N-dimensional, then the i-th pigeon of the swarm can be represented by a N-dimensional vector, $X_i = (x_{i,1}, x_{i,2}, \dots, x_{i,N})$. The velocity of this pigeon, which represents the position change of this pigeon, can be represented by another N-dimensional vector $V_i = (v_{i,1}, v_{i,2}, \dots, v_{i,N})$. The best previously visited position of the i-th pigeon is denoted as $P_i = (p_{i,1}, p_{i,2}, \dots, p_{i,N})$. The global best position of the swarm is $\mathbf{g} = (g_1, g_2, \dots, g_N)$, and let the superscripts denote the iteration number, then each pigeon is flying according to the following two equations:

$$v_{i,j}^t = v_{i,j}^{t-1} e^{-Rt} + r(g_j^{t-1} - x_{i,j}^{t-1}) \tag{1}$$

$$x_{i,j}^t = x_{i,j}^{t-1} + v_{i,j}^t \tag{2}$$

where $j = 1, 2, \dots, N$; $i = 1, 2, \dots, M$, and M is the size of the swarm; R is the map and compass factor, which is used to control the impact of the latest velocity on the current velocity; r is random number, uniformly distributed in [0, 1); and $t = 1, 2, \dots$, determines the iteration number. Eq. (1) is used to calculate the pigeon's new velocity according to its latest velocity and the distance of its current position from the global best experience. Then the pigeon flies toward a new position according to (2).

When pigeon is in familiar areas which are near their home they will fly mainly according the landmark. Landmark operator is designed to mimic this homing behaviour. In the landmark operator, the number of pigeons is decreased by half in each generation. The better part will fly to destination guided by the center, and the worse part will follow the better part. Let $\boldsymbol{c}^t$ be the center of pigeons in better part at the t-th iteration. The position updating rule for pigeon i at the t-th iteration can be given by:

$$c_j^t = \frac{\sum_{i=1}^{M} (x_{i,j}^{t-1} f(x_i^{t-1}))}{M \sum_{i=1}^{M} (f(x_i^{t-1}))} \tag{3}$$

$$x_{i,j}^t = x_{i,j}^{t-1} + r(c_j^t - x_{i,j}^{t-1}) \tag{4}$$

where $j = 1, 2, \dots, N$; $i = 1, 2, \dots, M$, and M is the size of the better part; r is random number, uniformly distributed in [0, 1); and $t = 1, 2, \dots$, determines the iteration number. Eq. (3) is used to calculate the center of pigeons. Then the pigeon flies toward a new position according to (4).

Two features of PIO algorithm, the dividing of its search process and the dividing of swarm in the second stage, greatly contribute to its success. Because the search process is divided into first stage and second stage, PIO algorithm can use different search operator in different stage, which may balance the explorative ability and the exploitative ability more easily. The dividing of swarm, which reduces the remaining swarm size by half, divides the swarm into successful group and unsuccessful group, then PIO algorithm can design different search strategy for different group. Dividing of swarm is a kind of division of labor which is considered as a necessary property of SI [38]. Due to its simplicity and flexibility, PIO algorithm has been successfully applied in many fields, such as active disturbance rejection control for small unmanned helicopters [39], linear-quadratic regulator controller design for quadrotor [40], fuzzy energy management strategy for parallel hybrid electric vehicle [41], detecting protein complexes from dynamic protein-protein interaction networks by density based clustering [42], three-dimensional path planning for uninhabited combat aerial vehicle [43], node self-deployment for underwater wireless sensor networks [44], control parameters optimization in automatic carrier landing system [45], and robust attitude control for reusable launch vehicles [46] etc.

2.2. Travelings salesman problem

TSP is one of the most famous NP-hard combinatorial optimization problems. No known exact algorithm with polynomial time complexity can guarantee it to find a global optimal route. Consider a TSP instance with n cities, we can model TSP as follows. Suppose a distance matrix $D = (d_{i,j})_{n*n}$ is used to store distances between all the pair of cities, where each element $d_{i,j}$ of matrix D represents the distance from city i to city j. We can use $\boldsymbol{x}$, a permutation of cities which indicates the visiting sequence of cities, to represent a solution. The goal of TSP is to find a solution $\boldsymbol{x}$ that minimizes

$$f(\boldsymbol{x}) = \sum_{i=1}^{n-1} d_{x_i, x_{i+1}} + d_{x_n, x_1} \tag{5}$$

2.3. Swarm intelligence algorithms for large-scale TSP

In recent years, many SI algorithms have been proposed for large-scale TSP. Those SI algorithms may be construction-based,

perturbation-based, or combination of them. Most of the construction-based algorithms are ant-inspired. Ismkhan [2] proposed an ant colony optimization algorithm (ACO) with three effective heuristics (ESACO) to handle large-scale TSP instances. The three effective heuristics used in ESACO include 2-opt local search with candidates-set updated according to pheromone information, pheromone representation using sparse matrix with linear space complexity, quick next move selection with the help of pheromone sparse matrix. Yan et al. [3] proposed a modified ant system (MAS) where adaptive parameter control strategies are used both by tour construction and pheromone updating. Using those adaptive parameter control strategies, MAS can obtain better balance between exploitation and exploration. Yong [4] proposed a hybrid Max-Min ant (HMMA) system which is integrated with a four vertices and three lines inequality. In each iteration of HMMA, if the tour constructed by an ant is not better than the previous tour, the four vertices and three lines inequality is used to enhance the new tour. Escario et al. [5] proposed an ant colony extended (ACE) algorithm with role division of ants. In ACE, ants are divided into patrollers and foragers and the number of each kind of ant is adaptively changing. A patroller uses pheromone or heuristic information to locate sources of food before the foraging activity begins. A forager uses pheromone only to exploit the food sources discovered by patrollers. Zhang and feng [6] proposed an ACO algorithm with two-stage pheromone update rule. In the first stage, the top r iterative optimal solutions are employed to update pheromone, and in another stage, only the iteration-best solution or the global-best solution is used to update pheromone. Ning et al. [7] proposed a best-path-updating information-guided ACO algorithm where strengthened pheromone update mechanism and a novel pheromone smoothing mechanism are designed to improve its performance. The only construction-based SI algorithm, which is not ant-inspired, is set-based PSO (S-PSO) [8]. In S-PSO, solution and velocity are defined as crisp set and set with possibilities respectively. S-PSO uses edges in velocity and original position to construct new position, and this is quite different from the traditional PSO algorithm where perturbation-based updating is used to produce new solution.

Perturbation-based algorithms may be inspired by many biological systems, such as bird flocking, social insects, animal herding, etc. Zhong et al. [9] proposed a discrete comprehensive learning PSO (D-CLPSO) algorithm where Metropolis acceptance criterion is used to decide whether to accept newly produced solution. Ouaarab et al. [10] proposed an improved discrete cuckoo search algorithm which uses 2-opt move and double-bridge move to produce new solution. Zhou et al. [11] proposed a novel discrete cuckoo search algorithm which uses learning operator, 'A' operator and 3-opt to enhance the optimal tour in each iteration. Marinakis et al. [14] proposed a honey bees mating optimization (HBMO) which combines the multiple phase neighborhood search (MPNS), greedy randomized adaptive search procedure (GRASP), and the expanding neighborhood search (ENS) strategy. In HBMO, MPNS-GRASP is used to create initial population, and ENS is used to improve the broods produced by the mating flight of the queen. Wong et al. [15] proposed a bee colony optimization (BCO) which is integrated with a fixed-radius near neighbor 2-opt heuristic. In BCO, a frequency-based pruning strategy is proposed to limit that only a subset of the promising solutions to undergo 2-opt heuristic. Zhong et al. [16] proposed a hybrid discrete artificial bee colony (HDABC) algorithm where threshold acceptance criterion is used to decide whether the produced new solution is accepted. Osaba et al. [12] proposed an improved discrete bat algorithm (IDBA) which uses hamming distance between bats to represent velocity. In IDBA, 2-opt and 3-opt operators are used to produce a number of neighbors, and the best one is selected as current solution. Saji et al. [13] proposed a novel discrete bat algorithm (DBA) where solution and velocity are represented by vector of cities and permutation of cities respectively. In DBA, 2-exchange crossover heuristic is used to produce new solutions. Odili et al. [17] proposed African buffalo algorithm where buffalo searches for best route guided by the personal best location and global best location. Ezugwu and Adewumi [18] proposed a discrete symbiotic organisms search (SOS) algorithm where *swap*, *inverse*, and *insert* operators are used to convert the continuous SOS into discrete form. Ezugwu et al. [19] proposed a SA-based SOS (SOS-SA) algorithm where Metropolis acceptance criterion of SA is used to decide whether to accept newly produced solution. This probabilistic acceptance strategy can assist the SOS in avoiding being trapped into local minimum and also increase its level of diversity.

Some hybrid SI algorithms combine both construction-based and perturbation-based strategies. Chen et al. [20] presented a hybrid ant colony system which uses the ideas of PSO, SA, and genetic algorithm (GA) to enhance its performance. Deng et al. [21] presented a hybrid ACO algorithm where GA and PSO are used to produce good initial allocation of pheromone for ACO.

Most of the SI algorithms are first proposed for continuous optimization problems. Several strategies, such as using map function to bridge continuous space and discrete space and redesigning basic operators for discrete space etc., have been used to extend those algorithms for combinatorial optimization problems. Among those strategies, redesigning basic operators for discrete space has been successfully used in S-PSO [8], D-CLPSO [9] and HDABC [16] for TSP. S-PSO uses edges in velocity and original position to construct new position. In general, the intensification ability of construction-based meta-heuristics is not as good as perturbation-based meta-heuristics. D-CLPSO, which always uses comprehensive-learning, may have not enough diversity in later stage. HDABC, which always uses randomly selected bee as exemplar, may have not enough intensification ability in the early stage. Those shortcomings impel us to strive for new algorithm with better balance between exploration and exploitation. To this aim, we redesign the basic operators of PIO algorithm to extend it for solving the TSP.

3. Discrete pigeon-inspired optimization algorithm

To apply (1)–(4) of PIO algorithm on TSP, we must redefine the position, velocity and all necessary operators for TSP. We use a vector $\boldsymbol{x}$, which is a linked list storing the set of edges in tour, to represent position. Each element x_j in $\boldsymbol{x}$ represents an edge e_{j,x_j} from city j to city x_j. In order to guarantee that each position is a valid solution, $\boldsymbol{x}$ must be a permutation of cities and $x_j \neq j$ for each $j \in \{1, 2, 3, \ldots, n\}$. Using a TSP instance with five cities as an example, suppose $\boldsymbol{x} = (3, 5, 2, 1, 4)$, then $\boldsymbol{x}$ represents a solution with five edges $(e_{1,3}, e_{2,5}, e_{3,2}, e_{4,1}, e_{5,4})$. So the route of solution $\boldsymbol{x}$ is $1 \rightarrow 3 \rightarrow 2 \rightarrow 5 \rightarrow 4 \rightarrow 1$. Similarly, we use a vector $\boldsymbol{v} = (v_1, v_2, \ldots, v_n)$ to represent velocity, where each element v_j represents an edge e_{j,v_j} from city j to city v_j. For a velocity vector $\boldsymbol{v}$, the only constrain is $v_j \neq j$ for each $j \in \{1, 2, 3, \ldots, n\}$. This constrain can guarantee that each element of $\boldsymbol{v}$ is a valid edge.

3.1. Basic operators of DPIO algorithm

Given two elements $x_{1,j}$ and $x_{2,j}$ from two position vector X_1 and X_2, the *minus* operator $\ominus$, which is used to produce velocity v_j in j-th dimension, is defined as (6):

$$v_j = x_{2,j} \ominus x_{1,j} = \begin{cases} x_{2,j}, & x_{2,j} \neq x_{1,j} \\ e \in nc(j), & x_{2,j} = x_{1,j} \end{cases} \tag{6}$$

where $nc(j)$ is next visiting city list of city j and e is a city randomly selected from all cities in $nc(j)$. Using (6), the result of *minus* operator between two position X_2 and X_1 is a velocity V where each element v_j is equal to $x_{2,j} \ominus x_{1,j}$. For j-th velocity component v_j, if $x_{2,j}$ is not equal to $x_{1,j}$, then $v_j = x_{2,j}$, otherwise, velocity component is randomly selected from $nc(j)$. There may have three different ways to create $nc(j)$ for a city j: (1) *Empty*, which means that $nc(j)$ is empty; (2) *Full*, which means that $nc(j)$ includes all other cities; and (3) *Heuristic*, which means that $nc(j)$ is a nearest city list which is made of k nearest cities, where

k is a parameter representing the length of $nc(j)$. According to the stage pigeon is in, pigeon can select a most suitable way to create $nc(j)$.

In the second stage of basic PIO algorithm, pigeons use (3) to calculate the center and use center as exemplar to guide its flying. Apparently, (3) is meaningless for discrete data in X. Intuitionally, we may use mode (the value that appears most often) in place of center for discrete data. But using mode only may limit the exemplar to the most frequent one, and this is quite different from (3) where every component has contribution the center. Considering this, we randomly select a component from current swarm to represent the center. In this way, each component has a chance to be selected and more frequent one has bigger probability to be selected. To accomplish this, we define a new *selection* operator as follows. Given a list of position components $(x_{1,j}, x_{2,j}, \ldots, x_{m,j})$ in j-th dimension, the *selection* operator $\uplus$, which is used to calculate the exemplar component c_j in j-th dimension, is defined as (7). Similar to position component x_j and velocity component v_j, the meaning of exemplar component c_j also represents an edge from city j to city c_j.

$$c_j = \uplus_{i=1}^{m} x_{i,j} = x_{k,j}, \text{ where } k \text{ is randomly selected from } (1, 2, \ldots, m) \quad (7)$$

Given a position $\boldsymbol{x}$ and a velocity component v_j, the *plus* operator $\oplus$, which is used to produce a new position, is defined as (8).

$$\boldsymbol{x} \oplus v_j = \min(inverse(\boldsymbol{x}, v_j), insert(\boldsymbol{x}, v_j), swap(\boldsymbol{x}, v_j)) \quad (8)$$

where $inverse(\boldsymbol{x}, v_j)$, $insert(\boldsymbol{x}, v_j)$, and $swap(\boldsymbol{x}, v_j)$ are *inverse*, *block insert*, and *swap* operator respectively. The $inverse(\boldsymbol{x}, v_j)$ operator produces a new solution by inversing the visiting sequnce of cities between x_j and v_j. The $insert(\boldsymbol{x}, v_j)$ operator produces a new solution by moving a block of cities leading by v_j to the front of x_j. The $swap(\boldsymbol{x}, v_j)$ operator produces a new solution by swapping the positions of x_j and v_j. This *plus* operator is a greedy hybrid operator which has been used by zhong et al. in Ref. [9]. This is a kind of operator with multiple neighbors, which selects the best one from those neighbors. Specifically, for the edge e_{j,v_j} represented by velocity component v_j, it uses *inverse* operator, *block insert* operator and *swap* operator to produce three neighbor solutions. And the best one is used as the candidate solution. In *block insert* operator, the length of block is randomly produced for each insert operation.

Using the new operators defined above, we can define *flying* operator as (1).

$$\begin{aligned} \boldsymbol{x} &= \boldsymbol{x} \oplus (P \ominus \boldsymbol{x}) \\ &= ((((\boldsymbol{x} \oplus (p_1 \ominus x_1)) \oplus (p_2 \ominus x_2)), \ldots) \oplus (p_n \ominus x_n)) \end{aligned} \quad (9)$$

In DPIO algorithm, both the map and compass operator and the landmark operator use (1) to produce new solution, but the meaning of exemplar component p_j and the next visiting city list nc in *minus* operator (6) are dependent on the stage pigeon is in. If a pigeon is in the first stage, then p_j is the j-th element of personal best solution of a randomly selected pigeon and *Full* strategy is used to create nc. If a pigeon is in the second stage, then p_j is the j-th element of the 'center' produced by *selection* operator (7) and *Heuristic* strategy is used to create nc.

In the case of map and compass operator, there are two significant differences between (1) and the original one (represented by (1) and (2)):

(a) In the original one, only the global best solution is used to guide the flying of pigeon. In (1), a pigeon can not only learn from the global best solution, but also can learn from best solution of different pigeon for different dimension. This is a kind of comprehensive learning strategy, which was first proposed by Liang et al. [37], and was also successfully used for TSP [9]. This feature makes the pigeons have more exemplars to learn from and a larger potential space to fly, as a result, it may enable the DPIO to make use of the information in swarm more effectively to generate better quality solutions frequently.

(b) Different from the original one, (1) does not use original velocity. In stead, the *minus* operator (6) will randomly produce a new velocity from next visiting city list in case the two operands are the same. For map and compass operator, we use *Full* strategy to create the next visiting city list for (6). The philosophy behind this scheme lies in two aspects: (1) in most cases, the edges in original velocity are already in current position. In this case, to add the edges in original velocity into current position is meaningless; (2) to produce new velocity from all available cities may enhance diversity.

In the case of landmark operator, there are also two significant differences between (1) and the original one (represented by (3) and (4)):

(a) The meaning of center-guiding is different. In DPIO, exemplar component is a randomly selected component from current swarm. This is a kind of cooperative learning which not only can guarantee that each pigeon has a chance to be selected as exemplar, but also can keep appropriate diversity for the swarm.

(b) In DPIO algorithm, heuristic information is used to improve its intensification ability. Specifically, in the landmark operator, we use *Heuristic* strategy to create the next visiting city list for (6).

3.2. Metropolis acceptance strategy

Aim at obtaining better balance between diversification and intensification, DPIO uses the Metropolis acceptance criterion of SA to determine whether a worse solution should be accepted as new current solution. Suppose $\boldsymbol{x}$ is the current solution with a cost $f(\boldsymbol{x})$ and $\boldsymbol{y}$ is the newly generated solution with a cost $f(\boldsymbol{y})$. When $f(\boldsymbol{y}) \leq f(\boldsymbol{x})$, it means the generated solution $\boldsymbol{y}$ is not worse than the current solution $\boldsymbol{x}$. Then $\boldsymbol{y}$ is accepted as new current solution. On the contrary, when $f(\boldsymbol{y}) > f(\boldsymbol{x})$, it means $\boldsymbol{y}$ is worse than $\boldsymbol{x}$, then DPIO will use the probability mechanism of Metropolis acceptance criterion to determine whether or not to accept $\boldsymbol{y}$. The acceptance probability is described as (10).

$$p = \begin{cases} 1, & if\ f(\boldsymbol{y}) \leqslant f(\boldsymbol{x}) \\ e^{-(f(\boldsymbol{y})-f(\boldsymbol{x}))/t}, & otherwise \end{cases} \quad (10)$$

where $t > 0$ is the parameter temperature.

Algorithm 1 Algorithm for creating initial temperature list.
Input: *len* The length of initial temperature list, X Set of solutions
Output: A priority list of temperature
1: Create an empty priority list *lst*
2: **while** the length of *lst* is less than $2 \times len$ **do**
3: Select a solution X_i from X randomly
4: Produce candidate solution $\boldsymbol{y}$ using *inverse*, *insert*, or *swap* operator
5: Insert $|f(\boldsymbol{y}) - f(X_i)|$ into *lst*
6: **if** $\boldsymbol{y}$ is better than X_i **then**
7: $X_i = \boldsymbol{y}$
8: **end if**
9: **end while**
10: Remove the top $len/2$ elements from *lst*
11: Remove the bottom $len/2$ elements from *lst*
12: **Return** *lst*

In order to apply the Metropolis acceptance criterion of SA algorithm, DPIO algorithm must specify a cooling strategy for parameter t. In the literature of SA algorithm, there exists many cooling strategies. To simplify the implementation of DPIO algorithm, we use the idea of list-based cooling scheme proposed in list-based SA [33]. In list-based cooling scheme, the initial temperature and the temperature cooling are determined by the algorithm automatically. Specifically, a list of temperature is created first, and then, in each iteration, the maximum

value t_{max} in the list is used as current temperature t to be used in (10). The temperature list is updated adaptively according to the topology of the solution space of the problem and the search process (see Algos. 2 and 3). Algo. 1 is used to create initial temperature list. Like in Ref. [9], DPIO uses absolute value of cost difference as initial temperature. Different from Ref. [9], some extreme values are deleted to reduce the disturbing of noise.

3.3. Implementation of DPIO algorithm

Algo. 2 describes the flying operator of pigeon in each iteration, and Algo. 3 describes the pseudo code of DPIO algorithm. In Algo. 2, variable *total* and variable *counter* are used to produce new temperature for temperature list. In case a worse position is accepted, line 18 calculates the corresponding temperature t, and then it is added to *total* in line 19 and the corresponding counter *counter* is increased by 1 in line 20. Late in Algo. 3, the average of t is used to update the temperature list in line 23. In Algo. 3, input parameter M, MG, R, and L represents the size of swarm, the maximum generation, the ratio of first stage to the maximum generation, and the length of temperature list. Variable *fs* represents the iteration times of first stage, variable *dg* is used to store the generation where next swarm size decreasing happens, and variable *ss* is used to store the swarm size of remaining successful pigeons. In the second stage of DPIO algorithm, line 9 and line 10 of Algo. 3 divide the remaining successful swarm into successful group and unsuccessful group, and the successful group includes at least 2 pigeons. This dividing will keep unchanged for the half remaining generations. Line 11 of Algo. 3 is used to calculate the generation where next swarm size decreasing happens. The philosophy behind keeping a minimum size of 2 for successful group is to have exemplar to learn from. In line 17, the pigeons in remaining successful swarm call Algo. 2 to search for new solution. In line 19, the pigeons in unsuccessful swarm will randomly follow a pigeon in successful swarm to search for new solution.

Algorithm 2 Algorithm of flying operator.

1: **for** each dimension j in (1, 2, …,n) **do**
2: **if** pigeon X_i is in first stage **then**
3: Select another pigeon k randomly
4: $v_j \Longleftarrow p_{k,j} \ominus x_{i,j}$
5: **else**
6: $c_j \Longleftarrow \uplus_{k=1}^{m} x_{k,j}$
7: $v_j \Longleftarrow c_j \ominus x_{i,j}$
8: **end if**
9: $\boldsymbol{y} \Longleftarrow X_i \oplus v_j$
10: $p \Longleftarrow$ acceptance probability calculated using (10)
11: $r \Longleftarrow$ a random number in range [0, 1)
12: **if** $r \leq p$ **then**
13: $X_i \Longleftarrow \boldsymbol{y}$
14: **if** $f(X_i) < f(P_i)$ **then**
15: $P_i \Longleftarrow X_i$//To store new personal best position
16: **end if**
17: **if** $f(\boldsymbol{y}) - f(X_i) > 0$ **then**
18: $t \Longleftarrow -()f(\boldsymbol{y}) - f(X_i)/\ln(r)$//To calculate the corresponding temperature
19: $total \Longleftarrow total + t$
20: $counter \Longleftarrow counter + 1$
21: **end if**
22: **end if**
23: **end for**

The time complexity of DPIO algorithm can be analyzed as follows. Among the four basic operators of DPIO algorithm, *minus* operator $\ominus$ and *selection* operator $\uplus$ are independent of the city number n. The *plus* operator $\oplus$ can be implemented by two steps. The first step is to evaluate *inverse* neighbor, *insert* neighbor, and *swap* neighbor. The time complexity of evaluation is $O(1)$. The second step is to create the best neighbor. If linked list is used to represent a solution, then the time complexity of *insert* operator and *swap* operator is $O(1)$, but the time complexity of *inverse* operator is $O(n)$. As a result, the worst-case time complexity of *plus* operator is $O(n)$. Because the *flying* operator uses *plus* operator n times, the time complexity of *flying* operator is $O(n^2)$. The function of Algo. 2 is to implement *flying* operator, so the time complexity of Algo. 2 is $O(n^2)$. The main structure of Algo. 3 is a nested loop. The iteration times of outer loop is maximum generation MG, and the iteration times of inner loop is swarm size M for each outer loop. In each iteration of inner loop, Algo. 2 with time complexity $O(n^2)$ is called to produce a neighbor solution. As a result, the time complexity of DPIO algortihm is $O(MG \times M \times n^2)$.

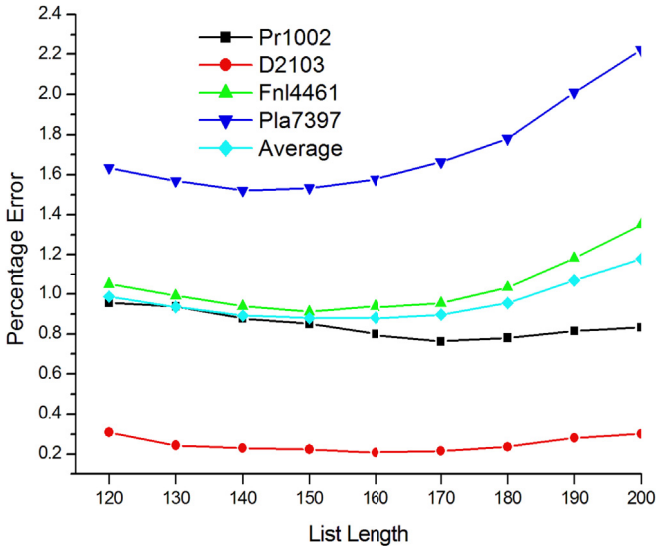

Fig. 1. Performance comparison of DPIO algorithms with different temperature list length L.

4. Behaviours analysis

In this section, three experiments were performed on four benchmark TSP instances to analyze the behaviors of DPIO algorithm. The first experiment was used to tune the parameters of DPIO Algorithm and analyze the convergence behaviours of DPIO algorithm. The second experiment was used to analyze the behaviour of map and compass operator. And the third was carried to analyze the behaviour of landmark operator. Those three experiments were carried on Pr1002, D2103, Fnl4461, and Pla7397 instances from TSPLIB. The best known integer solution of those problems are 259045, 80450, 182566 and 23260728 respectively. The stop condition is 1000 generations and the swarm size is 10. Percentage error of average tour length to best known tour length, which was calculated on 100 runs, is used to compare DPIO algorithm with different parameters or different variants of DPIO Algorithm.

4.1. Parameter tuning and convergence analysis

There are two main parameters in DPIO Algorithm. Parameter L, which is the length of temperature list, is used to control the temperature change for Metropolis acceptance criterion. Parameter R, which is the ratio of first stage to total generation, is used to control the iteration times of first stage and second stage. To find a suitable list length L, we have tested 9 different values for L from 120 to 200 with a step 10. Fig. 1 is the relation between percentage error and L. Similar to those results obtained by Refs. [9,33], we have following results: (1) the optimal list length is instance-dependent, for example, the optimal list length is 170 for Pr1002, but the optimal list length is 140 for Pla7397;

Algorithm 3 DPIO algorithm.
Input: M Swarm size, MG Maximum generation, R Ratio of first stage, L Length of temperature list;
Output: best solution found;
1: $fs \Longleftarrow R \times MG$ //The generation where first stage ends
2: $dg \Longleftarrow fs + 1$ //The generation where swarm size decreases by half
3: $ss \Longleftarrow M$ //The actual swarm size
4: Initialize X_i, V_i, and P_i for each pigeon i;
5: $\boldsymbol{g} \Longleftarrow$ the best P among all pigeons;
6: Call Algo. 1 to create initial temperature list lst with length L;
7: **for** $cg = 1$ to MG **do**
8: **if** $cg = dg$ **then** //Need to decrease the swarm size
9: Sort the pigeons from best to worst;
10: $ss \Longleftarrow ss \div 2 + 1$ //Decrease the swarm size by half
11: $dg \Longleftarrow (dg + MG) \div 2$ //The next generation when the swarm size is decreased by half
12: **end if**
13: $t_{\max} \Longleftarrow$ The maximum value in lst. //$t_{\max}$ is used by line 10 in Algo. 2
14: $total \Longleftarrow 0$, $counter \Longleftarrow 0$ //Those variables are maintained by line 19 and line 20 in Algo. 2
15: **for** $i = 1$ to M **do**
16: **if** $i \leq ss$ **then** //For pigeons in successful swarm
17: Use X_i to call Algo. 2
18: **else**//For pigeons in unsuccessful swarm
19: Randomly select a successful pigeon to call Algo. 2
20: **end if**
21: **end for**
22: **if** $counter > 0$ **then**
23: Replace the maximum value of lst with $total/counter$
24: **end if**
25: $\boldsymbol{g} \Longleftarrow$ the best P among all pigeons;
26: **end for**
27: **Return** $\boldsymbol{g}$.

(2) list length is more robust on small instances than on large instances, for example, big list length will notably deteriorate the performance of DPIO on Pla7397. As a result, list length should not be too big for large TSP instances. In terms of the average performance on those four instances, DPIO has best performance when L is 150. According to the simulation results, the temperature list length L is set to 150 in the following simulations.

To find a suitable ratio R, we have tested 9 different values for R from 0.1 to 0.9 with a step 0.1. Fig. 2 is the relation between percentage error and R. The optimal ratio R for Pr1002, D2103, Fnl4461, and Pla7397 is 0.4, 0.4, 0.5, and 0.3 respectively. Obviously, the optimal ratio R depends on the instance. In terms of the average performance on those four instances, DPIO has best performance when R is 0.4. According to the simulation results, the ratio R is set to 0.4 in the following simulations.

To analyze the convergence behaviors of DPIO algorithm, we compare the convergence process of DPIO algorithms with greedy acceptance strategy and Metropolis acceptance strategy. When Metropolis acceptance strategy is used, we use three different list lengths, that is 50, 150, and 250. Figs. 3–6 are the convergence process of DPIO algorithm on Pr1002, D2103, Fnl4461, and Pla7397 respectively. Those figures clearly show that: (1) Metropolis acceptance strategy is apparently better than greedy acceptance strategy. If greedy acceptance strategy is used, DPIO algorithm will converge quickly and lead to premature convergence easily; (2) the convergence process of DPIO algorithm is heavily controlled by parameter list length. The smaller the list length is, the quicker DPIO algorithm converges. As a result, DPIO algorithm with smaller list length will be more easily to fall into local minimum. On the contrary, The bigger the list length is, the slower DPIO algorithm converges. As a result, DPIO algorithm with bigger list length may waste more time on accepting worse solution. A suitable list length is important for DPIO algorithm to have good performance.

4.2. Analysis of map and compass operator

In the specific implementation of map and compass operator, we may have following two main kinds of alternatives: (1) considering the

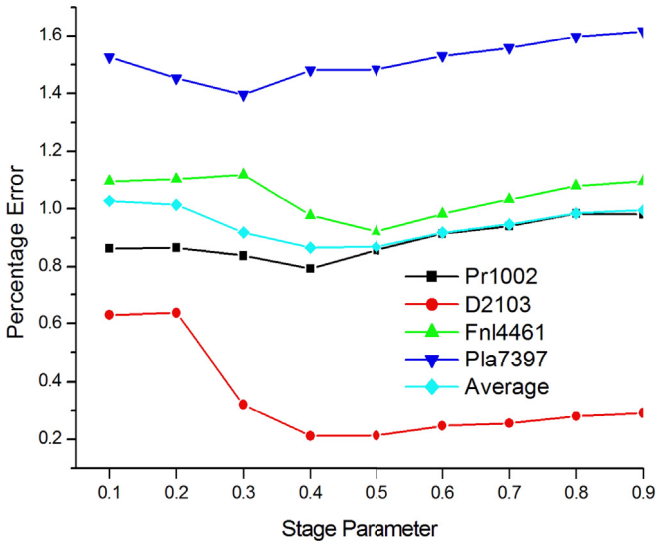

Fig. 2. Performance comparison of DPIO algorithms with different first stage ratio R.

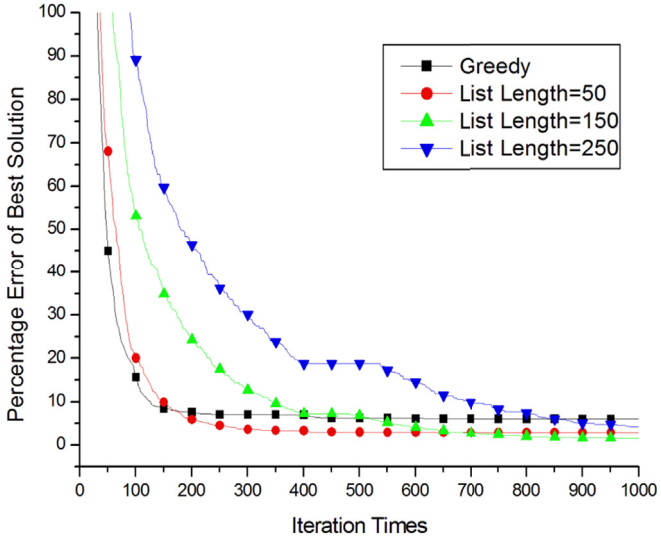

Fig. 3. Convergence process on Pr1002 instance.

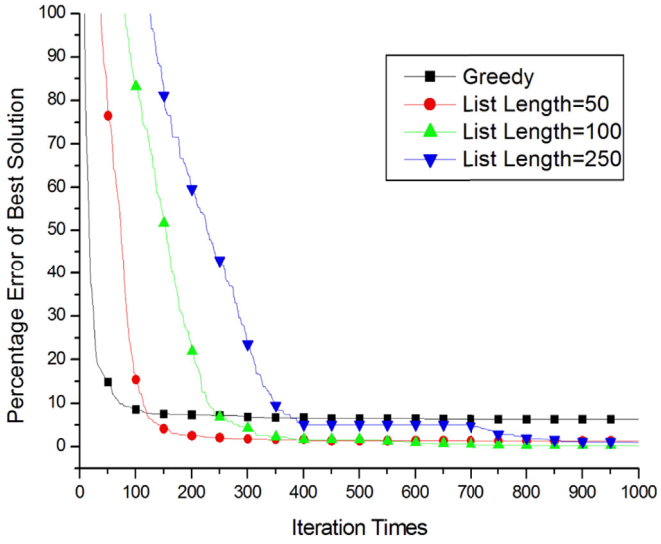

Fig. 4. Convergence process on D2103 instance.

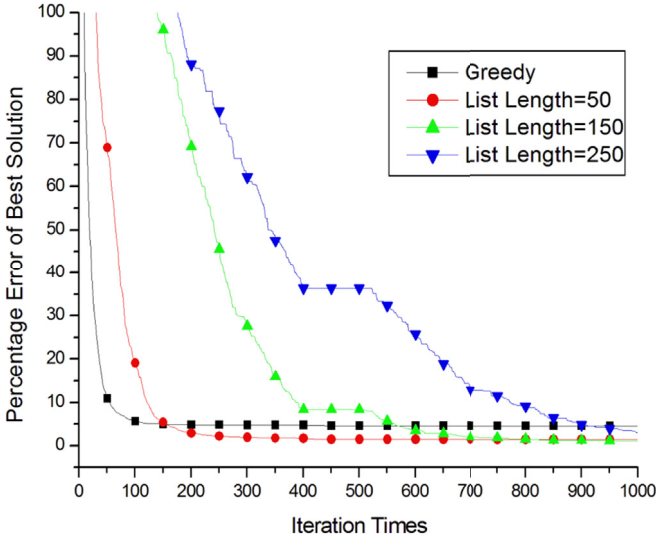

Fig. 5. Convergence process on Fnl4461 instance.

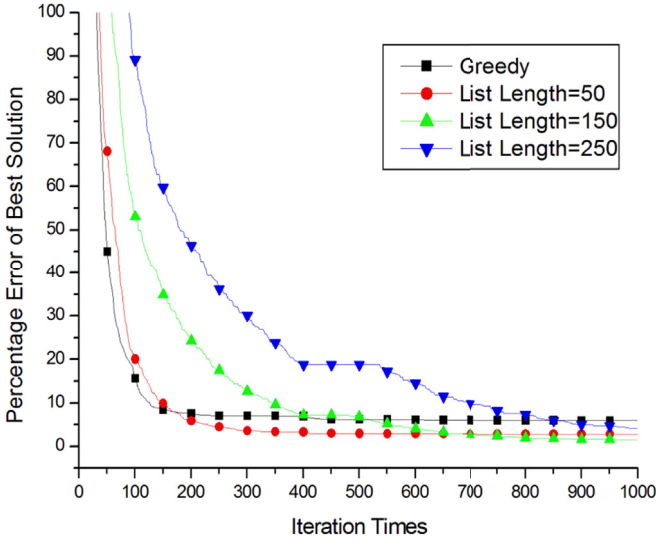

Fig. 6. Convergence process on Pla7397 instance.

exemplar P in (1), we may use global best position (global best-based learning) or personal best position of randomly selected pigeon (comprehensive learning); (2) considering the next visiting city list *nc* in (6), we may use *Empty* list, *Full* list, or *Heuristic* list. To observe which combination has best performance, we implement six variants of DPIO algorithm, DPIO with global best-based leaning and *Empty* list, DPIO with global best-based learning and *Full* list, DPIO with global best-based learning and *Heuristic* list, DPIO with comprehensive learning and *Empty* list, DPIO with comprehensive learning and *Full* list, and DPIO with comprehensive learning and *Heuristic* list. When *Heuristic* list is used, the number of nearest cities in *nc* is equal to 15. Fig. 7 is the simulation results. In Fig. 7, GL means global best-based leaning and CL means comprehensive learning. Fig. 7 clearly shows that comprehensive learning has better effect than global best-based leaning. When comprehensive learning is used, *Heuristic* list is apparently worse than other two schemes. Among the 4 TSP instances, *Full* list has better effect than *Empty* list on 3 instances. Based on those results, map and compass operator with comprehensive learning and *Full* list has best performance.

4.3. Analysis of landmark operator

Similarly, in the specific implementation of landmark operator, we may have following two main kinds of alternatives: (1) considering the exemplar p_j in (1), we may use mode edge (mode-based learning) or a randomly selected edge (cooperative learning) from the set of edges $\{x_{i,j}, i \in (1, 2, \ldots, M)\}$; (2) considering the next visiting city list *nc* in (6), we may use *Empty* list, *Full* list, or *Heuristic* list. To observe which combination has best performance, we implement six variants of DPIO algorithm, DPIO with mode-based learning and *Empty* list, DPIO with mode-based learning and *Full* list, DPIO with mode-based learning and *Heuristic* list, DPIO with cooperative learning and *Empty* list, DPIO with cooperative learning and *Full* list, and DPIO with cooperative learning and *Heuristic* list. When *Heuristic* list is used, the number of nearest cities in *nc* is equal to 15. Fig. 8 is the simulation results. In Fig. 8, ML means mode-based learning and CL means cooperative learning. Fig. 8 clearly shows that, no matter whether mode-based leaning or cooperative learning is used, *Heuristic* list significantly outperforms *Empty* list and *Full* list. When *Heuristic* list is used, cooperative learning outperforms mode-based learning on 3 TSP instances. Based on those results,

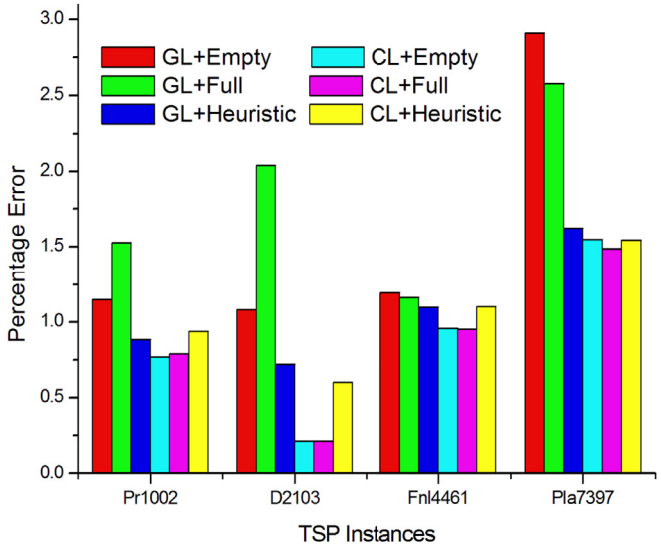

Fig. 7. Performance comparison of DPIO algorithms with different map and compass operator.

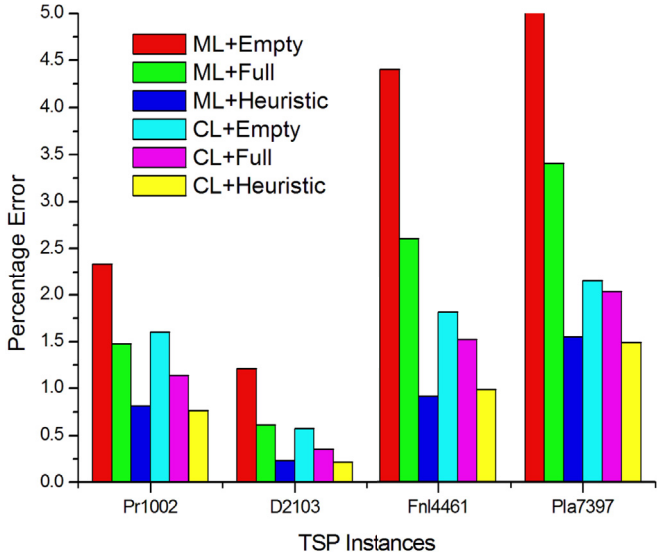

Fig. 8. Performance comparison of DPIO algorithms with different landmark operator.

landmark operator with cooperative learning and *Heuristic* list has best performance.

5. Comparative experiments

In order to observe the performance of DPIO algorithm, DPIO algorithm was tested on 33 large-scale TSP instances from TSPLIB with city number from 1000 to 85900. In order to observe the competitiveness of DPIO algorithm, DPIO algorithm was compared with the state-of-the-art meta-heuristics which were published in recent two years and were tested on more than 10 large-scale TSP instances from TSPLIB. In all the following experiments, the length of temperature list L is 150, the first stage ratio R is 0.4, and the maximum generation is 1000. Like in HDABC [16] and D-CLPSO [9], a suitable population size is selected for each instance such that DPIO algorithm can have good robustness with reasonable run-time. The specific population size M is set according to (11). Each instance was independently run 25 times. Percentage error of average solution (PEav) is used to compare the performance of different algorithms. Wilcoxon signed ranks test with 0.05 significant level is used to compare the PEav of DPIO with compared algorithms. The following experiments were run on an Intel Core i5-3470 CPU, with 3.2 GHz and a RAM of 4 GB. Java was used as the programming language. We are not able to obtain all the source codes of the compared algorithms. Except for those being explicitly explained, we obtained the simulation results of those algorithms from their original literature directly. As a result, the comparison was performed on different sets of instances and those simulation experiments of the compared algorithms were tested on different platforms.

$$M = \begin{cases} 30, & \text{else if } n < 2000 \\ 20, & \text{else if } n < 4000 \\ 10, & \text{else if } n < 50000 \\ 6, & \text{otherwise} \end{cases} \tag{11}$$

where n is the city number of TSP instance.

DPIO algorithm is compared with SOS-SA [19], HDABC [16], and D-CLPSO [9] on 17 benchmark instances with float distance. The SOS-SA was encoded with Matlab R2014b and run on a 2.83 GHz CPU Desktop with 2 GB RAM. Table 1 is the simulation results where the best result is highlighted in bold. In Table 1, the run-time of SOS-SA algorithm has been normalized according to the frequency of the CPU used by DPIO algorithm. The average PEav of SOS-SA, HDABC, D-CLPSO, and DPIO are 1.896, 1.255, 1.204, and 1.006 respectively. Among the 17 instances, DPIO obtains best PEav on 12 instances, SOS-SA obtains best PEav on 3 instances, and HDABC obtains best PEav on 2 instances. Wilcoxon signed ranks test is used to compare the PEav of DPIO with SOS-SA, HDABC, D-CLPSO. For DPIO and SOS-SA, the computed R^+, R^-, and p-value are 120, 16, and 0.004 respectively. For DPIO and HDABC, the computed R^+, R^-, and p-value are 120, 16, and 0.004 respectively. For DPIO and D-CLPSO, the computed R^+, R^-, and p-value are 119, 17, and 0.005 respectively. It means DPIO is significantly better than SOS-SA, HDABC, and D-CLPSO. The superiority of DPIO over HDABC and D-CLPSO can be explained as follows. Compared with HDABC, DPIO, which uses comprehensive learning strategy in first stage, can have better exploitation ability. Compared with D-CLPSO, DPIO, which uses *full* list in the first stage and uses randomly selected pigeon as exemplar in the second stage, can have better exploration ability. As a result, DPIO can obtain better balance between exploration and exploitation.

DPIO algorithm is compared with ESACO [2], MAS [3], and a self-organizing map neural network (SOM) [28] on benchmark instances with integer distance. The ESACO algorithm was encoded with C++ language and run on an Intel CPU 2.0 GHz with 1.0G RAM. The ESACO algorithm has 10 ants and stops after 300 itertions, during which the 2-opt local search was used to improve solution constructed by each ant. The MAS algorithm was encoded with C++ language and run on an Intel Xeon E5440 CPU 2.83 GHz with 32 GB of RAM. The MAS algorithm stops after the evaluation of $n * 5000$ feasible tours, during which the 3-Opt local search algorithm was executed $n * 50$ times. SOM was run on an AMD Athlon 2.0 GHz. Table 2 is the simulation results where best result is highlighted in bold. The run-times of ESACO, MAS, and SOM have been normalized according to the frequency of the CPU used by DPIO algorithm. Wilcoxon signed ranks test is used to compare the PEav of DPIO with ESACO, MAS and SOM.

DPIO algorithm is compared with ESACO [2] on 11 benchmark instances. The average PEav of DPIO and ESACO are 0.993 and 0.700 respectively. The average run time of DPIO and ESACO are 216.7s and 253.3s respectively. DPIO obtains better PEav on 4 instances and ESACO obtains better PEav on 7 instances. For DPIO and ESACO, the computed R^+, R^-, and p-value are 43, 12, and 0.06 respectively. It means there is no significant difference between ESACO and DPIO. It

Table 1
Compare DPIO with SOS-SA, DHABC, and D-CLPSO on 17 benchmark instances with float distance.

No.	Instance	Optimal	SOS-SA		HDABC	D-CLPSO	DPIO	
			PEav	Time	PEav	PEav	PEav	Time
1	Pr1002	259045	1.06	11.33	0.71	0.8	**0.533**	14.3
2	Pcb1173	56892	1.19	7.72	0.77	0.89	**0.617**	17.8
3	D1291	50801	**0.96**	10.68	1.64	1.33	1.296	19.4
4	Rl1323	270199	0.56	9.75	0.5	0.42	**0.387**	22.2
5	Fl1400	20127	**0.52**	13.04	1.29	1.19	1.069	24.6
6	D1655	62128	3.19	14.32	1.28	1.42	**1.031**	27.5
7	Vm1748	336556	**0.05**	16.16	0.72	0.65	0.464	34.3
8	U2319	234256	0.46	16.02	**0.26**	0.4	0.812	34.2
9	Pcb3038	137694	1.46	22.7	1.03	1.08	**0.719**	43.5
10	Fnl4461	182566	1.63	28.95	1.3	1.37	**1.092**	44.2
11	Rl5934	556045	1.83	44.21	1.79	1.23	**1.049**	48.7
12	Pla7397	23260728	2.32	87.31	**1.47**	1.55	1.481	110.5
13	Usa13509	19982859	7.09	276.91	1.57	1.46	**1.174**	271.7
14	Brd14051	468385	1.8	328.00	1.45	1.46	**1.151**	304.6
15	D18512	645238	2.2	532.26	1.51	1.48	**1.143**	495
16	Pla33810	66048945	3.07	1680.3	1.81	2.1	**1.721**	1260.7
17	Pla85900	142382641	2.84	6714.0	2.23	1.64	**1.371**	5100.3
		Average	1.896	577.28	1.255	1.204	**1.006**	463.15

Table 2
Compare DPIO with ESACO, MAS and SOM on 33 benchmark instances with integer distance.

No.	Instance	Optimal	ESACO		MAS		SOM		DPIO	
			PEav	Time	PEav	Time	PEav	Time	PEav	Time
1	Dsj1000	18659688	–	–	1.928	139.21	6.46	38.32	**0.388**	16.6
2	Pr1002	259045	0.179	22.39	**0.062**	158.21	4.78	34.85	0.51	14.1
3	U1060	224094	–	–	0.554	170.14	5.12	42.46	**0.374**	15.3
4	Vm1084	239297	–	–	0.657	146.35	5.86	42.83	**0.327**	17.4
5	Pcb1173	56892	–	–	**0.242**	179.29	7.5	44.08	0.392	17.8
6	D1291	50801	–	–	0.679	296.67	9.66	50.99	**0.668**	19.3
7	Rl1304	252948	–	–	**0.047**	187.9	10	49.15	0.313	21.5
8	Rl1323	270199	–	–	0.52	149.88	9.45	51.58	**0.408**	22
9	Nrw1379	56638	–	–	1.234	200.86	4.61	50.26	**0.519**	23.2
10	Fl1400	20127	–	–	1.019	175.78	4.32	146.59	**0.419**	24.5
11	U1432	152970	–	–	**0.335**	263.17	5.02	60.49	0.388	23.9
12	Fl1577	22249	0.197	29.03	0.358	344.58	17.46	67.21	**0.178**	25.3
13	D1655	62128	–	–	**0.356**	236.61	9.6	62.13	0.369	27.2
14	Vm1748	336556	–	–	0.647	286.03	6.68	83.04	**0.454**	33.8
15	U1817	57201	–	–	1.304	326.94	9.68	78.14	**0.561**	30.3
16	Rl1889	316536	–	–	**0.154**	241.81	9.54	80.64	0.688	36.6
17	D2103	80450	–	–	0.514	286.63	19.15	82.81	**0.145**	23.8
18	U2152	64253	–	–	**0.799**	500.88	10.43	90.35	0.838	25.9
19	U2319	234256	–	–	**0.381**	336.51	1.72	119.61	0.838	34.1
20	Pr2392	378032	–	–	**0.319**	377.95	7.04	100.96	0.612	29.7
21	Pcb3038	137694	–	–	–	–	7.88	122.04	**0.624**	43.7
22	Fl3795	28772	**0.388**	74.58	–	–	16.13	243.26	1.52	67.3
23	Fnl4461	182566	**0.482**	120.39	–	–	5.62	206.4	0.961	44.1
24	Rl5915	565530	**0.602**	135.58	–	–	12.94	290.99	1.005	63.1
25	Rl5934	556045	–	–	**0.257**	–	13.02	298.97	1.041	68.7
26	Pla7397	23260728	**0.553**	133.71	2.099	–	10.19	426.44	1.441	113.6
27	Rl11849	923288	**0.764**	359.88	4.724	–	11.49	771.38	1.062	299.4
28	Usa13509	19982859	**1.062**	571.38	4.675	–	7.62	987.06	1.168	318.3
29	Brd14051	469388	1.217	426.56	–	–	6.18	912.13	**1.051**	347.7
30	D15112	1573084	1.03	485.46	4.049	–	5.95	1126.5	**0.984**	522.1
31	D18512	645244	1.227	427.78	–	–	6	1302.5	**1.049**	569.2
32	Pla33810	66050535	–	–	–	–	13.23	2992.88	**1.726**	1385.1
33	Pla85900	142383704	–	–	–	–	10.94	15646.25	**1.378**	5279.1
		Average	0.700	253.3	1.117	250.3	8.826	809.2	0.739	291.0

is worth noting that DPIO obtained better results on the three largest instances. It means DPIO has better scalability than ESACO.

DPIO algorithm is compared with MAS [3] on 25 benchmark instances. The average PEav of DPIO and MAS are 0.603 and 1.117 respectively. The average run time of DPIO and MAS are 24.1s and 250.3s respectively. DPIO obtains better PEav on 15 instances and MAS obtains better PEav on 10 instances. For DPIO and MAS, the computed R^+, R^-, and p-value are 198, 102, and 0.1 respectively. Although there is no significant difference between DPIO and MAS in terms of PEav, it is worth noting that DPIO obtained those results with far less time than MAS.

DPIO algorithm is compared with SOM [2] on 33 benchmark instances. The average PEav of DPIO and SOM are 0.585 and 8.826 respectively. The average run time of DPIO and SOM are 291.0s and

Table 3
Compare DPIO algorithm with 10 algorithms published in recent years.

No.	Algorithm	T	N	Dims	PEav	Time	PEav1	Time1	$R+$	$R-$	p-value
1	GA-EAX	I	33	[1000, 85900]	**0.052**	4961.9	0.739	291.0	513	15	2.10e-6
2	HBMO	I	33	[1000, 85900]	**0.076**	251.4	0.739	291.0	528	0	5.40e-7
3	ASA-GS	F	17	[1002, 85900]	3.317	1321.6	**1.006**	463.2	136	0	2.93e-4
3	MSA-IBS	I	17	[1002, 85900]	1.412	580.3	**0.853**	491.9	125	11	1.93e-3
4	LBSA	I	17	[1002, 85900]	1.467	577.9	**0.853**	491.9	135	1	3.52e-4
5	EHS	I	17	[1002, 85900]	1.732	239.7	**0.853**	491.9	131	5	7.13e-4
7	AHSA-TS	I	20	[1002, 18512]	1.402	431.8	**0.666**	91.7	186	4	1.63e-4
8	SSA	I	12	[318, 33810]	1.083	145.5	**0.521**	159.0	66	0	2.22e-3
9	TSHACO	I	12	[318, 33810]	3.871	–	**0.521**	159.0	66	0	2.22e-3
10	PCGA	I	13	[318, 18512]	4.588	2345.8	**0.577**	151.6	78	0	1.47e-3

809.2s respectively. DPIO is better than SOM on all instances. For DPIO and SOM, the computed R^+, R^-, and p-value are 528, 0, and 5.4e-7 respectively. It means DPIO is significantly better than SOM.

DPIO is also compared in brief with 10 meta-heuristics which have been tested on TSP instances with more than 10000 cities. Those meta-heuristics include GA using edge assembly crossover (GA-EAX) [24], honey bees mating optimization (HBMO) [14], adaptive SA with greedy search (ASA-GS) [32], multiagent SA with instance-based sampling (MSA-IBS) [34], list-based SA (LBSA) [33], evolutionary harmony search (EHS) [36], adaptive hybrid SA-tabu search (AHSA-TS) [30], swarm SA (SSA) [35], two-stage hybrid swarm intelligence optimization (TSHACO) [21], permutation-coded genetic algorithm (PCGA) [23]. Table 3 highlights the comparison of the DPIO algorithm with those meta-heuristics. The column of T denotes the type of distance between cities, where I and F represent integer and float respectively. The column of N denotes the number of total instances. The column of Dims denotes the city numbers of the smallest and the biggest instances. The column of PEav and the column of PEav1 denote the PEavs of the compared algorithm and the DPIO algorithm respectively. The better one is highlighted in bold in Table 3. The column of Time and the column of Time1 denote the run-times of the compared algorithm and the DPIO algorithm respectively. Except for PCGA, which didn't provide the frequency of the CPU, the run-times of the compared algorithms have been normalized according to the frequency of the CPU used by the DPIO algorithm. Except for GA-EAX, the results of other algorithms are directly obtained from corresponding papers. We obtained the source code of GA-EAX from the author and ran it on the 33 large-scale TSP instances from TSPLIB. To limit the run-time of GA-EAX to a reasonable value, the population size is set to 100. Table 3 clearly shows that GA-EAX and HBMO outperform other algorithms. Results of Wilcoxon signed ranks test show that, except GA-EAX and HBMO, DPIO algorithm is significantly better than the other 8 meta-heuristics. We also use Wilcoxon signed ranks test to compare the performance of GA-EAX and HBMO, the computed R^+, R^-, and p-value are 395, 133, and 0.04 respectively. It means GA-EAX is significantly better than HBMO, but GA-EAX costs much run-time than HBMO on the three largest TSP instances. Both the operators of GA-EAX and HBMO are heavily enhanced by the characteristics of TSP instance, it means that embedding the characteristics of TSP instance into operators is an effective way to improve the performance of meta-heuristics for solving TSP.

6. Conclusions

PIO algorithm is a novel SI algorithm which has been successfully applied to many continuous optimization fields. Using the two features, dividing of its search process and dividing of swarm in the second stage, PIO algorithm can flexibly adjust its operators to keep good balance between intensification and diversification. To keep those two good features for discrete problem, this paper redesigns the basic operators of PIO algorithm for TSP. Using those new basic operators, a DPIO algorithm with Metropolis acceptance criterion is implemented for TSP. In the map and compass operator of DPIO algorithm, comprehensive learning and *Full* next visiting city list are combined to improve DPIO's exploration ability in the first stage. In the landmark operator of DPIO algorithm, cooperative learning and heuristic information are combined to enhance DPIO's exploitation ability in the second stage. Furthermore, Metropolis acceptance criterion of SA algorithm is used to obtain better balance between intensification and diversification. The performance of DPIO algorithm was tested on the 33 large-scale TSP instances from TSPLIB and was compared with state-of-the-art meta-heuristics published in recent years. Simulation results confirm the effectiveness and competitiveness of DPIO Algorithm. Although DPIO algorithm has shown its effectiveness and competitiveness, some additional aspects are worth of further study:

- One shortage of DPIO algorithm is that it has several parameters that need fine-tuning, and those parameters are instance-dependent. One future direction is to investigate how to use adaptive strategy to find optimal parameters online.
- Another shortage of DPIO algorithm is the large run-time on large TSP instances. Although DPIO algorithm is population-based, and it is easy to implement iteration-level parallelism for population-based metaheuristics, two characteristics of DPIO algorithm are worth studying while exploring its parallelization. The first characteristic is that the second stage of DPIO algorithm is partial serial because different pigeons in unsuccessful group may follow the same pigeon in successful group. The second characteristic is that the *flying* operator (1) is serial. To implement a parallel DPIO algorithm, we may produce new solutions for all pigeons at the same time and define a parallel version of *flying* operator as (12):

$$\boldsymbol{x} = \min(\boldsymbol{x} \oplus (p_1 \ominus x_1), \boldsymbol{x} \oplus (p_2 \ominus x_2), \ldots, \boldsymbol{x} \oplus (p_n \ominus x_n)) \tag{12}$$

  It is worth investigating whether those strategies have good properties for parallelization while preserving the solution quality.
- DPIO algorithm needs centralized access to data, and thus it is impractical for large-scale applications where the data is too large to be stored on one single machine. In this new era of big data, there are many such applications, such as subset selection [47] and multivariate discretization for big data [48] etc. An interesting research direction is to study how to extend DPIO algorithm to leverage distributed environments for solving big data applications.

Conflicts of interest

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

Acknowledgement

The authors like to thank Yuichi Nagata for the source code of GA-EAX algorithm. This work was supported by Nature Science Foundation of Fujian Province of P. R. China (No. 2016J01280, No. 2019J01401)

and Special Fund for Scientific and Technological Innovation of Fujian Agriculture and Forestry University of P. R. China (No. CXZX2016026, No. CXZX2016031).

References

[1] H. Duan, P. Qiao, Pigeon-inspired optimization: a new swarm intelligence optimizer for air robot path planning, Int. J. Intell. Comput. Cybern. 7 (1) (2014) 24–37.

[2] H. Ismkhan, Effective heuristics for ant colony optimization to handle large-scale problems, Swarm Evol. Comput. 32 (2017) 140–149, https://doi.org/10.1016/j.swevo.2016.06.006.

[3] Y. Yan, H.-s. Sohn, G. Reyes, A modified ant system to achieve better balance between intensification and diversification for the traveling salesman problem, Appl. Soft Comput. 60 (2017) 256–267.

[4] W. Yong, Hybrid max–min ant system with four vertices and three lines inequality for traveling salesman problem, Soft Comput. 19 (3) (2015) 585–596.

[5] J.B. Escario, J.F. Jimenez, J.M. Giron-Sierra, Ant colony extended: experiments on the travelling salesman problem, Expert Syst. Appl. 42 (1) (2015) 390–410, https://doi.org/10.1016/j.eswa.2014.07.054.

[6] Z. Zhang, Z. Feng, Two-stage updating pheromone for invariant ant colony optimization algorithm, Expert Syst. Appl. 39 (1) (2012) 706–712 0.1016/j.eswa.2011.07.062.

[7] J. Ning, Q. Zhang, C. Zhang, B. Zhang, A best-path-updating information-guided ant colony optimization algorithm, Inf. Sci. 433 (2018) 142–162.

[8] W.-N. Chen, J. Zhang, H.S. Chung, W.-L. Zhong, W.-G. Wu, Y.-H. Shi, A novel set-based particle swarm optimization method for discrete optimization problems, IEEE Trans. Evol. Comput. 14 (2) (2010) 278–300, https://doi.org/10.1109/TEVC.2009.2030331.

[9] Y. Zhong, J. Lin, L. Wang, H. Zhang, Discrete comprehensive learning particle swarm optimization algorithm with metropolis acceptance criterion for traveling salesman problem, Swarm Evol. Comput. 42 (2018) 77–88.

[10] A. Ouaarab, B. Ahiod, X.-S. Yang, Discrete cuckoo search algorithm for the travelling salesman problem, Neural Comput. Appl. 24 (7–8) (2014) 1659–1669, https://doi.org/10.1007/s00521-013-1402-2.

[11] Y. Zhou, X. Ouyang, J. Xie, A discrete cuckoo search algorithm for travelling salesman problem, Int. J. Collab. Intell. 1 (1) (2014) 68–84, https://doi.org/10.1504/IJCI.2014.064853.

[12] E. Osaba, X.-S. Yang, F. Diaz, P. Lopez-Garcia, R. Carballedo, An improved discrete bat algorithm for symmetric and asymmetric traveling salesman problems, Eng. Appl. Artif. Intell. 48 (2016) 59–71, https://doi.org/10.1016/j.engappai.2015.10.006.

[13] Y. Saji, M.E. Riffi, A novel discrete bat algorithm for solving the travelling salesman problem, Neural Comput. Appl. 27 (7) (2016) 1853–1866, https://doi.org/10.1007/s00521-015-1978-9.

[14] Y. Marinakis, M. Marinaki, G. Dounias, Honey bees mating optimization algorithm for the euclidean traveling salesman problem, Inf. Sci. 181 (20) (2011) 4684–4698, https://doi.org/10.1016/j.ins.2010.06.032.

[15] L.-P. Wong, M.Y.H. Low, C.S. Chong, Bee colony optimization with local search for traveling salesman problem, Int. J. Artif. Intell. Tools 19 (03) (2010) 305–334, https://doi.org/10.1142/S0218213010000200.

[16] Y. Zhong, J. Lin, L. Wang, H. Zhang, Hybrid discrete artificial bee colony algorithm with threshold acceptance criterion for traveling salesman problem, Inf. Sci. 421 (2017) 70–84.

[17] J.B. Odili, M.N. Mohmad Kahar, Solving the traveling salesman's problem using the african buffalo optimization, Comput. Intell. Neurosci. 2016 (2016) 3.

[18] A.E.-S. Ezugwu, A.O. Adewumi, Discrete symbiotic organisms search algorithm for travelling salesman problem, Expert Syst. Appl. 87 (2017) 70–78.

[19] A.E.-S. Ezugwu, A.O. Adewumi, M.E. Frîncu, Simulated annealing based symbiotic organisms search optimization algorithm for traveling salesman problem, Expert Syst. Appl. 77 (2017) 189–210.

[20] S.-M. Chen, 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. 38 (12) (2011) 14439–14450, https://doi.org/10.1016/j.eswa.2011.04.163.

[21] W. Deng, R. Chen, B. He, Y. Liu, L. Yin, J. Guo, A novel two-stage hybrid swarm intelligence optimization algorithm and application, Soft Comput. 16 (10) (2012) 1707–1722, https://doi.org/10.1007/s00500-012-0855-z.

[22] J. Wang, O.K. Ersoy, M. He, F. Wang, Multi-offspring genetic algorithm and its application to the traveling salesman problem, Appl. Soft Comput. 43 (2016) 415–423, https://doi.org/10.1016/j.asoc.2016.02.021.

[23] P.V. Paul, N. Moganarangan, S.S. Kumar, R. Raju, T. Vengattaraman, 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. 32 (2015) 383–402, https://doi.org/10.1016/j.asoc.2015.03.038.

[24] Y. Nagata, S. Kobayashi, A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem, Inf. J. Comput. 25 (2) (2013) 346–363.

[25] L.T. Kóczy, P. Földesi, B. Tü\H{u}-Szabó, Enhanced discrete bacterial memetic evolutionary algorithm-an efficacious metaheuristic for the traveling salesman optimization, Inf. Sci. 460 (2018) 389–400.

[26] Z. Xu, Y. Wang, S. Li, Y. Liu, Y. Todo, S. Gao, Immune algorithm combined with estimation of distribution for traveling salesman problem, IEEJ Trans. Electr. Electron. Eng. 11 (S1) (2016) S142–S154, https://doi.org/10.1002/tee.22247.

[27] Y. Zhou, Q. Luo, H. Chen, A. He, J. Wu, A discrete invasive weed optimization algorithm for solving traveling salesman problem, Neurocomputing 151 (2015) 1227–1236, https://doi.org/10.1016/j.neucom.2014.01.078.

[28] H. Wang, N. Zhang, J.-C. Créput, A massively parallel neural network approach to large-scale euclidean traveling salesman problems, Neurocomputing 240 (2017) 137–151.

[29] S. Hore, A. Chatterjee, A. Dewanji, Improving variable neighborhood search to solve the traveling salesman problem, Appl. Soft Comput. 68 (2018) 83–91.

[30] Y. Lin, Z. Bian, 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. 49 (2016) 937–952, https://doi.org/10.1016/j.asoc.2016.08.036.

[31] H. Zhang, J. Zhou, Dynamic multiscale region search algorithm using vitality selection for traveling salesman problem, Expert Syst. Appl. 60 (2016) 81–95, https://doi.org/10.1016/j.eswa.2016.05.007.

[32] X. Geng, Z. Chen, W. Yang, D. Shi, K. Zhao, Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search, Appl. Soft Comput. 11 (4) (2011) 3680–3689, https://doi.org/10.1016/j.asoc.2011.01.039.

[33] S.-h. Zhan, J. Lin, Z.-j. Zhang, Y.-w. Zhong, List-based simulated annealing algorithm for traveling salesman problem, Comput. Intell. Neurosci. 2016 (2016) 8.

[34] C. Wang, M. Lin, Y. Zhong, H. Zhang, Solving travelling salesman problem using multiagent simulated annealing algorithm with instance-based sampling, Int. J. Comput. Sci. Math. 6 (4) (2015) 336–353, https://doi.org/10.1504/IJCSM.2015.071818.

[35] C. Wang, M. Lin, Y. Zhong, H. Zhang, Swarm simulated annealing algorithm with knowledge-based sampling for travelling salesman problem, Int. J. Intell. Syst. Technol. Appl. 15 (1) (2016) 74–94.

[36] C. Wang, J. Lin, M. Lin, Y. Zhong, Evolutionary harmony search algorithm with metropolis acceptance criterion for travelling salesman problem, Int. J. Wirel. Mob. Comput. 10 (2) (2016) 166–173.

[37] J.J. Liang, A.K. Qin, P.N. Suganthan, S. Baskar, Comprehensive learning particle swarm optimizer for global optimization of multimodal functions, IEEE Trans. Evol. Comput. 10 (3) (2006) 281–295, https://doi.org/10.1109/TEVC.2005.857610.

[38] M.N. Ab Wahab, S. Nefti-Meziani, A. Atyabi, A comprehensive review of swarm optimization algorithms, PLoS One 10 (5) (2015) e0122827https://doi.org/10.1371/journal.pone.0122827.

[39] D. Zhang, H. Duan, Y. Yang, Active disturbance rejection control for small unmanned helicopters via levy flight-based pigeon-inspired optimization, Aircraft Eng. Aero. Technol. 89 (6) (2017) 946–952.

[40] Y. Sun, N. Xian, H. Duan, Linear-quadratic regulator controller design for quadrotor based on pigeon-inspired optimization, Aircraft Eng. Aero. Technol. 88 (6) (2016) 761–770.

[41] J. Pei, Y. Su, D. Zhang, Fuzzy energy management strategy for parallel hev based on pigeon-inspired optimization algorithm, Sci. China Technol. Sci. 60 (3) (2017) 425–433.

[42] X. Lei, Y. Ding, F.-X. Wu, Detecting protein complexes from dpins by density based clustering with pigeon-inspired optimization algorithm, Sci. China Inf. Sci. 59 (7) (2016) 070103.

[43] B. Zhang, H. Duan, Three-dimensional path planning for uninhabited combat aerial vehicle based on predator-prey pigeon-inspired optimization in dynamic environment, IEEE ACM Trans. Comput. Biol. Bioinform 14 (1) (2017) 97–107.

[44] S. Yu, Y. Xu, P. Jiang, F. Wu, H. Xu, Node self-deployment algorithm based on pigeon swarm optimization for underwater wireless sensor networks, Sensors 17 (4) (2017) 674.

[45] R. Dou, H. Duan, Lévy flight based pigeon-inspired optimization for control parameters optimization in automatic carrier landing system, Aero. Sci. Technol. 61 (2017) 11–20.

[46] Q. Xue, H. Duan, Robust attitude control for reusable launch vehicles based on fractional calculus and pigeon-inspired optimization, IEEE/CAA J. Autom. Sin. 4 (1) (2017) 89–97.

[47] C. Qian, G. Li, C. Feng, K. Tang, Distributed pareto optimization for subset selection, in: International Joint Conference on Artificial Intelligence, 2018, pp. 1492–1498.

[48] S. Ramirezgallego, S. Garcia, J.M. Benitez, F. Herrera, A distributed evolutionary multivariate discretizer for big data processing on Apache spark, Swarm Evol. Comput. 38 (2018) 240–250.