*mathematics*

*Article*

# Opposition-Based Ant Colony Optimization Algorithm for the Traveling Salesman Problem

**Zhaojun Zhang [1,†], Zhaoxiong Xu [1,†], Shengyang Luan [1,†], Xuanyu Li [1] and Yifei Sun [2,*]**

1 School of Electrical Engineering and Automation, Jiangsu Normal University, Xuzhou 221116, China; zzj921@163.com (Z.Z.); xzx94819@163.com (Z.X.); luan@jsnu.edu.cn (S.L.); lixuanyu5705@163.com (X.L.)
2 School of Computer Science & School of Physics and Information Technology, Shaanxi Normal University, Xi'an 710119, China
* Correspondence: yifeis@snnu.edu.cn; Tel.: +86-157-0521-5977
† These authors contributed equally to this work.

Received: 18 August 2020; Accepted: 22 September 2020 ; Published: 24 September 2020

**Abstract:** Opposition-based learning (OBL) has been widely used to improve many swarm intelligent optimization (SI) algorithms for continuous problems during the past few decades. When the SI optimization algorithms apply OBL to solve discrete problems, the construction and utilization of the opposite solution is the key issue. Ant colony optimization (ACO) generally used to solve combinatorial optimization problems is a kind of classical SI optimization algorithm. Opposition-based ACO which is combined in OBL is proposed to solve the symmetric traveling salesman problem (TSP) in this paper. Two strategies for constructing opposite path by OBL based on solution characteristics of TSP are also proposed. Then, in order to use information of opposite path to improve the performance of ACO, three different strategies, direction, indirection, and random methods, mentioned for pheromone update rules are discussed individually. According to the construction of the inverse solution and the way of using it in pheromone updating, three kinds of improved ant colony algorithms are proposed. To verify the feasibility and effectiveness of strategies, two kinds of ACO algorithms are employed to solve TSP instances. The results demonstrate that the performance of opposition-based ACO is better than that of ACO without OBL.

**Keywords:** opposition-based learning; ant colony optimization; opposite path; traveling salesman problems

## 1. Introduction

As an important branch of computational intelligence, swarm intelligence (SI) [1] provides a competitive solution for dealing with large-scale, nonlinear, and complex problems, and has become an important research direction of artificial intelligence. In the SI model, each individual constitutes an organic whole by simulating the behavior of natural biological groups. Although each individual is very simple, the group shows complex emergent behavior. In particular, it does not require prior knowledge of the problem and has the characteristics of parallelism, so it has significant advantages in dealing with problems that are difficult to solve by traditional optimization algorithms. With the deepening of research, more and more swarm intelligence algorithms have been proposed, such as ant colony optimization algorithm (ACO) [2], particle swarm optimization (PSO) [3], artificial bee colony algorithm (ABC) [4], firefly algorithm (FA) [5], cuckoo algorithm (CA) [6], krill herd algorithm [7], monarch butterfly optimization (MBO) [8], and moth search algorithm [9], etc.

ACO as one of the typical SI is first proposed by Macro Dorigo [2] based on the observation of group behaviors of ants in nature. During the process of food searching, ants will release pheromones in the path when they pass through. Pheromones can be detected by other ants and can affect their further path choices. Generally, the shorter the path is, the more intense the pheromones will be, which means the shortest path will be chosen with the highest probability. The pheromone in other paths will disappear with time. Therefore, given enough time, the optimal path will have the most condensed pheromone. In this way, ants will find the shortest path from their nest to the food source in the end.

ACO has advantages in reasonable robustness, distributed parallel computing, and easy combination with other algorithms. It has been successfully applied in many fields, including traveling salesman problem (TSP) [10,11], satellite control resource scheduling problem [12], knapsack problem [13,14], vehicle routing problem [15,16], and continuous function optimization [17–19]. However, conventional ACO is still far from perfect due to issues like premature convergence and long search time [20].

Many scholars have made substantial contributions to improve ACO, mainly focusing on two perspectives, including model modification and algorithms combination. For example, in the line of model improvement, an ant colony system (ACS) [21] employs a pseudo-random proportional rule, which leads to faster convergence. In ACS, only the pheromone of the optimal path will be increased after each iteration. To prevent premature convergence caused by excessive pheromones concentration in some paths, the max-min ant system (MMAS) modifies AS with three main strategies for pheromone [22], including limitation, maximum initialization, and updating rules. To avoid the early planning of the blind search, an improved ACO algorithm by constructing the unequal allocation initial pheromones is proposed in [23]. Path selecting is based on the pseudo-random rule for state transition. The probability is decided by the number of iterations, and the optimal solution. Introducing a penalty function to the pheromone updating, a novel ACO algorithm is addressed in [24] to improve the solution accuracy.

Considering the other primary kind of modification to the original ACO, algorithm combination, several approaches are proposed as well. A multi-type ant system (MTAS) [25] is proposed combining ACS and MMAS, inheriting advantages from both of these algorithms. Combining particle swarm algorithm (PSO) with ACO, a new ant colony algorithm was proposed in [26] and named PS-ACO. PS-ACO employs pheromones updating rules of ACO and searches mechanisms of PSO simultaneously to keep the trade-off between the exploitation and exploration. A multi-objective evolutionary algorithm via decomposition is combined with ACO, an algorithm, termed MOEA/D-ACO [27], which proposes a series of single-objective optimization problems to solve multi-objective optimization problems. Executing ACO in combination with a genetic algorithm (GA), a new hybrid algorithm is proposed in [28]. Embedding GA into ACO, this method improves ACO in convergence speed and GA in searching ability.

Besides the above primary improvement strategies considering model modification and algorithm combination, approaches based on machine learning are also proposed in recent decades [29]. On the one hand, swarm intelligence can be used to solve the optimization problems in deep learning. In deep neural networks, for example, convolutional neural network (CNN), the optimization of hyperparameters is an NP hard problem. Using the SI method can solve this kind of problem better. PSO, CS, and FA were employed to properly select dropout parameters concerning CNN in [30]. The hybridized algorithm [31] based on original MBO with ABC and FAs was proposed to solve CNN hyperparameters optimization. On the other hand, we can learn from machine learning to improve performance of SI. For example, information feedback models are used to enhance the ability of algorithms [32–34]. In addition, opposition-based learning (OBL) [35], which was first proposed by Tizhoosh, is a famous algorithm. Its main idea is to calculate all the opposite solutions after current iteration, and then optimal solutions are selected among the generated solutions and their opposite solutions for the next round of iteration. OBL has been widely accepted in SI, including ABC [36], differential evolution (DE) [37–39], and PSO [40,41], leading to reasonable performances.

Since opposite solutions to continuous problems are convenient to construct, OBL has been used to solve continuous problems more commonly as above, compared with discrete problems. OBL is combined with ACS and applied to solve the TSP as an example for discrete problems in [42] to acquire the better solution. The solution construction phase and the pheromone updating phase of ACS are the primary foci of this hybrid approach. Besides TSP, the graph coloring problem is also employed as a discrete optimization problem in [43], and an improved DE algorithm based on OBL is proposed, which introduces two different methods of opposition. In [44], a pretreatment step was added in the initial stage when the two-membered evolution strategy was used to solve the total rotation minimization problem. The opposite solutions generated by OBL is compared with the initial solutions randomly generated, and a better solution is selected for the subsequent optimization process.

Inspired by the idea of OBL, in this paper, a series of methods, focusing on the opposite solution construction and the pheromone updating rule, are proposed. Aiming to solve TSP, our proposed methods introduce OBL to ACO and enable ACO no longer limited to the local optimal solutions, avoid premature convergence, and improve its performances.

The rest of this paper is organized as follows. In Section 2, the background knowledge of ACO and OBL are briefly reviewed. In Section 3, the opposition-based extensions to ACO are presented. In Section 4, the effectiveness of the improvement is verified through experiments. Section 5 presents the conclusions of this paper.

## 2. Background

In this section, we will take AS as an example to introduce the main process of ACO algorithm. At the same time, some necessary explanations of OBL will also be given.

### *2.1. Ant System*

TSP can be described as finding the shortest route for a salesman who needs to visit each city at least once and no more than once [45]. TSP is a classical combination optimization problem which is employed to test ACO algorithms, and, therefore, TSP is used here as an example as well. The TSP includes symmetric TSP and asymmetric TSP. We only discuss symmetric TSP in this paper.

There are two primary steps in the AS algorithm, path construction, and pheromone updating [2]. During the first step, a solution is established according to the random proportion rule, and it can be described in detail as follows.

In the beginning, $m$ ants are randomly assigned to $n$ cities. At the $t$-th iteration, the probability, called the state transition probability, for the $k$-th ant to travel from the city $i$ to $j$ is

$$p_{ij}^{k}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}(t)]^{\beta}}{\sum\limits_{s\in J_k(i)}[\tau_{is}(t)]^{\alpha}[\eta_{is}(t)]^{\beta}}, & \text{if } j \in J_k(i) \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where $\tau_{ij}$ is the pheromone trail and $\eta_{ij}$ is the heuristic information, accordingly, while $\alpha$ and $\beta$ are parameters deciding their relative influences, respectively. Generally, $\eta_{ij} = 1/d_{ij}$ and $d_{ij}$ is the distance of the path $(i, j)$. $J_k(i)$ is the feasible neighborhood of $k$-th ant at the $i$-th city.

When all the ants finish touring around each city, pheromone updating is as follows:

$$\tau_{ij}(t+1) = (1-\rho)\,\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{2}$$

where $\rho$ $(0 < \rho \leq 1)$ is the evaporation rate, $\Delta\tau_{ij}^{k}$ represents the extra pheromone left in the path $(i, j)$ by the $k$-th ant. $\Delta\tau_{ij}^{k}$ could be decided through

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} \frac{Q}{L_k}, & \text{if ant } k \text{ passes the path } (i, j) \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

where $Q$ is the pheromone enhancement coefficient and $L_k$ is the total path length for the $k$-th ant.

### *2.2. Opposition-Based Learning*

In the continuous domain, OBL is employed to evaluate the current solutions and their opposite solutions. Among these solutions, optimal ones are selected to boost the searchability [46]. Relative definitions are given as follows.

**Definition 1.** *Let $x \in R$ be a real number defined on a specific interval $x \in [a,b]$. The opposite number $\tilde{x}$ is defined according to the following formula*

$$\tilde{x} = a + b - x \tag{4}$$

**Definition 2.** *Let $X_i = (x_{i1}, x_{i2}, \cdots, x_{iD})$ be a point in D dimensional space, $x_{ij} \in [a_{ij}, b_{ij}]$, $j = 1, 2, \cdots, D$. The opposite point $\tilde{X}_i = (\tilde{x}_{i1}, \tilde{x}_{i2}, \cdots, \tilde{x}_{iD})$ is defined by*

$$\tilde{x}_{ij} = a_{ij} + b_{ij} - x_{ij} \tag{5}$$

Experiments show that, if there is no prior knowledge of optimization problem, the probability that the opposite solution can reach the global optimum is higher than that of the random solution [47]. Based on the OBL, quasi-opposition based learning [48] and quasi-reflective based learning [49] are proposed later. In this paper, we only consider OBL.

Taking TSP as an example, its solution is a sequence of numbers as the indices of cities. In addition, according to the opposite solutions for a continuous domain, it is challenging to construct opposite solutions for TSP due to the features of a discrete domain. Therefore, only a few scholars have made contributions towards this topic, and Ergeze is one of them. In [43], Ergeze addresses the definition of opposite paths according to the moving direction. For example, the initial path for $n$ cities is given by

$$P = [1, 2, \cdots, n] \tag{6}$$

where the entries stand for the order of the cities that a salesman travels through. Then, the corresponding opposite path in a clockwise (CW) direction could be given by

$$P^{\mathrm{CW}} = \left[1, 1 + \frac{n}{2}, 2, 2 + \frac{n}{2}, \cdots, \frac{n}{2} - 1, n - 1, \frac{n}{2}, n\right] \tag{7}$$

where $n$ is even.

In the case when $n$ is odd, append an auxiliary city and make $n$ even. In the end, find opposite solutions according to Equation (7) and then remove this city. Since different moving directions may lead to different opposite paths or solutions, moving in a counterclockwise (CCW) will result in different opposite solutions compared with $P^{\mathrm{CW}}$.

When the number of cities is odd, one way of implementing CW opposition is to add an auxiliary city to the end of the path. After the opposite path is found, remove the auxiliary city.

## 3. Opposition-Based ACO

The method of construction opposite path based on OBL is given in this section. At the same time, in order to use the opposite path information, three kinds of frameworks of opposite-based ACO algorithms including ACO-Index, ACO-MaxIt, and ACO-Rand will also be proposed. In order to unify the content, the construction method of the opposite path will be combined with the specific algorithm. Details will be given in the following subsections.

### *3.1. ACO-Index*

According to the definition given in Equation (7), the same route may lead to different opposite paths. Taking a TSP of six cities as an example, path $(1, 2, 3, 4, 5, 6)$ and path $(2, 3, 4, 5, 6, 1)$ are the same path; however, their opposite paths, $(1, 4, 2, 5, 3, 6)$ and $(2, 5, 3, 6, 4, 1)$, are different.

In addition, the initialization procedure of ACO is not random compared with DE, but more similar to the greedy algorithm, which selects a closer according to the rule of state transition. Therefore, opposite paths are always longer than the original ones generally and cannot be used pheromone updating. Aiming to solve the shortcomings, a novel ACO algorithm, namely ACO-Index, is proposed based on a modified strategy of opposite path construction.

Opposite path construction is mainly composed of two steps. The first step is the path sorting, and the second is the decision of opposite path. Suppose the number of cities $n$ is even, then, during path sorting, put the path $P$ back into a cycle and appoint a particular city A as the starting city with index 1. In addition, the rest of the cities will be given indices according to their position in this cycle. In this way, we could get the indices $P_{\text{ind}} = [1, 2, ..., n]$.

During the second step, indices of the opposite path $P_{\text{ind}}^{\text{CW}}$ should be found through Equation (7) and the opposite path $P^{\text{CW}}$ can be found based on the indices $P_{\text{ind}}^{\text{CW}}$ appointed previously.

Moreover, when the number of cities $n$ is odd, an auxiliary index should be added to the end of the indices $P_{\text{ind}}$, and we could get $P_{\text{aux}}$. According to Equation (7), we could get the opposite indices $P_{\text{aux}}^{\text{CW}}$, and its last index is the auxiliary index itself. Remove the latest index, and we can get $P_{\text{ind}}^{\text{CW}}$. In addition, then, decide the opposite path $P^{\text{CW}}$ according to the opposite indices $P_{\text{ind}}^{\text{CW}}$.

In this way, opposite paths for different paths that share the same cycle route are the same. Pseudocode for opposite path construction is addressed in Algorithm 1.

**Algorithm 1** Constructing the opposite path

```
Input: original path $P$
1: Put the path back into a circle
2: Appoint a specific city A with index 1
3: Appoint other cities in this circle with indices 2, 3, $\cdots$, and get the indices $P_{\text{ind}} = [1, 2, \cdots, n]$
4: if $n$ is even
5:     Calculate the opposite indices $P_{\text{ind}}^{\text{CW}}$ according to Equation (7)
6: else
7:     Add an auxiliary index at the end of $P_{\text{ind}}$ and get $P_{\text{aux}}$
8:     Calculate the opposite indices $P_{\text{aux}}^{\text{CW}}$ according to Equation (7)
9:     Delete the final index from $P_{\text{aux}}^{\text{CW}}$ and get $P_{\text{ind}}^{\text{CW}}$
10: endif
11: Calculate the $P^{\text{CW}}$ based on $P_{\text{ind}}^{\text{CW}}$
Output: opposite path $P^{\text{CW}}$
```

Although some paths may be longer than the optimal path, they still contain useful information within themselves, which inspires us to apply them to reasonably modifying pheromone. For ACO algorithms, if the number of ants is $m$, the number of paths should also be $m$ for pheromone updating. In the proposed ACO-Index, the top $m_1$ shortest original paths and the top $m_2$ shortest opposite paths will be chosen to form the $m = m_1 + m_2$ paths. Algorithm 2 presents the pseudocode for pheromone updating.

**Algorithm 2** Updating pheromone

```
Input: original paths and opposite paths
1: Sort original paths and opposite paths by length
2: Select the top $m_1$ shortest paths and the top $m_2$ shortest opposite paths
3: Construct $m = m_1 + m_2$ new paths
4: Update pheromone according to Equation (2)
Output: Pheromone trail in each path
```

Algorithm 3 shows the pseudocode for the primary steps of ACO-Index for total iterations $N_{\max}$.

**Algorithm 3** ACO-Index algorithm

```
Input: parameters: $m, n, \alpha, \beta, \rho, Q, m_1, m_2, N_{\max}$
1: Initialize pheromone and heuristic information
2: for iteration index $N_c \leq N_{\max}$ do
3:     for $k = 1$ to $m$ do
4:         Construct paths according to Equation (1)
5:         Construct opposite paths through Algorithm 1
6:     endfor
7:     Update pheromone according to Algorithm 2
8: endfor
Output: the optimal path
```

### 3.2. ACO-MaxIt

Although ACO-Index modifies ACO with a better path construction strategy, it inherits a similar opposite path generation method from [43]. In this section, a novel opposite path generation method, together with a novel pheromone updating rule, is proposed as an improved ACO algorithm, named ACO-MaxIt, which will be described in detail as follows.

The mirror point $M$ is defined by

$$M = \left\lceil \frac{1+n}{2} \right\rceil \tag{8}$$

where $\lceil \cdot \rceil$ denotes the ceiling operator.

Considering the case when $n$ is odd, the opposite city $\tilde{C}$ for the current city $C$ could be defined as follows:

$$\tilde{C} = \begin{cases} C, & \text{if } C = M \\ C + M, & \text{if } C < M \\ C - M, & \text{if } C > M \end{cases} \tag{9}$$

Considering the case when $n$ is even, the opposite city $\tilde{C}$ for the current city $C$ could be defined as follows

$$\tilde{C} = \begin{cases} C, & \text{if } C = n/2 \text{ or } (n/2 + 1) \\ C + M, & \text{else if } C < M \\ C - M, & \text{else if } C > M \end{cases} \tag{10}$$

The pseudocode for opposite path construction is shown in Algorithm 4.

**Algorithm 4** Constructing the opposite path based on the mirror point

```
Input: original path
1: Decide mirror point $M$, according to Equation (8)
2: for $C = 1$ to $n$ do
3:     Calculate $\tilde{C}$ through Equation (9) or Equation (10) according to the parity of $n$
4: endfor
Output: opposite path
```

The pheromone update process consists of two stages. For the first stage, when $N_c \leq gN_{\max}$ and $0 < g < 1$, opposite paths will be decided through Algorithm 4. Meanwhile, the pheromone will be updated according to Algorithm 2. In the later stage, when $N_c > gN_{\max}$, no more opposite paths could be calculated, and pheromones will still be updated according to Equation (2).

The pseudocode of ACO-MaxIt is presented in Algorithm 5.

**Algorithm 5** ACO-MaxIt algorithm

```
Input: parameters: $m, n, \alpha, \beta, \rho, Q, g, m_1, m_2, N_{\max}$
 1: Initialize pheromone and heuristic information
 2: for iteration index $N_c \leq N_{\max}$ do
 3:   for $k = 1$ to $m$ do
 4:     Construct paths according to Equation (1)
 5:   endfor
 6:   if $N_c \leq gN_{\max}$ then
 7:     Construct opposite paths according to Algorithm 4
 8:     Update pheromone according to Algorithm 2
 9:   else
10:     Update pheromone according to Equation (2)
11:   endif
12: endfor
Output: the optimal solution
```

## 3.3. ACO-Rand

In the pheromone updating stage of ACO-Index or ACO-MaxIt, it is decided based on experiences of when to calculate the opposite paths. Therefore, in this section, another strategy to update pheromones is addressed, and the novel ACO algorithm is named ACO-Rand since whether or not to construct the opposite path is decided by two random variables.

The whole procedure of ACO-Rand is much like that of ACO-MaxIt; however, two random variables $R_0$ and $R$ are introduced. $R_0$ is chosen randomly but fixed after generated, and $R$ is randomly selected during each iteration. The pseudocode of ACO-Rand is given in Algorithm 6.

**Algorithm 6** ACO-Rand algorithm

```
Input: parameters: $m, n, \alpha, \beta, \rho, Q, m_1, m_2, N_{\max}, R_0$
 1: Initialize pheromone and heuristic information
 2: for iteration index $N_c \leq N_{\max}$ do
 3:   for $k = 1$ to $m$ do
 4:     Construct paths according to Equation (1)
 5:   endfor
 6:   Generate a random variable $R$
 7:   if $R < R_0$ then
 8:     Construct opposite paths according to Algorithm 4
 9:     Update pheromone according to Algorithm 2
10:   else
11:     Update pheromone according to Equation (2)
12:   endif
13: endfor
Output: the optimal solution
```

## 3.4. Time Complexity Analysis

The main steps of the three improved ant colony algorithms include initialization, solution construction and pheromone updating. The time complexity of initialization is $O(n^2 + m)$. The time complexity of constructing the solution is $O(mn^2)$. The time complexity of pheromone updating is $O(n^2)$. In addition, the time complexity of constructing and sorting the inverse solutions is $O(n^2)$. Therefore, the complexity of the final algorithm is $O(N_{max}mn^2)$. It is the same time complexity as the basic ant colony algorithm. Therefore, the improved algorithm does not increase significantly in time.

## 4. Experiments and Results

AS and PS-ACO are employed as ACO algorithms to verify the feasibility of three opposition-based ACO algorithms. The experiments were performed in the following hardware and software environments. CPU is Core i5@2.9 GB, and RAM is 16 GB. The operating system is Windows 10. TSP examples are exported from TSPLIB (http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/).

### *4.1. Parameter Setting*

In the following experiments, the parameters are setting as $m = 50$, $m_1 = 40$, $m_2 = 10$, $\alpha = 1$, $\beta = 2$, $\rho$=0.05, $Q$=1, $N_{\max} = 2000$, $g = 0.5$ for ACO-MaxIt, $R_0 = 0.6$ while $R \in (0, 1)$ for ACO-Rand. Twenty cycles of experiments are carried out for each example independently. Then, minimum solution $S_{\min}$, maximum solution $S_{\max}$, average solution $S_{avg}$, standard deviation $S_{td}$, and average runtime $T_{avg}$ for different examples of 20 times are given in the tables, where minimum solution $S_{\min}$, maximum solution $S_{\max}$, and average solution $S_{avg}$ are the percentage value deviation against the known optimal solution. The minimum value in each result is bolded in the tables.

### *4.2. Experimental Results Comparison Based on AS*

First, we employ AS to three kinds of opposite based ACO, called AS-Index, AS-MaxIt, and AS-Rand, to verify the effectiveness of the improved algorithm. Twenty-six TSP examples are divided into three main categories, the small-scale, the medium-scale, and the large-scale according to the number of cities, respectively.

Small-scale city example sets are selected from TSPLIB, including eil51, st70, pr76, kroA100, eil101, bier127, pr136, pr152, u159, and rat195. The results are shown in Table 1.

From Table 1, the proposed AS-Index, AS-MaxIt, and AS-Rand show superior performances over AS for the examples, st70, kroA100, eil101, bier127, pr136, and u159. For other examples, the proposed algorithms outperform AS in general, except eil51. Meanwhile, stability by standard deviation is the not the primary concern when evaluating an algorithm. Compared among three proposed algorithms, AS-MaxIt illustrates superior performances for most cases.

To show more details in the process of evolutionary, curves for different stages are given in Figure 1 based on the case of kroA100.

According to Figure 1, AS shows faster convergence speed than the other three proposed methods in early iterations, while AS-Index, AS-MaxIt, and AS-Rand all surpass AS in average path length in later iterations. Meanwhile, AS-MaxIt performs best among all these algorithms, which also verifies the results in Table 1.

In the early stage, opposite path information introduced by OBL has negative impact on the convergence speed for all three proposed algorithms; however, it can provide extra information which guarantees the boost in accuracy for the later stage. The results lie in the fact that introducing extra information of opposite paths help to increase the diversity of the population, which balances the exploration and exploitation of solution space.

Medium-scale city example sets are selected from TSPLIB, including kroA200, ts225, tsp225, pr226, pr299, lin318, fl417, pr439, pcb442, and d493. The results are shown in Table 2.

From Table 2, it can be found that the proposed algorithms outperform AS in all the cases except ts225. Among all three algorithms, AS-Index and AS-MaxIt perform similarly, but better than AS-Rand generally. From these results, it can be seen that, with the help of extra information from opposite paths, three proposed methods all improve the original AS in solution accuracy.

**Table 1.** Results comparison for small-scale example sets.

| Instance | Algorithm | $S_{\min}(\%)$ | $S_{\max}(\%)$ | $S_{avg}(\%)$ | $S_{td}$ | $T_{avg}$ |
|---|---|---|---|---|---|---|
| eil51 | AS | **2.58** | 6.1 | **3.56** | 4.94 | 65.21 |
| | AS-Index | 2.81 | 7.51 | 4.27 | 5.26 | 64.73 |
| | AS-MaxIt | **2.58** | 7.04 | 4.25 | 6.66 | **63.1** |
| | AS-Rand | 2.82 | **5.63** | 3.86 | **4.67** | 63.76 |
| st70 | AS | 5.03 | 7.41 | 6.22 | 5.16 | 90.38 |
| | AS-Index | **3.85** | 7.56 | 6.03 | 7.05 | 104.02 |
| | AS-MaxIt | 4.59 | **6.81** | **5.83** | **4.63** | 89.87 |
| | AS-Rand | 4.59 | 8.15 | 6.25 | 5.45 | **85.29** |
| pr76 | AS | 6.03 | 9.11 | 7.49 | **897.25** | 101.6 |
| | AS-Index | 6.22 | 9.83 | 7.81 | 926.53 | 118.32 |
| | AS-MaxIt | 5.04 | **8.46** | **6.66** | 1151.2 | 95.9 |
| | AS-Rand | **4.71** | 8.78 | 7.18 | 1336 | **95.72** |
| kroA100 | AS | 4.86 | 6.78 | 5.32 | 94.22 | 146.29 |
| | AS-Index | **4.29** | 6.36 | 4.93 | 111.14 | 163.71 |
| | AS-MaxIt | 4.35 | **5.66** | **4.79** | 80.21 | **123.84** |
| | AS-Rand | 4.53 | 5.8 | 5.11 | **56.40** | 124.99 |
| eil101 | AS | 8.11 | 12.1 | 9.99 | **5.78** | 140.25 |
| | AS-Index | **6.2** | **10.81** | **8.55** | 9.55 | 140.2 |
| | AS-MaxIt | 7.15 | 11.8 | 9.3 | 6.86 | **128.16** |
| | AS-Rand | 7.15 | 12.4 | 9.96 | 8.23 | 145.56 |
| bier127 | AS | 4.75 | 7.15 | 6.05 | **828.19** | 195.33 |
| | AS-Index | 4.07 | **6.75** | 5.32 | 870.88 | 176.32 |
| | AS-MaxIt | **3.34** | 6.82 | **5.03** | 904.32 | **173.58** |
| | AS-Rand | 3.52 | 6.8 | 5.05 | 961.25 | 175.14 |
| pr136 | AS | 9.83 | 13.5 | 11.82 | 924.44 | **189.68** |
| | AS-Index | 9.64 | 12.47 | 11.47 | **832.93** | 209.51 |
| | AS-MaxIt | **8.13** | **12.34** | **10.73** | 976.21 | 190.95 |
| | AS-Rand | 10.68 | 12.62 | 10.95 | 890.29 | 191.23 |
| pr152 | AS | 4.3 | 7.34 | 5.79 | 552.91 | 214.30 |
| | AS-Index | 3.56 | 8.03 | 6.25 | 804.54 | 238.72 |
| | AS-MaxIt | **3.49** | **6.83** | 5.33 | 739.76 | **1982** |
| | AS-Rand | 4.16 | 6.95 | **5.14** | **537.49** | 220.52 |
| u159 | AS | 7.67 | 10.3 | 6.86 | **348.22** | 219.92 |
| | AS-Index | 6.31 | 8.97 | 7.44 | 349.95 | 223.39 |
| | AS-MaxIt | **3.85** | **8.32** | **6.28** | 479.93 | **206.23** |
| | AS-Rand | 4.88 | 8.55 | 7.25 | 412.44 | 229.64 |
| rat195 | AS | 3.92 | 9.38 | 7.43 | 35.28 | 286.38 |
| | AS-Index | **3.49** | 8.52 | 6.47 | 37.02 | 283.31 |
| | AS-MaxIt | 3.83 | 7.58 | 5.59 | 37.69 | **278.63** |
| | AS-Rand | 4.00 | **6.54** | **5.31** | **17.01** | 280.39 |

Taking fl417 as the example, evolutionary curves in detail for different iteration stages are given in Figure 2, accordingly. According to Figure 2, AS also converges faster than the other three proposed methods in early iterations—for example before 1000 iterations. In addition, in later iterations, the other three proposed methods all exceed AS in average path length. This further validates the conclusions obtained from the analysis of Figure 1.

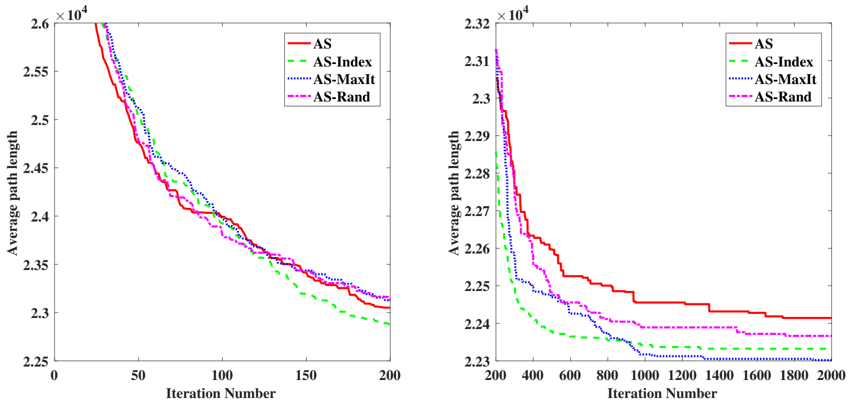

**Figure 1.** Evolutionary curves for different iteration periods based on kroA100.

**Table 2.** Results comparison for medium-scale example sets.

| Instance | Algorithm | $S_{\min}$(%) | $S_{\max}$(%) | $S_{avg}$(%) | $S_{td}$ | $T_{avg}$ |
|---|---|---|---|---|---|---|
| kroA200 | AS | 10.33 | 16.54 | 12.36 | 454.71 | 330.43 |
| | AS-Index | 10.17 | 13.06 | 11.12 | **243.66** | 324.495 |
| | AS-MaxIt | 8.09 | 13.48 | 11.27 | 367.58 | **270.07** |
| | AS-Rand | **6.69** | **13.01** | **10.66** | 485.83 | 292.98 |
| ts225 | AS | 3.53 | **4.27** | **3.89** | **275.55** | 345.62 |
| | AS-Index | **3.13** | 4.74 | 3.91 | 488.06 | 329.03 |
| | AS-MaxIt | 3.35 | 4.94 | 4.01 | 560.16 | 329.35 |
| | AS-Rand | 3.63 | 5.52 | 4.23 | 607.16 | **325.54** |
| tsp225 | AS | 9.4 | **12.03** | 10.37 | **30.67** | 346.81 |
| | AS-Index | 8.02 | 12.16 | **9.9** | 38.02 | 345.35 |
| | AS-MaxIt | **7.51** | 12.87 | 10.91 | 46.8 | **319.75** |
| | AS-Rand | 9.5 | 12.39 | 10.88 | 36.13 | 355.0 |
| pr226 | AS | 5.5 | 7.94 | 6.76 | 632.0 | 327.3 |
| | AS-Index | 4.96 | 7.28 | 6.47 | **458.47** | 348.97 |
| | AS-MaxIt | **4.29** | **7.24** | 6.34 | 631.53 | 335.54 |
| | AS-Rand | 4.39 | 7.5 | **5.92** | 578.15 | **325.31** |
| pr299 | AS | 13.31 | 19.48 | 17.03 | 732.30 | 512.6 |
| | AS-Index | **9.43** | **17.41** | **14.95** | 1193.8 | 495.6 |
| | AS-MaxIt | 15.53 | 18.24 | 16.91 | **419.72** | **487.8** |
| | AS-Rand | 11.35 | 18.73 | 16.41 | 836.03 | 500.35 |
| lin318 | AS | 12.57 | 17.15 | 15.33 | 517.72 | **508.6** |
| | AS-Index | **11.8** | 17.16 | **14.97** | 544.18 | 554.6 |
| | AS-MaxIt | 13.93 | **16.89** | 15.59 | **376.06** | 542.2 |
| | AS-Rand | 14.28 | 17.74 | 16.22 | 415.81 | 542.55 |
| fl417 | AS | 8.16 | 12.55 | 10.61 | 132.22 | 811.45 |
| | AS-Index | **7.79** | 12.57 | **9.91** | 141.28 | 804.6 |
| | AS-MaxIt | 8.35 | **11.55** | 10.3 | **108.13** | **773** |
| | AS-Rand | 8.67 | 13.24 | 10.39 | 146.72 | 795.35 |
| pr439 | AS | 9.5 | 13.74 | 11.56 | 1410.4 | 881.25 |
| | AS-Index | **8.38** | **11.75** | **10.22** | **1010.9** | 845.75 |
| | AS-MaxIt | 9.34 | 15.96 | 11.8 | 1555.1 | **841.9** |
| | AS-Rand | 11.73 | 15.91 | 13.76 | 1205.6 | 859.8 |
| pcb442 | AS | 13.57 | 18.87 | 16.84 | **610.98** | 933.5 |
| | AS-Index | **11.62** | **16.37** | **14.22** | 640.69 | 930.7 |
| | AS-MaxIt | 13.68 | 19.27 | 16.66 | 656.89 | 899.35 |
| | AS-Rand | 14.54 | 18.85 | 16.68 | 644.38 | **888.45** |
| d493 | AS | 13.4 | 19.18 | 16.26 | 459.03 | 1032.25 |
| | AS-Index | **12.23** | **16.31** | **14.71** | 412.28 | 1043.25 |
| | AS-MaxIt | 14.08 | 17.01 | 15.7 | **280.32** | **969.5** |
| | AS-Rand | 13.46 | 19.21 | 16.4 | 449.97 | 1063.8 |

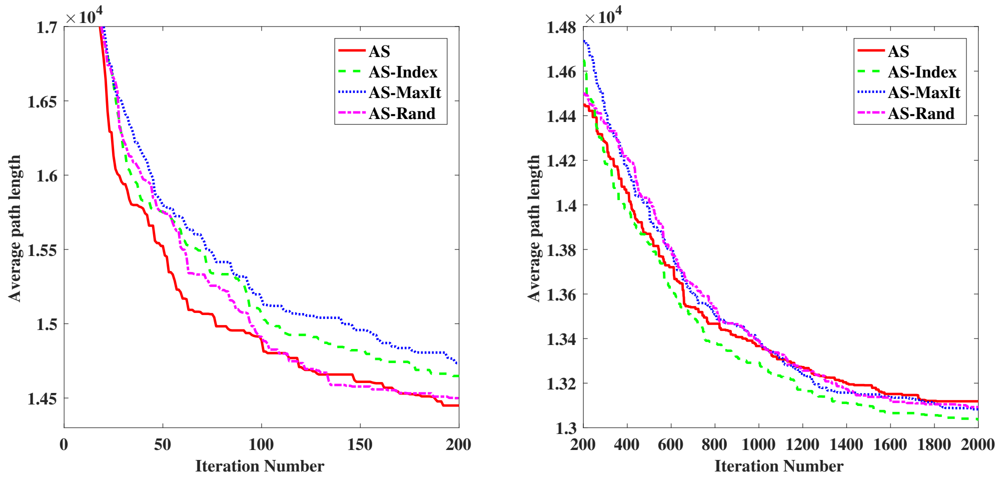

**Figure 2.** Evolutionary curves for different iteration periods based on fl417.

Large-scale city example sets are selected from TSPLIB, including att532, rat575, d657, u724, vm1084, and rl1304. The results are shown in Table 3.

From Table 3, it can be discovered that the AS-Index shows the obvious superior performance over all the other algorithms, which reveals a fact that the advantages of AS-Index appears as the scale of the example increases based on these results.

Taking vm1084 as the example, evolutionary curves in detail for different iteration stages are given in Figure 3, accordingly. According to Figure 3, AS still shows faster convergence speed than the other three proposed methods in early iterations, but AS-Index outperforms all the others in the end.

Based on all the tables and figures, it can be found that, in most scenarios, at least one of AS-Index, AS-MaxIt, and AS-Rand outperforms AS in average path length. For small-scale examples, AS-MaxIt shows better performance, while, for medium-scale cases, AS-Index and AS-MaxIt perform similarly better than the others. For large-scale city sets, AS-Index is the best algorithm, while AS-Rand ranks in the middle for most cases regarding figures, and it illustrates its stability to some extent. Therefore, it can be drawn that the strategy to introduce OBL into AS provides more information, namely better exploration capability, which explains the superiority of these proposed methods over the original AS. By comparing the results of the running time from Tables 1–3, we can also find that the running time of the three improved algorithms is not significantly increased compared with AS. It also validates our previous discussion on time complexity.

## *4.3. Experimental Results Comparison Based on PS-ACO*

To further verify the effectiveness of the proposed algorithm, we employed another PS-ACO to three kinds of opposite based ACO, PS-ACO-Index, PS-ACO-MaxIt, and PS-ACO-Rand to verify the effectiveness of the improved algorithm. The number of ants is 50, and the other parameters are the same as in [26]. Twelve sets of TSP examples are eil51, st70, kroA100, pr136, u159, rat195, tsp225, pr299, lin318, fl417, att532, and d657. The results are given in Table 4.

From Table 4, the proposed PS-ACO-Index, PS-ACO-MaxIt, and PS-ACO-Rand show superior performances over PS-ACO for the examples, eil51, st70, rat195, tsp225, and pr299. For other examples, the proposed algorithms outperform PS-ACO in general, except lin318 and fl417. Compared among three proposed algorithms, PS-ACO-Rand illustrates superior performances for most cases. By comparing the results of the running time, we can also find that the running time of the three improved algorithms is not significantly increased compared with PS-ACO.

**Table 3.** Results comparison for large-scale example sets.

| Instance | Algorithm | $S_{\min}(\%)$ | $S_{\max}(\%)$ | $S_{avg}(\%)$ | $S_{td}$ | $T_{avg}$ |
|---|---|---|---|---|---|---|
| att532 | AS | 13.79 | 20.21 | 17.3 | 1415.4 | 1407.05 |
| | AS-Index | **13.37** | 19.49 | **15.64** | 1203.4 | 1436.4 |
| | AS-MaxIt | 14.33 | 18 | 16.15 | **766.9** | **1384.65** |
| | AS-Rand | 14.63 | **17.98** | 16.34 | 803.93 | 1460.35 |
| rat575 | AS | 18.69 | 22.52 | 20.5.3 | 75.22 | 1651.5 |
| | AS-Index | **16.31** | **20.71** | **18.94** | **61.3** | 1661.7 |
| | AS-MaxIt | 20.24 | 24.32 | 22.23 | 80.88 | 1602.55 |
| | AS-Rand | 22.88 | 26.93 | 25.36 | 86.12 | **1598** |
| d657 | AS | 17.6 | 23.88 | 21.97 | 823.36 | 2685.45 |
| | AS-Index | **16.32** | **21.5** | **19.7** | 636.0 | 2673.65 |
| | AS-MaxIt | 19.4 | 24.09 | 21.69 | **526.44** | **2114.15** |
| | AS-Rand | 18.21 | 23.82 | 21.24 | 615.17 | 2138.1 |
| u724 | AS | 20.9 | 26.43 | 24.19 | 625.6 | 2630.95 |
| | AS-Index | **15.53** | **23.45** | **20.46** | 871.17 | 2624.7 |
| | AS-MaxIt | 21.6 | 26.72 | 24.05 | **469.3** | 2651.65 |
| | AS-Rand | 19.74 | 27.13 | 24.3 | 825.26 | **2609.7** |
| vm1084 | AS | 22.65 | 27.78 | 25.76 | 3490.5 | 6722 |
| | AS-Index | **17.69** | **24.73** | **21.1** | 4700 | 6726.5 |
| | AS-MaxIt | 19.91 | 26.59 | 23.29 | 4476.6 | **6594** |
| | AS-Rand | 20.72 | 25.71 | 23.04 | **3111** | 6695.5 |
| rl1304 | AS | 19.51 | 24.37 | 21.76 | 3633.6 | 7436.5 |
| | AS-Index | **16.65** | **21.63** | **18.95** | **3632.3** | **7383** |
| | AS-MaxIt | 18.45 | 24.19 | 20.63 | 3709.8 | 8767.5 |
| | AS-Rand | 17.09 | 22.82 | 20.12 | 4359.2 | 8652.5 |

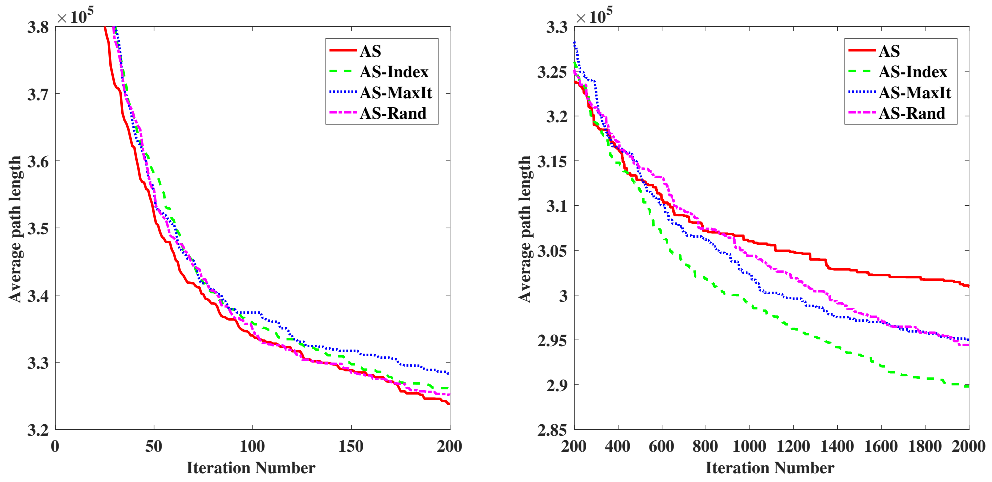

**Figure 3.** Evolutionary curves for different iteration periods based on vm1084.

**Table 4.** Comparisons of PS-ACO, PS-ACO-Index, PS-ACO-MaxIt, and PS-ACO-Rand.

| Instance | PS-ACO | | | | | PS-ACO-Index | | | | | PS-ACO-MaxIt | | | | | PS-ACO-Rand | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $S_{\min}$(%) | $S_{\max}$(%) | $S_{avg}$(%) | $S_{td}$ | $T_{avg}$ | $S_{\min}$(%) | $S_{\max}$(%) | $S_{avg}$(%) | $S_{td}$ | $T_{avg}$ | $S_{\min}$(%) | $S_{\max}$(%) | $S_{avg}$(%) | $S_{td}$ | $T_{avg}$ | $S_{\min}$(%) | $S_{\max}$(%) | $S_{avg}$(%) | $S_{td}$ | $T_{avg}$ |
| eil51 | 0 | 0.7 | 0.52 | 0.89 | 77.4 | 0 | 0.7 | 0.42 | 1.15 | 81.59 | 0 | 0.7 | 0.46 | 1.15 | 78.06 | 0 | 0.7 | 0.46 | 0.94 | 78.58 |
| st70 | 0.15 | 2.67 | 1.07 | 5.84 | 99.35 | 0 | 2.67 | 0.88 | 4.67 | 121.44 | 0.15 | 2.96 | 0.93 | 5.61 | 99.31 | 0.15 | 692 | 2.52 | 3.79 | 99.02 |
| kroA100 | 0.06 | 1.16 | 0.54 | 58.85 | 163.6 | 0.12 | 2.2 | 0.72 | 107.74 | 176.34 | 0.06 | 1 | 0.55 | 59.94 | 151.06 | 0 | 1.11 | 0.54 | 79.47 | 146.32 |
| pr136 | 7.64 | 12.2 | 10.08 | 1200 | 206.41 | 7.98 | 13.12 | 10.19 | 1452.7 | 230.68 | 8.03 | 11.97 | 9.96 | 1116.4 | 203.95 | 7.48 | 12.51 | 9.81 | 1274.4 | 204.24 |
| u159 | 0 | 0.88 | 0.12 | 112.74 | 886.25 | 0 | 0.88 | 0.17 | 117.7 | 875.5 | 0 | 0.22 | 0.02 | 28.93 | 860.8 | 0 | 0.75 | 0.09 | 98.07 | 868.45 |
| rat195 | 0.43 | 1.33 | 0.7 | 5.86 | 882.25 | 0.43 | 1.21 | 0.6 | 4.24 | 875.65 | 0.43 | 1.33 | 0.6 | 6.45 | 907.85 | 0.43 | 0.99 | 0.57 | 3.88 | 906.4 |
| tsp225 | 3.7 | 5.98 | 4.93 | 23.78 | 1087.35 | 3.5 | 5.36 | 4.21 | 21.14 | 1140.75 | 3.73 | 5.87 | 4.66 | 25.18 | 1158.85 | 3.52 | 5.57 | 4.76 | 20.51 | 1150.8 |
| pr299 | 9.78 | 14.08 | 11.68 | 605.79 | 1927.8 | 9.03 | 12.78 | 10.63 | 450.92 | 1919.15 | 7.8 | 14.74 | 11.14 | 885.13 | 1882.4 | 9.17 | 14.49 | 11.6 | 792.39 | 1881.45 |
| lin318 | 10.29 | 15.59 | 12.53 | 523.89 | 571.9 | 11.42 | 15.14 | 13.16 | 417.59 | 651.4 | 9.66 | 15.64 | 12.85 | 600.33 | 608.75 | 10.64 | 15.71 | 13.52 | 498.17 | 607.8 |
| fl417 | 10.51 | 14.56 | 12.43 | 155.86 | 782.6 | 11.1 | 17.45 | 14.7 | 226.592 | 778.95 | 11.45 | 17.77 | 14.17 | 216.73 | 714.3 | 10.35 | 17.56 | 14.45 | 232.11 | 724.6 |
| att532 | 19.9 | 25.06 | 22.82 | 1106.8 | 1518.45 | 19.58 | 25.26 | 22.42 | 1372.8 | 1529.25 | 20.02 | 25.3 | 22.3 | 1113.1 | 1342.4 | 20.59 | 26.17 | 23.25 | 1114 | 1363.95 |
| d657 | 22.67 | 27.06 | 24.66 | 534.7 | 2247.4 | 21.04 | 27.91 | 24.57 | 780.93 | 2202.4 | 22.08 | 25.74 | 24.32 | 547.2 | 2002.5 | 23.17 | 28.16 | 25.31 | 697.21 | 2175.95 |

## 5. Conclusions

The performances of swarm optimization algorithms based on OBL present advantages when handling problems of continuous optimization. However, there are only a few approaches proposed to solve problems of discrete optimization. The difficulty in opposite solution construction is considered as one top reason. To solve this problem, two different strategies, direction and indirection, of constructing opposite paths are presented individually in this paper. For indirection strategy, other than using the order of cities from the current solution directly, it studies the positions, noted as indices, of the cities rearranged in a circle, and then calculates the opposite indices. While for direction strategy, opposite operations are carried out directly to the cities in each path.

To use the information of the opposite path, three different frameworks of opposite-based ACO, called ACO-Index, ACO-MaxIt, and ACO-Rand, are also proposed. All ants need to get the increment of pheromone in three improved frameworks. Among three proposed algorithms, ACO-Index employs the strategy of indirection to construct the opposite path and introduces it to pheromone updating. ACO-MaxIt also employs direction strategy to obtain opposite path but only adopts it in the early updating period. Similar to ACO-MaxIt in opposite path construction, ACO-Rand employs this opposite path throughout the stage of pheromone updating. In order to verify the effectiveness of the improvement strategy, AS and PS-ACO are used in three frameworks, respectively. Experiments demonstrate that all three methods, As-Index, As-MaxIt, and AS-Rand, outperform original AS in the cases of small-scale and medium-scale cities while AS-Index performs best when facing large-scale cities. The three improved PS-ACO also showed good performance.

Constructing the opposite path mentioned in this paper is only suitable for symmetric TSP. This is mainly because the path (solution) of the problem is an arrangement without considering the direction. However, if it is replaced by the asymmetric TSP, this method needs to be modified. In addition, if it is replaced by a more general combinatorial optimization problem, it is necessary to restudy how to construct the opposite solution according to the characteristics of the problem. Therefore, our current method of constructing opposite solution is not universal. This is one of the limitations of this study. At the same time, the improved algorithm requires all ants to participate in pheromone updating in order to use the information of opposite path. However, now many algorithms use the best ant to update pheromone, so the method in this paper will have some limitations when it is extended to more ant colony algorithm. However, we also find that it is effective to apply reverse learning to combinatorial optimization problems. Therefore, we will carry out our future research work from two aspects. On the one hand, we plan to continue to study the construction method of more general opposite solution for combinatorial optimization problems, so as to improve its generality. In addition, it will be applied to practical problems such as path optimization to further expand the scope of application. Meanwhile, applying OBL to more widely used algorithms is also one interesting and promising topic. Therefore, on the other hand, we plan to study more effective use of the reverse solution and extend it to the more wildly used ACO, such as MMAS and ACS, and even some other optimization algorithms such as PSO and ABC, to solve more combinatorial optimization problems more effectively.

**Author Contributions:** Conceptualization, Z.Z.; methodology, Z.Z.and Z.X.; software, Z.X. and X.L.; formal analysis, Z.Z.and Z.X.; resources, Z.Z.; writing—original draft preparation, Z.X.; writing—review and editing, Z.Z.and S.L.; supervision, Z.Z.and Y.S. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was supported by the National Natural Science Foundation of China (Grant No. 61703256, 61801197), Jiangsu Natural Science Foundation (Grant No. BK20181004), Natural Science Basic Research Plan In Shaanxi Province of China (Program No. 2017JQ6070), and the Fundamental Research Funds for the Central Universities (Grant No. GK201603014, GK201803020).

**Acknowledgments:** The authors are grateful to the anonymous reviewers and the editor for the constructive comments and valuable suggestions.

**Conflicts of Interest:** The authors declare no conflict of interest.The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

## References

1. Karaboga, D.; Akay, B. A survey: Algorithms simulating bee swarm intelligence. *Artif. Intell. Rev.* **2009**, *31*, 61–85.
2. Dorigo,M.; Maniezzo,V.; Colorni,A. The ant system: Optimization by a colony of cooperating agents. *IEEE Trans. Syst. Man Cybern. Part B* **1996**, *26*, 29–41. [CrossRef]
3. Kennedy, J.; Eberhart, R. Particle swarm optimization. In Proceedings of the ICNN'95-International Conference on Neural Networks, Perth, Australia, 27 November–1 December 1995; pp. 1942–1948.
4. Karaboga, D.; Akay, B. A comparative study of artificial bee colony algorithm. *Appl. Math. Comput.* **2009**, *214*, 108–132. [CrossRef]
5. Yang, X.S. Firefly algorithm, stochastic test functions and design optimisation. *Int. J. Bio-Inspired Comput.* **2010**, *2*, 78–84. [CrossRef]
6. Gandomi, A.H.; Yang, X.S.; Alavi, A.H. Cuckoo search algorithm: A metaheuristic approach to solve structural optimization problems. *Eng. Comput.* **2013**, *29*, 17–35. [CrossRef]
7. Wang, G.G.; Guo, L.G.; omi, A.H.; Hao, G.S.; Wang, H. Chaotic krill herd algorithm. *Inf. Sci.* **2014**, *274*, 17–34. [CrossRef]
8. Wang, G.G.; Deb, S.; Cui, Z. Monarch butterfly optimization. *Neural Comput. Appl.* **2019**, *31*, 1995–2014. [CrossRef]
9. Wang, G.G. Moth search algorithm: A bio-inspired metaheuristic algorithm for global optimization problems. *Memet. Comput.* **2018**, *10*, 151–164.
10. Mollajafari, M.; Shahhoseini, H.S. An efficient ACO-based algorithm for scheduling tasks onto dynamically reconfigurable hardware using TSP-likened construction graph. *Appl. Intell.* **2016**, *45*, 695–712. [CrossRef]
11. Elloumi, W.; El Abed, H.; Abraham, A.; Alimi, A.M. A comparative study of the improvement of performance using a PSO modified by ACO applied to TSP. *Appl. Soft Comput.* **2014**, *25*, 234–241. [CrossRef]
12. Zhang, Z.Z.; Hu, F.N.; Zhang, N.; Ant colony algorithm for satellite control resource scheduling problem. *Appl. Intell.* **2018**, *48*, 3295–3305. [CrossRef]
13. Rahim, S.; Javaid, N.; Ahmad, A.; Khan, S.A.; Khan, Z.A.; Alrajeh, N.; Qasim, U. Exploiting heuristic algorithms to efficiently utilize energy management controllers with renewable energy sources. *Energy Build.* **2016**, *129*, 452–470. [CrossRef]
14. Bhattacharjee, K.K.; Sarmah, S.P. Modified swarm intelligence based techniques for the knapsack problem. *Appl. Intell.* **2017**, *46*, 158–179. [CrossRef]
15. Huang, S.H.; Huang, Y.H.; Blazquez, C.A.; Paredes-Belmar, G. Application of the ant colony optimization in the resolution of the bridge inspection routing problem. *Appl. Soft Comput.* **2018**, *65*, 443–461. [CrossRef]
16. Lee, C.Y.; Lee, Z.J.; Lin, S.W.; Ying, K.C. An enhanced ant colony optimization (EACO) applied to capacitated vehicle routing problem. *Appl. Intell.* **2010**, *32*, 88–95. [CrossRef]
17. Kumar, A.; Thakur, M.; Mittal, G. A new ants interaction scheme for continuous optimization problems. *Int. J. Syst. Assur. Eng. Manag.* **2018**, *9*, 784–801. [CrossRef]
18. Yang, Q.; Chen, W.N.; Yu, Z.; Gu, T.; Li, Y.; Zhang, H.; Zhang, J. Adaptive multimodal continuous ant colony optimization. *IEEE Trans. Evol. Comput.* **2017**, *21*, 191–205. [CrossRef]
19. Liao, T.J.; Stützle, T.; Oca, M.A.M.; Dorigo, M. A unified ant colony optimization algorithm for continuous optimization. *Eur. J. Oper. Res.* **2014**, *234*, 597–609. [CrossRef]
20. Dorigo, M.; Blum, C. Ant colony optimization theory: A survey. *Theor. Comput. Sci.* **2005**, *344*, 243–278. [CrossRef]
21. Dorigo, M.; Gambardella, L.M. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Trans. Evol. Comput.* **1997**, *1*, 53–66. [CrossRef]
22. Stützle, T.; Hoos, H.H. Max-min ant system. *Future Gener. Comput. Syst.* **2000**, *16*, 889–914.
23. Luo, Q.; Wang, H.; Zheng, Y.; He, J. Research on path planning of mobile robot based on improved ant colony algorithm. *Future Gener. Comput. Syst.* **2020**, *32*, 1555–1566. [CrossRef]
24. Huang, M.; Ding, P. An improved ant colony algorithm and its application in vehicle routing problem. *Future Gener. Comput. Syst.* **2013**, *2013*, 1–9.
25. Deng, Y.; Zhu, W.; Li, H.; Zheng, Y.H. Multi-type ant system algorithm for the time dependent vehicle routing problem with time windows. *J. Syst. Eng. Electron.* **2018**, *29*, 625–638.
26. Shuang, B.; Chen, J.; Li, Z. Study on hybrid PS-ACO algorithm. *Appl. Intell.* **2011**, *34*, 64–73. [CrossRef]

27. Ke, L.J.; Zhang, Q.F.; Battiti, R. MOEA/D-ACO: A multiobjective evolutionary algorithm using decomposition and ant colony. *IEEE Trans. Cybern.* **2013**, *43*, 1845–1859.
28. Akpınar, S.; Bayhan, G.M.; Baykasoglu, A. Hybridizing ant colony optimization via genetic algorithm for mixed-model assembly line balancing problem with sequence dependent setup times between tasks. *Appl. Soft Comput.* **2013**, *13*, 574–589.
29. Ting, T.O.; Yang, X.S.; Cheng, S.; Huang, K. Hybrid metaheuristic algorithms: past, present, and future. In *Recent Advances in Swarm Intelligence and Evolutionary Computation*; Yang, X.S., Ed.; Springer International Publishing: Cham, Switzerland, 2015; pp. 71–83.
30. Rosa, G.H.D.; Papa, J.P.; Yang, X.S. Handling dropout probability estimation in convolution neural networks using meta-heuristics. *Soft Comput.* **2018**, *22*, 6147–6156. [CrossRef]
31. Bacanin, N.; Bezdan, T.; Tuba, E.; Strumberger, I.; Tuba, M. Monarch butterfly optimization based convolutional neural network design. *Mathematics* **2020**, *8*, 936. [CrossRef]
32. Wang, G.G.; Tan, Y. Improving metaheuristic algorithms with information feedback models. *IEEE Trans. Cybern.* **2019**, *49*, 542–555. [CrossRef]
33. Gao, D.; Wang, G.G.; Pedrycz, W. Solving fuzzy job-shop scheduling problem using DE algorithm improved by a selection mechanism. *IEEE Trans. Fuzzy Syst.* **2020**. [CrossRef]
34. Li, W.; Wang, G.G.; Alavi, A.H. Learning-based elephant herding optimization algorithm for solving numerical optimization problems. *Knowl. Based Syst.* **2020**, *195*, 105675. [CrossRef]
35. Mahdavi, S.; Rahnamayan, S.; Deb, K. Opposition based learning: A literature review. *Swarm Evol. Comput.* **2017**, *39*, 1–23. [CrossRef]
36. Wang, B. A novel artificial bee colony algorithm based on modified search strategy and generalized opposition-based learning. *J. Intell. Fuzzy Syst.* **2015**, *28*, 1023–1037. [CrossRef]
37. Rahnamayan, S.; Tizhoosh, H.R.; Salama, M.M.A. Opposition-based differential evolution. *IEEE Trans. Evol. Comput.* **2008**, *12*, 64–79. [CrossRef]
38. Chen, J.; Cui, G.; Duan, H. Multipopulation differential evolution algorithm based on the opposition-based learning for heat exchanger network synthesis. *Numer. Heat Transf. Part A Appl.* **2017**, *72*, 126–140. [CrossRef]
39. Park, S.Y.; Lee, J.J. Stochastic opposition-based learning using a beta distribution in differential evolution. *IEEE Trans. Cybern.* **2016**, *46*, 2184–2194. [CrossRef] [PubMed]
40. Dong, W.; Kang, L.; Zhang, W. Opposition-based particle swarm optimization with adaptive mutation strategy. *Soft Comput.* **2017**, *21*, 5081–5090. [CrossRef]
41. Kang, Q.; Xiong, C.; Zhou, M.; Meng, L. Opposition-based hybrid strategy for particle swarm optimization in noisy environments. *IEEE Access* **2018**, *6*, 21888–21900. [CrossRef]
42. Malisia, A.R.; Tizhoosh, H.R. Applying opposition-based ideas to the ant colony system. In Proceedings of the 2007 IEEE Swarm Intelligence Symposium (SIS), Honolulu, HI, USA, 1–5 April 2007; pp. 182–189.
43. Ergezer, M.; Simon, D. Oppositional biogeography-based optimization for combinatorial problems. In Proceedings of the 2011 IEEE Congress of Evolutionary Computation (CEC), New Orleans, LA, USA, 5–8 June 2011; pp. 1496–1503.
44. Srivastava, G.;Singh, A. Boosting an evolution strategy with a preprocessing step: Application to group scheduling problem in directional sensor networks. *Appl. Intell.* **2018**, *48*, 4760-4774. [CrossRef]
45. Venkatesh, P.; Alok, S. A swarm intelligence approach for the colored traveling salesman problem. *Appl. Intell.* **2018**, *48*, 4412–4428.
46. Sarkhel, R.; Das, N.; Saha, A.K.; Nasipuri, M. An improved harmony search algorithm embedded with a novel piecewise opposition based learning algorithm. *Eng. Appl. Artif. Intell.* **2018**, *67*, 317–330. [CrossRef]
47. Wang, H.; Wu, Z.; Rahnamayan, S. Enhanced opposition-based differential evolution for solving high-dimensional continuous optimization problems. *Soft Comput.* **2011**, *15*, 2127–2140. [CrossRef]
48. Guha, D.; Roy, P.K.; Banerjee, S. Load frequency control of large scale power system using quasi-oppositional grey wolf optimization algorithm. *Eng. Sci. Technol. Int. J.* **2016**, *19*, 1693–1713. [CrossRef]
49. Ewees, A.A.; Elaziz, M.A.; Houssein, E.H. Improved grasshopper optimization algorithm using opposition-based learning. *Expert Syst. Appl.* **2018**, *112*, 156–172. [CrossRef]