

Article

An Optimized Discrete Dragonfly Algorithm Tackling the Low Exploitation Problem for Solving TSP

Bibi Aamirah Shafaa Emambucus¹, Muhammed Basheer Jasser^{1,*}, Angela Amphawan¹ and Ali Wagdy Mohamed²

¹ Department of Computing and Information Systems, School of Engineering and Technology, Sunway University, Petaling Jaya 47500, Selangor, Malaysia

² Operations Research Department, Faculty of Graduate Studies for Statistical Research, Cairo University, Giza 12613, Egypt

* Correspondence: basheerj@sunway.edu.my

Abstract: Optimization problems are prevalent in almost all areas and hence optimization algorithms are crucial for a myriad of real-world applications. Deterministic optimization algorithms tend to be computationally costly and time-consuming. Hence, heuristic and metaheuristic algorithms are more favoured as they provide near-optimal solutions in an acceptable amount of time. Swarm intelligence algorithms are being increasingly used for optimization problems owing to their simplicity and good performance. The Dragonfly Algorithm (DA) is one which is inspired by the swarming behaviours of dragonflies, and it has been proven to have a superior performance than other algorithms in multiple applications. Hence, it is worth considering its application to the traveling salesman problem which is a predominant discrete optimization problem. The original DA is only suitable for solving continuous optimization problems and, although there is a binary version of the algorithm, it is not easily adapted for solving discrete optimization problems like TSP. We have previously proposed a discrete adapted DA algorithm suitable for TSP. However, it has low effectiveness, and it has not been used for large TSP problems. In this paper, we propose an optimized discrete adapted DA by using the steepest ascent hill climbing algorithm as a local search. The algorithm is applied to a TSP problem modelling a package delivery system in the Kuala Lumpur area and to benchmark TSP problems, and it is found to have a higher effectiveness than the discrete adapted DA and some other swarm intelligence algorithms. It also has a higher efficiency than the discrete adapted DA.



Citation: Emambucus, B.A.S.; Jasser, M.B.; Amphawan, A.; Mohamed, A.W. An Optimized Discrete Dragonfly Algorithm Tackling the Low Exploitation Problem for Solving TSP. *Mathematics* **2022**, *10*, 3647. <https://doi.org/10.3390/math10193647>

Academic Editors: Humberto Rocha and Ana Maria Rocha

Received: 21 July 2022

Accepted: 7 September 2022

Published: 5 October 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: discrete optimization; dragonfly algorithm; metaheuristics; optimization; swarm intelligence algorithms; traveling salesman problem

MSC: 49M37

1. Introduction

Optimization is crucial in almost every domain where certain processes, designs, or systems are adjusted so as to be the most effective possible. Its aim is to find the best solution among a set of available solutions which either maximizes or minimizes an objective function. Hence, optimization algorithms are usually iterative where the objective function is evaluated repeatedly and the best solution found is chosen. Optimization algorithms can be classified into two types; deterministic algorithms, and heuristic algorithms [1]. Deterministic algorithms consist of exact methods which find the best solution to a problem. However, they are computationally costly and time-consuming, especially for large-scale real-world applications. Conversely, heuristic algorithms try to find a near-optimal solution in a feasible amount of time. Heuristic algorithms can be further developed to produce metaheuristic algorithms which use a combination of diversification and intensification techniques to perform better than simple heuristic algorithms. They can be classified as either trajectory-based or population-based algorithms [1].

Swarm intelligence algorithms are population-based metaheuristic algorithms which are inspired by various biological organisms. In nature, the simple and self-organizing interactions that individuals from a specific biological population have with each other and with their environment cause a functional global pattern to emerge [2]. Swarm intelligence algorithms are inspired by these simple interactions of different groups of animals or swarms of insects. They make use of a population of search agents which replicate the interactions of a biological population for solving complex optimization problems.

The Dragonfly Algorithm (DA) [3] is a swarm intelligence algorithm which is inspired by dragonfly swarms, in particular, their hunting and migrating behaviours. Dragonflies swarm statically and dynamically while hunting and migrating, respectively, and these two types of swarming aptly represent the two requisite phases of optimization algorithms; exploration, and exploitation. DA has been found to have a better performance than multiple other swarm intelligence algorithms in various applications [4]. In [3], three versions of the dragonfly algorithm are proposed, namely, the original continuous DA for solving continuous optimization problems, the binary DA (BDA) to cater for discrete or binary optimization problems, and the multi-objective DA (MODA) to cater for multi-objective optimization problems.

The Traveling Salesman Problem (TSP) is a combinatorial optimization problem with a discrete search space. A myriad of real-world problems can be represented as TSP and hence it has numerous real-world applications. Large TSP problems are difficult to be solved using exact algorithms and the solvable instances consume a significant amount of computational time and resources [5]. Hence, heuristic algorithms are often used to solve TSP by providing near-optimal solutions. A number of swarm intelligence algorithms such as the Particle Swarm Optimization (PSO) [6], and the Ant Colony Optimization (ACO) [7], and their variants [8–11] have been successfully employed for solving TSP.

Considering the good performance of DA and its superior performance over other swarm intelligence algorithms in multiple applications, it is worth applying it for solving TSP. The original DA was proposed to solve continuous optimization problems and although a binary version of the original DA is proposed in [3], it is difficult to be adapted for discrete problems like TSP. Hence, in [12], we proposed a discrete adapted dragonfly algorithm suitable for TSP. This is to propose a new discretized variant of the dragonfly algorithm which is suitable for solving discrete optimization problems like TSP and which can be further improved. However, this algorithm has not been applied to large TSP problems and it has low effectiveness.

In this paper, we propose to improve the low effectiveness of the adapted discrete dragonfly algorithm in [12] by employing the steepest ascent Hill Climbing (HC) algorithm as a local search to improve the exploitation phase. The algorithm is tested using a TSP problem consisting of 50 locations in the area of Kuala Lumpur. The TSP problem models a package delivery system where the shortest route to deliver parcels at specific locations and return to the initial location needs to be found. Moreover, the performance of the algorithm is compared to other swarm intelligence algorithms in solving benchmark TSP problems from TSPLIB. From the experiments conducted, the proposed algorithm is found to have higher effectiveness, that is, it produces solutions with a lower cost as compared to the adapted discrete DA [12], and the enhanced Swap Sequence based PSO (SSPSO) [8] algorithms, and it also has higher effectiveness as compared to some other swarm intelligence algorithms as it provides the optimal solution or close to the optimal solutions for benchmark TSP problems. Moreover, it has a higher efficiency than the adapted discrete DA as it converges to the optimal solution in a shorter amount of time.

The contributions of this paper include an optimized discrete dragonfly algorithm suitable for solving discrete optimization problems such as TSP, which provides optimal or near-optimal solutions for benchmark TSP problems, an application of the proposed algorithm to a TSP problem which models a package delivery system in the area of Kuala Lumpur, and a comparison of the performance of the proposed algorithm to that of other swarm intelligence algorithms in solving benchmark TSP problems.

The remaining of the paper is structured as follows: In Section 2, a background on swarm intelligence algorithms, the original dragonfly algorithm, and the hill climbing algorithm is provided. In Section 3, an explanation on the traveling salesman problem is given. In Section 4, some previous works using swarm intelligence algorithms for solving TSP, and the adapted discrete DA algorithm are presented. In Section 5, a description of the proposed algorithm is given. In Section 6, the results and discussions are presented and, finally, in Section 7, the conclusions and some future work are presented.

2. Background on Swarm Intelligence, Dragonfly Algorithm, and Hill Climbing Algorithm

2.1. Swarm Intelligence Algorithms

Swarm intelligence algorithms are metaheuristic optimization algorithms that are inspired by the simple and self-organizing interactions of biological organisms that give rise to a functional global pattern [2]. They make use of a number of search agents which replicate the actions of individuals in a specific biological population as they interact among themselves and their environment. Each search agent, which represents a solution, moves in the state space by considering a fitness function. This allows the algorithm to solve complex optimization problems. Some well-known swarm intelligence algorithms include the particle swarm optimization that is inspired by the swarming of bird flocks or fish schools, and the ant colony optimization that is based on the food searching behaviour of ants.

Swarm intelligence algorithms have a plethora of applications in various domains as they can be used for solving different types of optimization problems including continuous optimization problems, discrete optimization problems, and multi-objective optimization problems. A continuous optimization problem is one in which the solution can be any real value within a certain range of values whereas a discrete optimization problem is one in which the solution can be a specific one from a set of possible solutions. A multi-objective optimization problem is one which has more than one objective function.

Some recent applications of swarm intelligence algorithms include in agricultural technology drones used for improving the productivity of farming areas [13], in fog computing systems for task scheduling [14], in gene selection profile for the classification of microarray data [15], in feature selection [16], and in artificial neural networks for optimizing the parameters of the network [17,18]. Furthermore, they have various applications in data science [19], in Internet of Things (IoT) systems [20], in surveillance systems [21], in water resources engineering [22], and in supply chain management [23].

2.2. Dragonfly Algorithm

The dragonfly algorithm [3] is a metaheuristic optimization algorithm classified under swarm intelligence algorithms. It is inspired by the swarming behaviours of dragonflies during hunting and migrating. During hunting, the dragonflies swarm statically, that is they form small groups and fly over a small area by abruptly changing their flying path. This type of swarming is congruent with the exploration phase of optimization algorithms where the algorithm tries to find a good region of the search space. Conversely, during migration, the dragonflies fly together in a sole group and along one direction over long distances. This type of swarming is congruent with the exploitation phase of optimization algorithms where the algorithm tries to converge to the optimal solution. Figure 1 shows a static and a dynamic swarm of dragonflies.

Five factors are used to control the movement of the dragonflies in the search space during both the exploration and exploitation phases; separation, alignment, cohesion, attraction to food, and distraction from enemy. Each factor has a corresponding weight which is used to tune the factor to enable the algorithm to transition between the exploration and exploitation phases. The factors, along with the weights, also ensure the survival of the swarm by causing it to attract towards food sources and distract away from enemies.

The best solution which has been obtained by the population of search agents is taken as the food source and the worst solution is taken as the enemy.

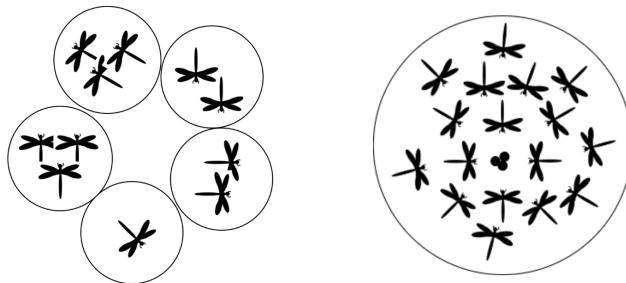


Figure 1. Static and dynamic dragonfly swarms.

The separation factor of a dragonfly is used to prevent its static collision with its neighbours, and is calculated using (1):

$$S_i = - \sum_{j=1}^N X_i - X_j \quad (1)$$

where X_i and X_j are the current dragonfly's position and the j -th neighbour's position respectively, and N is the number of dragonflies in the neighbourhood.

The alignment factor of a dragonfly matches its velocity to that of its neighbours, and is calculated using (2):

$$A_i = \frac{\sum_{j=1}^N V_j}{N} \quad (2)$$

where V_j is the j -th neighbour's velocity and N is the number of dragonflies in the neighbourhood.

The cohesion factor of a dragonfly is its tendency towards the centre of mass of the neighbourhood, and is calculated using (3):

$$C_i = \frac{\sum_{j=1}^N X_j}{N} - X_i \quad (3)$$

where X_j is the j -th neighbour's position and N is the number of dragonflies in the neighbourhood.

The attraction to food factor of a dragonfly is its attraction towards a food source, and is calculated using (4):

$$F_i = X^+ - X_i \quad (4)$$

where X^+ is the food source's position.

The distraction from enemy factor of a dragonfly is its repulsion from an enemy, and is calculated using (5):

$$E_i = X^- + X_i \quad (5)$$

where X^- is the enemy's position.

To allow the dragonflies to move in the search space by considering these factors, two vectors are used: the step vector (ΔX) and the position vector (X).

The step vector determines the direction of the movement, and it is calculated using (6):

$$\Delta X_i^{t+1} = (sS_i + aA_i + cC_i + fF_i + eE_i) + w\Delta X_i^t \quad (6)$$

where S_i , A_i , C_i , F_i , and E_i are the separation, alignment, cohesion, food factor, and enemy factors of the i -th dragonfly respectively, s , a , c , f , and e are the separation, alignment, cohesion, food factor, and enemy factor's weights respectively, w is the inertia weight, and t is the iteration counter.

The position vector allows the movement in the search space, and it is calculated using (7):

$$X_i^{t+1} = X_i^t + \Delta X_i^{t+1} \quad (7)$$

In DA, in order to replicate the static and dynamic swarming behaviours of dragonflies, it is important to consider the neighbourhood of each search agent. This is achieved by considering a radius around each artificial dragonfly. The radius is increased proportionally to the iteration number so as to change the static swarms to dynamic ones until the whole population form one dynamic swarm and converges to the global optimum during the final iterations. This is another way by which the algorithm transitions from the exploration phase to the exploitation phase.

If a dragonfly has no neighbouring dragonfly, its position is updated using the Levy flight mechanism, which is a random walk employed to increase the stochasticity of the algorithm. The position vector of the dragonfly is calculated using (8):

$$X_i^{t+1} = X_i^t + Levy(d) \times X_i^t \quad (8)$$

where t is the current iteration number and d is the dimension of the position vectors.

The pseudocode of DA is given in Algorithm 1.

Algorithm 1: Dragonfly Algorithm

```

1 Initialize the population's positions randomly;
2 Initialize the step vectors;
3 while end condition do
4   Calculate the objective values of all dragonflies;
5   Update the food source and enemy;
6   Update the weights;
7   Calculate the factors using (1)–(5);
8   Update radius of neighbourhoods;
9   if dragonfly has one or more neighbours then
10    | Update step vector using (6);
11    | Update position vector using (7);
12   else
13    | Update position vector using (8);
14   end
15   Check and correct new positions based on upper
      and lower bounds;
16 end
```

2.3. Hill Climbing Algorithm

The hill climbing algorithm is a heuristic local search algorithm which is used for optimization mainly in the field of artificial intelligence. Generally, starting from one position, it considers all the possible neighbouring solutions in a search region and selects the one which either maximizes or minimizes an objective function the most.

There are three main types of hill climbing, namely, the simple hill climbing algorithm, the steepest ascent hill climbing algorithm, and the stochastic hill climbing algorithm.

The simple hill climbing algorithm checks only one neighbouring solution at a time. If it is better than the current solution, the neighbouring solution is selected as the current solution. This type of hill climbing algorithm requires less computing power. However, a good solution is not guaranteed.

The steepest ascent hill climbing algorithm checks all the neighbouring solutions and then selects the best one. This type of hill climbing algorithm requires more computing power, however, it is more likely to find the most optimal solution.

The stochastic hill climbing algorithm checks a random neighbouring solution and if it is better than the current solution, the neighbouring solution is selected as the current solution, otherwise, another random neighbouring solution is selected.

3. Problem Formulation

The traveling salesman problem, a combinatorial optimization problem, is classified as a Non-Deterministic Polynomial-hard (NP-hard) problem as large TSP problems are difficult to solve [5]. It consists of a discrete state space with a finite set of possible solutions and the aim is to find the best solution from the set. TSP can be defined as follows: given a number of inputs called cities and the cost of travel between each possible pair, the aim is to find the route with the least cost to visit each city exactly once and to return to the starting city.

TSP has numerous real-world applications since a large number of real-world problems can be represented as TSP. For example, it can be used in X-Ray crystallography, computer wiring, vehicle routing, and order picking in warehouses [24]. Some more recent applications of TSP include route planning for Unmanned Aerial Vehicles (UAVs) [25], in delivery services using UAVs [26], in emergency air logistics [27], in robotic automated storage and retrieval system [28] and robot path planning [29].

4. Related Works

4.1. Existing Swarm Intelligence Algorithms Applied to TSP

In this section, some previous works in which swarm intelligence algorithms have been adapted and enhanced for solving TSP are presented.

In [30], the firefly algorithm is adapted and enhanced for solving TSP by using the method of swap sequences and Genetic Algorithm (GA). GA is first used to initialise the population of search agents. The serial number coding method is used for representing the discrete state space of TSP and the method of swap sequences is used to redefine the equations of the firefly algorithm in order to adapt the algorithm to TSP. Three neighbourhood structures are also used to redefine the disturbance mechanism of the firefly algorithm. The algorithm is tested using five TSP problems from TSPLIB and it is found to provide solutions which are close to the known optimal solutions.

In [31], a discrete sparrow search algorithm is proposed for solving TSP. For initialization of the population's positions, the roulette wheel selection is used. The path representation method is used for representing a TSP path. The position of the search agents is updated using the same equation as the original sparrow search algorithm and a decoding method called the order-based decoding is used to decode the solution obtained. To increase the diversity of the population, the Gaussian mutation perturbation is used together with swap operators. Furthermore, the 2-opt algorithm is used as a local search to improve the quality of the solutions and to increase the convergence rate. The algorithm is tested using 34 TSP instances and is found to be robust and have good convergence characteristics.

In [32], the Grey Wolf Optimization (GWO) is adapted for solving TSP by the use of swap sequences and swap operators. It makes use of the general steps of the Grey Wolf optimization algorithm. In the initialization stage, the population is initialized with a random TSP path and the agents with the three best solutions are chosen as the Alpha, Beta, and Delta wolves. The position of each search agent is then updated by considering the Alpha, Beta, and Delta wolves by making use of the method of swap sequences. The algorithm is used for solving benchmark TSP problems and is found to provide better results than ACO and GA for several TSP problems.

In [33], the chicken swarm optimization algorithm is adapted for solving TSP by using the methods of swap operators, order crossover, and reverse order mutation. The integer coding method is used for the solution representation, and the method of swap operators is used for updating the position of the search agents. The order crossover, and reverse order mutation are also used for updating the position of the search agents in the state space by

increasing the population diversity. Five instances from TSPLIB are used for testing the algorithm which is found to be effective in solving TSP.

In [34], a discrete spider monkey optimization algorithm is proposed for solving TSP. It makes use of the method of swap sequences and swap operators for updating the position of the search agents. The position of each search agent is updated based on the position of a subgroup leader, known as the local leader, and the position of a main group leader, known as the global leader, and also their self-experience. The algorithm is tested using TSP instances from TSPLIB and its performance is compared with other swarm intelligence algorithms. It is found to have a good performance in solving TSP.

In [8], an enhanced swap sequence based PSO algorithm is proposed for solving TSP. It makes use of path representation for representing solutions in the state space, and the method of swap operators for updating the positions. It incorporates the three strategies of the continuous XPSO algorithm and uses the method of swap operators in order to be suitable for solving TSP. The three strategies include a forgetting ability for each particle, the adjustment of the acceleration coefficients by making use of the population's experience, and the use of both the global and local exemplars for learning. The algorithm is found to provide better results than the swap sequence-based PSO.

4.2. Discrete Adapted Dragonfly Algorithm

In this section, a description of the adapted discrete dragonfly algorithm that we proposed in [12] is given. The algorithm makes use of the path representation method to represent a potential solution, that is a TSP path, and it makes use of the method of swap sequence [35] which was originally used for adapting PSO to TSP. The adapted discrete DA adapts the original DA algorithm to be suitable for solving TSP by the following: firstly, the method of path representation is used to represent a potential solution, that is, a TSP path which is taken as the position of a search agent. Secondly, the equations for calculating the five factors used in DA, the step vector, and the position vector are adapted to be suitable for the path representation. Thirdly, the five factors, the step vector, and the position vector are calculated using the method of swap sequence. The pseudocode of the adapted discrete DA is given in Algorithm 2.

Algorithm 2: Adapted Discrete DA Algorithm for TSP

```

1 Initialize the population's positions with random TSP paths;
2 Initialize the step vectors with random swap sequences;
3 while end condition do
4   Calculate the objective values of all dragonflies;
5   Update the food source and enemy;
6   Update the weights;
7   Calculate the factors using (9), (10), (11), (12), (13);
8   Update step vector using (14);
9   Update position vector using (15);
10 end
```

In the discrete adapted DA algorithm, the position and step vector of the search agents are first initialized with a random TSP path and a random swap sequence respectively. Then in each iteration, the objective value of each search agent's position, that is the cost of the TSP path represented by the search agent's position, is calculated. The position with the lowest objective value is taken as the food source and that with the highest objective value is taken as the enemy. The separation, alignment, cohesion, attraction to food, and distraction from enemy factors are calculated, and their corresponding weights are updated. Finally, the position of the search agents is updated using the step and the position vectors. Contrary to the original continuous DA, the radius of neighbourhood of the dragonflies is not considered in the discrete adapted DA. This is because the Euclidean distance between

the dragonflies in a discrete state space cannot be easily calculated and hence all the search agents are considered to be in the same neighbourhood.

4.3. Initialization

In the initialization phase, the positions of the search agents are first initialized with a random solution, that is a TSP path. The TSP path is represented using path representation, that is, a permutation of numbers representing cities. The numbers indicates the order of visit of the cities. An example of a TSP path for a TSP problem consisting of 4 cities can be: 3 2 1 4 3. This indicates that starting from the city denoted by the number 3, city 2 will be visited next, followed by city 1 and city 4, before returning to the starting point, that is, city 3.

The step vector of the search agents is then initialized with a velocity, which in this case, is a random swap sequence. A swap sequence consists of a number of swap operators, and each swap operator contains a pair of indices which represents cities in a TSP path. It indicates that the position of the two cities will be swapped when the swap operator is applied to a TSP path. For example, a swap operator, SO , can be represented by $SO(2, 4)$, and a swap sequence, SS , can be represented by $SS = (SO(2, 4), SO(1, 2))$.

4.4. Calculation of Factors

The separation, alignment, cohesion, attraction to food, and distraction from enemy factors are calculated using Equations (9)–(13). These equations have been produced by adapting the Equations (1)–(5) from the original DA. This is because the factors of a search agent are calculated using the positions and step vectors of its neighbours, and in the adapted discrete DA, the positions and step vectors are TSP paths and swap sequences respectively. Hence the equations used in the original DA are not suitable for the adapted discrete DA.

The separation factor is calculated as follows:

$$S_i = Inv\left(\bigoplus_{j=1}^N X \ominus X_j\right) \quad (9)$$

where X , X_j , and N are the the current search agent's position, the j -th neighbour's position, and the total number of neighbours respectively.

The ' \ominus ' operator indicates the subtraction of two positions, that is two TSP paths, to produce a swap sequence. For example, the subtraction of two paths, $X = 1 3 4 2$ and $X_j = 2 3 1 4$, is $X \ominus X_j = SO(1, 3), SO(3, 4)$.

The ' \oplus ' operator indicates the merging of the swap sequences into only one swap sequence which contains all the swap operators from the different swap sequences in sequential order. For example, for the swap sequences $SS_1 = SO(2, 3), SO(4, 1)$ and $SS_2 = SO(1, 3)$, $SS_1 \oplus SS_2 = SO(2, 3), SO(4, 1), SO(1, 3)$.

' Inv ' indicates that the swap sequence is inversed. For example, for $SS = SO(2, 3), SO(4, 1)$, $Inv(SS) = SO(1, 4), SO(3, 2)$.

The alignment factor is calculated as follows:

$$A_i = V_{avg} \quad (10)$$

where V_{avg} is the step vector of the neighbour having the fitness closest to the average fitness in the neighbourhood.

The cohesion factor is calculated as follows:

$$C_i = X_{avg} \ominus X \quad (11)$$

where X_{avg} is the position of the neighbour having the fitness closest to the average fitness in the neighbourhood and X is the current search agent's position.

The attraction to the food source factor is calculated as follows:

$$F_i = X_f \ominus X \quad (12)$$

where X_f is the food source's position and X is the current search agent's position.

The distraction from the enemy factor is calculated using the procedure ' $CalculateE_i()$ ' which provides a swap sequence.

$$E_i = CalculateE_i(X_e, X) \quad (13)$$

X_e is the enemy's position and X is the current search agent's position. The procedure compares each city in X_e and X and if a city is similar in both positions, it generates a swap operator which consists of that city and another different random city. This is to decrease the similarity between the two TSP paths.

4.5. Update of Positions

For updating the position of the search agents, the step vector is first calculated as follows:

$$\Delta X_{t+1} = (sS_i \oplus aA_i \oplus cC_i \oplus fF_i \oplus eE_i) \oplus w\Delta X_t \quad (14)$$

where S_i , A_i , C_i , F_i , and E_i are the separation, alignment, cohesion, food factor, and enemy factors of the i -th dragonfly respectively, s , a , c , f , and e are the separation, alignment, cohesion, food factor, and enemy factor's weights respectively, w is the inertia weight, and t is the iteration counter.

In this equation, ' \oplus ' indicates the merging of two swap sequences, resulting in one swap sequence with all the swap operators in the first swap sequence followed by all the swap operators in the second swap sequence.

The position vector of the search agent is then calculated as follows:

$$X_{t+1} = X_t \otimes \Delta X_{t+1} \quad (15)$$

In this equation, ' \otimes ' indicates that the swap sequence ' ΔX_{t+1} ' will be applied to the path ' X_t '. The application of a swap sequence to a path means that each swap operator in the swap sequence will be applied to the path sequentially to produce a new path. An example of applying a swap operator to a path is given below.

Considering a path $X = 2 4 1 3$ and a swap operator $SO(2, 3)$, the cities in the second and third positions of X will be swapped. Hence, the resultant path will be $X = 2 1 4 3$.

5. Proposed Enhanced Adapted Discrete DA

In this section, a description of the proposed enhanced discrete adapted DA algorithm is provided. The proposed algorithm improves the exploitation phase of the adapted discrete DA by using the steepest ascent hill climbing algorithm as a local search. After the position of the search agents is updated using Equation (15), the hill climbing algorithm is employed to further exploit the region obtained and to update the position of the search agent to a better one. The steepest ascent hill climbing algorithm starts at the position obtained by Equation (15) and then looks for every possible position in that area of the search space. It then selects the one with the lowest cost. Hence, the steepest ascent hill climbing algorithm is able to locate the optimum solution in the area initially obtained by the search agent. Moreover, to prevent the algorithm from getting stuck in a local optimum, the position of a search agent is changed to a random solution when it cannot be further improved. This is done by keeping track of the personal best solution that is found by a search agent. If the personal best solution does not change over a certain number of iterations, the search agent's position is changed to another random position so as to allow it to get out of the local optimum and to search other regions of the state space. The pseudocode of the proposed algorithm is given in Algorithm 3.

Algorithm 3: Enhanced Adapted Discrete DA Algorithm for TSP

```

1 Initialize the population's positions with random TSP paths;
2 Initialize the step vectors with random swap sequences;
3 PBest_Stagnancy = 0;
4 while end condition do
5   Calculate the objective values of all dragonflies;
6   Update the food source and enemy;
7   Update the weights;
8   Calculate the factors using (9), (10), (11), (12), (13);
9   Update step vector using (14);
10  Update position vector using (15);
11  Initialize position as current position for hill climbing;
12  while local optima is not reached do
13    Generate neighbours;
14    for each neighbour do
15      if cost of neighbour < cost of current position then
16        | current position = neighbour position;
17      end
18    end
19  end
20  if cost of position < cost of personal best position then
21    | best position = position;
22  else
23    | PBest_Stagnancy = PBest_Stagnancy + 1;
24  end
25  if PBest_Stagnancy > 3 then
26    | Change position to random position;
27  end
28 end

```

In the initialization phase, the position and step vectors are initialized with a random TSP path, and a random swap sequence respectively. This step is similar to the adapted discrete DA in Section 4.2. In addition, a personal best stagnancy variable is initialized to zero. This variable is used to keep track of the number of iterations in which the best solution found by a search agent has not improved.

In the main loop of iteration, the objective cost of each search agent is first calculated, and the food and enemy are updated with the best and worst positions respectively. Then the separation, alignment, cohesion, attraction to food and distraction from enemy factors are calculated. Contrary to the adapted discrete DA in Section 4.2, in the proposed algorithm, these factors are calculated based on the path obtained after applying the previous factor; that is the separation factor is first calculated and the swap operators obtained are immediately applied to the TSP path represented by the position of the search agent to update the path. Then the alignment factor is calculated based on the updated path after applying the separation factor. Similarly, the path is updated and then the cohesion factor is calculated. The food factor is then calculated based on the updated path after applying the swap operators of the cohesion factor and the enemy factor is calculated based on the updated path after applying the food factor. The path with the lowest cost is then chosen as the next position of the search agent. This is done so as to increase the efficiency of the algorithm so that it can provide good solutions in a short amount of time.

After the position of a search agent is updated using Equation (15), the hill climbing algorithm is employed to further update the position as follows: the position obtained by (15) is taken as the current position for the hill climbing algorithm. Then a set of neighbouring solutions is generated and the one with the lowest cost is selected as the

current position. The steps of generating neighbours and selecting the one with the lowest cost as the current position are repeated until a solution with a lower cost cannot be found. The use of the hill climbing algorithm is to improve the exploitation of the dragonfly algorithm.

In order to prevent the algorithm from being stuck in local optima, the personal best solution of each search agent, which is the best solution found by the search agent, is recorded. In each iteration, after the position of the search agent has been updated, the personal best solution is checked and updated if it has changed. If the personal best solution remains unchanged over a certain amount of iterations, then the position of the search agent is changed to a random solution. This is to allow the search agent to get out of the local optimum and explore other regions of the search space.

5.1. Solution Representation

There are several ways to represent a TSP path as the position of a search agent in a discrete state space such as binary, path, adjacency, ordinal, and matrix representations [31]. In this paper, the path representation is used to encode the TSP solutions since this is the most natural representation of a path. This is the same representation used in [12] for the adapted discrete DA algorithm.

5.2. Objective Function

The objective function is the cost of the TSP path, which in this case is the total distance of the TSP path. This is obtained by calculating the distance between each pair of adjacent cities in the TSP path. It is considered that the distance to travel from city i to city j is the same as the distance to travel from city j to city i .

5.3. Update of Positions

Similar to the adapted discrete DA algorithm in Section 4.2, the method of swap sequences is used for updating a position, that is a TSP path. A sequence contains a number of swap operators which indicate that the two cities in the position denoted by the swap operator will be swapped to produce a new TSP path. The swap operators in the swap sequence are applied sequentially to the TSP path. For example, for a TSP path 2 4 1 3, and swap sequence $SO(1,2)$, $SO(3,2)$, the TSP path 4 1 2 3 will be obtained when the swap sequence is applied to the TSP path.

5.4. Experimental Parameters

Table 1 shows the parameters used for the optimized DA algorithm, their description, and their values.

Table 1. Experimental parameters.

Parameter	Description	Value
X_i	The position of search agent i	A TSP path, example: 1 3 4 2 1
V_i	The step vector of search agent i	A swap sequence, example: $SO(1,3)$ $SO(2,1)$
X_f	The food position	A TSP path, example: 1 4 2 3 1
X_e	The enemy position	A TSP path, example: 1 2 4 3 1
S_i	The separation factor of the i^{th} search agent	A swap sequence, example: $SO(2,4)$ $SO(1,2)$
A_i	The alignment factor of the i^{th} search agent	A swap sequence, example: $SO(1,4)$ $SO(3,2)$
C_i	The cohesion factor of the i^{th} search agent	A swap sequence, example: $SO(1,3)$ $SO(3,4)$
F_i	The food factor of the i^{th} search agent	A swap sequence, example: $SO(3,4)$ $SO(1,2)$

Table 1. Cont.

Parameter	Description	Value
E_i	The enemy factor of the i^{th} search agent	A swap sequence, example: $SO(3, 2) SO(4, 1)$
s	The separation weight	A real value, example: 1.5
c	The cohesion weight	A real value, example: 1.2
a	The alignment weight	A real value, example: 2.3
f	The attraction to food weight	A real value, example: 1.2
e	The distraction from enemy weight	A real value, example: 0.5
w	The step vector weight	A real value, example: 0.1

6. Experimental Results and Analysis

In this section, a description of the experimental datasets used, the experimental setup, and the results with some discussions are provided.

6.1. Experimental Dataset

The test data consist of a TSP problem with 50 nodes, where each node represents a location in the area of Kuala Lumpur (KL). The cost between each pair of nodes is the distance needed to travel from one node to the other. It is considered that the distance to travel from city i to city j is the same as the distance to travel from city j to city i . The aim is to find the shortest route to visit each node once and to return to the initial node. The distance between two locations is taken as the shortest distance in kilometers (km) that can be taken by a vehicle based on Google Maps.

To test the proposed algorithm with TSP problems of different sizes, the TSP data is changed to 10, 20, and 40 nodes by taking the first 10, 20, and 40 locations respectively.

Moreover, several benchmark datasets from TSPLIB are used to test the performance of the proposed algorithm. Specifically, the burma14, ulysses16, ulysses22, bays29, eil51, berlin52, st70, eil76, rat99 and kroA100 datasets consisting of 14, 16, 22, 29, 51, 52, 70, 76, 99, and 100 nodes respectively are used. The datasets are in the form of coordinates and the distance matrix of each dataset is constructed by calculating the Euclidean distance between the nodes.

6.2. Experimental Setup

The proposed algorithm is used for solving the TSP problem with 50, 40, 20, and 10 locations in KL and the results are recorded in terms of the cost of the solution obtained, the time taken to converge to the optimal solution, and the total time taken by the algorithm.

To compare the performance of the proposed algorithm, the discrete adapted dragonfly algorithm in [12], and the enhanced SSPSO in [8] are used for solving the same TSP problems of 50, 40, 20, and 10 locations and the results are recorded in terms of the cost of the solution obtained, the time taken to converge to the optimal solution, and the total time taken by the algorithm. The results of the proposed algorithm are compared to that of the discrete adapted dragonfly algorithm and the enhanced SSPSO algorithm. The enhanced SSPSO is used for comparing the performance of the proposed algorithm since it is our own algorithm which had been used for the same dataset, and TSP problem, that is the delivery system in the area of Kuala Lumpur. In order to have a better algorithm for the delivery system, the new optimized DA algorithm is proposed in this paper.

The experiments are repeated for different numbers of maximum iteration and search agents. Specifically, the number of maximum iterations used are 20, 50, 100, 200, and 500, and the number of search agents used is 5, 10, 20, and 40.

Furthermore, in order to compare the performance of the proposed algorithm to that of other swarm intelligence algorithms, the algorithm is applied to benchmark TSP problems

from TSPLIB and the best solution obtained by the proposed algorithm is compared to the best solution obtained by Ant Colony Optimization (ACO), Velocity Tentative PSO (VTPSO), Artificial Bee Colony with Swap Sequence (ABCSS), Discrete Spider Monkey Optimization (DSMO), Genetic Algorithm (GA), Producer Scrounger Method (PSM), and Grey Wolf Optimizer (GWO) algorithms. The results of ACO, VTPSO, ABCSS, and DSMO are taken from [34]. The results of ACO GA, PSM, and GWO are taken from [32]. The maximum number of iterations used for the proposed optimized DA is 500 and the number of search agents used is between 20 and 100.

Both the proposed optimized discrete adapted DA and the discrete adapted DA were implemented in MATLAB and all experiments were conducted on a MacOs Monterey operating system, Apple M2 chip CPU, and 8 GB RAM.

6.3. Results and Discussion

6.3.1. Greater Kuala Lumpur TSP Problem

In this section, we present a comparison and discussion on the performance of the proposed algorithm when applied to our own dataset consisting of locations in the area of Greater Kuala Lumpur which models a delivery system.

Tables 2–5 show the results obtained when the proposed optimized discrete DA, the discrete adapted DA, and the enhanced swap sequence based PSO are used for solving the TSP problem with 50, 40, 20, and 10 locations respectively. The experiments are conducted by using different number of maximum iterations and search agents and the results are recorded in terms of the cost of the solution, that is TSP path, provided by the algorithms, the time taken to converge to the global optimal solution, and the total time taken by the algorithms.

Figures 2–5 show the convergence curve of the proposed optimized discrete adapted DA and the discrete adapted DA in solving a TSP of 50, 40, 20, and 10 locations respectively. The figures show the convergence of the algorithms for 5, 10, 20, and 40 search agents and for a maximum iteration of 200. The figures show the convergence rate of the two algorithms and also the cost of the solution, that is the TSP path provided.

An example of a TSP path for 20 locations in Kuala Lumpur area provided by the optimized discrete adapted DA algorithm is given in Figure 6. The cost of the TSP path is 105.4 and the path is as follows: 11 15 18 16 10 9 4 13 2 3 12 1 20 19 17 7 6 5 8 14 11. The numbers represent the cities and their order represent the order in which they will be visited.

From Tables 2–5 it can be deduced that the proposed optimized discrete adapted DA algorithm has a higher effectiveness than the discrete adapted DA algorithm as the cost of the TSP path provided by the proposed algorithm is significantly lower in all of the experiments conducted. For example, for a maximum iteration of 500 and 40 search agents, the costs of the TSP paths obtained by the discrete adapted DA for 50, 40, 20, and 10 locations are 507.8, 409.3, 161.9, and 69.6 respectively while those obtained by the proposed optimized adapted discrete DA are 200.0, 178.7, 105.4, and 65.9 respectively. This means that the optimized adapted discrete DA improves the solution obtained by the adapted discrete DA by 60.6%, 56.3%, 34.9%, and 5.3% for 50, 40, 20, and 10 locations respectively.

Even for smaller number of iterations and search agents, the proposed algorithm converges to solutions with lower costs than the discrete adapted DA. For example, for a maximum of 20 iterations and only five search agents, the costs of the TSP paths obtained by the discrete adapted DA for 50, 40, 20, and 10 locations are 556.2, 478.1, 191.9, and 82.9 respectively while those obtained by the optimized discrete adapted DA are 229.4, 217.8, 117.5, and 65.9 respectively. This indicates that the optimized adapted discrete DA provides solutions which are 58.8%, 54.4%, 38.8%, and 20.5% better than those provided by the adapted discrete DA for 50, 40, 20, and 10 locations respectively.

Table 2. Performance comparison of proposed optimized discrete DA and discrete adapted DA in solving TSP of 50 cities.

Maximum No. of Iterations	No. of Search Agents	Enhanced SSPSO			Discrete Adapted DA			Proposed Adapted DA		Optimized		Discrete	
		Cost of Solution (km)	Time Taken to Converge to Optimum (s)	Total Taken (s)	Time	Cost of Solution (km)	Time Taken to Converge to Optimum (s)	Total Taken (s)	Time	Cost of Solution (km)	Time Taken to Converge to Optimum (s)	Total Time Taken (s)	
20	5	532.0	0.0359	0.0481	556.2	3.519	7.91	229.4	3.5325	6.6253			
20	10	550.0	0.04071	0.1139	571.0	18.0617	27.6912	223.3	6.9313	9.0644			
20	20	542.0	0.2214	0.2959	570.7	15.7653	130.8685	223.5	25.5084	26.5358			
20	40	547.0	0.1365	0.5762	546.0	2.1136	770.9831	232.3	57.8731	58.7077			
50	5	525.0	0.0892	0.3160	561.6	6.2466	54.9875	217.2	10.734	12.7241			
50	10	525.0	0.0021	0.6858	539.0	3.3454	190.3065	225.4	19.015	24.0916			
50	20	541.0	0.5838	1.2465	524.0	53.0974	823.7265	211.8	64.8796	67.1365			
50	40	509.0	3.3133	6.02159	542.3	587.3994	4322.1249	206.0	147.4982	150.3943			
100	5	525.0	1.0883	1.3751	537.5	6.8421	200.1581	223.5	7.1802	18.6268			
100	10	510.0	0.1848	3.7681	538.0	178.4626	788.3791	213.7	46.6898	48.3536			
100	20	518.0	3.1911	9.6636	539.2	1429.93	3302.5555	209.4	29.4048	122.456			
100	40	504.0	7.3404	17.9399	534.8	426.2547	15,899.7002	210.6	229.6718	240.2979			
200	5	530.0	4.8935	6.3373	537.0	180.8259	429.2883	217.9	48.1498	48.6066			
200	10	501.0	0.7947	14.5220	529.2	798.1125	1676.7151	206.6	59.3682	109.5615			
200	20	497.0	17.5760	38.0765	527.7	4210.0023	7259.5666	199.4	126.6875	195.9365			
200	40	494.0	20.0064	82.9766	519.5	1934.8816	34,189.6828	206.7	324.0921	554.3716			
500	5	478.0	26.4978	42.7225	519.6	504.9482	6311.1325	213.1	96.5744	100.8358			
500	10	494.0	45.6093	101.8348	487.2	17,838.5761	26,586.0832	196.8	121.3713	245.6557			
500	20	488.0	21.9401	309.6251	517.3	16,875.0334	113,766.0094	193.6	262.4892	500.1441			
500	40	494.0	30.9125	588.0762	507.8	283,915.1445	546,679.3105	200.0	534.5106	1198.7924			

Table 3. Performance comparison of proposed optimized discrete adapted DA and discrete adapted DA in solving TSP of 40 cities.

Maximum No. of Iterations	No. of Search Agents	Enhanced SSPSO			Discrete Adapted DA			Proposed Adapted DA		Optimized	Discrete
		Cost of Solution (km)	Time Taken to Converge to Optimum (s)	Total Taken (s)	Time	Cost of Solution (km)	Time Taken to Converge to Optimum (s)	Total Taken (s)	Time	Cost of Solution (km)	Time Taken to Converge to Optimum (s)
20	5	431.0	0.02126	0.03446	478.1	0.13406	11.4666	217.8	2.7567	4.3227	
20	10	461.0	0.02927	0.0843	463.9	1.9294	29.2706	222.8	5.2488	5.4381	
20	20	442.0	0.0140	0.1880	452.3	107.0152	131.4918	197.2	10.0264	12.3698	
20	40	435.0	0.01673	0.4016	430.6	20.2805	686.2293	188.9	29.322	36.2265	
50	5	448.0	0.06111	0.1847	428.9	32.3926	65.8366	207.3	3.9897	6.7687	
50	10	415.0	0.4449	0.8789	436.8	95.0279	256.7351	195.5	12.7688	13.4064	
50	20	427.0	0.1988	1.0586	449.8	87.115	1004.6713	200.4	14.9048	28.4775	
50	40	418.0	1.8880	3.1482	444.3	2194.2967	4545.5709	194.3	65.8466	71.1544	
100	5	422.0	0.0027	1.2992	422.6	114.8649	270.0771	190.7	7.247	11.6643	
100	10	419.0	1.3789	3.1608	458.0	609.6101	950.9694	193.3	20.9147	35.4175	
100	20	433.0	1.6942	4.0857	444.6	2547.628	3991.4558	185.0	40.165	49.4274	
100	40	407.0	1.6516	15.0465	441.4	1860.1602	25,521.9187	194.0	103.962	119.8753	
200	5	402.0	1.4485	5.3046	422.8	1.31	361.4874	186.8	13.2504	20.6362	
200	10	421.0	2.9958	11.0945	444.0	862.4717	1301.4488	199.8	44.9657	57.7674	
200	20	418.0	1.7099	22.8645	418.6	2661.0911	5393.6283	181.6	44.927	110.284	
200	40	420.0	45.7771	56.4051	436.0	6848.2599	25,271.4722	178.7	212.0086	257.4372	
500	5	421.0	30.1391	30.2641	435.5	2122.968	7159.2775	190.0	43.844	63.6809	
500	10	417.0	14.5123	93.8408	429.8	4701.7931	23,959.7151	185.0	85.5298	137.7637	
500	20	389.0	22.4112	194.3102	423.1	100,073.4386	124,577.1437	184.8	182.1964	292.4497	
500	40	404.0	7.1911	498.5430	409.3	72,913.9465	357,630.9028	179.4	501.9859	681.6409	

Table 4. Performance comparison of proposed optimized discrete adapted DA and discrete adapted DA in solving TSP of 20 cities.

Maximum No. of Iterations	No. of Search Agents	Enhanced SSPSO			Discrete Adapted DA			Proposed Adapted DA		Optimized	Discrete
		Cost of Solution (km)	Time Taken to Converge to Optimum (s)	Total Taken (s)	Time	Cost of Solution (km)	Time Taken to Converge to Optimum (s)	Total Taken (s)	Time	Cost of Solution (km)	Time Taken to Converge to Optimum (s)
20	5	195.0	0.004488	0.01227	191.9	2.0448	3.8275	117.5	0.32687	1.0266	
20	10	173.0	0.02451	0.0298	187.6	5.9708	14.7066	109.1	1.7194	1.8762	
20	20	182.0	0.0096	0.0752	192.7	7.3479	64.1203	107.1	2.2349	3.5711	
20	40	168.0	0.09121	0.1348	186.1	18.9694	396.1467	106.1	3.7612	6.3904	
50	5	173.0	0.0035	0.0603	175.0	6.4056	24.9236	105.4	0.63344	1.9217	
50	10	176.0	0.1615	0.2060	160.1	0.0092196	97.1998	106.1	3.8454	3.9797	
50	20	162.0	0.0033	0.4717	168.9	192.3332	501.511	105.4	3.4697	6.4025	
50	40	151.0	1.0569	1.4717	168.8	341.0088	2616.6784	105.4	1.6023	16.6091	
100	5	171.0	0.0048	0.2566	186.1	68.1113	125.241	108.8	1.0233	3.0716	
100	10	173.0	0.2345	0.6208	171.6	269.7031	475.5543	106.8	3.3758	5.4659	
100	20	167.0	0.3571	1.9250	171.1	7.419	2047.3412	105.7	11.401	12.6007	
100	40	170.0	2.0494	4.0331	174.9	5646.5838	8242.855	105.4	1.6434	32.2757	
200	5	163.0	0.9734	1.1322	180.1	89.7254	164.0154	105.7	0.8572	6.74	
200	10	166.0	0.1077	3.3940	175.4	11.5837	624.1664	105.4	3.9401	10.1722	
200	20	163.0	0.0767	6.4863	162.7	202.5637	2352.474	105.4	9.3677	21.9359	
200	40	155.0	0.0475	19.5566	170.7	4165.0315	11,489.6363	105.4	23.7822	60.105	
500	5	164.0	0.0311	10.7217	163.4	875.6302	3632.2101	105.4	9.4902	13.2498	
500	10	164.0	3.2880	25.2614	175.8	1215.4295	15,191.4602	105.4	9.2816	30.2117	
500	20	153.0	0.0044	56.4343	155.7	37,104.5216	61,427.5274	105.4	2.6679	64.409	
500	40	155.0	1.7389	111.9266	161.9	83,639.0656	247,067.0486	105.4	79.663	157.4536	

Table 5. Performance comparison of proposed optimized discrete adapted DA and discrete adapted DA in solving TSP of 10 cities.

Maximum No. of Iterations	No. of Search Agents	Enhanced SSPSO			Discrete Adapted DA			Proposed Adapted DA		Optimized	Discrete
		Cost of Solution (km)	Time Taken to Converge to Optimum (s)	Total Taken (s)	Time	Cost of Solution (km)	Time Taken to Converge to Optimum (s)	Total Taken (s)	Time	Cost of Solution (km)	Time Taken to Converge to Optimum (s)
20	5	80.0	0.0022	0.0028	82.9	0.50424	3.4169	65.9	0.26697	0.56227	
20	10	78.0	0.0072	0.0074	71.2	4.8409	9.6044	65.9	0.25308	0.91832	
20	20	75.0	0.0055	0.0171	78.0	21.0104	35.849	65.9	0.22486	1.2627	
20	40	69.0	0.0248	0.0369	79.0	0.72344	146.1099	65.9	0.41628	2.3977	
50	5	75.0	0.0124	0.01961	81.7	13.2891	15.7673	65.9	0.12756	1.0816	
50	10	71.0	0.0055	0.0440	80.2	25.3691	48.5602	65.9	0.61573	1.4417	
50	20	72.0	0.0996	0.1174	73.4	13.4782	222.6126	65.9	0.22748	2.5893	
50	40	67.0	0.1659	0.2397	76.9	661.0827	915.7478	65.9	0.37537	5.4787	
100	5	73.0	0.0715	0.0733	81.3	42.137	71.0761	65.9	0.21733	1.6297	
100	10	72.0	0.2668	0.2721	77.3	27.525	229.6083	65.9	0.35535	2.3059	
100	20	66.0	0.4829	0.4832	76.7	102.0761	959.875	65.9	0.22507	4.4552	
100	40	68.0	0.3644	0.8834	72.1	607.6719	5944.7225	65.9	1.1377	11.2092	
200	5	75.0	0.3177	0.3179	75.8	1.117	70.7217	65.9	0.47184	3.1885	
200	10	65.0	0.0234	1.080	77.5	6.5001	253.8453	65.9	0.17053	4.7651	
200	20	71.0	1.7071	1.7748	72.0	372.1562	991.3132	65.9	0.24187	8.242	
200	40	67.0	3.4541	4.0137	68.8	416.1922	4307.4956	65.9	0.56337	19.9458	
500	5	69.0	0.0880	2.4140	74.7	0.59851	1637.606	65.9	0.15524	5.3221	
500	10	71.0	5.7717	5.9135	69.8	502.382	6421.5464	65.9	0.90636	9.7569	
500	20	68.0	12.0173	12.7403	71.4	8413.495	28,095.8444	65.9	0.25608	19.0239	
500	40	67.0	25.6158	25.8277	69.6	8756.6288	117,232.4453	65.9	0.37032	47.7772	

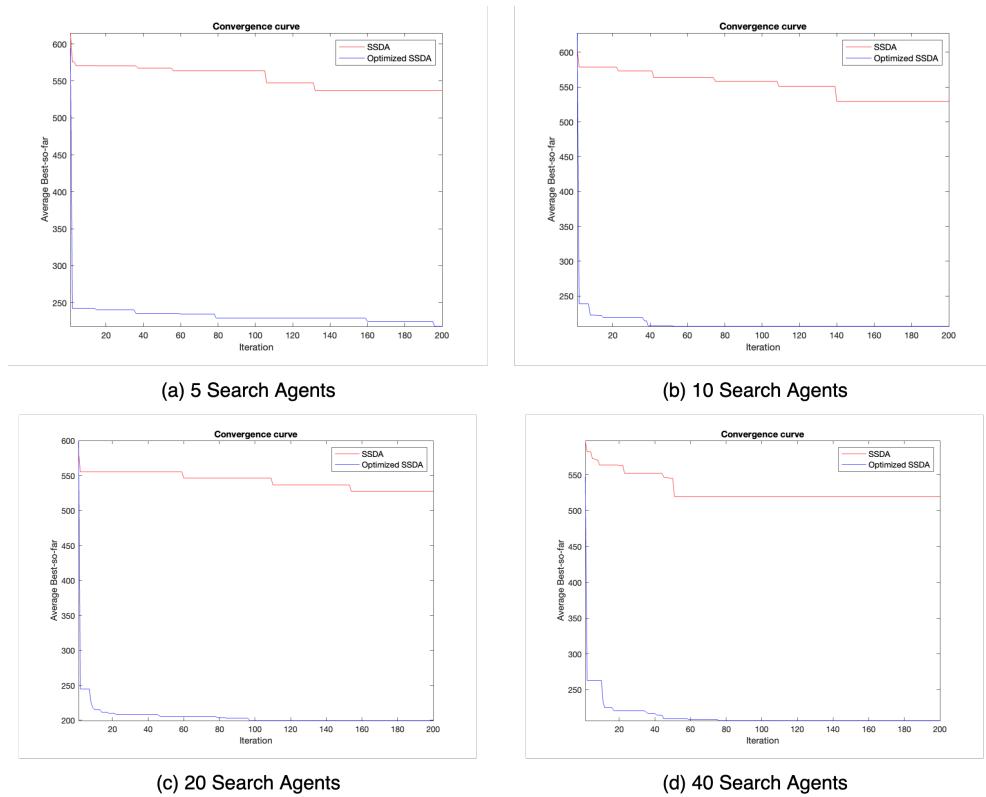


Figure 2. Convergence curve of optimized discrete adapted DA and discrete adapted DA in solving TSP of 50 cities.

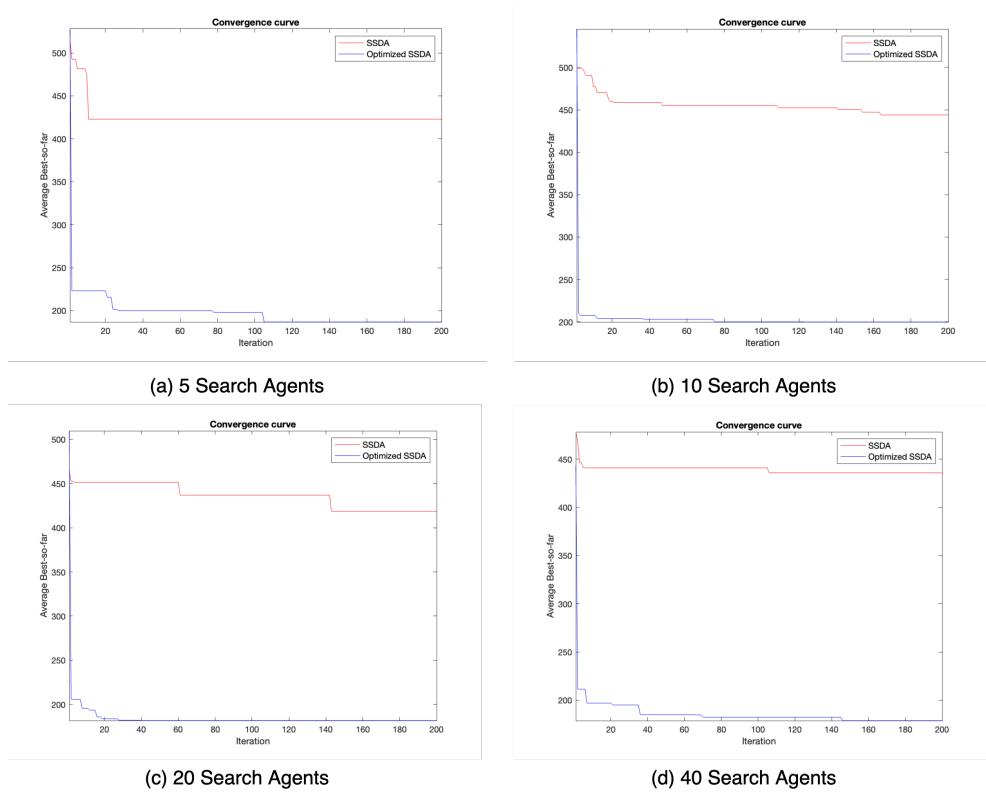


Figure 3. Convergence curve of optimized discrete adapted DA and discrete adapted DA in solving TSP of 40 cities.

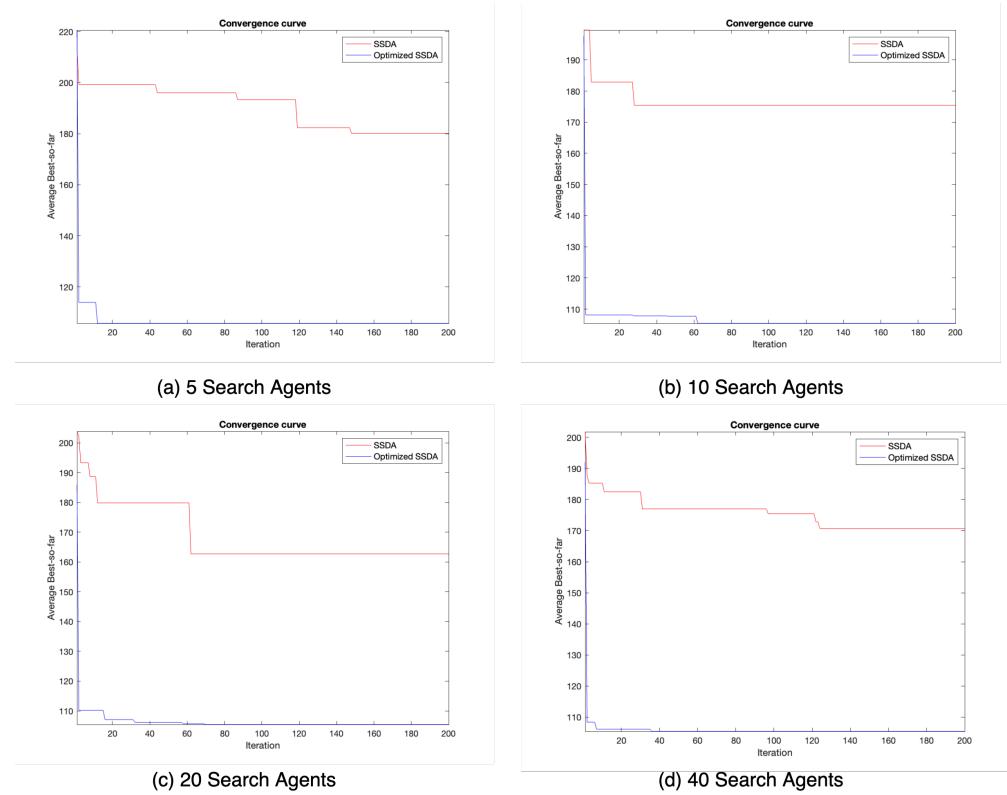


Figure 4. Convergence curve of optimized discrete adapted DA and discrete adapted DA in solving TSP of 20 cities.

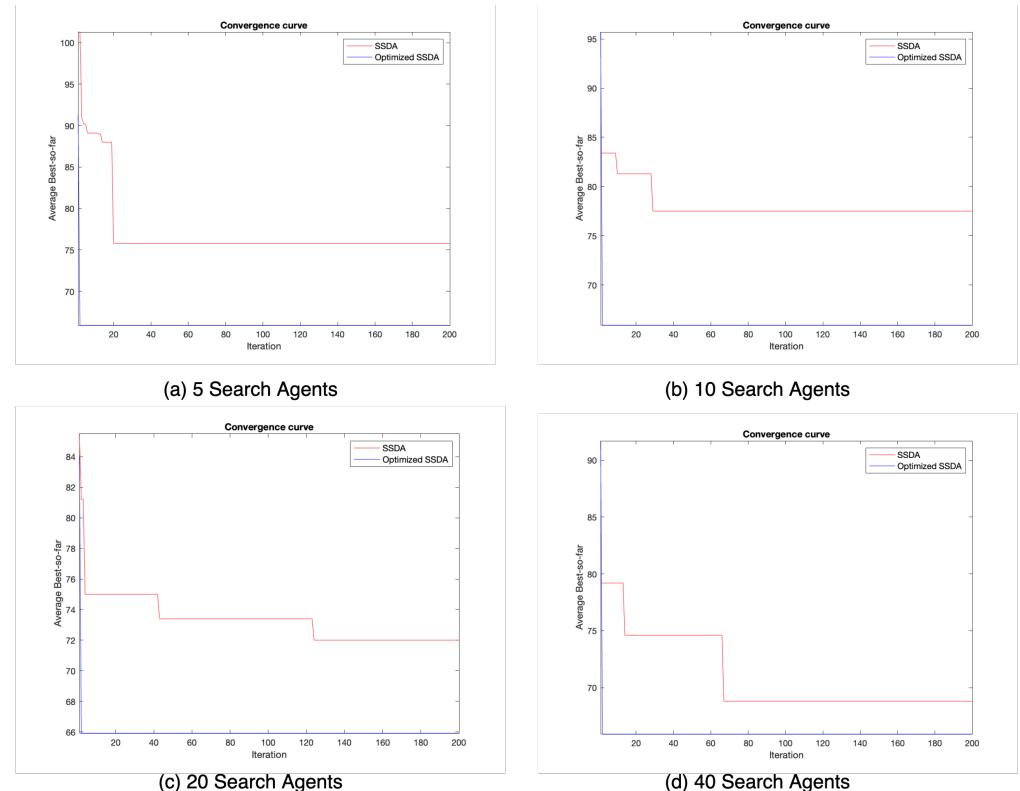


Figure 5. Convergence curve of optimized discrete adapted DA and discrete adapted DA in solving TSP of 10 cities.

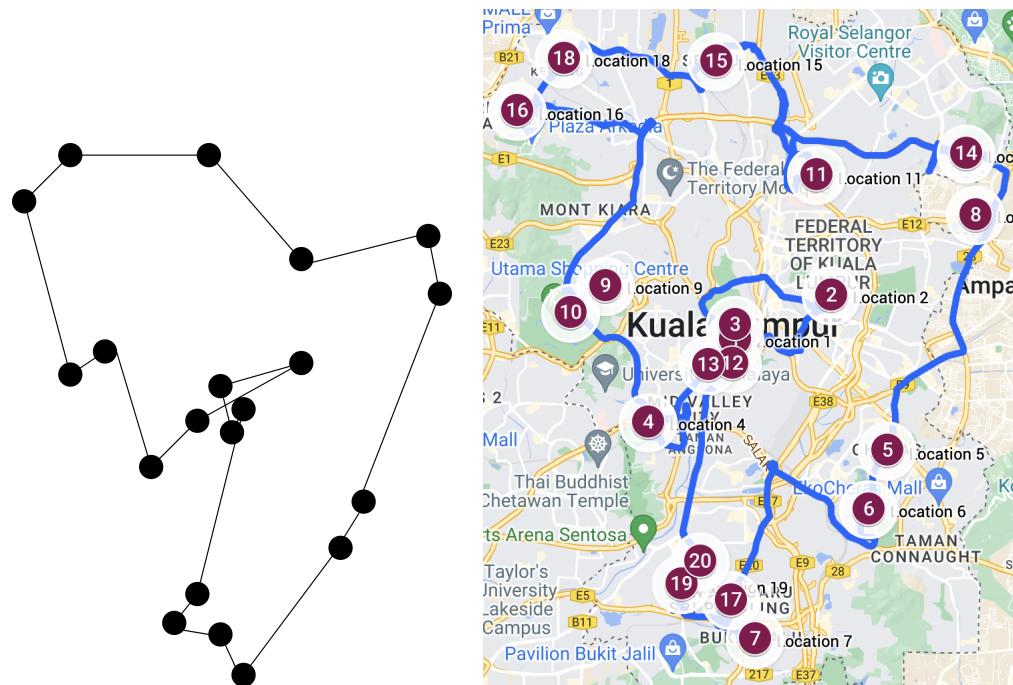


Figure 6. TSP path provided by optimized discrete adapted DA for 20 locations.

Moreover, the costs of the best TSP paths that can be provided by the adapted discrete DA for 50, 40, 20, and 10 locations are 487.2, 409.3, 155.7, and 69.6 respectively while the costs of the best TSP paths that can be provided by the proposed optimized adapted discrete DA for 50, 40, 20, and 10 locations are 193.6, 179.4, 105.4, and 65.9 respectively.

In comparison to the enhanced SSPSO algorithm in [8], the proposed optimized discrete DA algorithm has a higher effectiveness as it provides solutions with a lower cost in all the experiments conducted.

In terms of efficiency, it can be seen from Tables 2–5 that the proposed optimized adapted discrete DA algorithm takes less time than the adapted discrete DA algorithm for execution until the maximum number of iterations. Moreover, in terms of the time taken to converge to the global optimal solution, it can be seen that the proposed algorithm is better than the discrete adapted DA as it takes less time to converge. Hence, it can be deduced that the proposed algorithm has a higher convergence rate as compared to the adapted discrete DA algorithm.

From Figures 2–5, it can be seen that the proposed optimized adapted discrete DA algorithm provides better solution than the adapted discrete DA as the proposed algorithm converges to solutions with lower costs in all cases. Moreover, it can be seen that in multiple cases the proposed algorithm has a higher convergence rate than the adapted discrete DA as the proposed algorithm converges at earlier iterations.

6.3.2. Benchmark TSP Problems

In this section, we present a comparison and discussion on the performance of the proposed algorithm in solving benchmark TSP problems.

Table 6 shows a comparison of the performance of the proposed optimized DA algorithm and other swarm intelligence algorithms, namely ACO, GA, PSM, and GWO in solving benchmark TSP problems. The results of ACO, GA, PSM, and GWO are taken from [32]. The results are compared in terms of the cost of the best solution that can be obtained by the algorithm. The maximum number of iterations used for the proposed optimized DA, and the other swarm intelligence algorithms is 500, and the maximum number of search agents used is 100.

Table 7 shows a comparison of the performance of the proposed optimized DA algorithm and other swarm intelligence algorithms, namely ACO, VTPSO, ABCSS and DSMO

in solving benchmark TSP problems. The results for ACO, VTPSO, ABCSS and DSMO are taken from [34]. The results are compared in terms of the cost of the best solution that can be obtained by the algorithm. The maximum number of iterations used for ACO, VTPSO, and DSMO is 500, while that for ABCSS is 1000. The number of search agents used for ACO, VTPSO, ABCSS and DSMO is between 100 and 300. For the proposed optimized DA, the maximum number of iterations used is 500, and the maximum number of search agents used is 100. Although the number of search agents and maximum iteration for the proposed algorithm and the other swarm intelligence algorithms are not the same, the results are shown for comparison purposes.

Table 6. Performance comparison of proposed optimized discrete adapted DA and other swarm intelligence algorithms in solving benchmark TSP using a maximum of 100 search agents.

TSP Instance	Cost of Best Solution Obtained				
	Proposed Optimized DA	ACO	GA	PSM	GWO
burma14	30.8785	31.21	30.87	30.87	30.87
ulysses16	73.9876	77.13	74.0	73.99	73.99
ulysses22	75.3097	86.74	76.09	75.51	75.51
bays29	9074.148	9964.78	9336.82	9076.98	9076.98
eil51	430.244	499.92	524.18	438.7	455.24
berlin52	7544.3659	8046.06	9184.19	8109.91	8048.91
st70	687.0724	734.19	1015.0	767.65	752.84
eil76	566.5564	595.58	805.78	591.89	604.32
kroA100	24,205.4508	24,504.9	51446.8	26,419.8	25,983.8

Table 7. Performance comparison of proposed optimized discrete adapted DA and other swarm intelligence algorithms in solving benchmark TSP using different number of search agents.

TSP Instance	Cost of Best Solution Obtained				
	Proposed Optimized DA	ACO	VTPSO	ABCSS	DSMO
burma14	30.8785	31.21	30.87	30.87	30.87
ulysses16	73.9876	77.13	73.99	73.99	73.99
ulysses22	75.3097	84.78	75.31	75.31	75.31
bays29	9074.148	9964.78	9074.15	9074.15	9074.15
eil51	430.244	499.92	429.51	428.98	428.86
berlin52	7544.3659	7870.45	7544.37	7544.37	7544.37
st70	687.0724	734.19	682.57	682.57	677.11
eil76	566.5564	581.42	559.25	550.24	558.68
rat99	1298.888	1366.3	1256.25	1242.32	1225.56
kroA100	24,205.4508	24,504.9	21,307.44	21,299.0	21,298.21

From Table 6, it can be seen that the proposed algorithm achieves the same optimal solution for burma14 as GA, PSM, and GWO. For the ulysses16 dataset, the proposed algorithm provides the same optimal solution as PSM, and GWO. As for all the other datasets, that is, ulysses22, bays29, eil51, berlin52, st70, eil76, and kroA100, the proposed optimized DA provides better solutions as compared to ACO, GA, PSM, and GWO when the same number of search agents and maximum iterations is used.

From Table 7, it can be seen that the proposed algorithm can achieve the optimal solution for five of the datasets, namely for burma14, ulysses16, ulysses22, bays29, and berlin52, even though a smaller number of search agents and maximum iteration is used as compared to ACO, VTPSO, ABCSS, and DSMO. Although in some cases, VTPSO, ABCSS, and DMSO can provide a solution with lower cost as compared to our proposed algorithm,

it is expected that our proposed algorithm will be able to provide similar or even better solutions if the same number of search agents and maximum iteration is used.

7. Conclusions and Future Work

Swarm intelligence algorithms are popular metaheuristic algorithms for solving complex optimization problems owing to the simple interactions of the search agents which give rise to a global intelligent behaviour. The dragonfly algorithm is a swarm intelligence algorithm inspired by the swarming behaviours of dragonflies while hunting and migrating. It has been found to have a higher performance than multiple other swarm intelligence algorithms in various applications.

Since DA has been found to have a better performance than multiple swarm intelligence algorithms in various applications, it is worth considering its application to the traveling salesman problem which is a popular discrete optimization problem having a plethora of real-world applications. In [3], a binary version of the dragonfly algorithm is proposed. However, this algorithm is difficult to be adapted for discrete problems like TSP. Hence, in [12], we proposed a discrete adapted dragonfly algorithm suitable for TSP. However, this algorithm has not been applied to large TSP problems and it has low effectiveness.

In this paper, we propose an optimized adapted discrete dragonfly algorithm which improves the low effectiveness of the adapted discrete DA in [12]. The proposed algorithm improves the exploitation phase of the adapted discrete DA by using the steepest ascent hill climbing algorithm as a local search. The proposed optimized adapted discrete DA has been tested using TSP instances consisting of 50, 40, 30, 20, and 10 cities. It has been found to provide better solutions, that is TSP paths with a lower cost, as compared to the adapted discrete DA [12], and the enhanced swap sequence based PSO [8] algorithms. It also has a higher convergence rate than the adapted discrete DA. Moreover, it has been tested using several benchmark TSP problems and it has been found to provide optimal solutions or solutions close to the optimal solutions.

For future work, the proposed algorithm can be applied to other real-world applications such as channel routing, planning, scheduling, and logistics.

Author Contributions: Conceptualization, B.A.S.E. and M.B.J.; writing—original draft preparation, B.A.S.E. and M.B.J.; writing—review and editing, M.B.J., A.A. and A.W.M.; supervision, M.B.J.; project administration, M.B.J.; funding acquisition, M.B.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Sunway University Internal Grant Scheme 2022 grant number GRTIN-IGS-DCIS[S]-11-2022.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

DA	Dragonfly Algorithm
TSP	Traveling Salesman Problem
BDA	Binary Dragonfly Algorithm
MODA	Multi-Objective Dragonfly Algorithm
PSO	Particle Swarm Optimization
ACO	Ant Colony Optimization
HC	Hill Climbing

GA	Genetic Algorithm
GWO	Grey Wolf Optimizer
XPSO	Expanded PSO
SO	Swap Operator
SS	Swap Sequence
VTPSO	Velocity Tentative PSO
ABCSS	Artificial Bee Colony with Swap Sequence
DSMO	Discrete Spider Monkey Optimization
PSM	Producer Scrounger Method

References

- Yang, X.S. *Nature-Inspired Optimization Algorithms*; Academic Press: Cambridge, MA, USA, 2020.
- Slowik, A.; Kwasnicka, H. Nature inspired methods and their industry applications—Swarm intelligence algorithms. *IEEE Trans. Ind. Inform.* **2017**, *14*, 1004–1015. [[CrossRef](#)]
- Mirjalili, S. Dragonfly algorithm: A new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems. *Neural Comput. Appl.* **2015**, *27*, 1053–1073. [[CrossRef](#)]
- Emambocus, B.A.S.; Jasser, M.B.; Mustapha, A.; Amphawan, A. Dragonfly algorithm and its hybrids: A survey on performance, objectives and applications. *Sensors* **2021**, *21*, 7542. [[CrossRef](#)] [[PubMed](#)]
- Gutin, G.; Punnen, A.P. *The Traveling Salesman Problem and Its Variations*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2006; Volume 12.
- Zhi, X.H.; Xing, X.; Wang, Q.; Zhang, L.; Yang, X.; Zhou, C.; Liang, Y. A discrete PSO method for generalized TSP problem. In Proceedings of the 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No. 04EX826), Shanghai, China, 26–29 August 2004; IEEE: Piscataway, NJ, USA, 2004; Volume 4, pp. 2378–2383.
- Yang, J.; Shi, X.; Marchese, M.; Liang, Y. An ant colony optimization method for generalized TSP problem. *Prog. Nat. Sci.* **2008**, *18*, 1417–1422. [[CrossRef](#)]
- Emambocus, B.A.S.; Jasser, M.B.; Hamzah, M.; Mustapha, A.; Amphawan, A. An enhanced swap sequence-based particle swarm optimization algorithm to solve TSP. *IEEE Access* **2021**, *9*, 164820–164836. [[CrossRef](#)]
- Yao, X.S.; Ou, Y.; Zhou, K.Q. TSP solving utilizing improved ant colony algorithm. *J. Phys. Conf. Ser.* **2021**, *2129*, 012026. [[CrossRef](#)]
- Wei, B.; Xing, Y.; Xia, X.; Gui, L. A novel particle swarm optimization with genetic operator and its application to tsp. *Int. J. Cogn. Inform. Nat. Intell.* **2021**, *15*, 1–17. [[CrossRef](#)]
- Rokbani, N.; Kumar, R.; Abraham, A.; Alimi, A.M.; Long, H.V.; Priyadarshini, I.; Son, L.H. Bi-heuristic ant colony optimization-based approaches for traveling salesman problem. *Soft Comput.* **2021**, *25*, 3775–3794. [[CrossRef](#)]
- Emambocus, B.A.S.; Jasser, M.B.; Amphawan, A. A discrete adapted dragonfly algorithm for solving the traveling salesman problem. In Proceedings of the 2021 Fifth International Conference on Intelligent Computing in Data Sciences (ICDS), Fez, Morocco, 20–22 October 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 1–6.
- Spanaki, K.; Karafili, E.; Sivarajah, U.; Despoudi, S.; Irani, Z. Artificial intelligence and food security: Swarm intelligence of AgriTech drones for smart AgriFood operations. *Prod. Plan. Control.* **2021**, *32*, 1–19. [[CrossRef](#)]
- Boveiri, H.R.; Khayami, R.; Elhoseny, M.; Gunasekaran, M. An efficient swarm-intelligence approach for task scheduling in cloud-based internet of things applications. *J. Ambient. Intell. Humaniz. Comput.* **2019**, *10*, 3469–3479. [[CrossRef](#)]
- Jahwar, A.; Ahmed, N. Swarm intelligence algorithms in gene selection profile based on classification of microarray data: a review. *J. Appl. Sci. Technol. Trends* **2021**, *2*, 1–9. [[CrossRef](#)]
- Brezočnik, L.; Fister, I.; Podgorelec, V. Swarm intelligence algorithms for feature selection: A review. *Appl. Sci.* **2018**, *8*, 1521. [[CrossRef](#)]
- Bacanin, N.; Bezdan, T.; Tuba, E.; Strumberger, I.; Tuba, M. Optimizing convolutional neural network hyperparameters by enhanced swarm intelligence metaheuristics. *Algorithms* **2020**, *13*, 67. [[CrossRef](#)]
- Emambocus, B.A.S.; Jasser, M.B. Towards an optimized dragonfly algorithm using hill climbing local search to tackle the low exploitation problem. In Proceedings of the 2021 International Conference on Software Engineering & Computer Systems and 4th International Conference on Computational Science and Information Management (ICSECS-ICOCSIM), Pekan, Malaysia, 24–26 August 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 306–311.
- Yang, J.; Qu, L.; Shen, Y.; Shi, Y.; Cheng, S.; Zhao, J.; Shen, X. Swarm intelligence in data science: Applications, opportunities and challenges. In *International Conference on Swarm Intelligence*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 3–14.
- Sun, W.; Tang, M.; Zhang, L.; Huo, Z.; Shu, L. A survey of using swarm intelligence algorithms in IoT. *Sensors* **2020**, *20*, 1420. [[CrossRef](#)]
- Paramanandham, N.; Rajendiran, K. Infrared and visible image fusion using discrete cosine transform and swarm intelligence for surveillance applications. *Infrared Phys. Technol.* **2018**, *88*, 13–22. [[CrossRef](#)]
- Janga Reddy, M.; Nagesh Kumar, D. Evolutionary algorithms, swarm intelligence methods, and their applications in water resources engineering: A state-of-the-art review. *H2Open J.* **2020**, *3*, 135–188. [[CrossRef](#)]

23. Soni, G.; Jain, V.; Chan, F.T.; Niu, B.; Prakash, S. Swarm intelligence approaches in supply chain management: Potentials, challenges and future research directions. *Supply Chain. Manag. Int. J.* **2018**, *24*, 107–123. [[CrossRef](#)]
24. Davendra, D. *Traveling Salesman Problem: Theory and Applications*; BoD—Books on Demand: Rijeka, Croatia, 2010.
25. Xu, Y.; Che, C. A brief review of the intelligent algorithm for traveling salesman problem in UAV route planning. In Proceedings of the 2019 IEEE 9th International Conference on Electronics Information and Emergency Communication (ICEIEC), Beijing, China, 12–14 July 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1–7.
26. Huang, S.H.; Huang, Y.H.; Blazquez, C.A.; Chen, C.Y. Solving the vehicle routing problem with drone for delivery services using an ant colony optimization algorithm. *Adv. Eng. Inform.* **2022**, *51*, 101536. [[CrossRef](#)]
27. Muren; Wu, J.; Zhou, L.; Du, Z.; Lv, Y. Mixed steepest descent algorithm for the traveling salesman problem and application in air logistics. *Transp. Res. Part Logist. Transp. Rev.* **2019**, *126*, 87–102. [[CrossRef](#)]
28. Foumani, M.; Moeini, A.; Haythorpe, M.; Smith-Miles, K. A cross-entropy method for optimising robotic automated storage and retrieval systems. *Int. J. Prod. Res.* **2018**, *56*, 6450–6472. [[CrossRef](#)]
29. Nedjatia, A.; Vizvárib, B. Robot path planning by traveling salesman problem with circle neighborhood: Modeling, algorithm, and applications. *arXiv* **2020**, arXiv:2003.06712.
30. Teng, L.; Li, H. Modified discrete firefly algorithm combining genetic algorithm for traveling salesman problem. *TELKOMNIKA (Telecommun. Comput. Electron. Control.)* **2018**, *16*, 424–431. [[CrossRef](#)]
31. Zhang, Z.; Han, Y. Discrete sparrow search algorithm for symmetric traveling salesman problem. *Appl. Soft Comput.* **2022**, *118*, 108469. [[CrossRef](#)]
32. Sopto, D.S.; Ayon, S.I.; Akhand, M.; Siddique, N. Modified grey wolf optimization to solve traveling salesman problem. In Proceedings of the 2018 International Conference on Innovation in Engineering and Technology (ICIET), Dhaka, Bangladesh, 27–28 December 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 1–4.
33. Liu, Y.; Liu, Q.; Tang, Z. A discrete chicken swarm optimization for traveling salesman problem. *J. Phys. Conf. Ser.* **2021**, *1978*, 012034. [[CrossRef](#)]
34. Akhand, M.; Ayon, S.I.; Shahriyar, S.; Siddique, N.; Adeli, H. Discrete spider monkey optimization for travelling salesman problem. *Appl. Soft Comput.* **2020**, *86*, 105887. [[CrossRef](#)]
35. Wang, K.P.; Huang, L.; Zhou, C.G.; Pang, W. Particle swarm optimization for traveling salesman problem. In Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No. 03EX693), Xi'an, China, 5 November 2003; Volume 3, pp. 1583–1585. [[CrossRef](#)]