New Hybrid Meta-heuristic to optimize the low exploitation of Discrete Adapted Dragonfly Algorithm for solving TSP

Atmane Ayoub Mansour Bahar, Kamel Soaid Ferrahi, Samy Salma, Mohamed Yacine Touahria Miliani, Souhail Abdelmouaiz Sadat, Karim Laouchedi

New Hybrid Meta-heuristic to optimize the low exploitation of Discrete Adapted Dragonfly Algorithm for solving TSP

Mansour Bahar Athmane Ayoub[1], Ferrahi Kamel Soaïd[1], Salma Samy[1], Sadat Souhail Abdelmouaiz[1], Laouchedi Karim[1], Touahria Miliani Mohamed Yacine[1]

[a]*5th year, Computer Systems students, École Nationale Supérieure d'Informatique (ex. INI), Oued Smar, 16309, Algiers, Algeria*

Abstract

Optimization problems drive the use of efficient algorithms. Swarm intelligence algorithms, like the Dragonfly Algorithm (DA), have proven effective. However, the original DA is not adapted for discrete problems like the traveling salesman problem (TSP). Existing adaptations showed limited effectiveness, especially for large instances. To overcome this, a new hybrid meta-heuristic algorithm is proposed. It combines the Discrete Adapted DA with Simulated Annealing Search, resulting in significantly improved efficiency. The algorithm demonstrates promising results on large-scale TSP instances from the TSPLIB data set.

Keywords: meta-heuristics, hybridisation, discrete adapted dragonfly, simulated annealing, tsp

1. Introduction

Optimization is crucial, adjusting processes or systems to offer the best solutions. Deterministic algorithms provide exact solutions but at a high computational cost, while heuristic algorithms offer efficient near-optimal solutions. Meta-heuristics combine diversification and intensification techniques for improved performance, with trajectory-based and population-based approaches being common [1].

Swarm intelligence algorithms, inspired by biological organisms, solve complex optimization problems through simple interactions [2]. The Dragonfly Algorithm (DA) is a superior swarm intelligence algorithm with different versions for continuous, discrete, and multi-objective optimization [3].

The Traveling Salesman Problem (TSP) requires finding the shortest route visiting all locations once. Exact algorithms become computationally demanding as the problem size increases [4]. Heuristic algorithms like PSO and ACO have succeeded in solving TSP using collective intelligence and optimization methods [5] [6].

Applying the DA to solve TSP is valuable, but adapting it for discrete problems like TSP has been challenging [3]. Previous attempts have shown limited effectiveness for large-scale TSP problems [7]. To overcome these limitations, we propose a novel discretized variant of the DA specifically tailored for TSP, aiming to enhance its performance.

Our study aims to improve the effectiveness of the adapted discrete dragonfly algorithm [7]. We propose a two-step approach for TSP: generating initial solutions using Annealing Search and applying a simulated annealing search to refine the solution. We evaluate our algorithm using benchmark TSP problems and demonstrate its superiority in terms of solution quality and cost compared to existing algorithms [8] [9] [10] [11] [12].

The paper is organized as follows: Section 2 provides an overview of the traveling salesman problem, the original dragonfly algorithm, the discrete adapted dragonfly algorithm, and the simulated annealing algorithm. Section 3 describes the proposed hybrid algorithm, including details about the data structure employed. Section 4 presents the results, comparison tests, and discussions. Finally, Section 5 concludes the paper and offers future perspectives.

2. Problem Formulation

In this section we will define the frame of the problem we are solving, i.e, the traveling salesman problem, as well as expose the algorithms that has been used to solve the problem before, and the algorithms we will hybridise.

2.1. Traveling Salesman Problem

The traveling salesman problem (TSP) is an NP-hard combinatorial optimization problem [13]. It involves finding the optimal route through a set of cities, visiting each city exactly once, and going back to the starting point, while minimizing the total travel cost. TSP has numerous real-world applications, including delivery services, routing, and robotic automated storage and retrieval systems [14].

2.2. Dragonfly Algorithm

The dragonfly algorithm [3] is a swarm intelligence-based optimization method. Inspired by the swarming behaviors of dragonflies, it differentiates between their static hunting trait of the swarms and the dynamic migratory trait of the swarms. The former reflects the exploration phase of optimization, where groups explore the search space by rapidly changing paths within a limited area. The latter represents the exploitation phase, as dragonflies migrate together towards a common direction over long distances, they are exploiting a solution. Figure

1 shows the difference between static and dynamic behaviour of dragonflies.

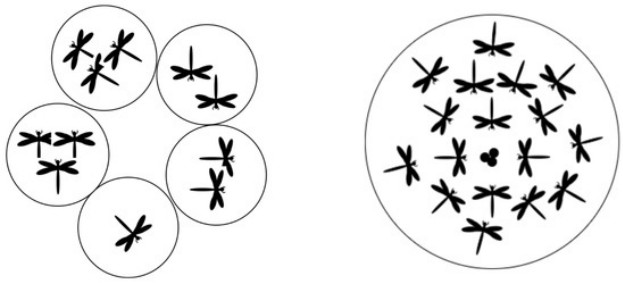

Figure 1: Dynamic versus static dragonfly swarms [9]

Dragonfly swarms exhibit five key principles illustrated in Figure 2.

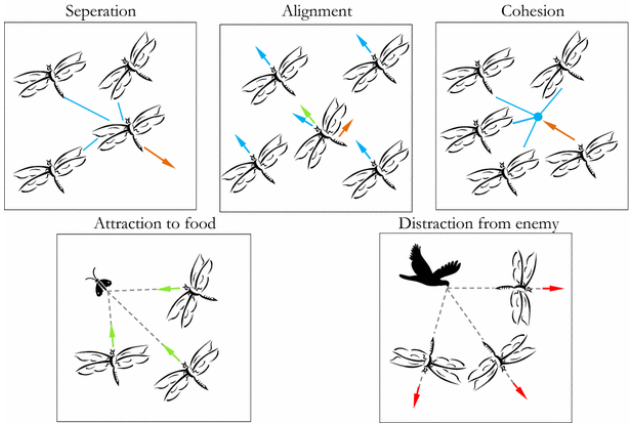

Figure 2: Five factors of dragonfly swarms movement [3]

These principles can be adapted to DA algorithm by Equation (1)-(5), and to strike a balance between exploration and exploitation, [9] introduces weight factors (s,a,c,f, and e) for the aforementioned principles. Furthermore, two dragonflies are considered neighbors if their distance is within the neighborhood radius. The algorithm of Dragonfly is shown in Algorithm 1

- separation, used to prevent its static collision with its neighbours. Where Xi, and Xj are the current dragonfly's position and the j-th neighbour's position respectively.

$$S_i = -\sum_j X_i - X_j \tag{1}$$

- alignment, matches a dragonfly velocity to that of its neighbours. Where Vj is the j-th neighbour's velocity

$$A_i = \frac{\sum_j V_j}{N} \tag{2}$$

- cohesion, is a dragonfly tendency towards the centre of the neighbourhood.

$$C_i = \frac{\sum_j X_j}{N} - X_i \tag{3}$$

- attraction to food, represents a dragonfly's attraction towards a food source. Where X+ is the food source's position

$$F_i = X^+ - X_i \tag{4}$$

- distraction from enemies, represents a dragonfly's repulsion from an enemy source. Where X- is the enemy's position

$$E_i = X^- + X_i \tag{5}$$

. To guide the updating of each dragonfly's position with these factors, three vectors are used:

- step vector, determines the direction of the movement. Where Si, Ai, Ci, Fi, and Ei 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 current iteration counter

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

- position vector, allows the movement in the search space by updating the actual position using the step vector to define the direction of movement

$$X_i^{t+1} = X_i^t + \Delta X_i^{t+1} \tag{7}$$

- levy vector, is a random walk employed to increase the stochasticity and exploration of the algorithm. Where d is the dimension of the position vectors

$$X_i^{t+1} = X_i^t + Levy(d) * X_i^t \tag{8}$$

Algorithm 1 Dragonfly Algorithm

```
Initialize the population's position randomly;
Initialize the step vectors;
while end conditions not reached do
    Calculate the objective values of all dragonflies;
    Update the food source and the enemy;
    Update the weights;
    Calculate the factors using (1)-(5);
    Update radius of the neighbourhoods;
    if dragonfly has one or more neighbours then
        Update step vector using (6);
        Update position vector using (7);
    else
        Update position vector using (8);
    end if
    Check and correct new positions based on upper and
    lower bounds;
end while
```

2.3. Related Works

2.3.1. Discrete Adapted Dragonfly Algorithm

The adapted discrete Dragonfly Algorithm (DA) [7] customizes the original DA algorithm for solving the Traveling Salesman Problem (TSP) using the following steps:

- A potential solution is represented by a TSP path, which serves as the position of a search agent.
- The equations for calculating the five factors (Separation, Alignment, Cohesion, Food Attraction, Enemy Dispersion), the step vector, and the position vector in DA are modified to be compatible with the path representation.
- The five factors, step vector, and position vector are computed using the swap sequence technique [15].

The pseudo-code for the adapted discrete DA [7] is provided and detailed in Algorithm 2.

Algorithm 2 Discrete Adapted Dragonfly Algorithm for TSP

```
Initialize the population's positions with random TSP paths;
Initialize the step vectors with random swap sequences;
while end conditions not reached do
    Calculate the objective values of all dragonflies;
    Update the food source and the enemy;
    Update the weights;
    Calculate the factors using (9)-(13);
    Update step vector using (14);
    Update position vector using (15);
end while
```

The Equations used in this discrete adapted algorithm are slightly different from the ones in original dragonfly algorithm since it is adapted to discrete problems such as TSP. The five previous factors as well as the step and position vectors are now as follows :

- separation factor, here Xi, Xj are the the current and the j-th neighbour agent's position, and N is the total number of neighbours. The O-minus operator is to produce a swap sequence, the O-plus operator is to merge two swap sequences, and Inv() is to inverse a swap sequence [38]

$$S_i = inv(\oplus_{j=1}^{N} X_i \ominus X_j) \tag{9}$$

- alignment factor, here Vavg represents the step vector of the neighbor with the closest fitness to the average fitness in the neighborhood.

$$A_i = V_{avg} \tag{10}$$

- cohesion factor, here Xavg is the position of the neighbour having the fitness closest to Vavg

$$C_i = X_{avg} \ominus X_i \tag{11}$$

- attraction to food factor, here Xf is the food source's position

$$F_i = X_f \ominus X_i \tag{12}$$

- distraction from enemies factor, here Xe is the enemy's position, and distraction() is a procedure that examines each city in Xe and Xi, and if a city is found to be in the same position in both paths, it replaces that city with a randomly chosen city different from the one replaced [9]

$$E_i = distraction(X_e, X_i) \tag{13}$$

- step vector of direction, here the O-plus operand represents the merging of two swap sequences. This operation combines all the swap operators from the first swap sequence with all the swap operators from the second swap sequence, resulting in a single swap sequence.

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

- position vector, here O-plus operand means that we apply the swap sequence delta(Xt+1) to the path Xt

$$X_i^{t+1} = X_t \oplus \Delta X_{t+1} \tag{15}$$

2.3.2. Simulated Annealing Search with Non-Monotonic Cooling for TSP

Simulated annealing is a popular meta-heuristic algorithm used for combinatorial optimization problems like the Traveling Salesman Problem (TSP). It mimics the annealing process in metallurgy, gradually exploring solutions with random changes based on a probability function. The algorithm adjusts a temperature parameter to escape local optima and explore the search space effectively. A variant, non-monotonic cooling [16], increases the acceptance of sub-optimal solutions after a certain number of iterations to avoid getting trapped as shown in Algorithm 3. The algorithm has three main equations, mainly to update the temperature, as described below :

- Acceptance probability, is set to accept sub-optimal solutions; where Delta is the new cost, and T is the current temperature

$$p = e^{\frac{-\Delta}{T}} \tag{16}$$

- Increasing of Temperature, this operation is meant to avoid local-optimums; where beta is the augmentation rate

$$T = T + \beta * T \tag{17}$$

- Decreasing of Temperature; this operation is meant to advance in the SA process, where alpha is the cooling rate

$$T = \alpha * T \tag{18}$$

Algorithm 3 Simulated Annealing with non-monotonic cooling for TSP

```
Set the initial tour and its cost as the best tour and the best cost;
while number of iterations not reached do
    Generate a neighbor of the current tour;
    Calculate the cost of the new tour;
    if the new tour has a lower cost then
        Update the current tour and its cost with the new tour and its cost;
        if the new cost is lower than the best cost then
            Update the best tour and its cost;
        else
            calculate the probability of acceptance using (16);
            if a randomly generated number is less than the acceptance probability then
                Update the current tour and its cost with the new tour and its cost;
                Reset the counter for iterations without improvement;
            else
                Increment the counter for iterations without improvement;
            end if
            Reset the counter for iterations without improvement;
        end if
    end if
    if number of iterations without improvement reaches the threshold then
        Update the temperature using (17);
    else
        Update the temperature using (18);
    end if
    if the temperature falls below the minimum temperature then
        Stop the local search;
    end if
end while
Add the starting city to the best tour to complete the cycle;
Return the best solution found;
```

3. Proposed solution

3.1. Hybridisation Scheme

Since the Adaptive Discrete DA (ADDO) is well suited for the exploration phase but much less for the exploitation, we decided to boost its exploitation with a specified meta-heuristic. The meta-heuristic chosen is Simulated Annealing with non-monotonic cooling because it gave us best performance for solving TSP than other meta-heuristics such as Ant Colony ACO, Genetic Algorithms GAO, and Taboo Search, as well as quickly converging. The hybridisation schemes followed is a **High Level Relay hybrid algorithm**, as shown in Figure 3

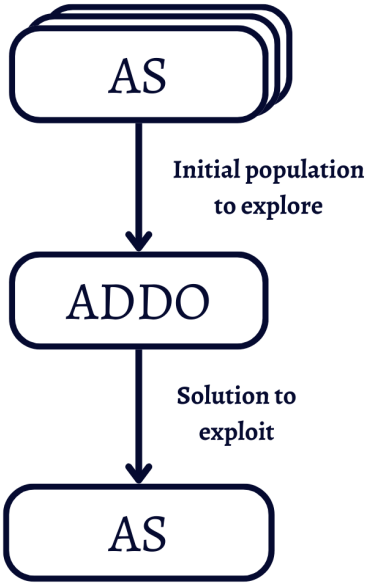

Figure 3: High Level Relay hybridisation scheme

3.2. Data Structures

To implement an algorithm that realises the hybridisation of discrete adaptive DA with non-monotonic cooling SA, we need to encode our graph and the solutions of the graph to solve the TSP, and thus, we need to define two main data structures to use.

3.2.1. Graph of Cities

One of the easiest way to represent a weighted non directed graph is to use a square Matrix where an index represents a city in the graph, and an element (i,j) represents the weight of the arc that connect the i-th city to the j-th city; the diagonal is set to zero to not loop over the same city. In TSP, we call this matrix a distance matrix.

3.2.2. A path of cities

A path in TSP is considered a solution, we can represent a solution using a list of cities to show the order of visit of each city, the index of each city is equivalent to the indexes of rows and columns of the distance matrix.

3.3. Description of the Hybridisation

The Hybrid Algorithm consists of three essential parts: We first initialize initial solutions using simulated annealing with non-monotonic cooling. These solutions are then passed to the adapted dragonfly algorithm for exploitation. The final result obtained is again passed through simulated annealing to further improve the solution. Initialising a first population with SA will give us a well chosen population that can converge quickly instead of a random initialisation; since it will reduce the search space of exploration to a smaller space that contains best solutions. We then take advantage of the exploration of discrete adapted DA to search for the best solution within the new search space. Once found; this solution is again exploited with non-monotonic cooling SA to optimise it even more and find a more

precise result. The Simulated Annealing with non-monotonic cooling is then executed (n+1) times, where n is the number of dragonflies. The corresponding algorithm for the hybridisation is shown in Algorithm 4.

Algorithm 4 Hybrid Discrete Adapted Dragonfly & Annealing Search with non-monotonic cooling for TSP

```
/* Generate an initial population of dragonflies with Simulated Annealing with non-monotonic cooling */
for individual in population do
    if it's in the first half of the population then
        Create a random initial tour and apply Simulated Annealing with non-monotonic cooling to it;
    else
        Create a tour using Nearest Neighbor and apply Simulated Annealing with non-monotonic cooling to it;
    end if
    Add the tour to the population;
end for
Initialize the best solution as the first tour in the population;
/* Explore the solutions with Discrete Adaptive Dragonfly */
while number of iterations not reached do
    for dragonflies in the population do
        Calculate the objective values of all dragonflies;
        Update the food source and the enemy;
        Update the weights;
        Calculate the factors using (9)-(13);
        Update step vector using (14);
        Update position vector using (15);
        Apply local search to the updated position;
        Update the current dragonfly in the population with the new position;
        if the new position has a smaller distance than the best solution then
            Update the best solution with the new position;
        end if
    end for
end while
Return the best solution found by Discrete Adapting Dragonfly;
/* Exploit the solution with Simulated Annealing */
Apply Simulated Annealing on the solution found to optimize it more;
Return the solution found by Simulated Annealing;
```

The objective function used in TSP is the cost of a path, the cost is calculated by adding the distances of each two successive cities, we add to it the distance from the last city to the starting city in order to close the cycle. As for the positions of dragonflies, they are updated with the same way in the original adapted discrete DA [7] using the Equations (9)-(13) shown before to calculate the five factors of separation, alignment, cohesion, food attraction, and enemy dispersion, as well as Equations (14)-(15) to update the position of each dragonfly.

4. Tests and Results

After discussing the technical implementation details of our solution, we proceed to the testing phase. This part of the article aims to evaluate the algorithm's performance by conducting various tests. These tests not only enhance the solution's quality by optimizing the technique's parameters for a specific data set but also compare its efficiency and execution time to other techniques introduced in previous studies addressing the traveling salesman problem.

4.1. Testing Plan

For the testing plan we will use 20 different TSP instances which are (burma14, ulysses16, gr17, gr21, ulysses22, bays29, swiss42, berlin52, st70, pr76, gr96, kroA100, lin105, pr107, gr229, gr666) in order to asset the performance of our hybridisation scheme and compare it to other well-known heuristics and meta heuristics (Taboo Search, Simulated Annealing, Ant Colony Optimization and Genetic Algorithm). For the performance we will use the metric with the formula:

$$P = \left(1 - \frac{D - D_c}{D_c}\right) 100\% \tag{19}$$

Where P is the performance to be calculated, D is the best found distance, and D_c is the best known distance for the given instance.

4.2. Execution Environment

The tests were performed on a i5 11th generation CPU with 16Gb of RAM. The benchmarks used are 20 data sets of different complexity, all taken from TSPLIB.

4.3. Parameter Tuning

In this program, we have two categories of parameters, ones for the adapted discrete DA and ones for the non-monotonic cooling SA.

In our case, we want to give priority to the exploration of DA, thus in this paper, we will focus on finding the optimal number of dragonflies and number of iterations to do. The parameter tuning shown below was performed on the swiss42 and gr17 data sets respectively for the number of iterations and the number of dragonflies to use in the exploration phase i.e in the adapted discrete DA part.

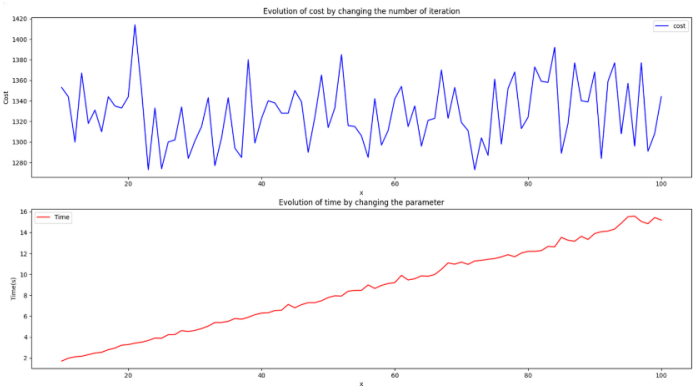

Figure 4: Variation of cost and time with the number of DA iterations

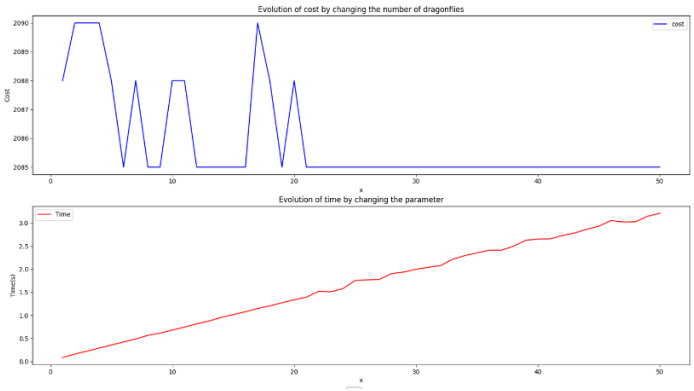

Figure 5: Variation of cost and time with the number of dragonfly

From Figures 4 and 5 we can conclude; firstly, that the number of iterations doesn't affect the performances much, and that's because of the good search space that the SA gave us at first, thus, iterating much over a reduce space won't be much beneficial. And secondly, that the number of dragonflies help find better solutions, but a larger population means greater computational costs. However, for 15 dragonflies, we can find optimal solutions with lower costs. For the weights of the five factors of DA, since they are constants we can give them values depending on the results we would like to have, we have chosen to use these values :

- Attraction factor, we want our dragonflies to follow the best solution, but yet without getting to close in order to give priority to exploration the reduces search space, so we initialize it at 0.5
- Dispersion factor, we want to escape all the bad solutions in order to avoid local optimums and give more importance to exploration, so we initialize it at 0.8
- Randomness factor, Factor of changing a given solution by randomly swapping two cities given a probability (can generate a worst solution) = 0.1
- Local search probability, is the probability of doing a local search (with swapping the cities, as in original adapted discrete DA), we initialize to a relatively small value to give more priority to exploration than exploitation with DA, we initialize it at 0.05

As for the parameters of SA, the optimal parameters for solving TSP are already known from previous tests we made on the performance of SA for solving TSP, we have found the optimal values of parameters as follows :

- Initial Temperature = 1000
- Minimal Temperature = 1e^12
- Cooling rate alpha = 0.99
- Augmentation rate beta = 0.1
- Number of iterations without augmentation = 1000
- Number of iterations of the algorithm = 20000

4.4. Comparative Study

In this comparative study, we aim to evaluate and compare the performance of our solution to various algorithms on different data sets mentioned before. The algorithms under investigation are selected based on their relevance and applicability to the problem domain. The data sets used in the study are carefully chosen to represent diverse characteristics and complexities. To ensure a fair comparison, we will employ rigorous experimental design and statistical analysis techniques. The evaluation process will involve running the selected algorithms on each data set, collecting and analyzing the results, and drawing meaningful conclusions based on the observed performance differences. The comparisons are all detailed in the tables below. We start with Table 1 that shows the percentage of precision of our hybrid algorithm in finding the best solution over 20 known TSPLIB data sets. Then, the Table 2 that shows a contrast between known meta heuristics in finding the optimal cost of our test data sets. And finally, a contrast in the execution time of our hybrid algorithm ADDO-SA with the other algorithms. By conducting this study, we seek to gain insights into the strengths and weaknesses of our solution across different data set scenarios.

Table 1: Performance for the hybridisation for 20 TSP Instances

Instance	Cost	Best Know Cost	Performance	Time Execution
burma14	3323	3323	100%	0.14
ulysses16	6859	6859	100%	0.17
gr17	2085	2085	100%	0.18
gr21	2707	2707	100%	0.27
ulysses22	7013	7013	100%	0.29
bays29	2020	2020	100%	0.78
swiss42	1273	1273	100%	1.54
berlin52	7542	7542	100%	2.93
st70	696	675	97.04%	8.57
pr76	109625	108159	98.64%	12.28
gr96	57127	55209	96.52%	33.35
kroA100	22631	21282	93.66%	35.59
lin105	15278	14379	93.74%	42.23
pr107	45164	44303	98.05%	50.39
gr120	7276	6942	96.05%	218.45
ch150	6875	6528	94.67%	219.65
rat195	2396	2323	96.85%	648.20
kroA200	31490	29368	92.77%	872.71
gr229	136486	134602	98.6%	1263.6
gr666	297005	294358	99.1%	1612.71

Table 2: Comparison of cost Algorithms for 20 TSP Instances

Instance	ADDO	EADDO	SA	TS	ACO	GA	ADDO+SA	Best Known Cost
burma14	4562	3323	3323	3323	3841	3323	3323	3323
ulysses16	9665	6922	6875	7276	7943	7063	6859	6859
gr17	4722	2135	2088	2218	2178	2098	2085	2085
gr21	6620	2758	2707	3001	3003	2997	2707	2707
ulysses22	12198	7183	7316	7013	7287	7287	7013	7013
bays29	5752	2095	2051	2169	2134	2452	2020	2020
swiss42	2834	1407	1416	1349	1401	1934	1273	1273
berlin52	22205	7863	7809	8455	8181	11844	7542	7542
st70	3410	748	784	784	770	1584	696	675
pr76	150781	119486	119842	122156	131607	290339	109625	108159
gr96	81007	63597	63046	65942	63947	178902	57127	55209
kroA100	191387	23713	25648	30840	24698	89641	22631	21282
lin105	36480	15832	17866	21608	16935	60317	15278	14379
pr107	62752	47707	45825	55106	46060	227314	45164	44303
gr120	50021	8006	8175	11975	8352	27069	7276	6942
ch150	52814	7634	7579	27474	7075	32594	6875	6528
rat195	4030	2824	2737	2860	2557	14222	2396	2323
kroA200	373938	33764	34940	150993	34543	214020	31490	29368
gr229	179819	160238	158230	168298	158680	874123	136486	134602
gr666	423710	325558	365086	392681	353146	4120857	297005	294358

Table 3: Comparison of time execution of Algorithms for 20 TSP Instances

Instance	**ADDO**	**EADDO**	**SA**	**TS**	**ACO**	**GA**	**ADDO+SA**
burma14	0.13	0.11	0.017	0.018	0.031	0.19	0.14
ulysses16	0.14	0.185	0.0180	0.025	0.035	0.21	0.17
gr17	0.148	0.28	0.0180	0.03	0.039	0.22	0.18
gr21	0.16	0.45	0.019	0.059	0.055	0.25	0.27
ulysses22	0.17	0.52	0.02	0.067	0.058	0.26	0.29
bays29	0.21	1.34	0.024	0.156	0.093	0.34	0.78
swiss42	0.30	4.92	0.027	0.47	0.16	0.49	1.54
berlin52	0.36	10.53	0.031	0.76	0.24	0.65	2.93
st70	0.50	35.92	0.039	0.99	0.42	0.90	8.57
pr76	0.53	49.07	0.041	1.03	0.48	1.02	12.28
gr96	0.72	101.2	0.05	1.32	0.77	1.30	17.65
kroA100	0.75	143.49	0.051	1.35	0.83	1.34	25.59
lin105	0.79	180.6	0.052	1.40	0.91	1.40	42.23
pr107	155	350.4	0.053	0.82	0.94	1.44	50.39
gr120	0.96	534.4	0.059	1.59	1.15	1.71	118.45
ch150	1.39	697	0.07	1.94	1.77	2.33	189.65
rat195	2.32	978	0.087	2.51	2.88	3.31	348.20
kroA200	2.42	1365	0.088	2.63	2.99	3.41	472.71
gr229	3.17	2143	0.093	2.99	3.88	4.09	663.61
gr666	52.94	3945	0.32	10.40	31.40	24.88	712.71

4.5. Discussion of the results

Based on the obtained results, it is evident that our solution consistently outperformed other meta heuristics in each data set, demonstrating a consistently high performance across all instances. The worst performance observed among the compared algorithms was 92.77%, highlighting the superiority of our solution.

Furthermore, our solution consistently achieved the best-known solution for small instances, indicating its effectiveness in finding optimal or near-optimal solutions. This showcases the strength of our algorithm in handling and solving complex optimization problems.

In contrast, the Adaptive Discrete Dragonfly Optimization algorithm (ADDO) exhibited poor performance compared to other algorithms. To address this issue, the Enhanced Adaptive Discrete Dragonfly Optimization algorithm (EADDO) was developed to enhance the former algorithm's performance for exploitation. However, the execution time of EADDO became excessively long for large instances, rendering it impractical for real-world applications.

This is where our algorithm proves advantageous. By combining the exploration capabilities of the Dragonfly algorithm with the exploitation and execution efficiency of Simulated Annealing, our algorithm achieves exceptional results within reasonable time frames. It strikes a balance between exploration and exploitation, allowing for efficient and effective optimization on large instances.

Overall, the comparative analysis highlights the superiority of our algorithm in terms of performance and efficiency compared to other meta heuristics. Its ability to provide excellent results for a wide range of instances, including both small and large problems, positions it as a promising solution for various real-world optimization challenge

5. Conclusion and Future Work

Swarm intelligence algorithms excel in solving complex optimization problems by leveraging the collective behavior of search agents. The dragonfly algorithm, drawing inspiration from dragonflies' swarming behaviors, surpasses other swarm intelligence algorithms across different applications. Given its success, utilizing the dragonfly algorithm for the traveling salesman problem (TSP) is a promising endeavor. Although a binary version of the algorithm has been introduced, adapting it for discrete problems like TSP remains a challenge. In our prior research, we proposed a tailored discrete adapted dragonfly algorithm for TSP. However, its effectiveness on larger TSP problems is currently limited and calls for further improvement. In this paper, we propose a new hybrid meta-heuristic that combines the adapted discrete dragonfly algorithm with non-monotonic cooling annealing search. Our algorithm improves the effectiveness of the adapted discrete DA presented in [7] by enhancing its exploitation phase through SA initialization. Experimental results on different TSP instances demonstrate superior solutions compared to discrete DA, annealing search SA, Ant Colony ACO, Genetic Algorithms GAO, Taboo Search, and Enhanced Adapted Discrete DA [9]. Our algorithm also achieves optimal or near-optimal solutions on various benchmark TSP problems.

For future work, the proposed algorithm can be parallelised with threads or GPU in order to make it converge and obtain results quicker. We also look further to use the new algorithm in real-world applications in scheduling and network routing.

Acknowledgements

Thanks to Pr. BESSEDIK Malika & Mr. KECHID Amine for their constructive comments and suggestions, and for giving us this opportunity to do this research through the Module of Combinatorial Optimisation.

References

[1] Yang, X.S. Nature-Inspired Optimization Algorithms; Academic Press: Cambridge, MA, USA, 2020.

[2] Slowik, A.; Kwasnicka, H. Nature inspired methods and their industry applications—Swarm intelligence algorithms. IEEE Trans. Ind. Inform. 2017, 14, 1004–1015.

[3] 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.

[4] Gutin, G.; Punnen, A.P. The Traveling Salesman Problem and Its Variations; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2006; Volume 12.

[5] 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.

[6] Yang, J.; Shi, X.; Marchese, M.; Liang, Y. An ant colony optimization method for generalized TSP problem. Prog. Nat. Sci. 2008, 18, 1417–1422.

[7] 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.

[8] Yang, J.; Shi, X.; Marchese, M.; Liang, Y. An ant colony optimization method for generalized TSP problem. Prog. Nat. Sci. 2008, 18, 1417–1422.

[9] Emambocus, B. A. S., Jasser, M. B., Amphawan, A., & Mohamed, A. W. (2022). An Optimized Discrete Dragonfly Algorithm Tackling the Low Exploitation Problem for Solving TSP. Mathematics, 10(19), 3647.

[10] Yao, X.S.; Ou, Y.; Zhou, K.Q. TSP solving utilizing improved ant colony algorithm. J. Phys. Conf. Ser. 2021, 2129, 012026.

[11] 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.

[12] 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.

[13] Gutin, G.; Punnen, A.P. The Traveling Salesman Problem and Its Variations; Springer Science& Business Media: Berlin/Heidelberg, Germany, 2006; Volume 12.

[14] 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.

[15] 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.

[16] A Comparison of Cooling Schedules for Simulated Annealing (Artificial Intelligence). (n.d.). What-When-How.com.