International Journal of Engineering Trends and Technology
ISSN: 2231–5381 / https://doi.org/10.14445/22315381/IJETT-V73I8P126

Volume 73 Issue 8, 303-311, August 2025
© 2025 Seventh Sense Research Group®

Original Article

Modified Sailfish Optimizer with Opposition Based Learning to Optimize Traveling Salesman Problem

Prayoga Yudha Pamungkas

Industrial Engineering Department, Faculty of Engineering, Bina Nusantara University, Jakarta, Indonesia.

[1]*Corresponding Author : prayoga.pamungkas@binus.ac.id*

Received: 22 February 2025 Revised: 01 July 2025 Accepted: 31 July 2025 Published: 30 August 2025

Abstract - *The newly proposed Modified Sailfish Optimizer (MSFO) is designed to address the issue of the Traveling Salesmen Problem (TSP) effectively. In this modification, Opposition-Based Learning (OBL) is hybridized to enhance population diversity and speed up convergence. A simplified attack power mechanism improves the exploration-exploitation balance, which helps MSFO escape local optima and find better solutions. Experimental results on benchmark TSP instances prove that MSFO outperforms among the compared algorithms. It achieves optimal solutions with zero deviation and is better than the compared algorithms. MSFO can discover the minimum possible tours that other algorithms cannot reach, and the best solution. These results confirm that the proposed modifications significantly improve the effectiveness of the original Sailfish Optimizer.*

Keywords - *Modified Sailfish Optimizer, metaheuristics, Opposition based learning, Traveling Salesman Problem, Discrete optimization.*

1. Introduction

Delivery and logistics systems underlying the contemporary global economy are key to ensuring the effective delivery of goods to customers. From the perspective of companies, saving on logistics is essential in terms of profit maximization along the supply chain. One of the challenges to logistics is the optimization of routes by reducing transportation distance. Perhaps one of the most significant combinatorial optimization issues in logistics is the Traveling Salesman Problem (TSP), which attempts to determine the optimal tour a salesperson could take to stop at a group of cities and then return to the original city such that the overall travel distance is kept to a minimum. The TSP is extensively studied in optimization and serves as a benchmark for evaluating various optimization methods [1]. Since the TSP is at the core of most industries, such as manufacturing, telecommunications, and logistics, its optimization is an important research area. The problem has been identified as NP-hard [2], in that the number of possible routes increases exponentially with the number of cities, and thus brute-force approaches cannot be applied to large instances. Therefore, an efficient solution to the TSP is important, especially for practical uses where computational viability is paramount. The classical methods for solving TSP are Linear Programming (LP), enumeration, and branch-and-bound. While linear programming is able to cast the TSP as a constrained optimization problem with linear constraints, it is not very effective in dealing with the combinatorial aspects of the problem and hence performs poorly for large data sets. Enumeration techniques, which investigate all possible paths and identify the optimal path, yield precise solutions at a high computational cost that increases exponentially in the number of cities. Likewise, branch-and-bound techniques assist in pruning bad solutions but encounter scalability problems with increasing problem sizes. While exact algorithms can guarantee optimal solutions [3], they are often unsuitable for large-scale problems due to their computational intensity. As a result, heuristics and metaheuristics have become more prevalent in solving the TSP, as they can often find good solutions more quickly. Heuristic methods, such as the nearest neighbor and greedy algorithms, provide fast solutions but do not guarantee optimality. These approaches are often used in practical scenarios where a good, fast solution is more important than an exact one. Metaheuristic-based techniques have become more popular in recent years for addressing the TSP and other difficult optimization problems. These algorithms, analogues of biological and physical system processes, apply genetic evolution, simulated annealing and particle swarm optimization to large search spaces. Metaheuristics contain randomization together with local search mechanisms, that allow them to escape local minima and to explore broader solution domains. Metaheuristics are useful for real-world problems, and although they do not always guarantee an optimal solution [4], they frequently yield a nearly optimal solution in a much shorter time than traditional exact methods. Inspired by sailfish hunting behavior, the Sailfish Optimizer (SFO) is a new member of the metaheuristic family [5].

By mimicking the sailfish's movement patterns during hunting, the SFO algorithm balances exploration and exploitation to efficiently search for optimal solutions. Initial studies show that the SFO algorithm outperforms other traditional metaheuristics in solving the TSP, providing more efficient solutions. However, as with most metaheuristics, SFO may converge to local optima, meaning it may not always find the global optimum, particularly in larger or more complex instances of the problem. Even though metaheuristics like SFO have some benefits, they can be hard to use because they are expensive to compute, especially when dealing with big problems. As the problem gets bigger, the search space gets bigger, and it takes more computing power to find good solutions. Furthermore, there is an inherent tradeoff between solution quality and computational resources, making it essential to strike a balance between the two. Hybridizing multiple metaheuristics can help mitigate some of these challenges by improving convergence speed and solution quality [6]. The Sailfish Optimizer has been implemented in wireless sensor networks [7], oral cancer classification [8], feature selection to enhance model accuracy [9], nutrient deficiency detection for accurate diagnosis [10], Apple Leaf Disease Detection [11], and salient feature determination [12].

Table 1. Research gap in metaheuristics on the traveling salesman problem

No	Author	Problem	Objective	Decision Variable	Approach	Methods
1	Shadravan [5]	Engineering (welding, gear, I-beam)	Minimize cost	Weld thickness & length, beam height & thickness	Meta-heuristics	Sailfish Optmizer
2	Kumar [7]	Wireless Sensor Network	Minimize node power consumption	Cluster heads position	Meta-heuristics	Sailfish Optimizer
3	Tomazella [13]	Flowshop Scheduling Problem	Minimize time completion	Machine assignment, operation sequence, starting time	Exact	Branch and bound
4	Rajarajeswari [14]	Traveling Salesman Problem	Minimize total distance	Path x_{ij} to take	Exact	Branch and bound
5	Yang [15]	Traveling Salesman Problem	Minimize total distance	Path x_{ij} to take	Meta-heuristics	Genetic Algorithm, Shuffled Frog Leaping
6	Li [16]	Traveling Salesman Problem	Minimize total distance	Path x_{ij} to take	Meta-heuristics	Particle Swarm, Discrete Artificial Bee

Table 1 illustrates how different researchers have applied both metaheuristic and exact methods to address various optimization problems, especially the issue of the salesman traveling. Genetic Algorithm, Shuffled Frog Leaping, Particle Swarm Optimization, and Discrete Artificial Bee Colony are some of the most common metaheuristic algorithms. They can find good solutions in less time than other algorithms.

However, these methods often do not achieve the same level of solution accuracy as exact algorithms like branch and bound, which are designed to find the best possible solution but require significantly more processing time. This results in a tradeoff in the performance of efficient but less effective metaheuristics compared to the exact algorithm, which is accurate but not practical for large-scale problems. Furthermore, a large number of classic metaheuristics are not well-adapted to the discrete character of problems such as the TSP. Hence, the requirement persists to enhance the metaheuristic methods, such that the metaheuristic methods could get quality solutions without compromising the computation speed. Filling this gap is crucial in solving real-world problems that demand both high accuracy and fast computation. A newer version of the Sailfish Optimizer is showing a lot of promise for combinatorial problems. Its search steps were redesigned so it can look more widely across possible answers without slowing down too much. As a result, it often gets very close to the best-known answers that exact solvers find, but it uses far less computing power. That mix of speed and accuracy makes it a good fit for the Traveling Salesman Problem. Early results are encouraging, and there is room to improve the method even more. Over time, it could help close the distance between quick metaheuristics and exact, highly accurate algorithms. The TSP is tough and shows up in many real applications, so people use many tools to tackle it: brute-force enumeration, linear programming, branch-and-bound, plus heuristics and metaheuristics. The Sailfish Optimizer is one of the newer options that can produce strong tours, though it still faces issues like getting stuck in local optima or running longer on big cases. With thoughtful tweaks, however, it can be a practical choice for solving the TSP in logistics and other optimization settings.

2. Related Works

This study examines a wide range of work on the Traveling Salesman Problem and methods based on the Sailfish Optimizer.

2.1. Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is about finding the best way to visit all the cities while keeping the total distance traveled as short as possible [17]. Dantzig [18] came up with the mathematical model for the TSP.

2.1.1. Objective Function

TSP seeks the shortest tour that visits every city once, as shown in Equation 1.

$$\min \sum_{i=1}^{N} \sum_{j=1}^{N} c_{ij}\, x_{ij} \tag{1}$$

2.1.2. Constraints

Standard formulations add constraints to ensure each city is visited exactly once and that the tour respects feasible travel rules. These restrictions narrow the search space and guarantee that any solution corresponds to a valid route.

$$\sum_{i=1}^{N} x_{ij} = 1,\ j = 1,2,\dots,N \tag{2}$$

$$\sum_{j=1}^{N} x_{ij} = 1,\ i = 1,2,\dots,N \tag{3}$$

$$x_{ij} = \{0,1\},\ i,j = 1,2,\dots,N \tag{4}$$

In this model, cij denotes the distance (or cost) from city i and city j, and xij indicates whether that route is used. The objective in Equation (1) minimizes the total distance traveled.

Constraints (2) and (3) require exactly one departure from each city and exactly one arrival to each city, ensuring every location is visited once. Finally, equation (4) makes xij=1 if the route from i to j is chosen and xij=0 otherwise.

2.2. Metaheuristics

Metaheuristics are high-level strategies for solving difficult optimization problems when traditional exact methods are too costly to run [19]. These algorithms are based on natural or physical processes, like genetic evolution, simulated annealing, or swarm intelligence, and aim to explore large, multidimensional solution spaces to balance exploration and exploitation.

Unlike exact optimization techniques, which guarantee optimal solutions under certain conditions, metaheuristics typically focus on finding near-optimal solutions within a computationally feasible timeframe, making them particularly suitable for real-world applications. By incorporating both randomization and structured search mechanisms, metaheuristics are capable of escaping local optima and exploring a wider array of potential solutions. Although they do not guarantee global optimality, their ability to quickly find good solutions has led to their widespread use in diverse fields, including logistics, machine learning, and network optimization.

2.3. Sailfish Optimizer

2.3.1. Source of Inspiration

With a top speed of 100 km/h, sailfish is known as one of the fastest fish in the ocean [20]. Swift movements of the sardines make it challenging for the sailfish, which uses its rostrum to either slash at multiple sardines or destabilize a single one. Although most attacks do not immediately result in a catch, repeated strikes leave more sardines injured and separated from the school, making them easier to capture. This coordinated hunting behavior, similar to pack hunting in wolves, sees sailfish changing their body color and fin positions to signal each other and avoid injuring fellow hunters. The alternation strategy in sailfish hunting inspired the creation of the Sailfish Optimizer (SFO) algorithm, which models these behaviors for optimization problems.

2.3.2. Mechanism

The SFO algorithm is motivated by the group hunt behavior of sailfish, which take turns attacking a sardine school [1]. Here, the SHO uses two populations, one for the prey and another for the predators, to simulate the group hunting approach [5]. The attack technique used by the algorithm is aimed at destroying the collective defense of prey. Third, the movement of prey in the search space is updated, attracting the hunter to the appropriate hunt and increasing its fitness.

Initialization

One population-based metaheuristic method is the SFO, which is used to represent candidate solutions; in contrast, the variables of the problem are represented by the sailfish's location in the search space. The solution's initial population is produced at random across the solution space.. The sailfish are free to explore 1, 2, 3 or higher-dimensional spaces, each with a position vector.

Elitism

Sometimes, while updating the search agents' positions, some of the best solutions are lost by chance; a reduction in the new position of the search agents with respect to the previous location may be generated unless an elitism strategy is used. The elite get the best solution(s) and hand them down to the next generation. Every iteration of the SFO process saves the position of the top sailfish, which is referred to as an elite solution. The best sailfish is the "fittest" solution found thus far, and it has an indirect impact on how the sardines move and accelerate during hunting.

As mentioned above, sardines in between are also injured by the rostrum of the sailfish when the former group-hunt. Therefore, the location of the harmed sardine in each iteration is memorized and assigned to be the best target for assisting hunting from the sailfish, respectively. The positions of the elite sailfish and the injured sardine with the highest fitness values at the ith iteration are $X_{eliteSF}^{i}$ and $X_{injuredS}^{i}$. They are crucial for helping improve the global performance of the

SFO method and avoiding exploration of the bad solutions that have been discarded previously.

Attack-Alternation Strategy

Sailfish enhance their hunting success rate through a temporally coordinated attack strategy [21]. They chase and herd their prey, where other hunters are around the prey school, even if they do not make a direct contribution. The Sailfish Optimization (SFO) algorithm mimics this group hunting strategy, particularly the attack-alternation approach. During the exploration phase, search agents scan a large area of the search space to find possible solutions that need more work. Sailfish do not limit their attacks to specific directions, such as up-down or left-right. Otherwise, they can strike from all directions within a shrinking circular area. As a result, sailfish dynamically update their positions within a spherical region centered around the best solution. At the *i*th iteration, the updated position of the sailfish, X^i_{newSF}, is determined as follows:

$$X^i_{newSF} = X^i_{eliteSF} - \lambda_i \times \left(rand(0,1) \times \left(\frac{X^i_{eliteSF} + X^i_{injuredS}}{2} \right) - X^i_{oldSF} \right) \quad (5)$$

X^i_{newSF} represents the position of the elite sailfish identified up to the current iteration, $X^i_{injuredS}$ denotes the best position of the injured sardine found so far, and X^i_{oldSF} is the current position of the sailfish. The term $rand(0,1)$ refers to a randomly generated number between 0 and 1, while λ_i is a coefficient calculated at the *i*th iteration, defined as follows:

$$\lambda_i = 2 \times rand(0,1) \times PD - PD \quad (6)$$

PD is prey density, which is the number of prey in each iteration. Given that the number of prey decreases during sailfish group hunting, the PD parameter is one of the main factors that influences the updating of the sailfish position around prey school. The adaptive formula of this parameter can be expressed as:

$$PD = 1 - \left(\frac{N_{SF}}{N_{SF} + N_S} \right) \quad (7)$$

N_{SF} and NS_S indicate how many sardines and sailfish there are in each algorithm cycle, respectively. Since the initial number of sardines is typically greater than that of sailfish, N_{SF} is computed as $N_S \times PP$, where the percentage of the sardine population that makes up the original sailfish population is indicated by PP.

Hunting & Catching Prey

At the start of the hunt, sailfish are highly energetic and capable of pursuing prey, while sardines are not yet exhausted or injured. As a result, sardines can maintain high escape speeds and demonstrate strong manoeuvring abilities. Over time, however, the strength of the sailfish's attacks decreases as the hunt progresses. Due to the relentless and frequent attacks, the prey's energy reserves diminish, and they might become less able to recognize directional clues about the sailfish's location. The prey school's ability to flee is hampered by this impairment. After a while, the sailfish's bill strikes the sardines, separating them from the others and swiftly capturing them. In order to replicate this process, each sardine has to adjust its position at each iteration according to the sailfish's best position at that moment and the force of its attack. At the ith iteration of the SFO algorithm, the sardine's new location of $X^i_{new\_S}$, can be expressed as:

$$X^i_{new\_S} = r \times \left(X^i_{elite\_SF} - X^i_{old_S} + AP \right) \quad (8)$$

$X^i_{elite\_SF}$ represents the best position of the elite sailfish identified up to the current iteration, $X^i_{old_S}$ denotes the current position of the sardine, *r* is a random number between 0 and 1, and *AP* indicates the Attack Power of the sailfish at each iteration, which is calculated as follows:

$$AP = A \times \left(1 - (2 \times Iteration \times \varepsilon) \right) \quad (9)$$

A and ε are coefficients used to linearly reduce the attack power value from A to 0. By the last stage of feeding, the injured fish that has broken away from the school is overwhelmed. It is also supposed in the present algorithm that a sardine catches its prey if it gets more fit than the sailfish to which it corresponds. Under these conditions, the position of the sailfish is updated to that of the last captured sardine to facilitate searching for new prey. We postulate the following equation to describe this process:

$$X^i_{SF} = X^i_s \quad if \quad f(S_i) < f(SF_i) \quad (10)$$

2.4. Opposition Based Learning

OBL is an optimization method that pairs every candidate with its opposite and takes both of them into account. Taking into account both sides of the space improves navigation and can shrink the distance to an optimal or near-optimal solution [22]. The opposite is taken as the complementary point to the original one in order to achieve more complete coverage of the search space and minimize the risk of local optima. This mechanism has yielded dividends on a number of approaches, with strong outcomes on evolutionary algorithms due to the increased diversity in the search. As a result, OBL has become a powerful tool for improving the global search ability and overall strength of optimization algorithms in areas with complicated problems.

2.4.1. Source of Inspiration

The concept of opposition first appeared in ancient Chinese philosophy, and the figure of the Yin-Yang symbol [23] is shown in Figure 1. This is a representation of the dual concept where white and black are Yin (negative / passive /

feminine / dark) and Yang (positive / active / male / light), respectively. The opposition concepts, such as fire (hot and dry) versus water (cold and wet) and earth (cold and dry) versus air (hot and wet), were also explained in the Greek classics of natural pattern. Wet, dry, hot, and cold are all characteristics of natural entities, as are the opposites [23]. It seems that contrast is used to convey the idea of many things or circumstances in the world. Indeed, when the opposition notion is available, the description becomes much simpler for various objects. Contrary pairs like east and west, and south and north, cannot be established by themselves, but can be explained.

Fig. 1 The opposition concept has its origins in the Yin-Yang symbol

3. Modified Sailfish Optimizer

3.1. Opposition based Learning

Initial solutions of heuristic optimization algorithms are typically generated randomly when attempting to find the optimal solution for a given problem. To make it more likely that the best solution will be found, both a random solution and its opposite can be made at the same time. This encourages a more thorough search of the solution space. [23]. Let x be a real number within the range of a and b(where a ≤ b), the opposite of the number. $\breve{x}$ of x can be presented below.

$$\breve{x} = a + b - x \tag{11}$$

The extended definition for higher dimensions is provided in [24]. In a D-dimensional search space, let $P = \{x_1, x_2, ..., x_D\}$ represent a point, where each coordinate x_i $(1 \leq i \leq D)$ falls within the range of a_i and b_i. The corresponding opposite point $\breve{P} = \{\breve{x}_1, \breve{x}_2, \dots, \breve{x}_D\}$ can be determined using the given Equation 12.

$$\breve{x}_i = a_i + b_i - x_i \tag{12}$$

From Equations (11) and (12), it can be observed that generating both the random guess and its opposite concurrently increases the likelihood of identifying potential promising regions where the optimal solution might be found.

3.2. Exploration Phase

In the MSFO, the search is divided into two main phases: exploration and exploitation. The exploration mode is enabled if the Attack Power (AP) ≥ 0.5 at the initial optimization stage. In this phase, the global search is enhanced, in which sailfish and sardines are not trapped in the local niche and can swim in different areas of the search space to prevent premature convergence. The position updates of the population are driven by randomness and broad movements, enabling the algorithm to cover a wide solution space. This behavior is controlled by the equation Equation 13.

$$AP = 1 - \frac{current\ iteration}{total\ iteration} \tag{13}$$

Through those equations, ensuring that the early iterations prioritize diversity in the population. The exploration mechanism is crucial in avoiding local optima, allowing the algorithm to discover potentially promising regions that may contain the global best solution.

3.3. Exploitation Phase

As the algorithm progresses and AP drops below 0.5, the optimizer transitions into the exploitation phase, where the focus shifts toward refining solutions around the best-found candidates. During this phase, the motion of sardines becomes more directed, with location updates based on the best-known solutions so that the algorithm can efficiently converge to the best area. The decreasing AP value reduces random fluctuations, leading to more precise adjustments in the search space. This balance between exploration and exploitation allows MSFO to maintain a robust search capability while ensuring convergence efficiency. The transition between these phases is dynamically controlled, making MSFO adaptable across different optimization landscapes.

3.4. Algorithm

The pseudocode of the improved version of the MSFO with OBL is given in the algorithm below to improve the exploration and exploitation of the search process. It simplifies the attack power mechanism, which is conducive to improving the efficiency of the algorithm to solve the optimization problem.

```
Pseudocode of modified sailfish optimizer
 1. Initialize a random population of sailfish and sardines.
 2. Set parameters ($\Lambda = 4$, $\varepsilon = 0.001$).
 3. Calculate the fitness of all sailfish and sardines.
 4. Use Opposition-Based Learning (OBL) to improve
    diversity.
 5. Identify the best sailfish as the elite sailfish and the
    sardine as the weakest sardine.
 6. While current iteration < maximum iteration:
 7.     For each sailfish
 8.         Compute $\lambda_i$ using Equation 6, where PD is
            calculated using Equation 7.
 9.         Update sailfish positions using Equation 5
10.         Apply OBL to check opposite positions for better
            results using Equations 11 and 12.
11.     End for
12.     Calculate AttackPower (AP) to decide exploration or
        exploitation using Equation 13
13.     If AP < 0.5 (exploration phase):
```

```
14.         Compute opposite positions using OBL as
            Equations 11 and 12
15.         Update selected sardine position using Equation 8
16.     Else:
17.         Update the position of all the sardines using
            Equation 8
18.     Evaluate the fitness of all sardines.
19.     Sort sailfish and sardines by fitness.
20.     For each sailfish
21.         If the fitness value of the sardine is better than that
            of the sailfish:
22.             Replace the weak sailfish with its sardine using
                Equation 10
23.             Remove the selected sardine from the
                population.
24.             Update the best sailfish and best sardine.
25.         End if
26.     End For
27.     Regenerate missing sardines to maintain population
        size.
28. Return the best sailfish.
```

4. Results and Discussion

The Modified Sailfish Optimizer (MSFO) is applied to address the issue of the traveling salesman, which is a classic problem in combinatorial optimization. MSFO could find near-optimal solutions with lower computational complexity by simplifying the attack power decay mechanism and eliminating some parameters. On the Burma14 dataset, containing 14 cities, the algorithm proves its effectiveness in competitive solution quality. Although the algorithm is stochastic in nature, with some variability in results, MSFO consistently provides high-quality solutions.

4.1. Dataset

The Burma14 instance is a benchmark dataset for the Traveling Salesman Problem (TSP) containing 14 cities, which is used in order to assess the effectiveness of optimization algorithms [25]. That data is a well-known combinatorial optimization problem, where one seeks the shortest tour that visits each city exactly once and returns to its origin. Its simplicity, with fewer parameters, is its strength and computational efficiency, and thus it is a nice approach for solving TSP instances of this size.

Table 2. Cities coordinate with the instance

City	Longtitude	Latitude
0	16.47	96.10
1	16.47	94.44
2	20.09	92.54
3	22.39	93.37
4	25.23	97.24
5	22.00	96.05
6	20.47	97.02
7	17.20	96.29
8	16.30	97.38
9	14.05	98.12
10	16.53	97.38
11	21.52	95.59
12	19.41	97.13
13	20.09	94.55

In this paper, three well-known metaheuristic algorithms are comparatively studied and applied to solve the traveling salesman problem. The main goal is to compare the performance of the algorithms in achieving near-optimal solutions to the problem in an overall sense of travel distance. The dataset's Best Known Solution (BKS) under consideration is taken as the basis for comparison.

The effectiveness of each algorithm is evaluated based on whether or not it can converge to a solution that best approximates the BKS, with a focus on solution quality and computation. The experiments are depicted in order to set up the ability of each of the three algorithms to generate competitive-quality solutions; however, large discrepancies are apparent in their closeness to the BKS. This comparative study identifies the strengths and weaknesses of GA, PSO, and SA in solving combinatorial optimization problems like the TSP and provides insights into their relative success in finding optimal or near-optimal solutions for different instances of problems.

Table 3. Comparative analysis of well-known metaheuristics

Method	Solution Value	Gap
Best Known Solution	30.87	-
Genetic Algorithm	33.96	9.97%
Particle Swarm Optimization	35.37	14.5%
Simulated Annealing	34.98	13.27%

4.2. Computational Result

The Modified Sailfish Optimizer (MSFO) satisfactorily solved the problem of the salesman's travel route through the Burma14 dataset using Python, as seen from the computational results. The program was run on a laptop with an AMD Ryzen 5600H processor and 16GB RAM. The setup was ideal to determine the solution to the problem.

The optimization utilized the following settings: 500 iterations, population of 100, and different values for the pp parameter (SailFish to Sardines ratio) of 0.1. These parameters resulted in a search space that was balanced between exploitation and exploration.

Table 4. MSFO parameters for TSP

Parameter	Value
Iteration number	500
Population size	100
Pp	0.1

The algorithm worked well, finding good solutions close to optimal ones within the given number of iterations. The MSFO could search the solution space efficiently with a population size of 100 and get competitive routes that minimized the total distance covered. The pp values were attempted in different ways, and each way gave relatively different results. However, they all yielded good solutions within a reasonable period of time. The hardware configuration of the laptop was not too demanding, and hence the computational cost was low. Optimization proceeded without compromising the performance too much, and this suggests that the MSFO is effective on TSP problems of this size. Table 5 shows the shortest route that the Modified Sailfish Optimizer (MSFO) found to find the solution of the salesman's travel path.

The Modified Sailfish Optimizer (MSFO) found a solution to the Traveling Salesman Problem (TSP) for the Burma14 dataset, creating an optimized route that worked very well. The algorithm was able to find the shortest route that went through all the cities exactly once, starting in City 11, going through the other cities, and ending in City 6 before going back to City 11. This last route meets the TSP goal because it follows the rules of the problem and makes sure that the distance traveled is as short as possible.

The fact that MSFO can create this route shows that it can explore the solution space well and get close to the best solution in a reasonable number of iterations. With a population of 100, the algorithm went through the 14 cities, looking at different combinations and picking the best route that would cover the least distance overall. The route that was created, which started in City 13 and ended by returning there after visiting all the other cities, is another example of how strong and useful the algorithm is for solving the TSP, even with small datasets like the Burma14. This success shows that the MSFO can efficiently improve TSP routes while still being fast at computing.

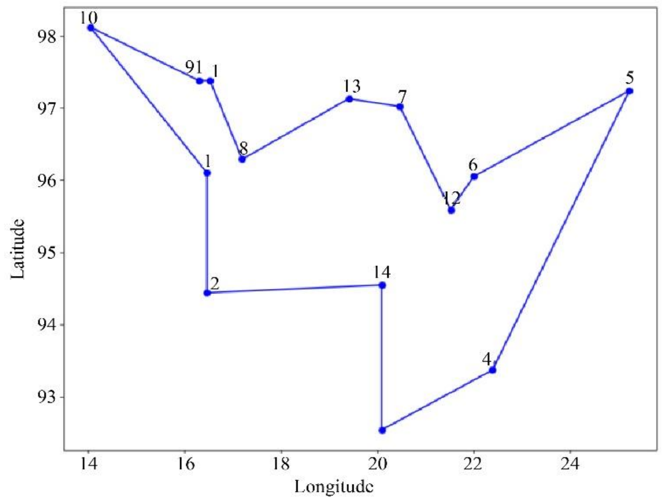

Fig. 2 Constructed route using MSFO

Table 5 shows a route for the issue of a salesman traveling with a total distance of 30.87. This is the best value for this case. This distance is the shortest route that goes to each city only once. So, for its instance dataset, the solution can be thought of as the best one.

Table 5. Generated solution

Route	Distance
13–2–3–4–5–11–6–12–7–10–8–9–0–1	30.87

The traveled distance is 30.87. The results show that the travel starts at city 13 and ends at city 1, with the visits to all the other cities once. The total distance shows that MSFO has the ability to achieve the shortest distance, which is the same as the best-known solution. This shows that MSFO has the ability to effectively solve difficult combinatorial problems using the best exploration and exploitation techniques. The algorithm is able to escape local optima and achieve the best solution using Opposition-Based Learning (OBL) and a better attack power mechanism. These observations render MSFO an even better candidate than traditional optimization algorithms for solving routing and logistics problems.

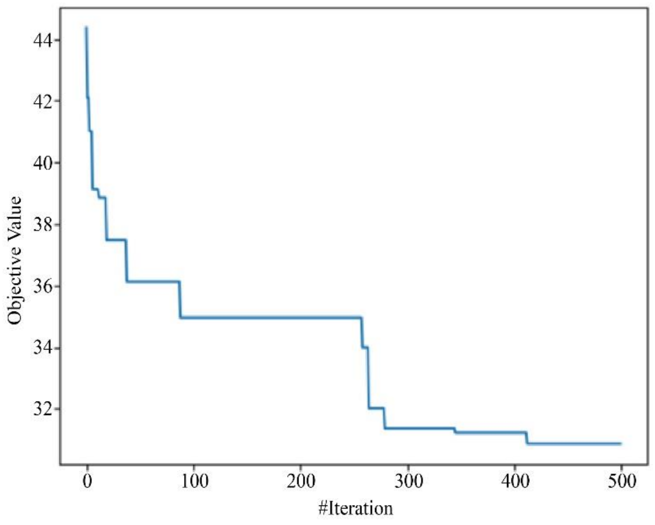

Fig. 3 Convergence rate

The Modified Sailfish Optimizer (MSFO) found the optimal solution for the salesman travel problem in Table 5. The distance traveled is 30.87. As can be seen from the results, the tour starts in city 13 and ends in city 1, with a visit to each of the other cities once. The total distance shows that MSFO can achieve the shortest distance, as it is equal to the best-known solution. It proves that MSFO is efficient in solving hard combinatorial problems by using exploration and exploitation in the most suitable way. By utilizing Opposition-Based Learning (OBL) and a new attack power mechanism, the algorithm can get rid of local optima and converge to the best solution. These results show that MSFO is an even better option than traditional optimization algorithms when it comes to the resolution of routing and logistics issues.

Table 6. Comparative analysis of MSFO

Method	Solution value	Gap
Best Known Solution	30.87	-
Genetic Algorithm	33.96	9.97%
Particle Swarm Optimization	35.37	14.5%
Simulated Annealing	34.98	13.27%
Modified Sailfish Optimizer	30.87	0%

Table 6 compares different optimization algorithms for the traveling salesman route. The Modified Sailfish Optimizer (MSFO) has a solution value of 30.87, which is exactly equal to the Best-Known Solution. This means that there is an optimality gap of 0%. These results show that MSFO outperforms the other metaheuristic algorithms in finding the optimal solutions and optimizing them. The fact that MSFO can find the optimal solution proves that the hybridized Opposition-Based Learning (OBL) and enhanced attack power mechanism works effectively in strengthening search exploration and exploitation. On the other hand, the inferior performance of GA, PSO, and SA shows that they are plagued by escaping local optima and slow convergence. The results show that MSFO is a competitive optimization algorithm for solving challenging combinatorial problems like the TSP and can be used in other real-world settings.

5. Conclusion

In conclusion, the modified Sailfish Optimizer (SFO) has performed well in developing the best possible solutions of the Traveling Salesman Problem (TSP). The algorithm is better as it uses Opposition-Based Learning (OBL) that improves the diversity of solutions and speeds up convergence. It also modulates the attack strength in a simpler but efficient way.

These changes render the SFO more effective at balancing exploration and exploitation, leading to improved results compared to other optimization methods like Genetic Algorithm (GA), Particle Swarm Optimization (PSO), and Simulated Annealing (SA). Experimental results reveal that the proposed changes help the algorithm identify shorter paths faster, making the method suitable for solving hard combinatorial problems.

References

[1] Petrică C. Pop et al., “A Comprehensive Survey on the Generalized Traveling Salesman Problem,” *European Journal of Operational Research*, vol. 314, no. 3, pp. 819-835, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[2] Mohd Arfian Ismail, “A GPU Accelerated Parallel Genetic Algorithm for the Traveling Salesman Problem,” *Journal of Soft Computing and Data Mining*, vol. 5, no. 2, pp. 137-150, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[3] Irina Dumitrescu, and Thomas Stützle, *Usage of Exact Algorithms to Enhance Stochastic Local Search Algorithms*, Annals of Information Systems, Boston, Massachusetts, vol. 10, pp. 103-134, 2009. [CrossRef] [Google Scholar] [Publisher Link]

[4] Agung Chandra, and Aulia Naro, “S-Metaheuristics Approach to Solve Traveling Salesman Problem,” *Metris: Journal of Science and Technology*, vol. 21, no. 2, pp. 111-115, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[5] S. Shadravan, H.R. Naji, and V.K. Bardsiri, “The Sailfish Optimizer: A Novel Nature-Inspired Metaheuristic Algorithm for Solving Constrained Engineering Optimization Problems,” *Engineering Applications of Artificial Intelligence*, vol. 80, pp. 20-34, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[6] Mamta Kumari et al., “Utilizing A Hybrid Metaheuristic Algorithm to Solve Capacitated Vehicle Routing Problem,” *Results in Control and Optimization*, vol. 13, pp. 1-15, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[7] Battina Srinuvasu Kumar, S.G. Santhi, and S. Narayana, “Sailfish Optimizer Algorithm (SFO) for Optimized Clustering in Wireless Sensor Network (WSN),” *Journal of Engineering, Design and Technology*, vol. 20, no. 6, pp. 1449-1467, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[8] Mesfer Al Duhayyim et al., “Sailfish Optimization with Deep Learning Based Oral Cancer Classification Model,” *Computer Systems Science and Engineering*, vol. 45, no. 1, pp. 753-767, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[9] Safaa. M. Azzam, O.E. Emam, and Ahmed Sabry Abolaban, “An Improved Differential Evolution with Sailfish Optimizer (DESFO) for Handling Feature Selection Problem,” *Scientific Reports*, vol. 14, pp. 1-27, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[10] R. Sathyavani, K. JaganMohan, and B. Kalaavathi, “Sailfish Optimization Algorithm with Deep Convolutional Neural Network for Nutrient Deficiency Detection in Rice Plants,” *Journal of Pharmaceutical Negative Results*, vol. 14, no. 2, pp. 1713-1728, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[11] Mazen Mushabab Alqahtani et al., “Sailfish Optimizer with EfficientNet Model for Apple Leaf Disease Detection,” *Computers, Materials and Continua*, vol. 74, no. 1, pp. 217-233, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[12] Utkarsh Mahadeo Khaire et al., “Instigating the Sailfish Optimization Algorithm based on Opposition-Based Learning to Determine the Salient Features from a High-Dimensional Dataset,” *International Journal of Information Technology & Decision Making*, vol. 22, no. 5, pp. 1617-1649, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[13] Caio Paziani Tomazella, and Marcelo Seido Nagano, “A Comprehensive Review of Branch-and-Bound Algorithms: Guidelines and Directions for Further Research on the Flowshop Scheduling Problem,” *Expert Systems with Applications*, vol. 158, pp. 1-19, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[14] P. Rajarajeswari, and D. Maheswari, “Travelling Salesman Problem Using Branch and Bound Technique,” *International Journal of Mathematics Trends and Technology*, vol. 66, no. 5, pp. 202-206, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[15] Wenqiang Yanget al., “Improved Shuffled Frog Leaping Algorithm for Solving Multi-aisle Automated Warehouse Scheduling Optimization,” *Communications in Computer and Information Science*, Berlin, Germany, pp. 82-92, 2013. [CrossRef] [Google Scholar] [Publisher Link]

[16] Li Li et al., “A Discrete Artificial Bee Colony Algorithm for TSP Problem,” *Lecture Notes in Computer Science*, Berlin, Heidelberg, pp. 566-573, 2012. [CrossRef] [Google Scholar] [Publisher Link]

[17] Rajesh Matai, Surya Singh, and Murari Lal Mittal, *Traveling Salesman Problem: An Overview of Applications, Formulations, and Solution Approaches*, InTech, pp. 1-26, 2010. [CrossRef] [Google Scholar] [Publisher Link]

[18] G. Dantzig, R. Fulkerson, and S. Johnson, “Solution of a Large-Scale Traveling-Salesman Problem,” *Journal of the Operations Research Society of America*, vol. 2, no. 4, pp. 393-410, 1954. [CrossRef] [Google Scholar] [Publisher Link]

[19] Dalia T. Akl et al., “IHHO: An Improved Harris Hawks Optimization Algorithm for Solving Engineering Problems,” *Neural Computing and Applications*, vol. 36, pp. 12185-12298, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[20] Woong Sagong, Woo-Pyung Jeon, and Haecheon Choi, “Hydrodynamic Characteristics of the Sailfish (Istiophorus Platypterus) and Swordfish (Xiphias gladius) in Gliding Postures at their Cruise Speeds,” *PLoS One*, vol. 8, no. 12, pp. 1-14, 2013. [CrossRef] [Google Scholar] [Publisher Link]

[21] James E. Herbert-Read et al., “Proto-Cooperation: Group Hunting Sailfish Improve Hunting Success by Alternating Attacks on Grouping Prey,” *Proceedings of the Royal Society B: Biological Sciences*, vol. 283, no. 1842, pp. 1-9, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[22] Tae Jong Choi, Julian Togelius, and Yun-Gyung Cheong, “A Fast and Efficient Stochastic Opposition-Based Learning for Differential Evolution in Numerical Optimization,” *Swarm and Evolutionary Computation*, vol. 60, pp. 1-25, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[23] Shahryar Rahnamayan, Hamid R. Tizhoosh, and Magdy M. A. Salama, “Opposition-Based Differential Evolution,” *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 1, pp. 64-79, 2008. [CrossRef] [Google Scholar] [Publisher Link]

[24] Hamid R. Tizhoosh, and Mario Ventresca, *Oppositional Concepts in Computational Intelligence*, Studies in Computational Intelligence, vol. 155, 2008. [CrossRef] [Google Scholar] [Publisher Link]

[25] TSPLIB, Discrete and Combinatorial Optimization, Ruprecht-Karls-University Heidelberg, 2013. [Online]. Available: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/