International Journal of Computational Methods and Experimental Measurements
Vol. 13, No. 2, June, 2025, pp. 371-379
Journal homepage: http://iieta.org/journals/ijcmem

Charting New Routes: Comparing Swarm-Based Approaches to the Traveling Salesman Problem

Ali Hassan Ahmed Wadi[1*], Shahla Uthman Umar[2]

[1] Computer Science Department, College of Computer Science and Information Technology, University of Kirkuk, Kirkuk 36001, Iraq
[2] Software Department, College of Computer Science and Information Technology, University of Kirkuk, Kirkuk 36001, Iraq

Corresponding Author Email: stcm23009@uokirkuk.edu.iq

https://doi.org/10.18280/ijcmem.130214

Received: 19 December 2024
Revised: 15 March 2025
Accepted: 26 March 2025
Available online: 30 June 2025

Keywords:
Traveling Salesman Problem, Elephant Herding Optimization, Ant Colony Optimization, Particle Swarm Optimization, Branch and Bound, Dynamic Programming, optimization algorithms, combinatorial optimization

ABSTRACT

To solve the Traveling Salesman Problem (TSP), this research compares three swarm-based optimization algorithms: Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), and Elephant Herding Optimization (EHO). Finding the shortest path to visit each city once and return to the starting point is the goal of the traditional combinatorial optimization problem, TSP. Exact techniques such as Branch and Bound (BB) and Dynamic Programming (DP) can effectively handle smaller TSP cases, but they become unfeasible as the number of cities increases. The solutions offered by metaheuristic algorithms are more scalable. The algorithms' performance is assessed in this study based on execution time, scalability, and solution quality for a range of city sizes (5 to 150). Results reveal that EHO surpasses the others in achieving lower optimal costs.

1. INTRODUCTION

Since the creation of mortal beings, they have constantly sought perfection in all aspects of life. One of the most important trials in the world is to find a stylish result. In reality, numerous complex problems, similar to transportation, warehousing, where to vend products, communication network design, scheduling, and planning, are frequently too large and complex to be optimally answered in a reasonable time. Nonetheless, chancing a result is still pivotal, so the volition is to originally accept a sour result with a respectable position of delicacy and optimization time [1].

Optimization problems have become so complicated that they are difficult for traditional programming approaches to decompose and optimize efficiently. A mass grounded metaheuristic optimization methods have been developed recently [2].

The machine learning models, especially ensemble learning approaches, to show great promise in solving complicated optimization issues by utilizing a variety of data-driven strategies to improve decision-making and prediction accuracy [3]. Used address challenging optimization issues is represented by swarm intelligence (SI) algorithms. Its goal is to model the collective behavior of basic agents as they attempt to accomplish goals like protecting against attacks and finding food. Even though each agent is one capable of basic tasks, when the work together and share knowledge, they can display extraordinary intelligence [4]. SI algorithms were first developed at the University of Michigan in the 1960s. John Holland and his associates wrote the first book on the GA in 1960, and it was later developed and published in 1970 and 1983 [5]. Have been extensively used by experimenters to optimize results and give sufficiently fit results for objective functions in optimization problems [2]. In similar problems, the ultimate thing is frequently to maximize or minimize an objective function, which is used to estimate the quality of the performing result. These algorithms aim to ameliorate or minimize the problem's objective function, and the Traveling Salesman Problem (TSP) is constantly used to test their effectiveness and estimate their performance. The TSP as it needs changing the shortest path to visit a set of big cities, the making it a perfect tool for assessing the effectiveness of different algorithms. A number of metaheuristic algorithms, including ACO, PSO, and EHO, are utilized to find a solution to the TSP, a classic problem in route optimization. Yet, a thorough comparison of the algorithms based on execution time, use of resources, and solution quality is still required. It is seen that (EHO) is quicker than the others, and hence all the more useful for big instances of (TSP) where repair has to be executed in a hurry. With emphasis laid on computational complexity and the quest for finding a balance between accuracy and efficiency, the present study attempts to evaluate the performance of these algorithms over various sets of datasets such that one can offer recommendations towards the optimal strategy to adopt for use in applications related to automated manufacturing, smart transportation, and logistics optimization.

2. LITERATURE REVIEW

From 2000 till 2024, it has been two decades of research to implement artificial intelligence (AI) and metaheuristic algorithms to determine the solution of the Traveling Salesman Problem (TSP), a popular NP-hard combinatorial optimization problem. It was all about creating and enhancing various algorithms like Branch and Bound (BB), Dynamic Programming (DP), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), and recently Elephant Herding Optimization (EHO).

1) Evolution and Trends in TSP Algorithms:

TSP solutions in the early stages were exact in nature like BB and DP [6, 7], which did not work for large datasets since they were exponential in terms of memory and time requirements. SI techniques like ACO and PSO were introduced to combat the same. ACO is inspired by ant foraging behavior [8] and PSO incorporates social optimization. Both of them were susceptible to premature convergence and parameter sensitivity [9]. EHO inspired by elephant herd behavior assisted in improving execution times and solution quality for large TSP instances [10], and the strengths and weaknesses of the studies reviewed are summarized in Table 1.

2) Thematic Grouping of Literature:

The literature studied can be classified as:

➢ **Algorithmic Efficiency Improvements:**
- Zhang [11] strengthened often used paths to improve ACO, speeding up the solution process.
- Emambocus [12] achieved more accurate results using PSO with genetic algorithms.
- Marqas et al. [13] improved performance on huge datasets by optimizing EHO parameters.

➢ **Handling Large-Scale Datasets:**
- Robati et al. [14] modified PSO to make efficient usage feasible in larger instances of TSP.
- Li et al. [15] and Zhang and Gao [16] confirmed EHO's effectiveness in scaling across large-dimensional datasets.

➢ **Dynamic and Adaptive Algorithms:**
- A model of ACO that is capable of adapting to varying city distances was introduced by Zhou et al. [17].

3) Relevance to Current Study:

This paper evaluates three popular algorithms ACO, PSO, and EHO using different dataset sizes. EHO consistently outperforms ACO in terms of optimal cost, especially as the number of cities increases. This highlights the increasing use of swarm-based methods in areas such as dynamic routing, transportation planning, and intelligent logistics systems. and presents an EHO framework that decomposes the problem to enable solution quality and scalability. The framework fills current research gaps and provides a more flexible method than TPS in resolving complex and large-size instances.

Table 1. Comparative summary of key studies

Year	Algorithm	Main Contribution	Strengths	Weaknesses
2010	ACO	Enhanced pheromone-based search	High efficiency on small datasets	Limited on large datasets
2012	PSO	Scalable search behavior	Fast for large instances	Parameter sensitive
2014	BB	Accurate subproblem elimination	Optimal for small inputs	Impractical for large data
2015	PSO + GA	Hybridized PSO for accuracy	Improved results	Complex implementation
2017	ACO	Adaptive to dynamic data	Responsive to changes	Slower performance
2018	EHO	New metaheuristic with fast convergence	High quality on big data	Less efficient on dynamic data
2019	DP	Accurate results with caching	Guarantees optimality	High memory usage
2020	EHO	Parameter-tuned EHO	Enhanced efficiency	Needs tuning expertise
2021	EHO	Contextual performance analysis	Versatile across scenarios	May require longer time
2022	EHO	Accelerated search and precision	Better performance	Requires high computing power
2023	EHO	Large-scale dataset handling	Robust output	Not adaptive to dynamic input
2024	EHO	Practical applications in logistics	Real-world relevance	May vary in unpredictable settings

3. TRAVELING SALESMAN PROBLEM (TSP)

In 1932, the mathematician Karl Menger first proposed the TSP. The problem formulation sounds surprisingly simple: consider a salesman who has to travel between several towns. He starts in his home town, visits each of the cities on some list exactly once, and then returns to the starting point. Reducing the overall distance traveled is the aim. Even while it seems straightforward, the more cities there are, the more challenging it becomes to solve this problem optimally. Mathematicians and scholars have been looking for effective answers for nearly a century. From its simple definition to the difficulty of illustrating its solutions is where TSP's beauty lies. The cities are very often real locations in practical applications, while travel routes are determined by distances. We will focus on the TSP instances that represent cities connected with road networks, where the distances represent the actual driven distance by automobiles. Maps will be used to display the results in order to improve comprehension and show how the solutions have practical applications. One could initially think that the issue can be resolved by just figuring out how long each potential tour is and choosing the shortest one. However, since the number of alternative tours grows factorially with the number of cities, this strategy is only practical for extremely small examples. For example, over 3 billion hours are feasible in only 14 cities. This "brute force" approach is computationally impractical for larger instances. Due to this, more complicated algorithms must be devised to successfully handle the problem, especially when large input datasets are involved [18], a classic graph-based optimization problem where a traveler must visit a given set of cities only once before returning to the starting point, while minimizing the total travel cost.

To date, no known polynomial time algorithm can solve every TSP instance. This means that it is NP-hard. Owing to this complexity, there have been numerous research regarding combinatorial optimization. TSP has proved itself as a benchmark to test any new optimization techniques as it has broad applications in different fields such as manufacturing, chip design, and logistics [19-21].

Another perspective on the limitations of AI is the inability of the conventional AI techniques to scale up with developments in machine learning and optimization for information systems possessing big datasets, as well as the recent expansion in the industry, particularly energy and pharmaceuticals. Computational intelligence, a field dedicated to developing intelligent computational models that can interpret raw numerical data in real time and provide high reliability and minimal errors for engineering and commercial applications, has been made possible by this gap [22].

Several heuristic and metaheuristic methods, for example, PSO, ACO, and EHO, have been developed for seeking an approximate solution for TSP. Each has its merits in a different way by trading off the accuracy of the solution against computing efficiency. For the best answers in smaller TSP scenarios, precise methods like branch-bound and Dynamic Programming have also been investigated. However, even though these exact methods ensure optimality, they are usually restricted to issues with fewer cities due to their large computational demands [12-14].

The length of the optimal tour of TSP problems can be found as shown below [14], can be calculated by Eq. (1).

$$optimal\ tour = d_{p(n)p(1)} + \left(\sum_{i=1}^{n-1} d_{p(i)p(i+1)}\right) \quad (1)$$

where, p is an ordered list of cities, and p(i) and p(i+1) are successive locations in the tour, and p(i) and p(i+1) are consecutive cities, The distance between city $p(i)$ p(i) and city $p\ (i+1)$ p(i+1) is shown by the formula d(p(i), p(i+1)).

TSP Applications

- **Logistics and supply chains**: By minimizing delivery vehicle and truck routes, TSP reduces fuel usage and travel time [23].
- **Manufacturing and production**: It is used to schedule machine tasks and reduce travel time in electronics manufacturing, e.g., factories that manufacture printed circuit boards (PCBs) [24].
- **Communications and networking**: It is used to build wireless and wired networks and enhance data routing protocols to reduce delay [25].
- **Health and medicine**: It enables DNA sequencing to speed up diagnosis and planning of ambulance routes [26].
- **Power and resource management**: It conserves operating costs by allocating work crews and planning power plant maintenance [27].
- **Traffic control and urban planning**: It conserves operating costs and traffic jams by route planning for trash collection and regulating traffic lights [28].

This overall distance is to be minimized over all city orderings. Because of its practical relevance as well as its theoretical importance, TSP remains one of the most studied optimization problems. It has been applied in network architecture enhancement, reduction of production costs, and optimization of delivery routes. Further, the problem applicability has grown in a number of areas because of novel variants, which include the Vehicle Routing Problem and the multiple Traveling Salesman Problem.

4. CLASSICAL ALGORITHMS TO SOLVE TSP

Different exact algorithms such as BB and DP are used to solve the TSP.

4.1 Branch and Bound (BB)

General fashion for BB algorithms involves modeling the result space as a tree and also covering the tree exploring the most promising subtrees first [29].

This is continued until either there are no subtrees into which to further break the problem, or we have arrived at a point where, if we continue, only inferior results will be set up. can be used to process TSP containing 40–60 cities [18].

4.2 Dynamic Programming (DP)

Dynamic Programming (DP) is a very important fashion for efficiently calculating recurrences by storing partial results and reusing them when demanded [30].

It is a system for working on a complex problem by breaking it down into a collection of simpler sub problems. It demands veritably elegant expression of the approach and simple thinking and the rendering part is veritably easy. The idea is veritably simple if you have answered a problem with the given input, also save the result for future reference, to avoid working the same problem again, shortly' Flash back your history'. Still, in this process, if you observe some over-lapping sub-problems, the given problem can be broken up into lower sub-problems and these lower sub-problems are in turn divided into still-lower bones Also, the optimal results of the sub-problems contribute to the optimal result of the given problem (appertained to as the Optimal Substructure Property [31].

There are two ways of doing this.

1. Top-down launch working the given problem by breaking it down. However, and also just returns the saved answer, if you see that the problem has been answered already. However, break it and save the answer, if it has not been answered. This is generally easy to suppose and veritably intuitive. This is appertained to as Memorization.

2. Bottom-Up dissect the problem and see the order, in which the sub-problems are answered and start working from the trivial sub-problem, up to the given problem. In this process, it's guaranteed that the sub-problems are answered before working on the problem. This is appertained to as Dynamic Programming.

Steps followed while enforcing Dynamic Programming 1. Characterize the recursive structure of an optimal result, define recursively the value of an optimal result, Cipher, bottom up, the cost of a result, and construct an optimal result. This approach is also used to break the traveling salesperson problem but only for a limited number of metropolises Steps followed while implementing Dynamic Programming:

i. Characterize the recursive structure of an optimal solution.
ii. Define recursively the value of an optimal solution.
iii. Compute, bottom up, the cost of a solution.
iv. Construct an optimal solution.

This approach is also used to solve the TSP but only for a limited number of cities [24].

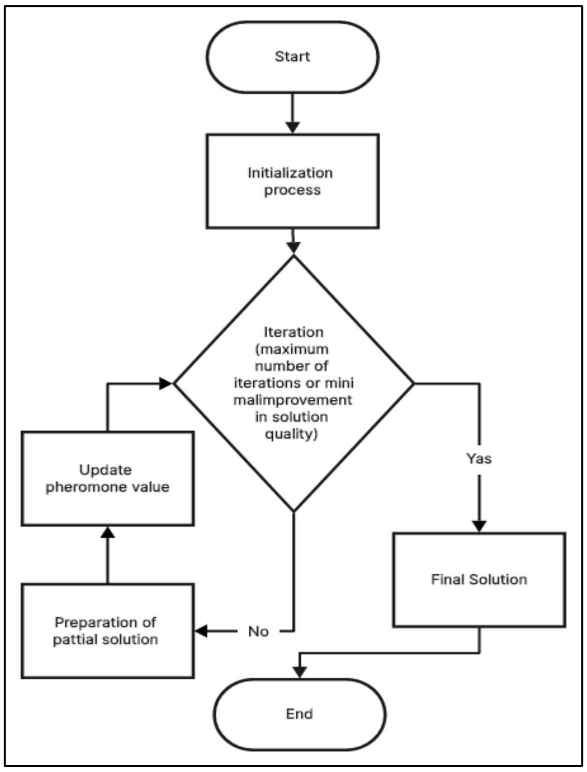
5. HEURISTIC ALGORITHMS TO SOLVE TSP

The different optimization algorithms such as PSO, ACO and EHO This section describes the methods used to solve the

TSP.

5.1 Ant Colony Optimization (ACO)

Is a metaheuristic search and optimization method inspired by the "intelligent" foraging behavior of natural ant colonies, and is widely used to solve (mostly combinatorial) optimization problems, The basic principle of ACO is that a colony of artificial ants work together to find the best path in a graph that represents a possible solution to the target problem, The way the artificial ants cooperate with each other is inspired by the way natural ants cooperate to find the shortest path between two points in a given terrain, such as their nest and a food source, When an ant constructs a possible solution, it deposits pheromones proportional to the quality of the solution in the region of the search space where the solution is located. Over time, the ants tend to converge on paths that represent close to optimal solutions in the search space [32].



Start
Initialization process
Iteration (maximum number of iterations or mini malimprovement in solution quality)
Update pheromone value
Yas
Final Solution
Preparation of pattial solution
No
End

Figure 1. Flowchart of the ACO algorithm

Steps of the ACO Algorithm for Solving the TSP as shown in the Figure 1:

1. **Parameter Initialization**: The number of ants, the number of cities, the initial pheromone level, and the algorithm parameters are defined as follows:
   - α: pheromone influence
   - Distance influence β
   - Pheromone evaporation rate (ρ)
2. **Solution Construction**: Each ant builds probabilistically a tour to choose the next city based on pheromone level. Distances will be calculated by using the formula expressed in Eq. (2).

$$P_{ij} = \frac{T_{ij}^{a} * \eta_{ij}^{\beta}}{\sum_{k \epsilon allowed} T_{ij}^{a} * \eta_{ij}^{\beta}} \quad (2)$$

P_{ij} = probability of moving from city (i) to city (j), T_{ij}^{a} = Pheromone amount on edge i, j,
$\eta_{ij}^{\beta} = \frac{1}{d_{ij}}$: attractiveness (inverse of distance d_{ij}).

3. **Pheromone Update**: After all ants complete their tours, update the pheromone levels on the paths used by Eq. (3).

$$T_{ij} = (1 - p) * T_{ij} + \sum_{k=1}^{m} \triangle T_{ij}^{k} \quad (3)$$

p: Pheromone evaporation rate,
$\triangle T_{ij}^{k} = \frac{Q}{L_k}$: Pheromone deposited by ant k, Q: constant,
L_k: length of the tour constructed by ant k.

4. **Iteration:** Repeat the above steps until a stopping criterion is met (e.g., a maximum number of iterations or minimal improvement in solution quality).
5. **Final Solution:** The best tour found over all iterations represents the optimal or near-optimal solution for the TSP.

ACO Pseudocode

```
Initialize parameters: α, β, ρ, Q, number of ants, number of cities
Initialize pheromone levels τ on all edges to a small constant

For each iteration:
  For each ant k (from 1 to number of ants):
    Place ant k on a random starting city
    For each city in the tour:
      Select the next city j to visit based on probability:
        P_{ij} = [τ_{ij}^α * η_{ij}^β] / Σ[τ_{ik}^α * η_{ik}^β] for allowed cities k
      Add city j to the ant's tour and move the ant to city j
    Complete the tour and return to the starting city
    Compute the tour length L_k for ant k

  Update pheromones on all edges:
    For each edge (i, j):
      Evaporate pheromone: τ_{ij} = (1 - ρ) * τ_{ij}
      Deposit new pheromone for each ant k that used edge (i, j):
        τ_{ij} = τ_{ij} + Δτ_{ij}^k where Δτ_{ij}^k = Q / L_k

Repeat until stopping condition (e.g., max iterations or no improvement)
```

5.2 Particle Swarm Optimization (PSO)

The PSO algorithm was derived from the collective behavior of birds and fish; in the PSO algorithm, the population consists of a large number of particles. Each particle representing a potential solution is a point in the search space, with a fitness value and velocity. PSO is conceptually very simple, requiring no derived information about the optimization function and using only elementary mathematical operators [33].

Steps of the PSO Algorithm for Solving the TSP as shown in the Figure 2:

1. **Initialization of Particles**:
   - Each particle represents a solution (order of visiting cities).
   - Each particle is assigned a random position and velocity.
2. **Fitness Evaluation**: **Fitness = Length of the Tour**

(Calculate the total distance between cities in the tour. The goal is to minimize this length).

3. **Velocity Update**: The velocity of each particle is updated based on its personal best position (**pBest**) and the global best position of the swarm (**gBest**) can be calculated by Eq. (4):

$$V_i = \omega * V_i + c_1 * r_1 * (pBest_i - X_i) + c_2 * r_2 * (gBest - X_i) \tag{4}$$

ω : Inertia weight controls the impact of the previous velocity,

c_1, c_2: Learning coefficients, with one directing towards pBest and the other towards $gBest$,

r_1, r_2: Random numbers between (0 and 1).

4. **Position Update**: The position of the particles (the order of the cities) is changed based on the velocity updated in the TSP (this is done by swapping or adjusting the order of the cities).
5. **Update pBest and gBest**:
   - If the new solution is better than pBest, it is updated.
   - If the new solution is better than gBest, it is also updated.
6. **Iteration**: Repeat updating velocity and position until the specified number of iterations or convergence is reached.

PSO Pseudocode

```
Initialize parameters:
  num_particles = Number of particles, num_iterations = Maximum number of iterations
cities = List of cities (coordinates)
 w = Inertia weight, c1 = Cognitive coefficient, c2 = Social coefficient
Initialize particles:
  For each particle i from 1 to num_particles:
    Initialize position X[i] randomly (random tour of cities)
    Initialize velocity V[i] randomly
    Initialize pBest[i] = X[i] (best known position of particle)
    Calculate fitness(pBest[i]) and set pBestFitness[i] = fitness(pBest[i])
  gBest = Best particle's position in the swarm
  gBestFitness = Best fitness value among all particles
For iteration = 1 to num_iterations:
  For each particle i from 1 to num_particles:
    Update velocity:
      V[i] = w * V[i] + c1 * r1 * (pBest[i] - X[i]) + c2 * r2 * (gBest - X[i])
    Update position:
      X[i] = UpdatePosition(X[i], V[i])
    Calculate fitness(X[i])
   Update pBest
    If fitness(X[i]) < pBestFitness[i]:
      pBest[i] = X[i]
      pBestFitness[i] = fitness(X[i])
    // Update gBest
    If fitness(X[i]) < gBestFitness:
      gBest = X[i]
      gBestFitness = fitness(X[i])
 Return gBest as the best tour found
```

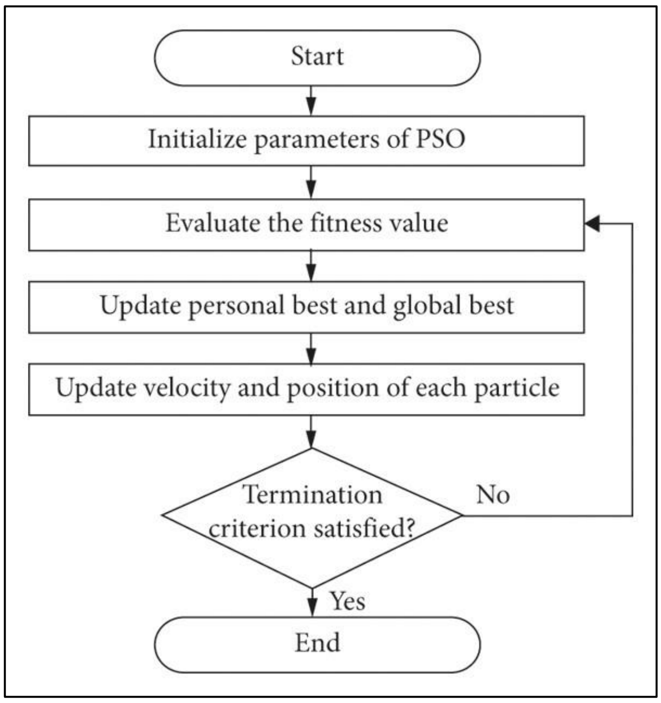

Start
Initialize parameters of PSO
Evaluate the fitness value
Update personal best and global best
Update velocity and position of each particle
Termination criterion satisfied?
No
Yes
End

Figure 2. Flowchart of the PSO algorithm

5.3 Elephant Herding Optimization (EHO)

EHO are optimization problems that require a swarm-based metaheuristic search approach, which was defined by Wang towards the end of 2015, The algorithm simulates how real elephants in a clan would herd their herds [2, 9].

The following is a summary of the herding behavior:

• The swarms of elephants are divided into several smaller groups, known as clans, that are made up of several female elephants and their calves [2, 9].

• A matriarch, or adult female, is in charge of overseeing every clan [2, 9].

• A male calf in a clan leaves the group when it reaches adulthood [2, 9].

Steps of the EHO Algorithm for Solving the TSP as shown in the Figure 3:

1. **Initialization**: Randomly initialize positions (tours) for each elephant in clans.
2. **Fitness Evaluation**:
   - Fitness = Total distance of the tour (sum of distances between cities).
   - Minimize this fitness function.
3. **Clan Grouping**: Divide elephants into clans (subgroups).
4. **Position Update**: Update position of each elephant by Eq. (5).

$$X_{ij}^{t+1} = X_{best,j}^{t} + a * \left(X_{ij}^{t} - X_{center,j}^{t}\right) * r \tag{5}$$

X_{ij}^{t}: position of elephant i, $X_{best,j}^{t}$: Best elephant in the clan, $X_{center,j}^{t}$: clan center, a: learning rate, r: random value in [0,1].

5. **Migration (Separation Operator)**: Replace worst elephant (longest tour) with a random tour by Eq. (6).

$$X_{worst} = X_{new\ random} \tag{6}$$

X_{worst}: is the current solution of the worst elephant, i.e., the elephant with the longest tour, $X_{new\,random}$: is a newly generated random solution, created by randomly shuffling the cities to form a new tour.

6. **Update Global Best**: If a new best tour is found, update the global best solution.
7. **Iteration**: Repeat until convergence or maximum iterations reached.

EHO Pseudocode

```
# Initialize parameters
Initialize elephants with random tours
Set number of clans, elephants per clan, and max iterations
Initialize global best (gBest)

# Main loop
For iteration = 1 to max_iterations:
    For each clan:
        Compute clan center (E_center)
        For each elephant:
            Update position: X_i = X_best + α * (X_i - E_center) * random_factor
            Ensure valid tour and update personal best (pBest)
        Update clan's best (gBest)

    # Migration step
    Replace worst elephant with a new random tour
    Update gBest if a better solution is found

# Termination
Return best tour (gBest)
```

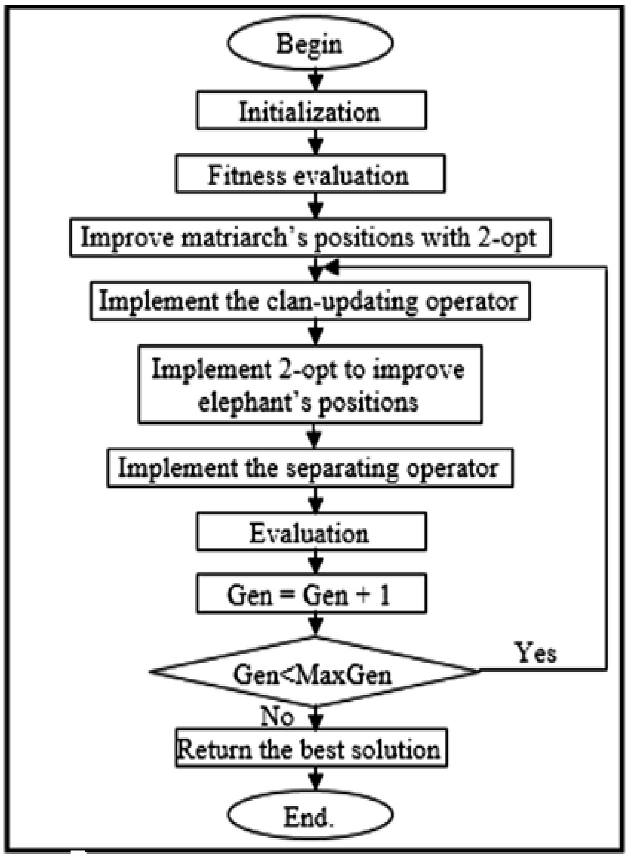

Begin
Initialization
Fitness evaluation
Improve matriarch's positions with 2-opt
Implement the clan-updating operator
Implement 2-opt to improve elephant's positions
Implement the separating operator
Evaluation
Gen = Gen + 1
Gen<MaxGen
Yes
No
Return the best solution
End.

Figure 3. Flowchart of the EHO algorithm

6. EXPERIMENT

This section will analyze the results of the proposed algorithms for solving the TSP by using a number of performance indicators that came from the implementation of the following algorithms: ACO, PSO, BB, DP, and EHO. A number of criteria were used to analyze the performance, including memory consumption (data space, instruction space, environment stack space), execution time, CPU consumption, time and space complexity, and solution quality (optimal cost), these metrics evaluate the effectiveness and practicality of different methods for real-world applications in logistics, transport, and robotics.

- **Execution Time:** In a real-time scenario, such as robotics and transport route optimization, this metric gauges how fast an algorithm can compute a solution.
- **CPU Consumption:** It is a measure of computational efficiency having an impact on cost in the cloud and the lifespan of the battery in the embedded device.
- **Space and Time Complexity:** This describes scalability—exact algorithms (i.e., Branch & Bound) require exponential memory and time, but metaheuristic algorithms (ACO, PSO, EHO) offer effective, low-memory solutions.
- **Solution Quality:** This analyzes how close a computed solution is to the optimal one, also known as the answer, and tradeoffs in speed and accuracy for scenarios like network planning and logistics.

The effectiveness of the algorithms (ACO, PSO, EHO) for solving TSP depends on the parameter values. Tuning the parameters ensures faster convergence rates and higher-quality solutions.

Key Parameters' Effect on Performance:

- ACO: (β, p) shape the balance between exploration and exploitation.
- PSO: (ω, c_1, c_2) Compute the convergence speed and diversity of searches.
- EHO:(N, a) control exploration and stability.

Optimization Strategies:

- Manual Tuning: Empirically made changes on the basis of experimental findings.
- Adaptive Tuning: Dynamic parameter adjustments during execution.

On a laptop computer (HP EliteBook x360 1030 G3) with an Intel(R) Core (TM) i5-8350U CPU @ 1.70GHz 1.90GHz, 16 GB of RAM, and Microsoft Windows 10 Pro, the code was run and the results were extracted using MATLAB. The comparison is shown below.

7. RESULT BASED ON THE NUMBER OF CITIES

In order to have a better understanding of how different algorithms perform when the number of cities is changed in the Traveling Salesman Problem (TSP), the findings have been structured into a formal overview that captures dominant trends and general observations. Instead of describing each data point separately, we provide a summary analysis supported by a Table 2 that shows performance measures.

General Observations:

- BB and DP: These classical algorithms performed well for small instances (5 or 10 cities), which returned optimal and correct solutions. However, with an increasing number of cities, their time complexity and space complexity soar exponentially, and hence they are not appropriate for large instances.
- ACO and PSO: These swarm-based techniques provided a good trade-off between optimal cost and execution time. ACO was cost-effective while PSO was quick to execute but both were parameter-sensitive.

Table 2. Excremental results to solve TSP

Cities	Algorithm	Best Cost	Execution Time	CPU Time Used	Time Complexity
	BB	160.6408513	0.0635367	0.015625	120
	DP	152.9782251	0.0631584	0	800
5	ACO	265.3240882	0.0966347	0.03125	12500
	PSO	259.8042864	0.0748312	0.015625	2500
	EHO	24.93147748	0.0765446	0	2500
	BB	242.2875741	0.3512854	0.328125	3628800
	DP	310.9579789	0.181423	0.125	102400
10	ACO	264.8897796	0.1290255	0.0625	100000
	PSO	302.4137614	0.1214331	0.03125	10000
	EHO	192.0919305	0.0764755	0.015625	10000
	BB	291.6526084	88.3170388	88.328125	87178291200
	DP	352.7964137	4.4405244	4.4375	3211264
14	ACO	393.5055628	0.1603183	0.078125	274400
	PSO	481.7481027	0.0740104	0.015625	19600
	EHO	273.6401486	0.0749235	0	19600
	BB	Reaching the solution requires an unacceptably high period of time.			
	DP	426.2904349	267.732404	268.078125	189267968
19	ACO	426.1901479	0.2017091	0.125	685900
	PSO	702.5979029	0.0744863	0.015625	36100
	EHO	571.0097348	0.0779557	0	36100
	BB	Reaching the solution requires an unacceptably high period of time.			
	DP	Reaching the solution requires an unacceptably high period of time.			
100	ACO	1007.00986	2.009875	1.9375	100000000
	PSO	4495.766986	0.0987641	0.03125	1000000
	EHO	3924.741041	0.1360135	0.046875	1000000
	BB	Reaching the solution requires an unacceptably high period of time.			
	DP	Reaching the solution requires an unacceptably high period of time.			
150s	ACO	1333.117111	3.0940642	3.03125	337500000
	PSO	6886.673	0.1309863	0.0625	2250000
	EHO	6352.343576	0.1166032	0.03125	2250000

- EHO: This algorithm always provided the optimal cost in most cases and had low execution time, especially in big city sets. It was highly scalable and efficient and thus particularly well-suited for big and intricate datasets.

Trends:

- For 5 and 10 cities: All algorithms provide solutions within reasonable execution time. EHO performs better than others regarding solution quality.
- For 14 and 19 cities: Classical algorithms begin lagging behind. EHO performs very well; ACO and PSO scale fairly well.
- For 100 and 150 cities: Swarm-based algorithms alone (ACO, PSO, EHO) provide solutions within reasonable time. EHO and ACO perform better than PSO on cost quality.

8. ANALYSES

We may infer from the data that the meta-heuristic algorithms (ACO, PSO, and EHO) performed better as the number of cities rose as compared to the BB and DP algorithms. EHO came out for providing notably lower optimal costs than the other algorithms, while ACO and PSO demonstrated balanced performance between execution time and ideal cost. The conventional algorithms (BB and DP) are less efficient when working with more cities because of their great temporal and spatial complexity.

Critical Analysis:

Based on the research results, the performance of each algorithm is highly sensitive to the size and type of problem. ACO and PSO are suitable for deterministic methods, while EHO and BB are ideal for small TSP cases.

9. CONCLUSIONS

follows from the analysis in the above discussion that both as can be seen from the carried-out analysis, execution of TSP algorithms primarily relies on both dataset size (number of cities) and time of execution necessary. Traditional algorithms such as BB and DP executed perfectly with small problem sizes (e.g., 10 or 5 cities), yielding accurate and efficient outcomes. However, their lack of computational efficiency was felt as problems turned increasingly complex. As a contrast, metaheuristic algorithms, EHO, ACO, and PSO were found more scalable and flexible with bigger data sets.

Out of the tested algorithms, EHO returned the best costs with highest quality for all except one instance of the problem sizes and demonstrated its global search capability and convergence property. ACO, on the other hand, had a very good cost-effectiveness/executions time ratio, particularly on large instances. PSO was also good but demonstrated variability with parameter settings.

Future directions:

For increasing the convergence rate and solution quality in the future research on TSP, researcher could focus on hybrid solutions incorporating local search and swarm intelligence together with AI-supported learning mechanisms. More sophisticated variations of TSP such as the Vehicle Routing Problem and Dynamic TSP can further be augmented by adaptive and real-time optimization routines. Machine learning algorithms can further facilitate dynamic adjustment of parameters such that algorithms may automatically adapt to evolving problem instances. Additionally, the use of deep learning frameworks can facilitate faster computation and increased accuracy in real-world applications like smart logistics, autonomous navigation, and cooperative robots.

REFERENCES

[1] Erfani, H., Zakizadeh, M. (2024). Meta-heuristic algorithms A comprehensive review. http://doi.org/10.13140/RG.2.2.34895.80801

[2] Almufti, S., Marqas, R., Asaad, R. (2019). Comparative study between elephant herding optimization (EHO) and U-turning ant colony optimization (U-TACO) in solving symmetric traveling salesman problem (STSP). Journal Of Advanced Computer Science & Technology, 8(2): 32. https://sciencepubco.com/index.php/JACST/article/view /29403.

[3] Nooruldeen, O., Baker, M.R., Aleesa, A.M., Ghareeb, A., Shaker, E.H. (2023). Strategies for predictive power: Machine learning models in city-scale load forecasting. e-Prime-Advances in Electrical Engineering, Electronics and Energy, 6: 100392. https://doi.org/10.1016/j.prime.2023.100392

[4] Rashid, T.A., Shekho Toghramchi, C.I., Sindi, H., Alsadoon, A., Bačanin, N., Umar, S.U., Shamsaldin, A.S., Mohammadi, M. (2021). An improved BAT algorithm for solving job scheduling problems in hotels and restaurants. Artificial Intelligence: Theory and Applications, 973: 155-171. https://doi.org/10.1007/978-3-030-72711-6_9

[5] Umar, S.U., Rashid, T.A., Ahmed, A.M., Hassan, B.A., Baker, M.R. (2024). Modified bat algorithm: A newly proposed approach for solving complex and real-world problems. Soft Computing, 28(13): 7983-7998. https://doi.org/10.1007/s00500-024-09761-5

[6] Kennedy, J., Eberhart, R. (1995). Particle swarm optimization. In Proceedings of ICNN'95-International Conference on Neural Networks, Perth, WA, Australia, pp. 1942-1948. https://doi.org/10.1109/ICNN.1995.488968

[7] Stützle, T., Dorigo, M. (2004). Ant colony optimization. https://www.researchgate.net/publication/36146886_Ant_Colony_Optimization.

[8] Bellman, R. (1952). On the theory of dynamic programming. Proceedings of the national Academy of Sciences, 38(8): 716-719. https://doi.org/10.1073/pnas.38.8.716

[9] Wang, G.-G., Deb, S., Gao, X.-Z., Dos Santos Coelho, L. (2016). A new metaheuristic optimisation algorithm motivated by elephant herding behaviour. International Journal of Bio-Inspired Computation, 8(6): 394-409. https://doi.org/10.1504/IJBIC.2016.081335

[10] Bisseling, R.H. (2017). Algorithms for the travelling salesman problem. 2017. https://studenttheses.uu.nl/handle/20.500.12932/29854

[11] Zhang, X.X., Shi, P.J., Liu, L.Y., Tang, Y., et al. (2010). Ambient TSP concentration and dustfall in major cities of China: Spatial distribution and temporal variability. Atmospheric Environment, 44(13): 1641-1648. https://doi.org/10.1016/j.atmosenv.2010.01.035

[12] Emambocus, B.A.S., Jasser, M.B., Hamzah, M., Mustapha, A., Amphawan, A. (2021). An enhanced swap sequence-based particle swarm optimization algorithm to solve TSP. IEEE Access, 9: 164820-164836. https://doi.org/10.1109/ACCESS.2021.3133493

[13] Marqas, R.B., Almufti, S.M., Othman, P.S., Abdulrahma, C.M. (2020). Evaluation of EHO, U-TACO and TS metaheuristics algorithms in solving TSP. Journal of XI'AN University of Architecture & Technology, 12(4). https://www.researchgate.net/publication/340739815_Evaluation_of_EHO_U-TACO_and_TS_Metaheuristics_algorithms_in_Solving_TSP.

[14] Robati, A., Barani, G.A., Nezam Abadi Pour, H., Fadaee, M.J., Rahimi Pour Anaraki, J. (2012). Balanced fuzzy particle swarm optimization. Applied Mathematical Modelling, 36(5): 2169-2177. https://doi.org/10.1016/j.apm.2011.08.006

[15] Li, J., Guo, L., Li, Y., Liu, C. (2019). Enhancing elephant herding optimization with novel individual updating strategies for large-scale optimization problems. Mathematics, 7(5): 395. https://doi.org/10.3390/math7050395

[16] Zhang, Z., Gao, Y. (2023). Solving large-scale global optimization problems and engineering design problems using a novel biogeography-based optimization with Lévy and Brownian movements. International Journal of Machine Learning and Cybernetics, 14(1): 313-346. https://doi.org/10.1007/s13042-022-01642-3

[17] Zhou, Y., He, F., Qiu, Y. (2017). Dynamic strategy based parallel ant colony optimization on GPUs for TSPs. http://scis.scichina.com/en/2017/068102-supplementary.pdf.

[18] Abdulfattah, G.M., Ahmad, M.N., Asaad, R.R. (2018). A reliable binarization method for offline signature system based on unique signer's profile. International Journal of Innovative Computing, Information and Control, 14(2): 573-586. https://doi.org/10.24507/ijicic.14.02.573

[19] Asaad, R.R., Abdulnabi, N.L. (2018). Using local searches algorithms with Ant colony optimization for the solution of TSP problems. Academic Journal of Nawroz University, 7(3): 1-6. https://doi.org/10.25007/ajnu.v7n3a193

[20] Muawanah, S., Muzayanah, U., Pandin, M.G., Alam, M.D., Trisnaningtyas, J.P. (2023). Stress and coping strategies of Madrasah's teachers on applying distance learning during COVID-19 pandemic in Indonesia. Qubahan Academic Journal, 3(4): 206-218.

[21] Umar, S.U., Rashid, T.A. (2021). Critical analysis: Bat algorithm-based investigation and application on several domains. World Journal of Engineering, 18(4): 606-620. https://doi.org/10.1108/WJE-10-2020-0495

[22] Laporte, G. (1992). The vehicle routing problem: An

overview of exact and approximate algorithms. European Journal of Operational Research, 59(3): 345-358. https://doi.org/10.1016/0377-2217(92)90192-C

[23] Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J. (2007). The Traveling Salesman Problem. https://www.ceas3.uc.edu/ret/archive/2017/ret/docs/readings/Project%203/Project%203_Introduction%20to%20TSP.pdf.

[24] Fischetti, M., Salazar-Gonzalez, J.J., Toth, P. (2007). The generalized traveling salesman and orienteering problems. In the Traveling Salesman Problem and Its Variations, Boston, pp. 609-662. https://doi.org/10.1007/0-306-48213-4_13

[25] Johnson, D.S., McGeoch, L.A. (2003). 8. The traveling salesman problem: A case study. Local Search in Combinatorial Optimization. pp. 215-310. https://doi.org/10.1515/9780691187563-011

[26] Reinelt, G. (2003). The Traveling Salesman: Computational Solutions for TSP Applications (Vol. 840). Springer. https://doi.org/10.1007/3-540-48661-5

[27] Golden, B.L., Raghavan, S., Wasil, E.A. (2008). The Vehicle Routing Problem: Latest Advances and New Challenges (Vol. 43). Springer Science & Business Media. https://doi.org/10.1007/978-0-387-77778-8

[28] Ali, F.H., Jassim, S.M. (2018). New improved heuristic method for solving travelling salesman problem. Iraqi Journal of Science, 59(4C): 2289-2300. https://doi.org/10.24996/ijs.2018.59.4C.16

[29] Lawler, E.L., Wood, D.E. (1966). Branch-and-bound methods: A survey. Operations Research, 14(4): 699-719. https://doi.org/10.1287/opre.14.4.699

[30] Boddy, M. (1991). Anytime problem solving using dynamic programming. In Proceedings of the Ninth National Conference on Artificial Intelligence, pp. 738-743. https://dl.acm.org/doi/abs/10.5555/1865756.1865791

[31] Lähdeaho, O., Hilmola, O.P. (2024). An exploration of quantitative models and algorithms for vehicle routing optimization and traveling salesman problems. Supply Chain Analytics, 5: 100056. https://doi.org/10.1016/j.sca.2023.100056

[32] Anwar, I.M., Salama, K.M., Abdelbar, A.M. (2015). Instance selection with ant colony optimization. Procedia Computer Science, 53: 248-256. https://doi.org/10.1016/j.procs.2015.07.301

[33] Yang, Y., Deng, Y., Xiao, B., Zhao, X. (2024). The method to integrate species explode and deracinate algorithm with particle swarm optimization algorithm. IEEE Access, 12: 52439-52451. https://doi.org/10.1109/ACCESS.2024.3387308

NOMENCLATURE

ACO	Ant Colony Optimization
AI	Artificial Intelligence
BB	Branch And Bound
DP	Dynamic Programming
EHO	Elephant Herding Optimization
PSO	Particle Swarm Optimization
SI	Swarm Intelligence

Greek symbols

B	Distance influence
P	Pheromone evaporation rate
Ω	Inertia weight

Subscripts

α	Pheromone Influence
k	Ant
t	Elephant