Harnessing Meta-Heuristic Algorithms to Optimize Travelling Salesman Problem

Aryan Kothari, V.R.N.S Nikhil, Namana Rohit, Apurvanand Sahay
Department of Computer Science and Engineering
Amrita School of Computing, Bangalore
Amrita Vishwa Vidyapeetham, India
bl.en.u4aie22005@bl.students.amrita.edu, bl.en.u4aie22062@bl.students.amrita.edu,
bl.en.u4aie22071@bl.students.amrita.edu, a_sahay@blr.amrita.edu

***Abstract*—The Traveling Salesman Problem (TSP) is a challenging combinatorial optimization problem classified as NP-hard. This paper investigates the performance of various meta-heuristic algorithms, including Brute Force, Greedy, Dynamic Programming, Divide and Conquer, Simulated Annealing, Christofides, Lin-Kernighan-Helsgaun (LKH), Optimized LKH, Ant Colony Optimization, and Particle Swarm Optimization. The goal is to determine their effectiveness in solving large-scale TSP instances by focusing on solution quality and computational efficiency. Using a dataset of 256 cities, each algorithm was evaluated based on execution time and the cost of the final path. The Christofides algorithm proved the most balanced, with a time of 0.5027 seconds and a cost of 1321, making it the best in terms of cost efficiency. Simulated Annealing emerged as the fastest, completing in 0.095 seconds, while the Brute Force method, despite guaranteeing optimal solutions, was impractical for large datasets due to its exponential time complexity. The Greedy algorithm, although faster than some, often resulted in suboptimal solutions. This study underscores the importance of selecting the appropriate algorithm based on specific problem parameters and highlights the potential for future research in hybrid algorithms and parallel computing to further enhance performance.**

***Index Terms*—Travelling Salesman Problem, Meta-heuristic algorithms, Optimization techniques, Comparative analysis**

I. Introduction

The Traveling Salesman Problem (TSP) is a classic optimization challenge within the field of combinatorial optimization, posing the question: "Given a collection of cities and the distances between every pair of them, how can we determine the shortest route that includes each city exactly once and ends at the starting point?" The problem is famously difficult not only due to its simple and relatable premise but also because of its inherent computationally expensive approach. The TSP is categorised as an NP-hard, which means that there is no known efficient algorithm to find an exact solution in polynomial time as the number of cities increases. This exponential increase in complexity makes it impossible to look for exact solutions especially when dealing with large data sets; therefore, approximate solutions are sought. The paper is dedicated to meta heuristics that provide good approximations selecting the best solution from a vast space in a reasonable amount of time. Thus, it is the objective of the paper to outline the relevance of these approaches in generating effective solutions, at the overall goal of enhancing the TSP quality of the solutions needed for academic research and problem-solving in logistics, planning, among other areas.

The enduring appeal of the Traveling Salesman Problem lies in its broad applicability and the ongoing challenge it presents in the field of optimization. Despite decades of research, the quest for efficient and effective solutions remains a vibrant area of inquiry, particularly as problem sizes increase with demands from sectors like logistics and network design [20]. The central open research question this paper addresses is the efficacy of meta-heuristic algorithms in solving TSP for large-scale problems where traditional approaches fail due to computational constraints. Meta-heuristics are favored for their ability to find good enough solutions within reasonable time frames, a necessity in practical applications where exact solutions are computationally prohibitive. This research critically evaluates the potential of various meta-heuristic strategies, such as Genetic Algorithms, Simulated Annealing, and Particle Swarm Optimization, to discover which methods or combinations thereof can optimize performance and solution quality. Through this exploration, the paper aims to advance the understanding of algorithmic design and optimization in solving complex, NP-hard problems like the TSP.

The structure of this paper is organized as follows: relevant literature and the gap between current & proposed solutions are given in Section II, providing a theoretical foundation and highlighting previous advancements in solving TSP with meta-heuristic algorithms. Section III describes the methodology, including the dataset and detailed algorithmic approaches used in the comparative analysis. Section IV presents the experimental results, discussing the performance of each algorithm in terms of time efficiency and solution quality. Finally, Section V concludes with a discussion of the implications of the findings, offering insights for future research and practical algorithm selection for solving TSP in various real-world applications.

II. Literature Survey

The Christofides et al.[1] work is characterized by the study of heuristics for the optimization of combinatorial classes, with a specific focus on the solution of the Travelling Salesman Problem (TSP). The review of this literature includes the different heuristics that have been developed and the suggestion of a novel one with a polynomial increase

of $O(n^3)$.The algorithm finds the solution to a minimum cost perfect matching problem which is its first substep. The second substep is the computation of a shortest-spanning tree. Altogether, the algorithm is notable as it has a guaranteed strict lower bound of the worst-case ratio of solution quality to optimal solution which actually is below 3/2. It is an increase by about 50% of the previously known rations, which represents the extensive undertaking to improve heuristic methods and come up with strategies for these classes of complex combinatorial problems. Furthermore, X. Chen et. al[2] introduce a method of multicriterial travelling salesman problem (MOTSP) solving with the help of a new approach of genetic algorithm-based and physarum computational model (PCM.)Acknowledging several weaknesses of contemporary GA-based MOTSP heuristics including local optimum and shortage of diversity, authors are offering new initializing population and diversity preservation methods. The high effectiveness of the proposed method is based on using two strategies: the particle swarm optimization with the hill climbing and the genetical algorithm drives to the result that the proposed method is better than non-dominating sorting genetic algorithm II as shown in our experiments. Through this paper the reader will not only compile detailed formulations and comparisons but also develop a general understanding of nature-inspired computational elements to help solve complex combinatorial optimization problems such as MOTSP. Guo Chen et al.[3] highlight that identifying the best solutions to TSP, ultimate attention is paid on heuristic and metaheuristic algorithms that can provide approximate solutions for this NP- complete issue. Christofides algorithm makes the spotlight in that it is the example of the old technique producing solutions to within 1%. By using both minimum spanning trees and perfect matchings, this approach is enhanced 5 times greater at the optimal parameter. The corresponding techniques, in addition to these methods, is also an effective version of them: C-N-GA (Christofides Algorithm & Nearby Measures & Genetic Algorithm).

The paper by Chutian et al. [4] focuses on solving the TSP using a Genetic Algorithms approach and examines the impact of parameters like mutation and crossover probabilities on GA performance. Their research aims to enhance GA for faster convergence, with improved GA showing better solution quality and running speed, highlighting the importance of pre-setting parameters and conducting experiments to understand their effects. The paper by Chang et al. [5] address inefficiency and environmental impact in last-mile delivery within e-commerce by integrating parcel locker allocation, route scheduling, and customer self-collection preferences, developing an Iterated Local Search (ILS) algorithm with novel operators to optimize routes and reduce emissions. Computational experiments, including real-world scenarios, demonstrate the algorithm's effectiveness and competitiveness, promoting sustainable e-commerce logistics. The paper by Nemani et al. [6] conduct a comparative study on TSP algorithms by using 3 different algorithms and reviewing methods highlighting TSP's NP-hard nature and distinct approaches to solving it.
The paper of Nergiz Sözen[7] is an elaboration of a hybrid optimization method drawing upon the Genetic Algorithms (GA) and the combination of numerous selection, crossover, and mutation methods, so as disclose the enhanced performance of the proposed technique on the various datasets with respect to the Use of Roulette Wheel Selection and Stochastic Universal Selection with Partially Mapped, Cycle, and Finally, the article [8] presents updated SSA (nicholas) mechanisms to handle problems regarding local optima and stagnation by suggesting genetic operators, adaptive weights, and dynamic entities that are better than existing models including GA, SA, PSO, GWO, and ACO. Besides that the paper [9] creates a DNA computing algorithm for Family Traveling Salesperson Problem (FTSP), using DNA capabilities as a medium for parallelism and storage is expected to produce good results. This algorithm has O(N$\hat{2}$) complexity and shows the potential power of DNA computation in combinatorial problems. In aggregate, this data indicates continuing research on the subject, emphasizing the development of new hybrids and solutions for TSP and its related problems.

In the study conducted by Brucal et al[10], various algorithms were applied to TSP to evaluate their efficiency and effectiveness. The algorithms investigated include GA, Nearest Neighbor, ACO. The comparative analysis revealed that the Nearest Neighbor algorithm exhibited the shortest execution time, averaging 0.24 seconds, while the Genetic Algorithm required the longest time, averaging 13.70 seconds. ACO, although effective in theoretical applications, showed intermediate performance with an average execution time of 0.42 seconds. The paper by Mason Helmick[11] Analyzes different metaheuristics and TSP algorithms, the work emphasizes the high degree of accuracy and efficiencies achieved in the search for optimal solutions. The most 'exact' algorithms Tabu search and simulated annealing work well in smaller instances and more 'complex' methods Genetic algorithms and Ant systems colonization prove themselves in harder fixtures. While its functional implementation looks efficient enough its sop as the problem size becomes larger, its code vulnerability needs to be fixed to stay ahead of the advanced evolutionary algorithm computers with better process abilities.[14]

The paper by Kevin et al.[12] gives a full description of the general methods and approaches to the Traveling Salesman Problem (TSP) and discusses them in detail, taking into account the fact that it is not a simple task to search for the optimal solution. It analyzes the merits of SA, TS, GA, and the recent ACO methods of optimization. The research emphasizes mixed algorithms that have some constituent elements of different metaheuristics so that they can enjoy strengths and overlook weaknesses hence offering the possibility for tackling the most of the TSP problems with precision. This work recommends that future addition of quality to ACO techniques will be achieved through

optimization and integration.[15]

The existing literature on solving the Traveling Salesman Problem (TSP) focuses on various heuristic and meta-heuristic algorithms to tackle the NP-hard nature of the problem. Classic heuristics, such as those by Christofides et al., offer guaranteed solution quality within a bound, using minimum spanning trees and perfect matchings. Recent approaches, like those by X. Chen et al., incorporate multicriterial strategies with genetic algorithms and physarum computational models to improve diversity and local optima issues. Comparative analyses by Brucal et al. and Mason Helmick highlight the effectiveness and limitations of algorithms like Genetic Algorithms, Ant Colony Optimization, and Nearest Neighbor, with varying degrees of efficiency and accuracy depending on problem size and complexity.

Despite the advancements, a significant gap remains in achieving an optimal balance between solution quality and computational efficiency for large-scale TSP instances. The literature indicates a trade-off between execution time and accuracy, with some algorithms like the Nearest Neighbor offering quick solutions at the expense of optimality, while others like Genetic Algorithms provide near-optimal solutions but are computationally intensive. The proposed methodology in this paper aims to address these gaps by conducting a comprehensive evaluation of a broader spectrum of meta-heuristic algorithms, including Brute Force, Greedy, Dynamic Programming, Divide and Conquer, Simulated Annealing, Christofides, Lin-Kernighan-Helsgaun (LKH), Optimized LKH, Particle Swarm Optimization (PSO), and Ant Colony Optimization (ACO). Besides, it compares these algorithms against a uniform set of 256 cities and fine-tunes these algorithms' parameters in a very rigorous manner. Thus, by comparing the execution time and the quality of the solutions, the study gives a better understanding of the advantages and the weaknesses of each algorithm, which can help to make the proper choice and combine different algorithms in solving the certain practical TSP problems.

III. Methodology

A. Dataset Description

The dataset used in this study is made up of randomly generated coordinates for 256 cities, represented within a 1000x1000 grid. Each city is modelled as an instance of the `City` class, which includes attributes for x and y coordinates and a method to compute the Euclidean distance to another city using the formula $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$. The data is generated by the `generate_cities(size)` function, which creates a list of `City` objects with random coordinates. This function is used within `write_cities_and_return_them(size)`, which writes the coordinates to a text file named `cities_<size>.data`. The `read_cities(size)` function reads these coordinates from the corresponding text file in the `test_data` directory and reconstructs the list of `City` objects. The `path_cost(route)` function computes the total travel distance of a given route by summing the distances between consecutive cities in the route. Additionally, the `visualize_tsp(title, cities, sample_size=100)` function provides an animated visualization of the TSP route using Matplotlib, generating all possible permutations of the cities (or a sample subset to speed up the animation) and animating the paths.

B. Algorithm Description

1) **Brute Force Approach**:The direct approach that is applied to solve the TSP entails the determination of the cost of all possible permutations of the cities in order to determine which of the permutations is the cheapest. It produces all possible permutations to traverse all the cities exactly once and get back to the starting point, computing the total distance for each of the possibilities. Here the basic approach is to fix one layer and find the best possible solution for the rest of the layers but though very efficient and sure of delivering the solution with highest value of objective function it has exponential time complexity of $O(n!)$ The accuracy is again 100% since it analyses all the potential routes.
2) **Simulated Annealing Approach**: The simulated annealing algorithm solves the TSP by iteratively improving a single solution. It starts with an initial route and explores neighboring routes by making small changes, accepting changes that improve the total travel cost and, with decreasing probability, changes that increase the cost to escape local optima. The probability of accepting worse solutions decreases over time according to a cooling schedule, typically modeled by $T = T_0 \cdot \alpha^k$, where T is the temperature, T_0 is the initial temperature, α is the cooling rate, and k is the iteration number. The method is effective for finding good solutions in reasonable time but does not guarantee the optimal solution. Its time complexity is $O(n^2 \log n)$. The accuracy varies but typically provides near-optimal solutions.
3) **Christofides' Algorithm**: This heuristic algorithm for the TSP gives us a solution within One and a Half times the optimal tour length for metric TSPs. The algorithm constructs a minimum spanning tree (MST), finds a minimum-weight perfect matching for the odd-degree vertices, and combines these to form an Eulerian circuit. This circuit is then converted into a Hamiltonian circuit by shortcutting repeated vertices. The MST is computed using Kruskal's or Prim's algorithm, and the matching is found using the Blossom algorithm. The time complexity is $O(n^3)$. The accuracy is guaranteed to be at least 66.7% of the optimal solution.
4) **Divide and Conquer Approach**: The divide and conquer approach for TSP involves dividing the set of cities into smaller subsets, solving the TSP for each subset, and then combining the solutions. This method uses recursive partitioning and merging strategies. The efficiency depends on the specific division and merging strategy used, typically offering better performance than

brute force but still having an exponential time complexity. The accuracy can be high but is not guaranteed to be optimal.

5) **Dynamic Programming Approach**: The dynamic programming approach, specifically the Bellman-Held-Karp algorithm, solves the TSP by storing solutions to subproblems and combining them to solve larger problems. It uses a state-space representation where each state represents a subset of cities and the minimum cost to reach them. The cost function can be represented as $C(S, j) = \min_{i \in S, i \neq j}[C(S - \{j\}, i) + d_{ij}]$, where S is the set of visited cities, j is the current city, i is the previous city, and d_{ij} is the distance between cities i and j. The time complexity is $O(n^2 2^n)$ and space complexity is $O(n2^n)$. This approach is guaranteed to find the optimal solution, with 100
6) **Genetic Algorithm Approach**: Genetic algorithms solve the TSP by simulating the process of natural selection. A population of routes is evolved over generations using selection, crossover, and mutation operators. The fitness function evaluates each route, often using the total travel distance. Over successive generations, the population converges towards an optimal or near-optimal solution. This heuristic approach generally has a time complexity of $O(g \cdot n^2)$, where g represents the number of generations and n denotes the number of cities. The accuracy is high, but not guaranteed to be optimal.
7) **Greedy Algorithm Approach**: The greedy algorithm for TSP constructs a tour by repeatedly choosing the shortest available edge that does not form a cycle until the tour is complete. Starting from an initial city, it adds the nearest unvisited city to the tour. This heuristic method has a time complexity of $O(n^2)$. It is simple and fast but often provides suboptimal results, with accuracy depending on the problem instance.
8) **Lin-Kernighan-Helsgaun (LKH) Algorithm**: The LKH algorithm is an advanced heuristic for solving the TSP, based on the Lin-Kernighan heuristic. It iteratively improves an initial solution by performing edge exchanges. The algorithm incorporates various sophisticated techniques to escape local optima and find high-quality solutions efficiently. The time complexity varies but is typically much better than simpler heuristics. The accuracy is usually very high, often close to the optimal solution.
9) **Optimised Lin-Kernighan-Helsgaun (LKH) Algorithm**: This is an enhanced version of the LKH algorithm, incorporating additional optimization techniques and refinements to improve solution quality and computational efficiency further. The exact complexity can vary, but it is designed to be more efficient than the standard LKH algorithm. The accuracy is generally very high, often approaching the optimal solution.
10) **Particle Swarm Optimization (PSO) Approach**: The PSO algorithm similar to the one implemented by [16] solves the TSP by simulating the social behaviour of particles (potential solutions) in a search space. Each particle adjusts its position based on its own experience and the experience of neighbouring particles. The position update is often modeled by $v_i(t+1) = \omega v_i(t) + c_1 r_1(p_i - x_i(t)) + c_2 r_2(p_g - x_i(t))$, where $v_i(t)$ is the velocity, $x_i(t)$ is the position, p_i is the personal best position, p_g is the global best position, ω is the inertia weight, and c_1, c_2, r_1, r_2 are constants and random values. This iterative process converges towards a high-quality solution. $O(n \cdot t)$ is the time complexity, where t is iteration number. The accuracy is usually high but not guaranteed to be optimal.

Algorithm	**Accuracy**
Brute Force	100%
Simulated Annealing	Near-optimal
Christofides' Algorithm	At least 66.7%
Divide and Conquer	High, but not guaranteed optimal
Dynamic Programming	100%
Genetic Algorithm	High, but not guaranteed optimal
Greedy Algorithm	Suboptimal
Lin-Kernighan-Helsgaun (LKH)	Very high, close to optimal
Optimised Lin-Kernighan-Helsgaun	Very high, approaching optimal
Particle Swarm Optimization	High, but not guaranteed optimal

TABLE I
COMPARISON OF ACCURACY OF THE OPTIMISATION ALGORITHMS

Table I provides a comparison of the accuracy of various optimization algorithms for solving the Traveling Salesman Problem (TSP). Brute Force and Dynamic Programming (Bellman-Held-Karp) methods achieve 100% accuracy, guaranteeing optimal solutions, but are impractical for large datasets due to their computational complexity. Simulated Annealing, Genetic Algorithm, and Particle Swarm Optimization (PSO) offer high accuracy and are suitable for finding near-optimal solutions in a reasonable timeframe, making them useful for larger problems where exact methods are infeasible. Christofides' Algorithm guarantees at least 66.7% accuracy, providing a good approximation for specific cases. The Lin-Kernighan-Helsgaun (LKH) heuristic and its optimized variant deliver very high accuracy, approaching optimal solutions, and are generally more efficient than simpler heuristics, making them suitable for both small and large instances. Divide and Conquer methods offer high accuracy but without guaranteed optimality, making them a versatile choice depending on the problem context. Greedy algorithms, while suboptimal, provide fast and straightforward solutions for smaller or less critical instances. Table II compares the best case, average case, and worst case time complexities of various optimization algorithms for the Traveling Salesman Problem (TSP). Brute Force consistently has $O(n!)$ complexity, reflecting its exhaustive nature. Simulated Annealing varies from $O(n \log n)$ to $O(n^3 \log n)$, indicating variability based on problem configu-

Algorithm	Best Case	Average Case	Worst Case
Brute Force	$O(n!)$	$O(n!)$	$O(n!)$
Simulated Annealing	$O(n \log n)$	$O(n^2 \log n)$	$O(n^3 \log n)$
Christofides' Algorithm	$O(n^3)$	$O(n^3)$	$O(n^3)$
Divide and Conquer	Exponential	Exponential	Exponential
Dynamic Programming	$O(n^2 2^n)$	$O(n^2 2^n)$	$O(n^2 2^n)$
Genetic Algorithm	$O(g \cdot n)$	$O(g \cdot n^2)$	$O(g \cdot n^3)$
Greedy Algorithm	$O(n \log n)$	$O(n^2)$	$O(n^2)$
Lin-Kernighan-Helsgaun	$O(n \log n)$	Varies	Varies
Optimised LKH	$O(n \log n)$	Varies	Varies
PSO	$O(n \cdot t)$	$O(n \cdot t)$	$O(n \cdot t)$

TABLE II
COMPARISON OF TIME COMPLEXITIES FOR VARIOUS OPTIMIZATION ALGORITHMS IN DIFFERENT SCENARIOS

ration. Christofides' Algorithm maintains a stable $O(n^3)$ complexity across all cases. Divide-and-conquer approaches are always exponential. Dynamic Programming (Bellman-Held-Karp) consistently exhibits $O(n^2 2^n)$, making it impractical for very large problems despite optimal accuracy. Genetic Algorithms range from $O(g \cdot n)$ to $O(g \cdot n^3)$, depending on generation and population size. Greedy Algorithms show $O(n \log n)$ to $O(n^2)$, highlighting quick but often suboptimal performance. Lin-Kernighan-Helsgaun (LKH) and its optimized version have variable complexity, typically better than simpler heuristics. Particle Swarm Optimization (PSO) shows consistent $O(n \cdot t)$, dependent on iterations and particles. This comparison highlights the trade-offs between computational complexity and solution quality for each algorithm.

C. Evaluation Metrics

The performance of each algorithm was evaluated based on:

- **Solution Quality**: The total travel distance of the route found.
- **Computational Efficiency**: Time complexity and convergence rate.

D. Experimental Setup

The experiments were done on a system with the below specifications:

- Processor: Intel Core i5-11300K
- RAM: 8 GB
- OS: Windows 11
- Python Version: 3.7
- Key Libraries: NumPy, SciPy, Matplotlib, NetworkX

Optimal parameter values for the individual algorithms were always adjusted before a final design was implemented to ensure optimal performance.

IV. IMPLEMENTATION

Here, in this section, we give a brief discussion regarding the performance comparison of a number of optimization algorithms in solving TSP followed by the criteria and processes of their performance evaluation. For relationships between variables, the workflow of the experiment is in the figure. 1.

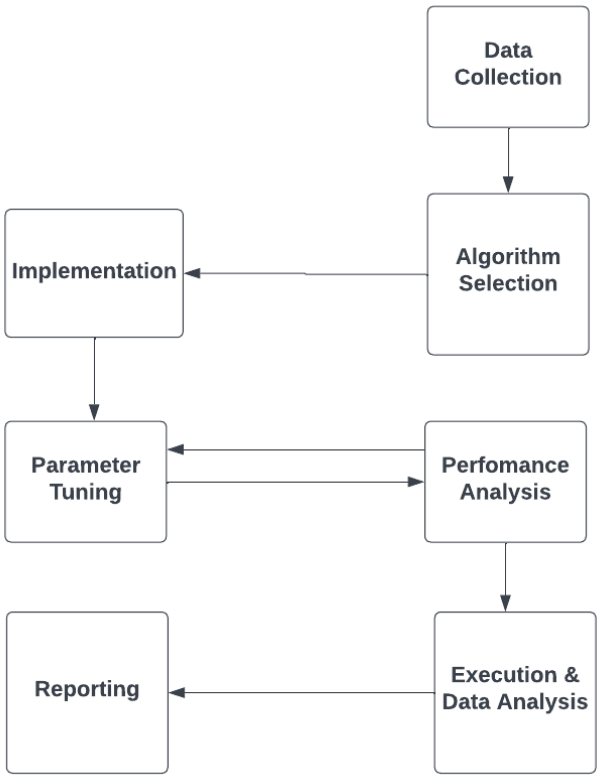

Fig. 1. Workflow Diagram

A. Rating Criteria

Execution Time is determined being the amount of time that each of the algorithms takes to come up with the solution in seconds. If an algorithm requires more than 600 Seconds or 10 minutes, we consider it is too much and not suitable for this size of problem.

Cost of the Final Path is used as the cost of the final path since the total travel distance is a remarkable component of the whole path. This shows how good the solution is where the lower the cost the better the solution being offered.

B. Evaluation Process

For each algorithm, the following steps are carried out:

1) **Execution Time Measurement**: The start time is recorded before the algorithm begins processing. The end time is recorded after the algorithm completes, and the execution time is calculated. If the execution time exceeds 600 seconds, the algorithm is marked as "too much".
2) **Path Cost Calculation**: The total distance of the route generated by the algorithm is computed. This distance is considered the cost of the final path.
3) **Visualization**: A visual representation of the computed route is generated using Matplotlib. This helps in understanding the efficiency and effectiveness of the algorithm visually.

C. Example Visualization

Below is an example of how the points look and a sample visualization of the computed TSP route using one of the algorithms as shown in Fig. 2.

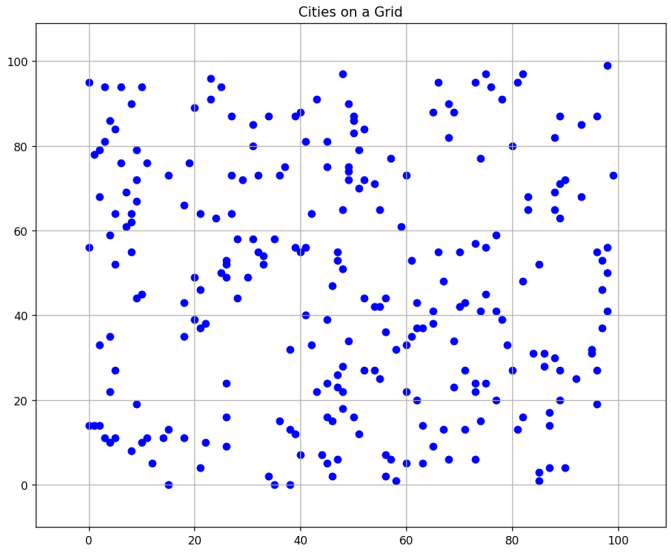

Fig. 2. Plotting the cities

D. Hyperparameter Table

Table III summarizes the key hyperparameters for various algorithms used. Each algorithm has distinct parameters that influence its performance and solution quality. For example, the Brute Force method utilizes permutation methods and timing, while the Christofides algorithm incorporates Euclidean distances and specific matching and circuit techniques. Advanced methods like Simulated Annealing and Ant Colony Optimization include parameters for temperature decay, pheromone evaporation, and iteration control. This comparison helps in understanding the specific configurations required for optimizing each algorithm's execution on the TSP.

TABLE III
TSP ALGORITHM HYPERPARAMETERS

Algorithm	**Hyperparameter Values**
Brute Force	`permutation_method`: `itertools.permutations`, `distance_calculation`: `path_cost function`, `timing`: Start and End using `time`
Christofides	`distance`: Euclidean, `MST`: Kruskal, `matching`: Blossom, `circuit`: Fleury
Divide & Conquer	`split_dimension`: Longer Dimension (x or y), `merge_strategy`: Minimum Cost
Dynamic Programming	`distance_matrix`: Precomputed, `subproblems`: Combinations of cities
Greedy	`initial_city`: First city, `selection`: Nearest neighbor, `plot`: False
Lin-Kernighan	`initial_route`: Random, `improvement`: 2-opt swaps, `distance_matrix`: Precomputed
Lin-Kernighan with Candidate List	`initial_route`: Random, `improvement`: 2-opt swaps, `distance_matrix`: Precomputed, `candidates`: 20
PSO	`num_particles`: 10, `iterations`: 30, `initial_route`: Nearest neighbor, `velocity_update`: Deterministic swaps
Simulated Annealing	`temperature`: sqrt(n), `alpha`: 0.999, `stopping_temperature`: 1e-8, `stopping_iter`: 1500, `initial_solution`: Greedy
Ant Colony Optimization	`n_ants`: 20, `n_best`: 5, `n_iterations`: 100, `decay`: 0.95, `alpha`: 1, `beta`: 2

V. RESULTS AND DISCUSSION

To establish a benchmark, we initially ran the brute force algorithm for a problem instance of 20 cities, which took approximately 0.08 seconds. Given the exponential nature of the brute force approach, where the time complexity is O(n!), scaling this up to 256 cities results in an impractical computation time. Specifically, the estimated time taken for 256 cities would be infeasibly large and around 1500 years. The Greedy algorithm provided a more practical solution compared to brute force. For the given problem instance, it was completed in 20.795 seconds with a cost of 1466.473. This approach constructs a path by always choosing the nearest unvisited city, making it straightforward but suboptimal for larger instances. While faster than brute force, the cost reflects that it may not always find the shortest possible path. The Divide and Conquer algorithm performed better in terms of time compared to the Greedy approach, completing in 15.931 seconds.

This method works by breaking down the problem into smaller sub-problems, solving each independently, and then combining the results. The solution cost was 1592, slightly higher than Greedy but achieved in less time, indicating a trade-off between computation time and solution quality. Fig. 3 illustrates the visualisation of the divide & conquer approach

Dynamic Programming (DP) is another approach that aims to improve efficiency by storing the results of subproblems to avoid redundant calculations. However, for our larger instances, the DP algorithm exceeded our time limits, rendering it impractical for very large problem sizes. This highlights the scalability issue of DP, as its time complexity is also exponential, although more efficient than brute force in smaller instances.

Simulated Annealing (SA) yielded impressive results, completing in just 0.095 seconds with a cost of 1612. SA replicates the actual process of heating a particular material and then gradually reduces the temperature to reduce the number of

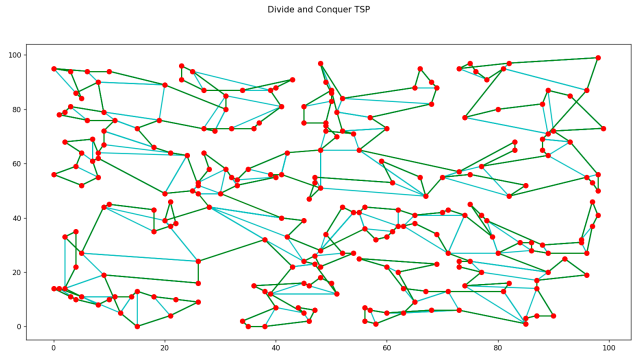

Fig. 3. Divide & Conquer Visualisation

defects present, hence identifying the best solution. This is useful for the efficiency and speed of the algorithm and its ability to get out of local optima by occasionally accepting worse solutions. Fig. 4 illustrates the progress of the Simulated

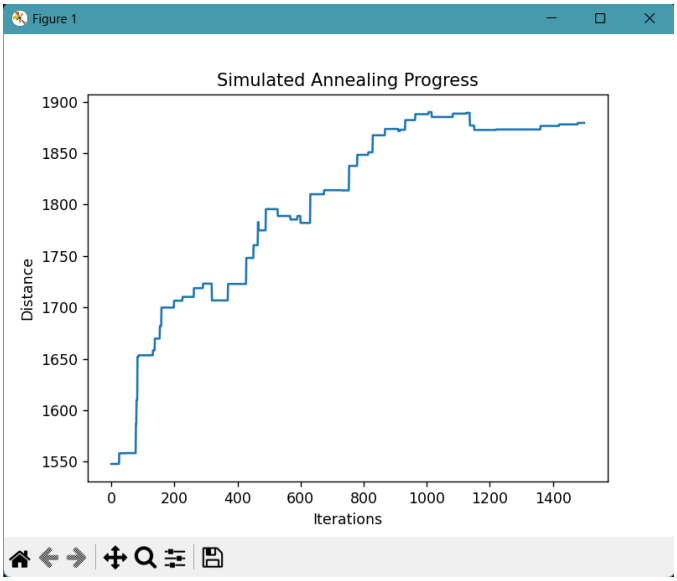

Fig. 4. Simulated Annealing

Annealing (SA) algorithm in solving the Traveling Salesman Problem (TSP) over 1500 iterations. Initially, the distance increases rapidly as the algorithm explores various solutions, allowing it to escape local minima. During the mid-phase, the distance continues to rise with occasional fluctuations, reflecting the algorithm's balance between exploration and fine-tuning of solutions. In the stabilization phase, the distance gradually stabilizes around 1850-1900, indicating convergence to a stable solution. This demonstrates the SA algorithm's effectiveness in progressively optimizing the tour.

The heuristic for TSP known as the Christofides algorithm balanced time and cost and the algorithm took 0. 5027 seconds at the rate of 1321. The elegance of this algorithm is that it guarantees a solution within 1.5 times the optimal cost by first constructing a minimum spanning tree and then using minimum weight perfect matching and joining from a tour. It is polynomial time complexity and performance guarantee that make it suitable for TSP.

The proposed Algorithm, Particle Swarm Optimization (PSO), was shown to be both efficient and of high quality, finishing in 0.16 seconds with a cost of 1,431. PSO, based on the social behavior of birds flying, enhances the TSP by making the particles (possible solutions) move and update the positions with the help of its own and neighbors' data. Exploration and exploitation are integrated well in this method which aids in identifying high quality solutions within a short span of time. The table gives the performance comparison

Algorithm	**Time Taken (s)**	**Cost**
Brute Force	Too high	Unable to Calculate
Greedy	20.795	1466.473
Dynamic Programming	Time limit exceeded	Unable to Calculate
Divide and Conquer	15.931	1592
Anneal	0.095	1612
Christofides	0.5027	1321
LKH	Time limit exceeded	Unable To Calculate
Optimized LKH	12.09	1799.33
PSO	0.16	1431
Ant Colony	61.67	1470.10

TABLE IV
PERFORMANCE COMPARISON OF TSP ALGORITHMS

of different TSP algorithms in terms of execution time and solution cost. The best algorithm in both cost and time is the Christofides algorithm with a cost of 1321 and time of 0. 5027 seconds. Similar to the Genetic Algorithm, the performance of Simulated Annealing is also very high with a very short time complexity of 0. 095 seconds and the total cost of 1612 dollars. The poorest algorithm is the brute force method, which is not efficient for larger instances since it has an exponential time bound. The line chart shown in Fig. 5 illustrates the performance of various TSP algorithms in the unit of solution cost and time efficiency. The blue line represents the time taken, while the green line indicates the cost of the solution. Key observations include that Simulated Annealing and Christofides are the fastest algorithms, with Christofides also providing the lowest-cost solution. In contrast, Ant Colony Optimization is the slowest but yields a relatively low cost. The Optimized Lin-Kernighan algorithm takes a moderate amount of time but results in the highest cost. This analysis highlights the trade-offs between execution time and solution quality, aiding in the selection of the most appropriate algorithm for specific needs.

Given the balance between computation time and solution quality, the Christofides algorithm stands out as the most efficient for our needs. However, considering implementation complexity and specific use case requirements, we may choose to use Simulated Annealing(For its speed) or Particle Swarm Optimization (PSO)(For its overall balance of speed and time) for their ease of use and adaptability to larger instances, while still providing good results.

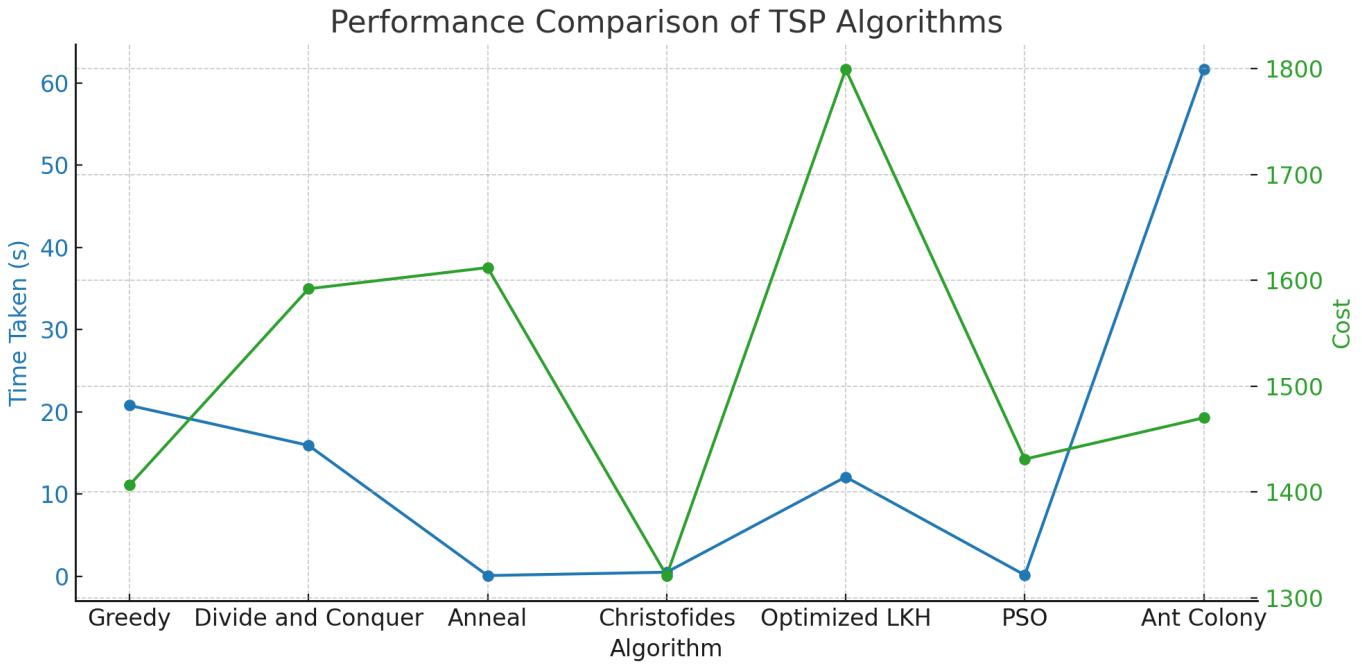

Fig. 5. Comparative Line Chart

VI. CONCLUSION & FUTURE WORKS

In conclusion, this study comprehensively analysed and compared various metaheuristic algorithms employed for solving the Traveling Salesman Problem (TSP). The study highlights the inherent trade-offs between computational efficiency and solution quality in various algorithmic approaches. While exact methods like brute force and dynamic programming guarantee optimal solutions, they become impractical for larger problems due to their high computational demands. On the other hand, the Christofides algorithm stands out as an effective heuristic, balancing efficiency and accuracy by providing solutions within 1.5 times the optimal cost.

Simulated Annealing and Particle Swarm Optimization also show strong performance, with Simulated Annealing being the fastest and Particle Swarm Optimization remaining stable even for larger problems. This research underscores the need to choose algorithms carefully based on specific requirements, taking into account the desired solution quality, computational efficiency, and scalability.

Future research aims to develop hybrid algorithms that combine different metaheuristic techniques and leverage parallel and distributed computing to improve performance for large-scale Traveling Salesman Problem (TSP) instances. These findings contribute to the ongoing discussion on optimizing algorithms for the TSP and guide decision-making in various real-world applications.

REFERENCES

[1] Christofides, N., 2022, March. Worst-case analysis of a new heuristic for the travelling salesman problem. In Operations Research Forum (Vol. 3, No. 1, p. 20). Cham: Springer International Publishing.

[2] X. Chen, Y. Liu, X. Li, Z. Wang, S. Wang and C. Gao, "A New Evolutionary Multiobjective Model for Traveling Salesman Problem," in IEEE Access, vol. 7, pp. 66964-66979, 2019, doi: 10.1109/ACCESS.2019.2917838.

[3] G. Chen, X. Shi, J. Kan, F. Dong and K. Chen, "A New Heuristic Algorithm Based on Christofides Algorithm and Nearby Measures for TSP Problem," 2023 2nd International Conference on Machine Learning, Cloud Computing and Intelligent Mining (MLCCIM), Jiuzhaigou, China, 2023, pp. 448-453, doi: 10.1109/MLCCIM60412.2023.00072.

[4] . Sun, Chutian. "A study of solving traveling salesman problem with genetic algorithm." In 2020 9th International Conference on Industrial Technology and Management (ICITM), pp. 307-311. IEEE, 2020.

[5] L. Jiang, H. Chang, S. Zhao, J. Dong and W. Lu, "A Travelling Salesman Problem With Carbon Emission Reduction in the Last Mile Delivery," in IEEE Access, vol. 7, pp. 61620-61627, 2019, doi: 10.1109/ACCESS.2019.2915634.

[6] R. Nemani, N. Cherukuri, G. R. K. Rao, P. V. V. S. Srinivas, J. J. Pujari and C. Prasad, "Algorithms and Optimization Techniques for Solving TSP," 2021 Fifth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Palladam, India, 2021, pp. 809-814, doi: 10.1109/I-SMAC52330.2021.9640907.

[7] N. Sözen, "Solving Travelling Salesman Problem: A Hybrid Optimization Algorithm," 2019 1st International Informatics and Software Engineering Conference (UBMYK), Ankara, Turkey, 2019, pp. 1-6, doi: 10.1109/UBMYK48245.2019.8965478.

[8] C. Wu, X. Fu, J. Pei and Z. Dong, "A Novel Sparrow Search Algorithm for the Traveling Salesman Problem," in IEEE Access, vol. 9, pp. 153456-153471, 2021, doi: 10.1109/ACCESS.2021.3128433. keywords: Heuristic algorithms;Optimization;Traveling salesman problems;Search problems;Particle swarm optimization;Genetic algorithms;Statistics;Sparrow search algorithm;traveling salesman problem;greedy algorithm;genetic operators;sin-cosine search strategy;combinatorial optimization,

[9] X. Wu, Z. Wang, T. Wu and X. Bao, "Solving the Family Traveling Salesperson Problem in the Adleman–Lipton Model Based on DNA Computing," in IEEE Transactions on NanoBioscience, vol. 21, no. 1, pp. 75-85, Jan. 2022, doi: 10.1109/TNB.2021.3109067. keywords: DNA;Electron tubes;Computational modeling;DNA computing;Heuristic algorithms;Traveling salesman problems;Quantum computing;The family traveling salesperson problem;DNA computing;the Adleman-Lipton model;NP-complete problem,

[10] X. Wu, Z. Wang, T. Wu and X. Bao, "Solving the Family Traveling Salesperson Problem in the Adleman–Lipton Model Based on DNA Computing," in IEEE Transactions on NanoBioscience, vol. 21, no. 1, pp. 75-85, Jan. 2022, doi: 10.1109/TNB.2021.3109067.

[11] S. G. E. Brucal and E. P. Dadios, "Comparative analysis of solving traveling salesman problem using artificial intelligence algorithms," 2017IEEE 9th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM), Manila, Philippines, 2017, pp. 1-6, doi: 10.1109/HNICEM.2017.8269423.

[12] Helmick, Mason, "The Traveling Salesman Problem: An Analysis and Comparison of Metaheuristics and Algorithms" (2022). Senior Honors Theses. 1195.https://digitalcommons.liberty.edu/honors/1195

[13] Kevin M. Curtin,* Gabriela Voicu,† Matthew T. Rice* and Anthony Stefanidis "A Comparative Analysis of Traveling Salesman Solutions from Geographic Information Systems"Transactions in GIS, 2014, 18(2): 286–301.

[14] Uthayasuriyan, Agash & Gnanavel, Hema & UV, Kavvin & Mahitha, Sabbineni & Jeyakumar, Gurusamy. (2023). A Comparative Study on Genetic algorithm and Reinforcement Learning to Solve the Traveling Salesman Problem. Research Reports on Computer Science. 1-12. 10.37256/rrcs.2320232642.

[15] T. Srinivas Rao; A performance evaluation of ACO and SA TSP in a supply chain network. AIP Conf. Proc. 19 July 2017; 1859 (1): 020098. https://doi.org/10.1063/1.4990251

[16] Dr. T. Srinivas Rao and Gopi, S., "Solving a Reverse Supply Chain TSP by Genetic Algorithm", in ICMME 2015 conference organized by SCSVM university Kanchipuram , 2015.

[17] Soumya Krishnan, M., Vimina, E.R. (2021). E-commerce Logistic Route Optimization Deciphered Through Meta-Heuristic Algorithms by Solving TSP. In: Bindhu, V., Tavares, J.M.R.S., Boulogeorgos, AA.A., Vuppalapati, C. (eds) International Conference on Communication, Computing and Electronics Systems. Lecture Notes in Electrical Engineering, vol 733. Springer, Singapore. https://doi.org/10.1007/978-981-33-4909-4_32

[18] Abhijit Suresh, K.V. Harish, N. Radhika, Particle Swarm Optimization over Back Propagation Neural Network for Length of Stay Prediction, Procedia Computer Science, Volume 46, 2015, Pages 268-275, ISSN 1877-0509, https://doi.org/10.1016/j.procs.2015.02.020.

[19] Vijayarangan, Jayakumar & Raju, Ramasamy. (2014). A Simulated Annealing Algorithm for Machine Cell Formation Under Uncertain Production Requirements. Arabian Journal for Science and Engineering. 39. 7345-7354. 10.1007/s13369-014-1306-1.

[20] Tatavarthy, S. R., & Sampangi, G. (2015). Solving a reverse supply chain TSP by genetic algorithm. Applied mechanics and materials, 813, 1203-1207.