

Vehicle Routing Problem Implementation using Two Stage Multi-Algorithm route optimization framework

Sai Satya Agasteswara Vishnu T¹, K Deva Narayana Sai², Dr Bharati Bidikar³
Velcheru Nanda Kishore⁴, YSS Vyshnavi Rasagna⁵

^{1,2,4,5} UG Students, Dept. of CSE, Andhra University, Andhra Pradesh, India

³ Adjunct Professor, Dept. of Computer Science and Engineering, Andhra University, Andhra Pradesh

Abstract: In modern logistics and e-commerce systems, optimizing a multi-agent delivery operation remains as a major challenge due to the complexity of large-scale routing and assignment problems, Efficient route planning has become very important in these systems, especially with the rapid growth of e-commerce and courier services. As it is essential to minimize travel time and cost while covering a large number of delivery locations. This problem is related to optimization problems such as the Traveling Salesman Problem (TSP) and the Vehicle Routing Problem (VRP). This paper presents a two-stage multi-agent route optimization framework which is designed to improve both agents delivery allocation and route optimization. In the first stage agent delivery assignment is done, and in the second stage Optimal Agents specific routing is done. To enhance real-world applicability, the system incorporates a hybrid distance computation approach that leverages the Google Maps, Distance Matrix API for accurate distances, along with a Haversine-based fallback for offline scenarios. Experimental results indicates that the proposed framework reduces total travel distance and execution time while maintaining scalability and adaptability of real-world delivery applications.

Key Words: Vehicle Routing Problem, Travelling Salesman Problem, Route Optimization, Agents Delivery Allocation, Distance Matrix, Haversine, Multi-Agent Delivery Operation

1. INTRODUCTION

The Traveling Salesman Problem (TSP) remains a fundamental problem in combinatorial optimization and it is known for its high computational complexity. In modern logistics, this problem relates to the Vehicle Routing Problem (VRP), which involves multiple agents along with constraints such as workload balancing and cost minimization. As delivery networks continue to grow, managing both agent assignment and route planning becomes challenging.

A key limitation in many existing systems is that the delivery assignment and route optimization are handled separately. This separation often leads to inefficient routing and uneven distribution of work among agents. So we need an integrated approach that can effectively handle both problems together.

This paper shows two-stage multi-agent route optimization framework, as shown in Fig. 1. In the first stage, delivery locations are assigned using spatial and optimization-based techniques to ensure balanced workload distribution. In the second stage, routes are optimized for each agent using multiple algorithms, and the best solution is selected based on performance metrics. By combining both the stages, the framework improves overall efficiency and provides a scalable solution for the real-world delivery systems.

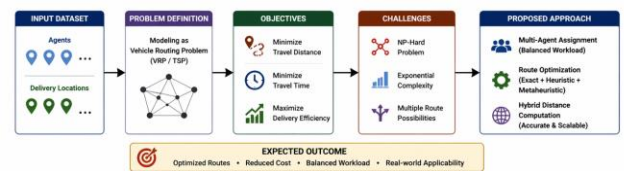


Fig 1 Multi-Agent Route Optimization Overview

2. METHODOLOGY

The proposed system uses a structured multi-stage methodology to solve the multi-agent route optimization problem. This approach is divided into three major components: data management and distance matrix construction, multi-agent assignment, and route optimization. We have used a real-world dataset to evaluate systems performance.

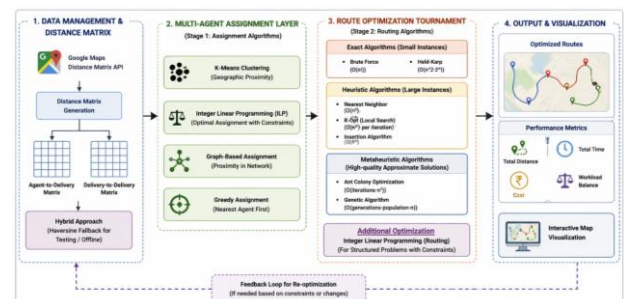


Fig 2 Methodology Framework of Algorithms

2.1 Data Management and Distance Matrix

A dedicated module is developed to generate precise distance matrices for both delivery assignment and route optimization. The Real-world traveling distances and durations are obtained using the Google Maps Distance Matrix API, ensuring realistic routing based on road networks. There are two types of matrices which are generated: (i) an agent-to-delivery matrix capturing distances between each delivery agent and all delivery locations, and (ii) a delivery-to-delivery matrix used for solving routing problems such as TSP. To ensure cost efficiency and flexibility, a hybrid approach is adopted, where approximate methods like Haversine-based calculations are used as a fallback for testing and offline scenarios.

2.2 Multi-Agent Assignment Layer

Now we evaluate four optimization approaches for the delivery assignment problem:

Stage 1: Delivery Assignment Algorithms

1. Incremental Cost Greedy Algorithm:

In this approach we assign deliveries step by step by choosing an option that adds the least extra travel cost at each stage. It focuses on making the best immediate decision, gradually building an efficient and reasonably balanced assignment.

2. Pure Spatial Partitioning Algorithm:

In this method, delivery locations are grouped based on how close they are to each other. Each group is then assigned to an agent, which helps in reducing travel distance by keeping deliveries within a geographic area

3. Min-Cost Max-Flow (MCMF) Algorithm:

In this algorithm we treat the assignment problem as a network, where agents and delivery locations are connected with costs. This algorithm finds the best possible assignment by minimizing the total cost while ensuring all constraints are satisfied, which leads to an efficient overall solution.

4. Hybrid Integer Linear Programming (Hybrid ILP) Algorithm:

This approach is used as an optimization model to assign deliveries by considering constraints like workload balance. It combines ILP with simpler techniques to reduce computation time, which results in a fair and efficient distribution of deliveries.

2.3 Route Optimization Tournament

Stage 2: Routing Algorithms for Deliveries

After the delivery assignment phase, each agent's route is optimized using a tournament-based approach, where

multiple routing algorithms are executed, and the best-performing routing algorithm is selected based on metrics such as total distance, travel time, and computational efficiency.

1. Nearest Neighbor Algorithm

The Nearest Neighbor algorithm constructs a route by iteratively selecting the closest unvisited delivery location from the current node. It is simple, fast, and suitable for real-time applications, as it may produce suboptimal solutions due to its greedy nature, and it does not consider the overall route and may lead to longer paths in the final result. The Time Complexity for this algorithm is $O(n^2)$

2. Strict Greedy Algorithm

The Strict Greedy approach selects the next location based on the lowest immediate cost, without considering how the decision will affect the overall route. Unlike the Nearest Neighbor method, it may take additional factors such as travel time or weighted distance, making it slightly more flexible. Since it focuses only on the current step and short-term gains, it misses better overall solutions and may not produce the most efficient route. The Time Complexity for this algorithm is $O(n^2)$

3. Ant Colony Optimization (ACO)

Ant Colony Optimization is inspired by the way that ants find the shortest path for their food. In this approach, multiple artificial agents explore different routes and share information by promising paths, which helps to guide future solutions. Over the time, this process allows the system to gradually improve and find better routes. It is effective for solving complex routing problems, but it requires more computation time compared to other algorithms. The Time Complexity for this algorithm is $O(\text{iterations} \times n^2)$

4. Genetic Algorithm (GA)

The Genetic Algorithm is based on the concept of natural evolution and works by maintaining possible routes. It improves these routes over the time using operations such as selection, crossover, and mutation. Using this process, the algorithm is able to explore a wide range of solutions and gradually finds better routes. It is effective in producing high-quality results, The Time Complexity for this algorithm is $O(\text{generations} \times \text{population} \times n)$

5. Insertion Heuristic

The Insertion algorithm builds the route step by step by inserting unvisited locations into positions that causes an increase of total travel distance. By gradually constructing the route, it maintains a good balance between efficiency and solution quality, performing better than basic greedy approaches. The Time Complexity for this algorithm is $O(n^2)$

6. Simulated Annealing (SA)

Simulated Annealing (SA) is a probabilistic optimization technique inspired from the annealing process in metallurgy.

It allows the system to occasionally accept worse solutions in the early stages to avoid getting stuck in the local solutions. The algorithm gradually improves the route and converges it towards a near-optimal solution. The Time Complexity for this algorithm is $O(\text{iterations} \times n)$

7. Deep Optimiser (Hybrid Local Search)

The Deep Optimizer is a hybrid approach that improves existing routes by repeatedly making small changes, such as swapping edges or refining route segments. It combines different local search techniques with iterative improvement to enhance solutions that are generated by greedy or heuristic methods, leading to better and more optimized routes. The Time Complexity for this algorithm is Typically $O(k \times n^2)$, where k is the number of refinement iterations

8. Held-Karp Dynamic Programming

Held-Karp is an exact dynamic programming method used to solve smaller instances of the TSP. It reduces repeated calculations through memorization, allowing to give the best possible solution. However, due to its high computational complexity, it becomes very slow and impractical for larger datasets. The Time Complexity for this algorithm is $O(n^2 \cdot 2^n)$

2.4 Dataset Description

The dataset used in this paper represents a structured collection of real-world dataset for delivery operations in the urban environment of Visakhapatnam, where multiple delivery agents are responsible for delivering items to various locations across the city.

The dataset consists of key attributes such as Agent_ID, Start_Latitude, Start_Longitude, Delivery_ID, Delivery_Latitude, and Delivery_Longitude. It is divided into two main components: The Agents dataset, which contains agent details and starting coordinates, and the Deliveries dataset, which contains delivery location information.

The dataset includes 1000 delivery locations and 50 agents, with each agent handling approximately 20–25 deliveries. The geographical coordinates ranges between latitudes 17.65 to 17.85 and longitudes 83.15 to 83.35. This ensures representation of both residential and commercial areas, providing a realistic testing environment for the algorithms.

3. RESULTS

3.1 Analysis of assignment algorithms

In this section, the analysis of results are done using assignment algorithms (Pure Spatial Partitioning, Hybrid ILP, Incremental Cost Greedy, Min-Cost Max-Flow) on given dataset are shown below

```

Selected Algorithm: Pure Spatial Partitioning

DELIVERY DISTRIBUTION:

AGENT A001 : 17 DELIVERIES
Sample: ['D1', 'D2', 'D5', 'D6', 'D22'] ...
AGENT A002 : 22 DELIVERIES
Sample: ['D4', 'D9', 'D11', 'D12', 'D28'] ...
AGENT A003 : 16 DELIVERIES
Sample: ['D7', 'D10', 'D14', 'D23', 'D27'] ...
AGENT A004 : 37 DELIVERIES
Sample: ['D3', 'D13', 'D17', 'D18', 'D19'] ...
AGENT A005 : 8 DELIVERIES
Sample: ['D8', 'D15', 'D16', 'D40', 'D62'] ...

TOTAL ASSIGNED DELIVERIES: 100

LOAD ANALYSIS:
Max Load : 37
Min Load : 8
Avg Load : 20.00

PERFORMANCE METRICS:
Total Distance : 759216.00
Load Std Dev : 10.75
    
```

Fig-3: Pure Spatial Partitioning for Delivery Assignment

```

=====
FINAL SYSTEM-WIDE OPTIMIZATION SUMMARY
=====
1. Delivery Assignment Algorithm : Pure Spatial Partitioning
2. Total Best Optimized Distance (Sum of all agents) : 153991.32m
3. Total Best Optimized Time (Sum of all agents) : 18871.03s
Initial Haversine Total Dist : 759216.00m
Improvement vs Haversine (if any) : 79.7%
=====
    
```

Fig-4: Final System-Wide Optimization Summary for Pure Spatial Partitioning

```

Selected Algorithm: Incremental Cost Greedy

DELIVERY DISTRIBUTION:

AGENT A001 : 20 DELIVERIES
Sample: ['D67', 'D30', 'D57', 'D31', 'D96'] ...
AGENT A002 : 20 DELIVERIES
Sample: ['D26', 'D18', 'D80', 'D1', 'D100'] ...
AGENT A003 : 20 DELIVERIES
Sample: ['D38', 'D49', 'D19', 'D56', 'D75'] ...
AGENT A004 : 20 DELIVERIES
Sample: ['D45', 'D50', 'D59', 'D99', 'D87'] ...
AGENT A005 : 20 DELIVERIES
Sample: ['D6', 'D65', 'D21', 'D86', 'D3'] ...

TOTAL ASSIGNED DELIVERIES: 100

LOAD ANALYSIS:
Max Load : 20
Min Load : 20
Avg Load : 20.00

PERFORMANCE METRICS:
Total Distance : 1011266.00
Load Std Dev : 0.00
    
```

Fig-5: Incremental Cost Greedy for Delivery Assignment

```

=====
FINAL SYSTEM-WIDE OPTIMIZATION SUMMARY
=====
1. Delivery Assignment Algorithm : Incremental Cost Greedy
2. Total Best Optimized Distance (Sum of all agents) : 537138.00m
3. Total Best Optimized Time (Sum of all agents) : 76820.00s
Initial Haversine Total Dist : 1011266.00m
Improvement vs Haversine (if any) : 46.9%
=====
    
```

Fig-6: Final System-Wide Optimization Summary for Incremental Cost Greedy

```

Selected Algorithm: Min-Cost Max-Flow Graph

DELIVERY DISTRIBUTION:

AGENT A001 : 20 DELIVERIES
Sample: ['D1', 'D4', 'D9', 'D11', 'D12'] ...
AGENT A002 : 20 DELIVERIES
Sample: ['D13', 'D17', 'D25', 'D29', 'D33'] ...
AGENT A003 : 20 DELIVERIES
Sample: ['D7', 'D10', 'D14', 'D23', 'D27'] ...
AGENT A004 : 20 DELIVERIES
Sample: ['D2', 'D3', 'D19', 'D20', 'D34'] ...
AGENT A005 : 20 DELIVERIES
Sample: ['D5', 'D6', 'D8', 'D15', 'D16'] ...

TOTAL ASSIGNED DELIVERIES: 100

LOAD ANALYSIS:
Max Load : 20
Min Load : 20
Avg Load : 20.00

PERFORMANCE METRICS:
Total Distance : 755063.00
Load Std Dev : 0.00
    
```

Fig-9: Min-Cost Max-Flow Graph for Delivery Assignment

```

Selected Algorithm: Hybrid ILP Assignment

DELIVERY DISTRIBUTION:

AGENT A001 : 20 DELIVERIES
Sample: ['D1', 'D4', 'D9', 'D18', 'D21'] ...
AGENT A002 : 20 DELIVERIES
Sample: ['D13', 'D17', 'D25', 'D29', 'D36'] ...
AGENT A003 : 20 DELIVERIES
Sample: ['D7', 'D10', 'D14', 'D23', 'D27'] ...
AGENT A004 : 20 DELIVERIES
Sample: ['D2', 'D3', 'D11', 'D19', 'D34'] ...
AGENT A005 : 20 DELIVERIES
Sample: ['D5', 'D6', 'D8', 'D12', 'D15'] ...

TOTAL ASSIGNED DELIVERIES: 100

LOAD ANALYSIS:
Max Load : 20
Min Load : 20
Avg Load : 20.00

PERFORMANCE METRICS:
Total Distance : 755043.00
Load Std Dev : 0.00
    
```

Fig-7: Hybrid ILP Assignment for Delivery Assignment

```

=====
FINAL SYSTEM-WIDE OPTIMIZATION SUMMARY
=====
1. Delivery Assignment Algorithm : Min-Cost Max-Flow Graph
2. Total Best Optimized Distance (Sum of all agents) : 296556.22m
3. Total Best Optimized Time (Sum of all agents) : 41508.94s
Initial Haversine Total Dist : 755063.00m
Improvement vs Haversine (if any) : 60.7%
=====
    
```

Fig-10: Final System-Wide Optimization Summary for Min-Cost Max-Flow Graph

```

=====
FINAL SYSTEM-WIDE OPTIMIZATION SUMMARY
=====
1. Delivery Assignment Algorithm : Hybrid ILP Assignment
2. Total Best Optimized Distance (Sum of all agents) : 357198.22m
3. Total Best Optimized Time (Sum of all agents) : 49031.94s
Initial Haversine Total Dist : 755043.00m
Improvement vs Haversine (if any) : 52.7%
=====
    
```

Fig-8: Final System-Wide Optimization Summary for Hybrid ILP Assignment

By analyzing the performance metrics for the 4 algorithms: Pure Spatial Partitioning, Hybrid ILP, Incremental Cost Greedy, Min-Cost Max-Flow on the given dataset show that Pure Spatial Partitioning gives the most optimal results. As the total best optimized distance is least for pure spatial partitioning with value 153991.32m. By considering total best optimized distance, the algorithms can be sorted in the ascending order of optimization as 1. Pure Spatial Partitioning (153991.32m), 2. Min-Cost Max-Flow graph (296556.22m), 3. Hybrid ILP (357198.22m), 4. Incremental Greedy (537138.00m).

Similarly, the ascending order of optimization based on time are 1. Pure Spatial Partitioning, 2. Min-Cost Max-Flow Graph, 3. Hybrid ILP Assignment, 4. Incremental Greedy.

3.2 Performance Evaluation of Route Optimization Algorithms

Number of Locations	Held-Karp DP	Nearest-Neighbour	Strict Greedy	Ant Colony	Genetic Algorithm	Insertion	Simulated Annealing	Deep Optimiser
9	0.0047693	0.0020279	0.005822	20.1908305	1.5337984	0.0021991	0.0186587	0.0014913
17	2.8661478	0.0348127	0.0821703	36.8364477	1.528512	0.0261001	0.0607123	0.0121697
18	10.4983743	0.0800671	0.1213959	41.5406628	2.5920558	0.0574798	0.1370843	0.0299044
23	7.20E+03	0.2136661	0.3295354	46.6085368	4.2661743	0.1513955	0.2848407	0.1414372
38	1.10E+09	0.9384906	1.6838078	23.3601516	1.7707828	0.697567	2.0109379	0.5048895

Fig-11: Algorithms Execution Time

The figure 11 describes us about the comparison of Route Optimization of 8 different algorithms. As we observe the table we can see that the values of Deep Optimizer gives us more efficient output.

Algorithm Performance: Execution Time Scaling

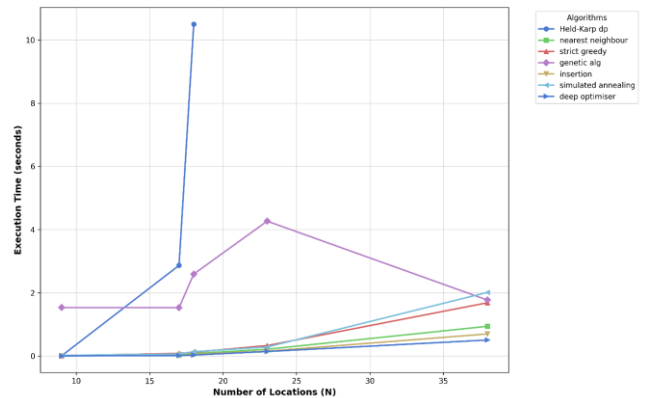


Fig-14: Algorithm Performance: Execution Time Scaling except Ant Colony

Figures 12, 13, and 14 shows us how the execution time of different algorithms changes as the number of delivery locations are increased. As, the problem size grows, the execution time increases for all algorithms, but the increase varies across the methods. The Insertion Algorithm shows the lowest execution time for most of the problems, indicating that it is both efficient and scalable. This concludes that it can handle larger routing problems without a significant increase in computation time.

The Ant Colony Optimization takes the more time, when the number of locations are increased. This is because it explores multiple possible solutions before converging, which requires more computation time. Similarly, the Held-Karp algorithm shows a sharp rise in execution time for larger datasets due to its exponential complexity. In normal heuristic methods like Nearest Neighbor, Strict Greedy, and Insertion maintain lower execution time, even though their performance may vary depending on the size and nature of the problem.

Algorithm Performance: Execution Time Scaling

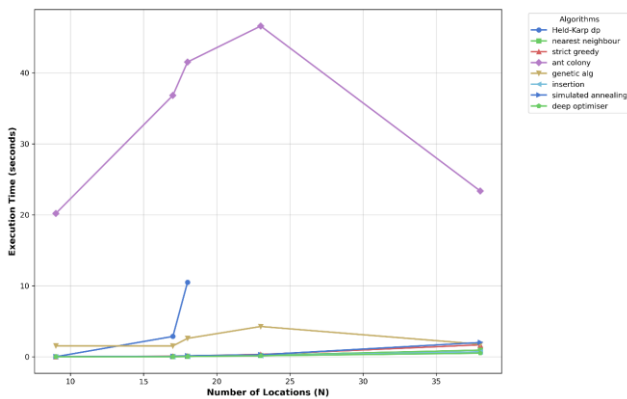


Fig 12 Execution Time Scaling of all 8 algorithms

Algorithm Performance: Execution Time Scaling

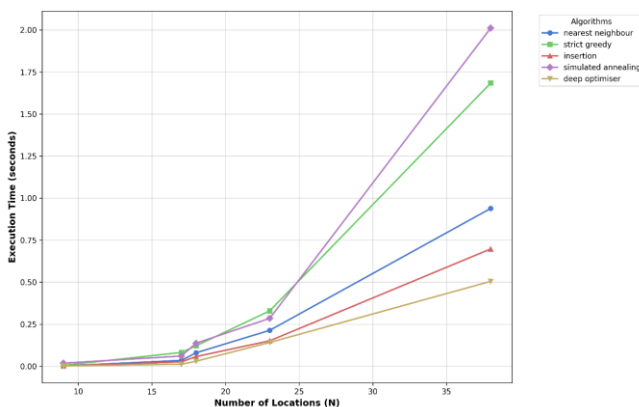


Fig-13: Execution Time Scaling of TSP Algorithms except Ant Colony, Held Karp DP, Genetic Algorithm

Number of Locations	held karp dp	nearest neighbor	strict greedy	ant colony	genetic	insertion	simulated annealing	deep optimizer
9	29956	29956	29956	29956	29956	29956	29956	29956
17	16316	16316	16316	16316	16316	16316	16316	16316
18	12577	12577	12577	12577	12577	12577	12577	12577
23	N/A	52270	52342	52270	52342	52270	52270	52270
38	N/A	43562	43982	45577	45912	42872	47661	43562

Fig-15: Algorithms Minimum Distance

The Figure 15 describes us about the comparison of Minimum Distance of 8 different algorithms. As we see that Insertion Algorithm gives us most efficient values.

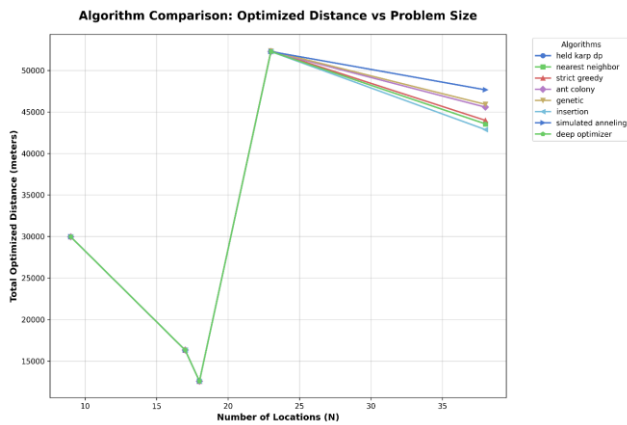


Fig-16: Optimized Distance vs Problem Size

Figure 16 shows how the optimized travel distance changes as the problem size increases. It compares how efficiently each algorithm reduces travel distance when the number of locations becomes larger. As the number of delivery locations increases, the total travel distance also increases for all methods due to the higher complexity of routing. Among all the algorithms tested, the Insertion method consistently produces the shortest routes, which indicates better performance in minimizing total travel distance. The Deep Optimizer achieves results that are very close to the best as it maintains much lower execution time, showing a good balance between speed and accuracy. Simulated Annealing results in relatively higher travel distances, indicates lower efficiency in route optimization for the given scenarios.

Overall, the results shows that different algorithms perform well in different aspects. The Insertion algorithm provides the best route quality, while the Deep Optimizer offers the best computational efficiency. Therefore, the Deep Optimizer is a suitable for choosing large-scale delivery problems where both speed and solution quality are important.

3.3 Performance Comparison of TSP Algorithms for Agents

Nearest Neighbour (min_dist)	: dist=43562.40m	time=4765.60s	speed=9.1410m/s
Nearest Neighbour (min_time)	: dist=42886.50m	time=4709.90s	speed=9.1056m/s
Nearest Neighbour (max_speed)	: dist=45003.30m	time=4829.60s	speed=9.3182m/s
Execution time: 1.2356341000413522 seconds			
Strict Greedy (min_dist)	: dist=43981.50m	time=4752.70s	speed=9.2540m/s
Strict Greedy (min_time)	: dist=43981.50m	time=4752.70s	speed=9.2540m/s
Strict Greedy (max_speed)	: dist=43981.50m	time=4752.70s	speed=9.2540m/s
Execution time: 2.245699599967338 seconds			
Ant Colony (min_dist)	: dist=45577.20m	time=4838.80s	speed=9.4191m/s
Ant Colony (min_time)	: dist=45577.20m	time=4838.80s	speed=9.4191m/s
Ant Colony (max_speed)	: dist=43562.40m	time=4765.60s	speed=9.1410m/s
Execution time: 26.32771500002127 seconds			
Genetic Algorithm (min_dist)	: dist=45911.90m	time=4917.30s	speed=9.3368m/s
Genetic Algorithm (min_time)	: dist=45577.20m	time=4838.80s	speed=9.4191m/s
Genetic Algorithm (max_speed)	: dist=43562.40m	time=4765.60s	speed=9.1410m/s
Execution time: 1.7820131999906152 seconds			
Insertion (min_dist)	: dist=42872.00m	time=4691.60s	speed=9.1380m/s
Insertion (min_time)	: dist=42872.00m	time=4691.60s	speed=9.1380m/s
Insertion (max_speed)	: dist=42872.00m	time=4691.60s	speed=9.1380m/s
Execution time: 0.6740429999772459 seconds			
Simulated Annealing (min_dist)	: dist=44437.60m	time=4799.50s	speed=9.2588m/s
Simulated Annealing (min_time)	: dist=44437.60m	time=4799.50s	speed=9.2588m/s
Simulated Annealing (max_speed)	: dist=44437.60m	time=4799.50s	speed=9.2588m/s
Execution time: 1.6803668000502512 seconds			
Deep Optimizer (min_dist)	: dist=43562.40m	time=4765.60s	speed=9.1410m/s
Deep Optimizer (min_time)	: dist=43562.40m	time=4765.60s	speed=9.1410m/s
Deep Optimizer (max_speed)	: dist=43562.40m	time=4765.60s	speed=9.1410m/s
Execution time: 0.4575474999146536 seconds			
[BEST PERFORMING ALGORITHM FOR AGENT A004]			
> Best Performing Algorithm : Insertion			
> Sequence : ['D86', 'D100', 'D58', 'D99', 'D80', 'D26', 'D18', 'D45', 'D50', 'D21', 'D38', 'D30', 'D70', 'D41', 'D17', 'D98', 'D25', 'D88', 'D20', 'D60', 'D78', 'D51', 'D54', 'D29', 'D75', 'D85', 'D87', 'D97', 'D13', 'D19', 'D3', 'D76', 'D34', 'D48', 'D56', 'D68', 'D71']			

Fig-17: Comparison of 8 Algorithms

The Figure 17 shows a comparison of different Traveling Salesman Problem (TSP) algorithms for route optimization of Agent A004, which includes 38 delivery locations. The algorithms that are compared are Nearest Neighbor, Strict Greedy, Ant Colony Optimization, Genetic Algorithm, Insertion, Simulated Annealing, and Deep Optimizer. This evaluation is based on key factors such as total travel distance, travel time, average speed, and execution time. From the results, the Insertion algorithm performs best for Agent A004, as it gives the lowest travel distance (42872.00m) also maintaining a reasonable travel time (4691.60s) and execution speed (9.1380m/s). This shows that the insertion method is effective in generating efficient routes for medium-sized problems. Algorithms like Ant Colony Optimization and Genetic Algorithm also produces good results, but they require more computation time. On

the other hand, simpler methods such as Nearest Neighbor and Strict Greedy run faster but they result in longer travel distances, making them less efficient.

The figure also indicates that performance can vary across different agents. While the Insertion algorithm works best for agents like A001, A004, and A005, the Deep Optimizer performs better for agents A002 and A003. This tells us that the effectiveness of an algorithm depends on factors like the number of locations and how they are distributed.

4 Conclusion

The experimental evaluation of two-staged multi-algorithms route optimization framework on a real-world dataset shows that pure spatial partitioning achieves the best performance in the delivery assignment stage, with the lowest total distance (153,991.32 m) and the least total time of (18,871.03 s) for completing all deliveries. In the routing stage, the Insertion algorithm performs better than the other methods for most agents, producing shorter routes with reduced travel time, higher average speed, and lower execution time. Based on these observations, it can be concluded that combining pure spatial partitioning for delivery assignment with the Insertion algorithm for route optimization provides the most efficient and reliable results for multi-agent delivery systems.

REFERENCES

- [1] G. B. Dantzig and J. H. Ramser, "The Truck Dispatching Problem," *Management Science*, vol. 6, no. 1, pp. 80–91, 1959.
- [2] G. Laporte, "The Vehicle Routing Problem: An Overview of Exact and Approximate Algorithms," *European Journal of Operational Research*, vol. 59, no. 3, pp. 345–358, 1992.
- [3] S. Lin, "Computer Solutions of the Traveling Salesman Problem," *Bell System Technical Journal*, vol. 44, no. 10, pp. 2245–2269, 1965.
- [4] S. Lin and B. W. Kernighan, "An Effective Heuristic Algorithm for the Traveling-Salesman Problem," *Operations Research*, vol. 21, no. 2, pp. 498–516, 1973.
- [5] M. Held and R. M. Karp, "A Dynamic Programming Approach to Sequencing Problems," *Journal of the Society for Industrial and Applied Mathematics*, vol. 10, no. 1, pp. 196–210, 1962.
- [6] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.
- [7] M. Dorigo, V. Maniezzo, and A. Coloni, "Ant System: Optimization by a Colony of Cooperating Agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 26, no. 1, pp. 29–41, 1996.
- [8] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Cambridge, MA: MIT Press, 1992.
- [9] J. MacQueen, "Some Methods for Classification and Analysis of Multivariate Observations," in *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, pp. 281–297, 1967.
- [10] D. Arthur and S. Vassilvitskii, "k-means++: The Advantages of Careful Seeding," in *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2007.
- [11] R. W. Sinnott, "Virtues of the Haversine," *Sky and Telescope*, vol. 68, no. 2, p. 159, 1984.
- [12] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, 1968.
- [13] D. Luxen and C. Vetter, "Real-Time Routing with OpenStreetMap Data," in *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2011.
- [14] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [15] Google Developers, "Google Maps Platform Routes & Distance Matrix APIs Documentation," [Online]. Available: <https://developers.google.com/maps/documentation/routes/overview>