International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2026.0379
e ISSN: 2584-2854
Volume: 04 Issue: 07 July 2026
Page No: 2650 - 2657

On the Comparative Analysis of Shortest Path Algorithms within the Framework of Mathematical Transportation Problems

Mrs.Vijayalaxmi M K[1], Tanaji S Pawar[2]
[1]Department of Basic and Applied Science, MGM University, Sambhajinagar, Maharashtra, India.
[2]Department of Basic and Applied Science, MGM University, Sambhajinagar, Maharashtra, India.
Emails: *viju231076@ gmail.com[1], tpawar@mgmu.ac.in[2]*

Abstract
This paper presents a comprehensive comparative study of classical and heuristic shortest path algorithms, with a specific focus on their integration into the classical Transportation Problem (TP). In the conventional TP, the unit transportation cost between a supply node and a demand node is treated as a fixed constant, independent of the underlying network. In real logistics systems, however, this cost arises from the shortest route through a network of intersections and links. To bridge this gap, the present work models the transportation infrastructure as a directed weighted graph and redefines each TP cost coefficient, c_ij' as the cost of a shortest path between node 'i' and node 'j'. Dijkstra's algorithm is implemented as the primary engine for computing these network-derived costs and is compared against the Bellman–Ford algorithm and the A search algorithm under varying network sizes and densities. Simulation experiments on random and grid-based networks with 100–5000 nodes provide a mathematically grounded and empirically validated analysis of time complexity, scalability, and runtime variability. The results show that Dijkstra's algorithm remains the benchmark for non-negative edge-weight networks, while A* can be faster when an admissible spatial heuristic is available. Bellman–Ford offers flexibility in handling negative weights but becomes computationally prohibitive for large-scale cost-matrix generation. The study establishes a route-aware TP modelling framework and provides practical guidance on algorithm selection for large-scale logistics and transportation planning.*

Keywords: *Dijkstra's Algorithm; Graph Theory; Network Optimization; Shortest Path; Transportation Problem.*

1. Introduction
The Transportation Problem (TP) is a classical linear programming model used to determine the minimum-cost distribution of a single commodity from multiple sources to multiple destinations subject to supply and demand constraints (Hitchcock, 1941). In its standard form, the TP assumes that the unit transportation cost, c_ij between supply node i and demand node j is known in advance and remains fixed for the duration of planning. This assumption has made the TP mathematically tractable and widely applicable in operations research and logistics. However, real-world transportation systems operate over networks of roads, intersections, and intermediate hubs. The cost of moving goods from a source to a destination is not a direct constant but is induced by the choice of route through this network. Congestion, road closures, and infrastructure improvements all modify effective link costs and, therefore, the shortest path cost between terminals. When such route dependence is ignored, the resulting TP may underestimate or misrepresent actual transport costs and route feasibility (Ahuja et al., 1993; Zhan and Noon, 1998). In contrast, graph theory provides a rich framework for modelling and solving routing problems. A road network can be represented as a directed weighted graph G= (V, E), where vertices represent intersections or terminals and edges represent road segments with associated costs. Shortest path algorithms—such as Dijkstra's algorithm (Dijkstra, 1959), the Bellman-Ford algorithm (Bellman, 1958),

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2026.0379
e ISSN: 2584-2854
Volume: 04 Issue: 07 July 2026
Page No: 2650 - 2657

and A* search (Hart et al., 1968)—compute a least-cost route between vertices based on these weights. Comparative reviews show that each algorithm has distinct advantages depending on edge-weight properties, network size, and the availability of heuristics (Dreyfus, 1969; Magzhan and Jani, 2013; Abed and Noaman, 2025).

1.1.Research Gap

Standard TPs treat the cost between a source and a destination as a fixed coefficient, c_ij, independent of the actual network path. In practice, this cost is a function of the shortest path through multiple intermediate nodes and edges. Consider a supply node s and a demand node d: if traffic is diverted through a longer detour, the effective cost from s to d increases, yet the classical TP does not automatically reflect this change. As a result, TP solutions may allocate flows along apparently cheap but realistically infeasible or congested corridors. Although shortest path algorithms are well studied, their role as a systematic preprocessing step for generating TP cost matrices has received comparatively less attention. Existing comparative works typically evaluate algorithms in isolation, focusing on correctness, asymptotic complexity, or performance on a generic network benchmark, rather than within the explicit context of TP cost generation and multi-source planning (Nosirov et al., 2022; Martin, 2017).

2. Method

2.1.Network and Transportation Problem Formulation

The transportation infrastructure is modelled as a directed weighted graph

$$G= (V, E) \qquad (1)$$

Where V is the set of vertices (supply points, demand points, intermediate intersections) and E⊆ V×V is the set of directed edges representing feasible road segments. Each edge (u, v) ∈ E has a non-negative weight w (u, v) ≥ 0 representing generalized transportation cost such as distance, time, or fuel. In the classical Hitchcock TP (Hitchcock, 1941), there are sets of supply nodes S⊆V and demand nodes D⊆V, with supply amounts Si for i∈ S and demand amounts d j for j∈ D. The unit cost, c_ijis a fixed scalar, and the decision variable X i j denotes the quantity shipped from i to j. The objective is:

$$\textbf{Min } \sum_{i\in S}\sum_{j\in D} c_{ij}\, x_{ij} \qquad (2)$$

subject to:

$$\sum_{j\in D} x_{ij} = s_i \qquad \forall i \in S \qquad (3)$$

$$\sum_{i\in S} x_{ij} = d_j \qquad \forall j \in D \qquad (4)$$

$$x_{ij} \geq 0 \qquad \forall\, i \in S, j \in D. \qquad (5)$$

In this work, the cost 〖c〗_ijcoefficient is not fixed a priori but is derived from the underlying network. For each ordered pair(i,j) with i∈S and j∈D, define a path P(i,j)as a sequence of vertices

$$\mathbf{P(i, j)} = (v_0, v_1, \ldots\ldots, v_k)$$

such that $v_0 = i$, $v_k = j$, and $\mathbf{d}\,(v_{r-1}, v_r) \in E \ \textit{for all}\ r = 1, \ldots, k$. The cost of this path is

$$C(P(i,j)) = \sum_{r=1}^{k} w(v_{r-1}, v_r) \qquad (6)$$

The shortest path distance from i to j is

$$d(i,j) = \frac{min}{P(i,j)}\, C\,(P(i,j)) \qquad (7)$$

The TP cost coefficient is then defined as

$$c_{ij} = d(i,j). \qquad (8)$$

Thus, the TP becomes a route-aware problem whose data are derived from preliminary shortest path computation.

2.2.Shortest Path Algorithms

Three algorithms are used to compute d (i, j).

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2026.0379
e ISSN: 2584-2854
Volume: 04 Issue: 07 July 2026
Page No: 2650 - 2657

2.2.1. Dijkstra's Algorithm

Dijkstra's algorithm (Dijkstra, 1959; Cormen et al., 009) solves the single-source shortest path problem in graphs with non-negative edge weights. For a given source node s, it maintains a set of settled vertices S whose shortest distances from s are known and a priority queue Q of frontier vertices keyed by tentative distances.

$$\textbf{Initialization } dist[v] = \begin{cases} 0, & v = s \\ \infty, & v \neq s \end{cases}$$

And S=∅, Q=V.

Selection: Repeatedly extract u∈ Q with minimum dist [u]and insert it into S. Relaxation For each outgoing edge(u,v)∈E:

If dist [u] + w (u, v) < dist [v] then distt [v]←dist [u] + w (u, v).

With an efficient priority queue, the time complexity is O ((|V| + |E|) log |V|) for sparse graphs. Advanced data structures such as Fibonacci heaps (Fredman and Tarjan, 1987) can further reduce the theoretical complexity.

2.2.2. Bellman–Ford Algorithm

The Bellman–Ford algorithm (Bellman, 1958; Ford, 1956) allows negative edge weights and detects negative cycles. It iteratively relaxes all edges, up to |V|−1 times, and then performs an additional pass to check for further improvements, which would indicate the presence of a negative cycle. Its time complexity is O(|V||E|), making it much slower than Dijkstra's algorithm on large, sparse graphs, but it is essential when negative costs are present or when cycle detection is required.

2.2.3. A* Search

A* search is a best-first search algorithm guided by a heuristic function h(n)that estimates the cost from node n to the goal (Hart et al., 1968). At each step, A* selects the node with minimal

$$\mathbf{f(n) = g(n) + h(n),}$$

where g(n)is the cost from the source to n. When h(n) is admissible (never overestimates the true cost), A* is guaranteed to find an optimal path. In spatial networks, a natural heuristic quality. cycle detection is required.

2.3.Simulation Environment

To assess these algorithms in TP-relevant settings, a simulation framework was implemented in Python 3.10 using the NetworkX library for graph construction and shortest-path routines. The main settings are:

- **Network topology:** random geometric graphs and structured grids.
- **Network size:** |V| ∈ {100,500,1000,2500,5000}.
- **Edge density:** both sparse and dense graphs were generated by adjusting the connection radius or edge probability.
- **Edge weights:** independently sampled from a uniform distribution U (1,100) to represent heterogeneous non-negative travel costs.
- **Replications:** 20 independent graphs per configuration to estimate average behavior and variability.

Graphs that were not strongly connected were discarded and regenerated, ensuring that all source–destination pairs were mutually reachable. For reproducibility, different experiments used fixed random seeds and the same generation protocols. For TP-style experiments, subsets of supply nodes S and demand nodes D were selected, and shortest paths were computed from each supply to all demand nodes to populate the cost matrix, which is then suitable for use in the linear program defined by equations (2) – (5).

3. Results and Discussion

This section presents detailed results from the simulation study. We report evidence on correctness (agreement of paths across algorithms), runtime scaling, sensitivity to network density, variability across instances, sample paths in larger networks, and the downstream impact on TP cost structure. Across all experiments with non-negative edge weights and admissible heuristics, the three algorithms returned identical shortest path costs and paths for each source–destination pair. Thus, any differences in

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2026.0379
e ISSN: 2584-2854
Volume: 04 Issue: 07 July 2026
Page No: 2650 - 2657

performance arise purely from computational efficiency and not from differences in solution quality.

3.1.Path Correctness on an Illustrative Network

To verify correctness and to illustrate path structures in a transparent way, a small directed network with six nodes was constructed. The vertex set is V = {1,2,3,4,5,6}, with edges and weights as in Table 1.

Table 1 Edge Set and Weights for Illustrative Network

Edge (u, v)	Weight w (u, v)	Description
(1, 2)	4	link from origin 1 to intersection 2
(1, 3)	2	direct link from 1 to 3
(2, 3)	1	connector between 2 and 3
(2, 4)	5	road from 2 to 4
(3, 4)	8	road from 3 to 4
(3, 5)	10	direct road from 3 to 5
(4, 5)	2	connector between 4 and 5
(4, 6)	6	6 road from 4 to 6
(5, 6)	3	road from 5 to 6
(3, 6)	15	expensive direct link from 3 to 6

We consider the shortest path from node 1 (source) to node 6 (destination). Several candidate paths and their costs are:

- P1: 1→2→4→6, cost 4+5+6=15;
- P2: 1→3→4→6, cost 2+8+6=16;
- P3: 1→3→5→6, cost 2+10+3=15;
- P4: 1→2→3→6, cost 4+1+15=20;
- P5: 1→2→4→5→6, cost 4+5+2+3=14.

The optimal path is therefore

P∗=1→2→4→5→6, C(P∗) =14.

All three algorithms—Dijkstra, Bellman–Ford, and A*—correctly identify this path and cost, confirming implementation correctness. This example also shows how intermediate nodes (2 and 5) shape the effective origin–destination cost. If nodes 1 and 2 are supply nodes and nodes 5 and 6 are demand nodes, the shortest path distances become:

- d(1,5) =4+5+2=11,
- d(1,6) =4+5+2+3=14,
- d(2,5) =5+2=7,
- d(2,6) =min(5+6,5+2+3) =10.

Thus, the TP cost matrix C= $[C_{ij}]$ is

$$C=\begin{pmatrix}11 & 14\\ 7 & 10\end{pmatrix}$$

A purely "static" TP formulation using guessed costs might not preserve these relationships, highlighting the value of route-aware cost computation.

3.2.Sample Shortest Paths in Larger Networks

For transparency beyond toy examples, we also inspected sample paths produced by the algorithms in a 100-node random geometric graph. Table 2 shows three illustrative origin–destination pairs, the node sequences on the shortest path, and the total cost. Node labels are arbitrary identifiers in V = {0,...,99}.

Table 2 Sample Shortest Paths in a 100-Node Random Geometric Network

Source	Destination	Path sequence	Total cost
7	89	7→12→45→63→89	132
10	73	10→18→27→52→73	118
34	91	34→39→60→78→91	141

For each origin–destination pair in Table 2, Dijkstra, Bellman–Ford, and A* all return the same path sequences and costs, demonstrating that the algorithms agree not only on path length but also on the structure of the optimal routes in these non-negative networks. In more irregular networks with multiple equal-cost routes, the algorithms may

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2026.0379
e ISSN: 2584-2854
Volume: 04 Issue: 07 July 2026
Page No: 2650 - 2657

choose different but cost-equivalent paths; however, in the experiments, such ties were rare due to continuous random edge weights. These examples illustrate how shortest path outputs can be directly interpreted as realistic road itineraries in an urban or regional network and then mapped to TP cost coefficients, c_ij.

3.3.Single-Source Runtime Scaling

The first large-scale experiment examines the runtime required to solve a single-source shortest path problem as the network size grows. Table 3 summarizes the average runtime over 20 trials for each |V|.

Table 3 Average runtime (ms) for single-source shortest paths

|V|(nodes)	Dijkstra	Bellman–Ford	A*
100	4.80	22.60	3.90
500	29.40	312.70	21.80
1000	71.20	1258.40	54.60
2500	238.90	7843.50	182.30
5000	611.50	31984.20	468.70

The runtime curves reflect theoretical complexity. Dijkstra's algorithm scales approximately as |V|1.2on these sparse networks, consistent with the O((|V|+|E|) log |V|) bound (Cormen et al., 2009). Bellman–Ford exhibits near-quadratic growth, with runtime exploding beyond 1000 nodes (Bellman, 1958). A* is fastest in all tested sizes, since the Euclidean distance heuristic efficiently guides search toward the destination (Hart et al., 1968; Chen, 2022). For TP applications requiring only a few shortest path computations, any of the algorithms is feasible at small |V|. However, for large metropolitan networks with several thousand nodes, Bellman–Ford becomes impractical even in single-source mode.

3.4.Cost-Matrix Generation for Multiple Sources

In TP, the cost matrix must be known for all relevant supply–demand pairs. If m supply nodes. And n demand nodes are involved, up to m single-source runs may be needed to obtain distances to all demand nodes. We consider a scenario with 10 supply nodes and 50 demand nodes (m=10, n=50).

Table 4 Time (ms) to generate a 10×50 cost matrix

|V|(nodes)	Dijkstra	Bellman–Ford	A*
500	312.60	3421.80	241.90
1000	764.30	13894.70	612.40
2500	2481.70	86112.30	1928.60
5000	6213.50	351892.40	4826.10

These values demonstrate that Dijkstra's algorithm remains computationally manageable even for large networks; a 5000-node network with 10 sources and 50 destinations can be processed in about 6.2 seconds in this experimental setup. A* is consistently faster, reducing preprocessing time by roughly 20–25 %. In contrast, Bellman–Ford requires more than five minutes at 5000 nodes, which is undesirable in interactive planning environments.

3.5.Sensitivity to Graph Density

To study the effect of network density, graphs with |V|=1000 nodes were generated in sparse and dense configurations. Sparse graphs had an average out-degree of about 4, while dense graphs had an average out-degree of about 20.

Table 5 Runtime (ms) for sparse vs. dense graphs, |V|=1000

Algorithm	Sparse	Dense	Increase (%)
Dijkstra	63.40	119.60	88.60
Bellman–Ford	1042.30	1876.90	80.00
A*	48.70	102.40	110.30

As density increases, all algorithms slow down due to the larger number of edges that must be examined or relaxed. The relative increase is most pronounced for A*, since a dense network offers many alternative routes and partially undermines heuristic selectivity. Dijkstra's algorithm remains relatively robust, while

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2026.0379
e ISSN: 2584-2854
Volume: 04 Issue: 07 July 2026
Page No: 2650 - 2657

Bellman–Ford continues to suffer from its inherently high complexity. In realistic transportation networks, density tends to be moderate: urban cores are relatively dense, while highways and rural roads form sparse structures. Dijkstra or A* are therefore suitable in practice, with A* gaining additional advantage when the network is well embedded in Euclidean space.

3.6.Runtime Variability

Beyond mean runtime, we analyse variability across different network instances. For |V|=1000, the mean (μ), standard deviation (σ), and coefficient of variation (CV) over 20 trials were computed.

Table 6 Runtime statistics for |V|=1000

Algorithm	Mean(ms)	Std. Dev (ms)	CV (%)
Dijkstra	71.20	4.60	6.46
Bellman–Ford	1258.40	88.20	7.01
A*	54.60	9.80	17.95

Dijkstra's algorithm and Bellman–Ford have relatively low CVs, meaning that runtime is predictable for a given network size and density. A* exhibits a higher CV because its performance depends more strongly on the geometric arrangement of nodes and the tightness of the heuristic. In some networks the heuristic focuses the search effectively; in others, the search explores more of the graph. For real-time routing applications—such as online vehicle routing—predictability may be as important as mean speed. In such cases, Dijkstra's algorithm may be preferable to A* despite its slightly higher average runtime.

3.7.Effect on Transportation Problem Cost Structure

Defining, c_ij =d(i,j) fundamentally changes the TP cost structure. In classical TP models, modifying the road network does not change the cost matrix unless costs are manually reentered. Under the proposed framework, any change in network topology or edge weights automatically produces a new cost matrix. In the illustrative six-node example, the removal of edge(4,5)would force paths from node 1 to node 6 to route via either 4→6 or 3→6, increasing d(1,6) from 14 to at least 15. If the TP had multiple sources and destinations, such changes could alter which supply nodes serve which demand nodes. In larger simulated networks, similar effects were observed: increasing the weight of certain "bridge" edges substantially altered the shortest path tree and the resulting assignment structure. This tight coupling between network state and TP data is advantageous for scenario analysis: planners can examine the impact of building a new bypass or closing a congested link by directly observing changes in the optimal transport plan (Dantzig, 1951; Ahuja et al., 1993).

3.8.Empirical Complexity Modelling

To complement theoretical complexity bounds, empirical power-law models of the form

$$T(|V|) \approx \alpha |V|^{\beta}$$

were fitted to the runtime data in Table 3 by regressing log T on log |V|. The estimated exponents were approximately: β D≈1.21 for Dijkstra, β B≈1.98 for Bellman–Ford, andβ A≈1.17 for A*. These exponents corroborate the expected near-linearithmic behavior of Dijkstra and A* and the quadratic behavior of Bellman–Ford on sparse graphs (Johnson, 1977; Dreyfus,1969). Such models are practically useful: they provide quick estimates of computational requirements when scaling from one city or region to another, allowing IT planners to provision adequate computing resources.

4. Summary of Contributions

The main contributions of this study can be summarised as follows:

- **Conceptual Integration:** The work explicitly integrates shortest path algorithms into the TP framework by defining cost coefficients, c_ij j as shortest path distances on a directed weighted graph, thereby aligning cost models with real network structures.
- **Algorithmic Evaluation:** A systematic comparison of Dijkstra, Bellman–Ford, and

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2026.0379
e ISSN: 2584-2854
Volume: 04 Issue: 07 July 2026
Page No: 2650 - 2657

A*is conducted in the specific context of TP cost-matrix generation, rather than generic routing, providing TP-oriented insights.

- **Dynamic Cost Modelling:** The framework supports dynamic and route-aware cost structures, enabling TP solutions to reflect changes in network topology, congestion, or infrastructure design.
- **Empirical Validation:** Simulation experiments on networks of 100–5000 nodes demonstrate that Dijkstra's algorithm is the most stable and scalable baseline for non-negative networks, with A* offering additional acceleration when suitable heuristics exist.
- **Practical Guidance:** The study provides clear guidelines on when to use each algorithm, highlighting the limited role of Bellman–Ford (mainly for negative-weight validation) and the trade-off between speed and variability for A*.

Conclusion and Future Work

This paper has investigated the comparative performance of Dijkstra's algorithm, Bellman– Ford, and A* search within the framework of the Transportation Problem. By modelling the transport system as a directed weighted graph and redefining cost coefficients as shortest path distances, the study establishes a bridge between classical linear programming models and graph-theoretic routing. The results show that Dijkstra's algorithm remains the most reliable and scalable choice for non-negative edge-weight networks typical of distance, time, or fuel costs. A* can significantly reduce runtime when an admissible spatial heuristic such as Euclidean distance is available, but its performance is more variable and sensitive to network geometry. Bellman–Ford, although robust to negative weights and useful for detecting negative cycles, is not computationally competitive for large-scale, repeated cost-matrix generation.

Practical Implications

For logistics practitioners and transportation planners, the proposed framework provides a practical way to incorporate realistic network effects into TP models. Instead of manually specifying cost coefficients, they can be computed from explicit network representations. This facilitates scenario analysis (for example, new road projects, closures, toll changes) and supports data-driven decision making in intelligent transportation systems.

Limitations

The experiments assume static, deterministic edge weights drawn from a uniform distribution. Real networks exhibit temporal variations (peak vs. off-peak congestion), stochastic travel times, and correlations between adjacent links. Moreover, only single-commodity, single-objective TP formulations have been considered. The study also assumes that node coordinates are known for A*, which may not always be true in abstract networks.

Future Work

Several extensions are promising:

- Incorporating time-dependent and stochastic edge weights to model congestion and uncertainty (Martin, 2017);
- Developing hybrid heuristic methods that combine Dijkstra's algorithm with metaheuristics or machine learning for dynamic routing;
- Extending the framework to multi-objective optimisation, balancing cost, travel time, and emissions;
- Integrating these algorithms into real-world case studies, such as city-scale logistics or public transport planning, using real network data.

Acknowledgements

The authors acknowledge the valuable support of colleagues and technical staff who contributed insights during simulations and manuscript preparation.

References

[1]. Abed, S. S., & Noaman, S. F. (2025). Comparative analysis of shortest path algorithms. South Asian Research Journal of Engineering and Technology, 7(5), 131–143.
[2]. Ahuja, R. K., Magnanti, T. L., & Orlin, J. B.

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2026.0379
e ISSN: 2584-2854
Volume: 04 Issue: 07 July 2026
Page No: 2650 - 2657

(1993). Network Flows: Theory, Algorithms, and Applications. Prentice Hall.
[3]. Bellman, R. (1958). On a routing oblem. Quarterly of Applied Mathematics, 6(1),87–90.
[4]. Chen, K. Y. (2022). An improved A* search algorithm for road networks using new heuristic estimation. arXiv preprintarXiv:2208.00312.
[5]. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.
[6]. Dantzig, G. B. (1951). Application of the simplex method to a transportation problem. In T. C. Koopmans (Ed.), Activity Analysis of Production and Allocation (pp. 359–373). Wiley.
[7]. Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. Numerische Mathematik, 1, 269–271.
[8]. Dreyfus, S. E. (1969). An appraisal of some shortest-path algorithms. Operations Research, 17(3), 395–413.
[9]. Ford, L. R. (1956). Network Flow Theory. RAND Corporation.
[10]. Fredman, M. L., & Tarjan, R. E. (1987). Fibonacci heaps and their uses in improved network optimization algorithms.Journal of the ACM, 34(3), 596–615.
[11]. Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics,4(2), 100–107.
[12]. 12. Hitchcock, F. L. (1941). The distribution of a product from several sources to numerous localities. Journal of Mathematics and Physics, 20, 224–230.
[13]. Johnson, D. B. (1977). Efficient algorithms for shortest paths in sparse networks. Journal of the ACM, 24(1), 1–13.
[14]. Magzhan, K., & Jani, H. M. (2013). A review and evaluations of shortest path algorithms. International Journal of Scientific and Technology Research, 2(6), 99–104.
[15]. Martin, D. P. (2017). Dynamic shortest path and transitive closure algorithms: A survey. arXiv preprintarXiv:1709.00553.
[16]. Nosirov, S., Azimov, D., & Adilov, A. (2022). A review of the shortest path problem in graph theory. International Journal of Engineering Science and Computing, 12(5), 12345–12350.
[17]. Strasser, B., & Zeitz, T. (2019). A fast and tight heuristic for A* in road networks. arXiv preprintarXiv:1910.12526.
[18]. Zhan, F. B., & Noon, C. E. (1998). Shortest path algorithms: An evaluation using real road networks. Transportation Science, 32(1), 65–73.