Journal of Statistics Applications & Probability
An International Journal

http://dx.doi.org/10.18576/jsap/150407

A Hybrid Bellman–Ford–Dijkstra (DB) Algorithm for Efficient Shortest Path Computation

Aijaz Ahmad Magray[1], *D. Raju*[2], *Aafaq A. Rather*[3,*], *Ikram Ali*[4], *Vidya Yerneni*[3], *Ahmed A.F Osman*[5], *Mohammed Ataelfadiel*[5], *Ala Abdullah*[5], *Maryam Nasser Almusallam*[5], *Osman Elwasila Elshiekh*[6], *Ali Salem Bin Sama*[6] *and P. Srinivasa Rao*[7]

[1] Department of Mathematics, JB institute of engineering and technology, Hyderabad, 500075, India
[2] Department of Mathematics, Jawaharlal Technological University, Kukatpally, Hyderabad - 500085, India
[3] Symbiosis Statistical Institute, Symbiosis International (Deemed University), Pune, India
[4] Department of CSE, Apex Institute of Technology, Chandigarh University, Gharuan Mohali Punjab-412207, India
[5] Applied College, King Faisal University, P.O. Box 400, Al-Ahsa 31982, Saudi Arabia
[6] Department of Management Information Systems (MIS), College of Business Administration, King Faisal University, Al-Ahsa, Eastern Province, Kingdom of Saudi Arabia
[7] Department of Computer Science, JB institute of engineering and technology, Hyderabad, 500075, India

Received: 2 May 2026, Revised: 5 Jun. 2026, Accepted: 28 Jun. 2026
Published online: 1 Jul. 2026

Abstract: In graph theory, detecting the shortest path between two active nodes remains a significant challenge in the research community. Finding the minimum-cost path between two nodes remains an important research problem with numerous practical applications. Although Dijkstra's algorithm provides efficient shortest-path computation for graphs with non-negative edge weights and Bellman-Ford handles negative edge weights, neither algorithm simultaneously offers efficient computation, reduced search space, and low computational overhead. This limitation motivates the development of a hybrid algorithm that combines the strengths of both approaches while overcoming their individual weaknesses. In this paper, we propose a novel hybrid DB Algorithm that integrates the Bellman–Ford and Dijkstra algorithms to efficiently compute shortest paths in weighted graphs containing negative edge weights while reducing computational time and search-space complexity. The proposed DB Algorithm addresses the limitations of existing shortest-path algorithms by reducing computational overhead while efficiently handling negative edge weights and minimizing time and space complexity. The proposed algorithm is evaluated using two key performance metrics: Time to search for the Shortest Distance (TSSD) and computational space complexity. Experimental results demonstrate that the proposed DB Algorithm significantly reduces computational overhead while accurately identifying shortest paths in weighted graphs. Simulation results validate the effectiveness and computational efficiency of the proposed DB Algorithm. In the future, the proposed DB Algorithm has potential applications to wireless communication systems, network routing, multidirectional communication protocols, and quantum computing applications.

Keywords: Hybrid, DB-Algorithm, Active Nodes, Shortest Distance (SD), Time to Search Shortest Distance

1 Introduction

The well-studied research topic in graph theory is the shortest-path. It is indeed a major issue and a serious challenge in the research community. The study of shortest path problems between the vertices (nodes) and edges (lines) in a graph is an active research topic for research community. Therefore, a great deal of attention has been paid in graph theory to optimizing shortest-path, with the minimum length criteria between two nodes. Therefore, numerous algorithms have been redesigned and implemented for finding a shortest path between two nodes. Due to its wide range of applications which include network routing protocols, route planning, traffic control, path finding in social networks, computer games, and transportation systems, forging us an exclusive state of shortest path issue in the graphs. The purpose to find the shortest path with all variants between two nodes in a graph is a complex act that needed to be resolved with the special

* Corresponding author e-mail: aafaq7741@gmail.com

techniques in mathematics. Several numbers of techniques suggested by the research scholars include Greedy algorithm, M. Fredman, Floyd Warshall algorithm [1,2], Dijkstra's shortest path algorithm [3,4], Bellman Ford-algorithm [5,6], and Genetic algorithm for optimizing the route between the two nodes in a graph. These algorithms are designed to determine the shortest path in a graph. Shortest path algorithms constitute a family of algorithms used to solve shortest path problems.

Several research solutions have been implemented to overcome the short path problems in a graph. However, they fail to address all the variant vulnerabilities of space and time that are associated with the shortest paths [7,8]. Our aim is to present a new hybrid algorithm model which handles all inherited limitations and vulnerabilities that occur in previous algorithms. The new hybrid model has been proposed to sort out the time and space complexity issues for graph in this paper. It has an ability to find the shortest path between two vertices or nodes in graph theory.

2 The Proposed DB-Algorithm

The proposed DB-algorithm is the Hybridization of two or more meta-heuristic optimization algorithms for searching for the most suitable and optimal solution in a specific search space [9, 10]. The proposed DB-algorithm works on both positive and negative edge weights through parallel operations. Therefore, parallel operations of the proposed algorithm hold good properties which make it capable of working on different methods and resolving the problems efficiently. For example, searching is efficient when the problems can be solved with many solutions.

Prime objective of Hybridization is as:

1.To address vulnerability of specific meta-heuristic optimization algorithm.
2.To address limitations of specific meta-heuristic optimization algorithm.
3.To counter disadvantages of specific meta-heuristic optimization algorithm.
4.To integrate the advantages of specific meta-heuristic optimization algorithm.

In the proposed DB Algorithm, Phase One (P1) represents a search-space pruning stage in which the Bellman–Ford algorithm is used to identify feasible shortest-path candidates while handling negative edge weights. This stage generates a reduced search space by eliminating infeasible or non-promising paths. Phase Two (P2) applies Dijkstra's algorithm to the reduced search space generated in Phase One, thereby improving computational efficiency. Therefore, the term "phase" refers to two sequential computational stages rather than graph clustering, and no partitioning of vertices into mathematical clusters is performed. Figure 1 depicts the concept of hybridization of two meta-heuristic optimization algorithms which addresses the limitations and vulnerabilities by integrating the advantages of other meta-heuristic optimization algorithm and facilitates commitment for convergence with fewer time parameters and space complexity.

Hence, the proposed DB-Algorithm is a possible method for solving the negative weights in the search space. It can overcome the limitations which are constituted by the previous designs. The proposed algorithm can perform parallel search of various nodes in space which has the ability to consume less time in space to achieve target. This ability will become a benchmark to be the fastest algorithm. Dijkstra's shortest path algorithm has a blind search. It calculates the entire path from source to destination, which makes algorithm inefficient because it wastes more time to reach to the necessary-resources. In addition, if the space search of an algorithm increases, the time required for the optimal path also

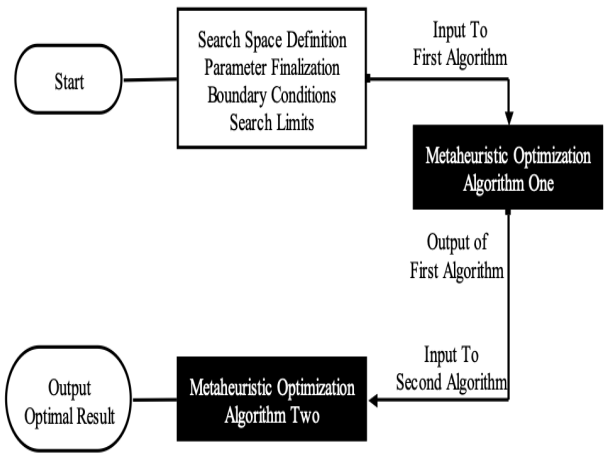

Fig. 1: Hybridization of Two Optimization Algorithm

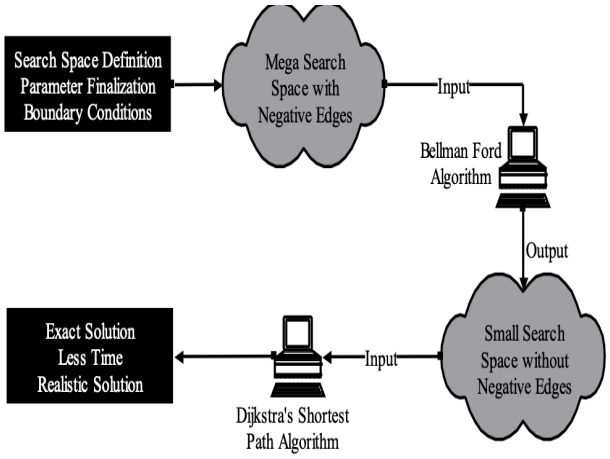

Fig. 2: The proposed DB-Algorithm

increases exponentially. In addition, it works with non-negative weights for the edges with slight modifications but fails for negative weight in the space graph, which swings the system and hangs the final results. However, nested loops taken by Dijkstra's algorithm leads to increase the search time in space [11, 12]. Due to some limitations of Dijkstra's algorithm like using nested loops, which increase the search time, it becomes necessary work for research community to implement a new algorithm which can span less time in space, using limited loops. In this regard a new DB-Algorithm algorithm has been proposed.

The proposed DB-algorithm provides a more accurate and realistic approach in nature in the context of real-world problems. The DB-algorithm works with both positive and negative edge weights. It overcomes the limitations of Dijkstra's Algorithm. Initially Bellman-Ford-algorithm is initiated with parameter subjected to target search space; the output of this search will be input for Dijkstra's Algorithm and is search space for the second algorithm. However, the proposed algorithm is shown in Figure 2, which is free of negative edges and is smaller in size than previous designs.

3 The Java Code Segment using for the proposed DB-Algorithm

Optimal Shortest Path selection process for the proposed DB-algorithm has been checked in the java cod as:

Declare Int N_D Number_of_Node, N_C Number_of_Cluster, D_N Distance Between Nodes
Int N_L Loop_1_Count, N_{L2} Loop_2_Count

Start phase one - Level 1
  Start phase one - Level 2
    Start phase one - Level 3
      Repeat-Step-3 for $i = 1,2,\ldots,n$ **Start-Loop One**
        Repeat-loop for $i = 1,2,\ldots,n$ **Start-Loop Two**
          Optimal Cluster Selection Process Start
          Check if $a[V_s,i] = 0$
          Optimal Shortest Path Selection Process Start
          In case no direct-edge among V_s & V_i
          Then: leng$[i]$ = Infinity [Infinity value is taken to be 999]
                path$[i] = 0$ [Invalid path]
          Else: leng$[i] = a[V_s,i]$ and path$[i] = V_s$
    End Phase One - Level_3
  End Phase_One - Level_2
End Phase_One Level - 1

Second Level Clustering Start with new non negative search space, shortened search space
Start Phase Two Level_1
  Start Phase Two Level_2
    Start Phase Two Level_3
      Repeat-Step3-for $i = 1,2,\ldots,n$ **Start Loop One**
        Repeat-loop for $i = 1,2,\ldots,n$ **Start Loop Two**
          Optimal Cluster Selection Process Start
          Check if $a[V_s,i] = 0$
          Optimal Shortest Path Selection Process Start
          In case no-direct edge among V_s & V_i
          Then: leng$[i]$ = Infinity [Infinity-value is taken to-be 999]
                path$[i] = 0$ [Invalid path]
          Else: leng$[i] = a[V_s,i]$ and path$[i] = V_s$
    End Phase_Two Level_3
  End Phase_Two Level_2
End Phase_Two Level_1

Optimal Shortest Path Selection Process End: Print Result.

4 Flow Chart

Figure 3 describes the flow chart of the proposed algorithm.

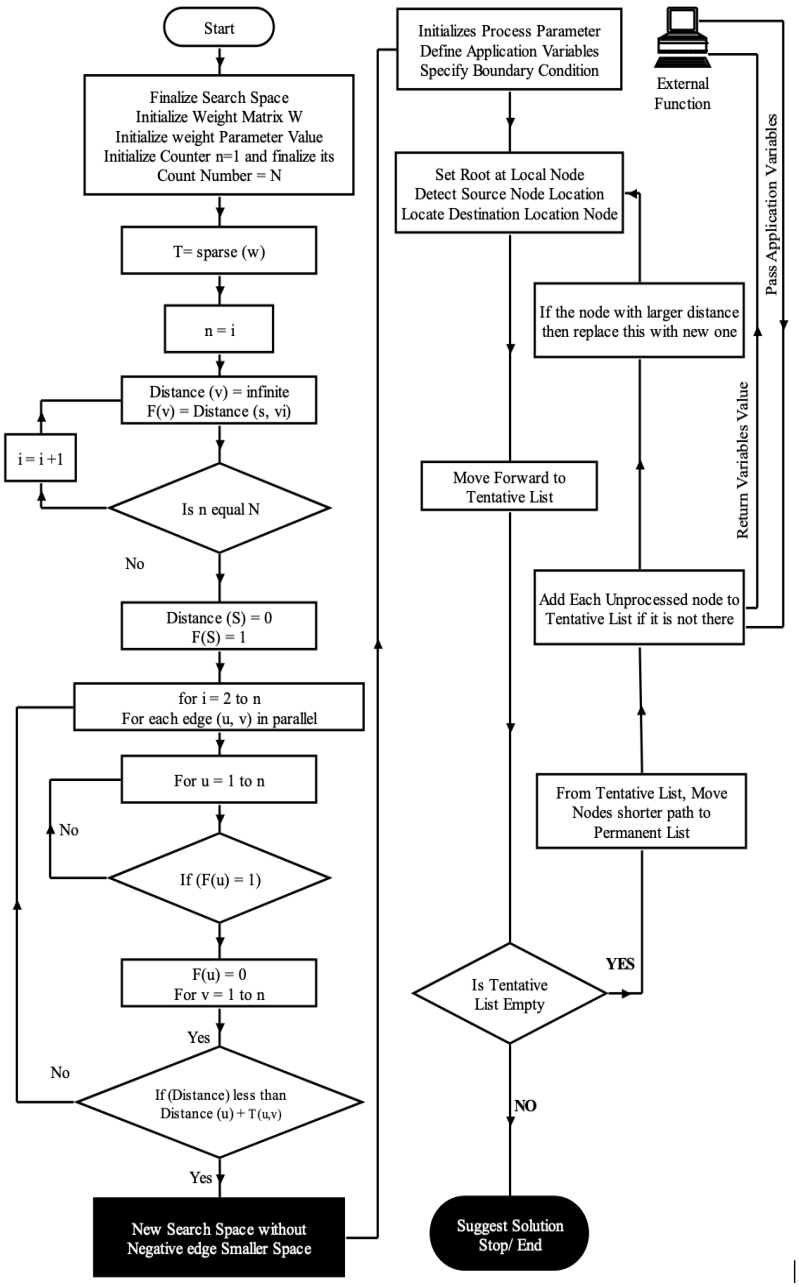

Fig. 3: The Proposed DB-Algorithm Flow Chart

5 Result

5.1 System Configuration

–Machine: Dell Latitude 3440
–Processor: 13th Gen Intel® Core™ i7-1355U, 1.70 GHz
–RAM: 16.0 GB (15.7 GB usable)
–Operating System: Windows 11 Education, Version 23H2 (OS Build 22631.4391)
–System Type: 64-bit operating system, x64-based processor
–Device ID: C7C2F718-4A80-4576-94C2-E6A7EEF45FCA
–Windows Feature Experience Pack: 1000.22700.1047.0

5.2 Graphs with 3 vertices

We tested the algorithms with graphs containing 0, 1 and 2 vertices, using different paths. The execution time of the DB algorithm was calculated and compared with that of Dijkstra's algorithm. All the test results are provided below (see Figure 4 for a graphical representation and Table 1 for detailed numerical values), and the result is clearly mentioned in Table 2.

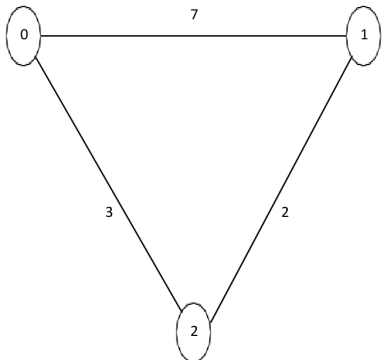

Fig. 4: Graph with Shortest path

Table 1: Adjacency Matrix of 3 vertices

	0	**1**	**2**
0	0	7	3
1	7	0	2
2	3	2	0

Table 2: Shortest path and execution time table (3 vertices)

Sr. No.	**Vertex**	**Target Target (Source-to-Dest.)**	**Possible Best Possible Best Shortest Path**	**Total Total Distance**	**Dijkstra Algorithms Execution Time Complexity**	**DB Algorithms**	
						Execution Time Complexity	**Convert to (Nano Second)**
01	03	0–1	0–2–1	5	19779800 ns	0.1005 ms	100,500 ns
02	03	1–0	1–2–0	5	18160900 ns	0.083 ms	83,000 ns
03	03	2–1	2–1	2	19289700 ns	0.0873 ms	87,300 ns
04	03	2–0	2–0	3	19910900 ns	0.0924 ms	92,400 ns
Average Execution Time					**19285325 ns**	–	**363200 ns**

Execution time calculated based on different shortest paths:

$$\begin{aligned}\text{Percentage Faster} &= \frac{\text{Dijkstra Algorithms Average} - \text{DB Algorithms Average}}{\text{Dijkstra Algorithms Average}} \times 100 \\ &= \frac{19285325 - 363200}{19285325} \times 100 \\ &= \frac{18922125}{19285325} \times 100 \\ &= 98.12\% \end{aligned} \tag{1}$$

Figure 4 and Table 2 demonstrate the performance of the proposed DB Algorithm on a small graph containing three vertices. The algorithm correctly identifies the shortest paths while reducing the search space before applying Dijkstra's algorithm. The experimental results show that the hybrid approach significantly decreases execution time without affecting the correctness of the computed shortest paths.

5.3 Graph with vertex -4

We tested the algorithms with graphs containing 4 vertices shown in Figure 5 using different paths. The execution time of the DB algorithm was calculated and compared with that of Dijkstra's algorithm. All the test results are provided and mentioned in Table 3 and Table 4 determines the shortest path and execution time which clearly shows that DB algorithm result is 99.50% better than Dijkstra's algorithm.

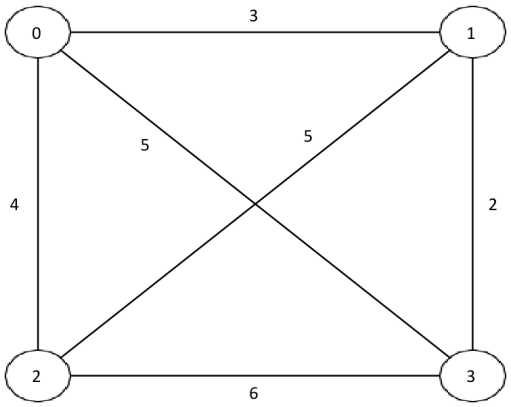

Fig. 5: Graph with shortest path

Table 3: Adjacency Matrix of 4 vertices

	0	**1**	**2**	**3**
0	0	3	4	5
1	3	0	5	2
2	4	5	0	6
3	5	2	6	0

Table 4: Shortest path and execution time table (4 vertices)

Sr. No.	**Vertex**	**Target Target (Source-to-Dest.)**	**Possible Best Possible Best Shortest Path**	**Total Total Distance**	**Dijkstra Algorithms Execution Time Complexity**	**DB Algorithms**	
						Execution Time Complexity	**Convert to (Nano Second)**
01	4	1–2	1–2	5	27752100 ns	0.0889 ms	88900 ns
02	4	0–3	0–1–3	5	21884000 ns	0.122 ms	122000 ns
03	4	3–0	3–0	5	20483600 ns	0.125 ms	125000 ns
04	4	2–3	2–3	6	23297200 ns	0.098 ms	98000 ns
05	4	2–0	2–0	4	17697000 ns	0.1159 ms	115900 ns
Average Execution Time					**22222780 ns**	–	**109960 ns**

Execution Time Calculated based on different shortest paths:

$$\begin{aligned}\text{Percentage Faster} &= \frac{\text{Dijkstra's Algorithms Average} - \text{DB Algorithms Average}}{\text{Dijkstra's Algorithms Average}} \times 100 \\ &= \frac{22222780 - 109960}{22222780} \times 100 \\ &= \frac{22112820}{22222780} \times 100 \\ &\cong 99.50\% \end{aligned} \tag{2}$$

Figure 5 and Table 4 evaluate the proposed algorithm on a graph with four vertices. As the graph size increases, the DB Algorithm maintains accurate shortest-path computation while efficiently handling the reduced search space generated during Phase One. The results indicate that the computational advantage is preserved as graph complexity increases.

5.4 Graph with vertex -6

The third test was on the 6 vertices with so many shortest paths between source to destination as shown in Figure 6 and also in Table 5 determines the weight and paths. The execution time mentioned in Table 6 was calculated and compared with Dijkstra's algorithm, which shows 96.23% better result than the available algorithm.

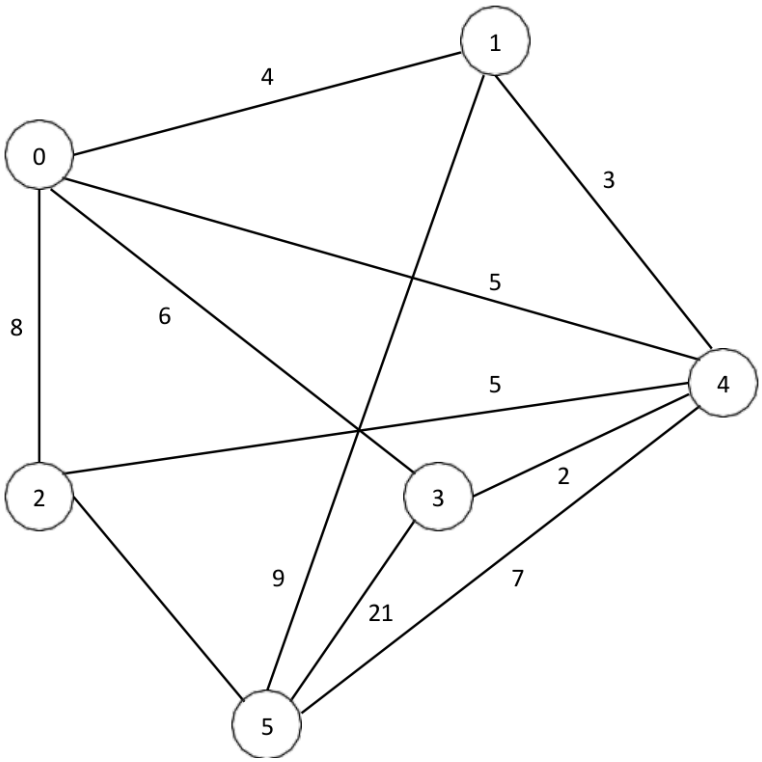

Fig. 6: Graph with Shortest Path

Table 5: Adjacency Matrix of 6 vertices

Vertex	**0**	**1**	**2**	**3**	**4**	**5**
0	0	4	8	6	5	0
1	4	0	0	0	3	9
2	8	0	0	0	5	6
3	6	0	0	0	2	4
4	5	3	5	2	0	7
5	0	9	6	4	7	0

Table 6: Shortest path and execution timetable (6 vertices)

Sr. No.	Vertex	Target Target (Source-to-Dest.)	Possible Best Possible Best Shortest Path	Total Total Distance	Dijkstra Algorithms Execution Time Complexity	DB Algorithms	
						Execution Time Complexity	Convert to (Nano Second)
01	6	0–5	0–3–5	10	15983400 ns	0.1304 ms	130400 ns
02	6	3–2	3–4–2	7	16126100 ns	0.1332 ms	133200 ns
03	6	1–3	1–4–3	5	15518600 ns	0.1343 ms	134300 ns
04	6	2–1	2–4–1	8	15083500 ns	0.1246 ms	124600 ns
05	6	4–0	4–0	5	21313100 ns	0.1094 ms	109400 ns
Average Execution Time					**16804940 ns**	–	**631900 ns**

Execution Time Calculated based on different shortest paths:

$$\begin{aligned}\text{Percentage Faster} &= \frac{\text{Dijkstra Algorithms Average} - \text{DB Algorithms Average}}{\text{Dijkstra Algorithms Average}} \times 100 \\ &= \frac{16804940 - 631900}{16804940} \times 100 \\ &= \frac{16173040}{16804940} \times 100 \\ &= 96.23\% \end{aligned} \tag{3}$$

Figure 6 and Table 6 illustrate the behavior of the proposed algorithm on a denser graph with six vertices and multiple candidate paths. The reduced search space enables the algorithm to avoid unnecessary computations, resulting in consistent execution-time improvements while maintaining optimal path selection.

5.5 Graph with vertex -10

Finally, the testing was done on the 10 vertices shown in Figure 7 and so many shortest paths were available from source to destination mentioned in Table 7 and the execution time for the determination of shortest path was calculated which shows 98.91% better result than the Dijkstra's algorithm which is mentioned in Table 8.

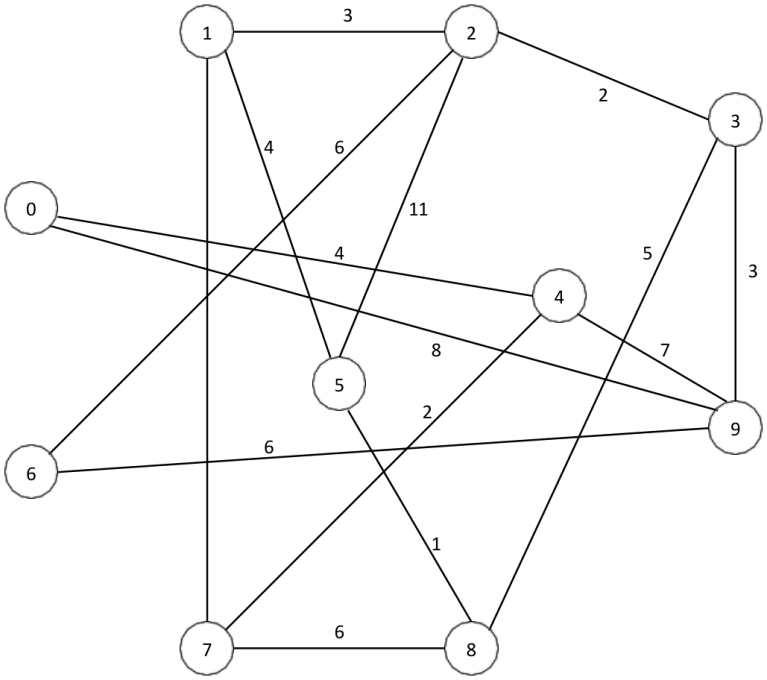

Fig. 7: Graph with Shortest path

Table 7: Adjacency Matrix of 10 vertices

Vertex	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	7	0	0	0	0	8
1	0	0	3	0	0	4	0	7	0	0
2	0	3	0	2	0	11	6	0	0	0
3	0	0	2	0	0	0	0	0	5	3
4	7	0	0	0	0	0	0	2	0	7
5	0	4	11	0	0	0	0	0	1	0
6	0	0	6	0	0	0	0	0	0	6
7	0	4	0	0	2	0	0	0	6	0
8	0	0	0	5	0	1	0	6	0	0
9	8	0	0	3	7	0	6	0	0	0

Table 8: Shortest path and execution timetable (10 vertices)

Sr. No.	Vertex	Target Target (Source-to-Dest.)	Possible Best Possible Best Shortest Path	Total Total Distance	Dijkstra Algorithms Execution Time Complexity	DB Algorithms	
						Execution Time Complexity	Convert to (Nano Second)
01	10	0–8	0–4–7–8	15	16122500 ns	0.2489 ms	248900 ns
02	10	2–7	2–1–7	10	17276000 ns	0.1233 ms	123300 ns
03	10	3–5	3–8–5	6	17898500 ns	0.153 ms	153000 ns
04	10	1–9	1–2–3–9	8	16636500 ns	0.1364 ms	136400 ns
05	10	4–6	4–9–6	13	15989900 ns	0.2502 ms	250200 ns
Average Execution Time					**16784680 ns**	–	**182360 ns**

Execution Time Calculated based on different shortest paths:

$$\begin{aligned}\text{Percentage Faster} &= \frac{\text{Dijkstra Algorithms Average} - \text{DB Algorithms Average}}{\text{Dijkstra Algorithms Average}} \times 100 \\ &= \frac{16784680 - 182360}{16784680} \times 100 \\ &= \frac{16602320}{16784680} \times 100 \\ &\cong 98.91\% \end{aligned} \tag{4}$$

Figure 7 and Table 8 demonstrate the scalability of the proposed DB Algorithm on a larger graph consisting of ten vertices. Despite the increased number of possible routes, the hybrid approach continues to identify optimal shortest paths efficiently. These results indicate that the proposed method scales well with increasing graph size while maintaining computational efficiency.

6 The DB -Algorithm facilitates various advantages like

a)Negative weight or edge present in the search space is a major issue in Dijkstra's algorithms which has been countered in initial stage in DB-Algorithm such that new search space is suggested for optimal result.

b)Dijkstra's algorithm implements recursive search technique which calls another function for optimization, this recursive calling results increases search time However, in the proposed DB-Algorithm already shortened the search space, now Dijkstra's algorithm required confining the search in smaller space, this technique saves time and resources.

c)The proposed DB Algorithm significantly reduces the possibility of entering infinite loops, thereby ensuring convergence during the search process.

7 Discussion

The proposed DB Algorithm effectively overcomes one of the major limitations of Dijkstra's algorithm by handling negative edge weights during the Bellman–Ford preprocessing stage. This generates a reduced search space before Dijkstra's algorithm is executed, thereby improving computational efficiency.

Furthermore, by reducing the search space before applying Dijkstra's algorithm implements recursive search technique which calls another function for optimization, this recursive calling results increases search time. However, in the proposed DB-Algorithm already shortened the search space. Consequently, Dijkstra's algorithm searches only within the reduced search space, which decreases computational time and improves routing efficiency of the search in smaller space; this technique saves time and resources.

In addition, the proposed DB-Algorithm reduces the possibility of entering infinite loops The proposed DB-Algorithm significantly reduces the possibility of entering infinite loops, thereby ensuring convergence during the search process. for infinite loop which Confirms convergence within search space.

The experimental evaluation conducted on graphs containing 3, 4, 6, and 10 vertices demonstrates the effectiveness and scalability of the proposed DB-Algorithm. The results consistently show that the hybrid Bellman–Ford–Dijkstra approach computes the correct shortest paths while reducing computational overhead across graphs of different sizes. Across all experimental cases, the proposed algorithm achieved an execution-time improvement ranging from approximately 96 to 99% compared with the conventional Dijkstra algorithm. The consistent performance improvement indicates that the search-space reduction achieved during Phase One enables faster execution without compromising shortest-path accuracy. These findings validate the robustness, scalability, and practical applicability of the proposed algorithm for large-scale shortest-path computation problems.

8 Future Scope

This research field has a great potential in the future. This research field can be applied in different areas such as;

–Wireless communication networks
–Internet routing
–Cloud computing
–Intelligent transportation systems
–IoT networks
–SDN
–Autonomous vehicles
–Missile Defence Application
–Rocket Trajectory Application
–Agriculture and Environment Assisting Applications

9 Conclusion

This study presents the DB-algorithm, which addresses time and space complexity issues in previous algorithms. The results obtained using the proposed method show suitable performance and efficiency in both time and space problems in the shortest paths. Comprehensive analysis and testing confirm its effectiveness and efficient solution, addressing significant challenges in computational performance. The results emphasize the proposed DB-algorithm has the potential to establish a new standard in algorithm development. The innovative design and optimization techniques open opportunities for widespread application in numerous areas where efficiency is paramount. Future studies aim to enhance its adaptability further and explore its integration into specialized cases. Hence, the DB-algorithm marks a significant step forward in computational science, providing a reliable solution to persistent challenges and inspiring further advancements in the research field.

10 Declarations

Ethical Approval: No ethical approval is required.

Data Availability: No datasets were analyzed during the current study.

Conflict of Interest: There is no conflict of interest among all the authors.

Author's Contribution: All the authors have equally contributed.

Acknowledgment: This work was supported by the Deanship of Scientific Research, Vice Presidency for Graduate Studies and Scientific Research, King Faisal University, Saudi Arabia [KFU263839].

References

[1] Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton, New Jersey. DOI: https://doi.org/10.1515/9781400835263

[2] Bellman, R. (1958). On a routing problem. *Quarterly of Applied Mathematics*, 16(1), 87–90. DOI: https://doi.org/10.1090/qam/102435

[3] Fredman, M. L. (1976). New bounds on the complexity of the shortest path problem. *SIAM Journal on Computing*, 5(1), 83–89. DOI: https://doi.org/10.1137/0205006

[4] Floyd, R. W. (1962). Algorithm 97: Shortest path. *Communications of the ACM*, 5(6), 345. DOI: https://doi.org/10.1145/367766.368168

[5] Warshall, S. (1962). A theorem on Boolean matrices. *Journal of the ACM*, 9(1), 11–12. DOI: https://doi.org/10.1145/321105.321107

[6] Kim, G., Kim, S., & Jang, I. G. (2022). Loopwise route representation-based topology optimization for the shortest path problems. *IEEE Access*, 10, 128835–128846. DOI: https://doi.org/10.1109/ACCESS.2022.3227388

[7] Coy, S., Czumaj, A., Scheideler, C., Schneider, P., & Werthmann, J. (2022). Routing schemes for hybrid communication networks. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing (PODC '22)*, pages 343–353. ACM. DOI: https://doi.org/10.1145/3519270.3538421

[8] Gajjar, K., Jha, A. V., Kumar, M., & Lahiri, A. (2024). Reconfiguring shortest paths in graphs. *Algorithmica*, 86(10), 3309–3338. DOI: https://doi.org/10.1007/s00453-024-01263-y

[9] Kim, G., Kim, C., Park, S., Kim, J., & Jang, I. G. (2025). Linear programming of the loopwise route representation for the shortest path problems by introducing the reference link flow. *IEEE Access*, 13, 38856–38864. DOI: https://doi.org/10.1109/ACCESS.2025.3546320

[10] Andreoletti, D., Rottondi, C., Giordano, S., & Bianco, A. (2025). Scalable and privacy-preserving inter-AS routing through machine-learning-based graph pruning. *IEEE Access*, 13, 21891–21905. DOI: https://doi.org/10.1109/ACCESS.2025.3536545

[11] Hlupic, N., Basch, D., Ivanjko, E., & Prister, J. (2025). An optimized route assignment for preventing traffic jams. *IEEE Access*, 13, 61207–61224. DOI: https://doi.org/10.1109/ACCESS.2025.3553492

[12] Dory, M., & Matar, S. (2025). Massively parallel algorithms for approximate shortest paths. *Distributed Computing*, 38(2), 131–162. DOI: https://doi.org/10.1007/s00446-025-00482-y