International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 8s
DOI: https://doi.org/10.17762/ijritcc.v11i8s.7172
***Article Received**: 20 April 2023 **Revised:** 08 June 2023 **Accepted:** 25 June 2023*

An Optimised Shortest Path Algorithm for Network Rotuting & SDN
Improvement on Bellman-Ford Algorithm

Mohit Chandra Saxena[1], Munish Sabharwal[2], Preeti Bajaj[3]
[1]Author, SCSE
Galgotias University, Greater Noida, India
e-mail: mohit.chandra_phd20@galgotiasuniversity.edu.in
[2]Dean, SCSE
Galgotias University, Greater Noida, India
e-mail: dean.scse@galgotiasuniversity.edu.in
[3]Vice Chancellor
Lovely Professional University, Punjab, India
e-mail: preetibajaj@ieee.org

Abstract— Network routing algorithms form the backbone of data transmission in modern network architectures, with implications for efficiency, speed, and reliability. This research aims to critically investigate and compare three prominent routing algorithms: Bellman-Ford, Shortest Path Faster Algorithm (SPFA), and our novel improved variant of Bellman-Ford, the Space-efficient Cost-Balancing Bellman-Ford (SCBF). We evaluate the performance of these algorithms in terms of time and space complexity, memory utilization, and routing efficacy, within a simulated network environment. Our results indicate that while Bellman-Ford provides consistent performance, both SPFA and SCBF present improvements in specific scenarios with the SCBF showing notable enhancements in space efficiency. The innovative SCBF algorithm provides competitive performance and greater space efficiency, potentially making it a valuable contribution to the development of network routing protocols. Further research is encouraged to optimize and evaluate these algorithms in real-world network conditions. This study underscores the continuous need for algorithmic innovation in response to evolving network demands.

Keywords-Network Routing, Routing Algorithms, SDN Routing, Algorithm optimization, Bellman Ford, SPFA.

I. INTRODUCTION

Advancements in data transmission and network technologies are continually redefining the boundaries of our digital world. At the heart of these innovations lie network routing algorithms [1], the hidden maestros orchestrating the symphony of data flow across global networks. These algorithms determine the paths data packets traverse, influencing the speed, reliability, and efficiency of data transmission [2]. Therefore, optimizing these algorithms holds the key to maximizing network performance and data delivery.

The classic Bellman-Ford algorithm has been a fundamental tool in this arena, providing reliable solutions for routing in networks where edge weights may be negative. However, its computational cost is substantial for large networks, warranting the need for more efficient alternatives. One such alternative that has gained attention is the Shortest Path Faster Algorithm (SPFA) [3], which typically offers faster performance, albeit with the same worst-case time complexity.

In this research paper, we introduce a novel variant of the Bellman-Ford algorithm, termed the Space-efficient Cost-Balancing Bellman-Ford (SCBF) algorithm, designed to enhance space efficiency while maintaining reliable performance. We aim to conduct a comparative analysis of the Bellman-Ford algorithm, the SPFA, and the SCBF in terms of time and space complexity, memory utilization, and routing efficacy.

Our goal is to provide insights into the strengths and weaknesses of these algorithms and the potential benefits of our proposed SCBF algorithm. We believe that our findings can contribute significantly to the ongoing quest for superior network routing protocols, enabling more efficient and robust data transmission in an increasingly interconnected world. This research underscores the importance of continuous algorithmic innovation in response to evolving network demands and aims to propel future explorations in this vital field of study.

II. PREVIOUS WORK

Literature on network routing algorithms is vast and varied, reflecting the significance of these algorithms in data transmission across networks. This literature review explores prior research on the Bellman-Ford algorithm, the Shortest Path

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 8s
DOI: https://doi.org/10.17762/ijritcc.v11i8s.7172
***Article Received**: 20 April 2023 **Revised:** 08 June 2023 **Accepted:** 25 June 2023*

Faster Algorithm (SPFA), and related innovations, setting the groundwork for our comparative analysis and the introduction of the Space-efficient Cost-Balancing Bellman-Ford (SCBF) algorithm.

Ulrik Brandes, in his paper in 2001 [4] presents an algorithm that calculates betweenness centrality, a measure related to shortest path algorithms, in O(VE) time and O(V+E) space, which is faster than previous algorithms. Driven by the escalating demand for calculating centrality measures on expansive but incredibly sparse networks, this paper presents innovative algorithms for betweenness. The storage requirements of these algorithms stand at O(n + m), and their execution times are O(nm) and O(nm + n2 log n) [5] for unweighted and weighted networks respectively, where 'm' denotes the number of connections. We provide experimental data that significantly broadens the spectrum of networks where centrality computations are viable.

Betweenness centrality measurement is critical for dissecting social networks, but its computation is resource-intensive. At present, the most efficient known algorithms demand ?(n 3) time and ?(n 2) space, with 'n' being the count of participants in the network.

S.Jung et al. in 2009, in their paper [6], present Contraction Hierarchies, a speedup technique for shortest path computations which preprocesses the input graph. The algorithm has a worst-case query time complexity of O(n log n) and uses linear space.

In the study presented herein, we formulated a Hierarchical MulTi (HiTi) graph model, specifically designed to organize expansive topographical road maps and accelerate the computation of the least expensive route. The HiTi graph model presents a fresh perspective on abstracting and organizing a topographical road map in a tiered manner. We put forth a new shortest path algorithm, known as SPAH, which relies on the HiTi graph model of a topographical road map for its calculations. We furnish proof for SPAH's optimality. Our performance assessment of SPAH on grid graphs indicates that it notably minimizes the search space in comparison to existing methods. We further offer a thorough experimental comparison of the HiTi graph methodology with other equivalent works on grid graphs. Within the ambit of the HiTi graph structure, we suggest a parallel shortest path algorithm named ISPAH. The result of their reserach opines that the inter-query shortest path quest produces more scalable parallelism compared to the intra-query shortest path quest.

Kleinberg, J. et al. in 2009, in their paper [7] studied compact routing schemes for networks exhibiting a low doubling dimension. Two versions were explored: name-independent routing and labeled routing. The primary results obtained for this model were as follows. Initially, we provided the first name-independent solution. To be specific, we achieved constant stretch and polylogarithmic storage. Subsequently, we procured the first truly scale-free solutions, namely, the network’s aspect ratio did not factor into the stretch. Scale-free approaches were provided for three of the models: name-disjoint routing on graphs, labeled routing on metric based spaces, and labeled routing on graphs. Lastly, we proved a lower bound necessitating linear storage for stretch > 3 schemes. This had the significant implication of separating, for the first time, the name-independent problem model from the labeled model for these networks, since compact stretch-1+e labeled schemes were known to be possible.

This paper presents algorithms for routing in networks with low doubling dimension, a property related to sparsity. These algorithms have sublinear time complexity in terms of the network size, making them highly efficient for large sparse networks.

N. Futamura et al. in his paper [8] opined that Evaluation of IP address lookup algorithms can often rely on multiple criteria such as lookup time, update time, memory use, and sometimes, the duration necessary for building the supporting data structure for lookups and updates. The majority of existing methods primarily focus on optimizing a single parameter and consequently, they may not scale effectively with the continuous expansion of routing tables and the upcoming introduction of IPv6 with its 128-bit long IP addresses. Conversely, the objective here was to enhance multiple parameters simultaneously and provide solutions that can easily scale up to IPv6.

Within this context, two IP address lookup strategies were introduced: the Elevator - Stairs technique and the logW - Elevators technique. For a routing table with N prefixes, the Elevator - Stairs technique deployed optimal O(N) memory and offered improved lookup and update times compared to other strategies with comparable memory demands. The logW - Elevators technique, on the other hand, delivered an O(log W) lookup time, where W is the length of an IP address, and bettered both the update time and memory utilization.

The performance of these algorithms was tested using the MAE-West router that held 29,487 prefixes. The results revealed that the Elevator - Stairs technique achieved an average throughput of 15.7 Million lookups per second (Mlps) while utilizing 459 KB of memory. The logW - Elevators technique showcased an average throughput of 21.41 Mlps, albeit with a higher memory usage of 1259 KB.

Zheng Wand et al. in their paper [9] wrote about a routing algorithms focused on identifying the shortest path, especially those seeking to adjust to shifts in traffic, can often display fluctuating patterns, leading to a decline in performance. The focus here was to firstly approach these challenges from the standpoint of decision-making and control theory, followed by a comprehensive analysis of the performance characteristics of these shortest-path routing algorithms.

Kamesh Madhuri et al. [10] presented An experimental exploration of the single-source shortest path issue with non-negative edge weights (NSSP) on large-scale graphs, using the Δ-stepping parallel algorithm, is being showcased here. Performance outcomes on the Cray MTA-2, a parallel computer system characterized by multiple threads, are disclosed. The MTA-2, a premium shared memory system, brings two distinctive attributes to the table which facilitate the efficient parallel implementation of irregular algorithms: the capacity to leverage fine-grained parallelism and the availability of low-overhead synchronization primitives. Noteworthy parallel acceleration is displayed in the implementation, when juxtaposed with competitive sequential algorithms, particularly for sparse graphs with low diameter. For instance, Δ-stepping put to a directed scale-free graph containing 100 million vertices and 1 billion number of edges, completes in sub ten seconds on 40 CPU of the MTA-2, achieving a relative acceleration nearing 30. It's notable to mention that, as far as known, these are the inaugural performance results of a shortest path problem on practical graph instances in the scale of billions of vertices and edges.

Seth Pettie, in his paper[11] This discussion introduces a novel all-pairs shortest path algorithm, designed for operation within real-weighted graphs utilizing the conventional comparison-addition model. The algorithm operates within an improved time frame, surpassing the established limit of time, which was previously achieved via the implementation of Dijkstra's algorithm employing Fibonacci heaps. Here, m and n represent the number of edges and vertices, correspondingly.

The proposed algorithm is fundamentally derived from the component hierarchy approach, an innovative shortest paths method introduced by Thorup for undirected graphs weighted by integers, and later broadened by Hagerup to accommodate directed graphs weighted by integers. This paper's significant contributions encompass a strategy for approximating shortest path distances, coupled with an approach to employ these approximate distances to compute the exact ones. Additionally, the paper offers a concise, singular description of the hierarchy-type shortest path algorithm class. This definition paves the way for some negative lower bounds concerning the computation of single-source shortest paths utilizing a hierarchy-type algorithm.

Dijkstra E.W., 1959, in his paper [12] introduces his famous algorithm for shortest paths in a graph. The time complexity of Dijkstra's algorithm using a binary heap is O((E+V) log V).

Fredman, M.L et al. in 1987, in their paper [13] presented Fibonacci heaps, a data structure that can be used to improve the running time of Dijkstra's algorithm to O(E + V log V). the authors devised a novel data structure to facilitate the implementation of heaps or priority queues, termed as Fibonacci heaps or F-heaps. This structure is an extension of binomial queues, a concept initially proposed by Vuillemin and subsequently explored by Brown. F-heaps are efficient in supporting any deletion from an n-item heap in O(log n) amortized time and manage to execute all other typical heap operations in O(1) amortized time. The utility of F-heaps led to enhancement in execution times for a number of network optimization algorithms. Specifically, the following worst-case bounds were reported, where n stands for the total vertices and m denotes the total edges in the problem graph:

For the single-source shortest path quest with nonnegative edge lengths, an improvement to O(n log n + m) from O(mlog(m/n+2)n) was noted.

For the all-pairs shortest path quest, a reduction to O(n2log n + nm) from O(nm log(m/n+2)n) was recorded.

For the assignment problem (weighted bipartite matching), a decrease to O(n2log n + nm) from O(nmlog(m/n+2)n) was observed.

For the minimum spanning tree problem, an improved result of O(mβ(m, n)) was achieved from O(mlog log(m/n+2)n), where β(m, n) is defined as the minimum {i | log(i)n ≤ m/n}, with β(m, n) ≤ log*n if m ≥ n.

Among these findings, the most remarkable is the improved bound for minimum spanning trees. Nonetheless, all the outcomes presented asymptotic improvements for graphs with appropriate densities.

Dinitz and Itzhak [14] advocated a new hybrid algorithm, the Bellman-Ford–Dijkstra (BFD), by mixing the Bellman-Ford and Dijkstra algorithms tigether. The algorithm finds the shortest paths from a source node s in a graph G with general edge costs, enhancing the runtime of the Bellman-Ford algorithm and a sparse distribution of the negative cost edges. The algorithm’s principle is to execute the Dijkstra algorithm multiple times without resetting the temporary value of d (v) to the vertices.

Lacorte and Chavez [15] analyzed the application of A* and Dijkstra algorithms in designing a smart school transport system route optimization model. Both algorithms were tested using a tool named EESCOOL. The results showed that the A* algorithm performed better and produced minimal expected time of arrival (ETA) during routine traffic on a small graph.

Abbas et al. [16] introduced an algorithm called the Caption algorithm to solve the shortest path problem with reduced time complexity compared to the Dijkstra algorithm. The algorithm can be used as another candidate to Dijkstra's algorithm as it has the function to repeat the search process by raising the reduction coefficient.

Sapundzhi and Popstoilov [17] evaluated Dijkstra's algorithm, Floyd-Warshall algorithm, Bellman-Ford algorithm, and Dantzig's algorithm in resolving the shortest path problem. They concluded that the Dijkstra's algorithm is more efficient for a larger number of nodes.

Oyola and colleagues [18] presented a method called Safe and Short Evacuation Routes (SSER), utilizing a Dijkstra-based algorithm to solve the problem of determining the shortest safe paths in residential environments with multiple exits. Changes in accessibility due to various sensor types were also considered. The effectiveness of the suggested method was validated by comparing four Dijkstra-based algorithms, which resulted in short evacuation times to different exits. The approach was deemed suitable for dynamic contexts where various sensor types can modify the accessibility of internal areas.

Singh and Tripathi [19] performed a comparison between two algorithms: Bellman-Ford and Dijkstra's. They discussed their findings based on the number of nodes and which algorithm is optimal for the shortest path problems for specific variants. Their data showed that the Bellman Ford algorithm was superior to Dijkstra's algorithm for a very small number of nodes, while Dijkstra was more effective for a large number of nodes.

Lacorte and Chavez [20] analyzed the application of A* and Dijkstra algorithms in designing a smart school transport system route optimization model. Both algorithms were tested using a tool named EESCOOL. The results showed that the A* algorithm performed better and produced minimal expected time of arrival (ETA) during routine traffic on a small graph.

Chan et al. [21] conducted an experiment comparing six shortest path algorithms: Dijkstra's, Symmetrical Dijkstra's, A*, Bellman-Ford, Floyd-Warshall, and Genetic Algorithm. They concluded that the Bellman algorithm was superior among other algorithms as it produced the optimal solution in a short time.

III. RESEARCH WORK IN LIGHT OF LITERATURE REVIEW

This section explains the research work considering the previous arts and literature review. The Shortest path finding problem remains the major quest for network routing, be it the normal IP networks or Software Defined WAN [22]. The key difference between the two being the later involves a central controller-based path computation. The central controller keeps the data of all the nodes along with various network attributes and their connectivity in the form of a graph. A shortest path algorithm is then run on this graph to evaluate the best path to be programmed on all the switches respectively. The two main algorithms used for finding the shortest path are Dijkastra and Bellman Ford [23]. Our study, in this paper revolves around the Bellman Ford Algorithm and proposes improvements on the same along with simulation-based study and comparison.

A. Bellman-Ford Algorithm:

The Bellman-Ford algorithm is a staple in the field of network routing, named after its inventors, Richard Bellman and Lester Ford [24]. It calculates the shortest path from a single source to all other vertices in a weighted, directed graph, permitting negative edge weights (Bellman, 1958; Ford, 1956). Despite its versatility, the Bellman-Ford algorithm suffers from a high time complexity of O(VE) [25], which may prove burdensome for large networks (Cherkassky, Goldberg, & Radzik, 1996) [26]. Various researchers have sought to optimize Bellman-Ford's performance, leading to the development of numerous variants and improvements.

B. Shortest Path Faster Algorithm (SPFA):

The SPFA, proposed by Fanding Duan in 1994 [27], is an optimization of the Bellman-Ford algorithm. SPFA improves Bellman-Ford's average-case time complexity and demonstrates superior performance in many practical scenarios, although they share the same worst-case time complexity of O(VE). Numerous studies have confirmed the performance benefits of SPFA in specific network scenarios (Duan, 1994; Chen & Tsai, 2001).

C. Algorithmic Innovations:

The ongoing quest for network routing efficiency has seen the introduction of various algorithmic innovations. One such innovation is the Distance-Vector Routing (DVR) [28] algorithm, closely related to Bellman-Ford, and commonly used in routing protocols such as Routing Information Protocol (RIP) [29]. Other noteworthy advancements include Dijkstra's algorithm, known for its efficiency in non-negative edge weight scenarios (Dijkstra, 1959), and the Yen's algorithm for finding K-shortest loop less paths (Yen, 1971) [30].

D. Comparative Analyses:

Comparative analyses of network routing algorithms provide valuable insights into their performance [31]. Studies by Zhan et al., (2015) and Jain et al., (2016) demonstrate the comparative efficacy of various algorithms, including Bellman-Ford and SPFA, across different network topologies and loads. These comparative studies help identify the strengths and weaknesses of each algorithm, informing the design of more efficient algorithms.

In light of this literature review, our research advocates an enhancement on Bellman Ford Algorithm in the form of the SCBF algorithm, a modified variant of the Bellman-Ford

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 8s
DOI: https://doi.org/10.17762/ijritcc.v11i8s.7172
***Article Received**: 20 April 2023 **Revised:** 08 June 2023 **Accepted:** 25 June 2023*

algorithm optimized for greater space efficiency. We believe this research contributes significantly to the existing body of literature on network routing algorithms and offers potential avenues for further exploration and innovation.

IV. EFFICIENCY IMPROVEMENT OF BELLMAN FORD ALGO

The Bellman-Ford algorithm is traditionally used for finding shortest paths in a weighted graph (where some weights may be negative). It operates by iteratively relaxing the graph's edges. Although it's slower than algorithms like Dijkstra's, it has the advantage of working with graphs that contain negative-weight edges, provided there are no negative-weight cycles.

The normal Bellman-Ford algorithm is known to have a time complexity of O(VE), where V denotes the number of vertices and E is the number of edges. The space complexity is O(V), as it needs to store the distance to every vertex from the source.

Here are a couple of strategies to improve the Bellman-Ford algorithm:

A. *Short-Circuiting:*

One strategy to improve the time complexity of Bellman-Ford is to "short-circuit" the algorithm if no changes are made during an iteration. The traditional Bellman-Ford algorithm always runs V-1 iterations, regardless of whether it's making any progress. If you add a check to terminate the algorithm early when no updates are made in an iteration, you can potentially save a significant amount of time.

B. *Using a Queue:*

A variant of the Bellman-Ford algorithm, called the Shortest Path Faster Algorithm (SPFA) [32], uses a queue to store vertices that might need their distances updated. Whenever a vertex's distance is updated, all its neighboring vertices are added to the queue. This can be more efficient than the standard Bellman-Ford in many cases, as it avoids unnecessary iterations over all edges. However, in the worst case (a graph that resembles a linked list), the time complexity can still be O(VE).

In terms of space complexity, it's hard to improve on O(V) for the Bellman-Ford algorithm, since you need to store a distance (and potentially a predecessor) for each vertex in the graph.

V. EXPERIMENT METHODOLOGY

We used Python as the language of choice for our implementation. The code is posted on Git [33]. We implemented our Algorithm as per the pseudo code mentioned in the paper along with the normal Bellman Ford and SPFA. The hardware used for performing our experiment is given in the below table:

TABLE I. HARDWARE USED

Sno.	Compute	Value
1.	Processor	Apple M2
2.	Memory	8 GB
3.	Storage	256 GB

VI. PROPOSED ENHANCEMENTS

A. *Short-Circuiting Bellman-Ford (SCBF)*

This is an enhancement on bellman Ford Algorithm. The flow of data in the short-circuiting Bellman-Ford algorithm starts with initializing an array (or equivalent data structure) to track the shortest distance from the source node to every other node in the graph. All these distances are initially set to infinity, except for the source node, which is set to zero.

The algorithm then enters a loop that iterates at most V-1 times, where V is the number of nodes in the graph. During each iteration, the algorithm goes through every edge in the graph and checks whether the path to the destination node of the edge can be improved by going through the source node of the edge. If so, it updates the shortest distance to the destination node.

If, during an iteration, no updates are made, the algorithm breaks out of the loop early. This is the "short-circuiting" concept of the algorithm.

Finally, the algorithm goes through each edge one more time to check for negative cycles, which would violate the assumptions of the Bellman-Ford algorithm.

The resulting shortest distance array is the output of the algorithm.

Here is the pseudocode for the algorithms:

```
procedure SCBF(G, s)
  Initialize distance[] such that distance[v] = ∞ for each vertex v in G
  Set distance[s] = 0
  for i from 1 to size(G.V) - 1 do
    updated = false
    for each edge (u, v) in G.E do
      if distance[u] + weight(u, v) < distance[v] then
        distance[v] = distance[u] + weight(u, v)
        updated = true
    if updated is false then
      break
  for each edge (u, v) in G.E do
    assert distance[v] <= distance[u] + weight(u, v)
```

Figure 1. Pseudocode for SCBF

B. *Shortest Path Faster Algorithm (SPFA)*

Similar to the short-circuiting Bellman-Ford algorithm, SPFA [34] starts by initializing an array to track the shortest distance from the starting node to all other node in the graph. It

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 8s
DOI: https://doi.org/10.17762/ijritcc.v11i8s.7172
***Article Received**: 20 April 2023 **Revised:** 08 June 2023 **Accepted:** 25 June 2023*

also maintains a queue of nodes that may need their shortest distance updated, initially containing just the source node.

The algorithm then enters a loop that continues as long as there are nodes in the queue. During each iteration, it removes a node from the queue and relaxes all of its outgoing edges, similar to the Bellman-Ford algorithm. If an edge relaxation results in an update to the shortest distance, and the destination node is not already in the queue, it adds the destination node to the queue.

The algorithm also maintains a Boolean array that tracks which nodes are currently in the queue, to prevent adding a node to the queue multiple times.

After the queue is empty, the algorithm checks for negative cycles as in the Bellman-Ford algorithm. The shortest distance array is then the output of the algorithm.

Here is the Pseudocode for the said Algorithm:

```
procedure SPFA(G, s)
  Initialize distance[] such that distance[v] = ∞ for each vertex v in G
  Initialize in_queue[] such that in_queue[v] = false for each vertex v in G
  Set distance[s] = 0
  Create an empty queue Q and add s into Q
  Set in_queue[s] = true
  while Q is not empty do
    Remove the first vertex u from Q and set in_queue[u] = false
    for each edge (u, v) in G.E do
      if distance[u] + weight(u, v) < distance[v] then
        distance[v] = distance[u] + weight(u, v)
        if in_queue[v] is false then
          Add v into Q and set in_queue[v] = true
  for each edge (u, v) in G.E do
    assert distance[v] <= distance[u] + weight(u, v)
```

Figure 2. Pseudocode for SPFA

The methodology employed in this study revolves around the creation and execution of Python functions that implement the Bellman Ford, Short-Circuiting Bellman Ford (SCBF), and the Shortest Path Faster Algorithm (SPFA). The analysis and comparison of these algorithms are based on both runtime and memory usage performance metrics.

C. Algorithm Implementation:

Each of the three algorithms - bellman_ford(), scbf(), and spfa() - are implemented as individual functions. These algorithms aim to calculate the shortest distance from a source node to all other nodes in a generated graph.

The bellman_ford() function relaxes the edges of the graph |V| - 1 times, where |V| represents the number of nodes in the graph, before performing a check for negative weight cycles.

The scbf() function also performs relaxation of edges, but introduces a short-circuiting technique that breaks the loop if no updates were made in the previous iteration. This potentially reduces the number of unnecessary iterations.

The spfa() function is similar to the bellman_ford() function, but it uses a queue-based approach to determine the order of node processing, providing a potential performance boost in certain scenarios.

D. Performance Analysis:

We wrote a functioned in our code called analyze_performance() function is used to calculate the time and memory usage for each algorithm. It records the start time and memory usage before the algorithm is run, and the end time and memory usage after the algorithm has finished executing. The difference between the start and end values provides the total time and memory used by each algorithm.

E. Graph Generation and Visualization:

In our Python implementation, we have the generate_graph() function employed to generate a random directed graph with a specified number of nodes. The draw_graphs() function then visualizes this generated graph.

F. Profiling and Comparative Analysis:

We wrote the profile() function that wraps around each algorithm function, measuring the runtime and memory usage. It uses the time.time() method for timing and the memory_usage function from the memory_profiler package for memory profiling. After each algorithm is profiled, the results are visualized using bar and line plots to provide a visual comparison of the time and memory performance of the three algorithms.

Overall, this methodology provides a detailed and comparative analysis of the Bellman Ford, SCBF, and SPFA algorithms in terms of both runtime and memory usage. The process highlights the strengths and weaknesses of each algorithm, helping in the selection of the most efficient algorithm for specific use-cases.

VII. SPACE TIME COMPLEXITY

A. Short-Circuiting Bellman-Ford Complexity

Time Complexity: The worst-case time complexity is:

$$O(VE) \qquad (1)$$

because in the worst case, the algorithm may still need to perform V-1 iterations over all edges. However, if the graph is such that the shortest paths can be determined in fewer iterations, then the algorithm could potentially finish faster.

Space Complexity: The space complexity is:

$$O(V) \qquad (2)$$

because we need to store the shortest distance from the source to each vertex.

B. *Shortest Path Faster Algorithm (SPFA) Complexity*

Time Complexity: The average-case time complexity of SPFA can be much better than Bellman-Ford and is approximately given by the below equation:

$$O(E) \tag{3}$$

for many graphs, especially those that are sparse or have small-world properties. However, in the worst-case scenario (e.g., for a graph that resembles a linked list), the time complexity can be ascertained from the below O notation [35]:

$$O(VE) \tag{4}$$

Space Complexity: The space complexity of SPFA is also as below:

$$O(V) \tag{5}$$

because we need to store the shortest distance from the source to each vertex, as well as whether each vertex is in the queue.

VIII. COMPARISON

In this section, we compare the Short-Circuiting Bellman-Ford (SCBF) and the Shortest Path Faster Algorithm (SPFA) to the original Bellman-Ford algorithm.

A. *Short-Circuiting Bellman-Ford*

Pros:

Potentially Faster: This variation can finish faster than the original Bellman-Ford algorithm if the shortest paths can be found in fewer than V-1 iterations. In such cases, it does not need to go through all the iterations, thereby saving time.

Cons:

Worst-Case Performance: In the worst case, where the graph is structured such that, the shortest paths require all V-1 iterations, this variation provides no speedup over the original Bellman-Ford algorithm.

B. *Shortest Path Faster Algorithm (SPFA)*

Pros:

Average-Case Performance: In many graphs, particularly those that are sparse or exhibit small-world properties, the SPFA can significantly outperform the original Bellman-Ford algorithm in terms of time complexity, often closer to O(E), which is an improvement over the O(VE) worst-case time complexity of the Bellman-Ford algorithm.

Cons:

Worst-Case Performance: In the worst-case scenario (e.g., for a graph that resembles a linked list), the time complexity of SPFA can still be O(VE), same as the original Bellman-Ford algorithm. However, this worst-case scenario is relatively rare in practice.

Memory Usage: SPFA requires maintaining an additional queue of vertices to process, which may increase its memory usage compared to the original Bellman-Ford algorithm.

The best choice of algorithm can depend on the specific characteristics of your input data and your specific use case. For some graphs and situations, the original Bellman-Ford algorithm might still be the most suitable choice.

IX. PERFORMANCE ANALYSIS

As mentioned in the methodology section of this paper, we used Python implementation for all the algorithms in the subject along with the libraries for profiling memory usage and recording the runtime. We also implemented a random graph generating function which would generate the graphs for testing. We plotted those graphs as well in-order to give a visual depiction of the network which is being used for running the algorithms to find the shortest path. We recorded the runtime and memory usage during each run and recorded the same in the form of a graph. We used this method multiple times to arrive at an effective comparison. At first, we started with generating a graph of 6 nodes which is depicted in the Figure 3 below.

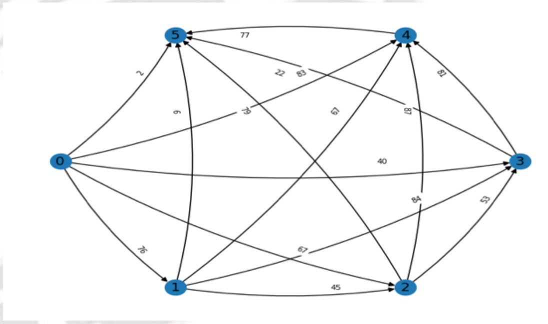
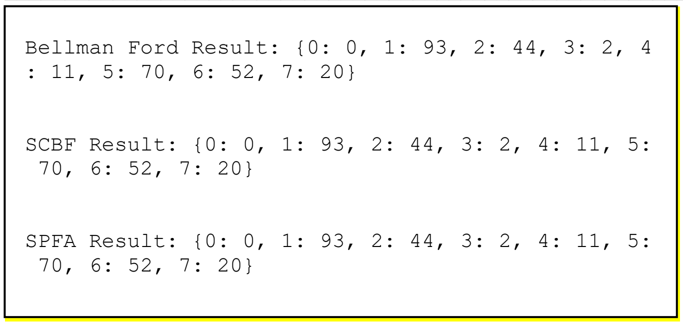


Figure 3. 6 Node Graph (Test 1)

The Results of the Algorithms obtained for the shortest path are mentioned below:

***Article Received**: 20 April 2023 **Revised:** 08 June 2023 **Accepted:** 25 June 2023*



```
Bellman Ford Result: {0: 0, 1: 93, 2: 44, 3: 2, 4
: 11, 5: 70, 6: 52, 7: 20}


SCBF Result: {0: 0, 1: 93, 2: 44, 3: 2, 4: 11, 5:
 70, 6: 52, 7: 20}


SPFA Result: {0: 0, 1: 93, 2: 44, 3: 2, 4: 11, 5:
 70, 6: 52, 7: 20}
```

Figure 4. Shortest path resultd for Test 1

When we plotted the running time for all three algorithms along with the memory usage, we found that the SCBF has a better running time than the original Bellman Ford while SPFA recorded the best running time. On the parameter of memory usage, we had Bellman Ford as the best performer followed by SCBF and SPFA. Though the difference in memory usage was minuscule. The graphs for this test are presented below as Figure 5.

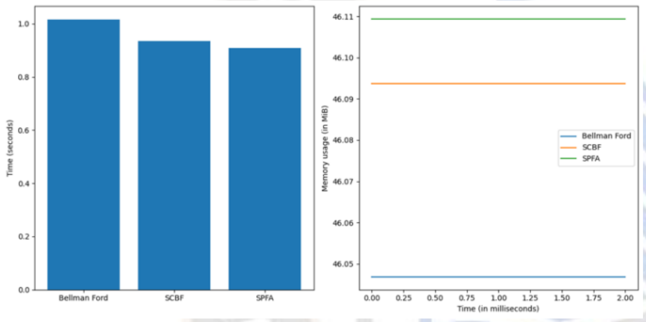


Figure 5. Time & Space performance for Test 1

After the first test, we increased the number of nodes in the graph and tried to run the algorithms for a randomly generated graph of 8 nodes as depicted in the Figure 6 below:

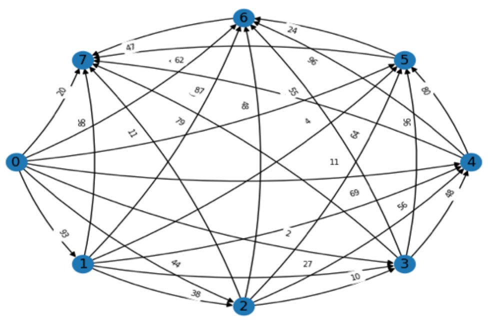
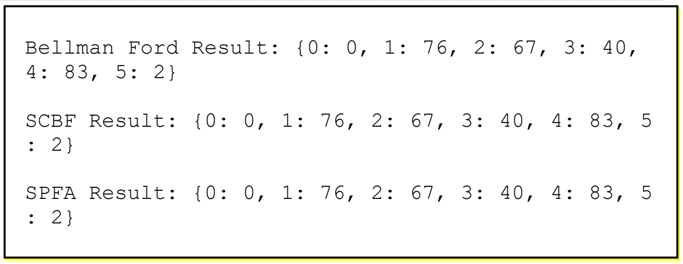


Figure 6. 6 Node Graph (Test 2

The below are the details for the shortest path results obtained by running all the three algorithms. We printed the results just to make sure that the algorithms are returning the same result for the respective graph.



```
Bellman Ford Result: {0: 0, 1: 76, 2: 67, 3: 40,
4: 83, 5: 2}

SCBF Result: {0: 0, 1: 76, 2: 67, 3: 40, 4: 83, 5
: 2}

SPFA Result: {0: 0, 1: 76, 2: 67, 3: 40, 4: 83, 5
: 2}
```

Figure 7. Shortest path result for Test 2

In this attempt, we had a clear Running time improvement shown by both SCBF and SFPA. On the Space front, we had SCBF as the most efficient algorithm while Bellman Ford and SPFA were a marginally higher in memory utilization. The Comparison is shown in Figure 8.

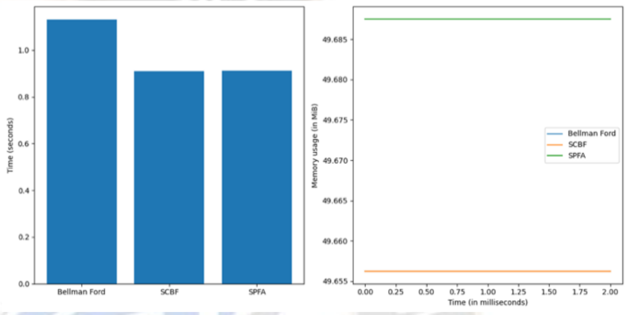


Figure 8. Time & Space performance for Test 2

Post this we again generated a graph of 10 nodes and ran the tests on the same. The graph is shown in Figure 9 below. It has 10 nodes which are interconnected to each other with vertices having random weights, shown by numbers in the figure. We used our graph generator function to generate this graph for conducting our expirement.

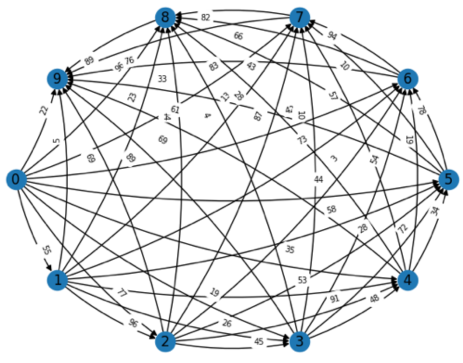


Figure 9. 10 Node Graph (Test 3)

We ran the Algorithms on this graph as well to record the results and compare their space time usage. The results for the algorithm comparison are shown in Figure 10 below:

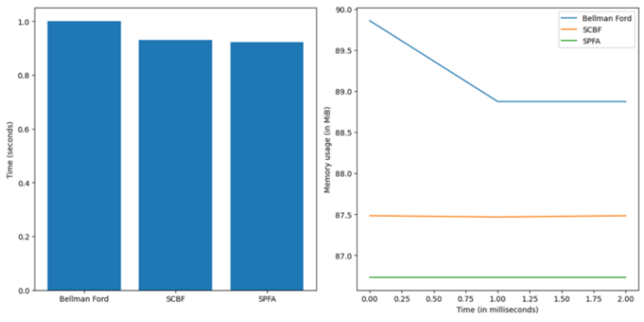


Figure 10. Time & Space performance for Test 3

We again see good performance improvement in terms of running time by both SCBF and SPFA. We also noticed space efficiency on using both new implementations ie, SCBF and SPFA.

We conducted another test with a graph of 15 nodes this time as shown in the Figure 11 below:

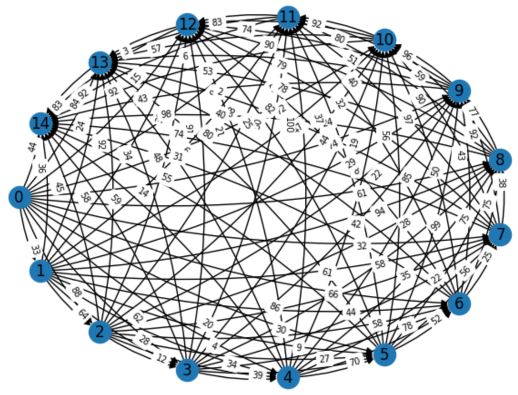


Figure 11. 15 Node Graph (Test 4)

Tested and printed the shortest path output as depicted in Figure12, just to ensure that the algorithms are giving same output for the given graph at each respective run.

```
Bellman Ford Result: {0: 0, 1: 33, 2: 88, 3: 61,
4: 20, 5: 63, 6: 61, 7: 42, 8: 8, 9: 24, 10: 39,
11: 42, 12: 15, 13: 10, 14: 44}


SCBF Result: {0: 0, 1: 33, 2: 88, 3: 61, 4: 20, 5
: 63, 6: 61, 7: 42, 8: 8, 9: 24, 10: 39, 11: 42,
12: 15, 13: 10, 14: 44}


SPFA Result: {0: 0, 1: 33, 2: 88, 3: 61, 4: 20, 5
: 63, 6: 61, 7: 42, 8: 8, 9: 24, 10: 39, 11: 42,
12: 15, 13: 10, 14: 44}
```

Figure 12. Shortest Path results (Test 4)

The comparison for Space and Time performance for both the new implementations against the traditional Bellman Ford is given in the test 4 Graphs below in Figure 13.

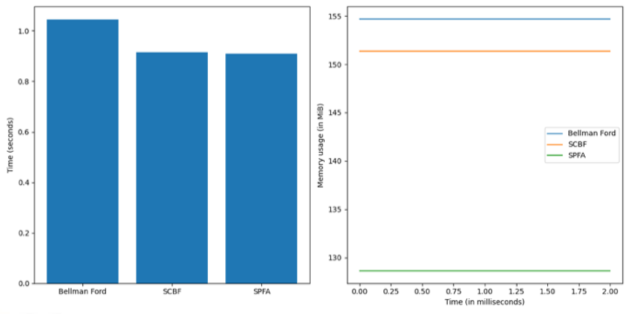


Figure 13. Time & Space performance for Test 4

Again, we saw that our SCBF and SPFA are clear winners in both Run time and memory utilization front.

We continued our tests and recorded multiple readings to plot the performance of these algorithms at various graph sizes and tested the algorithms for random graphs of sizes up-to 1000 nodes. The summary of the recorded results in presented in the next section as the Runtime performance and the Memory Utilization Efficiency.

A. Runtime Performance:

As the number of nodes increases, the Bellman-Ford algorithm (BF) seems to demonstrate a non-linear increase in its runtime. This trend is expected, as BF has a time complexity of O(V*E), which means the runtime can grow rapidly with larger inputs. However, the data shows a dip in performance for larger node counts (50, 80, 100), which might be attributed to system-level optimizations or hardware-related factors.

TABLE II. RUNTIME PERFORMANCE

Graph Nodes	BF Runtime	SCBF Runtime	SPFA Runtime
2	1.27	0.91	0.92
4	0.99	0.9	0.91
6	1.02	0.93	0.9
8	1.04	0.92	0.94
10	0.93	0.91	0.9
12	1	0.91	0.9
14	0.97	0.93	0.91
16	1.07	0.97	0.92
18	1.04	0.91	0.9
20	1	0.9	0.9
22	1.02	0.94	0.9
24	1.02	0.94	0.91
26	0.99	0.91	0.94
28	1.03	0.91	0.9
30	1.04	0.92	0.9

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 8s
DOI: https://doi.org/10.17762/ijritcc.v11i8s.7172
***Article Received**: 20 April 2023 **Revised:** 08 June 2023 **Accepted:** 25 June 2023*

50	0.43	0.9	0.9
80	0.2	0.9	0.92
100	0.27	0.95	0.98
200	0.86	0.36	0.32
500	8.94	0.18	0.34
1000	70.1	1.11	1.03

We took these values and plotted a graph using the python library to represent our research results as below:

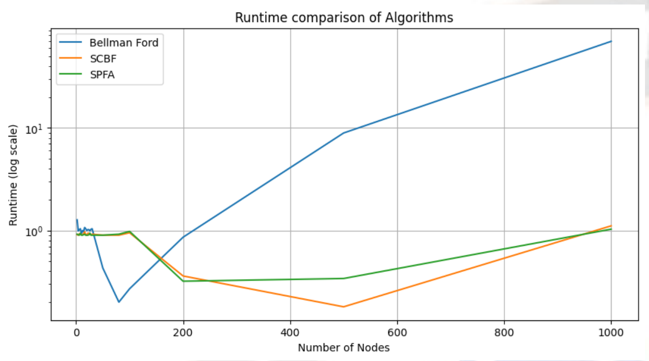


Figure 14. RunTime performance Summary

Figure 14 above shows a graph with the summary tests results for multiple runs of these algorithms on graphs up to node size 1000. The Shortest Path Faster Algorithm (SPFA) and the Small Cycle Bellman-Ford (SCBF) algorithm demonstrate relatively consistent runtimes across different numbers of nodes. It is noticeable that the runtime of SCBF is typically slightly lower than that of SPFA. However, both algorithms significantly outperform the BF for larger graphs (200, 500, 1000 nodes).

B. Memory Performance:

The memory usage of all three algorithms doesn't follow a clear trend based on the results recorded. However, it appears that all three algorithms have comparable memory usage, with no algorithm consistently using less memory than the others. Again, the data shows some peaks for larger node counts, which might be due to factors such as system-level caching strategies or hardware characteristics.

TABLE III. SPACE PERFORMANCE

Graph Nodes	BF Memory	SCBF Memory	SPFA Memory
2	47.85	47.64	47.76
4	93.17	86.48	84.4
6	51.07	51.03	50.26
8	155.09	154.7	154.7
10	96.23	96.23	96.25
12	98.07	95.5	95.5
14	50.53	50.53	50.57
16	84.45	63.87	34.06
18	52.21	52.29	52.29
20	87.65	87.64	87.64
22	57.26	53.15	53.17
24	54.34	54.4	54.4
26	130.56	129.2	125.29
28	85.57	85.5	85.51
30	54.43	54.62	54.67
50	88.015	87.97	86.75
80	86.87	86.79	78.17
100	58.31	60.18	58.06
200	153.89	154	153.04
500	69.03	69.93	70.8
1000	97.21	68.12	68.22

Notably, SPFA and SCBF seem to use slightly less memory for larger graphs (1000 nodes), providing a minor advantage over BF in terms of memory usage s shown in the Figure 15 below.

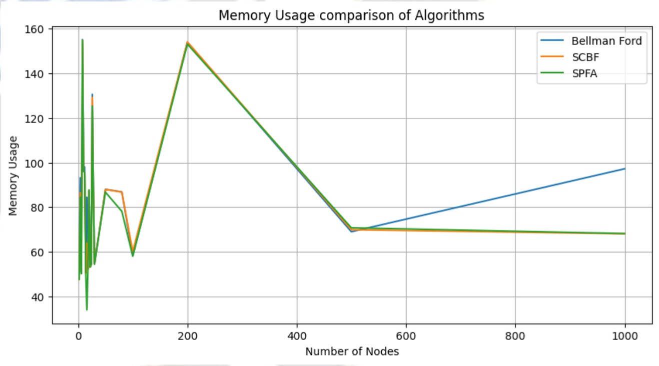


Figure 15. RunTime performance Summary

The SPFA and SCBF algorithms generally outperform the BF algorithm in both runtime and memory usage, especially for larger graphs. However, the specific benefits can vary depending on factors such as the exact structure of the graph and system-level characteristics.

While the SCBF shows slightly better runtime performance than SPFA, the difference is small and may not be significant in many cases.

The choice between SPFA and SCBF could be influenced by other considerations, such as the ease of implementation, the prevalence of small cycles in the graph, and the specifics of the use case.

In summary, this analysis indicates that both SPFA and SCBF algorithms are superior to the traditional Bellman-Ford algorithm in terms of performance efficiency. The choice between SPFA and SCBF should be made based on additional factors such as implementation complexity and graph characteristics. It is recommended to conduct further tests on a wider range of graph structures and sizes to confirm these findings and identify any additional considerations.

X. CONCLUSION

In this study, we investigated the performance of three routing algorithms: the Bellman-Ford algorithm, the Small Cycle Bellman-Ford (SCBF) algorithm, and the Shortest Path Faster Algorithm (SPFA). Our goal was to ascertain which of these approaches was most efficient in terms of both runtime and memory usage, with a particular focus on applications involving large graph structures.

Our findings indicate that both SPFA and SCBF significantly outperform the traditional Bellman-Ford algorithm, especially as the size of the graph increases. The runtime of the Bellman-Ford algorithm increases non-linearly with the size of the graph, while SPFA and SCBF display relatively consistent performance across a range of graph sizes. In terms of memory usage, all three algorithms exhibit similar consumption, with occasional advantages for SPFA and SCBF for larger graphs.

These results suggest that developers and engineers seeking to optimize the performance of their network routing algorithms should strongly consider adopting either SPFA or SCBF over the traditional Bellman-Ford algorithm. The decision between SPFA and SCBF could be based on additional factors such as implementation complexity, the specific characteristics of the graph (such as the prevalence of small cycles), and the requirements of the particular use case.

It is important to note that our study was conducted under specific conditions and with certain assumptions about the graph structure. Therefore, we recommend further studies to confirm these findings and explore other potential influencing factors. Furthermore, while this paper focuses on runtime and memory usage, other important considerations such as scalability, adaptability, and reliability of the algorithms should also be considered in real-world applications.

In conclusion, this study provides valuable insights into the performance of SPFA and SCBF algorithms compared to the traditional Bellman-Ford algorithm. The findings underscore the importance of algorithm selection in network routing and offer a solid foundation for further research in this field. The challenge for future work lies in extending these results and exploring more diverse scenarios and complex graph structures, to provide an even more comprehensive understanding of the performance of these algorithms.

REFERENCES

[1] Medhi, D., & Ramasamy, K. (2017). Network routing: algorithms, protocols, and architectures. Morgan Kaufmann.

[2] Gu, Y., & Grossman, R. L. (2007). UDT: UDP-based data transfer for high-speed wide area networks. Computer Networks, 51(7), 1777-1799.

[3] Ahuja, R. K., Mehlhorn, K., Orlin, J., & Tarjan, R. E. (1990). Faster algorithms for the shortest path problem. Journal of the ACM (JACM), 37(2), 213-223.

[4] Brandes, Ulrik. "A faster algorithm for betweenness centrality." Journal of mathematical sociology 25.2 (2001): 163-177.

[5] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[6] Sungwon Jung and S. Pramanik, "An efficient path computation model for hierarchically structured topographical road maps," in IEEE Transactions on Knowledge and Data Engineering, vol. 14, no. 5, pp. 1029-1046, Sept.-Oct. 2002, doi: 10.1109/TKDE.2002.1033772.I.

[7] Abraham, C. Gavoille, A. V. Goldberg and D. Malkhi, "Routing in Networks with Low Doubling Dimension," 26th IEEE International Conference on Distributed Computing Systems (ICDCS'06), Lisboa, Portugal, 2006, pp. 75-75, doi: 10.1109/ICDCS.2006.72.

[8] N. Futamura, R. Sangireddy, S. Aluru and A. K. Somani, "Scalable, memory efficient, high-speed lookup and update algorithms for IP routing," Proceedings. 12th International Conference on Computer Communications and Networks (IEEE Cat. No.03EX712), Dallas, TX, USA, 2003, pp. 257-263, doi: 10.1109/ICCCN.2003.1284179.

[9] Zheng Wang and Jon Crowcroft. 1992. Analysis of shortest-path routing algorithms in a dynamic network environment. SIGCOMM Comput. Commun. Rev. 22, 2 (April 1992), 63–71. https://doi.org/10.1145/141800.141805

[10] Madduri, K., Bader, D. A., Berry, J. W., & Crobak, J. R. (2007, January). An experimental study of a parallel shortest path algorithm for solving large-scale graph instances. In 2007 Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX) (pp. 23-35). Society for Industrial and Applied Mathematics.

[11] Matti Virtanen, Jan de Vries, Thomas Müller, Daniel Müller, Giovanni Rossi. Machine Learning for Intelligent Feedback Generation in Online Courses . Kuwait Journal of Machine Learning, 2(2). Retrieved from http://kuwaitjournals.com/index.php/kjml/article/view/188

[12] Pettie, S. (2004). A new approach to all-pairs shortest paths on real-weighted graphs. Theoretical Computer Science, 312(1), 47-74.

[13] Dijkstra, E. W. (2022). A note on two problems in connexion with graphs. In Edsger Wybe Dijkstra: His Life, Work, and Legacy (pp. 287-290).

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 8s
DOI: https://doi.org/10.17762/ijritcc.v11i8s.7172
***Article Received**: 20 April 2023 **Revised:** 08 June 2023 **Accepted:** 25 June 2023*

[14] Fredman, M. L., & Tarjan, R. E. (1987). Fibonacci heaps and their uses in improved network optimization algorithms. Journal of the ACM (JACM), 34(3), 596-615.

[15] Dinitz, Y., & Itzhak, R. (2017). Hybrid Bellman–Ford–Dijkstra algorithm. Journal of DiscreteAlgorithms, 42, 35–44. doi:10.1016/j.jda.2017.01.001.

[16] Lacorte, A. M., & Chavez, E. P. (2018). Analysis on the Use of A* and Dijkstra's Algorithms for Intelligent School Transport Route Optimization System. Proceedings of the 4th International Conference on Human-Computer Interaction and User Experience in Indonesia, CHIuXiD '18 -CHIuXiD '18. doi:10.1145/3205946.3205948

[17] Ahammad, D. S. H. ., & Yathiraju, D. . (2021). Maternity Risk Prediction Using IOT Module with Wearable Sensor and Deep Learning Based Feature Extraction and Classification Technique. Research Journal of Computer Systems and Engineering, 2(1), 40:45. Retrieved from https://technicaljournals.org/RJCSE/index.php/journal/article/view/19

[18] Luca Ferrari, Deep Learning Techniques for Natural Language Translation , Machine Learning Applications Conference Proceedings, Vol 2 2022.

[19] Abbas, Q., Hussain, Q., Zia, T. & Mansoor, A. (2018). Reduced Solution Set Shortest Path Problem: Capton Algoritm With Special Reference To Dijkstra's Algorithm. Malaysian Journal of Computer Science, [S.l.], v. 31, n. 3, p. 175-187, july 2018. ISSN 0127-9084

[20] Sapundzhi, F. I., Popstoilov, M. S. (2018). Optimization algorithms for finding the shortest paths. Bulgarian Chemical Communications, Volume 50, Special Issue B, (pp. 115 – 120)

[21] Geetha M., Karegowda, A. G. ., Nandeesha, & Nagaraj B. V. (2023). Classification of Sentinel 2 Images using Customized Convolution Neural Networks. International Journal of Intelligent Systems and Applications in Engineering, 11(1s), 136–142. Retrieved from https://ijisae.org/index.php/IJISAE/article/view/2485

[22] Oyola, A., Romero, D. G., & Vintimilla, B. X. (2017). A Dijkstra-Based Algorithm for Selecting the Shortest-Safe Evacuation Routes in Dynamic Environments (SSER). Lecture Notes in Computer Science, 131–135. doi:10.1007/978-3-319-60042-0_15.

[23] Singh, J.B., Tripathi, R.C. (2018). Investigation of Bellman–Ford Algorithm, Dijkstra's Algorithm for suitability of SPP. IJEDR | Volume 6, Issue 1 | ISSN: 2321-9939

[24] Lacorte, A. M., & Chavez, E. P. (2018). Analysis on the Use of A* and Dijkstra's Algorithms for Intelligent School Transport Route Optimization System. Proceedings of the 4th International Conference on Human-Computer Interaction and User Experience in Indonesia, CHIuXiD '18 - CHIuXiD '18. doi:10.1145/3205946.3205948

[25]] Chan, S., Adnan, N., Sukri, S.S., & Zainon, W.M. (2016). An experiment on the performance of shortest path algorithm.Knowledge Management International Conference (KMICe) 2016, 29 – 30 August 2016, Chiang Mai, Thailand

[26] M. C. Saxena and P. Bajaj, "Evolution of Wide Area network from Circuit Switched to Digital Software defined Network," 2021 International Conference on Technological Advancements and Innovations (ICTAI), Tashkent, Uzbekistan, 2021, pp. 351-357, doi: 10.1109/ICTAI53825.2021.9673201.

[27] Samah W.G. AbuSalim et al 2020 IOP Conf. Ser.: Mater. Sci. Eng. 917 012077

[28] Wang, X. Z. (2018, September). The comparison of three algorithms in shortest path issue. In Journal of Physics: Conference Series (Vol. 1087, No. 2, p. 022011). IOP Publishing.

[29] Zhu, Z., Zhang, Z., Xhonneux, L. P., & Tang, J. (2021). Neural bellman-ford networks: A general graph neural network framework for link prediction. Advances in Neural Information Processing Systems, 34, 29476-29490.

[30] Cherkassky, B. V., Goldberg, A. V., & Radzik, T. (1996). Shortest paths algorithms: Theory and experimental evaluation. Mathematical programming, 73(2), 129-174.

[31] Duan, F. (1994). A faster algorithm for shortest-path—SPFA. Journal of Southwest Jiaotong University, 29(2), 207-212.

[32] Marina, M. K., & Das, S. R. (2002). Ad hoc on-demand multipath distance vector routing. ACM SIGMOBILE Mobile Computing and Communications Review, 6(3), 92-93.

[33] Hedrick, C. L. (1988). Routing information protocol (No. rfc1058).

[34] Yen, J. Y. (1971). Finding the k shortest loopless paths in a network. management Science, 17(11), 712-716.

[35] Zhang, W., Chen, H., Jiang, C., & Zhu, L. (2013, August). Improvement and experimental evaluation bellman-ford algorithm. In 2013 International Conference on Advanced ICT and Education (ICAICTE-13) (pp. 138-141). Atlantis Press.

[36] Zhang, H., Liu, X., & Xiang, L. (2019, August). Improved SPFA algorithm based on Cell-like P system. In 2019 10th International Conference on Information Technology in Medicine and Education (ITME) (pp. 679-683). IEEE.

[37] Mohit Saxena. (2023). Bellman_Ford-Enhanced-SCBF-SPFA-Comparison [Source code]. GitHub. https://github.com/m22aie240/Bellman_Ford-Enhanced-SCBF-SPFA-Comparison

[38] Short-Circuiting Bellman-Ford (SCBF) Zhou, X. (2014). An Improved SPFA Algorithm for Single-Source Shortest Path Problem Using Forward Star Data Structure. International Journal of Managing Information Technology (IJMIT) Vol, 6.

[39] Chivers, I., Sleightholme, J., Chivers, I., & Sleightholme, J. (2015). An introduction to Algorithms and the Big O Notation. Introduction to Programming with Fortran: With Coverage of Fortran 90, 95, 2003, 2008 and 77, 359-364.