Degree Based Search: A Novel Graph Traversal Algorithm Using Degree Based Priority Queues

Shyma P V, Sanil Shanker K P
Department of Information Technology, Kannur University, India

***Abstract*—This paper introduces a novel graph traversal algorithm, Degree Based Search, which leverages degree-based ordering and priority queues to efficiently identify shortest paths in complex graph structures. Our method prioritizes nodes based on their degrees, enhancing exploration of related components and offering flexibility in diverse scenarios. Comparative analysis demonstrates superior performance of Degree Based Search in accelerating path discovery compared to traditional methods like Breadth First Search and Depth First Search. This approach improves exploration by focusing on related components. Using a priority queue ensures optimal node selection; the method iteratively chooses nodes with the highest or lowest degree. Based on this concept, we classify our approach into two distinct algorithms: the Ascendant Node First Search, which prioritizes nodes with the highest degree, and the Descent Node First Search , which prioritizes nodes with the lowest degree. This methodology offers diversity and flexibility in graph exploration, accommodating various scenarios and maximizing efficiency in navigating complex graph structures. The study demonstrates the Degree based Searching algorithm's efficacy in accelerating path discovery within graphs. Experimental validation illustrates its proficiency in solving intricate tasks like detecting communities in Facebook networks. Moreover, its versatility shines across diverse domains, from autonomous driving to warehouse robotics and biological systems. This algorithm emerges as a potent tool for graph analysis, efficiently traversing graphs and significantly enhancing performance. Its wide applicability unlocks novel possibilities in various scenarios, advancing graph-related applications.**

Keywords*—Graph traversal; degree based search algorithm; ascendant node; ascendant node first searching algorithm; descent node; descent node first searching algorithm

I. Introduction

In today's interconnected world, the analysis and comprehension of complex systems have become pivotal across various domains including social interactions, communication networks, and engineered systems such as the power grid and the Internet. These systems can be effectively modeled and analyzed using graph theory, which provides a powerful framework for representing and understanding relationships between entities. Graphs, consisting of vertices and edges, serve as a fundamental abstraction for representing diverse real-world phenomena. Graph traversal, the systematic exploration of vertices and edges within a graph, lies at the heart of many graph analysis tasks. Traditionally, algorithms such as Breadth First Search (BFS) and Depth First Search (DFS) have been extensively utilized for traversing graphs and solving fundamental problems like identifying spanning trees, finding shortest paths, and detecting strongly connected components. However, these traditional methods have inherent limitations that hinder their effectiveness in modern applications [1].

Breadth First Search explores the graph level by level, starting from a source vertex, and is particularly useful for finding shortest paths due to its systematic approach. However, it consumes significant memory and may not be suitable for large-scale graphs with millions of nodes. On the other hand, DFS explores the graph depth-wise, often employing recursion or stack data structures. While DFS is memory-efficient, it lacks the ability to guarantee the shortest path and may encounter issues such as stack overflow and infinite loops, especially in graphs with cycles.Moreover, neither BFS nor DFS adequately address the consideration of edge weights, limiting their applicability in scenarios where edge weights play a crucial role [2]. Additionally, these traditional algorithms may fail to cover all connected components of a graph, potentially missing vital information, and may become stuck in infinite loops, further complicating their use in dynamic or cyclic graph structures [4]. To overcome these challenges and develop more efficient traversal algorithms, researchers have focused on enhancing memory optimization, scalability for large networks, consideration of edge weights, and the ability to handle various graph structures effectively. The development of novel traversal methodologies that address these limitations is imperative to advance graph analysis techniques and address the growing demands of modern applications.

The effectiveness of unique priority queue-based degree-based graph search algorithm must be validated using this method. We seek to demonstrate the computational benefits of node-degree prioritised exploration by contrasting this algorithm with state-of-the-art optimisation techniques as well as classic graph search algorithms such as Depth-First Search (DFS) and Breadth-First Search (BFS) [3]. By leveraging priority queue-based methods and prioritizing nodes based on their degrees, the Degree based Search algorithm achieves increased efficacy and efficiency in graph traversal tasks. Furthermore, its dynamic selection mechanism enables adaptability to changing graph topologies, enhancing its robustness and versatility across diverse application domains. Through a comprehensive comparison with traditional algorithms and real-world applications, we demonstrate the effectiveness and scalability of the Degree based Search algorithm in addressing the fundamental challenges of graph traversal.

II. Related Works

It is becoming more and more crucial to analyse and comprehend social interaction data, relational data in general, complex engineered systems like the power grid and the Internet, communication data like email and phone networks, and biological systems using graph abstractions. These application fields frequently encounter graph-theoretic issues including

identifying and rating significant entities, seeing unusual patterns or rapid changes in networks, locating strongly connected clusters of entities, and others. For issues like discovering spanning trees, shortest paths, biconnected components, matching, and flow-based computations in these graphs, traditional techniques are frequently used as solutions. A traversal is a methodical exploration of each vertex and edge in a graph. Graph traversals, such as Breadth First Search and Depth First Search provide foundation for much higher-level graph analysis approaches and is the basic primitive for graph analytics [1]. Breadth First Search is a common graph analysis technique that examines every vertex in all levels of a graph starting from a source vertex [2]. The shortest path between vertices can be found using Breadth First Search as it always finds optimal solution. As the Breadth First Search examines level by level, it is impossible to locate a pointless or ineffective path [4]. The Depth First Search method starts at the root node and chooses an edge that originates from the most recently visited vertex that has unexplored edges [5]. In Depth First Search, a stack is used to hold a collection of old vertices with possibly unexplored edges and it's intricacy depends on the number of paths. Depth First Search has a flaw that cannot check for duplicate nodes and cannot guarantee the shortest path [6]. Depth First Search is difficult to apply when the graph is infinite while there are cycles within the graph [4]. The primary distinction between BFS and DFS is the order in which they are traversed; the former is in horizontal order and the latter is in vertical order. BFS is preferable for identifying components closer to the root, whereas DFS is preferred for locating elements deeper in the ground. In the worst-case scenario, however, both algorithms would take the same time O (V+E) because they visit all nodes once [7].

Although BFS ensures that the shortest path will always be found, it uses too much memory and is not suitable for large-scale graphs. Furthermore, it could be ineffective for networks with millions of nodes due to its thorough investigation of nearby nodes [8]. DFS, on the other hand, relies on recursion or stack data structures, which makes it susceptible to stack overflow issues and fails to guarantee the shortest path. Furthermore, DFS may not cover all connected components of the network, thereby missing important information, and it may become stuck in infinite loops, particularly in the presence of cycles [9], [10]. Additionally, neither algorithm takes edge weights into account, which limits its usefulness in situations when edge weights are important. In order to overcome these drawbacks and create an even better traversal algorithm, memory optimisation, scalability for big networks, edge weight consideration, strong termination conditions to avoid infinite loops, and adaptability to various graph structures should be prioritised [11]. Overcoming these obstacles can lead to a novel traversal method that provides enhanced scalability, performance, and versatility for a range of graph traversal problems in both practical and research fields.

The primary objective is to address the fundamental task of searching algorithm by utilising the concept of degree of a node for efficient graph traversal. This work presents a novel traversal methodology called the Degree based Search algorithm, which has several advantages over conventional graph traversal techniques like BFS and DFS. Through the integration of priority queue-based methods, the algorithm attains increased efficacy and efficiency when examining graph structures. Its capacity to rank nodes according to degrees is one of its main benefits, as it enables a more methodical and efficient traversal procedure. By ensuring that the algorithm concentrates on nodes with stronger connectivity, this prioritisation technique speeds up path identification and the investigation of pertinent graph components. Moreover, the Degree based Search algorithm may dynamically adjust to the graph's shifting topology while traversing thanks to the use of a priority queue. Identifying key pathways and components in an efficient manner, the algorithm traverses the network by repeatedly choosing nodes with the highest or lowest degrees from the priority queue. The algorithm's robustness and versatility are increased by this dynamic selection process, which makes it appropriate for a variety of graph types and applications. Additionally, the Degree based Search algorithm performs exceptionally well in memory management because it uses effective data structures like priority queues to reduce the memory footprint. This feature is especially helpful for large-scale graphs with millions of nodes, where performance and scalability are dependent on memory efficiency. In Table I, a comparison of the three algorithms is presented.

TABLE I. COMPARISON OF BFS, DFS AND DBS

Strategy	Complete	Optimal	Time complexity	Space complexity
BFS	Yes	Yes	O(b^d), where b and d are branching factor and depth respectivly	O(b^d)
DFS	No	No	O(b^m), where b and m are branching factor and maximum depth respectivly	O(bm)
DBS	Yes	Yes	O($(V+E)logV$), where m and n are number of edges and vertices respectivly	O($V+E$)

III. METHODOLOGY

This work explores the combination of degree-based traversal algorithms and priority queues, with a particular application to the exploration of related elements in graphs. Priority queues are an effective tool for node selection strategy optimisation because of their ability to manage items based on predetermined keys. Our suggested algorithms drive research towards nodes that are most likely to produce important information by prioritising high- or low-degree nodes according to their degrees. The degree-based traversal algorithms considered here demonstrate two different strategies: one giving priority to nodes with the highest degree, while the other prioritises nodes with the lowest degree. Selecting one of these approaches offers a framework that is adaptable to the particular features of the graph being studied. Graph exploration is given a new dimension by integrating degree-based traversal algorithms with priority queues, which makes it possible to find as many connected components as possible in an efficient manner. By methodically extracting and processing nodes, the algorithmic strategy ensures a thorough analysis of the graph while taking the connectivity structure into account. The purpose of this research is to ascertain the significance of the proposed techniques by means of an extensive empirical review. We compare degree-based traversal algorithms with traditional methods in

order to illustrate the advantages of using priority queues in the context of connected components. The results of the study may have implications for a wide range of applications, such as social network analysis and routing optimisation in communication networks.

A. Degree-Based Traversal Algorithms with Priority Queue

In order to track visited vertices, the method first initialises an empty priority queue Q and an array visited. The search is started by inserting the source vertex s into the priority queue. Although the priority queue is not empty, the algorithm selects the vertex u with the lowest degree from the queue and, if it hasn't been visited previously, marks it as visited. Next, the method investigates neighbouring vertices of u, adding each unexplored neighbouring vertex, according to its degree, to the priority queue. The graph search result is represented by the collection of visited vertices, and the process continues until the priority queue is empty.

Based on the ascending or descending order of node degrees within the priority queue, the proposed approach dynamically modifies its exploration method.

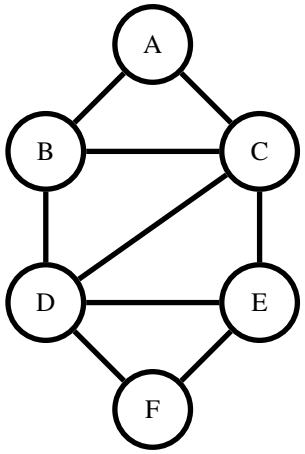

Fig. 1. Graph for illustration of the algorithm.

The algorithm begins with an initial state where no vertices have been visited, and the priority queue contains all vertices along with their respective degrees. In Fig. 1, the first step, vertex A is selected as the source vertex, marking it as visited and updating the priority queue by removing A. In the second step, the algorithm selects vertex F, the adjacent vertex to A with the highest degree, adds it to the visited set, and updates the priority queue accordingly. Next, vertex B, the highest-degree adjacent vertex to the current visited set, is selected and added to the visited set. This process continues with the selection of vertex E, followed by vertex D, each time updating the visited set and priority queue. Finally, vertex C is selected, completing the traversal with all vertices visited. The final state shows all vertices A, F, B, E, D, C in the visited set and an empty priority queue, demonstrating the efficient traversal based on the highest-degree selection criterion.

Let $G = (V, E)$ be an undirected graph. Let Q be a priority queue with keys based on node degrees and $visited$ be an empty set to keep track of visited nodes.

```
Algorithm ANFS(Graph G, Node s):
    Initialize an empty priority queue Q
    Initialize an array visited[] to keep track
    of visited vertices
    PQ_insert(Q, s)
    Insert the source vertex s into the
    priority queue
    while Q is not empty:
        u = PQ_extractMin(Q)
        Extract the vertex with the minimum
        degree
        if visited[u] is false:
            visited[u] = true [Mark u as visited]
            for each vertex v adjacent to u:
                if visited[v] is false:
                    PQ_insert(Q, v)
                    Insert v into the priority
                    queue
    return visited
```

```
Algorithm DNFS(Graph G, Node s):
    Initialize an empty priority queue Q
    Initialize an array visited[] to keep track
    of visited vertices
    PQ_insert(Q, s)
        Insert the source vertex s into the
        priority queue
        while Q is not empty:
         u = PQ_extractMax(Q)  Extract the
         vertex with the maximum degree
        if visited[u] is false:
            visited[u] = true  // Mark
            u as visited
            for each vertex v adjacent to u:
                if visited[v is false:
                    PQ_insert(Q, v)
                    Insert v into the priority
                    queue
    return visited
```

An empty priority queue Q and an array `visited[]` to record visited vertices are initialised at the start of the algorithm. To start the search, the source vertex s is added to the priority queue. The procedure retrieves the vertex u with the lowest degree from the priority queue, even though it is not empty, and if it hasn't been visited previously, it marks it as visited. Subsequently, the algorithm investigates u's neighbouring vertices, adding each unexplored vertex to the priority queue according to its degree. The set of visited vertices indicates the outcome of the graph search, and the process continues until the priority queue is empty. This unique notation employs symbols like `PQ_insert(Q, v)` and `PQ_extractMin(Q)` to describe operations on the priority queue and `visited[v]` to represent the status of a vertex. The phases and operations of the algorithm are represented clearly and concisely in this notation, which facilitates understanding and implementation.

IV. Definitions for the Degree-Based Traversal Algorithms

A. Degree Based Search Traversal Algorithm

The Degree Based Search Traversal Algorithm implicitly searches all the vertices from a given source vertex of a graph $G = (V, E)$. This computation is achieved by traversing in the order of degree of nodes of the graph. Based on this concept, a mathematical description of the Degree Based Search Traversal Algorithm is defined.

Definition 1: A sequence $P = P_1, P_2, \ldots, P_n$ represents the traversal of a graph in Degree level ordering where each $P_i = \{v_1, v_2, \ldots, v_m\}$ is the sequence of vertices traversed at Degree level order, where the degree sequence of the graph $\{\deg(v_1), \deg(v_2), \ldots, \deg(v_m)\}$ is in ascending or descending order.

Ascendant and Descent Nodes for the classification of the Degree based Traversal Algorithm, we introduce two preliminary concepts: Ascendant and Descent nodes of the graph. The node with the maximum degree in a graph is known as the Ascendant node of the graph.

Definition 2: If there exists a sequence of vertices $v_1, v_2, v_3, \ldots, v_n$ in graph $G = (V, E)$, then v_i is called the Ascendant node of the graph if $\deg(v_i)$ is the maximum degree of graph G.

The node with the minimum degree in a graph is known as the Descent node of the graph.

Definition 3: If there exists a sequence of vertices $v_1, v_2, v_3, \ldots, v_n$ in graph $G = (V, E)$, then v_i is called the Descent node of the graph if $\deg(v_i)$ is the minimum degree of graph G.

B. Ascendant Node First Search Algorithm (ANFS)

We cast the problem of finding a method for traversing a graph using an adjacency list. The algorithm initializes with the source node, then finds all of the adjacent nodes of the source node and continues with the ascendant node as the current node. In the ANFS Algorithm, there is an additional array to store the degrees of the vertices and the main constraint in the algorithm is to check whether the array becomes empty.

Definition 1: A sequence $P = P_1, P_2, \ldots, P_n$ represents the traversal of a graph in Degree level ordering where each $P_i = \{v_1, v_2, \ldots, v_m\}$ is the sequence of vertices traversed at Degree level order where the degree sequence of the graph $\{\deg(v_1), \deg(v_2), \ldots, \deg(v_m)\}$ is in descending order.

C. Descent Node First Search Algorithm (DNFS)

In the DNFS Algorithm, instead of starting from the Ascendant node, it starts from the Descent node.

Definition 1: A sequence $P = P_1, P_2, \ldots, P_n$ represents the traversal of a graph in Degree level ordering where each $P_i = \{v_1, v_2, \ldots, v_m\}$ is the sequence of vertices traversed at Degree level order where the degree sequence of the graph $\{\deg(v_1), \deg(v_2), \ldots, \deg(v_m)\}$ is in ascending order.

V. Computational Complexity Analysis

The time complexity analysis of the proposed Degree-Based Traversal Algorithm can be broken down into several key steps. Initially, the algorithm requires the initialization of various data structures, including the priority queue and the visited array, each of which has a time complexity of $O(V)$, where V represents the number of vertices in the graph. Inserting the source vertex into the priority queue incurs a logarithmic time complexity, specifically $O(\log V)$, due to the nature of the priority queue operations. The core of the algorithm is encapsulated in the main loop, which iterates through the vertices, resulting in $O(V)$ iterations. During each iteration, the algorithm performs several operations: extracting the minimum degree vertex from the priority queue, which has a time complexity of $O(V \log V)$, and processing all adjacent vertices, leading to a complexity of $O(E \log V)$, where E denotes the number of edges. Combining these operations, the overall time complexity of the algorithm can be expressed as $O((V+E) \log V)$. This comprehensive analysis highlights the efficiency of the DBS algorithm in traversing graphs, ensuring that it scales well with both the number of vertices and edges.

VI. Experimental Result

The experimental results highlight the exceptional efficiency of the new traversal approach, which intelligently prioritizes nodes based on their degree while avoiding revisiting already explored nodes. Across the Facebook, Twitter, and Email social network datasets, the new algorithm consistently demonstrated superior performance compared to both Breadth-First Search (BFS) and Depth-First Search (DFS) in terms of execution time. This signifies a substantial improvement in traversal efficiency, particularly evident when exploring highly connected nodes within the networks. By selecting nodes with higher degrees first, the new algorithm navigates through the graph in a manner that minimizes redundant visits and maximizes the coverage of important, central nodes.

These results strongly advocate for the effectiveness of the new approach in optimizing traversal tasks within social networks, offering a promising avenue for enhancing graph exploration and analysis in various network-based applications. The substantial reduction in traversal times across all three datasets underscores the significant impact of prioritizing highly connected nodes, reinforcing the potential of the new traversal strategy as a valuable tool in the realm of social network analysis and exploration.

In Fig. 2, the comparison of execution times for the three algorithms across the social networks Facebook, Twitter, and Email is depicted.

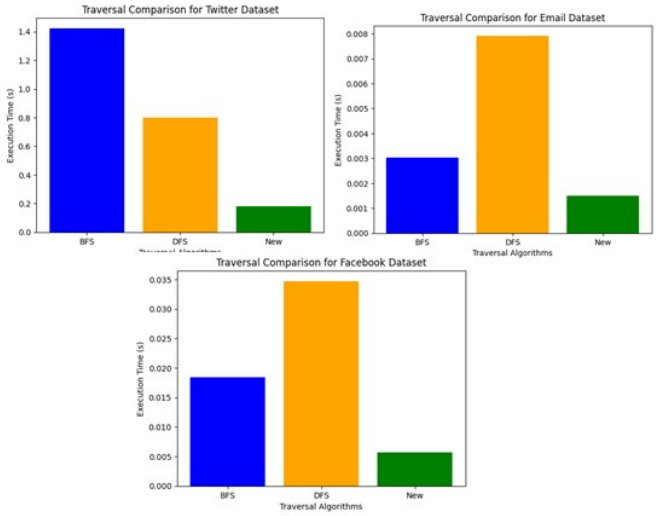

Fig. 2. Comparison of execution times for BFS, DFS, and the new traversal algorithm across the Facebook, Twitter, and Email datasets.

A. Datasets Overview

1) Facebook dataset: The Facebook dataset represents the friendship network among users of the popular social media platform. Comprising 4,039 nodes and 88,234 edges, each node in this undirected graph signifies a user, while an edge denotes a friendship connection. Researchers commonly use this dataset to study social network analysis, community detection, and information spreading on social media platforms [12].

2) Twitter dataset: The Twitter dataset captures the follower relationships among users of the microblogging platform. With 81,306 nodes and 1,768,149 edges, it is a large undirected graph where nodes represent Twitter users and edges represent the ”follow” relationship. This dataset is instrumental in studying information diffusion, influence maximization, and follower prediction on Twitter [12].

3) Email-Eu-core dataset: The Email-Eu-core dataset offers insight into email communication within a European research institution. This smaller directed graph consists of 1,005 nodes and 25,571 edges, where nodes represent email addresses and directed edges represent email exchanges. Researchers use this dataset to analyse email networks, study communication patterns, and detect anomalies in email traffic [12].

VII. Discussion

Using benchmark graph datasets with different sizes and architectures, we perform comprehensive experimental assessments to verify the efficacy of our optimisation technique. We assess how well our priority queue-based degree-based graph search algorithm performs in comparison to other cutting-edge optimisation methods and conventional degree-based graph search algorithms. Our tests illustrate the computational improvements from prioritised exploration based on node degrees, empirically demonstrating search efficiency, scalability, and computational overhead. The method that was developed is being compared to the Depth First Search (DFS) and Breadth First Search (BFS) algorithms [13]. The fundamental ideas of graph theory and algorithm design serve as the cornerstone of our research. BFS is a popular graph analysis technique that traverses all levels of the graph and methodically examines every vertex beginning from a given source [14]. As opposed to the traditional method of beginning at the root node, our Degree-Based Search Algorithm finds the vertex with the highest degree before beginning its search. This method is predicated on the idea that high-degree nodes frequently have important roles in network topologies, which could result in more effective graph exploration.

In the context of graph theory, both the efficiency of storage is high when using an adjacency list due to the requirement of storing edge values. Within the context of the Degree based Search algorithm, an adjacency list is utilised as a data structure to contain the values pertaining to the edges. In the context of BFS, it is necessary to utilise a queue data structure to maintain a record of the child nodes that have been examined but not yet traversed. The DFS algorithm employs a stack data structure to maintain a set of traversed vertices that may contain unexplored edges.

The experimental results show significant gains in search efficiency and scalability in addition to validating the effectiveness of our suggested optimisation technique. Our priority queue-based degree-based graph search method displays higher performance, particularly in cases involving large-scale graphs with heterogeneous degree distributions. We demonstrate the practical impact of our optimisation strategy by showing that prioritising nodes based on their degrees considerably improves the algorithm's ability to navigate complex graph structures. As we move forward, it will be crucial to further investigate the performance of our algorithm across an even wider range of graph types and sizes. Additionally, exploring potential hybridizations with other graph algorithms and adapting our method to dynamic or streaming graph scenarios could open up new avenues for research and application [15]. By continuing to refine and expand upon this work, we aim to contribute to the ongoing evolution of efficient graph algorithms, addressing the growing demands of complex network analysis in various domains.

VIII. Conclusion

In this paper, we introduced a novel degree-based traversal algorithm for graph exploration, emphasizing its efficiency and effectiveness in comparison to traditional Breadth-First Search (BFS) and Depth-First Search (DFS) methods. By prioritizing nodes based on their degrees and employing a priority queue to manage the traversal process, our algorithm significantly reduces redundant visits and enhances coverage of important nodes within the network.The theoretical analysis demonstrated that our algorithm achieves an overall time complexity of $O((V+E)\log V)$, highlighting its scalability and suitability for large-scale graphs. Experimental results across diverse datasets, including Facebook, Twitter, and Email networks, corroborated the theoretical findings, showing substantial improvements in execution time and traversal efficiency.

This research underscores the importance of leveraging node degree information to optimize graph traversal tasks,

presenting a promising avenue for enhancing social network analysis and other network-based applications. Future work will focus on further optimizing the algorithm for specific types of networks and exploring its applicability in real-time dynamic graph scenarios. Overall, the proposed degree-based traversal algorithm offers a valuable tool for efficient graph exploration, with potential impacts across various domains requiring effective network analysis and information dissemination.

REFERENCES

[1] Jialiang Zhang and Jing Li. (2018). Degree-aware Hybrid Graph Traversal on FPGA-HMC Platform. In Proceedings of 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3174243.3174245.

[2] Takuji Mitsuishi et al (September 2015). Breadth First Search on Cost-efficient Multi-GPU Systems, ACM SIGARCH Computer Architecture News Volume 43 Issue 4, pp 58–63. https://doi.org/10.1145/2927964.2927975.

[3] U. Kang, C. E. Tsourakakis and C. Faloutsos, "PEGASUS: A Peta-Scale Graph Mining System Implementation and Observations," 2009 Ninth IEEE International Conference on Data Mining, Miami Beach, FL, USA, 2009, pp. 229-238, doi: 10.1109/ICDM.2009.14.

[4] Richard E Korf (1999), Artificial intelligence search algorithms, Algorithms Theory Computation Handbook. Boca Raton, FL: CRC Press.

[5] John H. Reif (1985), Depth-first search is inherently sequential, Information Processing Letters, Volume 20, Issue 5, Pages 229-234. https://doi.org/10.1016/0020-0190(85)90024-9.

[6] Larry A. Taylor and Richard E. Korf (1993). Pruning duplicate nodes in depth-first search. In AAAI, pages 756–761.

[7] Jonathan L. Gross and Jay Yellen (2005), Graph Theory and Its Applications. (2nd Edition), Chapman and Hall/CRC.

[8] Michael D Atkinson, J-R Sack, Nicola Santoro, and Thomas Strothotte. Min-max heaps and generalized priority queues. Communications of the ACM, 29(10):996–1000, 1986. https://doi.org/10.1145/6617.6621.

[9] Ashish, D.D.V.S., Munjal, S., Mani, M., Srivastava, S. (2021). Path Finding Algorithms. In: Hassanien, A.E., Bhattacharyya, S., Chakrabati, S., Bhattacharya, A., Dutta, S. (eds) Emerging Technologies in Data Mining and Information Security. Advances in Intelligent Systems and Computing, vol 1286. Springer, Singapore. https://doi.org/10.1007/

[10] Xindong Wu, Xingquan Zhu, and Minghui Wu. 2022. The Evolution of Search: Three Computing Paradigms. ACM Trans. Manage. Inf. Syst. 13, 2, Article 20 (June 2022), 20 pages. https://doi.org/10.1145/3495214.

[11] L. Singh, S. Khare, A. Parvez and S. Verma, "Research Paper on Path-finding Algorithm Visualizer," 2022 International Conference on Cyber Resilience (ICCR), Dubai, United Arab Emirates, 2022, pp. 1-4. https://doi.org/10.1109/ICCR56254.2022.9995925.

[12] Jure Leskovec and Andrej Krevl, SNAP Datasets: Stanford Large Network Dataset Collection, http://snap.stanford.edu/data, Jun 2014.

[13] Shaukat, Fatima & Shafique, Ayesha & Islam Qadri, Ayesha. (2022). Comparative Analysis of Search Algorithms in AI. 10.13140/RG.2.2.29282.61123.

[14] T. H. Cormen et al., Introduction to Algorithms, 4th ed. Cambridge, MA: MIT Press, 2022

[15] Schiller, Benjamin & Deusser, Clemens & Castrillón, Jerónimo & Strufe, Thorsten. (2016). Compile- and run-time approaches for the selection of efficient data structures for dynamic graph analysis. Applied Network Science. 1. 10.1007/s41109-016-0011-2.