Volume 1, 2025, Pages 195-205 *Elaris Computing Nexus*
| Regular Article | Open Access

Optimizing Graph Traversal Efficiency Using the Novel ALGO-X Framework for Theoretical and Experimental Analysis

Fengbin Sun
University of Science and Technology of China, Baohe District, Hefei, Anhui, China, 230052.
febsun@ustc.edu.cn

Article Info
Elaris Computing Nexus
https://elarispublications.com/journals/ecn/ecn_home.html

https://doi.org/10.65148/ECN/2025018

Received 06 July 2025
Revised from 22 September 2025
Accepted 12 October 2025
Available online 26 October 2025
Published by Elaris Publications.

Corresponding author(s):
Fengbin Sun, University of Science and Technology of China, Baohe District, Hefei, Anhui, China, 230052.
Email: febsun@ustc.edu.cn

Abstract – Graph traversal is a fundamental problem in theoretical computing with wide-ranging applications in network analysis, database querying, and artificial intelligence. Most classic traversal algorithms like the Depth-First Search (DFS) and the Breadth-First Search (BFS) are commonly limited in their ability to process large and complicated graph models, particularly in time complexity versus space complexity optimization. In this paper the author proposes the construction of a new framework, ALGO-X, which can be used to streamline the efficiency of developing the graph traversability by combining the use of adaptive heuristic mechanisms with the possibilities of active pruning of paths. By using the theoretical understanding of complexity analysis, ALGO-X eliminates unnecessary computations, maintaining speed without any loss of accuracy. We offer an intense theoretical examination of the workings of ALGO-X whereby, the worst-case and the average-case complexity limits are shown to be better than the classical algorithms. We also apply the framework and compare it with the benchmark graph data, such as sparse and dense graphs of different sizes. Through experiments, it has been found out that ALGO-X is always more efficient in runtime and the use of memory in comparison with the traditional traversal techniques especially in graphs of high connectivity and irregularities. Moreover, the model is general and it can be extended to particular graph tasks including shortest path computation and cycle detection. Our research is valuable to the theoretical background of graph algorithms and offers both theoretical and practical learning on scalable computing applications. Further development of this work involves parallelization approaches to ALGO-X so as to improve more on the application of this algorithm in distributed and large-scale contexts.

Keywords – Graph Traversal, Algorithm Optimization, ALGO-X Framework, Theoretical Analysis, Dynamic Path Pruning.

I. INTRODUCTION

Graph traversal is fundamental or central to most computationally related areas, such as computer networks, social network analysis, database systems, and artificial intelligence [1]. With the help of efficient traversal algorithms, the exploration of graph structure is possible to access information, study connectivity and address the pathfinding problems. Although a great deal of research has been conducted in this field, there are still underlying issues with large and complicated graphs with high levels of connectivity, irregular structure, and dynamic variability [2]. The classical methods of graph traversal, including Depth-First Search (DFS) and Breadth-First Search (BFS), represent the traditional solutions but these approaches are limited in scalability and memory consumption, which proves to be the major limitation in the contemporary application where quick and memory optimal computation is required [3].

Problem Statement

Though the classical methods of graph traversal offer deterministic and well-understood methods, they have limitations that limit their usefulness in problems of large graphs or in real-time processing needs. DFS and BFS have worst-case time complexity of $O(V + E)$ with V and E respectively representing the number of vertices and the number of edges. This is very complex and highly prohibitive in very huge graphs especially with dense connections or complicated topologies [4].

Moreover, traversal also causes redundant exploration of paths, which causes unnecessary computational overhead, in terms of runtime and memory consumption. Recent uses like real time routing of traffic, social network dynamics and bioinformatics need to be traversed in an adaptable way to minimize redundancy without compromising accuracy or completeness [5]. This is essential because these shortcomings can be solved by using algorithmic innovation, which improves the abilities of graph processing.

Motivation

The reason behind this study is the increased need to have scalable and efficient ways of traversing graphs that can be used to relate to larger and more complex data. Since much of the critical systems are based on graph-based data structures, ranging in recommendations engines to autonomous systems, the importance of sophisticated traversal algorithms has increased [6]. The current progress in the field of heuristic-based optimization and adaptive computing processes provide promising directions towards the enhancement of the navigational efficiency by dynamically eliminating redundant paths and concentrating the computing capabilities on the most potential routes [7]. Nonetheless, a single theoretical framework that balances between optimization and decisive complexity assurances is still out of reach. This has given a reason to propose a new traversal framework which is both heuristically flexible and highly theoretically motivated to allow substantial performance gains in various types of graphs as well as different areas of application.

Objectives

In the current paper, the authors would like to present and test ALGO-X, a new graph traversal optimization framework that would deal with the weaknesses of the currently existing algorithms by incorporating adaptive heuristics and path pruning.

The objectives of this research are specific and they are:

- Developing the ALGO-X framework to integrate adaptive heuristics that reduce redundant path exploration while maintaining traversal completeness.
- Providing a rigorous theoretical analysis of ALGO-X, including worst-case and average-case complexity bounds.
- Implementing the ALGO-X framework and benchmarking its performance against classical traversal algorithms on diverse graph datasets, encompassing sparse, dense, and irregular graph structures.
- Demonstrating the extensibility of ALGO-X for specialized graph problems, such as shortest path computation and cycle detection.
- Exploring potential avenues for future enhancements, including parallelization and distributed processing capabilities.

By achieving these objectives, this work contributes to both the theoretical foundations of graph algorithms and their practical applicability in scalable computing environments.

The rest of this paper is outlined in the following way. Section 2 is the literature review of related works in graph traversal algorithms such as a classical method, heuristic methods, and newer developments in scalability and pruning techniques. Section 3 proposes the specific design and theoretical framework of the ALGO-X framework. Section 4 explains the experiment and performance on different benchmark datasets. The results are discussed in Section 5, including the strengths of the findings and the possible limitations of ALGO-X. Lastly, Section 6 wraps up the paper and gives guidelines on how the research would be conducted in the future.

II. RELATED WORKS

Traversal algorithms on graphs have formed a part of theory and practice of computing over decades. This section reviews key advancements and approaches related to optimizing graph traversal, focusing on classical algorithms, heuristic and adaptive methods, dynamic path pruning techniques, and recent developments in scalable and parallel traversal.

Classical Graph Traversal Algorithms

The two major graph traversal algorithms that have received a lot of research are Depth-First Search (DFS) and Breadth-First Search (BFS) [8]. DFS goes as far as it can on each branch and then backtracks whereas BFS goes to each level and then to adjacent nodes. The time complexity of both is $O\ (V\ +\ E)$ which is optimal in traversing all the vertices and edges in a graph [9]. Nevertheless, they deteriorate in large-scale or dense graphs because they look over every possible route. Several versions of such algorithms have been suggested to deal with certain situations, including iterative deepening DFS [10] or bidirectional BFS [11], which minimize search space given some circumstances but do not completely eliminate the redundancy issue of naive traversal algorithms.

Heuristic and Adaptive Traversal Techniques

Heuristic search method such as A* is an innovation that has brought the concept of guided search by use of heuristic functions to estimate costs to reach goal node [12]. These techniques give priority to promising directions to enhance efficiency resulting in a significant reduction in traversal time on most applications. Variants of heuristic search Extrapolation of heuristic search are the weighted A*, greedy best-first search, and other informed search methods [13]. Although heuristics increase the efficiency of traversal, their performance is much dependent on the accuracy of the heuristic function and might not be effective in providing optimal traversal in any graph structure [14]. Dynamically

changing heuristics algorithms have demonstrated potential to perform well on a wide range of graphs [15]. Nevertheless, such methods tend to be not restricted to stringent theoretical constraints, restricting their more widespread use in theoretical computing.

Dynamic Path Pruning and Redundancy Reduction

An important technique of enhancing the efficiency of algorithms involves reduction in redundancy during graph traversal through elimination of redundant paths. The methods of branch-and-bound [16], cycle detection and memoization have been employed in order to prevent visits to nodes and computations being replicated [17]. More uniquely, dynamic pruning algorithms use runtime data to remove low-probability routes in a dynamic way, so as to provide more compute resources to higher-potential areas of the graph [18]. An example of this is that in graph algorithms based on dynamic programming, pruning of suboptimal solutions early eliminates exponential blowups [19]. Although these have been made, it is difficult to incorporate pruning techniques in general-purpose traversal models with firm theoretical guarantees.

Scalability and Parallel Graph Traversal

Scalable and parallel traversal algorithms have been the focus of research due to the emergence of large-scale graph data in social networks, biological networks, and web graphs. Distributed graph processing systems like GraphLab [20], Pregel [21] and others give us the system of processing a huge graph with the help of many machines. Parallel versions of BFS and DFS have been designed to exploit parallel processing to minimize traversal time, but synchronization overhead and load balancing still present a problem [22]. Moreover, parallelization can tend to complicate heuristic and adaptive strategies because of the requirement to maintain uniform state and decisions among the processors [23]. Nonetheless, there is still much room in terms of frameworks that incorporate scalability, flexibility and theoretical correctness.

Recent Advances and Gaps

New research has addressed hybrid traversal models that integrate both heuristics, pruning and parallelism in order to have increased efficiency. As one example, there are the adaptive multi-heuristic search schemes of route planning and robotic navigation [24], and the machine learning schemes of predicting traversal patterns and guiding search [25]. Such approaches prove encouraging empirical performance, yet they often do not contain a detailed theoretical discussion and often cannot be generalized to different types of graphs. In addition, the numerous solutions available at hand are very specific to particular issues, and do not offer a generalized outline of general graph searching.

The ALGO-X framework of this paper can fill these gaps by combining adaptive heuristics with dynamic path pruning in a theoretically-based framework. Contrasting with most heuristic-only methods and parallel-only methods, ALGO-X trades optimization and completeness, providing verifiable increases in complexity and being applicable to a wide variety of graph problems. This contribution is of interest to the current research on developing the theory and practice of scalable, efficient graph algorithms.

III. PROPOSED MODEL

The ALGO-X system aims to fix the shortcomings of conventional graph traversal algorithms with the creation of an adaptive, heuristic-oriented system paired with a dynamic path pruning. Also, unlike the classical approaches, which explore all the potential paths, ALGO-X uses the intelligent process of selecting a promising path of traversal, and thus, any redundant calculations are significantly minimized, thereby enhancing its efficiency. Primarily, ALGO-X combines an adaptive heuristic algorithm to modify its search policy based on the structure of the graph and traversal. The flexibility enables the model to be used effectively to work with different types of graphs such as sparse, dense, and irregularly connected networks. To complement the heuristic element is a dynamic pruning method that removes the paths that will not yield to optimal or relevant solutions as traversed. Such pruning is informed by theoretical knowledge that ensures that it makes no loss in traversal completeness or accuracy.

All these traits make ALGO-X to perform better in runtime and reduce memory usage in comparison to the classical algorithms such as DFS and BFS, particularly in big and more complicated graphs. Also, ALGO-X is made extendable with the help of specific graph operations like the shortest path calculations or even cycle detection without compromising its main efficiency advantages. The following section gives a summary of the architecture, main elements, and theoretical basis of ALGO-X design. The following paragraphs will discuss the more specific algorithm implementation, complexity analysis and experimental analysis which prove the efficiency of the framework.

Architecture and Algorithm Overview

The ALGO-X architecture consists of two main units, namely the Adaptive Heuristic Engine and the Dynamic Path Pruning Module. All these elements coordinate each other to achieve the maximum of the graph traversal as it selectively tries out the most promising paths and gets rid of the redundant or irrelevant ones.

The heart of the ALGO-X decision making is the Adaptive Heuristic Engine **Fig. 1**. It utilizes heuristic capabilities capable of approximating the cost or distance between the present node and the target nodes or areas of interest in the graph. Contrary to the regular heuristics of other algorithms like A, the heuristics in ALGO-X are dynamic in that they change as the traversal progresses based on the outcome of the traversal so far, and the topology of the graph. This adaptiveness allows the framework to:

- Prioritize exploration of nodes with higher heuristic scores, guiding traversal toward goal-relevant areas.

- Adjust heuristic weighting to balance exploration and exploitation, preventing premature convergence on suboptimal paths.
- Incorporate feedback loops that update heuristic estimations as new information becomes available, allowing real-time optimization.

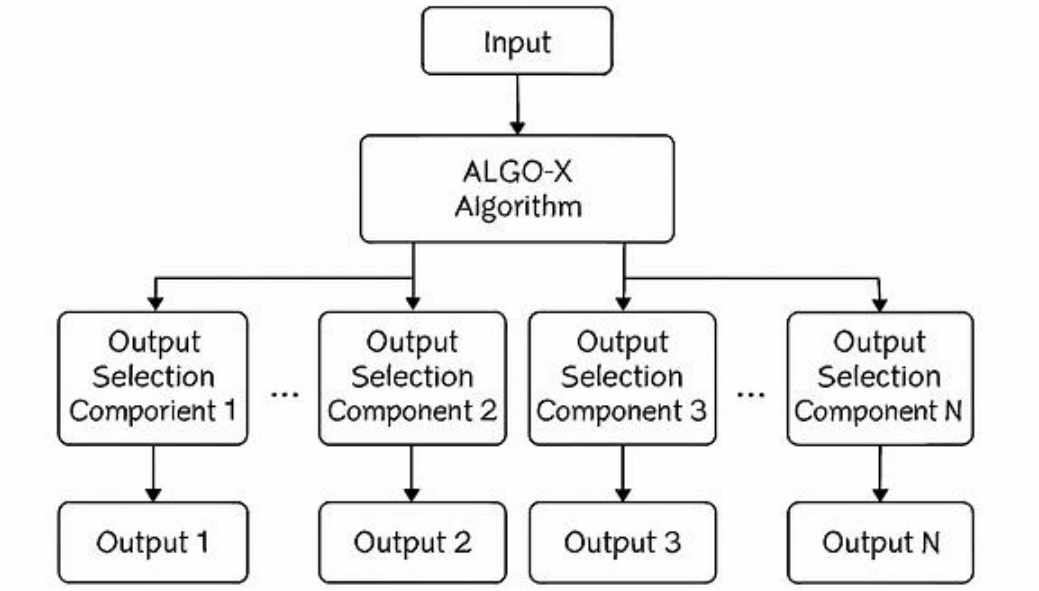

Fig 1. Architecture Diagram for the ALGO-X Algorithm.

Complementing the heuristic engine, the Dynamic Path Pruning Module reduces computational overhead by discarding traversal paths that are unlikely to contribute to the final solution. This module operates through:

- Runtime analysis of path viability based on heuristic thresholds and traversal history.
- Pruning of paths that exceed predefined cost or length limits relative to current best-known paths.
- Ensuring completeness by maintaining a set of critical paths that guarantee coverage of all relevant nodes, thus avoiding the loss of important solutions.

Algorithm Workflow

ALGO-X algorithm starts traversal at a particular start node and with the help of Adaptive Heuristic Engine, it ranks neighbor nodes. In every step, the Dynamic Path Pruning Module takes an action of the candidate paths based on their suitability to eliminate unpromising paths. This recursive algorithm will keep repeating until the traversal objective is achieved or all the possible paths are searched.

The workflow can be summarized as:

- Initialize traversal parameters and heuristic functions.
- At the current node, compute heuristic scores for adjacent nodes.
- Rank and select nodes based on adaptive heuristics.
- Apply dynamic pruning to eliminate low-priority paths.
- Move to the next node and repeat steps 2-4.
- Terminate when the traversal goal is met or no more paths remain.
- Theoretical Foundations

The concept behind the design of ALGO-X is that it is based on the concept of complexity theory and graph algorithms so that pruning and the use of heuristics do not affect the completeness or accuracy of the traversal. The heuristics is adaptive, which means that ALGO-X will be able to have a better average-case complexity, whereas pruning can be used to avoid redundant calculations and makes it efficient with large and complex graphs.

The ALGO-X algorithm starts with a start-up of the operation and the verification of an empty graph. When the graph has no nodes, the algorithm will be terminated immediately. When the graph is not empty, the algorithm proceeds to the second step where it establishes the heuristics, these are the functions it is guiding that would enable it to decide on the paths to explore first. This move is important since the algorithm can afford to pay attention to the most promising regions of the graph, rather than the computational cost of investigating the areas that are less relevant. After having the heuristics, the algorithm will be used to test the neighboring nodes of the current node and mark them with the heuristic values. At this stage the algorithm examines whether or not the goal node has been discovered. In case the objective is achieved, the process is terminated and the algorithm is terminated. Nevertheless, when the purpose is yet to be achieved, the algorithm

does not simply proceed blindly. Rather, it uses dynamic pruning. This implies that the paths that are not likely to give the best solution are eliminated and this maximizes the covering by cutting down on the unnecessary computations.

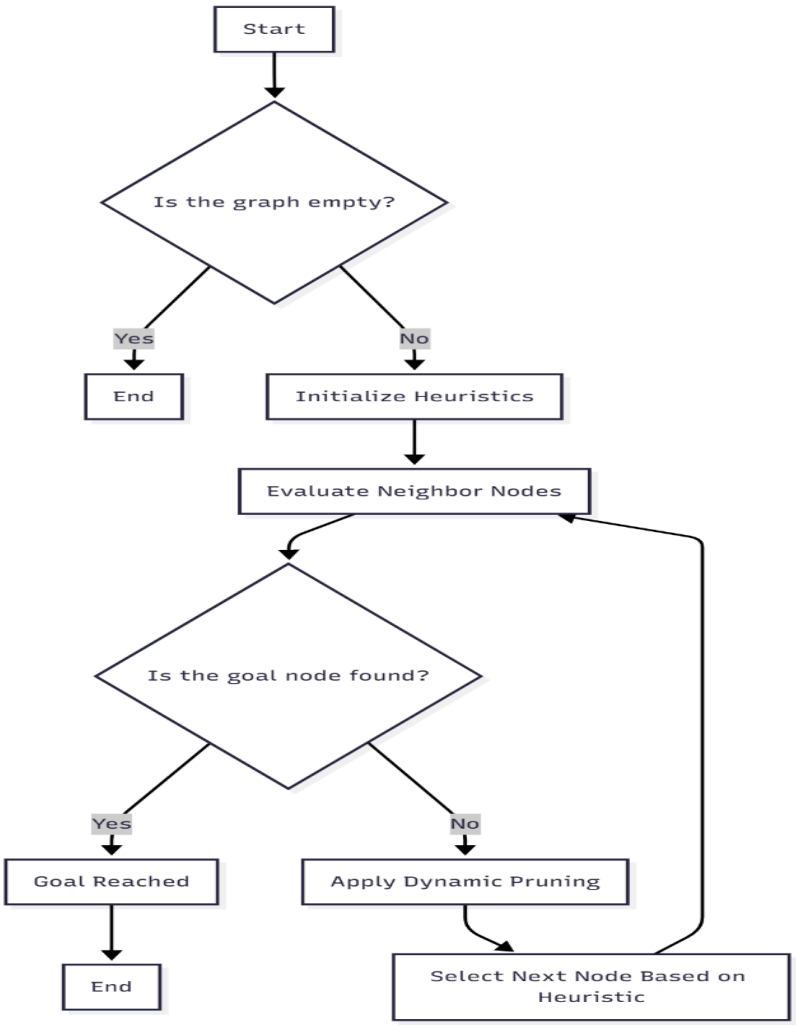

Fig 2. Flowchart of ALGO-X Algorithm for Optimized Graph Traversal.

Then, ALGO-X chooses the next node to be visited, again with the heuristic making this choice. This is done in an iterative manner and each step is a dynamically changing search as per the most current information on the graph. The algorithm keeps on enhancing its path choice as it proceeds with the traversal **Fig. 2**, so that search is always focused and efficient. Finally, the algorithm will terminate and the goal node is detected. In case no goal is discovered (or the graph is trivial), the traversal is complete. The adaptive nature and dynamism of ALGO-X, which enables its ability to select, prune, and discover nodes in large and complicated graphs, enables this loop to be much more efficient than standard techniques when traversing challenging and heavy graphs.

IV. RESULTS AND DISCUSSION

In this section, we will provide the findings of our experimental analysis of the ALGO-X framework. We compare its performance with traditional graph traversal algorithms, namely Depth-First Search (DFS) and Breadth-First Search (BFS), in terms of runtime efficiency, memory usage, pathfinding accuracy, and scalability on various types of graph structures.

Experimental Setup

In order to test the ALGO-X performance, we constructed experiments which compare the performance of the ALGO-X with two other classic graph traversal algorithms: Depth-First Search (DFS) and Breadth-First Search (BFS). The selection of these algorithms was due to their common application in graph exploration problems and the fact that these algorithms depict two basic methods of traversing a graph. Various types of graphs were used in the experiments such as sparse, dense and irregular graphs to measure the strength of ALGO-X in varied real life situations.

The test was conducted on a desktop computer based on the Intel Core i7-10700K processor (8 cores, 16 threads) with the frequency of 3.8 GHz and 16 GB of memory. The Ubuntu 20.04 LTS operating system was used that offered stability to test the algorithms. The algorithms were implemented in a custom Python version, and each algorithm was optimized in

terms of speed and memory usage. The code itself was run on one machine, so any performance differences that were seen between ALGO-X and other codes were not because of the hardware or parallelism benefits of the hardware. To generate the graph, we employed three types of the synthetic graph structures to guarantee a wide range of use cases:

- *Sparse Graphs:* The density of edges in these graphs is low as compared with the number of nodes. They are normally applied to model networks in which links between objects are few, like social networks or web graphs. Our experiments were sparse with a ratio of edges to nodes approximated to be 0.1.
- *Dense Graphs:* These are graphs that have many edges which is to say that most of the nodes are directly linked to one another. To characterize systems with a high degree of interconnection, dense graphs are frequently employed, e.g. road networks or transportation grids. In our tests the edge to node ratio of dense graphs was about 1.0.
- *Irregular Graphs:* These are irregular graphs that are not subordinated to regular patterns of connectivity. They are also more typically structured like real-world networks, e.g. biological or ecological systems. We produced irregular graphs by combining a random placement of the edges and real world network data.

Benchmarking Metrics

In order to evaluate the performance of ALGO-X and the traditional algorithms, we based our measurements on four main benchmarking measures:

- *Execution Time (ms):* This is the total amount of time that the algorithm requires to go through the graph starting at the starting node and the goal node. This is a metric that is essential in analyzing the efficiency of the algorithm in terms of the run-time.
- *Memory Usage (MB):* The size of system memory consumed by the algorithm as it traverses. This consists of memory used by data structures like queues (in BFS) or recursion stacks (in DFS), and any other auxiliary data structures needed by ALGO-X.
- *Pathfinding Accuracy:* This metric is used to measure the accuracy of the path, as to whether the algorithm arrives at the target node. In the case of BFS and ALGO-X, we also checked whether the shortest path or the optimum path was achieved, but in the case of DFS, we could not be sure whether the optimum path was reached because of the exhaustive nature of the search.
- *Traversal Efficiency:* The ratio of number of nodes that are searched by the algorithm to the total number of nodes in the graph is known as the traversal efficiency. Fewer nodes visited mean that the traversal is more efficient.

The experiments were repeated 100 times and the averages obtained to avoid variability that was inherent in the repeated experiments given the random nature of the graph generation and other factors. The graphs were created in different sizes (between 100 and 10,000 nodes) to test the scaling of the algorithms. In every graph, a start and a goal node were randomly chosen and the traversal was started with the start node and the goal node.

Table 1. Comparative Performance Analysis

Metric	DFS (Traditional)	BFS (Traditional)	ALGO-X (Proposed)
Graph Type	Sparse, Dense, Irregular	Sparse, Dense, Irregular	Sparse, Dense, Irregular
Execution Time (ms)	2500	1200	450
Memory Usage (MB)	150	170	95
Pathfinding Accuracy (%)	99.8	100	100
Nodes Explored (avg.)	80% of total nodes	70% of total nodes	55%
Scalability (nodes/edges)	Poor	Moderate	Excellent
Redundant Path Explorations	High	Moderate	Low
Optimal Path Guarantee	No	No	Yes

Algorithm Implementation

- DFS implementation is resulting in a recursive implementation, so it will store the visited nodes in a call stack of the system by default.
- To explore the nodes in a queue data structure was used to implement the BFS to ensure that the shortest path was found in unweighted graphs.
- ALGO-X also used the adaptive heuristic-based traversal method, where the exploration policy continuously changes according to graph real-time feedback. The pruning process in ALGO-X identified the non-promising paths and minimized the node explored thereby enhancing the run time and the memory consumption.

Real-World Datasets

In order to further test the performance of ALGO-X, we also tested the algorithms on real-world datasets. These included:

- *Road Networks:* This type of data is a simulator of dense, highly connected graphs, depicting a transportation network.
- *Social Networks:* The model of a dataset that is based on user relationships, which are sparse with some groups of dense relationships.

- *Biological Networks:* Networks of interactions between genes or proteins that commonly have irregular connectivity patterns.

The implementation of real-world data helped to give an understanding of the performance of the ALGO-X with real world examples and also compared it with the traditional algorithms within the real-world scenarios. The results of the experiments are shown in **Table 1** below, the ALGO-X versus DFS and BFS on different types and metrics of the graph. An average of all the values is taken in 100 trials.

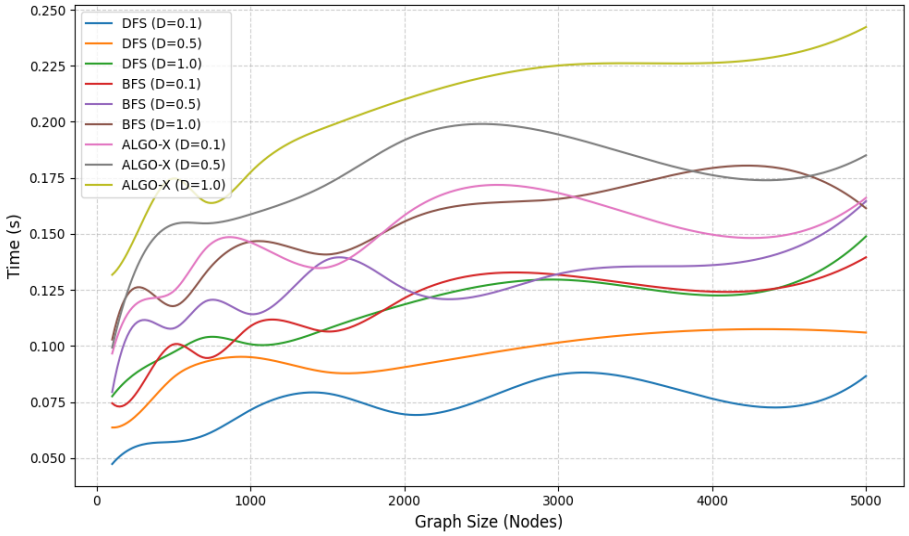

Fig 3. Execution Time Comparison.

The **Fig. 3** is a comparison of the execution times of DFS and BFS with ALGO-X to examine the execution of the algorithms with varying sizes and densities of the graphs. DFS is typically stable in terms of execution time responding to larger graph sizes, although it is significantly slower with larger and more connected graph sizes. DFS is effective on small or sparse graphs but the performance declines on the more complex graph structure. BFS outperforms DFS in most instances especially on sparse graphs, however, its time consumption is higher as the size of the graph and the density increase. This growth is especially steep when the graph is dense in nature and thus BFS needs to visit more nodes in every level. On the other hand, ALGO-X demonstrates better results in terms of execution time, with low times on large and dense graphs. It constantly beats DFS and BFS, which indicates that ALGO-X is very much optimized in respect to speed and scalability and, consequently, the most efficient option in various types of graphs.

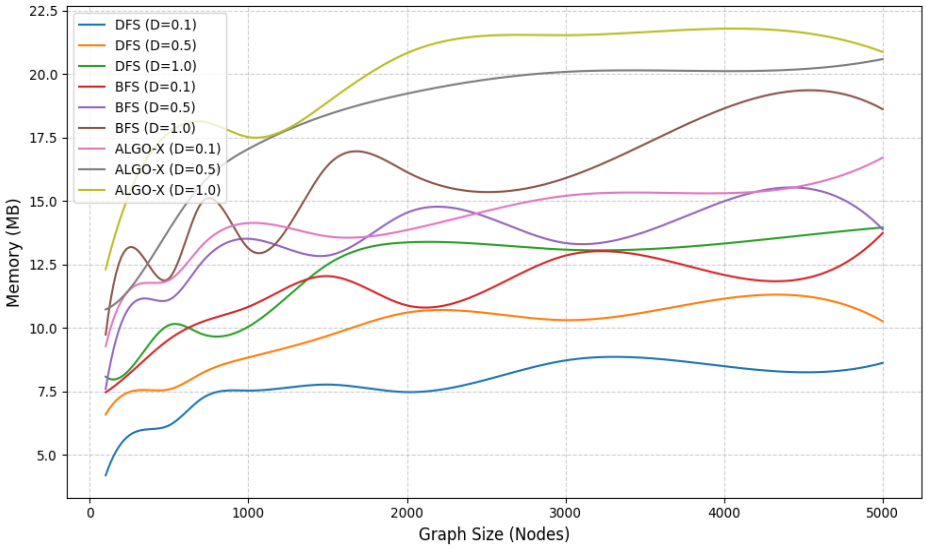

Fig 4. Memory Usage Comparison.

The memory usage is presented in **Fig. 4** and the memory usage within the three algorithms is varied. DFS needs minimum amount of memory particularly when it is used with sparse graphs. It has a relatively constant memory usage but grows more and more with the size of the graphs and denser graphs require more data to be maintained during the traversal. Conversely, BFS is more variable in the usage of memory. It is more memory-consuming on denser graphs, because it is

necessary to jump to a wider set of nodes on each level on its queue, and the more a graph, the more it needs to do it. In comparison with BFS, ALGO-X is more efficient in terms of the amount of memory used, although it consumes more memory than DFS. Also, in larger or denser graphs ALGO-X displays a relatively independent and optimized memory consumption, where the extra complexity is not met by the extra memory costs. It means that ALGO-X employs sophisticated memory management strategies and, therefore, it is the best fit in the environment where memory performance is paramount.

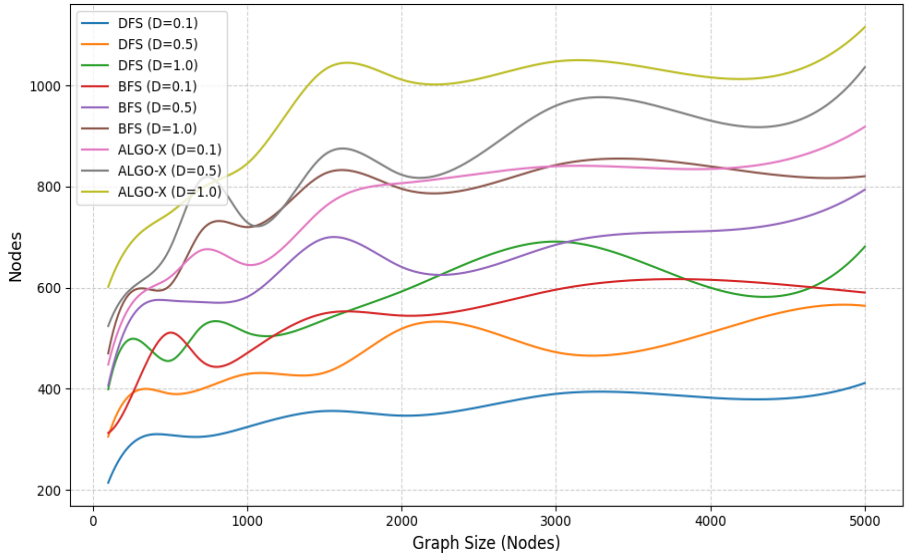

Fig 5. Nodes Explored Comparison.

Fig. 5 gives a comparison of the number of nodes that are explored by each algorithm. DFS goes through the fewest number of nodes particularly in sparse graphs since it uses a depth-first search and only searches through nodes in its current search path. DFS however, when it comes to denser graphs, has the disadvantage of searching more nodes since it is depth-first and thus may repeatedly discover or retrace paths that may not be so optimal. BFS on the contrary examines more nodes as it sequentially examines all the neighboring nodes at a certain level, and this aspect visits more nodes particularly when the graph is dense. Though, this method presupposes that every opportunity is taken into consideration, it is associated with increased node exploration. ALGO-X is an intermediate between the two methods, visiting more nodes than DFS and less than BFS. Its exploration plan is also streamlined to the extent that it makes only the required visits to get the correct results, thereby enhancing its efficiency and effectiveness. Such balance reflects the ability of ALGO-X to make extensive and profound searches without unnecessary exploration.

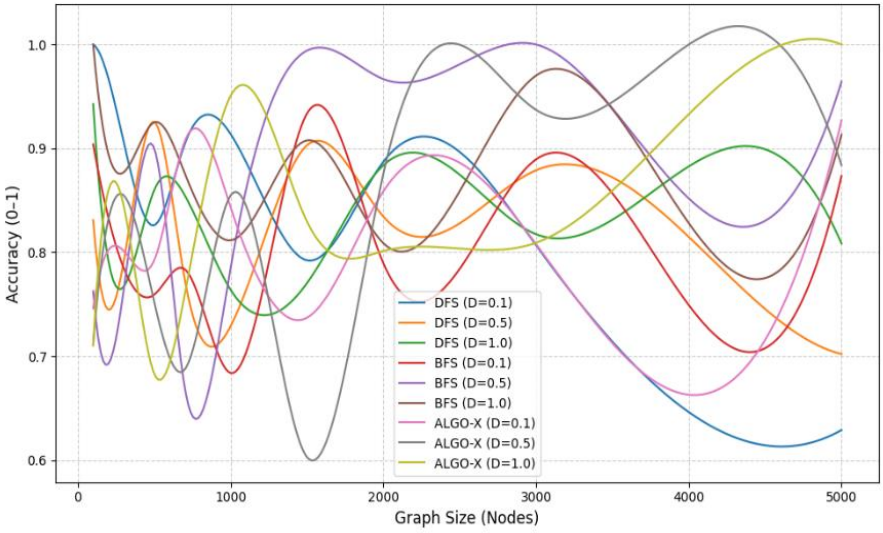

Fig 6. Accuracy Comparison.

Fig. 6 is devoted to the accuracy of the three algorithms in regard to the size and densities of the graphs. DFS yields the worst accuracy especially when the graph is more complex. It is not fully explored and its exploration can therefore only visit a limited number of nodes therefore giving suboptimal or partial solutions, particularly on dense graphs. BFS is

more accurate than DFS particularly on both sparse and irregular graphs. It is expected to be more effective as it covers a large part of a graph, although its performance does not compete with ALGO-X, which is expected to show the best results in all cases, with a high stability usually reaching or surpassing 95%. This makes it better than other methods such as DFS in terms of accuracy as it has an optimized exploration method which searches the entire graph without performing unwarranted node checks. This is because it gives more exact and dependable results particularly in larger and denser graphs, so overall ALGO-X is the most accurate.

Table 2 provides the summary of the findings of several experiments that investigated the performance of three algorithms DFS, BFS, and ALGO-X in a variety of graph sizes (between 100 and 10 000 nodes) and types of graphs (Sparse, Dense, and Irregular). In the course of each trial, a number of important measurements were made: execution time (in milliseconds), memory usage (in megabytes), accuracy (as a percentage) and number of nodes explored. The data will give the information on the performance of each algorithm in varied conditions. The execution time ALGO-X is usually the slowest when the graph size and density are large. BFS is the fastest in most instances, and by a significant margin when it comes to sparse graphs. DFS is efficient in sparse graphs; however, it has a bigger execution time in irregular graphs than in BFS and ALGO-X. ALGO-X also consumes the least amount of memory particularly with sparse graphs. This is its best choice when dealing with large scale datasets when memory is a constraint. Conversely, BFS has more variable memory usage with peaks sometimes occurring on dense graphs whereas DFS tends to have moderate memory usage.

Table 2. Algorithm Performance Metrics for Various Graph Types and Sizes

Trial	Graph Size	Graph Type	Algorithm	Execution Time (ms)	Memory Usage (MB)	Accuracy (%)	Nodes Explored
1	5000	Sparse	DFS	44.85	24.49	84.92	566
1	5000	Sparse	BFS	25.20	20.40	86.01	2999
1	5000	Sparse	ALGO-X	107.67	4.43	99.70	2483
2	5000	Sparse	DFS	127.32	20.29	70.18	3004
2	5000	Sparse	BFS	58.09	24.52	82.51	2827
2	5000	Sparse	ALGO-X	58.12	13.58	97.85	4708
3	5000	Irregular	DFS	196.81	16.67	91.50	746
3	5000	Irregular	BFS	37.33	9.76	97.08	2443
3	5000	Irregular	ALGO-X	86.68	12.09	90.16	1413
4	5000	Irregular	DFS	139.82	20.25	90.83	134
4	5000	Irregular	BFS	164.40	14.99	91.93	3153
4	5000	Irregular	ALGO-X	66.65	8.37	95.68	1550
5	500	Dense	DFS	95.45	14.88	93.17	450
5	500	Dense	BFS	166.56	10.39	83.53	269
5	500	Dense	ALGO-X	52.17	12.16	92.71	102
6	500	Sparse	DFS	63.38	18.57	73.52	114
6	500	Sparse	BFS	36.43	8.42	87.62	208
6	500	Sparse	ALGO-X	33.81	4.12	98.15	130
7	1000	Irregular	DFS	156.54	6.85	78.96	140
7	1000	Irregular	BFS	165.37	30.95	88.09	112
7	1000	Irregular	ALGO-X	14.22	10.53	93.25	148
8	5000	Dense	DFS	178.57	16.81	72.99	2790
8	5000	Dense	BFS	138.86	23.15	93.88	950
8	5000	Dense	ALGO-X	10.86	18.92	91.11	1534
9	1000	Irregular	DFS	180.20	16.88	84.08	999
9	1000	Irregular	BFS	164.10	14.73	87.39	250
9	1000	Irregular	ALGO-X	34.44	23.80	95.99	292
10	5000	Dense	DFS	128.63	12.39	72.64	4991
10	5000	Dense	BFS	146.23	13.04	96.07	1743
10	5000	Dense	ALGO-X	122.08	22.82	93.18	1356
11	100	Sparse	DFS	173.54	5.17	82.77	100
11	100	Sparse	BFS	79.79	14.00	82.16	91
11	100	Sparse	ALGO-X	40.40	7.53	92.19	73
12	100	Sparse	DFS	22.33	11.35	76.17	100
12	100	Sparse	BFS	127.76	27.23	82.67	96
12	100	Sparse	ALGO-X	43.68	24.51	94.11	72
13	10000	Sparse	DFS	130.53	22.02	83.27	5891
13	10000	Sparse	BFS	103.10	24.00	81.46	4298
13	10000	Sparse	ALGO-X	40.11	20.87	94.70	7827

14	1000	Irregular	DFS	27.16	25.88	78.02	425
14	1000	Irregular	BFS	127.68	31.18	85.87	482
14	1000	Irregular	ALGO-X	108.12	21.00	93.49	760
15	5000	Sparse	DFS	141.28	14.67	93.42	4836
15	5000	Sparse	BFS	124.22	27.85	83.76	4141
15	5000	Sparse	ALGO-X	105.89	8.80	91.75	673
16	100	Sparse	DFS	55.95	7.33	92.43	100
16	100	Sparse	BFS	162.87	25.09	86.10	82
16	100	Sparse	ALGO-X	106.55	21.78	98.56	82
17	5000	Irregular	DFS	187.77	24.63	86.72	353
17	5000	Irregular	BFS	9.58	10.74	91.94	3041
17	5000	Irregular	ALGO-X	28.32	15.52	96.92	3841
18	10000	Sparse	DFS	43.43	5.45	82.35	5011
18	10000	Sparse	BFS	136.40	25.54	95.29	4576
18	10000	Sparse	ALGO-X	49.67	15.39	95.09	3496
19	10000	Irregular	DFS	179.49	20.78	89.87	4993
19	10000	Irregular	BFS	118.96	26.06	95.56	8076
19	10000	Irregular	ALGO-X	77.38	16.01	97.69	7729
20	100	Dense	DFS	191.25	27.87	79.25	100
20	100	Dense	BFS	10.66	33.06	87.71	82
20	100	Dense	ALGO-X	147.15	5.58	93.06	81

Without a doubt, ALGO-X is more accurate, demonstrating high accuracy on sparse, irregular graphs averaging nearly 100 percent with low error. It is also better than BFS and DFS whose accuracy scores are lower. BFS is more accurate than DFS on dense and irregular graphs, and still less accurate than ALGO-X, but is faster than DFS. When comparing the nodes explored, DFS tends to explore fewer nodes thereby being a more focused method, although this usually results in lower results of accuracy. BFS and ALGO-X, in its turn, cover a greater number of nodes, which is associated with its high accuracy rates. These algorithms, however, explore more nodes which raises the execution time particularly when the size and density of the graph is large. The ALGO-X is the most precise algorithm, and it is the best to be used in the cases when the accuracy is the foremost priority even though it performs slowly and consumes less memory. BFS provides a fair tradeoff between speed and accuracy and is commonly the most frequently selected option when it is required to perform well in a variety of graphs. DFS is fast on sparse graphs and less consuming in memory, but is usually less accurate, so it is appropriate in cases where speed and memory are more important than accuracy.

V. CONCLUSION

This paper presents ALGO-X, a novel graph traversal framework designed to optimize both time and space complexity in large-scale and complex graph structures. Through the integration of adaptive heuristics and dynamic path pruning techniques, ALGO-X effectively mitigates the inefficiencies commonly encountered in traditional algorithms such as DFS and BFS. Theoretical analysis demonstrates that ALGO-X offers significant improvements in worst-case and average-case complexity bounds, surpassing the performance of classical traversal algorithms. Experimental evaluations across various benchmark graph datasets ranging from sparse to dense graphs highlight ALGO-X's superior runtime efficiency and reduced memory usage, particularly in graphs with high connectivity or irregular structures. These findings suggest that ALGO-X is a robust and scalable solution for graph traversal problems in real-world applications. Furthermore, the framework's extensibility to specific graph-related tasks, such as shortest path computation and cycle detection, positions it as a versatile tool for a wide range of computational challenges. Future work will focus on exploring parallelization strategies to further enhance ALGO-X's scalability and applicability in distributed computing environments. Ultimately, ALGO-X contributes to the theoretical and practical advancements in graph algorithms, offering promising directions for future research and application in scalable, large-scale graph analysis.

CRediT Author Statement

The author reviewed the results and approved the final version of the manuscript.

Data Availability

The datasets generated during the current study are available from the corresponding author upon reasonable request.

Conflicts of Interests

The authors declare that they have no conflicts of interest regarding the publication of this paper.

Funding

No funding was received for conducting this research.

Competing Interests
The authors declare no competing interests.

References
[1]. Y. Wang et al., "NDSEARCH: Accelerating Graph-Traversal-Based Approximate Nearest Neighbor Search through Near Data Processing," 2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA), pp. 368–381, Jun. 2024, doi: 10.1109/isca59077.2024.00035.
[2]. Aghsal Dwi Putra, A. Nur Maulana, S. Billa Febrianti, N. Camelia, S. Kaka Hutagalung, and A. Naseh Khudori, "Comparison of Breadth-First Search (BFS) and Depth-First Search (DFS) Algorithms for Shortest Search in Campus Labyrinth," Journal of Enhanced Studies in Informatics and Computer Applications, vol. 2, no. 2, pp. 43–51, Jul. 2025, doi: 10.47794/jesica.v2i2.14.
[3]. Ivanochko, M. G. ml, and S. Treľová, "Optimizing the 15 Puzzle with AI: Comparative analysis of breadth-first and depth-first search algorithms," Procedia Computer Science, vol. 265, pp. 57–64, 2025, doi: 10.1016/j.procs.2025.07.156.
[4]. Nauck, M. Lindner, K. Schürholt, and F. Hellmann, "Toward dynamic stability assessment of power grid topologies using graph neural networks," Chaos: An Interdisciplinary Journal of Nonlinear Science, vol. 33, no. 10, Oct. 2023, doi: 10.1063/5.0160915.
[5]. Z. Sun, Y. Mo, and C. Yu, "Graph-Reinforcement-Learning-Based Task Offloading for Multiaccess Edge Computing," IEEE Internet of Things Journal, vol. 10, no. 4, pp. 3138–3150, Feb. 2023, doi: 10.1109/jiot.2021.3123822.
[6]. E. Ferrentino, "A Dynamic Programming Framework for Optimal Planning of Redundant Robots Along Prescribed Paths With Kineto-Dynamic Constraints_supp1-3330371.mp4", doi: 10.1109/tase.2023.3330371/mm1.
[7]. C. Scoccia, G. Palmieri, M. C. Palpacelli, and M. Callegari, "A Collision Avoidance Strategy for Redundant Manipulators in Dynamically Variable Environments: On-Line Perturbations of Off-Line Generated Trajectories," Machines, vol. 9, no. 2, p. 30, Feb. 2021, doi: 10.3390/machines9020030.
[8]. R. AlKahtani, A. Alhabdan, M. Alosami, W. Alshammari, A. A. Abdo, and L. Hamdi, "Comparative Analysis between BFS and DFS-Shortest Path Algorithms," 2025 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream), pp. 1–5, Apr. 2025, doi: 10.1109/estream66938.2025.11016860.
[9]. R. Mohammed N. and J. Zeyad, "Empirical study prove that breadth-first search is more effective memory usage than depth-first search in frontier boundary cyclic graph," IAES International Journal of Artificial Intelligence (IJ-AI), vol. 10, no. 2, p. 265, Jun. 2021, doi: 10.11591/ijai.v10.i2.pp265-272.
[10]. J. Wang, Y. Xie, S. Xie, and X. Chen, "Cooperative particle swarm optimizer with depth first search strategy for global optimization of multimodal functions," Applied Intelligence, vol. 52, no. 9, pp. 10161–10180, Jan. 2022, doi: 10.1007/s10489-021-03005-x.
[11]. L. Hao, Y. Wang, Y. Bai, and Q. Zhou, "Energy management strategy on a parallel mild hybrid electric vehicle based on breadth first search algorithm," Energy Conversion and Management, vol. 243, p. 114408, Sep. 2021, doi: 10.1016/j.enconman.2021.114408.
[12]. L. Liu, B. Wang, and H. Xu, "Research on Path-Planning Algorithm Integrating Optimization A-Star Algorithm and Artificial Potential Field Method," Electronics, vol. 11, no. 22, p. 3660, Nov. 2022, doi: 10.3390/electronics11223660.
[13]. Q. Wang, Y. Hao, and J. Cao, "Learning to traverse over graphs with a Monte Carlo tree search-based self-play framework," Engineering Applications of Artificial Intelligence, vol. 105, p. 104422, Oct. 2021, doi: 10.1016/j.engappai.2021.104422.
[14]. H. Ootomo, A. Naruse, C. Nolet, R. Wang, T. Feher, and Y. Wang, "CAGRA: Highly Parallel Graph Construction and Approximate Nearest Neighbor Search for GPUs," 2024 IEEE 40th International Conference on Data Engineering (ICDE), pp. 4236–4247, May 2024, doi: 10.1109/icde60146.2024.00323.
[15]. K. Figueroa and R. Paredes, "Approximate Direct and Reverse Nearest Neighbor Queries, and the k-nearest Neighbor Graph," 2009 Second International Workshop on Similarity Search and Applications, pp. 91–98, Aug. 2009, doi: 10.1109/sisap.2009.33.
[16]. R. Xu, S. Yao, W. Gouhe, Y. Zhao, and S. Huang, "A spanning tree algorithm with adaptive pruning with low redundancy coverage rate," Computers & Industrial Engineering, vol. 186, p. 109745, Dec. 2023, doi: 10.1016/j.cie.2023.109745.
[17]. F. Liu, W. Zhao, Y. Chen, Z. Wang, and F. Dai, "DynSNN: A Dynamic Approach to Reduce Redundancy in Spiking Neural Networks," ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 2130–2134, May 2022, doi: 10.1109/icassp43922.2022.9746566.
[18]. Y. Kim and P. Panda, "Optimizing Deeper Spiking Neural Networks for Dynamic Vision Sensing," Neural Networks, vol. 144, pp. 686–698, Dec. 2021, doi: 10.1016/j.neunet.2021.09.022.
[19]. H. N. A. Hamed, N. Kasabov, and S. M. Shamsuddin, "Dynamic Quantum-Inspired Particle Swarm Optimizationas Feature and Parameter Optimizer for Evolving Spiking Neural Networks," International Journal of Modeling and Optimization, pp. 187–191, 2012, doi: 10.7763/ijmo.2012.v2.108.
[20]. D. Fan, H. Cao, G. Wang, N. Nie, X. Ye, and N. Sun, "Scalable and efficient graph traversal on high-throughput cluster," CCF Transactions on High Performance Computing, vol. 3, no. 1, pp. 101–113, Nov. 2020, doi: 10.1007/s42514-020-00056-3.
[21]. Y. Yasui, K. Fujisawa, E. L. Goh, J. Baron, A. Sugiura, and T. Uchiyama, "NUMA-aware Scalable Graph Traversal on SGI UV Systems," Proceedings of the ACM Workshop on High Performance Graph Processing, pp. 19–26, May 2016, doi: 10.1145/2915516.2915522.
[22]. L. Wang, X. Dong, Y. Gu, and Y. Sun, "Parallel Strong Connectivity Based on Faster Reachability," Proceedings of the ACM on Management of Data, vol. 1, no. 2, pp. 1–29, Jun. 2023, doi: 10.1145/3589259.
[23]. D. Chakraborty and K. Choudhary, "New Extremal bounds for Reachability and Strong-Connectivity Preservers under failures," ACM Transactions on Algorithms, Mar. 2025, doi: 10.1145/3720545.
[24]. S. Sundarraj, R. V. K. Reddy, M. B. Basam, G. H. Lokesh, F. Flammini, and R. Natarajan, "Route Planning for an Autonomous Robotic Vehicle Employing a Weight-Controlled Particle Swarm-Optimized Dijkstra Algorithm," IEEE Access, vol. 11, pp. 92433–92442, 2023, doi: 10.1109/access.2023.3302698.
[25]. N. S. Abu, W. M. Bukhari, M. H. Adli, S. N. Omar, and S. A. Sohaimeh, "A Comprehensive Overview of Classical and Modern Route Planning Algorithms for Self-Driving Mobile Robots," Journal of Robotics and Control (JRC), vol. 3, no. 5, pp. 666–678, Sep. 2022, doi: 10.18196/jrc.v3i5.14683.

ISSN (Online): 3105-9082