Comparative Evaluations of Pathfinding Techniques in Undirected Graphs

Ganesh Gundekarla
University of North Texas
11700551

Shreya Sri Bearelly
University of North Texas
11734229

Divya Sree Dandu
University of North Texas
11594287

Vikas Varala
University of North Texas
11704793

Ashritha Bollam
University of North Texas
11704793

***Abstract*—When it comes to delivering the best pathfinding approaches within undirected graphs, algorithmic designing uncovers the special feature in directed tasks. The objective behind this survey project is to investigate various algorithmic frontiers that might be used for evaluating various algorithms. In contrast to Dijkstra algorithms, it is clear that the newest algorithms, such as A* searches and Ant Colony Optimization (ACO), are more beneficial, particularly when it comes to figuring out how to navigate undirected graphs. The survey aims to use performance assessments and logical judgment to arrive at the optimum pathfinding technique for graphs that is applicable to real-world applications.Our goal is to examine and debate the most recent algorithms that have been developed in this area, as well as provide guidance for systematization and the development of researcher navigational aids. Furthermore, this initiative intends to extend the scope of algorithm evolution by putting forward historical studies for potential future innovations in graph theory frontiers. This has had a significant impact on the algorithms' performance and enabled them to take on challenging real-world issues.**

***Index Terms*—Dijkstra Algorithm, Ant Colony Optimization, A* search, Single-Source Shortest Path, Graph Algorithms, Breadth first search,**

I. Introduction

Algorithmic design for undirected graphs involves tackling the challenges in the field of modern computing tasks through computational science. In order to address these difficulties, scientists have researched and created various algorithm frameworks that are utilized in areas such as networking and robotics route. The researchers have demonstrated advanced methods like A* search and Ant Colony Optimization. The survey project focuses on investigating the latest advancements in pathfinding algorithms to tackle the challenges of undirected graphs' complexity. The literature review of several studies is intended to offer a thorough comprehension of the landscape, critical contributions to attaining the best solutions for undirected graphs in practical situations. Comparison of various algorithms such as Breadth-first search, A*, ACO, and Dijkstra algorithm have demonstrated their efficiency, computational effectiveness, and compromises. Through the examination of these algorithms, the researchers gain a comprehension of the most basic pathfinding method for each approach examined.

Identify applicable funding agency here. If none, delete this.

II. Defining the Objectives

The primary goal in our survey study is to carry out a detailed and thorough comparison of the different methods utilized to navigate undirected graphs. Our specific focus is on:

- Examine the computational efficiency of algorithms by considering the runtime complexity and other relevant metrics.
- Examine and contrast the effectiveness of different algorithms, including Linear-algebraic approaches like breadth first search (BFS), A* search, Ant Colony Optimization (ACO), and traditional algorithms such as Dijkstra, along with newer randomized algorithms.
- Assess each algorithm's advantage and disadvantages determining which role each should play in generating various pathfinding solutions.

III. Literature Review

A. Survey Report A Randomized Algorithm for Single- Source Shortest Path on Undirected Real- Weighted Graphs

The basic idea of shortest path problems in graph theory is used in network routing, social network analysis, and other fields. To find a way to address the SSSP challenges, this paper has put a lot of effort on the Dijkstra's algorithm, its revisited variances, and it's related to the recent developments. The single-source shortest path (SSSP) main purpose is to find the shortest path between each vertex in the undirected graph and every other vertex possible. The study has showed that when addressing the single source shortest path (SSSP) challenge, Dijkstra's algorithm is a mostly used for determining the shortest path to take from a given source to all destinations in the graph. As long as the alternative vertices remain unvisited, it computes this process using an update step loop that includes an arm to those vertices with a lower distance value.

The algorithm's priority queue, which is usually implemented using Fibonacci heap [2] or relaxed heap [3]data structures, ranks users according to their distance from one

another. The Dijkstra algorithm has an *O (m + n log n)* time complexity, where m *and n* are the edges and vertices, consecutively, and it includes addition and comparison procedures for real-number inputs for the method used. For undirected networks, Pettie and Ramachandran [4] suggest an SSSP algorithm with a temporal complexity of *O (m (n, m) + min n log n, n log log r)*, where r is the edge weight ratio and is an inverted Ackerman function. For real-weighted networks without ratio constraints, however, no Single-Source Shortest Path (SSSP) technique with a temporal complexity of *O (m + n log n)* has been defeated. The primary approach uses partitioned bundles and *set R*; the pseudocode for this technique is shown below.

For the first vertex *v*, we set *d(s) to 0 and d(v) to* . Next, we use Fibonacci numbers to incorporate every vertex in *R* in a heap structure. Every time we remove a vertex *(u)* from the heap in *set R*, we adjust the distances in accordance with the procedures that follow. *Vertex v's* value can be updated by *D* by relaxing it and adding *D to d(v)*. To find the exact distance from vertex *s* to every vertex related to *u*, compute the minimum value between the distances of *v and D*.

- To update the shortest distance to vertex *v*, first add the distance from *u to v*. Then, for each vertex *y* within the ball centered at *v*, add the distance from *y to v*. Update each vertex's shortest distance to v at the end. *Z1* is located in *N(z2)*, while *Z2* is found in Ball*(v)* or equal to v. Next, use *d(z1) + wz1, z2, and dist (z2, v) to relax v*. Put differently, we updated vertex *v* with its related vertex *u*, *Ball(v)* vertices, and vertices that are close to both *v and Ball(v)*.
- After we have changed the function *d(x)* for every *x* in *Bundle(u)*, we update the vertices z1 inside the ball centered at y as well as the vertices y that are near to it. This is adding *d(x) + wx, y* for each *y in N(x)* to update the value of *y*, and adding *d(x) + wx, y + dist (y, z1)* for each *z1 in Ball(y)* to update the value of *z1*.
- We add the value of *d(v) + dist (v, b(v))* to update the connected vertex *b(v)* when a vertex *v* outside of *set R* is modified. However, as *v* is combined with *u* in Step 1 and the distance *dist (s, u)* is already determined when u is removed from the heap, it will be discovered that this is only necessary in Step 2 and not Step 1.

B. Survey Report "A Comparative Analysis Among Three Different Shortest Path-Finding Algorithms"

Many research projects and evaluations have successfully incorporated the A* search method. The A* search algorithm was used by Kusuma et al. [5] to determine a humanoid robot's path. They were able to organize and modify plans in order to successfully accomplish the intended results. Additionally, the suggested approach showed to be effective for unmanned civilian devices. Yao et al. improved the A* search algorithm by employing a weighted cost function to develop a method for optimal path planning. This was covered in depth in the research done by Tseng et al. [6] and used to 3G communication-dependent aerial vehicle (UAV) trajectory planning. The following is expressed by the formula *fw(x) = (1 w) g(x) + wh(x)* [7]. Even though this is merely a small modification to the formula, it makes a difference when compared to the original A* algorithm. According to the study, the modified algorithm exhibits significant gains in iteration and execution time without compromising the system's acceptability.

The ACO technique is a tool with a wide variety of uses, one of which is figuring out the optimal approach in environments where only some information is available. Ants communicate by laying down pheromone trails that help them navigate and stay connected with each other. By imitating the behavior of ants, ACO tools can identify the fastest pathway agents use to move. In combine, the drawing feature is valuable for its ability to navigate and route in various ways. One of the successes that have been highlighted is the successful resolution of the shift scheduling problem in the nursing department. An advanced inquiry carried out at the Vienna Hospital facility utilizing the ACO algorithm [8] provided management with an opportunity to potentially reschedule activities in dynamic areas. ACO is superior to traditional structures for solving graph colouring problems. [9] Salari and Eshghi showed that the ACO algorithm can be used effectively in this task, indicating its success in both colouring objects/vertices in a graph and minimizing conflicts in such instances. It highlights the algorithm's universal features, allowing it to handle a broad spectrum of combinatorial optimization problems. ACO's versatility is proven when it efficiently solves complex real-life resource scheduling problems, showcasing its capabilities beyond just navigation tasks. Advanced Binary ACO (ABACO) is a modified variant of the ACO method that Kashef et al. [10] developed. This version brings about improvements that lead to better performance in multiple optimization tasks. ABACO's achievements highlight the continuous work to improve and develop ACO strategies for better outcomes in various problem fields.

Wang first implemented the Dijkstra algorithm [11] for the maze path traversal. It's most likely that the goal of this experiment was to have everyone navigate a maze in the shortest amount of time. The fastest route must be found in order to do it. The Dijkstra algorithm was used by researchers to determine the robot's path, which would take it to its goal in the least amount of time while traveling and encountering the fewest obstacles possible. The created algorithm in [12] converges with a modification of the well-known Dijkstra algorithm, which was required to regulate a specific vehicle's GPS navigation. This demonstrates that making modifications to the conventional algorithms is probably the right course of action to guarantee overcoming constraints like time constraints and the requirement for immediate orienting. In the work, the Dijkstra algorithm was modified to maximize both the passing of time intervals and the changes in route conditions. For the first time, Kang et al. [13] used the Dijkstra algorithm to create PSO (Particle Swarm Optimization), offering a novel approach to solving path planning problems. PSO is a population optimization method that you can use in accordance with how

fish in schools or birds in flocks behave. By enhancing the PSO with Dijkstra's algorithm and applying the concepts of swarm intelligence, researchers hoped to create the algorithm that determines the shortest path.

C. Survey Report Reformulating A Breadth- First Search Algorithm on An Undirected Graph in The Language of Linear Algebra

Graph algorithms are gaining importance in the fields of computational science and engineering as data intensive applications become more widespread in industry, science, and society. The scientific community and general public are becoming more interested in big data, which is the rapid expansion and accessibility of digital information. Graph algorithms are clearly essential for big data analysis, regardless of the data's structure. Data structures based on pointers or linked lists are frequently used in graph algorithms [14], [15]. It is widely believed that these data structures best illustrate the data and control flow complexity of various graph algorithms. It takes a lot of time and effort to code and maintain these sophisticated data structures, and creating working versions of them could lead to performance issues on existing computer systems with complicated memory architectures. Graph algorithms' performance has recently been targeted for improvement through the use of novel data structures and techniques. Novel methods exist in the field for using linear algebra operations to express graph algorithms [16]. Although graph theory has a long history of application in sparse linear algebra, graph algorithms have not typically focused on employing linear algebra.

IV. Analysis and Implications

A. Innovative Methodology

These approaches are the main divergences from the conventional graph algorithmic strategies used in data mining applications found in the surveyed papers. By adding a dash of linear algebra to the BFS algorithm modification, Survey 3 sets itself apart from the others opening up the possibility of combining numerical design with graph theory. This novel method not only allows for speedier calculation but also opens doors for techniques involving sparse matrix-vector products. A random approach for the SSSP problem is introduced to us in Survey 1. These algorithms are outside the confines of computational theory, and they defy the conventional wisdom that most of us were raised with when it comes to algorithm discovery.

B. Performance Evaluation and Algorithmic Trade-offs

The study carried out as survey 2 is highly informative in that it tests several path algorithms using a variety of variables. Even when compared to other algorithms in use, A* Search performs best in terms of complexity and distance while having the worst performance in terms of metrics like runtime. By this assessment, we suggest that algorithms in general have limits and emphasize the need for them to take into account a wide range of factors when intended to match a specific task. The researchers can use the research findings to determine which algorithm will work best for their problem area, taking into account their timeline and budgetary constraints.

Topic	ACO algorithm	A* Search Algorithm	Dijkstra Algorithm
Runtime	16 sec.	1.40 sec.	1.70 sec.
Diagonal Movement	1	4	4
Time Complexity	Comparatively more complex	$O(b^d)$	$O((V+E)logV)$
Distance	12.4 cells	9.242 cells	9.242 cells
Number of turns	7	2	2
Number of visited Nodes	0	0	51

TABLE I
Algorithm Comparison Results

C. Time Complexity Improvement

The researchers in Survey 1 introduced an unpredictability method to address the SSSP problem, although at the expense of increased algorithmic time complexity. Our algorithm processes information more efficiently than various conventional methods (like Dijkstra's), and it also outperforms time complexity, surpassing state-of-the-art computational limitations and suggesting the possibility of increased graph algorithm development efficiency. Furthermore, investigating different approaches to problem solutions through flexibility indicates that the research will be extremely important and will provide insights into the graph problem from a variety of perspectives.

D. Insights and Future Directions

The utilization of linear algebra principles in designing graph algorithms, as evidenced in survey 3, underscores the benefits of collaborating across disciplines to progress algorithmic research. By connecting various academic disciplines - like numerical computation and graph theory - researchers can utilize knowledge and techniques from a variety of areas to create stronger and more effective algorithms. This interdisciplinary method enhances the theoretical basis of graph algorithms and broadens the scope of graph-based computations in addressing various real-world issues. In the future, upcoming studies in graph algorithmic could still gain advantages from interdisciplinary perspectives, leading to creative approaches for intricate computational problems.

V. Results and Discussions

Table I displays the comparison findings of the three algorithms for solving problems. ACO Algorithm requires a longer run-time taken for execution is lower with A* search compared to other algorithms. In terms of difficulty, the Dijkstra it is easier to implement the algorithm. When traveling long distances The ACO algorithm utilizes greater distance and a higher number of turns and more rotations than the other two algorithms. In this experiment that has been shown, the A* search and Dijkstra algorithms are used to determine same number of turns. Shortest paths with equal distance ACO and the final parameter (number of visited nodes), A* does not explore any additional cells beyond the paths it has chosen therefore yielding same results. In contrast, Dijkstra shows a considerable amount (approximately 51) of cells that were visited.

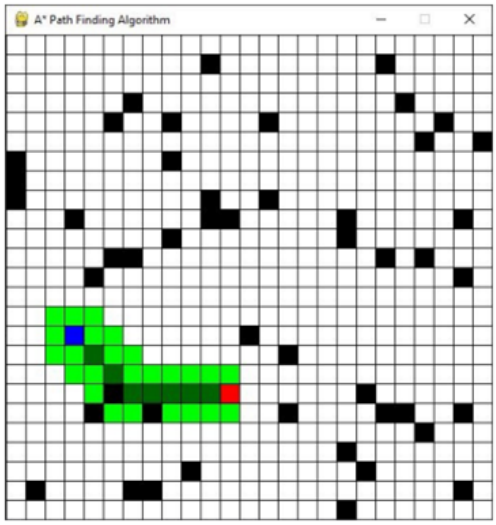

Fig. 1. A* search Result

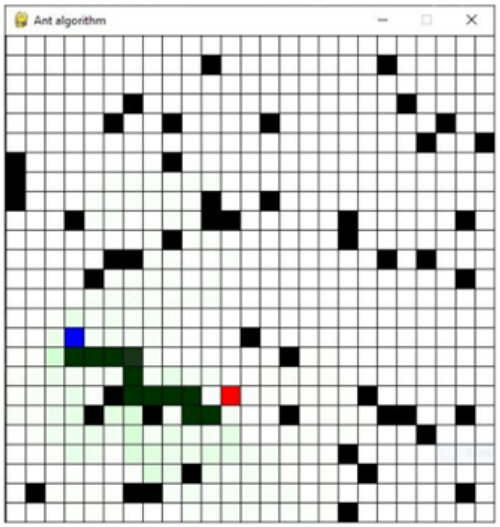

Fig. 2. ACO Results

In Fig 1and Fig 2, the blue cell is labeled as the beginning point while the red cell is labeled as the final point on the grid. The dark green line represents the selected route on the grid. Figure 3 show light green grids for opened nodes and yellow grids for visited nodes. All three algorithms operate within the same grid environment.

Figure 4 displays the BFS algorithm applied to a graph example. BFS starts at vertex s = 1, with the yellow edges showing the tree where s is the main node. At first, the vertices one unit away from s are recognized. The first vertices to visit are the ones adjacent to s, which are 4, 5, and 10. Following that, vertices 2, 7, and 8 are revealed, each having a distance of 2. To sum up, BFS explores vertices 3, 6, and 9, which are all situated 3 units away from s. The diagram separates edges into two sets: those included in the breadth-first search (yellow edges) and those that are not (black edges).

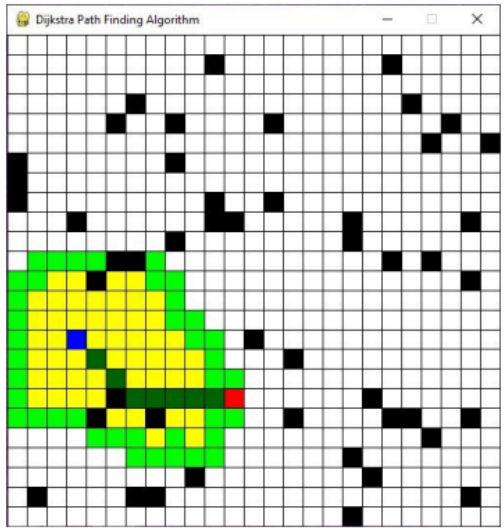

Fig. 3. Dijkstra Algorithm Results

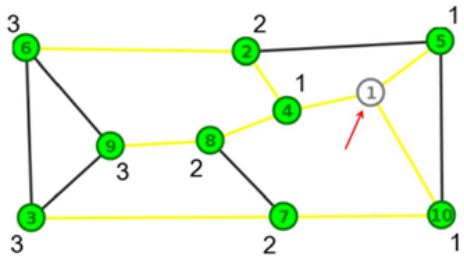

Fig. 4. BFS Algorithm Implemented on a Sample Graph

VI. Conclusion

The reviewed papers demonstrate the ongoing development of graph algorithm research by presenting new methodologies and algorithms that tackle key problems in graph theory. The inclusion of linear algebra methods in developing graph algorithms, showcased in the initial paper, indicates potential ways to improve algorithm efficiency. The comparison of shortest path-finding algorithms offers useful information for choosing algorithms for different problem scenarios, aiding in making informed decisions during algorithm development. The development of a method randomized algorithm for SSSP problems represents a significant breakthrough in addressing computational hurdles associated with traditional approaches. In general, these research studies demonstrate the ever-changing scope of graph algorithm research and its continuous importance in various application fields.

Acknowledgment

We are extremely grateful to our project supervisor for their guidance, support, and motivation throughout every step of the research process. Their expertise and direction have been extremely important in shaping the direction and caliber of our work. We want to express our gratitude to each and every member of our team for their dedication, cooperation,

and significant input during the survey process. We also wish to recognize the writers of the survey reports and research papers that were essential to our study. Their input in the field of graph algorithm research has been extremely valuable in guiding our analysis and discussions.

REFERENCES

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

[2] 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.

[3] Driscoll, J. R., Gabow, H. N., Shrairman, R., Tarjan, R. E. (1988). Relaxed heaps. Commun. ACM;(United States), 31(11).

[4] Pettie, S., Ramachandran, V. (2005). A shortest path algorithm for real-weighted undirected graphs. SIAM Journal on Computing, 34(6), 1398-1431.

[5] Kusuma, M., Machbub, C. (2019, July). Humanoid robot path planning and rerouting using A-Star search algorithm. In 2019 IEEE International Conference on Signals and Systems (ICSigSys) (pp. 110-115). IEEE.

[6] Tseng, F. H., Liang, T. T., Lee, C. H., Der Chou, L., Chao, H. C. (2014, August). A star search algorithm for civil UAV path planning with 3G communication. In 2014 Tenth International Conference on Intelligent Information Hiding and Multimedia Signal Processing (pp. 942-945). IEEE.

[7] Yao, J., Lin, C., Xie, X., Wang, A. J., Hung, C. C. (2010, April). Path planning for virtual human motion using improved A* star algorithm. In 2010 Seventh international conference on information technology: new generations (pp. 1154-1158). IEEE.

[8] Gutjahr, W. J., Rauner, M. S. (2007). An ACO algorithm for a dynamic regional nurse-scheduling problem in Austria. Computers Operations Research, 34(3), 642-666.

[9] Salari, E., Eshghi, K. (2005, December). An ACO algorithm for graph coloring problem. In 2005 ICSC Congress on computational intelligence methods and applications (pp. 5-pp). IEEE.

[10] Kashef, S., Nezamabadi-pour, H. (2015). An advanced ACO algorithm for feature subset selection. Neurocomputing, 147, 271-279.

[11] Wang, H., Yu, Y., Yuan, Q. (2011, July). Application of Dijkstra algorithm in robot path-planning. In 2011 second international conference on mechanic automation and control engineering (pp. 1067-1069). IEEE.

[12] Noto, M., Sato, H. (2000, October). A method for the shortest path search by extended Dijkstra algorithm. In Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0 (Vol. 3, pp. 2316-2320). IEEE.

[13] Kang, H. I., Lee, B., Kim, K. (2008, December). Path planning algorithm using the particle swarm optimization and the improved Dijkstra algorithm. In 2008 IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application (Vol. 2, pp. 1002-1004). IEEE.

[14] Hopcroft, J. E., Ullman, J. D., Aho, A. V. (1983). Data structures and algorithms (Vol. 175). Boston, MA, USA: Addison-wesley.

[15] Gloor, P., Dynes, S., Lee, I. Animated Algorithms: A Hypermedia Learning Environment for "Introduction to Algorithms".

[16] Kepner, J., Gilbert, J. (Eds.). (2011). Graph algorithms in the language of linear algebra. Society for Industrial and Applied Mathematics.