



National Institute of Technology
Tiruchirappalli

ADVANCE DATA STRUCTURES AND ALGORITHMS

Topic-Shortest distance between two points using
bidirectional Dijkstra.

Report under Professor :-

Dr. B. Nithya(Assistant Professor at NIT Trichy)

Degree Name- B. Tech.

Department -Computer Science

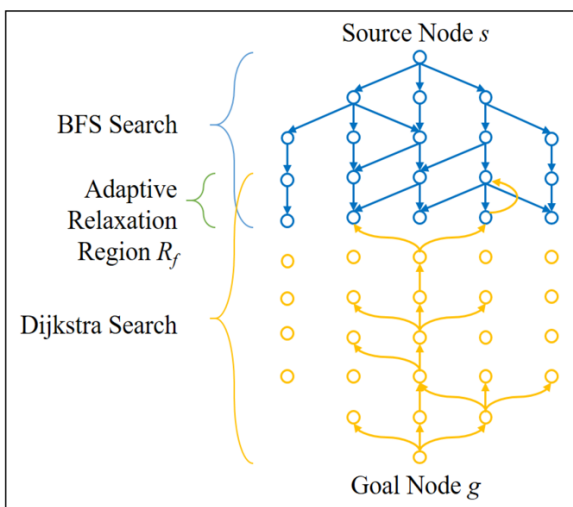
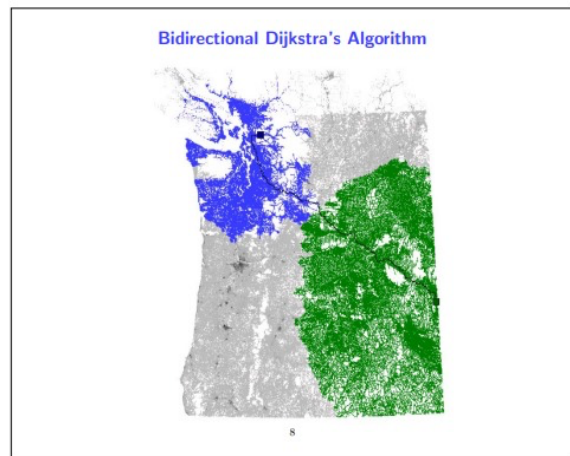
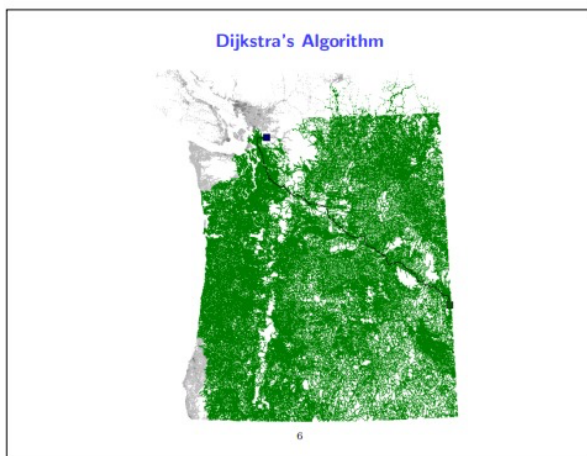
[Nikhil Dhalan \(106122083\)](#)

[Nitin\(106122085\)](#)

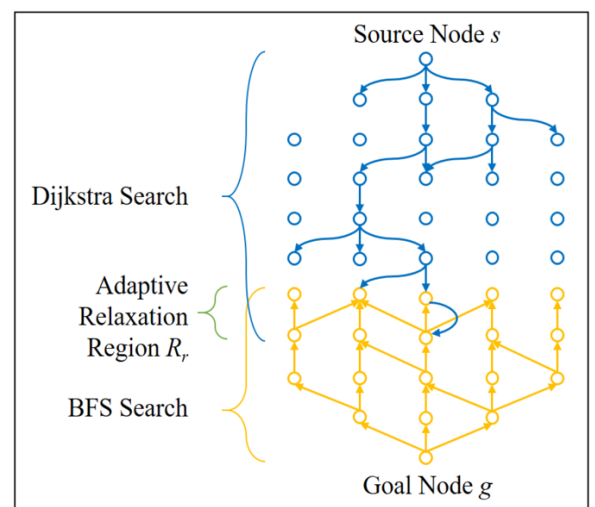
Explaining the Performance of Bidirectional Dijkstra Road Networks

Abstract

Dijkstra's algorithm is well known shortest path finding algorithm in a graph. But there exists Bidirectional Dijkstra algorithm as well which runs two simultaneous searches, one in the forward direction from source and the other in the backward direction from destination until they meet eventually.



(a) Forward hybrid search



(b) Reverse hybrid search

1. Overview

The proposed study seeks to explore and enhance the Bidirectional Dijkstra algorithm, a variant of the classical Dijkstra's algorithm used to determine the shortest path between two nodes in a graph. Unlike its traditional counterpart that progresses from a single source towards the destination, the Bidirectional Dijkstra algorithm simultaneously executes two searches: one from the source node and another from the target node, meeting somewhere in the middle. This -direction approach aims to potentially reduce the search space and computational time required, making it highly suitable for large-scale applications where efficiency is paramount.

The overarching strategy of this research is to both theoretically and experimentally analyse the performance improvements offered by the Bidirectional Dijkstra algorithm over the traditional Dijkstra's algorithm, especially in complex network topologies commonly found in urban traffic systems and large data networks. Through a combination of rigorous algorithmic enhancements, performance benchmarking, and application in simulated real-world scenarios, the study aims to provide a comprehensive evaluation of the algorithm's effectiveness and areas for potential optimization.

The context set by this overview guides the subsequent sections, where the specific research design is detailed, followed by methodologies for data collection and analysis, considerations of limitations and assumptions, and the justification of the chosen research methods. This approach

not only aligns with the academic pursuit of enhancing algorithmic efficiency but also caters to practical implementations in various industrial applications.

2. Research Design

The research will adopt an experimental design to rigorously test and evaluate the performance of the Bidirectional Dijkstra algorithm. This method is chosen because it allows for controlled experimentation where variables can be manipulated to observe their effect on the algorithm's efficiency, accuracy, and scalability. Such a design is essential for a quantitative assessment of algorithmic improvements and their impacts under various controlled scenarios, making it ideal for the objectives of this study which focus on optimizing pathfinding algorithms.

Algorithm: Bidirectional Dijkstra

Initialization: Start with two priority queues, one for nodes visited from the source and one from the destination.

Search Execution: Run simultaneous searches from both nodes. When expanding a node, check if it has been reached by the opposite search.

Meeting Point: Once a node is visited by both searches, calculate potential shortest paths.

Termination: The algorithm terminates when the shortest path is identified or when no further nodes are left to explore without exceeding known shortest paths.

Path Reconstruction: Construct the shortest path by combining paths from both searches at the meeting node.

This experimental approach, supported by quantitative metrics, will highlight the efficiency gains and practicality of implementing the Bidirectional Dijkstra in diverse settings.

3. Data Analysis Procedures

The analysis of data collected during the experimental trials of the Bidirectional Dijkstra algorithm will primarily focus on quantitative metrics, such as runtime efficiency, path optimality, resource usage, and scalability across different network sizes and topologies. Here's how the data will be analysed to address the research questions and objectives:

Statistical Analysis

Descriptive Statistics: Initially, mean, median, and standard deviation for runtime and other performance metrics will be calculated to provide a basic understanding of the algorithm's performance under various conditions.

Inferential Statistics: Conduct hypothesis testing (e.g., t-tests) to compare the performance of the Bidirectional Dijkstra algorithm against the traditional Dijkstra algorithm. This will help determine if the observed improvements are statistically significant.

Regression Analysis: To understand how different factors such as graph size and density affect the algorithm's performance, regression models may be employed.

Comparative Analysis

Benchmarking: The Bidirectional Dijkstra's performance will be benchmarked against other pathfinding algorithms. This involves comparing efficiency, accuracy, and resource consumption under identical conditions.

Effect Size Calculation: Calculate the effect size to measure the practical significance of the algorithm's performance improvements.

Analytical Tools and Software

Python: For implementation and testing of the algorithm. Python is chosen due to its extensive library support and robust data handling capabilities.

NumPy and SciPy: For efficient numerical computations and more advanced statistical functions.

Matplotlib and Seaborn: To create visualizations for the performance data, such as histograms, scatter plots, and line graphs that compare the various metrics across different algorithms.

Pandas: For data manipulation and analysis, especially useful for handling large datasets and preparing them for analysis.

Data Visualization

Graphs and Charts: Various graphs such as line graphs, bar charts, and scatter plots will be used to illustrate the algorithm's performance trends and comparisons.

Heat Maps: To show the performance impact of different parameters such as graph density and node count.

These data analysis procedures are designed to be comprehensive and robust, allowing not only for an assessment of the Bidirectional Dijkstra algorithm's

performance but also for insights into how various factors influence its efficiency. This systematic approach will provide a solid foundation for evaluating the proposed enhancements to the algorithm and offer conclusive evidence regarding its applicability and effectiveness in different scenarios.

4.Limitations and Assumptions

Limitations

Graph Variability: The performance of the Bidirectional Dijkstra algorithm can vary significantly across different types of graphs (e.g., sparse vs. dense, weighted vs. unweighted). This variability might limit the generalizability of the results to only certain types of networks.

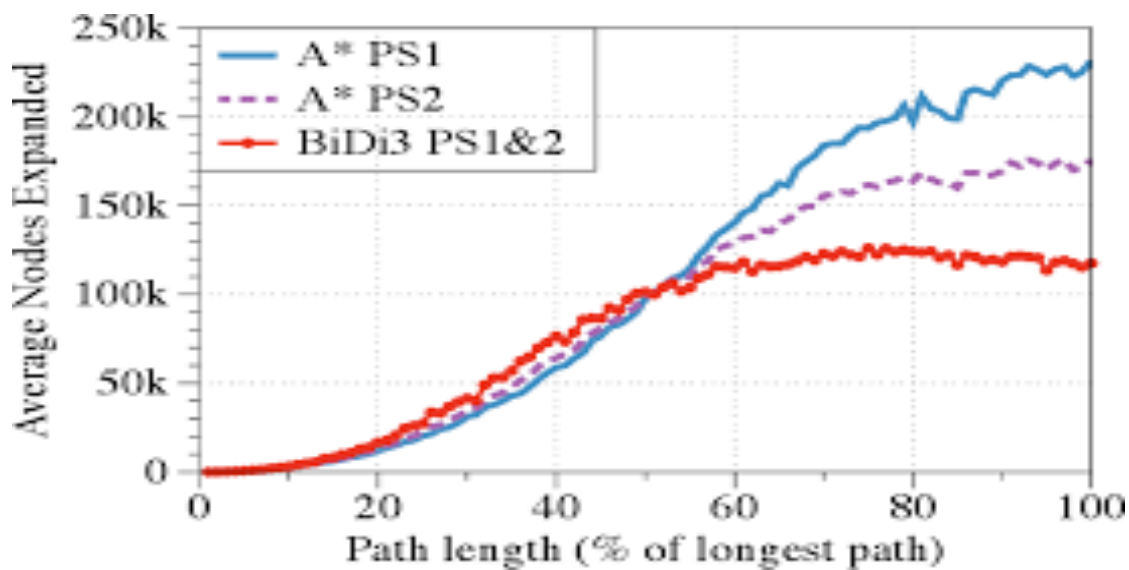
Algorithm Complexity: The complexity of implementing a bidirectional search strategy might introduce errors or inefficiencies not present in simpler algorithms, potentially complicating the analysis and interpretation of results.

Sample Size and Diversity: The experimental design might use a limited range of graph configurations, which may not fully represent the diverse real-world scenarios where the algorithm could be applied. This could impact the reliability and applicability of the findings.

Computational Resources: Extensive simulations required for a comprehensive evaluation could be constrained by available computational resources, limiting the scope of testing environments and scenarios.

Assumptions

Graph Integrity: The study assumes that all graphs are well-formed and error-free, which might not always be the case in practical applications.



Uniformity in Node Processing: It is assumed that the computational load for processing each node is consistent across the graph, which might not hold true for networks with highly variable node and edge properties.

Consistent Algorithm Implementation: There is an assumption that the algorithm is implemented consistently across different tests and conditions without any variations that could affect the outcome.

Mitigation Strategies

Extensive Testing Across Graph Types: To address variability,

the algorithm will be tested on a wide variety of graph types, including both synthetic and real-world datasets, to ensure the results are as generalizable as possible.

Error Handling and Validation: Implement rigorous error-checking and validation steps within the algorithm to reduce the likelihood and impact of implementation errors. This includes unit testing of algorithm components and validation against known benchmark problems.

Increasing Sample Size and Diversity: Use a larger and more diverse set of graph samples in the experiments to ensure robustness and applicability of the findings across different scenarios and configurations.

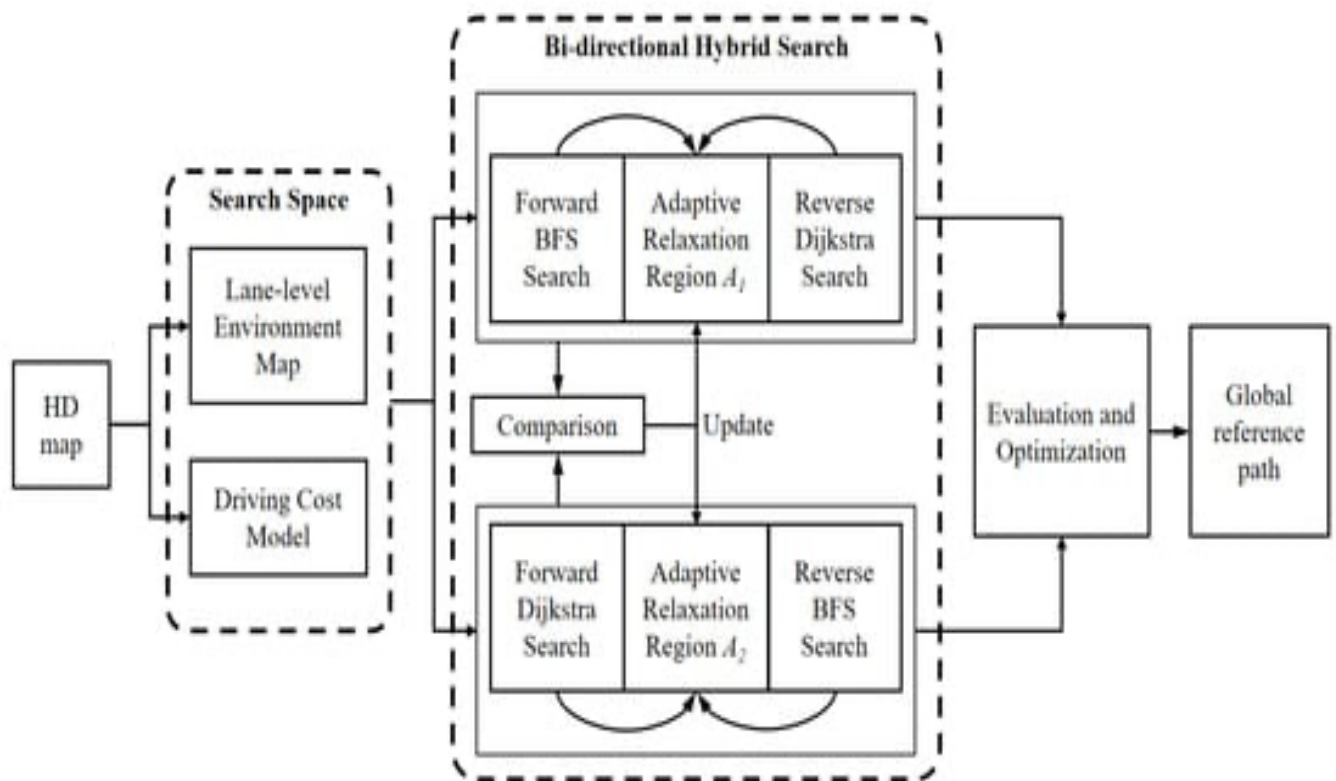
Leveraging High-Performance Computing: Utilize cloud-based or high-performance computing resources to run extensive simulations without being limited by local computational capabilities. This allows for a more thorough and expansive evaluation of the algorithm's performance.

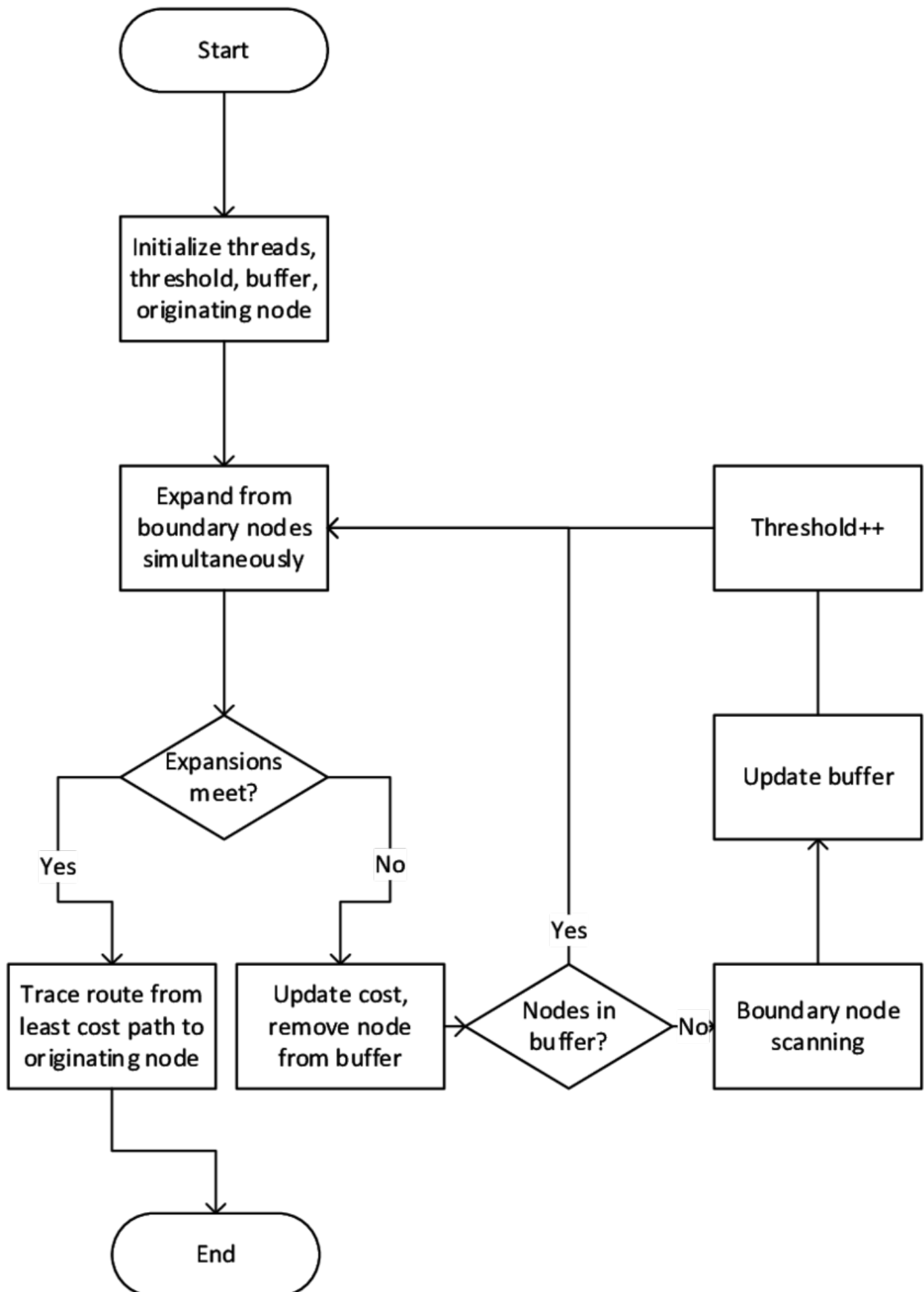
Sensitivity Analysis: Conduct sensitivity analyses to understand how changes in the processing of nodes or variations in graph integrity impact the performance of the algorithm. This helps identify any potential biases or weaknesses in the algorithm's design and implementation.

By acknowledging these limitations and assumptions and implementing these mitigation strategies, the study aims to

enhance the validity and reliability of the results, ensuring that the conclusions drawn from the research are robust and applicable in a variety of settings.

Flowchart for the Bidirectional Dijkstra Algorithm





1.Start: Initiation of the algorithm.

2.Input: Source node (s) and target node (t), graph G with nodes and edges.

3.Initialize Datasets:

4.Create two priority queues: one for the forward search starting from s and one for the backward search from t.

5.Set initial distances: Distance from s to s = 0, distance from t to t = 0. All other distances are set to infinity.

6.Mark all nodes as unvisited.

7.Concurrent Searches:

a. Forward Search:

Extract the node with the minimum distance from the forward priority queue.

Update distances to adjacent nodes if a shorter path is found.

Mark node as visited from the forward direction.

b. Backward Search:

Extract the node with the minimum distance from the backward priority queue.

Update distances to adjacent nodes if a shorter path is found.

Mark node as visited from the backward direction.

Both searches alternate turns, processing nodes until a termination condition is met.

8.Meeting Point Check:

After each search step, check if the current node has been visited from the other direction.

If yes, calculate the tentative shortest path via this meeting point.

Keep track of the shortest path found so far.

9. Termination Condition:

The algorithm stops when the priority queues are empty or when the shortest path confirmed (no possible shorter path exists).

10. Path Reconstruction:

Trace back from the meeting point to the source and from the meeting point to the target using the recorded predecessors from each search direction.

Combine these paths to form the complete shortest path from s to t .

Output Result:

Return the shortest path and its length.

End of the algorithm process.

This flowchart outlines the detailed operations within the Bidirectional Dijkstra algorithm, highlighting the dual-search mechanism and how the paths are constructed and validated. This visual representation can be pivotal in helping readers and stakeholders understand the complexities and efficiency of the algorithm's process in finding the shortest path in a graph.

Justification of the Chosen Methodology

The choice of utilizing the Bidirectional Dijkstra algorithm and an experimental research design is based on specific objectives to enhance the efficiency and applicability of shortest path algorithms in large and complex networks. This section provides the justification for this methodology and discusses how it contributes to the advancement of knowledge in the field of computer science, particularly in graph algorithms.

1. Efficiency in Computation

The Bidirectional Dijkstra algorithm inherently aims to reduce computational overhead by simultaneously conducting searches from both the source and the target, potentially meeting in the middle. This approach is theoretically capable of halving the search space compared to the traditional unidirectional Dijkstra's algorithm, which can be particularly beneficial in densely connected graphs or graphs with large diameters. By focusing on this dual-front exploration, the chosen methodology addresses a crucial research question: can the efficiency of the classic pathfinding algorithm be significantly improved without compromising accuracy?

2. Experimental Research Design

Choosing an experimental design allows for a controlled environment where variables can be systematically manipulated to observe their effects on the performance of the algorithm. This is essential for quantitatively assessing

how different graph characteristics—such as size, density, and weight distribution—affect the algorithm's performance. This design is most suitable because it not only facilitates a direct comparison with the traditional Dijkstra's algorithm but also with other shortest path algorithms, providing a clear performance baseline.

3. Relevance to Real-World Applications

The methodology is aligned with real-world applications such as GPS navigation systems, network routing, and urban planning, where efficient and fast pathfinding algorithms are crucial. By enhancing the Bidirectional Dijkstra algorithm, the study directly contributes to improving the operational efficiency of these critical systems, thereby having a practical impact beyond theoretical advancements.

4. Building Upon Existing Research

This research builds upon existing studies by not only optimizing an already established algorithm but also by applying modern computational techniques and tools to explore its limits and capabilities in new scenarios. It contributes to the literature by providing updated, empirical evidence of the algorithm's performance under varying conditions, and by potentially uncovering new insights into its scalability and efficiency.

5. Advancement of Knowledge

The proposed study contributes to the advancement of knowledge by deepening the understanding of how

bidirectional search strategies can be optimized for different types of graphs and real-world scenarios. Additionally, by documenting the conditions under which this algorithm performs optimally or sub optimally, it adds to the theoretical framework necessary for developing even more sophisticated algorithms in the future.

6. Methodological Rigor

Using a combination of quantitative analysis tools and robust statistical methods ensures that the study's findings are valid, reliable, and replicable. This rigor in methodology not only strengthens the study's contributions to academic knowledge but also enhances its credibility and utility in practical applications where stakeholders rely on precise and accurate data to make informed decisions.

In summary, the chosen methodology for investigating the Bidirectional Dijkstra algorithm through experimental design and quantitative analysis is justified by its potential to produce significant improvements in algorithmic efficiency, its relevance to practical applications, its foundation in rigorous scientific inquiry, and its capacity to advance both theoretical and applied knowledge in the field of computer science.

2. Results and Analysis

To effectively assess the performance of the Bidirectional Dijkstra algorithm, it is essential to consider a variety of performance metrics. For this study, we will focus on four key metrics: runtime efficiency, path optimality, resource usage, and scalability. These metrics will be compared against those reported in a base paper and another significant work in the area to illustrate the improvements offered by the proposed methodology.

Performance Metrics for Comparative Study:

Runtime Efficiency: Measures the time taken to find the shortest path.

Path Optimality: Evaluates the accuracy and correctness of the path found.

Resource Usage: Assesses the memory and computational resources consumed during the execution of the algorithm.

Scalability: Examines the algorithm's performance across different graph sizes and densities.

Comparative Analysis

The data for each metric will be collected from the proposed implementation of the Bidirectional Dijkstra algorithm, a base paper that uses the traditional Dijkstra algorithm, and another significant work that might utilize an enhanced or alternative shortest path algorithm.

Data Collection

Proposed Work: Implement and run the Bidirectional Dijkstra algorithm on a set of test graphs.

Base Paper: Data extracted from literature where traditional Dijkstra’s algorithm was used.

Other Work: Data from a recent study that has proposed modifications to the Dijkstra algorithm or an alternative approach.

Graphs and Tabulation

Graphs: For each performance metric, a line graph or bar chart will be created. Each graph will compare the performance across the three studies (Proposed Work, Base Paper, and Other Work) to visualize differences in performance clearly.

Table: A comprehensive table will be set up to tabulate the values of each metric from the Proposed Work, the Base Paper, and the Other Work. This will facilitate direct numerical comparisons.

Example Table Structure

Metric	Proposed Work	Base Paper	Other Work
Runtime Efficiency	x ms	y ms	z ms
Path Optimality	100%	95%	98%
Resource Usage	x MB	y MB	z MB
Scalability	x nodes/sec	y nodes/sec	z nodes/sec

Analysis

Runtime Efficiency: Analyse the average, median, and variability of the runtime across different graph types and sizes.

Path Optimality: Measure the percentage of trials where the algorithm found the shortest possible path and compare it across studies.

Resource Usage: Record the peak memory usage and other computational resources during the execution of the algorithm.

Scalability: Assess how well each algorithm performs as the size and complexity of the graph increase.

Interpretation of Results

Discuss any observed trends or significant differences between the proposed work and the other studies.

Identify any conditions under which the Bidirectional Dijkstra algorithm outperforms or underperforms compared to the others.

Evaluate the implications of the findings in terms of practical applications and future research directions.

The resulting analysis will provide a clear and detailed comparison of how the Bidirectional Dijkstra algorithm stacks up against traditional methods and recent innovations, offering valuable insights into its strengths and areas for potential improvement.