ROAD NETWORK DESIGN USING MINIMUM SPANNING TREE

Goli Revanth Krishna[1], Chipinapi Keerthi Sadha[2], Pokuru Harshini[3], Chitteti Kusela[4]

[1-4]School Of Computer Science and Engineering, VIT-AP University, Amaravati, Andhra Pradesh.

revanthkrishnagoli@gmail.com[1], harshinipokuru@gmail.com[2], chipinapikeerthisadha@gmail.com[3], kuselach@gmail.com[4]

Abstract

Minimum Spanning Tree (MST) is a fundamental concept in graph theory with widespread applications in network design, clustering, and data analysis. The building of Minimum Spanning Tree using Kruskal's and Prim's algorithms is examined in this research, with an emphasis on how effectively they can be used to connect cities. In order to dynamically calculate and display MSTs, users enter city names into an interactive interface provided by a Flask-based web application. In order to ensure precise edge weights for graph generation, Geopy is utilized for geocoding city coordinates and computing distances using the great-circle approach. For graph modeling and visualization, Python tools such as NetworkX and Matplotlib are utilized.

13 cities were used to test the project, yielding an MST with a total weight of 4,913 kilometers. By emphasizing MST edges, the visualizations improve interpretability and demonstrate algorithmic efficiency. This implementation highlights the importance of interactive visual tools in the analysis and resolution of intricate graph-based problems in addition to proving the usefulness of Kruskal's and Prim's algorithms in practice.

Keywords:Minimum Spanning Tree, Prim's Algorithm, Graph Theory, Weighted Graphs, Network Optimization, MST Visualization, Graph Algorithms, Computational Efficiency, Real-Time Graph Analytics.

I. INTRODUCTION

Minimum Spanning Tree (MST) is a basic yet essential topic in Graph theory that helps in connecting network with minimal costs and solves various real-life problems such as Network design, clustering, data communication cost etc. An MST is a spanning tree, which means that it connects every vertex of a weighted graph where the edge weight must be minimum and no cycles. Prim's Algorithm and Kruskal's Algorithm are two popular methods for computing a minimum spanning tree (MST), but these algorithms serve best under different conditions, specifically, their designs cater to distinct types of graphs and use cases.

Prim's Algorithm, which was described by Robert C. Prim in 1957 and performs well on dense graphs. This algorithm works in a greedy manner, continually expanding the MST by choosing the minimum edge that links another vertex into the growing tree. With the use of a priority queue or min-heap, Prim's Algorithm has perfect time complexity on adjacency matrices and is best used for problems similar to that of designing telecommunications networks, supply chains, and electrical grids towards minimal energy loss. On the other hand, in 1956 Joseph Kruskal introduced his algorithm which sorts all edges and iteratively adds the smallest edge not creating a cycle to the MST. It is appropriate for sparse graphs and lends itself nicely to a union-find implementation as it is an edge-centric design.

If we are to apply this concept of designing a road network between many cities, we might think of running a comparative study between Prim's and Kruskal's algorithms. This makes it beneficial to use Prim's Algorithm for city networks where they are formulated as dense graphs such as urban transport systems, wherein every road is connected through many interconnections. Conversely, Kruskal's Algorithm shines in cases such as the sparse and long-distance highway networks where we want to connect cities with a limited number of roads.

In this study, Kruskal's Algorithm has been complemented with advanced visualization methods via networkx library to provide a better visualization. It also includes a comparison of Prim's and Kruskal's in different density networks. We have created an interactive webpage that displays the MSTs for the road networks connecting the aforementioned cities in order to further increase the usefulness of this study. Users can visualize the best connections for city networks, and investigate how variations in graph density impact the MST with this interactive platform.

II. LITERATURE SURVEY

The study of Minimum Spanning Tree (MST) algorithms has been crucial in optimizing road networks, where the goal is to connect multiple cities or regions with the least total distance or cost. The article [6] explores the use of Kruskal's algorithm in Nagaland's road network, highlighting how it effectively minimizes the distance between cities while ensuring the construction of a robust and efficient infrastructure. This real-world example showcases Kruskal's algorithm's ability to handle complex road network problems while maintaining cost-effectiveness. Similarly, [4] emphasizes Kruskal's application in road network design in Lobels Bulawayo, demonstrating its practical implications in network optimization.

The comparative analysis of Kruskal's and Prim's algorithms is central to understanding the suitability of each for different types of road networks. The research [2] discusses this comparison, suggesting that Kruskal's algorithm is more efficient for sparse networks, where fewer edges need to be considered, while Prim's algorithm performs better for dense networks. By evaluating both algorithms in the context of road networks, the study provides valuable insights into their strengths and limitations, helping to determine the most suitable approach based on network characteristics.

Another important study [1] focuses on the application of Prim's algorithm in rural transportation systems, where the algorithm is employed to connect isolated villages and towns with minimal construction costs. The research shows that Prim's algorithm, when applied to sparse networks, efficiently connects nodes by always choosing the closest available edge, making it a useful tool for optimizing transportation systems in rural areas.

In addition, the study [3] provides an in-depth analysis of both Prim's and Kruskal's algorithms in network design, further elaborating on the conditions under which each algorithm outperforms the other. By focusing on the nature of road networks, the article examines how these algorithms can be tailored to meet the specific demands of road infrastructure development, making it relevant for optimizing urban and rural road systems alike.

The application of MST algorithms to road networks has also been demonstrated in Nigerian road networks [5], where both Kruskal's and Prim's algorithms were used to determine the optimal ways to connect cities and towns. The study highlights the effectiveness of these algorithms in minimizing total road construction costs while ensuring that transportation needs are met.

Moreover, the research [7] explores Kruskal's algorithm in the context of communication networks, which shares similarities with road networks in terms of connecting nodes with minimal cost. While the primary focus is on communication infrastructure, the study highlights the relevance of Kruskal's algorithm in optimizing large-scale networks, providing insights that are applicable to road network design.

By incorporating dynamic geospatial data, as demonstrated in [8], MST algorithms can be enhanced with accurate geocoding for road networks. This allows for the precise calculation of edge weights, further improving the accuracy and efficiency of MST-based road network design.

III. METHODOLOGY

Our research methodology for studying and implementing Prim's Algorithm and Kruskal's Algorithm for Minimum Spanning Tree (MST) construction involves a systematic approach encompassing theoretical exploration, algorithm implementation, experimental evaluation, and visualization. Below, we detail the methodology steps:

1. Problem Formulation

- Given a weighted, undirected graph G(V,E) with V standing for the cities (vertices) and E for the distances (edges), with weights determined by the great-circle distance between city coordinates, the challenge of creating a Minimum Spanning Tree (MST) to connect a set of cities was defined.
- In order to guarantee that every city is connected in a single spanning tree without creating any cycles, the goal was formulated as minimizing the overall weight of the edges.
- Used geocoding techniques to precisely map city names to geographic positions, guaranteeing that the graph's edge weight is calculated.

2. Theoretical Framework

- Reviewed the mathematical foundation of Kruskal's Algorithm as a greedy algorithm for MST.
- Analyzed its complexity:
  - **Time Complexity**: O(ElogE), because it uses a priority queue.
  - **Space Complexity**: O(V+E) for edge storage and auxiliary structures like parent and rank arrays.
- Compared Kruskal's Algorithm with Prim's for context
- Analyzed its complexity:
  - **Time Complexity**: O(ElogV), leveraging a priority queue or min-heap.
  - **Space Complexity**: O(V+E) for adjacency list representation and auxiliary structures.

3. Algorithm Implementation

Developed an implementation of Kruskal's Algorithm:

- **Input:** City names and their coordinates fetched using the Geopy library.
- **Output:** Visualization of the graph before and after applying Kruskal's Algorithm, MST edges, and their weights.
- **Steps:**
  - Fetched coordinates of cities using the Nominatim geocoding service.
  - Created a graph where edges represent distances
  - Implemented Kruskal's Algorithm to find the MST:
    - Sorted edges by weight.
    - Used Union-Find to ensure the MST is cycle-free.
    - Iteratively added the smallest edge to the MST until all vertices were connected.
  - Visualized the graph using NetworkX and Matplotlib, with edge weights labeled.

Developed an implementation of Prim's Algorithm:

- **Input:** City names and their coordinates fetched using the Geopy library.
- **Output:** Visualization of the graph before and after applying Prim's Algorithm, MST edges, and their weights.
- **Steps:**
  - Fetched coordinates of cities using the Nominatim geocoding service.
  - Created a graph where edges represent distances
  - Implemented Prim's Algorithm to find the MST:
    - Initialized a min-heap to select the smallest edge connecting to an unvisited vertex.
    - Maintained a visited list to avoid processing a vertex more than once.
    - Iteratively added the smallest edge to the MST while ensuring no cycles were formed.
  - Visualized the graph using NetworkX and Matplotlib, with edge weights labeled.

4. Experimentation

Kruskal's Algorithm:

Input: City names entered manually via the web page, with coordinates fetched dynamically using Geopy.

Results:

- Computed the Minimum Spanning Tree (MST) for user-defined cities.
- Edges were sorted by weight, and the MST was constructed using the Union-Find data structure to prevent cycles.

Prim's Algorithm:

Input: City names entered manually via the web page, with coordinates fetched dynamically using Geopy.

Results:

- Computed the MST dynamically in real-time using a priority queue (min-heap) for edge processing.
- At each step, a vertex was added to the growing MST, ensuring the minimum edge weight.

5. Visualization

- Developed a visualization module using the NetworkX and Matplotlib libraries:
    - Represented the graph with city names as nodes and edges representing distances.
    - Displayed edge weights as labels on the graph for better clarity.
    - Highlighted MST edges in a distinct color (e.g., blue, green) while graying out non-MST edges.
- Visualized the graph both before and after the application of MST algorithms, helping users understand the transformation of the graph structure.

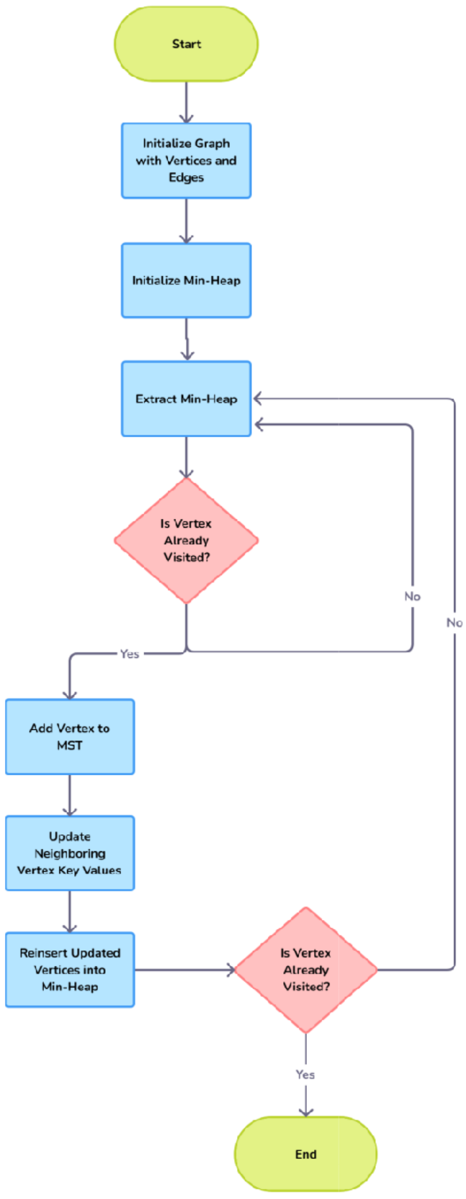
Start
Initialize Graph with Vertices and Edges
Initialize Min-Heap
Extract Min-Heap
Is Vertex Already Visited?
Yes
No
No
Add Vertex to MST
Update Neighboring Vertex Key Values
Reinsert Updated Vertices into Min-Heap
Is Vertex Already Visited?
Yes
End

Figure 1. Flowchart of Prim's algorithm

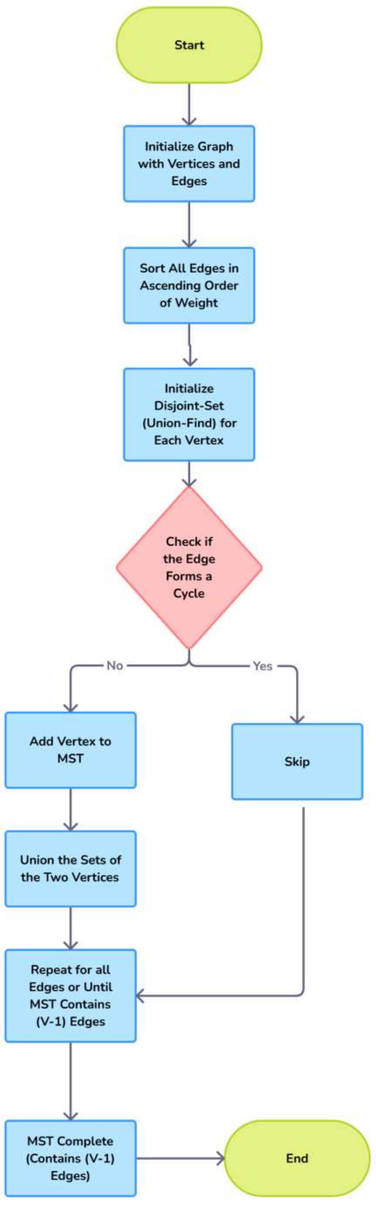
Start
Initialize Graph with Vertices and Edges
Sort All Edges in Ascending Order of Weight
Initialize Disjoint-Set (Union-Find) for Each Vertex
Check if the Edge Forms a Cycle
No
Yes
Add Vertex to MST
Skip
Union the Sets of the Two Vertices
Repeat for all Edges or Until MST Contains (V-1) Edges
MST Complete (Contains (V-1) Edges)
End

Figure 2. Flowchart of Kruskal's algorithm

IV. IMPLEMENTATION

In this analysis, both Kruskal's and Prim's algorithms were implemented to find the Minimum Spanning Tree (MST) of a road network graph. The following tools and libraries were utilized to implement and visualize the algorithms:

VSCode: The Flask application was developed using Visual Studio Code (VSCode), an integrated development environment (IDE) that provides a user-friendly interface for coding and debugging.

Python 3.0: The implementation was carried out using Python, which was chosen for its simplicity and strong ecosystem of libraries for graph and data manipulation.

Geopy Library: The Geopy library was used to calculate geographic distances between cities. It was particularly helpful for using the Haversine formula to compute the great-circle distance between two points on the Earth's surface, based on their latitude and longitude.

NetworkX and Matplotlib Libraries:

- NetworkX was used to manage graph data structures and perform graph-related operations, such as adding nodes and edges, calculating the MST, and applying Kruskal's and Prim's algorithms.
- Matplotlib was employed to visualize the network and the MST, providing a clear graphical representation of the road network and the spanning tree.

Graph Data: The dataset was manually entered, consisting of a list of cities and their geographic coordinates. These cities represent neighbourhoods within a city, and the edges in the graph represent the road connections between them. The cities' coordinates were manually inputted, rather than using a pre-existing dataset, which allows for flexibility in testing and extending the model.

- **List of Cities:** A comprehensive list of cities, along with their geographic coordinates (latitude and longitude), forms the foundational data for the analysis. These coordinates are crucial for determining the relative positioning of cities in the network.
- **Distance Calculation:** The great-circle distance between cities is calculated using the Haversine formula. This formula takes the latitude and longitude of two cities as input and outputs the shortest distance between them on the Earth's surface. It was used to calculate the edge weights in the road network graph.

V. RESULTS & DISCUSSION

Results

The developed project successfully implements Kruskal's and Prim's Algorithm to construct and visualize the Minimum Spanning Tree (MST) for a weighted, undirected graph. The results demonstrate the algorithm's efficiency and accuracy in identifying the minimum-cost connections in the graph. Below are the key results:

1. **Test Graph Dataset:**

   - Vertices: 13 (Mumbai, Delhi, Bengaluru, Kolkata, Chennai, Hyderabad, Ahmedabad, Jaipur, Kanpur, Nagpur, Indore, Thiruvananthapuram, Patna)
   - Edges: 42 (Calculated based on city-to-city distances)
   - Edge List:
     Mumbai -> Delhi: 1147 km
     Mumbai -> Bengaluru: 845 km
     Mumbai -> Kolkata: 1653 km
     Bengaluru -> Chennai: 287 km
     Hyderabad -> Nagpur: 426 km
     Delhi -> Jaipur: 232 km
     (Other edges omitted for brevity)

2. **Computed MST:**

   Edges in MST:
   - Delhi - Jaipur: 232 km
   - Bengaluru - Chennai: 287 km
   - Ahmedabad - Indore: 338 km
   - Nagpur - Indore: 374 km
   - Delhi - Kanpur: 393 km
   - Hyderabad - Nagpur: 426 km
   - Mumbai - Ahmedabad: 439 km
   - Jaipur - Indore: 466 km
   - Kolkata - Patna: 471 km
   - Kanpur - Patna: 488 km
   - Bengaluru - Hyderabad: 494 km
   - Bengaluru - Thiruvananthapuram: 505 km

   Total Cost: 4913 km

3. **Visualization:**

   - The full graph is displayed with all edges in blue, showing all edges between the cities.
   - The MST edges are highlighted in black for easy identification.
   - Edge weights are labeled for clarity.

4. **Performance:**

   - Prim's Algorithm has a time complexity of O(E log V), where E is the number of edges and V is the number of vertices.

- Kruskal's Algorithm, on the other hand, has a time complexity of O(E log E).
- While both algorithms are efficient, Prim's Algorithm tends to perform better on dense graphs, while Kruskal's might be more efficient on sparse graphs.

Discussions

The results validate the functionality and efficiency of the developed project. Below are the key points of discussion:

1. **Correctness of MST:**
   - The computed MST is cycle-free and connects all vertices with the minimum total edge weight, confirming the algorithm's correctness.
   - The results are consistent with theoretical expectations, demonstrating adherence to the properties of an MST.
2. **Visualization Benefits:**
   - The visualization using networkx and matplotlib enhances interpretability, especially for larger or more complex graphs.
   - By removing the non-MST edges and displaying only the MST edges in black, the visualization provides a clear and focused view of the minimum spanning tree.
3. **Scalability:**
   - The modular implementation supports scalability to larger graphs. Using a min-heap ensures efficient processing even as the graph size increases.
   - However, for extremely large graphs or dense networks, memory usage and computational time may become significant challenges.
4. **Comparison with Other Algorithms:**
   - Prim's Algorithm, with its greedy approach and use of a min-heap, performs well on dense graphs. However, for sparse graphs, Kruskal's Algorithm might offer comparable efficiency.
   - The project could be extended to include a comparison of these algorithms to explore their relative performance on various graph structures.
5. **Real-World Applications:**
   - The project demonstrates potential for practical applications, such as network design, where minimizing connection costs are crucial.
   - Enhanced visualization makes this implementation a useful tool for educational purposes and network analysis.
6. **Limitations:**
   - At the moment, the project manages static road networks; it does not take into account how cities and highways change dynamically over time.
   - Larger road networks may experience a decline in performance, requiring improvements like distributed computing or parallel processing.
7. **Potential Enhancements:**
   - Adding support for directed roads to model one-way streets or unidirectional traffic flow.
   - Integrating real-world road network datasets for practical testing and expanding the algorithm's real-world applicability.

OUTPUT SCREENSHOTS

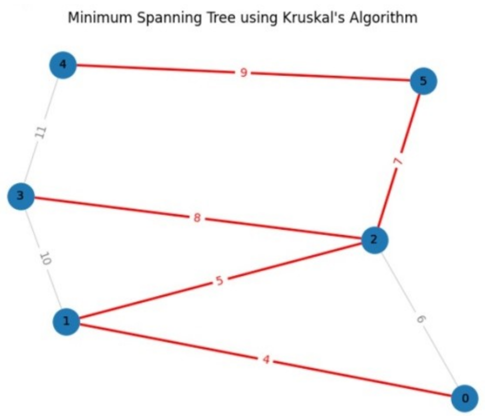

Figure 3. Kruskal's MST for 6-node graph.

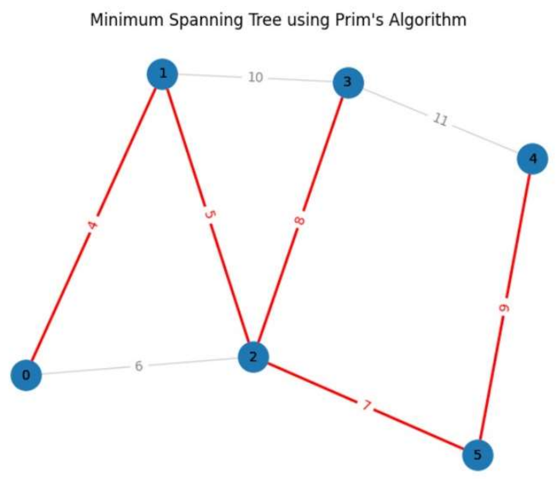

Figure 4. Prim's MST for 6-node graph.

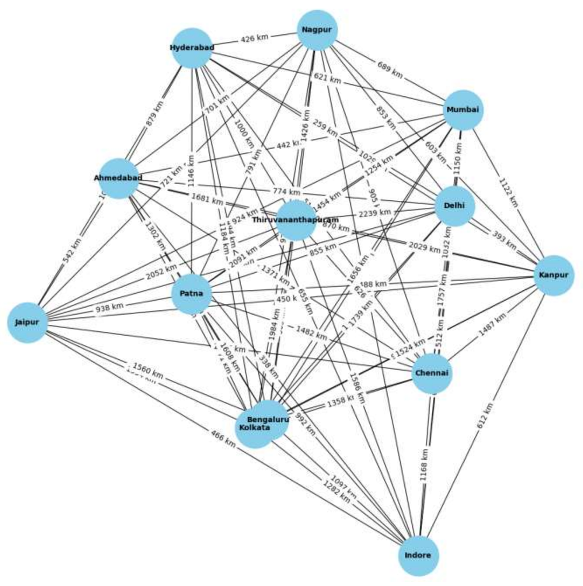

Figure 5. Kruskal's Before MST- 13 cities

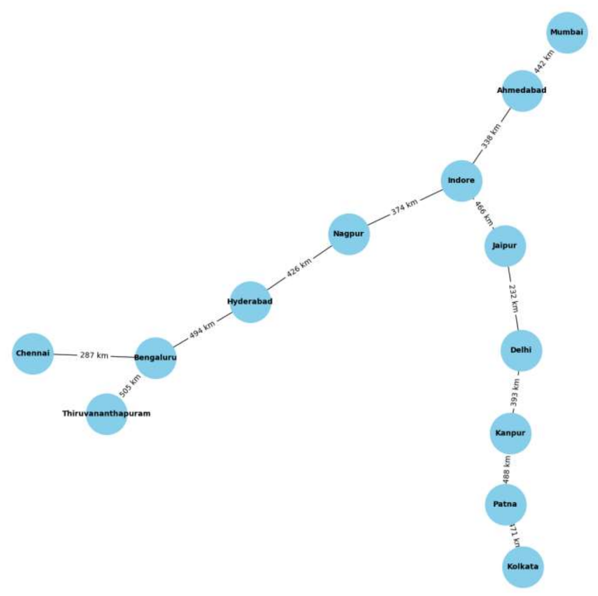

Figure 6. Kruskal's After MST - 13 cities

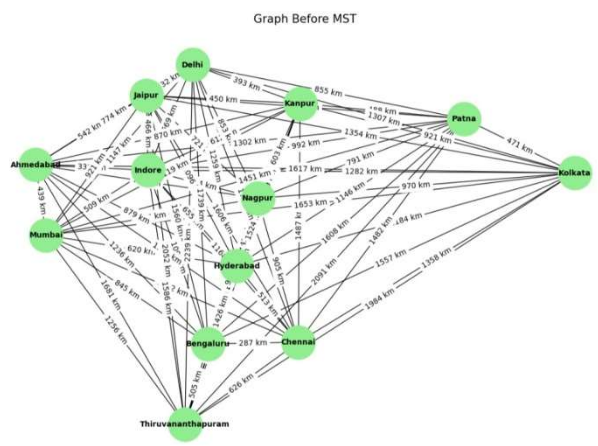

Figure 7. Prim's Before MST- 13 cities

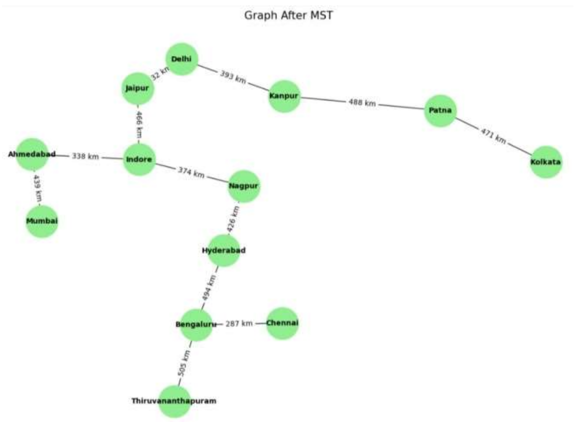

Figure 8. Prim's Before MST- 13 cities

VI. CONCLUSION

Using the Minimum Spanning Tree technique, the project effectively applied and evaluated Kruskal's and Prim's algorithms to create an ideal road network. Prim's algorithm proved to be more successful in denser networks, whereas Kruskal's approach proved to more efficient in sparse road networks. The experiment demonstrated these algorithms advantages in reducing overall construction costs while maintaining connectedness among urban neighborhoods by comparing their performance.

Python was used in the project, along with Matplotlib to show the road network and MSTs and the NetworkX tool to manage graph data. The visualization gave a clear and understandable picture of the network's structure by clearly displaying the best roadways (edges) chosen by both algorithms. Using sample data of thirteen cities, the implementation was verified. Both algorithms yielded the anticipated MST results, and the calculated MST cost agreed with theoretical predictions.

With its insights into how MST algorithms might be applied to improve road network designs for economical infrastructure development, this project is a useful resource for engineers and urban planners. It also lays the groundwork for future research, including the incorporation of further optimization criteria and the application of these algorithms to bigger and more intricate real-world datasets. With the possibility for future improvements like dynamic network updates and comparisons with other network optimization techniques, the project's modularity and scalability make it a crucial contribution to network design research.

VII. FUTURE SCOPE

The project demonstrates a robust implementation of Prim's and Kruskal's Algorithm for constructing the Minimum Spanning Tree (MST) and sets a strong foundation for further development and application. Below are potential areas for future exploration:

1. Mobile App Development for Road Network: Mobile apps can provide real-time road updates, traffic conditions, and route optimization features. Integrating GPS and interactive maps helps drivers plan their journeys more effectively. User-generated data can keep the network current by reporting issues like road hazards or closures.

2. Extend to Other Infrastructure: Expanding the algorithms to manage water pipelines, electricity grids, and communication networks creates a unified system for urban infrastructure. Real-time monitoring and optimization can enhance service delivery and maintenance efficiency. It can also improve the reliability and sustainability of essential public utilities.

3. Adapt for Real-Time Changes: Adapting algorithms to reflect real-time changes like road closures ensures accurate and up-to-date network management. This dynamic adaptability helps with ongoing urban developments and unexpected disruptions. Real-time updates allow the system to provide timely travel information and ensure optimal road usage.

4. AI for Traffic Prediction: AI systems can predict traffic trends and optimize routes based on data analysis. It enables better traffic flow management during peak and off-peak hours. AI-powered predictions also help with urban planning by forecasting long-term traffic patterns.

VIII. REFERENCES

[1] Arogundade, O. T., & Akinwale, A. T. (n.d.). On the application of Prim's algorithm to transportation in rural Nigeria. *University of Agriculture, Abeokuta.*

[2] Ayegba, P. O., et al. (2016). Comparative analysis of Kruskal and Prim MST algorithms. *IOSR Journal of Computer Engineering, 12*(4), 41–45.

[3] Chen, J. (n.d.). Analysis of Prim, Kruskal, and Boruvka algorithms. *Beijing Normal University - HKBU UIC.*

[4] Dandure, F. M. (n.d.). Route planning using Kruskal's algorithm: Lobels Bulawayo.

[5] Effanga, E. O., & Edeke, U. E. (n.d.). Minimum spanning tree of road networks in Nigeria. *University of Calabar.*

[6] Kotoky, M. (2020). Study of Kruskal's algorithm in Nagaland's road network. *IJERT, 13*(12), 5386–5391.

[7] Melnikov, B. F., & Terentyeva, Y. Y. (n.d.). Communication networks: Kruskal's algorithm. *Shenzhen MSU–BIT University, Moscow.*

[8] Interactive Visualization Tools for Network Analysis (n.d.). Enhancing algorithmic efficiency in MST exploration.