Comparative Analysis of Prim's and Kruskal's Algorithms for Minimum Spanning Tree Construction

Table of Contents

1. Introduction

- **Objective**: This report aims to compare Prim's and Kruskal's algorithms in terms of their time complexity, implementation, and efficiency in different types of graphs.
- **Background**: Minimum Spanning Tree (MST) is a fundamental problem in graph theory, used in various applications such as network design, clustering, and image processing. Prim's and Kruskal's algorithms are two classic approaches to solve this problem.

2. Theoretical Analysis

2.1. Prim’s Algorithm

- **Description:** Prim's algorithm builds the MST by starting from an arbitrary vertex and growing the MST one edge at a time by adding the smallest edge that connects a vertex inside the MST to a vertex outside.
- **Pseudocode:**
  Initialize MST as an empty set.
  Select an arbitrary vertex to start.
  While there are edges that can be added to the MST:
  Find the minimum weight edge that connects a vertex in the MST to a vertex outside.
  Add the edge to the MST.
- **Time complexity:**
  - **Adjacency Matrix:** $O(V^2)$
  - **Adjacency List with Binary Heap:** $O((V+E) \log V)$

2.2. Kruskal’s Algorithm

- **Description:** Kruskal's algorithm builds the MST by sorting all edges in ascending order of their weight and adding them to the MST if they don't form a cycle.
- **Pseudocode:**
  Sort all edges in non-decreasing order of their weight.

Initialize MST as an empty set.

For each edge in the sorted list:

If adding the edge to the MST does not form a cycle:

Add the edge to the MST.

- **Time complexity:**
  - **Sorting edge:** $O(E \log E)$
  - **Union-Find operation:** $O((V+E) \log V)$ per operation

3. Implementation

3.1. Environment and tools

- Programming language: Python
- Tools and Library: NetworkX for graph generation, Matplotlib for visualization
- Hardware: CPU AMD 5700X3D, 32GB RAM

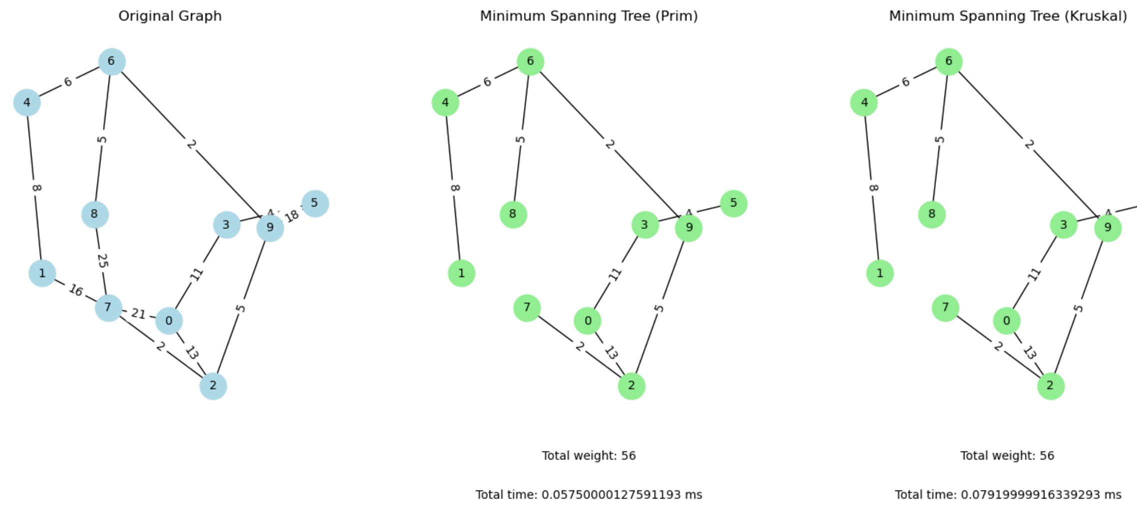
3.2. Code implementation

- Prim’s algorithm
- Kruskal’s algorithm

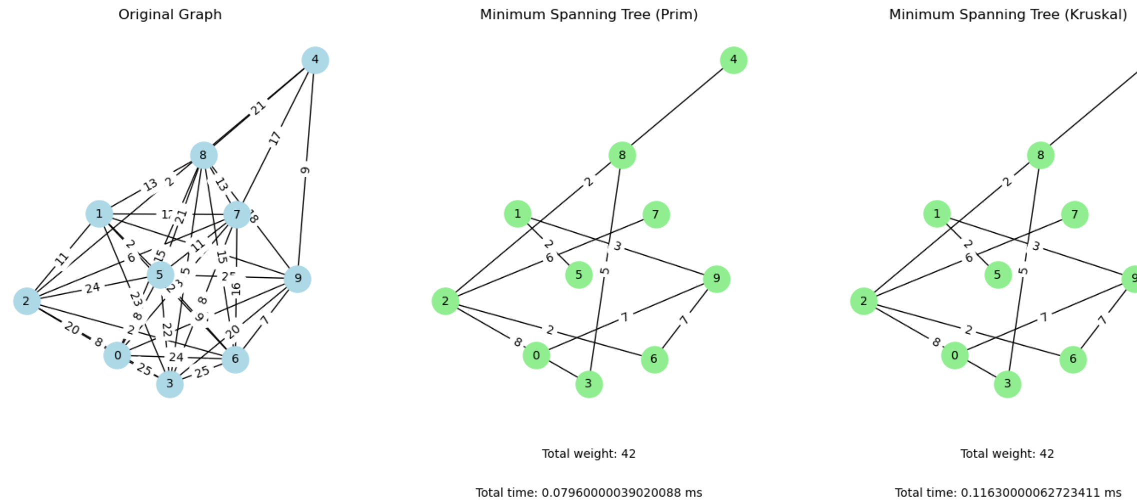
4. Experimental Analysis

4.1. Test case

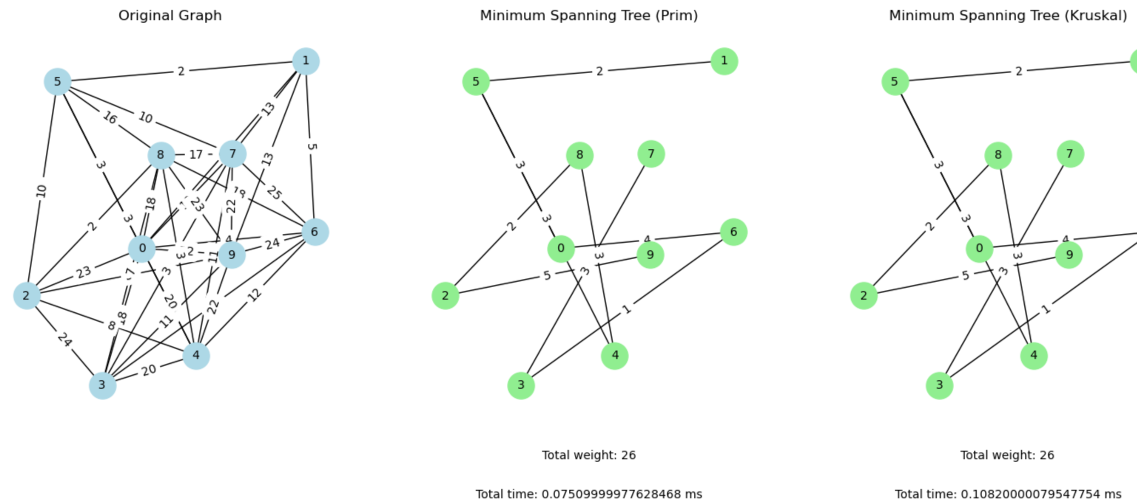
- Sparse graph: 10 vertices, 0.3 density



- Dense graph: 10 vertices, 0.8 density



- Random graph with negative weight: 10 vertices, 0.5 density

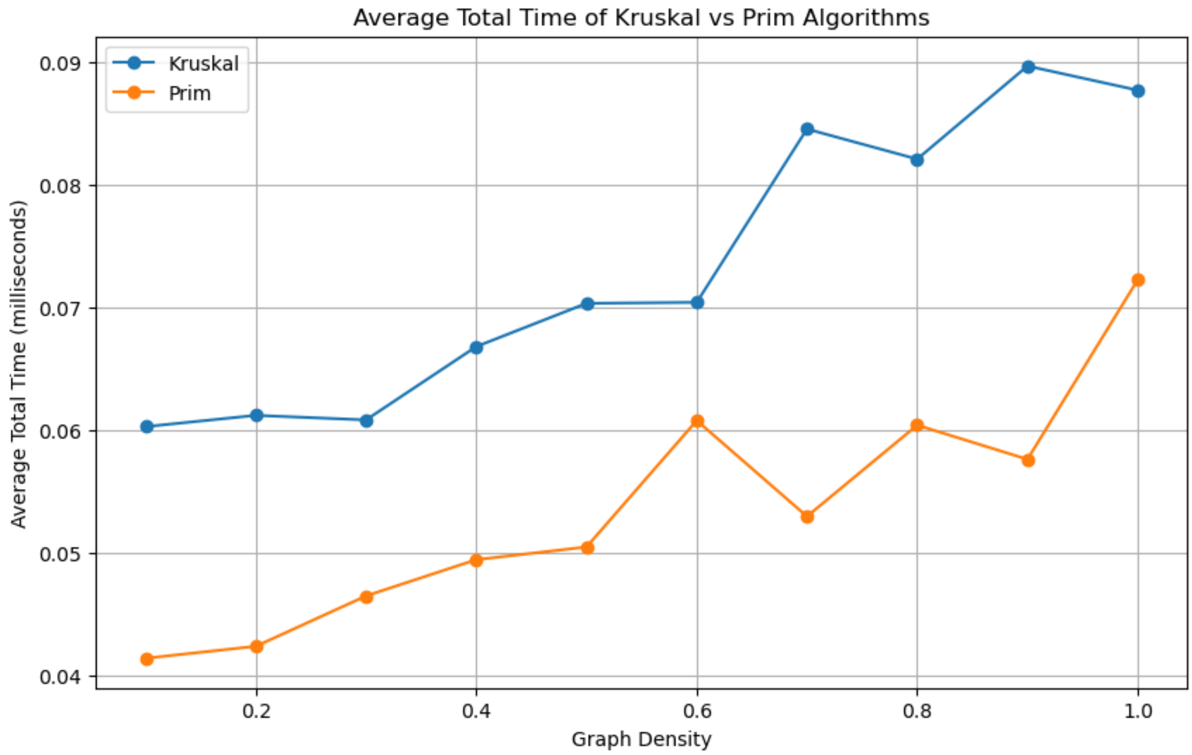


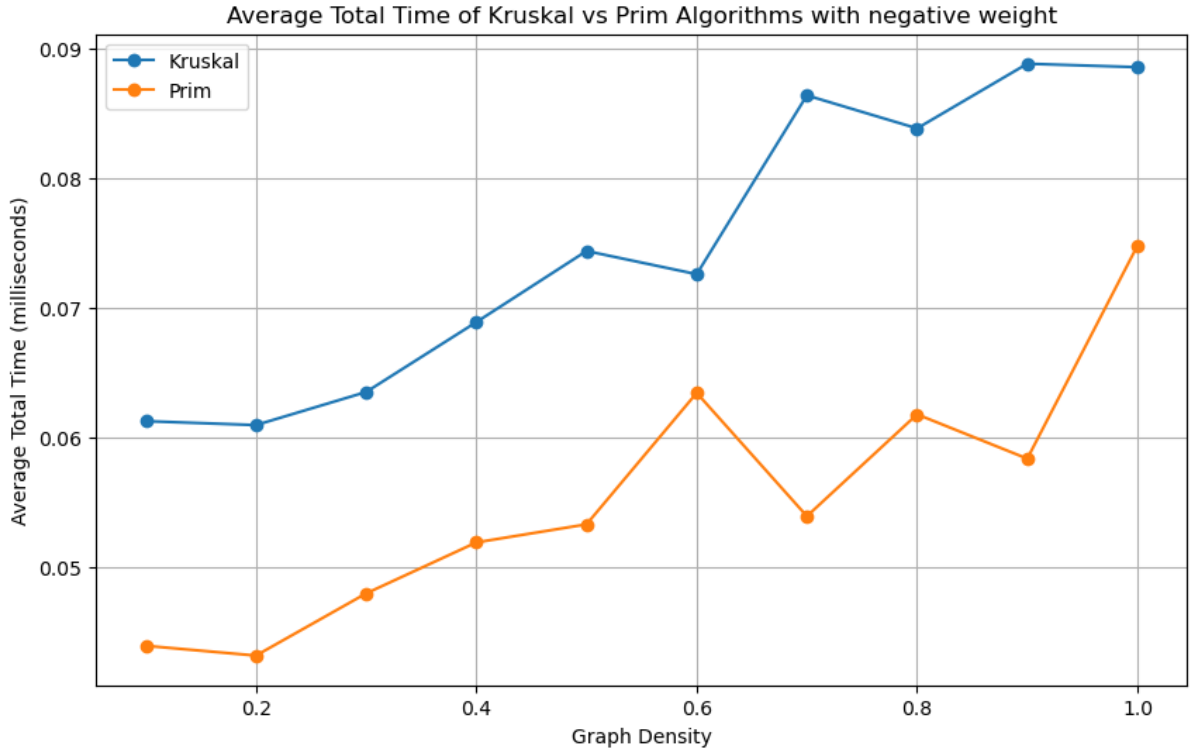
4.2. Performance metrics

- Execution time

- For a sparse graph, Prim's: 0.0575 ms; Kruskal's: 0.0791 ms.
- For a dense graph, Prim's: 0.0796 ms; Kruskal's: 0.1163 ms.
- For a random graph with negative weights, Prim's: 0.075 ms; Kruskal's: 0.1082 ms.

- Scalability





5. Result

5.1 Time complexity analysis

- In a sparse graph, Prim's algorithm has a faster execution time (0.0575 ms) compared to Kruskal's algorithm (0.0791 ms).
- In a dense graph, Prim's algorithm has a faster execution time of 0.0796 ms compared to Kruskal's algorithm, which takes 0.1163 ms.
- In a random graph with negative weights, Prim's algorithm, with an execution time of 0.075 ms, outperforms Kruskal's algorithm, which takes 0.1082 ms.

5.2 Scalability

- As the density of the graph increases, the execution time of both Prim's and Kruskal's algorithms also rises.

6. Discussion

6.1. Comparative summary

Kruskal's algorithm generally takes more time to process than Prim's algorithm due to its sorting step. In Kruskal's algorithm, all edges must first be sorted, which has a time complexity of $O(E \log E)$, where E is the number of edges. After sorting, the algorithm processes each edge with union-find operations, which also contribute to the overall time complexity of $O(E \log E)$. In contrast, Prim's algorithm processes edges as they are added to a priority queue, resulting in a time complexity of $O(E \log V)$, where V is the number of vertices. Since $E \log E$ typically grows faster than $E \log V$, Kruskal's algorithm can be more time-consuming, especially for graphs with a large number of edges.

6.2. Real-world application

6.2.1. Network Design:

- Application: Designing efficient network infrastructure, such as telecommunications networks, computer networks, or electrical grids.
- Prim's Algorithm: Particularly useful when expanding a network gradually, starting from a central node and adding the nearest, least expensive connection each time.
- Kruskal's Algorithm: Useful when you have a list of all potential connections (e.g., cables or links) and want to select the subset that connects all points with the least total cost.

6.2.2. Road and Railway Systems:

- Application: Building or optimizing road, railway, or pipeline networks that connect various cities or locations.
- Prim's Algorithm: Effective when starting from a particular city or hub and gradually expanding the network.
- Kruskal's Algorithm: Useful when planning the network without a starting point, focusing on minimizing the total construction cost by considering all possible routes.

6.2.3. Cluster Analysis in Data Mining:

- Application: Grouping similar data points into clusters based on distance or similarity measures.
- Prim's Algorithm: Can be used to find the nearest neighbors and create clusters by connecting similar data points.
- Kruskal's Algorithm: Useful in hierarchical clustering, where the goal is to connect data points in a way that minimizes the overall distance or dissimilarity.

6.3. Insights

- Kruskal’s algorithm is more efficient when dealing with graphs where the edges are already sorted by weight or when the graph is relatively sparse.
- Prim’s algorithm is more effective for dense graphs, especially when a starting node is given or when the graph is represented as an adjacency matrix. It is also preferable for connected graphs and real-time dynamic applications, where its implementation is often simpler and more efficient.

7. References

- Prim’s algorithm implementation: Geeksforgeeks Prim
- Kruskal’s algorithm implementation: Github Kruskal
- Real-world application: Chat GPT