Graph Algorithm Project

Spanning Tree for Utility Grid Design

2025-08-22

Roll numbers 03, 09, 22UCAM007, 20

by

Amithi Shangari [SE23UCAM003]

Madhav Basur [SE23UCAM009]

Sanket Motagi [SE23UCAM020]

Netan Reddy [SE22UCAM007]

Contents

1. Introduction 4
1.1. Objective 4
1.2. Relevance 4
1.3. Scope 4
2. Algorithms Used for MST 4
2.1. Prim's Algorithm 5
2.2. Kruskal's Algorithm 5
3. Methodology 5
3.1. Graph Representation 5
3.2. Algorithm Implementation 5
3.2.1. Prim's Algorithm 5
3.2.2. Kruskal's Algorithm 6
3.2.3. Performance Evaluation 7
3.3. Example and Results 7
4. Challenges Faced 8
4.1. Algorithmic Challenges 8
4.2. Practical Challenges 8
4.3. Debugging Challenges 9
5. Key Trade-off's Between Prim's and Kruskal's Algorithms 9
5.1. Time Complexity 9
5.2. Scalability 9
5.3. Edge Selection Process 9
5.4. Adaptability 10
5.5. Use cases 10
6. References 10

§1. Introduction

The design and optimization of utility grids, such as power, water, or communication networks, is a fundamental problem in network science and graph theory. One common approach to optimizing such networks is to model them as graphs, where nodes represent individual entities (e.g., houses or substations) and edges represent the potential connections with associated costs. The task is often to identify a Minimum Spanning Tree (MST), which ensures that all entities are connected with the least cost possible. The MST problem has been extensively studied and has applications in fields such as telecommunications, transportation, and urban planning.

§1.1. Objective

The objective of this project is to design an efficient utility grid using Minimum Spanning Tree (MST) algorithms, ensuring to connect all houses of a small town.

§1.2. Relevance

MST algorithms play a crucial role in real-world network design, particularly in optimizing utility grids, telecommunications, and transportation networks. They ensure minimal path costs while maintaining connectivity..

§1.3. Scope

This project focuses on applying MST algorithms, specifically Prim's and Kruskal's, to optimize a power distribution network by identifying the minimum-cost configurationto connect all houses of a small town.

§2. Algorithms Used for MST

This section presents the proposed MST algorithm. Each algorithm is explained with a common example. A single graph can have many different spanning trees.

The MST algorithms are classified as line based MST algorithms and node based MST algorithms. They are

1. Line based MST Algorithms
    - Kruskal's algorithm
    - Reverse Delete algorithm
2. Node based MST Algorithms
    - Prim's algorithm
    - Dijkstra's algorithm

§2.1. Prim's Algorithm

The algorithm was developed in 1930 by the Czech mathematician, Vojtech Jarnik, and later independently, by the computer scientist, Robert C. Prim, in 1957. This algorithm finds a subset of the nodes that form a tree that includes every node, where the total weight of all the lines in the tree is minimized.

The nature of the Prim's algorithm makes it necessary to provide two data values for every unselected node. The data values are provided through two arguments:

- First will be the unselected node's(V-V_T) connectivity to the currently selected node(V_T). It will be "nil" if no connectivity exists.
- The second entry (distance label) will be the respective weight. If there is no connection, then the value will be ∞.

§2.2. Kruskal's Algorithm

Kruskal's algorithm is an algorithm in graph theory that finds a minimum spanning tree for a connected weighted graph. This means that it finds a subgraph of the lines that form a tree. It includes every node of the network and the total weight of all the lines in the tree is minimized.

To implement the Kruskal's algorithm, two conditions have to be satisfied:

- The weight of the lines of the graph must be arranged in the increasing order.
- An empty subgraph T (called the Traversal matrix) is created.

§3. Methodology

The nodes of the graph represent houses, while edges represent connections with their weights representing the cost.

§3.1. Graph Representation

The graph is modeled as an undirected, weighted graph G=(V,E), where V is the set of houses (nodes) and E is the set of potential connections (edges). Edge weights were assigned randomly.

§3.2. Algorithm Implementation

§3.2.1. Prim's Algorithm

This algorithm grows the MST incrementally by starting from an arbitrary vertex, repeatedly selecting the smallest edge that connects a visited vertex to an unvisited vertex, and updating the visited set. The algorithm ensures the inclusion of the minimum-cost edges at every step.

1. Initialize an empty list “MST” to store the edges of the Minimum Spanning Tree.
2. Initialize a variable “total_cost” to 0 to store the total weight of the MST.
3. Create an empty set “visited” to keep track of nodes that are part of the MST.
4. Select an arbitrary starting node and add it to visited.
5. While MST contains fewer edges than V - 1 (where V is the number of nodes):
   a. Initialize min_edge as (∞, None, None) to represent the smallest edge.
   b. For each edge (weight, u, v) in the graph’s edges: i. If u is in visited and v is not in visited:
   - If weight $<$ the weight of min_edge (initially ∞), update min_edge to (weight, u, v), i.e. we update the weight of the min_edge to the weight of the visited node, and update it’s starting and end points to the vertices of said node.

   ii. If v is in visited and u is not in visited, we repeat above process, but min_edge becomes (weight, v, u). c. If the graph is disconnected, we break. d. Add the selected edge to MST. e. Add the weight of min_edge to total_cost. f. Add the new node to visited.
6. Finally we return MST and total_cost.

§3.2.2. Kruskal’s Algorithm

The objective is to pick an edge one by one according to the ascending order of their weight, while making sure no cycles are formed. This is done using the Union-Find dataset.

1. For each vertex v
   a. Assign itself as it’s own parent.
   b. Initialize rank[v]=0.
2. Prepare a list of all edges E and sort them by weight in ascending order.
3. Find function:
   a. If parent[node] is not equal to node, i.e. the node is not the root, we recursively call the function and update parent to it’s root.
   b. Then we return the root of the set containing the node.
4. Union Function:
   a. Find the roots of node1 and node2 using find.
   b. If the roots are different, we compare their ranks. The tree with the lower rank is attached under the tree with bigger rank./ c. If the ranks are equal, we arbitrarily attach the second tree under the first./
5. Initialize an empty list “MST” to store the edges of the Minimum Spanning Tree.
6. Initialize a variable “total_cost” to 0 to store the total weight of the MST.
7. For each edge (weight, u, v) in the sorted edges/ a. If the roots of u, v are different (found using find function), we join them using the union function. b. We then add the resultant edge to the MST list, and add the respective weight to total_cost.

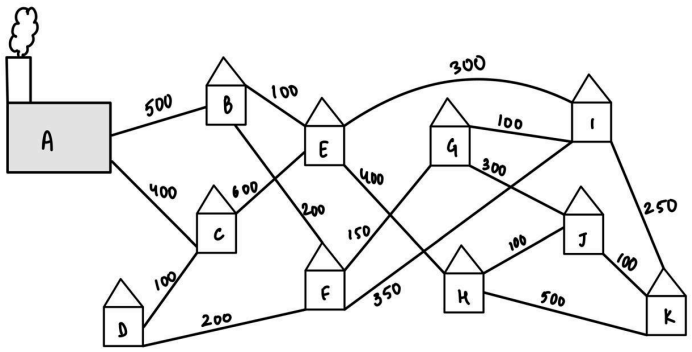
8. Finally we return MST and total_cost.

§3.2.3. Performance Evaluation

Execution times and total costs were measured for both algorithms across various graph sizes. A Python implementation was used, and random graphs were generated to assess scalability.

§3.3. Example and Results

We implemented Prim's and Kruskal's Algorithms on graph given below.

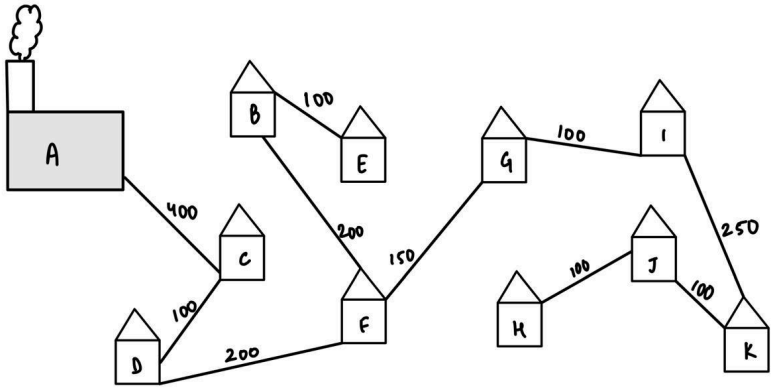


- **Output:**

– **Edges in the MST (Prim's):**
A – C = 400,
C – D = 100,
D – F = 200,
F – G = 150,
G – I = 100,
F – B = 200,
B – E = 100,
I – K = 250,
K – J = 100,
J – H = 100
Total cost (Prim's): 1700

– **Edges in the MST (Kruskal's):**
B – E = 100,
C – D = 100,
G – I = 100,
H – J = 100,
J – K = 100,
F – G = 150,
B – F = 200,

D – F = 200,
K – I = 250,
A – C = 400
Total cost (Kruskal's): 1700



We performed a time complexity analysis, and saw that Prim's Algorithm worked better for dense graphs while Kruskal's Algorithm worked better for sparse graphs.

Prim's Algorithm

- The time complexity is O(E.V), due to the usage of while and for/if loops.
- This can be optimized by using min_heap or adjacency matrices.

Kruskal's Algorithm

- Consistently operates at $O(E \log V)$ due to the need for sorting edges and managing disjoint sets for cycle detection.

§4. Challenges Faced

§4.1. Algorithmic Challenges

- Identifying all potential paths and connections in complex networks proved to be challenging, especially in large, interconnected graphs. Predefined rules helped in simpler cases but were insufficient for dynamic, variable configurations.
- Solving the minimum spanning tree (MST) problem under practical constraints, such as ensuring connectivity and avoiding redundant paths, required significant computational effort, particularly in scenarios involving large datasets.

§4.2. Practical Challenges

- Handling dynamic changes in the network, such as the addition or removal of nodes and edges, required a robust algorithmic framework to maintain efficiency.
- Balancing performance and reliability while ensuring the optimal selection of edges was complex due to variations in edge weights and structural changes across different scenarios.

§4.3. Debugging Challenges

- Ensuring the algorithm adhered to constraints like connectivity, cycle prevention, and minimizing total cost required extensive testing and debugging to avoid issues like incorrect edge selection or failure to build a spanning tree.
- Identifying edge cases where the algorithm might fail, such as disconnected components or unusual edge weight configurations, involved iterative refinement and testing.

§5. Key Trade-off's Between Prim's and Kruskal's Algorithms

§5.1. Time Complexity

- As discussed, Kruskal's Algorithm has a time complexity of O(ElogV), and is harder to implement on graphs with a large number of vertices.
- On the other hand, Prim's Algorithm has a time complexity of O(E.V) which can be optimized using a min heap.

§5.2. Scalability

Kruskal's Algorithm

- Scales well with larger datasets because its performance primarily depends on the number of edges rather than the number of vertices.
- The flip side is this algorithm is not as efficient against graphs with a high number of vertices, as the sorting algorithm will slow down, in turn slowing down the find function.

Prim's Algorithm

- While effective in dense networks, may struggle with scalability when implemented with less efficient data structures.
- The issue of too many edges does not arise in this algorithm as it is a node based algorithm, which finds a singular path without traversing through all edges first.

§5.3. Edge Selection Process

Prim's Algorithm

- Grows the MST by selecting edges that connect a vertex already in the tree to one outside of it.
- This means it may traverse nodes multiple times, leading to potential inefficiencies in certain scenarios.

Kruskal's Algorithm

- Focuses on adding the smallest available edge regardless of its connection to existing nodes in the MST.
- Each edge is considered only once, which can lead to a more straightforward implementation in some contexts.

§5.4. Adaptability

Both algorithms can adapt to changes in network structure (e.g., adding or removing nodes/edges). However, Kruskal's flexibility allows it to handle disconnected components more gracefully than Prim's, which requires a connected starting point.

§5.5. Use cases

Prim's Algorithm

- Ideal for applications where connectivity is crucial and where the graph is dense, such as telecommunications networks or road networks where many connections exist.

Kruskal's Algorithm

- Better suited for applications involving sparse networks or when working with data that can be represented as a series of edges, such as clustering algorithms or network design tasks where minimizing edge weight is critical.

For large-scale networks (e.g., urban utility grids), choosing an appropriate implementation strategy is crucial. Using efficient data structures for Prim's can improve its performance significantly but may still not match Kruskal's efficiency in sparse scenarios.

§6. References

1. Mainly referred to the following paper:
   - https://www.researchgate.net/publication/221926621_Power_Restoration_in_Distribution_Network_Using_MST_Algorithms
2. Minimum Spanning Trees – Prim's algorithm
   - https://ebooks.inflibnet.ac.in/csp01/chapter/minimum-spanning-trees-prims-algorithm/
3. Prim's and Kruskal's Algorithm for MST
   - https://byjus.com/gate/difference-between-prims-and-kruskal-algorithum/
4. https://cp-algorithms.com/graph/mst_prim.html
5. Which Minimum Spanning tree Algorithm is better
   - https://www.javatpoint.com/which-minimum-spanning-tree-algorithm-is-better