Minimum Spanning Tree Based Network Design Using Prim's and Kruskal's Algorithms

Sanjhai M.L[1], Rahul A[2]

[1] Kamaraj College of Engineering and Technology, Virudhunagar, Tamilnadu, India
[2] Kamaraj College of Engineering and Technology
24ucs139@kamarajengg.edu.in

How to Cite this Article:

M.L, S. (2026). Minimum Spanning Tree Based Network Design Using Prim's and Kruskal's Algorithms. International Journal of Creative and Open Research in Engineering and Management, 02(04). https://doi.org/10.55041/ijcope.v2i3.228

https://doi.org/10.55041/ijcope.v2i3.228

Abstract

Efficient network design plays a critical role in modern communication systems. The primary objective of network optimization is to connect multiple nodes with minimum cost while maintaining reliable connectivity. Graph theory provides effective methods for solving such problems using Minimum Spanning Tree (MST) algorithms. A Minimum Spanning Tree connects all vertices in a graph with the minimum possible total edge weight while avoiding cycles. This paper presents a comparative analysis of two widely used MST algorithms: Prim's Algorithm and Kruskal's Algorithm. The algorithms are implemented using Python and evaluated based on computational complexity and performance. Mathematical formulations, algorithmic analysis, and experimental results are presented to demonstrate their effectiveness in designing optimal communication networks such as campus networks and telecommunication infrastructure.

Keywords: Graph Theory, Minimum Spanning Tree, Prim's Algorithm, Kruskal's Algorithm, Network Optimization, Greedy Algorithms, Python

1. Introduction

Modern communication networks consist of multiple interconnected devices such as computers, routers, and servers. The design of such networks requires minimizing the cost of connecting nodes while ensuring that all nodes remain reachable.

Graph theory provides a mathematical framework to represent such networks.

A graph can be defined as shown in (1):

G = (V, E) (1)

Where:

V = $\{v_1, v_2, v_3 \dots v_n\}$ represents the set of vertices E = $\{e_1, e_2, e_3 \dots e_m\}$ represents the set of edges

Each edge has an associated weight as shown in (2):

w : E → R
(2)

where **w(e)** represents the cost or distance of the connection.

The objective of the Minimum Spanning Tree problem is to identify a subset of edges that connects all vertices while minimizing the total cost.

The total cost of the network is given by as shown in (3):

Cost(T) = Σ w(e), where e ∈ T (3)

Minimum Spanning Tree algorithms are widely used in several applications including:

- Computer network design
- Transportation planning
- Electrical grid optimization
- Wireless sensor networks
- Image segmentation

2. Mathematical Model of the Problem

Let:

G = (V, E)

be a connected undirected weighted graph.

Where:

$|V| = n$
$|E| = m$

A spanning tree **T** must satisfy as shown in (4):

$$|T| = n - 1 \quad (4)$$

and

$T \subseteq E$

The optimization objective is as shown in (5):

$$\textbf{Minimize } \Sigma\, w(u,v), \textbf{ where } (u,v) \in T \quad \textbf{(5)}$$

Subject to:

Connectivity constraint Acyclic constraint
The MST must satisfy the following properties.

Cut Property

For any cut (S, V−S), the minimum weight edge crossing the cut
belongs to the MST.

Cycle Property

For any cycle in the graph, the edge with the maximum weight cannot belong to the MST.

3. Minimum Spanning Tree Theory

A spanning tree connects all vertices of a graph without forming cycles.

For a graph with **n vertices**, the spanning tree must contain as shown in (6):

n − 1 edges
(6)

Total weight of spanning tree as shown in (7):

$$\textbf{W(T)} = \Sigma\, w(e_i) \quad \textbf{(7)}$$

The Minimum Spanning Tree is the spanning tree with the minimum total weight.

$$\textbf{MST} = \textbf{argmin } W(T) \quad \textbf{(8)}$$

4. Prim's Algorithm

Prim's algorithm builds the Minimum Spanning Tree by expanding from a starting vertex.

At each step, the algorithm selects the minimum weight edge connecting a visited vertex to an unvisited vertex.

Let:

S = set of visited vertices

Edge selection condition as shown in (9):

$$\mathbf{e = min \{ w(u,v) \mid u \in S \text{ and } v \notin S \}} \quad \mathbf{(9)}$$

Algorithm Steps:

Choose an initial vertex. Add the vertex to set S.
Select the smallest edge connecting S to V−S.

Add the edge to MST.

Repeat until all vertices are included.

Time Complexity:

O(E log V) when using a priority queue.

5. Kruskal's Algorithm

Kruskal's algorithm builds the MST by selecting edges in increasing order of weight.

Edges are selected while ensuring no cycle is formed. Let:
$E = \{e_1, e_2, e_3 \dots e_m\}$

Sort edges such that as shown in (10):

$$\mathbf{w(e_1) \leq w(e_2) \leq \dots \leq w(e_m)} \quad \mathbf{(10)}$$

Edges are added sequentially as shown in (11):

$$\mathbf{T = T \cup \{e_i\}} \quad \mathbf{(11)}$$

if the edge does not form a cycle.

Cycle detection is performed using the **Union-Find data structure**. Time complexity:
O(E log E)

6. Algorithm Analysis

The computational efficiency of MST algorithms depends on the graph structure.

Prim's algorithm complexity as shown in (12):

$$\mathbf{T_{prim} = O(E \log V)} \quad \mathbf{(12)}$$

Kruskal's algorithm complexity as shown in (13):

$$\mathbf{T_{kruskal} = O(E \log E)} \quad \mathbf{(13)}$$

For dense graphs as shown in (14):

$$E \approx V^2 \quad (14)$$

Prim's algorithm performs better.

For sparse graphs as shown in (15):

$$E \approx V \quad (15)$$

Kruskal's algorithm is more efficient.

7. Python Implementation

Example implementation of Prim's Algorithm:

```
import heapq

def prim(graph, start): visited=set([start])
edges=[(cost,start,to) for to,cost in graph[start].items()] heapq.heapify(edges)
mst=[]

while edges: cost,frm,to=heapq.heappop(edges) if to not in
visited:
visited.add(to) mst.append((frm,to,cost))

for next_node,next_cost in graph[to].items(): if next_node not in visited:
heapq.heappush(edges,(next_cost,to,next_node)) return mst
```

8. Experimental Example

Consider a network graph:

Edge	Weight
A-B	4
A-C	2
B-C	1
B-D	5
C-D	8
C-E	10
D-E	2

Selected MST edges: B-C = 1
A-C = 2
D-E = 2
B-D = 5

Total cost:
Cost = 1 + 2 + 2 + 5

Cost = **10**

9. Applications of MST

Minimum Spanning Trees have several real-world applications are : Computer network topology design, Road and railway network planning, Electrical power grid optimization, Image segmentation, Wireless sensor networks, Cluster analysis in data mining

10. Conclusion

This paper presented a detailed analysis of Minimum Spanning Tree algorithms for network optimization. Prim's algorithm and Kruskal's algorithm were studied with mathematical formulation, algorithmic implementation, and performance comparison. Both algorithms efficiently produce optimal solutions for network design problems. The results indicate that Prim's algorithm performs well for dense graphs, while Kruskal's algorithm is more suitable for sparse graphs. Future work may include integrating dynamic graph algorithms and applying MST techniques to large-scale smart infrastructure systems.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, Cambridge, MA, USA: MIT Press.

[2] J. Kleinberg and É. Tardos, *Algorithm Design*, Boston, MA, USA: Pearson.

[3] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, Boston, MA, USA: Pearson.

[4] R. E. Tarjan, *Data Structures and Network Algorithms*, Philadelphia, PA, USA: SIAM.

[5] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Englewood Cliffs, NJ, USA: Prentice Hall.

[6] D. B. West, *Introduction to Graph Theory*, Upper Saddle River, NJ, USA: Prentice Hall.

[7] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, New York, NY, USA: Elsevier.

[8] R. Diestel, *Graph Theory*, Berlin, Germany: Springer.

[9] S. Even, *Graph Algorithms*, Cambridge, UK: Cambridge University Press.

[10] J. Gross and J. Yellen, *Graph Theory and Its Applications*, Boca Raton, FL, USA: CRC Press.

[11] S. S. Skiena, *The Algorithm Design Manual*, New York, NY, USA: Springer.

[12] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Mineola, NY, USA: Dover Publications.