JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

TRANSPORTATION PROBLEM ANALYSIS USING GRAPH THEORY

Gayathri N[1], Shalini R[2], Saranya R[3], Raashmi R[4]

Department of Mathematics, Providence College for Women, Coonoor.

ABSTRACT:

The objective of this study is to examine the solution of transportation problem using graph theory. A general study is made on graph theory and its applications. The Prim's algorithm, Kruskal's algorithm, Dijkstra's algorithm are discussed on finding minimum spanning tree (MST) and shortest path. An analysis is made on solving transportation problem using Kruskal's algorithm and Dijkstra's algorithm. It is obtained that Dijkstra's method is most effective while comparing with the other method to find the shortest path.

1. INTRODUCTION

Graph theory is an important branch of mathematics that deals with the study of relationships between pairs of objects. These relationships are represented using mathematical structures known as graphs. A graph consists of vertices (also referred to as nodes or points) that are connected by edges, which may also be called arcs or lines. The foundation of graph theory was laid in the 18th century by the Swiss mathematician Leonhard Euler. The field developed from Euler's analysis of the Königsberg bridge problem, where he investigated whether it was possible to traverse each of the city's seven bridges exactly once in a single walk. This study resulted in the introduction of Eulerian graphs. The first formal textbook on graph theory was authored by Dénes Kőnig and published in 1936. Later, in 1969, Frank Harary published a book that has since been widely recognized as the most authoritative and influential text in the field.

Graph theory provides effective solutions to many real-world problems. For instance, graph-based algorithms are employed in applications such as Google Maps to determine the shortest path between two locations for navigation. In addition, graph theory has wide applications across several disciplines, including Chemistry, Physics, Biology, Genetics, Communication Networks, Computer Science, Operations Research, Linguistics, Architecture, Electrical Engineering, Civil Engineering, Psychology, Sociology, and Economics.

2.PRELIMINARIES

In this section some fundamental definitions in Graph Theory are given to have a better understanding .

2.1 Graph

A graph G is made up of three basic components: a non-empty set of vertices V(G), a set of edges $E(G)$, and an incidence function ΨG. The vertices represent the points in the graph, while the edges connect pairs of vertices. The incidence function tells us which two vertices are connected by each edge. The sets of vertices and edges are always separate from each other.

2.2 Subgraph

A **subgraph** of a graph G is a graph H is such that $(H) \subseteq (G)$ and $E(H) \subseteq E(G)$ and the assignment of endpoints to the edges in H is the same as in G.

2.3 Vertices

Vertices are also known as nodes, points and (in social networks) as actors, agents or players.

2.4 Edges

Edges are also known as lines and (in social networks) as ties or links. An edge e = (u, v) is defined by the unordered pair of vertices that serve as its end points.

2.5 Adjacent

Two vertices u and v are **adjacent** if there exists an edge (u, v) that connects them.

2.6 Incident

An edge (u, v) is said to be **incident** upon nodes u and v.

2.7 Self loop

An edge e = (u, u) that links a vertex to itself is known as a **self-loop.**

2.8 Adjacency matrix

Every graph has associated with it an **adjacency matrix,** which is a binary n x n matrix A in which $_j = 1$ and $a_{ji} = 1$ if vertex is v_i adjacent to vertex v_j,and $a_{ij} = 0$ and $a_{ji} = 0$.

2.9 Component

A **component** of a graph is defined as a maximal subgraph in which a path exists from every node to every other (i.e., they are mutually reachable). The size of a component is defined as the number of nodes it contains. A connected graph has only one component.

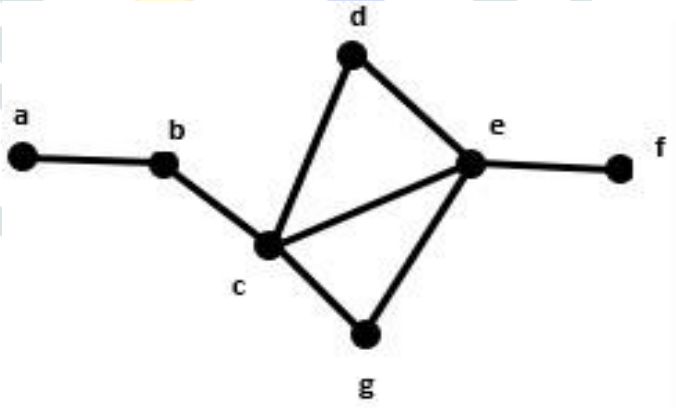


Figure 1

2.10 Walk

A sequence of adjacent vertices $v_{0,1}$, … v_n is known as a **walk**. In Figure 1, the sequence a, b, c, b, c, g is a walk. A walk can also be seen as a sequence of incident edges, where two edges are said to be incident if they share exactly one vertex.

2.11 Path

A walk in which no vertex occurs more than once is known as a **path**. In Figure 1, the sequence a, b, c, d, e, f is a path.

2.12 Trail

A walk in which no edge occurs more than once is known as a **trail**. In Figure 1, the sequence a, b, c, e, d, c, g is a trail but not a path. Every path is a trail, and every trail is a walk.

2.13 Cycle

A **cycle** can be defined as a closed path in which $n \geq 3$. The sequence c, e, d, c in Figure 1 is a cycle.

2.14 Tree

A **tree** is a connected graph that contains no cycles. In a tree, every pair of points is connected by a unique path. That is, there is only one way to get from A to B.

2.15 Spanning tree

A **Spanning tree** for a graph G is a sub-graph of G which is a tree that includes every vertex of G.

2.16 Length of a walk

The **length of a walk** (and therefore a path or trail) is defined as the number of edges it contains. For example, the path a, b, c, d, e in Figure 1 has length 4.

2.17 Degree of the vertex

The number of vertices adjacent to a given vertex is called the **degree of the vertex** and is denoted d(v).

3. MINIMUM SPANNING TREE (MST)

A **spanning tree's cost** is determined by calculating the total sum of the weights assigned to its edges. Since a single graph can produce multiple spanning trees, the **Minimum Spanning Tree** is defined as the specific tree that yields the lowest possible total cost compared to all other potential spanning trees.

Practical Applications

Minimum spanning trees are fundamental to network design and optimization. They serve as a base for algorithms that provide approximate solutions to complex computational challenges, such as the Traveling Salesman Problem, the multi-terminal minimum cut problem, and minimum cost weighted perfect matching.

Beyond theoretical mathematics, MSTs are utilized in several specialized fields:

- Cluster Analysis: For grouping data points in statistics and machine learning.
- Handwriting Recognition: For identifying patterns in written characters.
- Image Segmentation: For partitioning digital images into multiple segments for easier analysis.

Primary Algorithms

To identify the Minimum Spanning Tree within a graph, two primary and widely recognized algorithms are used:

- Prim's Algorithm: Builds the tree by adding the cheapest connection from the current tree to a new vertex.
- Kruskal's Algorithm: Finds the MST by sorting all edges by weight and adding the smallest ones that do not create a cycle.

i. Prim's Algorithm

Prim's Algorithm is a **vertex-based** approach used to construct a Minimum Spanning Tree (MST). It begins with a single starting node and progressively expands by exploring adjacent nodes and their connecting edges.

The process begins with an empty spanning tree and involves managing two distinct groups of vertices:

- **The MST Set:** Vertices that have already been integrated into the growing tree.
- **The Non-MST Set:** Vertices that have not yet been included.

At each stage of the algorithm, it evaluates all possible edges that bridge these two sets (connecting a vertex in the MST set to one outside of it). From these candidates, the algorithm selects the **edge with the lowest weight**. Once this edge is chosen, the vertex at its other end is moved into the MST set. This cycle repeats until all vertices are connected, ensuring the total weight of the tree is minimized.

ii. Kruskal's Algorithm

Kruskal's Algorithm is a **greedy approach** used to determine the Minimum Spanning Tree (MST) of a graph. It operates by prioritizing edges based on their weight, starting with the smallest and ending with the largest. Because the algorithm consistently selects the best available option at each individual stage to reach a global optimum, it is classified as a **Greedy Algorithm**.

Procedure for Finding the MST

The algorithm follows these specific steps to ensure an efficient and cycle-free connection of all vertices:

- **Edge Sorting:** Arrange every edge in the graph in ascending (non-decreasing) order according to its weight.
- **Edge Selection & Validation:** Select the edge with the lowest weight. Before adding it to the MST, verify if its inclusion would create a **cycle** (a closed loop).
    - If no cycle is formed, the edge is officially included.
    - If a cycle is created, the edge is discarded.
- **Iteration:** Continue repeating the second step until the spanning tree contains exactly (V - 1) edges (where V represents the total number of vertices).

4. SHORTEST PATH ANALYSIS

The **shortest path** refers to the route between two specific vertices (or nodes) in a graph that results in the lowest possible cumulative edge weight. This problem is applicable across various graph types, including **undirected, directed, or mixed**. While the general concept is often illustrated using undirected graphs, directed graphs require that the path follows the specific orientation of the connecting edges.

To differentiate from other formats, this is sometimes referred to as the **single-pair shortest path problem**. However, there are several key variations:

- **Single-Source Shortest Path:** The objective is to determine the shortest routes from one fixed starting vertex *v* to every other vertex within the graph.
- **Single-Destination Shortest Path:** This involves finding the shortest paths from all vertices in a directed graph to one specific target vertex *v*. By simply reversing the direction of all edges (arcs), this can be solved using single-source methods.
- **All-Pairs Shortest Path:** This more complex version requires calculating the shortest distances between every possible pair of vertices v, v' in the entire network.

One of the most critical methods for addressing these challenges is **Dijkstra's Algorithm**. It is specifically designed to solve the single-source shortest path problem, provided that all edge weights are **non-negative**. This section provides a detailed overview of how Dijkstra's algorithm functions.

Dijkstra's algorithm

Dijkstra's Algorithm is a popular algorithm used to find the shortest path or minimum distance between a starting point (source node) and other nodes in a weighted graph. It works by finding the shortest possible distance from the source node to every other node in the graph. While doing this, it creates a shortest-path tree that shows the most efficient routes from the starting node to the remaining nodes. The algorithm uses the weights assigned to the edges to calculate the total distance between nodes and selects the path with the lowest total weight. It is also

known as the **Single-Source Shortest Path Algorithm** because it finds the shortest paths from one source node to all other nodes.

Dijkstra's Algorithm works step by step, updating the shortest distances as it explores the graph. It is different from a Minimum Spanning Tree because its main purpose is to find the shortest paths from a particular starting node, rather than connecting all nodes with the minimum total edge weight.

Step-by-Step Process of Dijkstra's Algorithm

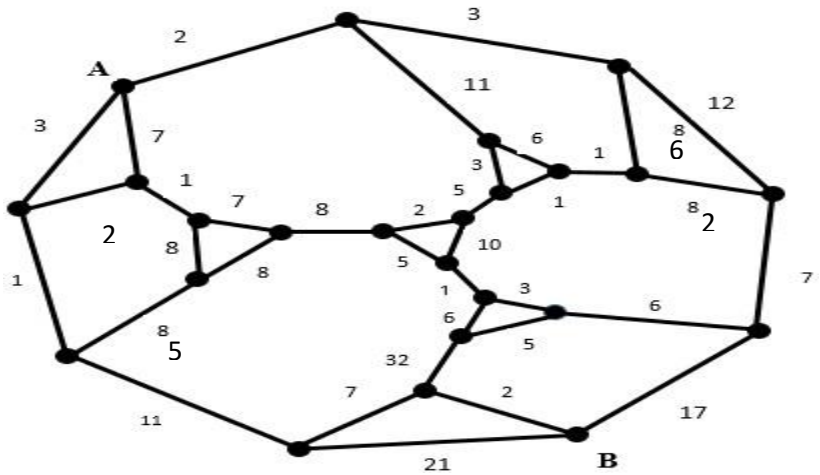
1. First, mark all the nodes in the graph as unvisited.
2. Assign a distance of **0** to the starting node (source node) and infinity (∞) to all the other nodes.
3. Select the starting node and mark it as the current node.
4. Check all the unvisited neighbouring nodes of the current node. Calculate their distances by adding the current node's distance to the weight of the connecting edge.
5. Compare the newly calculated distance with the previously assigned distance of each neighbouring node. If the new distance is smaller, update it with the new value.
6. Once all the unvisited neighbours have been checked, mark the current node as visited.
7. If the destination node has been visited, the algorithm can stop because its shortest distance has been found.
8. If the destination has not been visited, select the unvisited node with the smallest recorded distance and make it the new current node. Repeat the process from Step 4 until the destination is reached or all reachable nodes have been visited.

5. SOLVING TRANSPORTATION PROBLEM USING KRUSKAL'S ALGORITHM AND DIJSTRA'S ALGORITHM

Graph theory plays a significant role in Operations Research by providing mathematical tools to solve complex real-world problems. It is widely used in transportation, logistics, scheduling, resource allocation, and network optimization. Network models help represent relationships and connections between different elements in a simplified and structured way. These models make it easier to analyze different possibilities and identify optimal solutions. Various graph algorithms are used to minimize costs, reduce time, and improve efficiency. Thus, graph theory serves as an effective approach to decision-making and problem-solving in many fields.

EXAMPLE:

A company needs to transport its manufactured products from City A to City B. Since several routes connect these two cities, the company wants to identify the most efficient and cost-effective route. Given a graph representing the different routes and their associated costs, how can we determine the shortest path with the minimum transportation cost? Which route would be the best choice for the company?



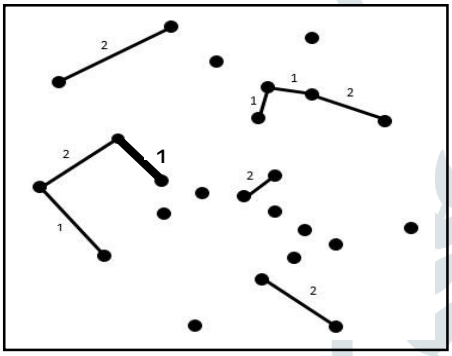
There are different path options to reach from node A to node B, but our aim is to find the shortest path with a minimum transportation costs, this requires a lot of efforts. Solving this problem by finding

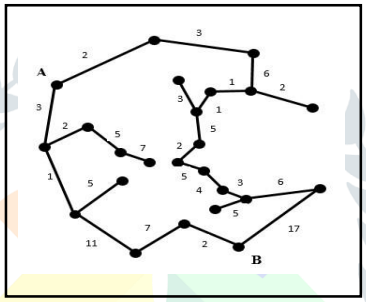
- Minimum spanning tree using Kruskal's algorithm
- Dijkstra's Shortest path algorithm

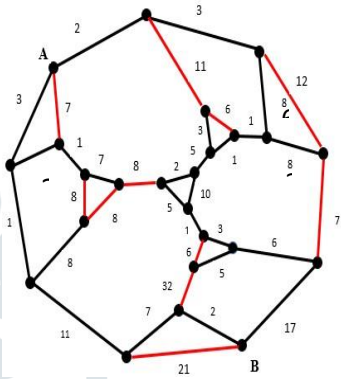
SOLUTION OF TRANSPORTATION PROBLEM WITH MINIMUM SPANNING TREE USING KRUSKAL'S ALGORITHM

(i) MINIMUM SPANNING TREE

The Minimum Spanning Tree is constructed by starting with all the vertices and adding edges in increasing order of weight, beginning with the lowest cost (1). Each edge is added only if it does not form a cycle. This process continues until all vertices are connected. The edges marked in red are removed because they would create cycles. Thus, we obtain the Minimum Spanning Tree with the minimum total cost.

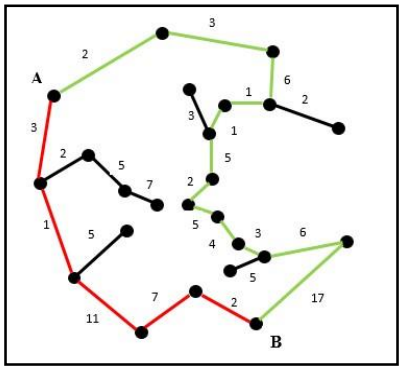






(ii) MINIMUM COST PATH:

From the Minimum Spanning Tree shown in below are able to find the minimum cost path (trajectory) from node A to node B, there are two alternative ways to reach from node A to node B, which are distinguished by red and green lines.



Start with first option to calculate the distance from node A to node B
(**green line**), the result is as follows:

$\mu = 2+3+6+7+4+5+2+5+4+3+6+17 = 64$ units

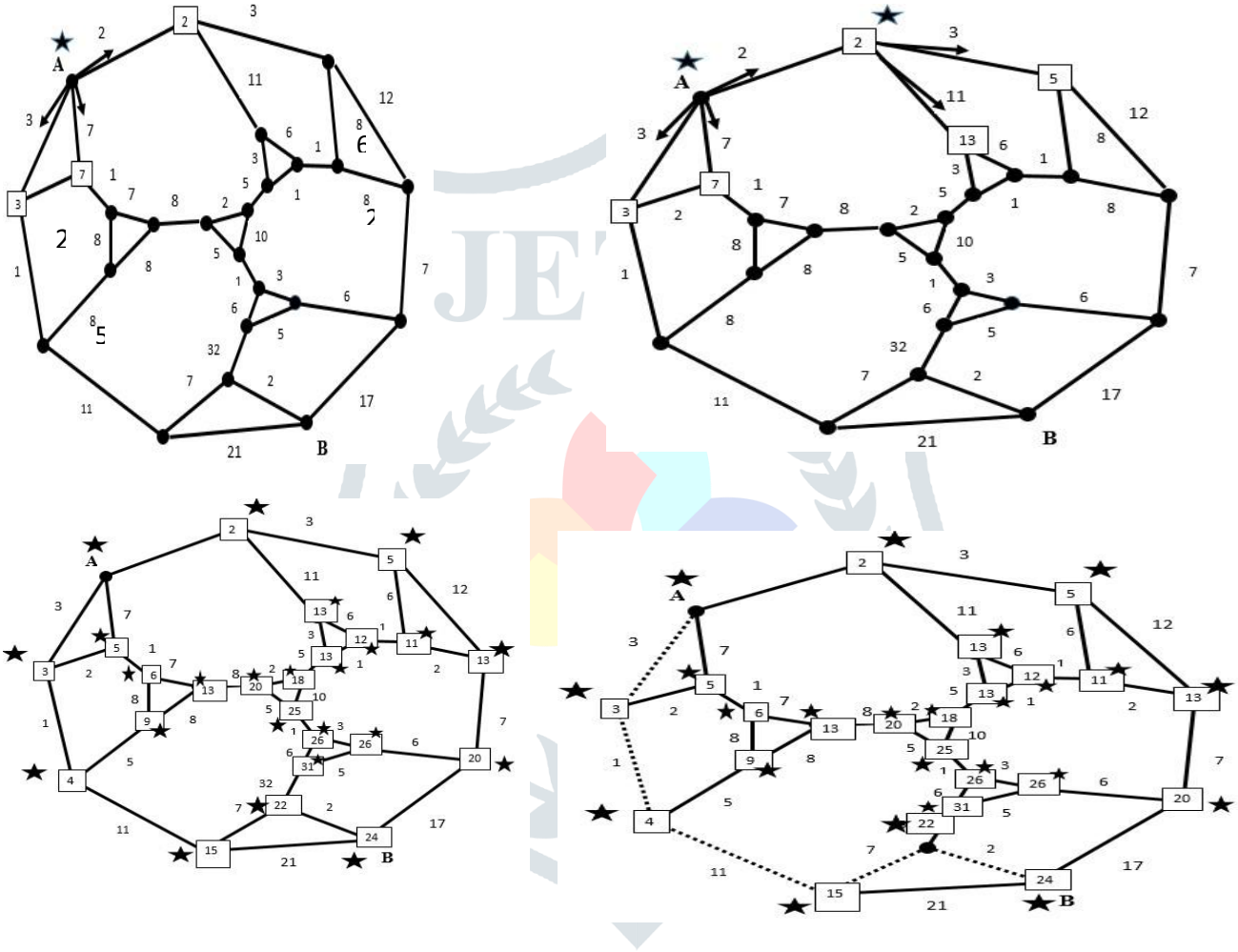
(this is the most expensive path)

For the second option (**red line**):

$\mu = 3+1+11+7+2 = 24$ units. This means that the second option represents the minimum cost path from node A to node B.

SOLUTION OF TRANSPORTATION PROBLEM USING DIJSKTRA ALGORITHM:

Dijkstra's algorithm is used to find the shortest distance from a starting node to all other nodes. First, we select node A as the permanent node and calculate the distances to its neighboring nodes. The node with the smallest distance (2) is then selected as the next permanent node. We repeat this process by adding the distance of the permanent node to its neighboring nodes and updating the shortest distances. If a new distance is greater than the existing one, the previous shorter distance is retained. For example, from node 6, the distance 6 + 8 = 14 is greater than 9, so 9 remains unchanged. This process continues until all nodes are visited, giving the shortest paths from node A.



The dotted line in the final figure represents the minimum cost path from node A to node B Minimum cost = 24 units.

CONCLUSION

An analysis was conducted to study how graph theory can be used to solve transportation problems using Kruskal's and Dijkstra's algorithms. The main conclusions are as follows:

- **Dijkstra's algorithm** is used to find the shortest and least-cost path between two nodes, while **Kruskal's algorithm** is used to find the minimum spanning tree that connects all the given nodes with the lowest total cost.
- Dijkstra's algorithm focuses on finding the best route between two specific nodes, whereas Kruskal's algorithm connects all the nodes efficiently.

- The results show that Dijkstra's algorithm is effective in finding the shortest and most economical route from node A to node B. Kruskal's algorithm also provides an efficient solution by connecting all nodes at the minimum total cost.

Therefore, a company can choose Dijkstra's algorithm when its main objective is to find the shortest and most cost-effective route between two locations.

REFERENCES

1. Lee, J. Y. (2014). An efficient implementation of Kruskal's algorithm for a minimum spanning tree. *Journal of The Korea Society of Computer and Information.* Article link .
2. Choi, M. B., & Lee, S. U. (2013). An efficient implementation of Kruskal's and reverse-delete minimum spanning tree algorithm. *Journal of the Institute of Webcasting, Internet and Telecommunication.* https://doi.org/10.7236/jiibc.2013.13.2.103
3. Li, H., Xia, Q., & Wang, Y. (2017). Research and improvement of Kruskal algorithm. *Journal of Computer and Communications.* https://doi.org/10.4236/jcc.2017.512007
4. (2023). Minimum spanning tree method for sparse graphs. *Journal of Mathematics.* https://doi.org/10.1155/2023/8591115
5. Haymond, R. E., Jarvis, J. P., & Shier, D. R. Computational methods for minimum spanning tree algorithms. *SIAM Journal on Computing.* Article link .
6. Lee, S. U. (2014). A point-to-point shortest path search algorithm in an undirected graph using minimum spanning tree. *Journal of The Korea Society of Computer and Information.* Article link .
7. Peyer, S., Rautenbach, D., & Vygen, J. (2009). A generalization of Dijkstra's shortest path algorithm with applications to VLSI routing. *Journal of Discrete Algorithms.* https://doi.org/10.1016/j.jda.2007.08.003
8. Ahuja, R. K., Mehlhorn, K., Orlin, J., & Tarjan, R. E. (1990). Faster algorithms for the shortest path problem. *Journal of the ACM.* https://doi.org/10.1145/77600.77615
9. Idwan, S., & Etaiwi, W. (2011). Dijkstra algorithm heuristic approach for large graph. *Journal of Applied Sciences.* Article link .
10. Rodríguez-Puente, R., & Lazo-Cortés, M. S. (2013). Algorithm for shortest path search in geographic information systems by using reduced graphs. *SpringerPlus, 2,* Article 291. https://doi.org/10.1186/2193-1801-2-291
11. Parmanand, Sahdev, & Dwivedi, A. (2024). Study the optimization of Dijkstra's algorithm. *Journal of Ravishankar University (Part-B: Science).* https://doi.org/10.52228/JRUB.2024-37-2- 18
12. Fredman, M. L., & Willard, D. E. (1994). Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *Journal of Computer and System Sciences.* Article link .