MINISTRY OF EDUCATION, CULTURE AND RESEARCH OF REPUBLIC OF MOLDOVA

TECHNICAL UNIVERSITY OF MOLDOVA

FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS

DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATICS

Algorithm Analysis

Laboratory work 5 :Empirical analysis of algorithms: Dijkstra Algorithm, Floyd-Warshall Algorithm

Elaborated:

st.gr. FAF-211 Corețchi Mihai

Verified:

asist.univ. Fiștic Cristofor

Chișinău, 2023

Content

Introduction

Dijkstra's algorithm and Floyd-Warshall algorithm are two popular algorithms used to solve shortest path problems in graphs.

Dijkstra's algorithm is a single-source shortest path algorithm, which means it finds the shortest path from a single source vertex to all other vertices in a weighted graph. The algorithm maintains a set of visited vertices and a set of unvisited vertices. It starts at the source vertex and iteratively selects the unvisited vertex with the smallest tentative distance, updating the distance of its neighboring vertices. Dijkstra's algorithm is particularly useful for finding the shortest path in a graph with positive edge weights, and it has a time complexity of O(—E—+—V—log—V—), where —E— is the number of edges and —V— is the number of vertices in the graph.

Floyd-Warshall algorithm, on the other hand, is an all-pairs shortest path algorithm, which means it finds the shortest path between all pairs of vertices in a weighted graph. The algorithm maintains a distance matrix, which stores the shortest path distances between all pairs of vertices in the graph. It iteratively updates the distance matrix by considering all intermediate vertices between any two pairs of vertices. Floyd-Warshall algorithm is particularly useful for finding the shortest path in a graph with both positive and negative edge weights, and it has a time complexity of O(—V—$^3), where |V| is the number of vertices in the graph.$

Both Dijkstra's and Floyd-Warshall algorithms are widely used in various fields such as network routing, transportation planning, and computer graphics. Choosing the appropriate algorithm depends on the characteristics of the graph and the specific problem being solved.

Dijkstra's algorithm and Floyd-Warshall algorithm are fundamental algorithms for solving the shortest path problem in a graph. They are widely used in many real-world applications such as network routing, GPS navigation, airline scheduling, and more.

Dijkstra's algorithm is particularly suitable for finding the shortest path in a graph with positive edge weights. The algorithm is widely used in network routing protocols, where the weights of the edges represent the costs of the network links. Dijkstra's algorithm has a time complexity of O(—E—+—V—log—V—), where —E— is the number of edges and —V— is the number of vertices in the graph. The time complexity can be further improved to O(—E—+—V—) using a priority queue data structure.

In summary, Dijkstra's algorithm and Floyd-Warshall algorithm are powerful tools for solving the shortest path problem in graphs. They have their own strengths and weaknesses, and choosing the appropriate algorithm depends on the characteristics of the graph and the specific problem being solved.

Objectives

1. Implement Dijkstra’s algorithm for finding the shortest path from a single source node to all other nodes in a graph with non-negative edge weights.
2. Implement the Floyd-Warshall algorithm for finding the shortest paths between every pair of nodes in a weighted, directed graph.
3. Generate random graphs to be used as input for both Dijkstra’s and Floyd-Warshall algorithms, ensuring various sizes and edge weights.
4. Compare the execution time of Dijkstra’s and Floyd-Warshall algorithms as the number of nodes in the input graph increases.
5. Visualize the input graph and the resulting shortest path trees produced by both Dijkstra’s and Floyd-Warshall algorithms using the NetworkX and Matplotlib libraries.

Dijkstra

The Dijkstra's Algorithm is a widely used method for traversing graphs to determine the shortest path from a starting point to all other nodes with non-negative edge weights. To begin, the algorithm sets the starting node's distance to 0 and assigns infinity to the distance of all other nodes. The algorithm then selects an unvisited node with the smallest known distance and updates the distances of its neighboring nodes by examining whether the sum of the current node's distance and the edge weight to the neighboring node is less than the current distance assigned to the neighbor. If this condition holds true, the neighbor's distance is updated with the new, smaller value. This process is repeated until all nodes have been visited. At the end of the algorithm, the shortest path from the starting node to each of the other nodes in the graph is discovered.

Code:

```
function Dijkstra(Graph, source):
     dist[source] ← 0
     create vertex set Q
     for each vertex v in Graph:
         if v  source
             dist[v] ← INFINITY
         prev[v] ← UNDEFINED
         Q.add_with_priority(v, dist[v])

    while Q is not empty:
        u ← Q.extract_min()
        for each neighbor v of u:
            alt ← dist[u] + length(u, v)
            if alt < dist[v]
                dist[v] ← alt
                prev[v] ← u
                Q.decrease_priority(v, alt)

    return dist, prev
```

Floyd-Warshall

The Floyd-Warshall algorithm is used to find the shortest paths between all pairs of nodes in a directed, weighted graph. This algorithm works by examining each possible combination of nodes and modifying the distance matrix based on the shortest path known between each node pair. First, a distance matrix is created, where diagonal elements are set to 0, and off-diagonal elements are set to edge weights or infinity if no direct edge exists between the nodes. Then, the algorithm iteratively searches for a shorter path between a pair of nodes by passing through an intermediate node. If a shorter path is found, the distance matrix is updated with the new, shorter distance value. This process is repeated for every possible intermediate node. Once completed, the distance matrix contains the shortest path distances between every pair of nodes in the graph.

Code:

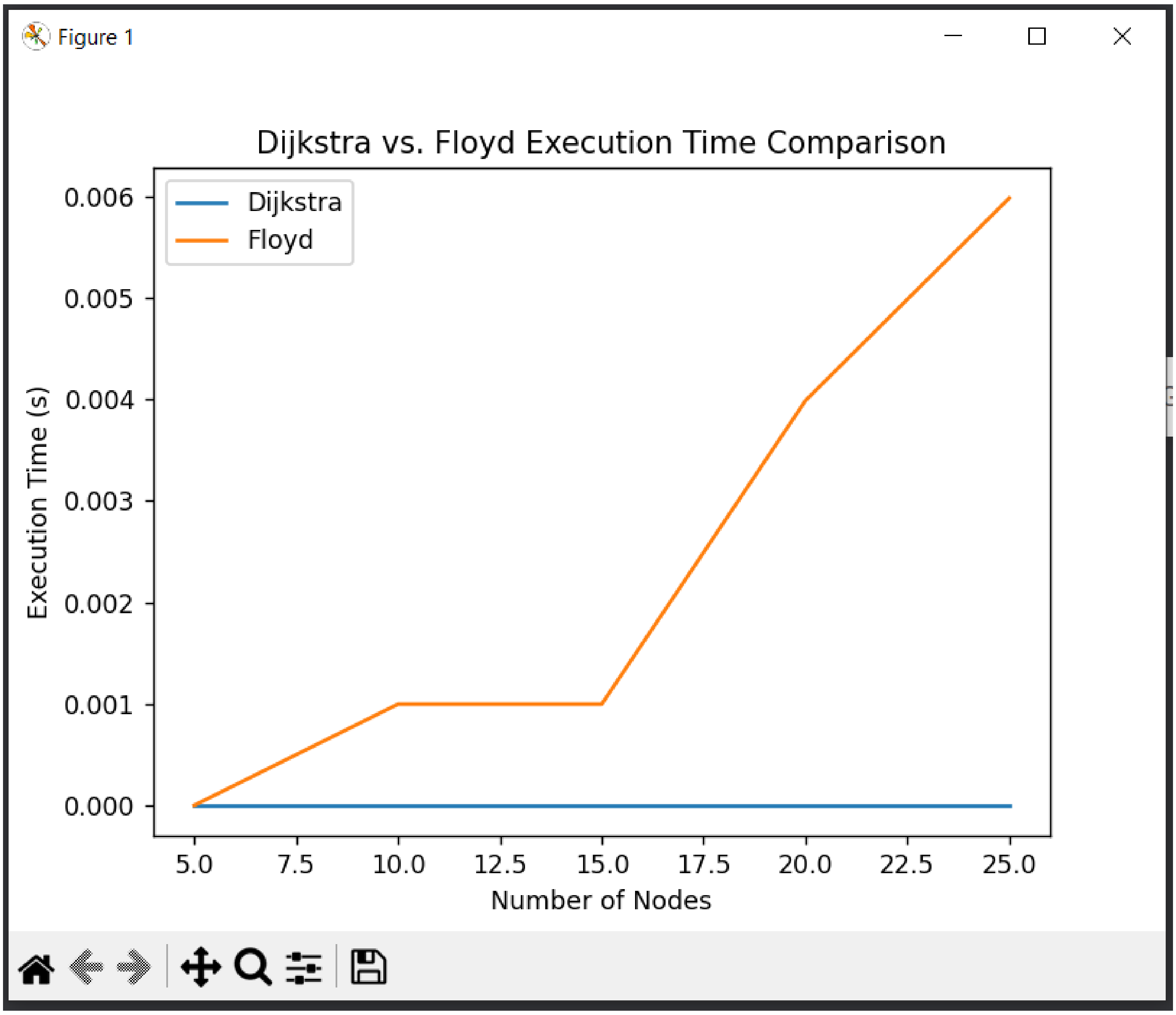
```
function FloydWarshall(Graph):
     let dist be a |V| x |V| matrix of minimum distances initialized as follows:
         - dist[i][j] ← weight(i, j) if (i, j) is an edge in Graph
         - dist[i][j] ← infinity if (i, j) is not an edge in Graph and i  j
         - dist[i][i] ← 0 for all i
     for k from 1 to |V|:
         for i from 1 to |V|:
             for j from 1 to |V|:
                 if dist[i][j] > dist[i][k] + dist[k][j]:
                    dist[i][j] ← dist[i][k] + dist[k][j]
    return dist
```

Implementation

```
def dijkstra(graph, start):
    dist = {node: float('inf') for node in graph}
    dist[start] = 0
    visited = set()

    while len(visited) != len(graph):
        min_node = None
        for node in dist:
            if node not in visited:
                if min_node is None or dist[node] < dist[min_node]:
                    min_node = node
        visited.add(min_node)

        for neighbor, weight in graph[min_node].items():
            new_dist = dist[min_node] + weight
            if new_dist < dist[neighbor]:
                dist[neighbor] = new_dist
    return dist

def floyd(graph):
    nodes = list(graph.keys())
    n = len(nodes)
    dist = np.full((n, n), np.inf)

    for i, node in enumerate(nodes):
        dist[i, i] = 0
        for neighbor, weight in graph[node].items():
            j = nodes.index(neighbor)
            dist[i, j] = weight
```

```
for k in range(n):
    for i in range(n):
        for j in range(n):
            if dist[i, k] + dist[k, j] < dist[i, j]:
                dist[i, j] = dist[i, k] + dist[k, j]

return {nodes[i]: {nodes[j]: dist[i, j] for j in range(n)} for i in range(n)}
```

Screenshot:



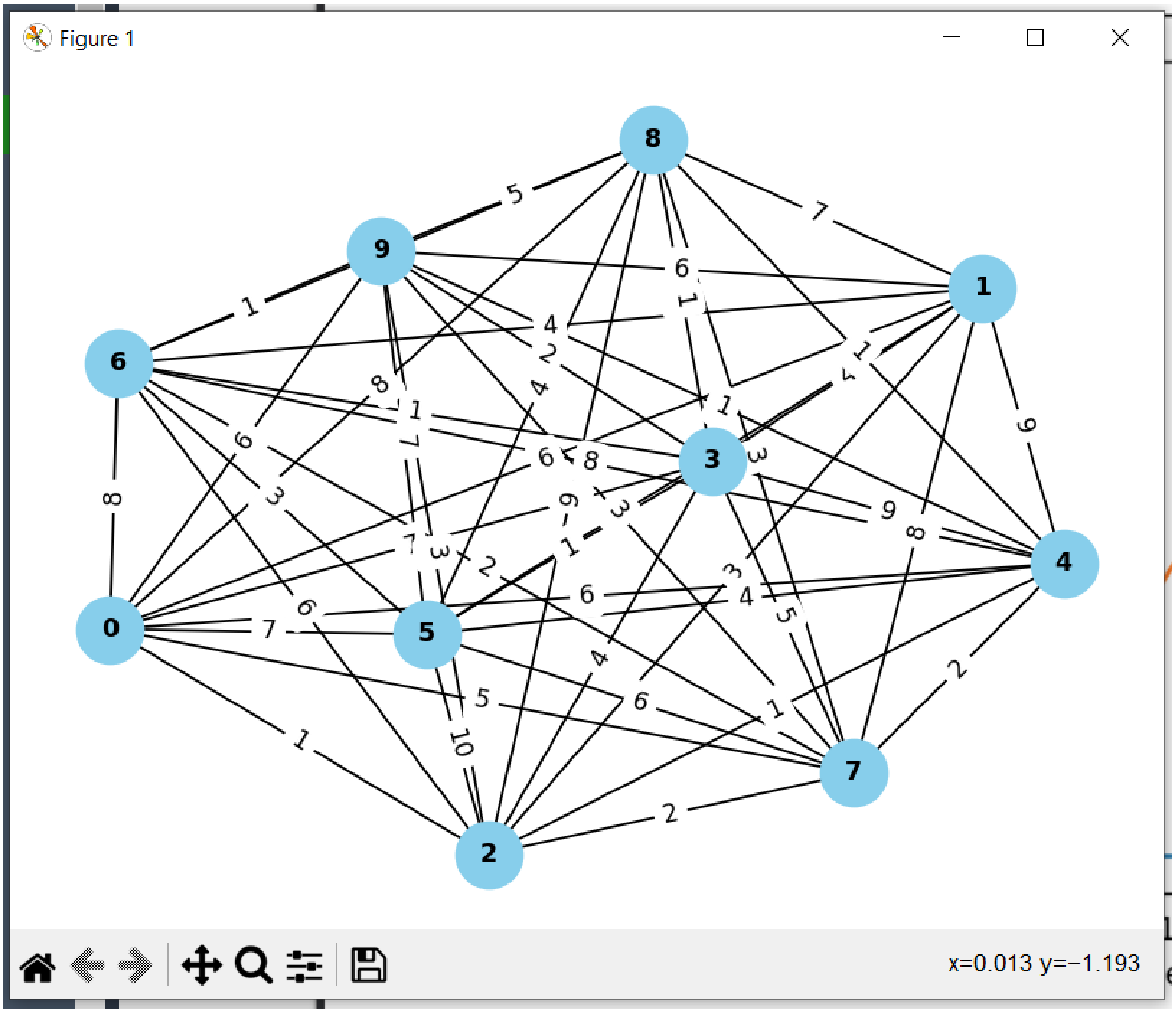


Figure 1
x=0.013 y=−1.193

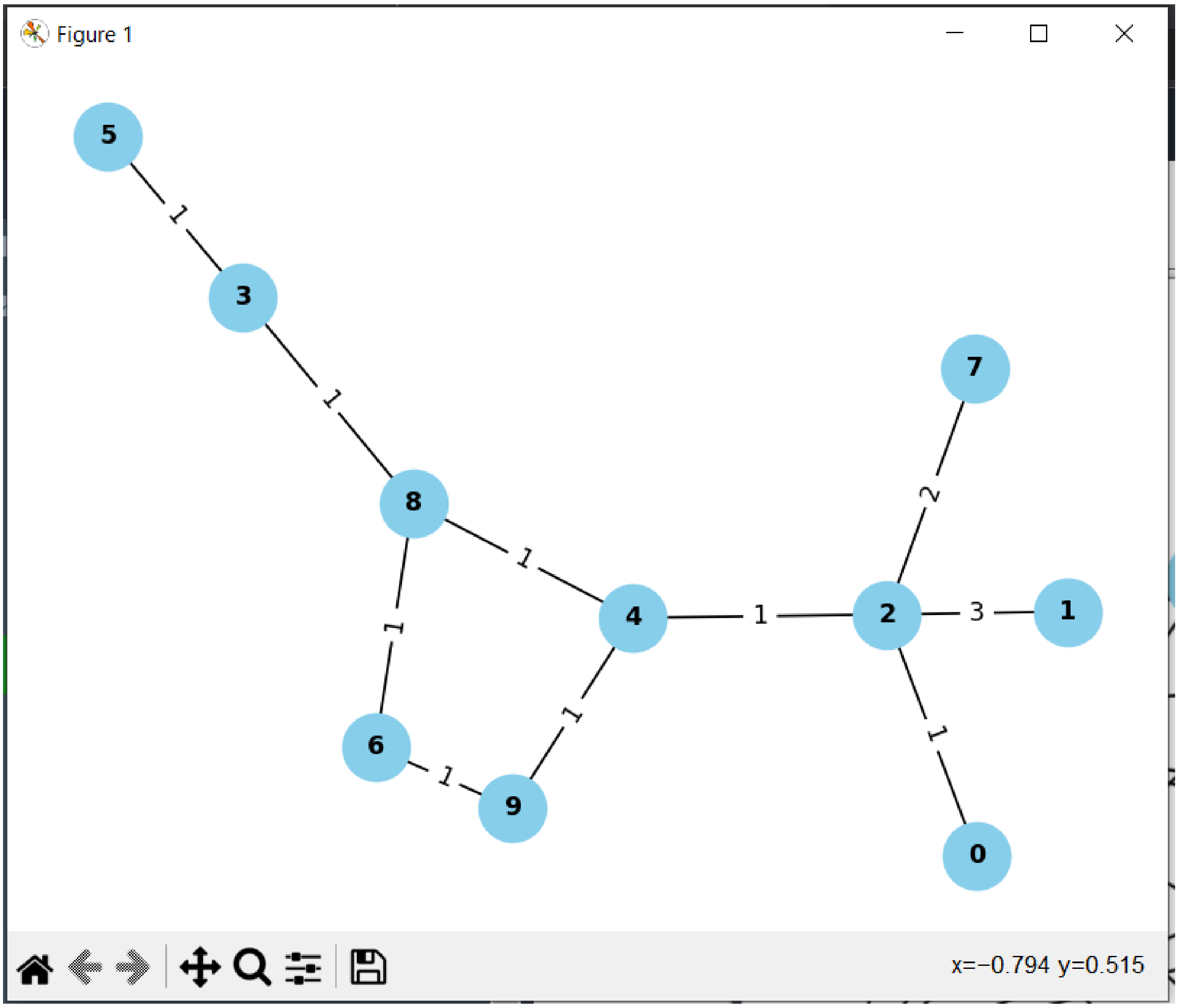


Figure 1
5
3
8
4
2
7
1
0
6
9
1
1
1
1
1
1
1
2
3
1
x=−0.794 y=0.515

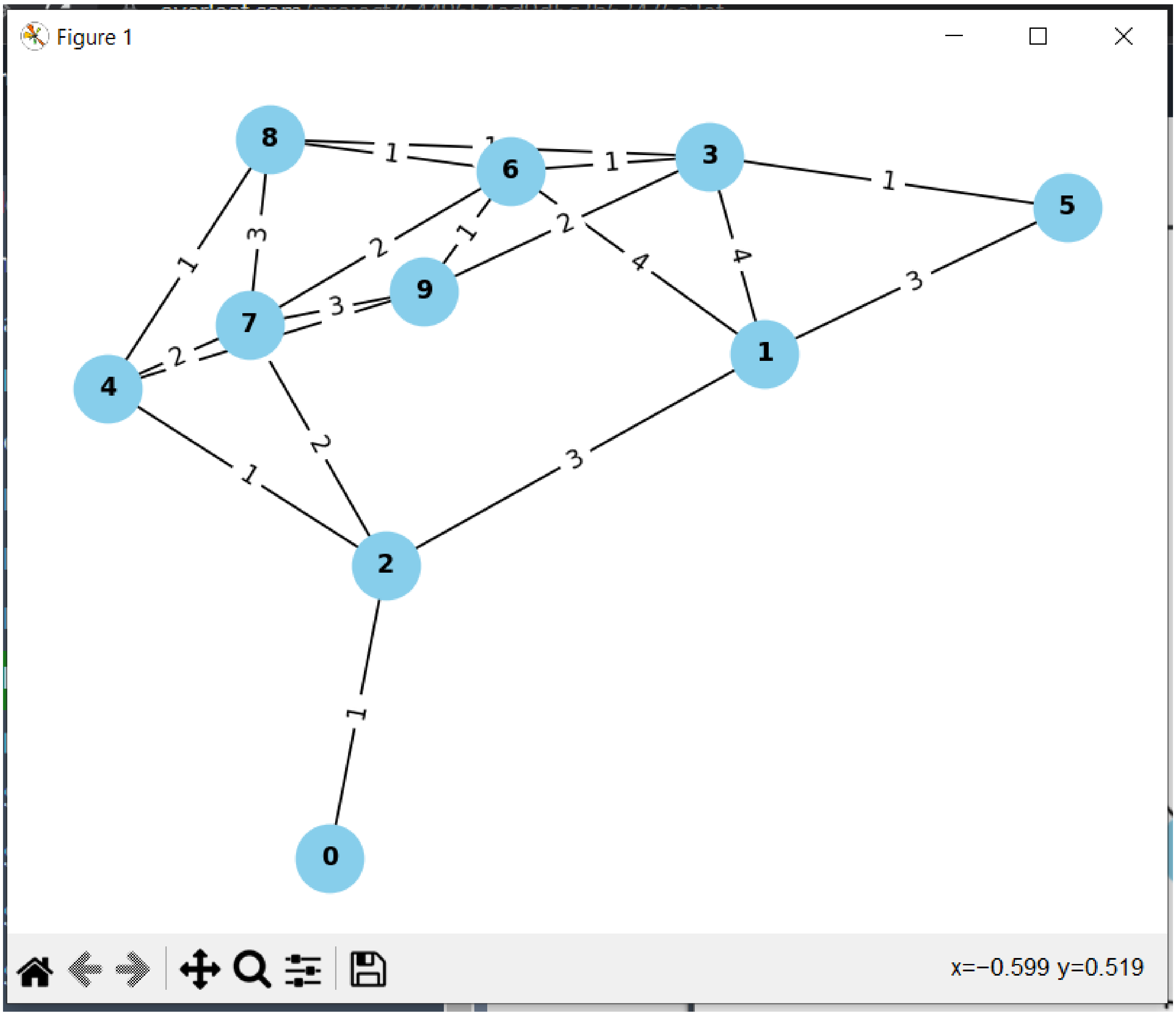


Figure 1
8
6
3
5
9
7
4
1
2
0
1
1
1
1
1
3
2
1
2
4
4
3
3
2
2
1
3
1
x=−0.599 y=0.519

Conclusion

In conclusion, Dijkstra's algorithm and Floyd-Warshall algorithm are two widely used algorithms in graph theory for solving shortest path problems. Dijkstra's algorithm finds the shortest path from a single source vertex to all other vertices in a weighted graph by selecting the vertex with the smallest distance and relaxing its outgoing edges. It is efficient in sparse graphs with non-negative edge weights, making it ideal for problems that involve finding the shortest path between a fixed source and various destinations. However, it does not work well with negative edge weights, and its time complexity can be high in dense graphs. In contrast, Floyd-Warshall algorithm computes the shortest paths between all pairs of vertices in a weighted graph by considering all possible intermediate vertices in a path and updating the shortest path distances accordingly. It is more efficient than running Dijkstra's algorithm for every vertex pair in dense graphs with non-negative edge weights, making it ideal for problems that require finding the shortest path between all pairs of vertices in a graph. It also works well with negative edge weights, but its time complexity can be high in large graphs. It is important to note that both algorithms are designed for graphs with non-negative edge weights. For graphs with negative edge weights, the Bellman-Ford algorithm is more appropriate as it can detect negative cycles and report them. Overall, both Dijkstra's algorithm and Floyd-Warshall algorithm have their advantages and limitations, and the choice of which algorithm to use depends on the specific problem and characteristics of the graph. However, they are both important tools in solving shortest path problems, and they will continue to play a significant role in the future of graph theory and related fields.

Github repository: *https://github.com/eamtcPROG/AA/tree/main/Lab5*