

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	3
Objectives	3
Algorithms	5
Dijkstra	5
Floyd-Warshall	5
Implementation	7
Code	7
Screenshot	7
Conclusion	16

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  $|V| \times |V|$  matrix of minimum distances initialized as follows:  
        -  $\text{dist}[i][j] \leftarrow \text{weight}(i, j)$  if  $(i, j)$  is an edge in Graph  
        -  $\text{dist}[i][j] \leftarrow \text{infinity}$  if  $(i, j)$  is not an edge in Graph and  $i \neq j$   
        -  $\text{dist}[i][i] \leftarrow 0$  for all  $i$   
    for k from 1 to  $|V|$ :  
        for i from 1 to  $|V|$ :  
            for j from 1 to  $|V|$ :  
                if  $\text{dist}[i][j] > \text{dist}[i][k] + \text{dist}[k][j]$ :  
                     $\text{dist}[i][j] \leftarrow \text{dist}[i][k] + \text{dist}[k][j]$   
    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:

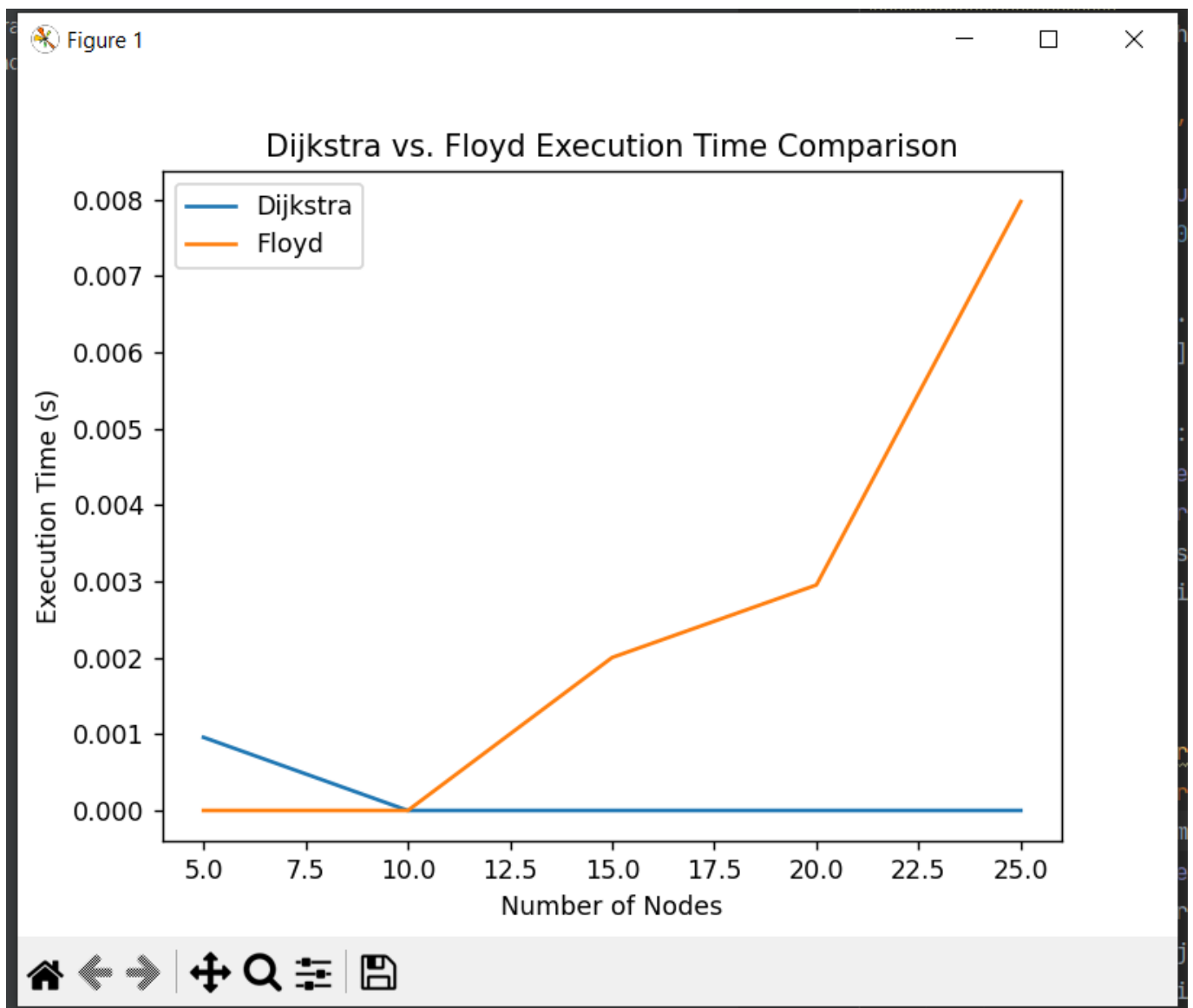
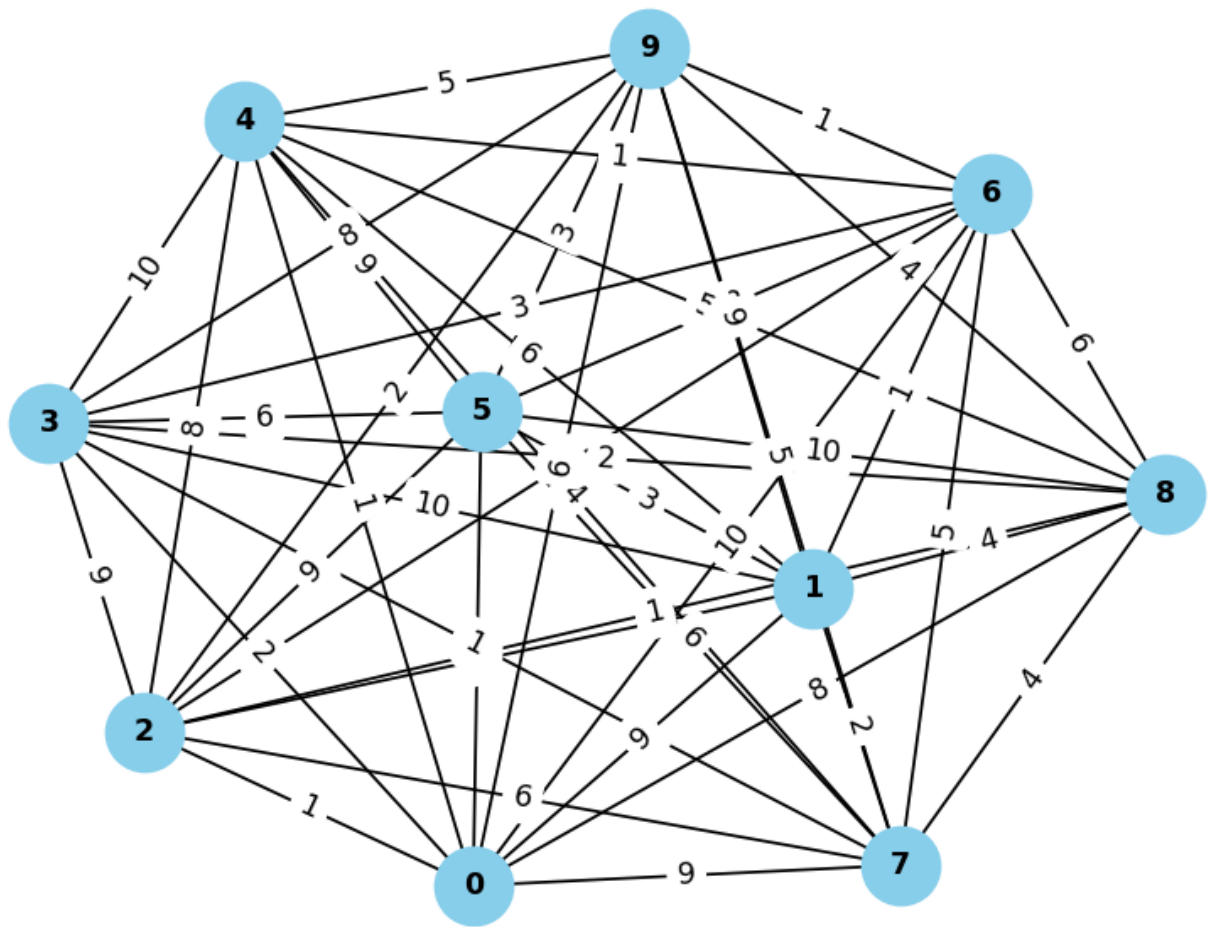
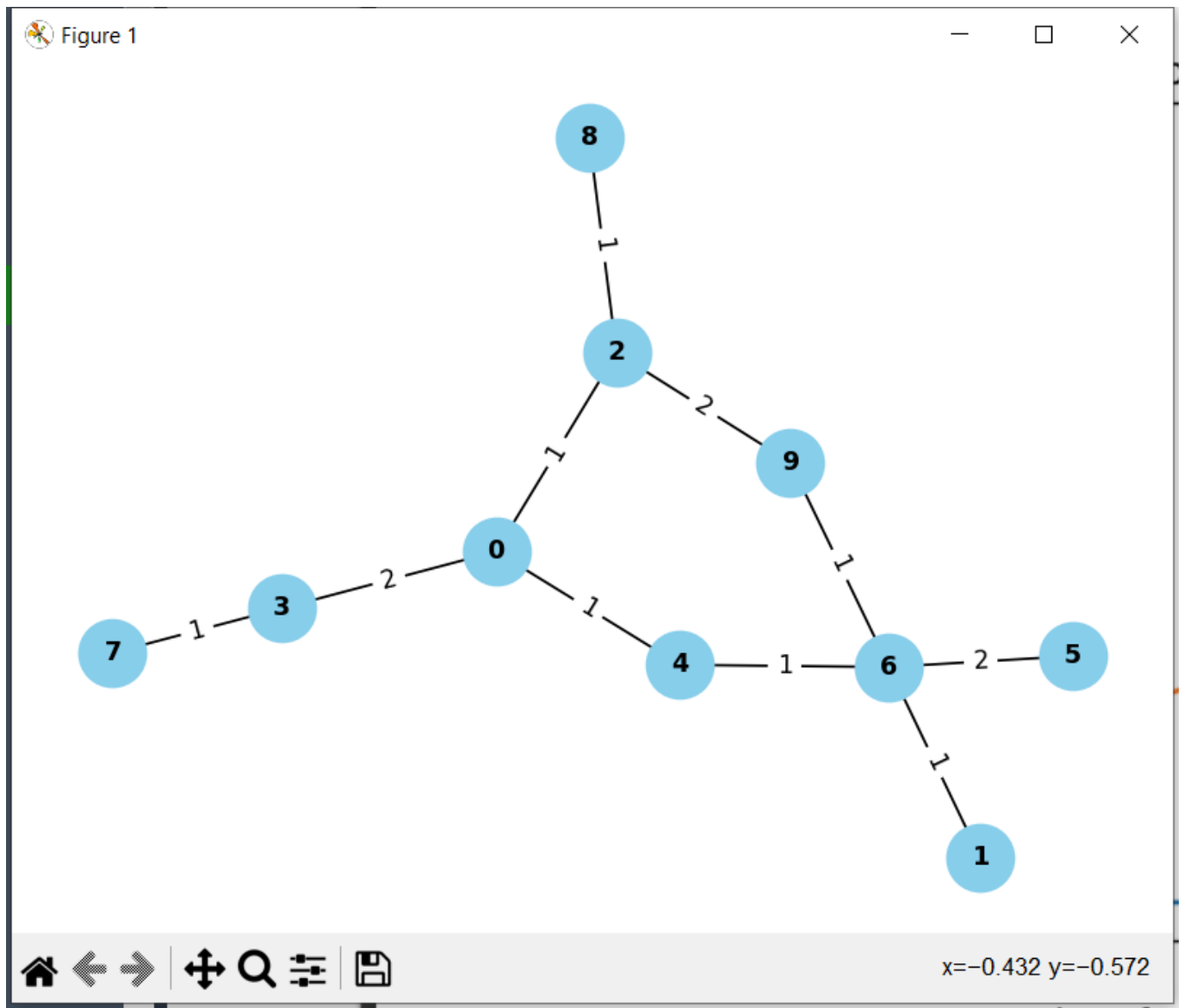
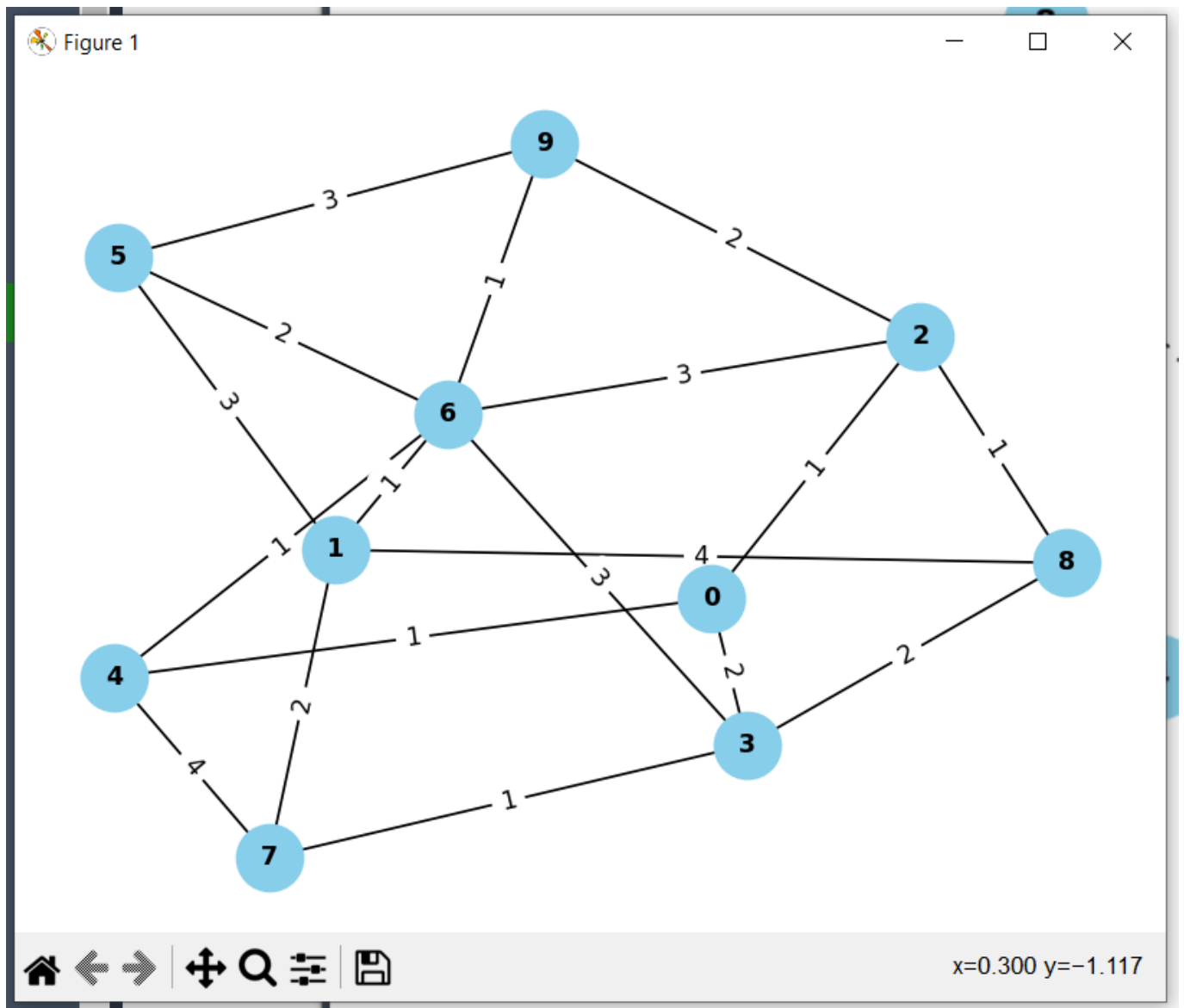
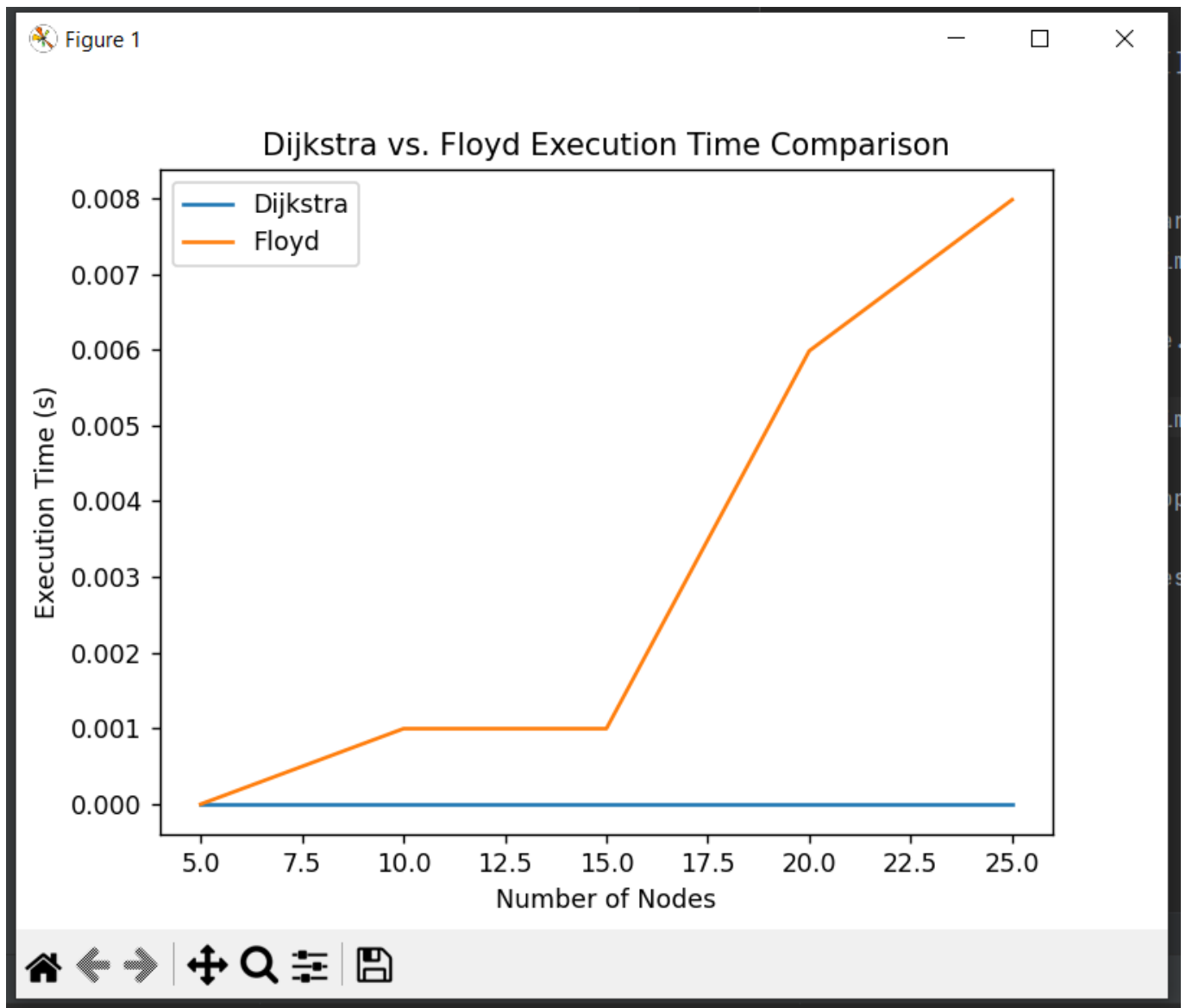


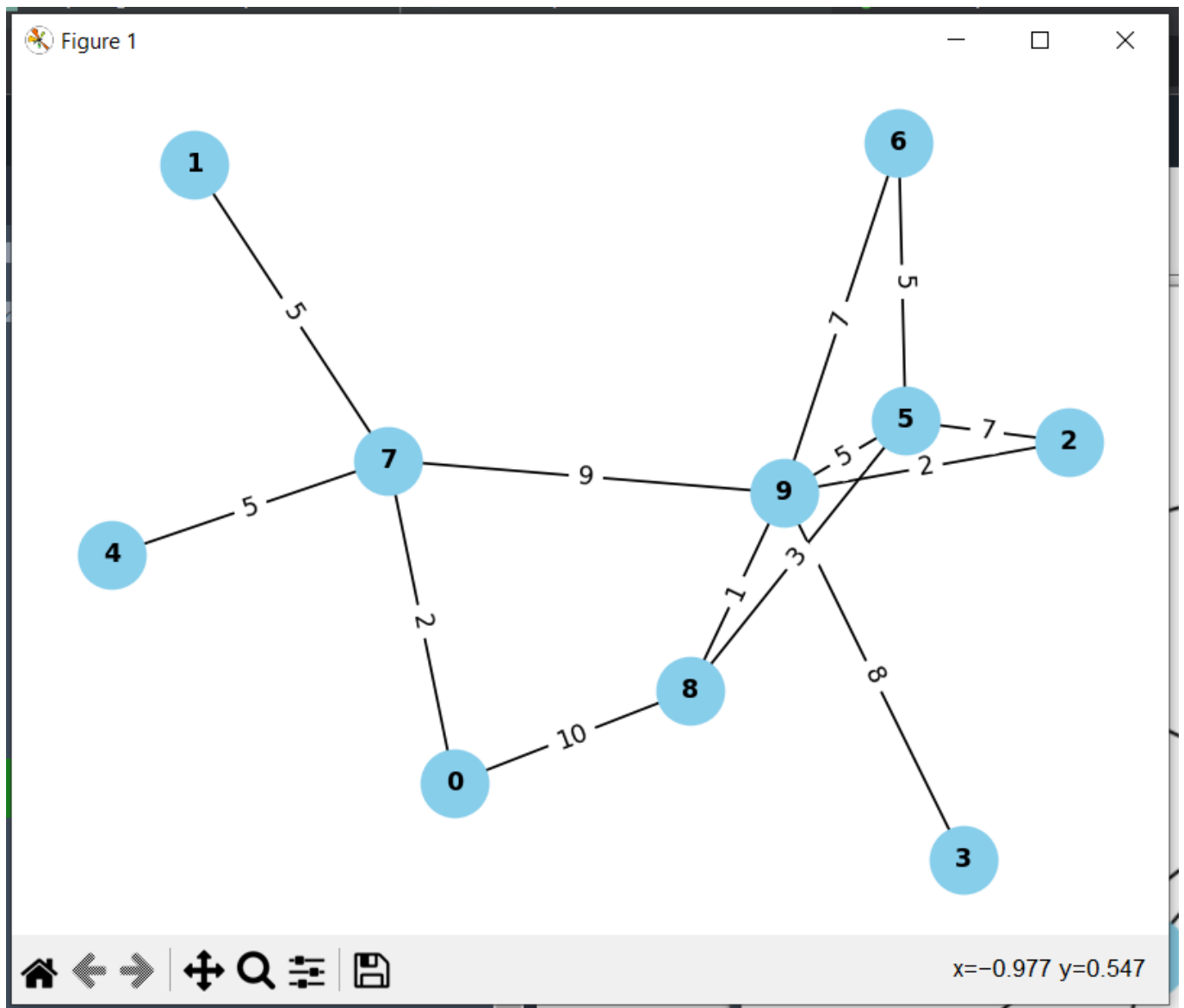
Figure 1



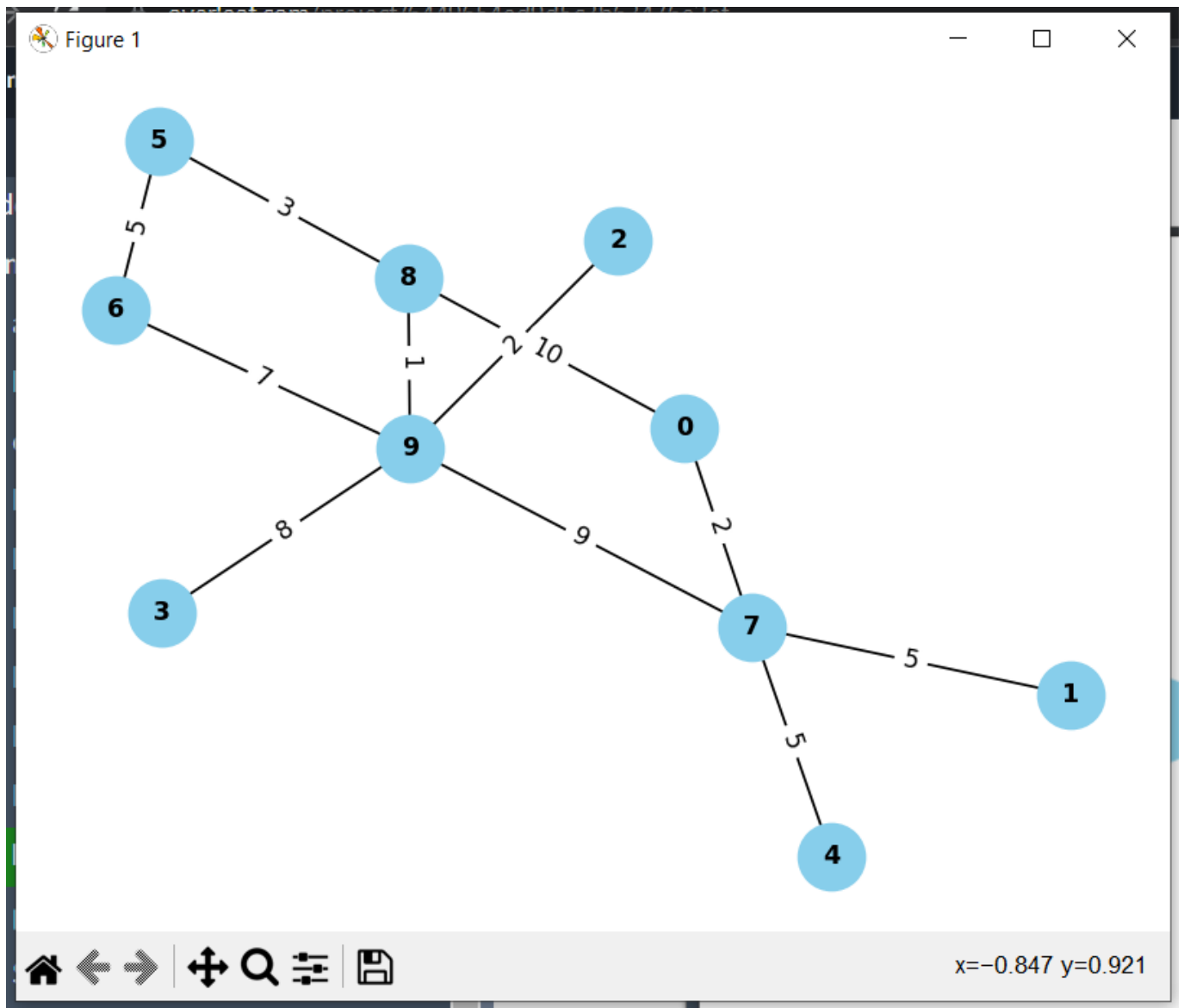












Conclusion

In this laboratory work, I thoroughly studied and analyzed Dijkstra's and Floyd's algorithms by implementing and comparing them on both dense and sparse graphs. These algorithms are fundamental for finding the shortest paths in a graph, but they differ in their methods and areas of efficiency. Dijkstra's algorithm operates by finding the shortest path from a start node to all other nodes in the graph. It systematically selects the node with the smallest tentative distance, then updates the distances to its adjacent nodes. This process continues until all nodes have been visited. Floyd's algorithm, alternatively, works on a different principle. It finds the shortest paths between all pairs of nodes in the graph, and it does so by gradually improving an estimate of the shortest distance between two nodes, considering each node as a possible step on the path. During my experimentations, I concluded that the time complexity of these algorithms is their most significant point of divergence. Dijkstra's algorithm, with a time complexity of $O(n^2)$ or $O((E+V)\log V)$ when using a binary heap, performs more efficiently on sparse graphs. Sparse graphs have fewer edges, meaning there are fewer neighbors to visit and distances to update. Floyd's algorithm, on the other hand, has a time complexity of $O(n^3)$, irrespective of whether the graph is sparse or dense, because it computes the shortest paths between all pairs of nodes. The laboratory work also involved the generation of visual representations of the graph and the resultant shortest path trees from both algorithms, which offered a deeper understanding of their operation.

To conclude, my laboratory work has shown that the choice between Dijkstra's and Floyd's algorithms is highly dependent on the specific needs of the problem and the nature of the graph in question. For a sparse graph or when you need the shortest path from a single source to all other nodes, Dijkstra's algorithm would be a more efficient choice. However, in scenarios where the shortest paths between all pairs of nodes are needed, especially in dense graphs, Floyd's algorithm would be more appropriate. This work has enhanced my understanding of these crucial graph algorithms and their respective applications.

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