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 Gazea Sandu
Verified:
asist.univ. Fiștic Cristofor

Chișinău, 2023

Content

Introduction

Dijkstra's algorithm and Floyd-Warshall algorithm are two popular methods used to find the shortest paths in graphs.

Dijkstra's algorithm is used when we want to find the shortest path from one specific starting point to all other points in a graph. It works by keeping track of the visited and unvisited points. Starting from the source point, it repeatedly selects the unvisited point with the shortest tentative distance and updates the distances of its neighboring points. Dijkstra's algorithm is especially useful for finding the shortest path in graphs where all edge weights are positive numbers. It has a time complexity that depends on the number of edges and vertices in the graph.

On the other hand, the Floyd-Warshall algorithm is used when we need to find the shortest paths between all pairs of points in a graph. It maintains a matrix that stores the shortest distances between every pair of points. By considering all possible intermediate points, the algorithm iteratively updates the matrix. Floyd-Warshall algorithm can handle graphs with both positive and negative edge weights. However, it has a higher time complexity compared to Dijkstra's algorithm, as it depends on the number of vertices cubed.

Both Dijkstra's and Floyd-Warshall algorithms are widely employed in various domains such as network routing, transportation planning, and computer graphics. The choice between them depends on the nature of the graph and the specific problem at hand.

In conclusion, Dijkstra's algorithm and Floyd-Warshall algorithm are essential tools for solving the shortest path problem in graphs. They find applications in numerous real-world scenarios like network routing, GPS navigation, and airline scheduling. Dijkstra's algorithm is suitable for graphs with positive edge weights, while Floyd-Warshall algorithm handles graphs with both positive and negative edge weights. The selection of the appropriate algorithm depends on the characteristics of the graph and the specific problem that needs to be solved.

Dijkstra

Dijkstra's Algorithm is a graph traversal algorithm that is commonly used to determine the shortest path between a starting node and all other nodes in a graph, assuming that the edge weights are non-negative. Initially, the starting node is assigned a distance of 0 and all other nodes are given a distance of infinity. The algorithm then proceeds by selecting the unvisited node with the smallest known distance at each step and updating the distances of its neighboring nodes. This is accomplished by comparing the sum of the current node's distance and the weight of the edge leading to the neighboring node to the current distance assigned to the neighbor. If the sum is less than the current distance, the algorithm updates the neighbor's distance with the smaller value. This process is repeated until all nodes have been visited, resulting in the shortest path from the starting node to each node in the graph being determined. Code:

```
 function Dijkstra(Graph, source):

for each vertex v in Graph.Vertices:
     dist[v] ← INFINITY
    prev[v] ← UNDEFINED
   add v to Q
dist[source] ← 0

while Q is not empty:
    u ← vertex in Q with min dist[u]
    remove u from Q

    for each neighbor v of u still in Q:
        alt ← dist[u] + Graph.Edges(u, v)
        if alt < dist[v]:
            dist[v] ← alt
            prev[v] ← u

return dist[], prev[]
```

Floyd-Warshall

The Floyd-Warshall Algorithm is a method to find the shortest paths between every pair of nodes in a directed, weighted graph. This algorithm works by systematically considering all combinations of nodes and updating a matrix of distances according to the shortest known path between each pair of nodes. Initially, a distance matrix is created, with the diagonal elements set to 0 and the off-diagonal elements set to the corresponding edge weights, or infinity if no direct edge exists between the node pair. The algorithm then repeatedly examines whether a shorter path between a pair of nodes can be found by going through an intermediate node. If a shorter path is discovered, the distance matrix is updated with the new, shorter distance value. This process continues until all possible intermediate nodes have been considered. At the end of the algorithm, the distance matrix will contain the shortest path distances between every pair of nodes in the graph.

Code:

```
      let dist be a |V| × |V| array of minimum distances initialized to infinity
for each vertex v
   dist[v][v] ← 0
for each edge (u,v)
   dist[u][v] ← w(u,v)  // the weight of the edge (u,v)
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]
```

Implementation

```
import numpy as np
import time
import networkx as nx
import matplotlib.pyplot as plt

class GraphAlgorithms:
    def __init__(self, graph):
        self.graph = graph
        self.n = len(graph)

    def dijkstra(self, start):
        dist = [float('inf')] * self.n
        visited = [False] * self.n
        dist[start] = 0
        for _ in range(self.n):
            u = min((x for x in range(self.n) if not visited[x]), key=dist.__getitem__)
            visited[u] = True
            for v in range(self.n):
                if self.graph[u][v] != 0 and not visited[v]:
                    alt = dist[u] + self.graph[u][v]
                    if alt < dist[v]:
                        dist[v] = alt
        return dist

    def floyd_warshall(self):
        dist = self.graph.copy()
        for k in range(self.n):
            for i in range(self.n):
                for j in range(self.n):
                    dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])
```

```
        return dist

    def draw_graph(self):
        G = nx.DiGraph()
        for i in range(self.n):
            for j in range(self.n):
                if self.graph[i][j] != 0:
                    G.add_edge(i, j, weight=self.graph[i][j])
        pos = nx.spring_layout(G)
        nx.draw(G, pos, with_labels=True)
        labels = nx.get_edge_attributes(G, 'weight')
        nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)
        plt.show()

def generate_graph(n, graph_type='sparse'):
    if graph_type == 'sparse':
        return np.random.choice([0, 1], size=(n, n), p=[0.9, 0.1])
    else:
        return np.random.randint(0, 10, size=(n, n))

def measure_algorithm_times(algorithm, graph_type):
    times = []
    for n in range(10, 101, 10):
        graph = generate_graph(n, graph_type)
        graph_algo = GraphAlgorithms(graph)
        start_time = time.time()
        if algorithm == 'dijkstra':
            graph_algo.dijkstra(0)
        else:
            getattr(graph_algo, algorithm)()
        end_time = time.time()
        times.append(end_time - start_time)
    return times
```

```
def plot_execution_times(times, algorithm_name, graph_type):
    plt.plot(range(10, 101, 10), times, label=f"{algorithm_name} ({graph_type.capitalize()
    plt.legend()
    plt.xlabel("Number of Nodes")
    plt.ylabel("Execution Time (Seconds)")
    plt.title(f"Execution Time of {algorithm_name} on {graph_type.capitalize()} Graph")
    plt.show()

if __name__ == "__main__":
    algorithms = ['dijkstra', 'floyd_warshall']
    graph_types = ['sparse', 'dense']

    for graph_type in graph_types:
        plt.figure(figsize=(10, 6))
        for algorithm in algorithms:
            times = measure_algorithm_times(algorithm, graph_type)
            plt.plot(range(10, 101, 10), times, label=f"{algorithm.replace('_', ' ').title

        plt.legend()
        plt.xlabel("Number of Nodes")
        plt.ylabel("Execution Time (Seconds)")
        plt.title(f"Execution Time of Shortest Path Algorithms on {graph_type.capitalize()
        plt.show()

    # Draw sample graphs
    for graph_type in graph_types:
        graph = generate_graph(10, graph_type)
        GraphAlgorithms(graph).draw_graph()
```

Screenshot:

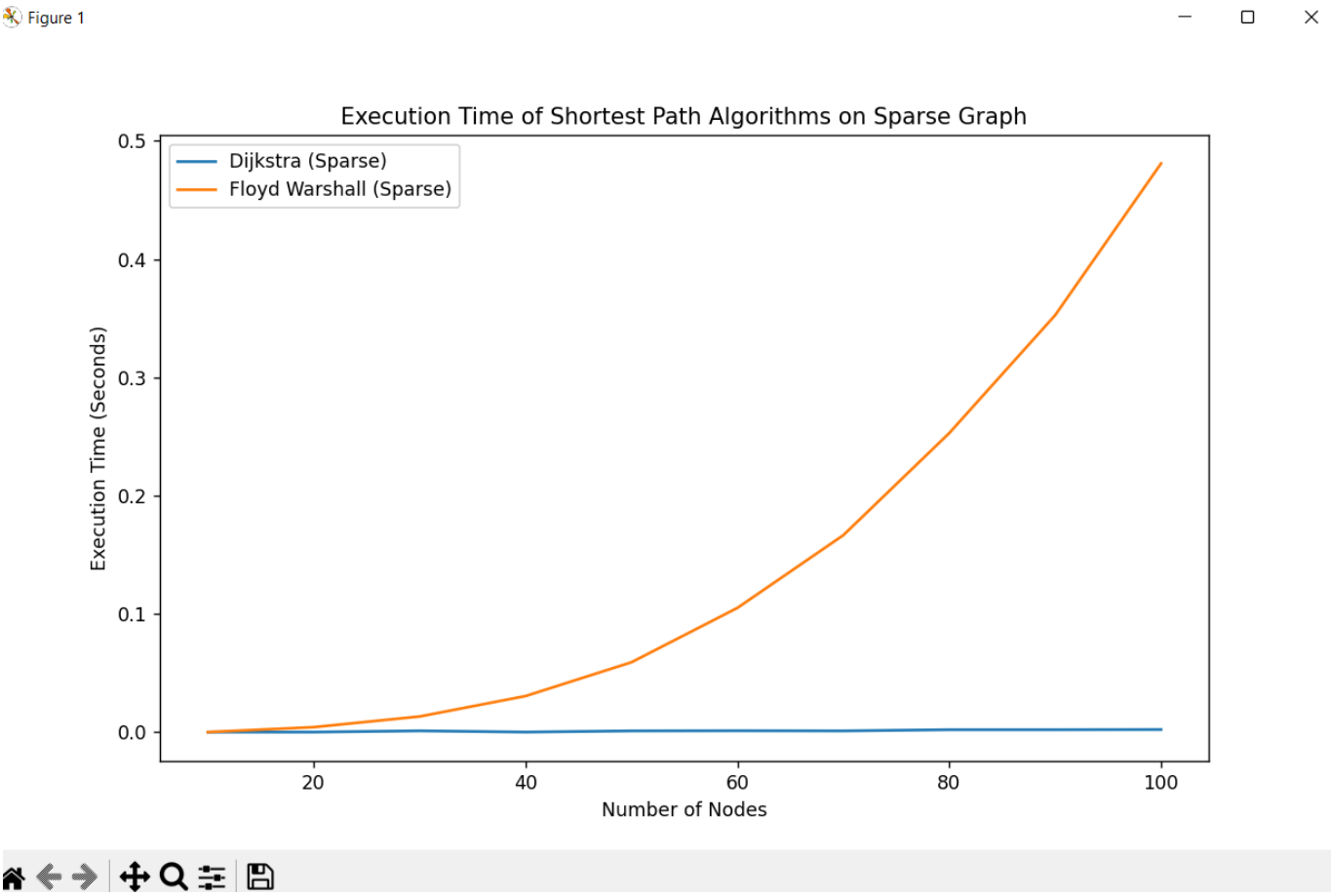
Figure 1
Execution Time of Shortest Path Algorithms on Sparse Graph
Dijkstra (Sparse)
Floyd Warshall (Sparse)
Execution Time (Seconds)
0.0
0.1
0.2
0.3
0.4
0.5
20
40
60
80
100
Number of Nodes

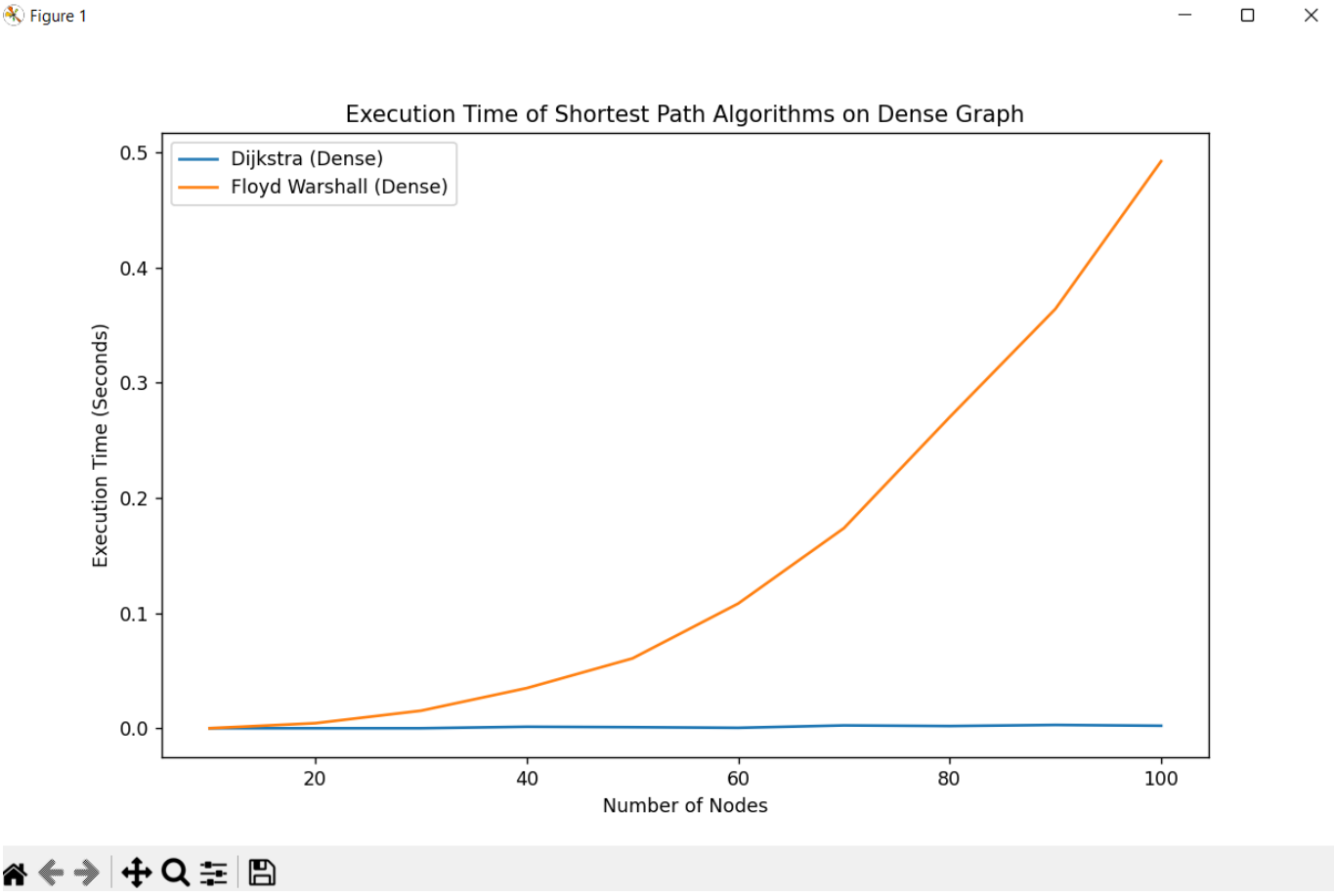
Figure 1
Execution Time of Shortest Path Algorithms on Dense Graph
Dijkstra (Dense)
Floyd Warshall (Dense)
Execution Time (Seconds)
0.0
0.1
0.2
0.3
0.4
0.5
20
40
60
80
100
Number of Nodes

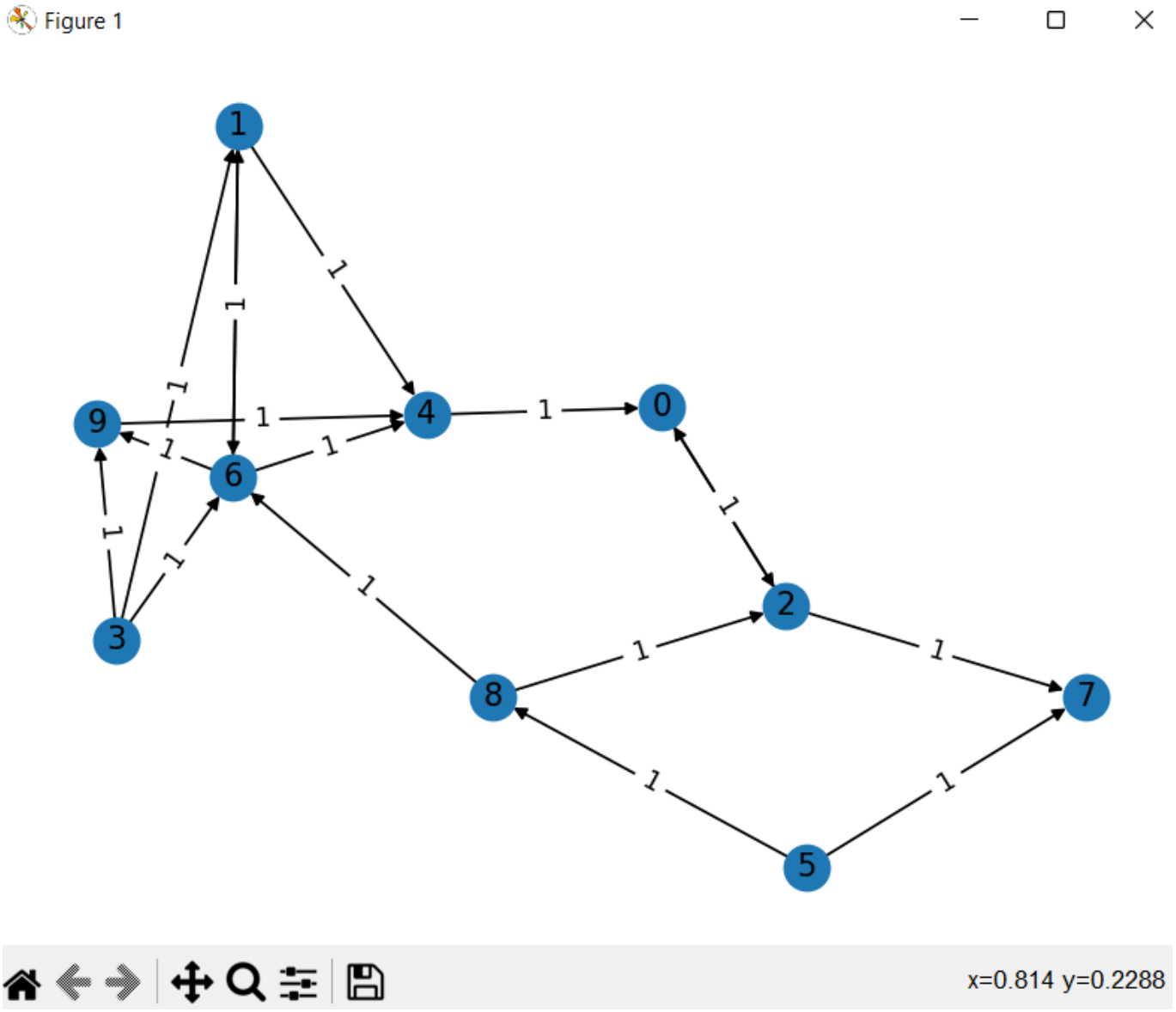
Figure 1
1
9
4
0
6
3
2
8
7
5
x=0.814 y=0.2288

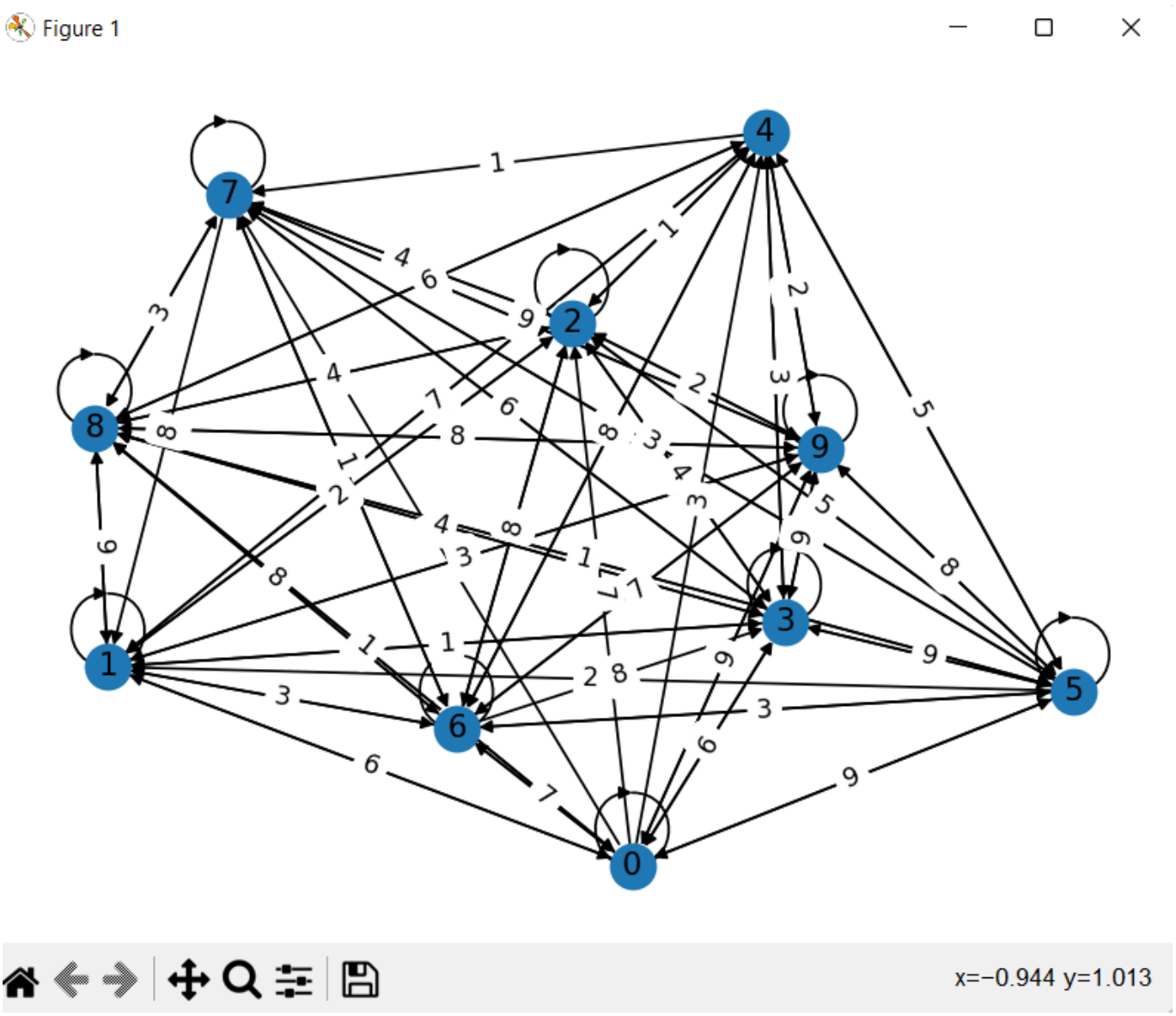
Figure 1
4
7
2
8
9
3
1
6
5
0
x=−0.944 y=1.013

Conclusion

To sum up, Dijkstra's algorithm and Floyd-Warshall algorithm are two well-known and commonly used algorithms in graph theory.

Dijkstra's algorithm is a method for finding the shortest path from a single source vertex to all other vertices in a weighted graph. It works by selecting the vertex with the smallest distance and updating the distances of its neighboring vertices. It is effective in sparse graphs with non-negative edge weights but not suitable for graphs with negative edge weights or dense graphs due to its time complexity.

In contrast, the Floyd-Warshall algorithm is a dynamic programming algorithm that computes the shortest paths between all pairs of vertices in a weighted graph. It considers all possible intermediate vertices and updates the shortest path distances accordingly. It is more efficient than using Dijkstra's algorithm for every vertex pair in dense graphs with non-negative edge weights and can handle negative edge weights.

Both algorithms have strengths and weaknesses, and the choice of algorithm depends on the specific problem and graph characteristics. It is essential to note that neither algorithm is designed to handle negative cycles, and the Bellman-Ford algorithm is more appropriate for graphs with negative edge weights.

In conclusion, Dijkstra's algorithm and Floyd-Warshall algorithm are powerful tools for solving shortest path problems and have significantly contributed to the advancement of graph theory and its applications. They will continue to play an essential role in the future of graph theory and related fields.

$https://github.com/GSandu1/LabsAA.git$