Shortest Path with Dynamic Programming

Optimisation
Project Report

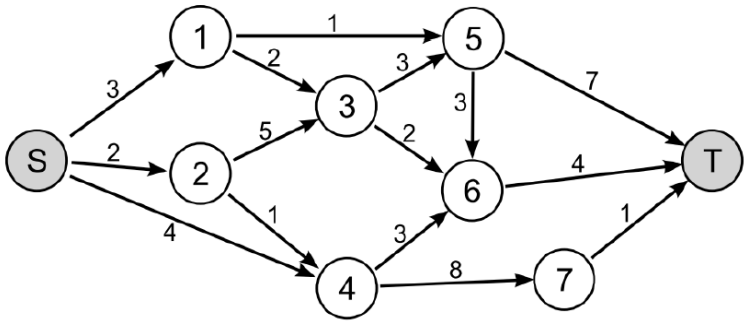
Serkan Burak Örs*

*orss19@itu.edu.tr

Contents

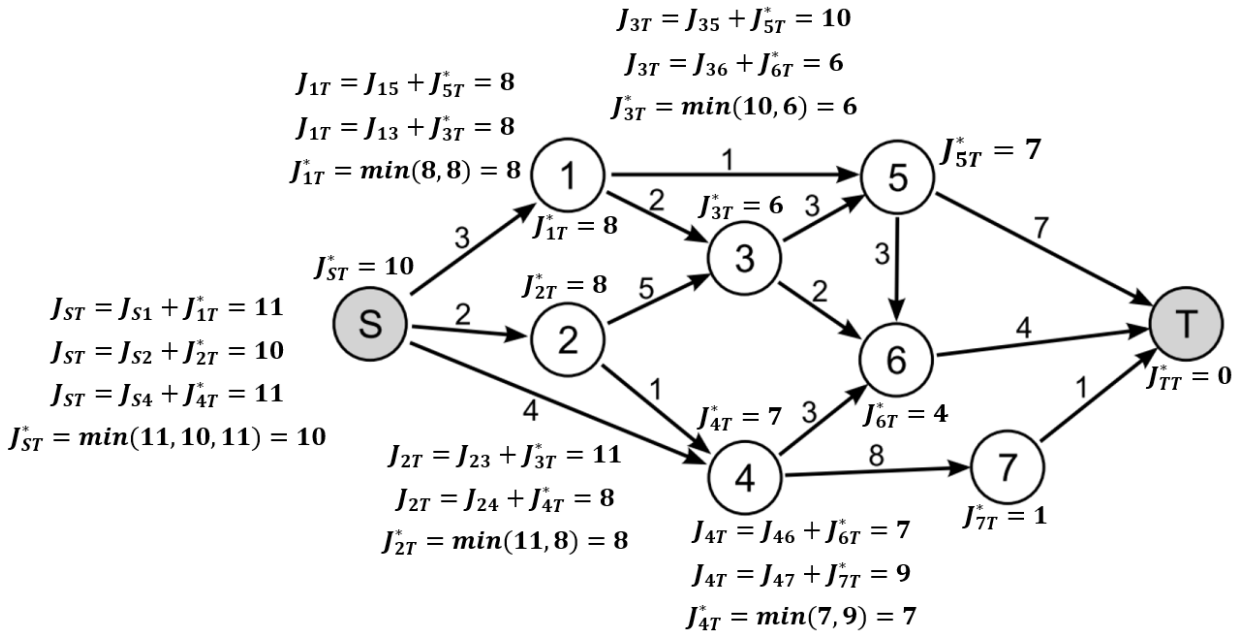
1 Introduction

In tihs project, the shortest path and its cost of the shortest path problem shown in figure below is determined using dynamic programming and checked by Djikstra's algorithm which also uses dynamic programming.



2 Solution

We can apply dynamic programming algorithm as in the figure below:



From figure above, we can determined that total minimum cost as

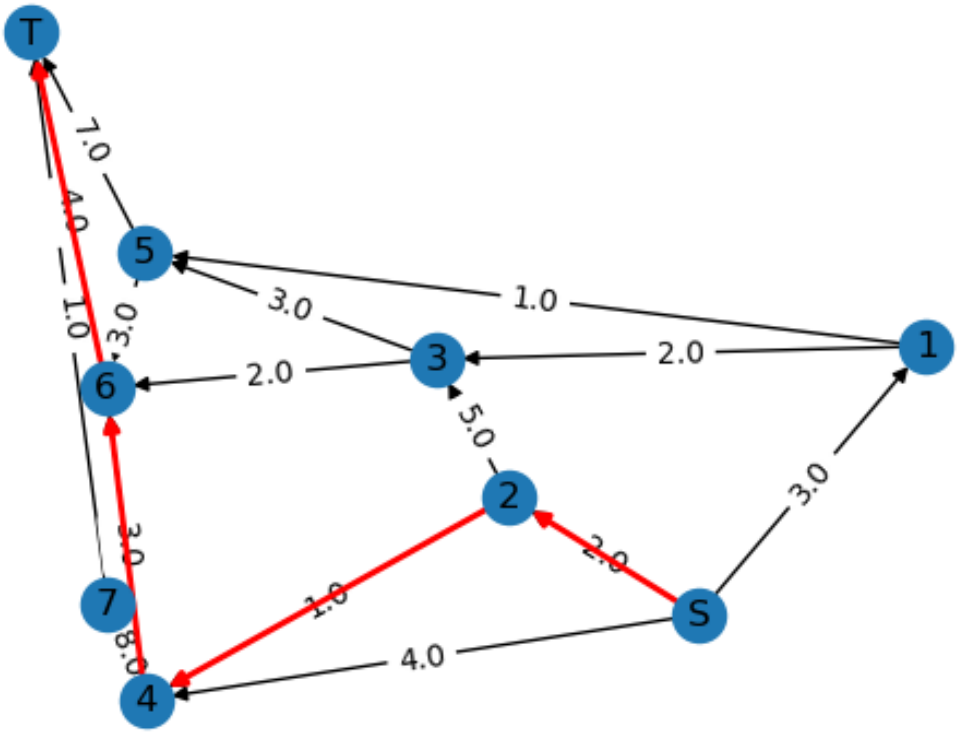
$$J^*_{ST} = J_{S2} + J^*_{2T} = J_{S2} + J_{24} + J^*_{4T} = J_{S2} + J_{24} + J_{46} + J^*_{6T} = 2 + 1 + 3 + 4 = 10 \quad (2.1)$$

and the optimum path is $S - 2 - 4 - 6 - T$

This problem also can be solved with computer algorithms. There are many algorithms using dynamic programming principles for finding shortest paths in weighted graphs as:

- Single-source shortest path on a weighted DAG
- Single-source shortest path on a weighted graph with nonnegative weights (Dijkstra's algorithm)
- Single-source shortest path on a weighted graph including negative weights (Bellman-Ford algorithm)

Our problem does not have negative weight. Therefore and because of simplicity, Djikstra's algorithm is used for solution of the problem. Weights which have infinite value denoted with Inf. It means it is impossible to transit with using Inf paths. Result is in figure below.



We can satisfy the result from computer code as total cost for optimum path is 10 and the optimum path is S - 2 - 4 - 6 - T

3 Appendix A: Python Code

```
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
# Define the cost matrix
costMatrix = np.array([
[np.inf, 3, 2, np.inf, 4, np.inf, np.inf, np.inf, np.inf], ...
                    #S
[np.inf, np.inf, np.inf, 2, np.inf, 1, np.inf, np.inf, np.inf], ...
                #1
[np.inf, np.inf, np.inf, 5, 1, np.inf, np.inf, np.inf, np.inf], ...
                #2
[np.inf, np.inf, np.inf, np.inf, np.inf, 3, 2, np.inf, np.inf], ...
                #3
[np.inf, np.inf, np.inf, np.inf, np.inf, np.inf, 3, 8, np.inf], ...
                #4
[np.inf, np.inf, np.inf, np.inf, np.inf, np.inf, 3, np.inf, 7], ...
                #5
[np.inf, np.inf, np.inf, np.inf, np.inf, np.inf, np.inf, np.inf, 4], ...
          #6
[np.inf, np.inf, np.inf, np.inf, np.inf, np.inf, np.inf, np.inf, 1], ...
          #7
[np.inf, np.inf, np.inf, np.inf, np.inf, np.inf, np.inf, ...
    np.inf,np.inf]]) #T
# Define the index of the initial and final nodes
indexInitial = 0
indexFinal = 8
# Initialize variables
n = costMatrix.shape[0]
S = np.zeros(n, dtype=bool)
dist = np.full(n, np.inf)
prev = np.full(n, n+1)
dist[indexInitial] = 0
# Dijkstra's algorithm
while not np.all(S):
candidate = np.where(¬S, dist, np.inf)
u = np.argmin(candidate)
S[u] = True
for v in range(n):
if not S[v] and dist[u]+costMatrix[u,v] < dist[v]:
```

```
dist[v] = dist[u] + costMatrix[u,v]
prev[v] = u
# Find the shortest path and its cost
path = [indexFinal]
while path[0] != indexInitial:
if prev[path[0]] ≤ n:
path.insert(0, prev[path[0]])
else:
raise Exception('No path exists')
totalCost = dist[indexFinal]
# Create a graph of the network
G = nx.DiGraph()
for i in range(n):
for j in range(n):
if costMatrix[i,j] < np.inf:
G.add_edge(i, j, weight=costMatrix[i,j])
# Create a dictionary of node labels
node_labels = {i: f'{i+1}' for i in range(n)}
node_labels[indexInitial] = 'S'
node_labels[indexFinal] = 'T'
# Create a list of edge labels
edge_labels = {(i,j): f'{costMatrix[i,j]}' for (i,j) in G.edges}
shortest_path = nx.shortest_path(G, source=indexInitial, ...
    target=indexFinal, weight='weight')
# Draw the graph with edge labels and node labels
pos = nx.spring_layout(G)
nx.draw_networkx_nodes(G, pos)
nx.draw_networkx_edges(G, pos)
labels = {0: 'S', 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7, 8:'T'}
nx.draw_networkx_labels(G, pos, labels=labels)
nx.draw_networkx_edge_labels(G, pos, ...
    edge_labels=nx.get_edge_attributes(G, 'weight'))
nx.draw_networkx_edges(G, pos, edgelist=[(shortest_path[i], ...
    shortest_path[i+1]) for i in range(len(shortest_path)-1)], ...
    edge_color='r', width=2)
plt.axis('off')
plt.show()
#Print the results
print('Optimal path is:S -', ...
    shortest_path[1],'-',shortest_path[2],'-',shortest_path[3],'-','T')
print('Optimal cost is', nx.shortest_path_length(G, ...
    source=indexInitial, target=indexFinal, weight='weight'))
```

4 References

[1] Kirk D.E. *Optimal Control Theory*. Dover Publications. 1998.

[2] Chong, Edwin K. P., and Stanislaw H. Zak. *Introduction to Optimization*. John Wiley & Sons. 2013.

[3] Wayne K., *Princeton University Algorithm Design Lecture Notes 18*.