



UNIVERSITY
of Prince Edward
ISLAND

CAIRO Campus

Combinatorial Optimization

*Finding Graph Shortest Path using Dijkstra's
Algorithm*

Authors' Name	Student ID
Islam Anwar	201900337

FACULTY OF SCIENCE & INNOVATION,
THE UNIVERSITIES OF CANADA

March 25, 2023

Abstract

The shortest path problem has been a problem that has been used in various applications to minimize the cost of transportation, or the path of least cost in a graph, as well as other business applications. Among the most popular shortest path algorithms is Dijkstra's algorithm, which aims at finding the shortest path of a problem by assigning a source node and exploring all other nodes in the network. This iterates until all nodes have been explored and each node has its respective minimum cost computed while also considering accumulated costs from previously visited nodes. This paper aims to perform Dijkstra's algorithm on a custom-made graph dataset. Results indicate that our minimum cost from a source node to a destination node using Excel Solver was indeed the minimum path. The algorithm functions as promised by interpreting edges in the network as either part of the shortest path, intermediate edges in the shortest path, or not part of the shortest path. However, there is still further improvement to be made to compare other algorithms in terms of complexity from the time and space perspective.

Shortest Path — Dijkstra's — Excel Solver

Contents

1	Introduction	1
2	Research Problem	1
3	Literature Review	1
4	Data Collection Process	2
5	Problem Formulation	2
6	Problem Solving	4
6.1	Constructing Distance Matrix	4
6.2	Building Solution Space & Constraints	4
6.3	Solutions	5
7	Analysis & Discussion	7
8	Recommendations	7
9	Appendix	8

1 Introduction

Combinatorial optimization is the process of using combinatorial solutions to maximize or minimize certain real-world applications. Real-world problems in this day and age are becoming more complex and difficult to solve manually. With the introduction of several algorithms and heuristic techniques brought to light, many problems such as finding the shortest path, maximizing flow in a system, and knapsack problems have become much easier to solve with a sophisticated methodology.

The problem of interest is the shortest path problem. Given a weighted graph with a collection of nodes and weighted edges, we will attempt to minimize the shortest path from a single node to a destination node using the Dijkstra shortest path algorithm. This method of Dijkstra was chosen in particular because of the interest in how it assesses the neighboring nodes of the node it is currently at without proceeding forward. It is a very sophisticated technology and is used in many applications, including social network graphs, minimizing data packet routes to reduce delay in a computer communication context, and many other backgrounds.

This paper covers Dijkstra's shortest path algorithm on a weighted graph set of data created by hand. The rest of the paper structure is as follows: Section 2 covers the relevant literature review regarding shortest path algorithms and variations of Dijkstra's algorithm. Section 3 entails the data collection process. Section 4 portrays the problem formulation aspect of the problem. Section 5 illustrates the problem-solving process and how we will reach a conclusive output. Section 6 provides an analysis of the results as well as further discussion. Finally, section 7 discusses recommendations for future work.

2 Research Problem

Our research problem is to compute the shortest path from a source node to a destination node using Dijkstra's algorithm. The application of the shortest path problem is used in a variety of real-world applications. Our simple experiment is a generalized application, which is just computing the shortest path for any weighted bidirectional graph. This can be related to real-world applications such as routing problems in networks, cost of transportation for vehicles to minimise the distance of product transportation and more.

Our objective function is to minimize the sum product of the decision variables of each edge and their respective weights. Regarding our constraints, we must ensure all edge weights are greater than or equal to 0 as Dijkstra's only works with non-negative weights. We also must ensure that our decision variables for each edge is either a value of 0 or 1 to indicate if this edge is part of the shortest path or not.

3 Literature Review

In recent years, extensive research has been performed regarding Dijkstra's algorithm, particularly in solving shortest-path problems in various applications. Among them was

D. Rachmawati et al. [1], who were concerned with finding the shortest path from one source location to a destination of a mapped region. They employed both Dijkstra's algorithm and the A* algorithm and found that their performances were almost the same when used on regional scale maps. However, on larger-scale maps, A* outperformed Dijkstra's. Their methodology for Dijkstra's algorithm was to set all point distances to infinity except the starting point (set to 0). Furthermore, they set the starting node as unvisited and assessed the neighboring nodes by accumulating minimum distances over iterations or unexplored nodes. This step would be repeated until the destination node would be reached or every node in the network was explored [1].

Another interesting study was performed by M. Enayattabar et al. [2], who optimized the shortest path problem using Dijkstra's algorithm on an interval-valued Pythagorean fuzzy environment. This means that the related edge costs are taken in form of interval-valued Pythagorean fuzzy numbers. These are effective optimality conditions set for directed graphs to design a more effective solution. Upon this, Dijkstra's is employed to take advantage of these improvements and results indicated that there was a substantial performance improvement when using extended Dijkstra's on an enhanced score function and optimality conditions via the Pythagorean fuzzy numbers as edge weights [2].

Finally, R. Gunawan et al. [3] implemented Dijkstra's algorithm for the shortest path for searching for medical specialists in the nearby region. In this particular case study, Dijkstra's is used to not only find the shortest path but is used as a search algorithm amongst a graph determining nearby medical specialists in the community. Based on the results, Dijkstra's performed as intended by finding the shortest path based on minimum weight from node to node until all nodes have been explored [3].

4 Data Collection Process

Regarding the data collection process, we created our own dataset for performing the shortest path algorithm. Fig. (1) provides an overview of the graph topology. Each node in the graph is assigned bidirectional weighted links to their respective neighbors. Some nodes' edge weights with their neighbors may be sub-optimal in terms of minimum distance. This was intended to test the effectiveness of Dijkstra's algorithm for computing the shortest path in the graph from any source node to any destination node.

5 Problem Formulation

Concerning the problem formulation, our objective is to use Dijkstra's algorithm to find the shortest path between vertices of a given graph. Referring to the graph topology, we are assigning node 'A' to be the source node and node 'O' as the destination node.

The formal problem formulation is illustrated in equation (1). We aim to minimize the sum product of the weights times the decision variable from node i to node j . D is the distance from node i to node j . X is the decision variable, where if it is equal to 1

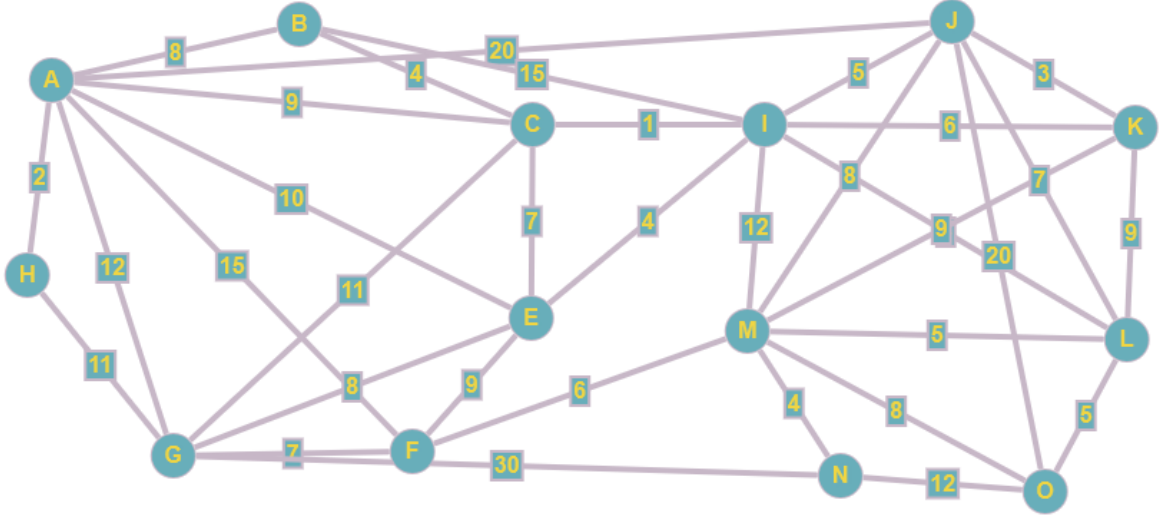


Figure 1: Graph Topology

then the arc between nodes i to node j is part of the shortest path, else it is set to 0 and is not part of the shortest path solution as shown in equations (2) and (3).

$$\text{Min} \sum D_{ij} X_{ij} \quad (1)$$

$$D_{ij} = \text{distance from node } i \text{ to } j \geq 0 \quad (2)$$

$$X_{ij} \text{ are the decision variables} = \begin{cases} 1 & \text{Arc (i,j) is part of the shortest path} \\ 0 & \text{Arc (i,j) is not part of the shortest path} \end{cases} \quad (3)$$

Along with the problem formulation mentioned above, there are a set of problem constraints that we had to define to adhere to Dijkstra's algorithm. Among those constraints is that all edge weights are positive per the assumption made by Dijkstra's algorithm. Furthermore, equation (4) shows the constraint that the sum of differences of weights be either a value of 1, 0, or -1 to indicate if a particular node is the source node, sink node, or an intermediate node corresponding to the shortest path. equation (5) illustrates that our decision variables are all binary given that they all belong to A, which is a collection of all edges in the graph.

$$\sum_j X_{ij} - \sum_j X_{ji} = \begin{cases} 1 & : \text{if } i \text{ is the source node} \\ 0 & : \text{if } i \text{ is an intermediate node} \\ -1 & : \text{if } i \text{ is the sink node} \end{cases} \quad (4)$$

$$X_{ij} \in [0, 1], \forall (i, j) \in A \quad (5)$$

6 Problem Solving

We will solve the shortest path problem using Dijkstra's algorithm. Dijkstra's algorithm is illustrated in fig. (2). The algorithm steps are as follows:

1. Iterate through all vertices in graph
2. Set their distances to infinity and previous distance as an undefined value
3. Add that visited vector to a set Q
4. While set Q is not empty then we explore adjacent vectors with minimum distance
5. Iterating through each neighbor that hasn't been explored, we update their distances based on accumulated distances from source node
6. We repeat this until all nodes in graph are in set Q and have been explored
7. Return the dist array and prev array of visited nodes in a certain order

```
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[]
```

Figure 2:

6.1 Constructing Distance Matrix

Our first step in solving our problem using Dijkstra's algorithm was to construct our distance matrix of the graph. This entails representing the weights of all edges within the graph from node to node. Fig. (3) illustrates the weights between each pair of nodes. There are naturally some pairs of nodes that do not contain an edge between them, therefore instead of inputting 0 in those cells, we inputted a very large value such as 100.

6.2 Building Solution Space & Constraints

Our next step in solving the problem of the shortest path was to construct the solution space which included a cell for each pair of nodes that would house a binary value of either 0 or 1 based on the decision variables that we mentioned previously.

Distance Matrix															
	A	B	C	E	F	G	H	I	J	K	L	M	N	O	
A		100	8	9	10	15	12	2	100	20	100	100	100	100	100
B		8	100	4	100	100	100	100	15	100	100	100	100	100	100
C		9	4	100	7	100	11	100	1	100	100	100	100	100	100
E		10	100	7	100	9	8	100	4	100	100	100	100	100	100
F		15	100	100	9	100	7	100	100	100	100	100	6	100	100
G		12	100	11	8	7	100	11	100	100	100	100	100	30	100
H		2	100	100	100	100	11	100	100	100	100	100	100	100	100
I		0	15	1	4	100	100	100	100	5	6	5	12	100	100
J		20	100	100	100	100	100	100	5	100	3	7	8	100	20
K		100	100	100	100	100	100	100	6	3	100	9	9	100	100
L		100	100	100	100	100	100	100	5	7	9	100	5	100	5
M		100	100	100	100	100	6	100	100	12	8	9	5	100	4
N		100	100	100	100	100	30	100	100	100	100	100	4	100	12
O		100	100	100	100	100	100	100	100	20	100	5	8	12	100

Figure 3: Graph Distance Matrix

Solution Space																		
	A	B	C	E	F	G	H	I	J	K	L	M	N	O	Total Out	Out-In	Relation	RHS
A															0	0	=	1
B															0	0	=	0
C															0	0	=	0
E															0	0	=	0
F															0	0	=	0
G															0	0	=	0
H															0	0	=	0
I															0	0	=	0
J															0	0	=	0
K															0	0	=	0
L															0	0	=	0
M															0	0	=	0
N															0	0	=	0
O															0	0	=	-1
Total-In	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			

Figure 4: Graph Solution Space

For our constraints, fig. (4) is a collection of columns including total-in, total-out, and out-in. Out-in represents the difference of sums of decision variables as illustrated in equations (6-8). Its purpose was to provide us with an objective value of whether a node was a part of the shortest path or not. After constraint construction, we added 1 to the right-hand side of the row corresponding to node A and -1 to the right-hand side of the row corresponding to node O to show that they are the source and destination nodes as mentioned earlier.

$$TotalOut = \sum X_{ij} \text{ per node} \quad (6)$$

$$TotalIn = \sum X_{ji} \text{ per node} \quad (7)$$

$$OutIn = TotalOut - TotalIn \quad (8)$$

6.3 Solutions

We have identified our objective function to minimize the sum product of our edge weights and decision variables. We have also identified our constraints for a shortest path problem.

Regarding the solutions, we first opted for the GRG Nonlinear methodology to solve our problem. Surprisingly, it appeared that GRG Nonlinear would be cycling and not converge to a globally optimal solution as shown in fig. (7) and is essentially constantly looping at approximately 400 subproblems and rising. With respect to the evolutionary solving method, we also could not reach convergence and solver could not find a feasible solution as shown in fig. (8).

However, when using the simplex linear programming we were able to find a solution. This could be due to the linearity of our objective function, which ensured that the simplex method could produce a viable result. Fig. (5) shows the setup we established for the simplex method to reach a solution.

The screenshot shows the Excel Solver dialog box with the following configuration:

- Set Objective:** \$B\$50
- To:** ☐ Max ☒ Min ☐ Value Of: 0
- By Changing Variable Cells:** \$B\$33:\$Q\$46
- Subject to the Constraints:**
 - \$B\$33:\$Q\$46 = binary
 - \$Q\$33:\$Q\$46 = \$S\$33:\$S\$46
- ☒ **Make Unconstrained Variables Non-Negative**
- Select a Solving Method:** Simplex LP
- Buttons:** Add, Change, Delete, Reset All, Load/Save, Options
- Solving Method:** Select the GRG Nonlinear engine for Solver Problems that are smooth nonlinear. Select the LP Simplex engine for linear Solver Problems, and select the Evolutionary engine for Solver problems that are non-smooth.

Figure 5: Simplex LP

7 Analysis & Discussion

Regarding the analysis of results, we can see from fig. (6) that our solution space illustrates values of 0 and 1. A value of 1 indicates that a certain edge is part of the shortest path. Furthermore, using a solver we concluded the shortest path of 20 units from node A to node O. The shortest path traversed is also shown in fig. (6). How these results were computed is dependent on how Dijkstra's approaches a problem as such to find the shortest path given a source node.

Solution Space																			
	A	B	C	E	F	G	H	I	J	K	L	M	N	O	Total	Out	Out-In	Relation	RHS
A	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	=		1
B	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=		0
C	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	=		0
E	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=		0
F	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=		0
G	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=		0
H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=		0
I	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	=		0
J	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=		0
K	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=		0
L	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	=		0
M	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=		0
N	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=		0
O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=		-1
Total-In	0	0	1	0	0	0	0	0	1	0	0	1	0	0	1				
Min Dist. =	20																		
Shortest Path	A -> C -> I -> L -> O																		

Figure 6: Solution Output

8 Recommendations

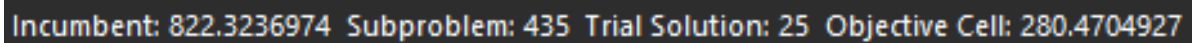
Regarding areas of improvement in our project, perhaps using different variations of Dijkstra's could be an effective way to compute the shortest path of our graph. A variation of Dijkstra's is the greedy best-first search, which uses a heuristic function that handles the edge weights differently as opposed to the regular Dijkstra's algorithm.

Another variation of Dijkstra's is A* algorithm. A* has 3 parameters: the cost of moving from one node to another node or the sum of the weights from one node to another, the heuristic value which is the cost of moving from the cell that the algorithm is currently at and the destination node, and finally the sum weights of both previously mentioned parameters. The sum weights are updated at every step of the algorithm, however, some nodes may not need to be visited during the algorithm unlike Dijkstra's [4].

Johnson's algorithm uses the Bellman-Ford algorithm to deal with negative weights and transforms the graph into a graph that can be processed later by Dijkstra's algorithm. Johnson functions in a slightly different manner from Dijkstra's. It works by adding a vertex to the initial graph with edges from that vertex to all other vertices. Moreover, it runs the Bellman-Ford algorithm as an underlying algorithm with the new node as the source node. Then it updates the new weights of edges in the new graph and removes the added vertex. Then after removing the vertex, Dijkstra's is run for every vertex to output the shortest path [5].

Variations of the algorithm offer lighter computational costs and sometimes yield better performance. Certain areas of improvement for our project are perhaps try different algorithms to compare the performance of each algorithm from a time and space complexity perspective. As Dijkstra's time complexity is $O(N^2)$, other variations could potentially have better time complexities. That is an important factor worth considering for future experiments.

9 Appendix



Incumbent: 822.3236974 Subproblem: 435 Trial Solution: 25 Objective Cell: 280.4704927

Figure 7: GRG Nonlinear Cycle



Incumbent: 20 Subproblem: 2376 Trial Solution: 0 Objective Cell: 20

Figure 8: Evolutionary Method Cycle

References

- [1] D. Rachmawati and L. Gustin, “Analysis of dijkstra’s algorithm and a* algorithm in shortest path problem,” in *Journal of Physics: Conference Series*, vol. 1566, p. 012061, IOP Publishing, 2020.
- [2] M. Enayattabar, A. Ebrahimnejad, and H. Motameni, “Dijkstra algorithm for shortest path problem under interval-valued pythagorean fuzzy environment,” *Complex & Intelligent Systems*, vol. 5, no. 2, pp. 93–100, 2019.
- [3] R. D. Gunawan, R. Napianto, R. I. Borman, and I. Hanifah, “Implementation of dijkstra’s algorithm in determining the shortest path (case study: Specialist doctor search in bandar lampung),” *Int. J. Inf. Syst. Comput. Sci*, vol. 3, no. 3, pp. 98–106, 2019.
- [4] “What is the a* algorithm?.”
- [5] “Johnson’s algorithm for all-pairs shortest paths,” Mar 2023.