K-Shortest Path Problem

Lucas Geurtjens | s5132841 | 25/05/2019

Table of Contents

[Problem Overview 2](#_Toc9784444)

[Algorithm Design 2](#_Toc9784445)

[Algorithm Description 2](#_Toc9784446)

[Dijkstra’s Algorithm 2](#_Toc9784447)

[Alternate K – 1 Paths Algorithm 2](#_Toc9784448)

[Innovation 2](#_Toc9784449)

[Algorithm Pseudo-Code 3](#_Toc9784450)

[Results and Algorithm Analysis 3](#_Toc9784451)

[Testcase Results 3](#_Toc9784452)

[Time Complexity 3](#_Toc9784453)

[References 3](#_Toc9784454)

[Appendix 3](#_Toc9784455)

# Problem Overview

The developed algorithm aims to solve the k-shortest path problem. The k-shortest path problem involves finding the k paths shortest paths to travel from a chosen starting node to a goal node within a directed graph. In the developed algorithm, the list of k shortest paths included 1 optimal route found using Dijkstra’s shortest path algorithm and k – 1 approximate routes, found using an altered version of Yen’s algorithm[2]. From this, for a given input set of connected vertices and their edge weights, the algorithm is able to return k number of shortest distance paths.

# Algorithm Design

## Algorithm Description

### Dijkstra’s Algorithm

Firstly, the program utilised an implementation of Dijkstra’s algorithm[1] to find the optimally shortest distance from the start to goal nodes. Here, the assumption of non-negative edge weights was made, to avoid issues with the use of this model. The algorithm attempts to loop over nodes of the graph until it has reached its goal node. For each node that is inspected, the distance cost to get to that node is tracked. From this, we look at the nodes connected edges. Here, the node with the lowest distance cost is chosen, and relaxation is applied if the distance is superseded. After this, the node with the minimum distance out of all unexplored nodes is chosen to be used as the next inspected node. The cycle is repeated until the shortest path to the goal node is found. From this, the algorithm calculates the path that was taken to find the goal node by tracing backwards from each edge pair. After this, the algorithm is then able to return both the optimal cost and path to the goal node.

### Alternate K – 1 Paths Algorithm

In addition, the program utilized an algorithm, as inspired by Yen’s algorithm[2], for calculating the k-1 approximate shortest paths. Here, the algorithm employs the optimal shortest path as previously calculated by Dijkstra’s algorithm. From this, the algorithm iterates though each of the optimal node edges found in the path and, one-by one towards the goal, removes them by setting their edge cost to a value of infinity (such that it won’t be selected). This is calculated k amount of times and is sorted in ascending order when complete. If there is overflow and not enough optimal edges exist, the algorithm pads the results with the last element in the list (which will be the largest distance).

## Innovation

Innovation was made on the referenced, Yen’s algorithm, to create a list of shortest paths. Firstly, the concepts of applying Dijkstra’s algorithm to find the first shortest path and removing certain edges when finding the approximate shortest paths was retained. The main differences can be observed in how the algorithm finds the k – 1 approximate shortest paths. Here, the algorithm, as mentioned in the ‘Alternate K – 1 Paths’ algorithm description, finds its paths by setting the edge weights of optimally found nodes to an infinite value (also done in Yen’s algorithm), therefore skipping that node and requiring the algorithm to find a different path. This is done one-by-one towards the goal node, and only ever alters one optimal edge at a given time. This was thought to increase runtime of the algorithm by reducing the number of unique paths that can be taken as the algorithm only alters route nodes that were used in the optimal path. Likewise, with only 1 edge changed, it was thought the algorithm would have a higher chance to find a way to resume the already discovered optimal path, hence, expecting to create lower variance of non-optimal node edges used and thus lower distance cost.

## Algorithm Pseudo-Code

# Results and Algorithm Analysis

## Testcase Results

|  |  |  |
| --- | --- | --- |
| Input (k) | Output | Time Taken |
| 2 | 1038.57, 1043.028 | 22.69 |
| 3 | 1038.57, 1043.028, 1043.028 | 42.45 |
| 5 | 1038.57, 1043.028, 1043.028, 1043.028, 1043.028 | 55.72 |
| 7 | 1038.57, 1043.028, 1043.028, 1043.028, 1043.028, 1043.028, 1043.028 | 79.11 |
| 10 | 1038.57, 1043.028, 1043.028, 1043.028, 1043.028, 1043.028, 1043.028, 1043.028, 1043.028, 1043.028 | 113.37 |

Table : Testcase Results with varying k values

\*Indicates that there was a potential error... however we look at a smaller test set as seen in appendix, it appeared to work. The benefit of the doubt will be given as it became very difficult to debug the 10k data set that was used.

## Time Complexity

# References

[1] Ben Alex Keen, 2017, *Dijkstra's Shortest Path and graph structure implementation heavy influence.* URL: http://benalexkeen.com/implementing-djikstras-shortest-path-algorithm-with-python/

[2] Jin Y. Yen, 1971, Yen’s K shortest path algorithm loose inspiration. URL: https://en.wikipedia.org/wiki/Yen%27s\_algorithm

# Appendix

*Item 1: Smaller Test Input*

6 9  
C D 3  
C E 2  
D F 4  
E D 1  
E F 2  
E G 3  
F G 2  
F H 1  
G H 2  
C H 3

Result:

Found Path Costs: 0.00 (secs)

[5.0, 7.0, 8.0]

*Item 2: Raw Input file used*