Best K Shortest Paths Algorithm

A Thesis

submitted by

RADHAKRISHNAN R V

for the award of the degree

of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF CIVIL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS

CERTIFICATE

This is to certify that the thesis entitled **Best K shortest Paths Algorithm**, submitted by Radhakrishnan R V to the Indian Institute of Technology, Madras, for the award of the degree of Bachelor of Technology is a bona fide record of the project work carried out by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma

Dr. Karthik K Srinivasan

Project Guide

Professor

Department of Civil Engineering

IIT Madras

Chennai 600036

Dr. K. Ramamurthy

Head of the Department

Professor

Department of Civil Engineering

IIT Madras

Chennai 600036

Date: 18th June 2017

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to Dr. Karthik K Srinivasan for his constant support and guidance throughout the study. It is through his guidance that the project has gained structure. His foresight and expertise have helped me make the right choices in the project. I am thoroughly indebted to him for the amount of time he has spent in reviewing my analyses and updates. I also thank him for offering the course *Transportation Network Analysis* which helped me have a formal and better understanding of the topics. I consider it a privilege to have worked under his guidance.

I would like to take this opportunity to thank the squash club of IIT Madras which made me reveal the sportsman in me. I would also like to thank the friends I found here at IIT Madras. Thank you for everything. Finally, I would like to thank my parents for the encouragement, love and support all these years.

ABSTRACT

The problem of finding the best K elementary paths in a network is studied here. Shortest path problem is one of the most widely studied classic network optimization problems. It started with the most basic problem of finding the shortest path between an OD pair in a given network with non-negative costs. Multiple variants of the problem like finding the best K shortest path between an OD pair and finding the shortest path in a network with time-varying costs have been studied over the years. The problem of finding the best K paths has many applications in vehicle routing, artificial intelligence, network optimisation, transportation planning among many others. The problem of finding the best K unconstrained paths also has its own applications in the form of scheduling rides which involve tours. In this paper, the computational complexity of the algorithm is optimised in two steps after discussing the existing algorithms. First, the data structure for the storage of the K labels for every node is made optimal. Then, the presence of cycles in a path is detected in a constant time instead of linear. It is done by storing the product of the primes of nodes in a path in different 128 bit variables. A theoretical computational time complexity of O(Km(log(Kn)+C)) was accomplished after this step. Then, the graph is reduced to contain the nodes and arcs present in the best K shortest paths and a few more. Then, the algorithm is run in both the directions to find the paths faster. A better practical running time was observed when tested in various real world networks and square grid networks after the second set of optimizations. Finally the practical relevance of large values of K is explained through the NP hard multiobjective optimization problem.

Keywords: K shortest paths, loopless, computational complexity, K labels, bidirectional search

ABBREVIATIONS AND NOTATIONS

G(V,E) : Graph, G with arc set E and node set V

G(m,n) : Graph, G with m arcs and n nodes

FIFO : First in First out

LIFO : Last in First out

TABLE OF CONTENTS

CHAPTER - 1

INTRODUCTION

1.1 Background and Motivation

Transportation network optimisation problems are one of most important problems used in transportation planning. The route choice at any given point of time requires the users to know the shortest path at that point of time. The best K shortest paths problem not only solves the shortest path but also the second, third and so on up to the K^{th} shortest path. The best K shortest path problem helps in knowing the next few shortest paths which might possibly be the shortest path at some point of time. The best K shortest paths problem has been solved by people like Hoffman, Pavley, Katoh and Yen, Pascoal and Martin among many others. The problem has been solved from 1959, more or less the same time when the case K = 1 was getting published. The three major solution methods used by them were the Deviation Based Algorithms, Label Setting Algorithms and the Label Correcting Algorithms. There are two kinds of sub-problems in this problem. One class has the constraint that the paths should not have cycles. The other class doesn't have the same constraint. The Yen's algorithm is one of the classic example for the constrained optimization problem whereas the Eppstein's algorithm is one for the unconstrained optimization problem. D'Shier published various label setting and label correcting algorithms in his papers in the 1990's. Pascoal and Martin published research work on the same with the latest being in 2006. The application of best K shortest paths algorithm in navigation softwares and other transportation planning problems makes it significant to study the problem. Most of the real world networks are not static. The cost of transportation networks say travel time or the fuel usage vary with time depending on the time of the day or the road traffic. The best K shortest paths algorithm can also be used to solve other variants of shortest path problems like the shortest path problem in a stochastic network and the time dependent shortest path problem. Substituting the dynamic networks with a static network in both these cases help us find the next few shortest paths that might possibly be the shortest path at a given point of time. Solving the best K shortest paths problem on a suitable network helps us solve the problem heuristically with a complexity independent of time.

1.2 Objective

The objective of the study is to reduce the computational time complexity of existing K shortest paths algorithm in four steps. First, better data structures enable us to optimize cycle checking and sorting of temporary labels of each node. Then, the network reduction reduces the size of the network on which the problem is solved. Then, the algorithm is run in both the directions to obtain really practically efficient running time on smaller networks.

1.3 Organisation of the thesis

This thesis involves eight broad chapters. We have discussed the application, scope and the overview of the problem in the current chapter. The second chapter follows it up with a brief literature review about the previous work of a few reputed authors. The third chapter brings up about the data used and the random data that were generated. The network representation used in the algorithm and the related proofs are discussed in the fourth chapter. The modification done to the algorithm is explained in the fifth chapter starting with the explanation about the data structure used. The pseudocode, proof of correctness and the complexity analysis of the algorithm follow it up. The sixth chapter deals with the modifications done to the algorithm to reduce its practical computational complexity. The seventh chapter involves the computational experiments that were performed using the C++ code run on a 64 bit g++ linux compiler. The data used were the ones described in the third chapter. The eighth chapter includes the practical significance of the improvements for large values of K. Then, the results, conclusion and the possible future work finishes the thesis. The C++ code used can be seen in the appendix of the thesis which follows the reference section.

CHAPTER - 2

LITERATURE REVIEW

The literature of best K shortest paths is found from the year 1957 which is about the same time when study of the special case of K = 1 began. The first algorithm published by Bock, Kantner and Hayes [9] is a brute force solution method in which all the possible paths between two nodes are listed and they are sorted. This involves an exponential computational time complexity. Eppstein (1997) solved the unconstrained version of the problem using a heap of shortest paths. The proposed algorithm has a computational time complexity of $O(m + nlogn + K)$ to find K shortest paths from one node to another and a complexity of $O(m + nlogn + Kn)$ to find K shortest paths from one node to all other nodes. Based on the constraint involved, the problems can be classified into two with one being the best K loopless paths problem and the unconstrained best K shortest paths problem. The other classification is based on the solution method involved. The three solution methods are the label setting algorithms, the label correcting algorithms and the deviation based algorithms.

2.1 Label Setting Algorithms

This class of algorithm includes the Dijkstra's algorithm which solves the case K = 1. While solving the K shortest paths problem using this method, each node has a set of K labels. This method involves initializing the source with zero distance label and other nodes with distance label infinity. Then the temporary label with the lowest weight is used to modify other labels in every iteration. The algorithm discussed in this thesis belongs to this class. The Dijkstra's algorithm is one of the first few algorithms proposed for finding the shortest path between two nodes. The algorithm involves having 1 label for every node and one node becomes permanent every iteration. The method of storing the labels needed improvements in its early stages. Various data structures like binary heap, radix heap, Fibonacci heap were proposed and the best complexity was $O\,(m + nlogn)$

D. Shier has published various label setting and label correcting algorithms until the 1990's. We

look at the one[2] that he proposed in 1979. The algorithm involves storing K labels for every node. The best temporary label is used to modify other labels in every iteration and the label is made permanent after this step. The best temporary label of each node is stored in a priority queue and the best label is selected in every iteration and made permanent. In Leo's modification[3], the k-1th label of all nodes are made permanent before the kth label of any node is made permanent. The algorithm involves maintaining K priority queues. It works on the principle that the kth label of a node can not improve the first k-1 labels of another node. Dreyfus' algorithm [18] is a modification of the deviation based solution proposed by Hoffman et al. [17]. It involves finding the shortest path by labelling for every node by considering one arc deviations, two arc deviations and so on. For finding unconstrained paths, Dreyfus' method involved a complexity of $O(Kn^2)$.

2.2 Label Correcting Algorithms

This class of algorithm is similar to the label setting algorithms but the condition that the temporary label with the least weight being selected in a label setting algorithm is not present in this algorithm. A FIFO priority queue is maintained and the selection of a node updates the labels of many other nodes. Bellman Ford's algorithm[15] to find the shortest path between two nodes is an example. It maintains a scan eligible list and processes nodes in a FIFO format. Floyd Warshall's algorithm[16] to solve the all pairs shortest paths problem is another example for this class. . This method of solution is mostly slower than its label setting counterpart.

2.3 Deviation Based Algorithms

This class of algorithm involves finding the shortest path first. Then the paths obtained by deviating from the shortest path are obtained. Hoffman and Pavley [17] proposed the first algorithm of this class. Pollack [10] later published an algorithm in which he found the shortest paths by eliminating one arc at a time from the previous k-1 paths. This algorithm involved a complexity of $O(n^k)$. So the complexity was exponential. Yen's algorithm [1] was proposed in 1971. Lawler (1972), Perko (1986) and Martin and Pascoal (2003) proposed their modifications of Yen's algorithm each of which had better order of selection of nodes. The Yen's algorithm involved the shortest path first. Then the nodes from the first to the kth node of a path is kept

fixed in k^{th} iteration. The shortest path from the k^{th} node to the sink is found and the resulting path is stored in a queue and the shortest path in the queue is processed after the previous path is fully processed. The algorithm terminates after K unique paths are processed. The complexity is $O(Kn(m + nlogn))$ when Fibonacci heap is used in the Dijkstra's algorithm to find shortest path in every iteration.

Katoh et. al (1982) proposed a method to solve it in undirected graphs. Katoh's method used the properties of undirected graph. The method generated three derivative paths for each found path. The ANSPR1 solution method of Cartyle and Wood (2005) solves the problem in a very fast time. Martin's and Pascoal (2006) is a further modification of the Yen's algorithm. This algorithm is a mixture of Perko's successive shortest paths algorithm and the one proposed by the same authors in 2003. This algorithm places bound on the maximum cost for a deviation path to be considered after finding K loopless paths. The worst case complexity is still the same as Yen's but under a very optimistic consideration that the paths are loopless, the algorithm has a computational complexity of $O(Kn + mlogn)$.

CHAPTER - 3

NETWORK AND DATA

3.1 Network Information

Data is one of the key components of network optimization research. The data of 22 real world and test networks that were obtained from https://github.com/bstabler/TransportationNetworks [14] were used to validate the algorithm. In addition to these, Chennai network with 61 nodes and 143 links were also used. A node in a network is a junction from which two or more pathways branch. A link is a connection between any two nodes. Each link of the network had 10 different variables along with the tail node and head node. The variables were the capacity, free flow time, length, power, speed limit and a few factors like toll, link type and B. The capacity of a link is the maximum flow that can pass through a link. The B and Power values are the parameters used in BPR function. Free flow time is the link travel time when the flow is zero. The link travel time is calculated using the BPR formula given by,

$$Link\ Travel\ Time\ =\ Free\ Flow\ Time\ *\ (1 + B * (flow/capacity)^{Power})$$

After substituting the given toll factor and distance factor values, the link cost is calculated using

$$Link\ Cost\ =\ Link\ travel\ time\ +\ toll\_factor\ *\ toll\ +\ distance\_factor\ *\ distance$$

This network can also be used with dynamic flow to arrive at a dynamic time-dependent network. The following is a test network named Sioux Falls with 24 nodes, 76 links and both toll factor and distance factor zero used in various publications. Note: This is not the real Sioux Falls network

Table 1: Sioux Falls Network Data

Tail Node	Head Node	Capacity (veh/hr)	Length (miles)	Free Flow Time (minutes)	B	Power	Speed Limit (mph)	Toll	Link Type
1	2	25900.20064	6	6	0.15	4	0	0	1

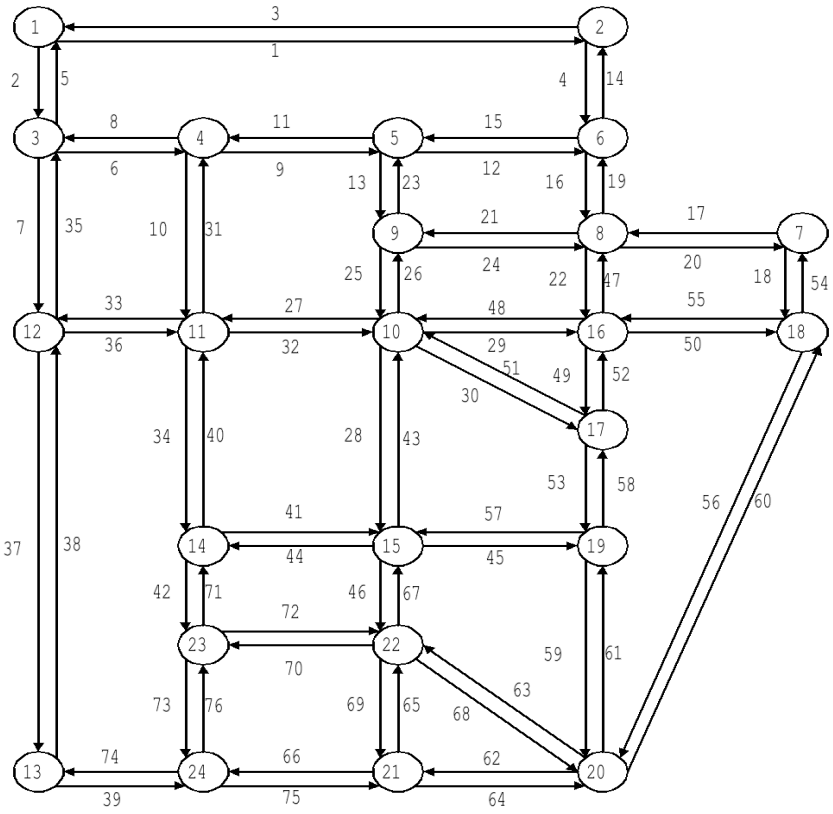
1	3	23403.47319	4	4	0.15	4	0	0	1
2	1	25900.20064	6	6	0.15	4	0	0	1
2	6	4958.180928	5	5	0.15	4	0	0	1
3	1	23403.47319	4	4	0.15	4	0	0	1
3	4	17110.52372	4	4	0.15	4	0	0	1
3	12	23403.47319	4	4	0.15	4	0	0	1
4	3	17110.52372	4	4	0.15	4	0	0	1
4	5	17782.7941	2	2	0.15	4	0	0	1
4	11	4908.82673	6	6	0.15	4	0	0	1
5	4	17782.7941	2	2	0.15	4	0	0	1
5	6	4947.995469	4	4	0.15	4	0	0	1
5	9	10000	5	5	0.15	4	0	0	1
6	2	4958.180928	5	5	0.15	4	0	0	1
6	5	4947.995469	4	4	0.15	4	0	0	1
6	8	4898.587646	2	2	0.15	4	0	0	1
7	8	7841.81131	3	3	0.15	4	0	0	1
7	18	23403.47319	2	2	0.15	4	0	0	1
8	6	4898.587646	2	2	0.15	4	0	0	1
8	7	7841.81131	3	3	0.15	4	0	0	1
8	9	5050.193156	10	10	0.15	4	0	0	1
8	16	5045.822583	5	5	0.15	4	0	0	1
9	5	10000	5	5	0.15	4	0	0	1

9	8	5050.193156	10	10	0.15	4	0	0	1
9	10	13915.78842	3	3	0.15	4	0	0	1
10	9	13915.78842	3	3	0.15	4	0	0	1
10	11	10000	5	5	0.15	4	0	0	1
10	15	13512.00155	6	6	0.15	4	0	0	1
10	16	4854.917717	4	4	0.15	4	0	0	1
10	17	4993.510694	8	8	0.15	4	0	0	1
11	4	4908.82673	6	6	0.15	4	0	0	1
11	10	10000	5	5	0.15	4	0	0	1
11	12	4908.82673	6	6	0.15	4	0	0	1
11	14	4876.508287	4	4	0.15	4	0	0	1
12	3	23403.47319	4	4	0.15	4	0	0	1
12	11	4908.82673	6	6	0.15	4	0	0	1
12	13	25900.20064	3	3	0.15	4	0	0	1
13	12	25900.20064	3	3	0.15	4	0	0	1
13	24	5091.256152	4	4	0.15	4	0	0	1
14	11	4876.508287	4	4	0.15	4	0	0	1
14	15	5127.526119	5	5	0.15	4	0	0	1
14	23	4924.790605	4	4	0.15	4	0	0	1
15	10	13512.00155	6	6	0.15	4	0	0	1
15	14	5127.526119	5	5	0.15	4	0	0	1
15	19	14564.75315	3	3	0.15	4	0	0	1

15	22	9599.180565	3	3	0.15	4	0	0	1
16	8	5045.822583	5	5	0.15	4	0	0	1
16	10	4854.917717	4	4	0.15	4	0	0	1
16	17	5229.910063	2	2	0.15	4	0	0	1
16	18	19679.89671	3	3	0.15	4	0	0	1
17	10	4993.510694	8	8	0.15	4	0	0	1
17	16	5229.910063	2	2	0.15	4	0	0	1
17	19	4823.950831	2	2	0.15	4	0	0	1
18	7	23403.47319	2	2	0.15	4	0	0	1
18	16	19679.89671	3	3	0.15	4	0	0	1
18	20	23403.47319	4	4	0.15	4	0	0	1
19	15	14564.75315	3	3	0.15	4	0	0	1
19	17	4823.950831	2	2	0.15	4	0	0	1
19	20	5002.607563	4	4	0.15	4	0	0	1
20	18	23403.47319	4	4	0.15	4	0	0	1
20	19	5002.607563	4	4	0.15	4	0	0	1
20	21	5059.91234	6	6	0.15	4	0	0	1
20	22	5075.697193	5	5	0.15	4	0	0	1
21	20	5059.91234	6	6	0.15	4	0	0	1
21	22	5229.910063	2	2	0.15	4	0	0	1
21	24	4885.357564	3	3	0.15	4	0	0	1
22	15	9599.180565	3	3	0.15	4	0	0	1

22	20	5075.697193	5	5	0.15	4	0	0	1
22	21	5229.910063	2	2	0.15	4	0	0	1
22	23	5000	4	4	0.15	4	0	0	1
23	14	4924.790605	4	4	0.15	4	0	0	1
23	22	5000	4	4	0.15	4	0	0	1
23	24	5078.508436	2	2	0.15	4	0	0	1
24	13	5091.256152	4	4	0.15	4	0	0	1
24	21	4885.357564	3	3	0.15	4	0	0	1
24	23	5078.508436	2	2	0.15	4	0	0	1

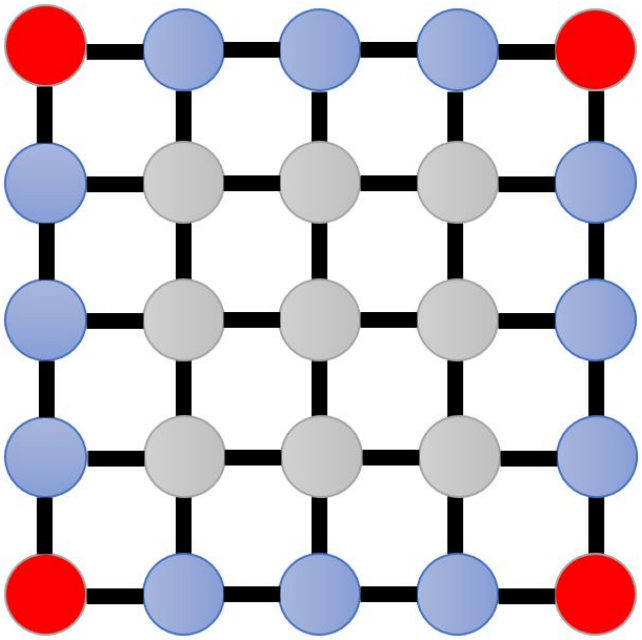
Fig 1: Sioux Falls Network



3.2 Grid Network Generation

Another efficient way of validating the algorithms is by generating grid networks with random costs in a given range. Grid network is a network in which each node except the corner nodes has 4 links with 1 link in each of the north, west, south, east direction. Using grid networks help us arrive at the results independent of the network topology which is a major deciding factor in arriving at the computational complexity. A rectangular grid network with *l* nodes in one row of the longer sides and *b* nodes in one column of the longer side has *lb* nodes and *4lb - 2l - 2b* arcs. For example, the following network has 25 nodes and 80 links (each link considered in both the directions)

Fig 2: 5*5 square grid network



CHAPTER - 4

PRELIMINARIES

4.1 Path and Network Representation

The k^{th} permanent label from the source to a node, i is represented using a weight, $d^k(i)$ and predecessor, $pred^k(i)$ and the path is represented as $p^k(i)$. The linear cycle check is done by backtracking. The paths are also outputted after backtracking. While searching in both the directions the forward and backward labels are used for each node. The k^{th} backward label from a node, i to the sink is represented using weight, $d^{*k}(i)$ and predecessor, $pred^{*k}(i)$ and the path is represented as $p^{*k}(i)$. Let the path made permanent in the k^{th} iteration with label i and j in forward and backward direction be $q^k(i)$ and $q^{*k}(j)$.

The graph is stored in forward star and backward star representations. Forward star representation is one in which the arcs form a list and the arcs are sorted in an increasing order of their tail node. The backward star representation consists of those arcs sorted in an increasing order of their head node. Another list of values called point is used to track the starting position of the arcs of each tail node (head node) in a forward (backward) star representation. $outdegree(i) = point(i+1) - point(i)$ in a forward star representation and

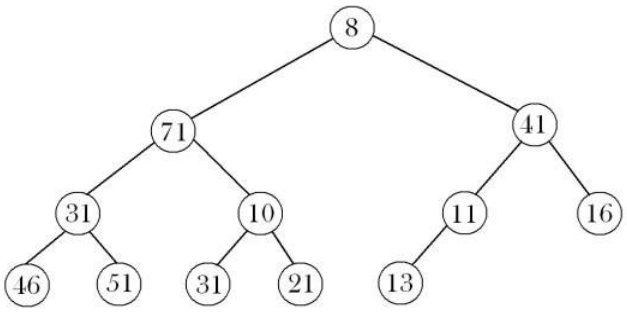
$indegree(i) = point(i+1) - point(i)$ in a reverse star representation.

4.2 Minmax Heap

A minmax heap is a double ended priority queue as explained by . It is used to insert or delete both the maximum element and the minimum element of a list in logarithmic time. The minimum and the maximum element in a list is obtained in constant time. This is unlike the classic min heap or max heap that help in inserting or deleting only one of the minimum and maximum element in logarithmic time and the other in linear time. Also finding one of the minimum or maximum element is done in constant time while the other is done in linear time. While the odd levels form a min heap, the even levels form a max heap. A minmax heap containing the elements 8, 10, 11, 13, 16, 21, 31, 31, 41, 46, 51, 71 looks like the following

image.

Fig 2: Minmax Heap Representation



4.3 Proof 1

Claim: Consider a set consisting of the union of shortest path from the source to each node and from each node to the sink. The network obtained by the nodes present in the best K loopless unique paths in this set (if K exist) and the arcs connecting them consist the best K shortest paths in the original network.

Proof: Let us prove this by contradiction. Assume that the claim is false. This would imply that a node which is not a part of the obtained subgraph is present in the best K shortest paths. This would require at least one of the paths from the source to sink passing through this node to be a part of the best K loopless unique paths in that set. Since no such path existed, the claim is true.

4.4 Proof 2

Claim: In the set of paths obtained from the union of $q^{k1}(i)$ and $q^{*k2}(j)$ at the end of k^{th} iteration for all k1, k2 ≤ k such that (i,j) forms an arc, the subset formed by the paths whose $d^{k1}(i) + d^{*k2}(j) \leq d^{k}(a) + d^{*k}(b)$ is not worse than the paths that are not a part of the subset

Proof: Let us prove this by contradiction. Assume that a path formed by the union of $q^{k1}(i)$ and $q^{*k2}(j)$ such that k1 ≥ k is better than one of the paths in the subset. This would imply that $d(i,j)+d^{*k2}(j) \leq d^{*k}(b)$. But if i had been made permanent in the backward direction, that label along with the label corresponding to pred(i) in the forward direction should have formed a union and resulted in the path being a part of the subset. Since the path is not present, the claim is true.

CHAPTER - 5

ALGORITHM WITH THEORETICAL IMPROVEMENTS

5.1 About this Algorithm

Every node has a permanent label set and temporary label set associated with it. The total number of labels for every node has a maximum limit of K. The permanent label set is stored in the form of an insertion only no deletion list. A heap, H stores the nodes in the form of a heap arranged according to their best temporary label.

The temporary label set is stored in a minmax heap. Maintaining a double ended priority queue helps us obtain the smallest element in every node selection step and the largest element if required in the arc processing step in constant time. The computational complexity while using a min-max heap is logarithmic when compared to the linear complexity for a list. The rearranging of the heap on insertion/deletion of an element can be done in logarithmic time as opposed to the linear time in a list.

Each label has a few product elements associated with it. The cycle checking is usually done by checking the presence of the new node across the existing path. This takes up linear time and grows with path size. Each node can be associated with a prime number say, the n^{th} prime number can be mapped with node, n. Each path can have a few products associated with it which are the product of the prime numbers mapped with the nodes that satisfy a specific criteria in the path. For example, the prime numbers whose node number modulo 10 can be grouped together. Checking the presence of a node in the path can be done by checking if the respective product divides the prime number mapped with the node. This helps in achieving a constant time complexity.

The algorithm is similar to that of D.Shier except for the changes mentioned earlier. This algorithm also works on the principle of label setting and maintains K labels. The smallest temporary label is chosen in every iteration. Following is the pseudocode

5.2 Pseudo Code

K Shortest Paths (Graph, Origin, Destination, K)

- Initialization (graph, source)
    - The permanent label set of every node is declared empty
    - The temporary label set is initialized with the source with a weight of 0
    - Heap, H is initialized with the source
- Node Selection (graph, heap, labelsets)
    - The root node of the heap, H is selected
    - The corresponding label is removed from the minmax heap and the heap is rearranged
    - Heap, H is rearranged with the new best temporary label of the root node
- Arc Processing (graph, root node, labelsets)
    - For all the arcs coming out of the selected node
        - If the label, i is better than the k^{th} label of the head node
            - If acyclic (tail node, i)
                - If the number of labels equals K
                - Remove the k^{th} label and rearrange the minmax heap
                - Insert the new label into the minmax heap and rearrange it
                - Rearrange heap, H if the best temporary label of the head node is updated
    - The heap, H is rearranged
- Termination
    - If the number of permanent labels of the sink equals K or If there is no temporary label
        - Terminate
    - Else
        - Go to node selection step

5.3 Time Complexity

- Initialization is done with a time complexity of O(n)
- Node selection:
    - Identifying the root node is done in constant time. Rearranging heap, H takes O(logn) time since it has a maximum of n elements. Rearranging the minmax heap takes O(logK) time since they have a maximum of K elements. This is done a maximum of Kn times. Thus the overall complexity in this step is O(nKlog(nK))
- Arc processing:
    - Since the graph is stored in a forward star format, identifying the arcs takes constant time.
    - Cycle checking is done by checking the divisibility with 1 product element hence resulting in a complexity of O(C)
    - Rearranging heap, H takes O(logn) time since it has a maximum of n elements
    - Re-arranging the minmax heap takes O(logK) time since it has a maximum of K elements
    - This is done a maximum of Km times
    - Thus the overall complexity of this step is $O(Km(log(nK) + C))$
- Thus the overall complexity is $O(Km(log(nK) + C))$

5.4 Proof of Correctness and Convergence

Claim 1: The label that is made permanent in ith iteration is the ith best label of the network from the source

Proof: The labels that are selected in an iteration is the best of all the temporary labels available at that instant. So all the labels created out of them after adding an arc will have greater weight than them since the arc costs are non-negative. The label {source} whose weight = 0 is the best label since all the arc costs are non-negative. Let us assume that the kth permanent label is the kth best label and all the labels before that are in order. So the (k+1)th permanent label was temporary when those labels were made permanent. So it is the (k+1)th best label.

Claim 2: The algorithm terminates after some time

Proof: The maximum number of permanent labels that can be created is Kn. The algorithm terminates when there is no temporary label available. So the maximum steps that the algorithm can progress is Kn.

CHAPTER - 6

ALGORITHM WITH PRACTICAL IMPROVEMENTS

Practical improvements are still feasible in this algorithm. The practical improvements possible are discussed in this section. Network reduction is done on the graph to reduce the size of the graph on which we solve the problem. Section 4.3 states the method and the proof of correctness for this step. The bidirectional search used to arrive at the solution faster practically. Section 4.4 deals with the method and the proof of correctness for this step. Network reduction is done first and then the algorithm is run on the resulting subgraph. The computational experiments discussed in the next section also gives the practical running time of the previous steps.

6.1 Network Reduction:

Network reduction is an efficient method to reduce the problem size of graph problems. The claim given in section 4.3 helps us arrive at a smaller graph than the given graph with an optimal theoretical complexity of $O(n^2)$ and a worst case theoretical complexity of $O(n^3)$. The nodes in the best K unique elementary paths in the set of paths formed by union of paths from source to each node and from each node to the sink and the arcs connecting them are used to form the new subgraph. This method can be used only when such best K unique elementary paths exist. So if n is not much greater than K, this method need not be used. The pseudo code of the method is as follows.

6.1.1 Graph Reduction (Graph, source, sink, K)

- Find the shortest path of each node from the source and then from each node to the sink
- The paths are sorted
- A set is used to store the nodes that are part of the best K paths
- For every path
    - if the same path exists before move to the next path else go to the cycle checking step
    - if cycle is present after merging both direction paths go to the next path else the nodes of the path are stored in the set

6.1.2 Computational Complexity

The shortest path of each node from the source and the from each node to the sink are done using Dijkstra's Algorithm. So the complexity of this step would depend on the implementation of Dijkstra's Algorithm. Implementing it using advanced data structures like Fibonacci heap involves a complexity of $O(m + nlogn)$. The paths when sorted using count sort which will reduce the complexity to linear time $O(n)$. The worst case of uniqueness checking would involve checking the next path in the queue against all the previously added paths. Hence the step would involve a complexity of $O(n^3)$. Finding the presence of cycle after merging the forward and reverse paths would involve a complexity of $O(n^2)$. So the worst case complexity is $O(n^3)$ and an optimal complexity is $O(Kn)$. Consider that the graph, G with m arcs and n nodes is reduced to a graph G* with m* arcs and n* nodes after this step.

6.2 Bidirectional Search

This method is used to reduce the number of iterations that the algorithm runs for. Practically, if we run the same algorithm that we discussed in the previous section in both the directions, the number of iterations decreases a lot and it improves the practical efficiency. We require the backward star method of storing the arcs in addition to the forward star to run the algorithm in the reverse direction. We maintain two heaps H_1 and H_2 to store the nodes arranged according to their best temporary labels in forward and backward direction respectively. Each node has two label sets h_1 and h_2 in the form of min max heaps for forward and backward directions respectively.

6.2.1 Pseudocode

- Step 1: Network reduction is done as discussed in the previous section and the nodes are renumbered from 1 to n*
- Step 2: Initialize the 1st distance label of the source and the 1st distance label of the sink to zero and the other labels to infinity. H_1 is initialized with the source and H_2 with the sink
- Step 3: The following four steps are done 10 times
    - Forward Direction Node Selection (graph, heap, labelsets)
        - The root node of the heap, H_1 is selected

        - The corresponding label is removed from the minmax heap and the heap is rearranged
        - Heap, H_1 is rearranged with the new best temporary label of the root node
    - Forward Direction Arc Processing (subgraph, root node, labelsets)
        - For all the arcs coming out of the selected node
            - If the label, i is better than the k^{th} label of the head node
                - If acyclic (tail node, i)
                    - If the number of labels equals K
                        - Remove the k^{th} label and rearrange the minmax heap
                    - Insert the new label into the minmax heap and rearrange it
                    - Rearrange heap, H_1 if the best temporary label of the head node is updated
        - The heap, H is rearranged
    - Backward Direction Node Selection (subgraph, heap, labelsets)
        - The root node of the heap, H_1 is selected
        - The corresponding label is removed from the minmax heap and the heap is rearranged
        - Heap, H_1 is rearranged with the new best temporary label of the root node
    - Backward Direction Arc Processing (subgraph, root node, labelsets)
        - For all the arcs coming out of the selected node
            - If the label, i is better than the k^{th} label of the head node
                - If acyclic (tail node, i)
                    - If the number of labels equals K
                        - Remove the k^{th} label and rearrange the minmax heap
                    - Insert the new label into the minmax heap and rearrange it

        - Rearrange heap, H_1 if the best temporary label of the head node is updated
    - The heap, H is rearranged
- Step 4: The sum of weights of labels in forward and backward directions is set as the upper bound. The union of permanent paths in forward and backward direction that share the last node or that are separated by one arc is found. The unique loopless paths are put in a separate set, P
- Step 5: If the size of P is greater than or equal to K, the algorithm terminates. Else step 3 is performed

6.2.2 Computational Complexity

- Network reduction involves a worst case complexity of $O(n^3)$ and a best case complexity of $O(Kn)$ as discussed earlier
- Initialization is done with a time complexity of O(n*) in both the directions
- Node selection for both the directions:
    - Identifying the root node is done in constant time. Rearranging heap, H_1 and H_2 takes O(logn*) time since it has a maximum of n elements. Rearranging the minmax heaps takes O(logK) time since they have a maximum of K elements. This is done a maximum of Kn* times. Thus the overall complexity in this step is O(n*Klog(n*K))
- Arc processing:
    - Since the graph is stored in a forward star format, identifying the arcs takes constant time.
    - Cycle checking is done by checking the divisibility with 1 product element hence resulting in a complexity of O(C)
    - Rearranging heap, H_1 and H_2 take O(logn*) time since they have a maximum of n* elements
    - Rearranging the minmax heaps takes O(logK) time since they have a maximum of K elements

    - This is done a maximum of Km* times
    - Thus the overall complexity of this step is $O(Km^*(log(n^*K) + C))$
- The post processing step can be performed on a maximum of K^2 combinations for each node. So it can be performed a maximum of K^2n times. Each combination would involve cycle check. So the complexity of this step is $O(K^2(n^*)^2)$.
- Thus the overall complexity is $O(Km^*(log(n^*K) + C) + K^2(n^*)^2 + n^3)$
- Since the complexity term of the algorithm when run on both the directions involves a K^2n^2 term, the algorithm is optimal for small Kn values and is worse than the single directional version for larger values of Kn

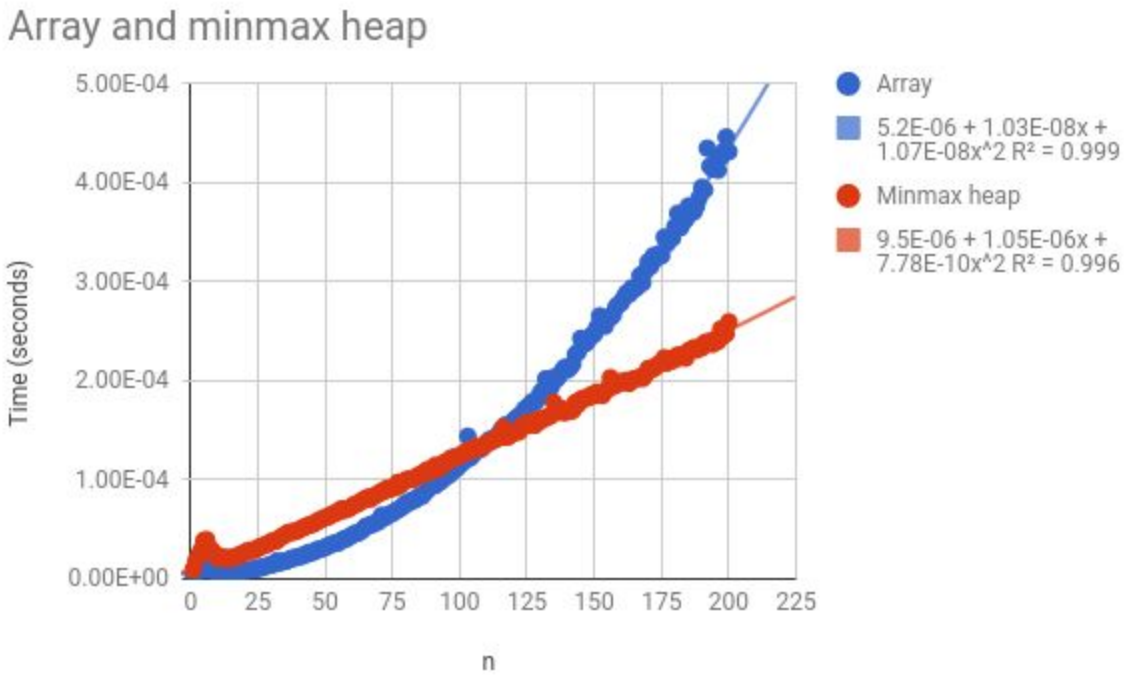
CHAPTER - 7

EXPERIMENTS

First we check the computational performance module wise and then implement it together in the overall algorithm

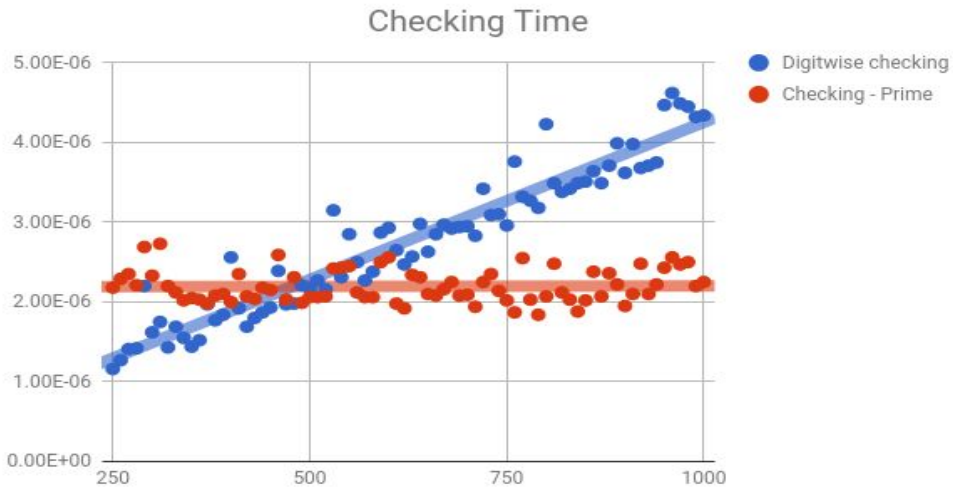
7.1 Double ended priority queue vs Linear array Performance

The time complexity involved in rearranging the array when stored linearly and when stored in the form of a double ended priority queue are computed. The time complexity involved in heapification of the minmax heap after addition or deletion of an element is observed to increase logarithmically whereas the time complexity involved for the same in a linear array is linear. The logarithmic curve is observed to cut the line somewhere around n = 110. This implies that the data structure is asymptotically better. The experiment involved performing the operations n times which gave a complexity of $O(n^2)$ for a linear array and a complexity of $O(nlogn)$ for a minmax heap.



7.2 Cycle checking by linear search vs checking with divisibility of prime

The time consumed to find the number when searched linearly is plotted. The time consumed to find the number by checking if the respective product of primes of the nodes in the path is divisible by the prime of the node. Assuming a fair distribution in nodes the product of primes of nodes whose remainder with 10 are the same are grouped together. The experiment gave a computational complexity of $O(C)$ as observed in the following figure. It is observed from the figure that the computational complexity of the algorithm is better for large path values



7.3 Experiments for Setting Parameters

Now the algorithm is tested on various test networks and grid networks to arrive at the practical values of the different parameters associated with the algorithm in its single directional version and its bidirectional version. The mean, median, maximum, 90th percentile and 99th percentile of the size of the temporary label set of the temporary nodes, the size of the temporary label set of the permanent nodes, the size of heap of temporary labels and path size are measured with different OD pairs and the results are averaged.

Barcelona Network (Nodes - 1020; Arcs - 2522):

K	Temporary node label set size 99th percentile	Permanent node label set size 99th percentile	max path size	heap size 99th percentile
100	92	88	31	279

200	182	173	33	323
300	272	258	33	346
400	361	342	34	363
500	450	426	34	376
600	538	511	34	386
700	626	595	35	395
800	714	679	35	401
900	802	762	35	408
1000	889	846	35	414

Winnipeg Network (Nodes - 1052; Arcs - 2836):

K	Temporary node label set size 99th percentile	Permanent node label set size 99th percentile	max path size	heap size 99th percentile
100	99	96	70	351
200	197	190	70	351
300	296	283	70	351
400	395	376	70	351
500	492	468	70	351
600	589	560	70	351
700	686	650	70	351
800	782	741	70	351
900	878	830	71	351
1000	973	921	71	351

Austin Network (Nodes - 7388; Arcs - 18961):

K	Temporary node label set size 99th	Permanent node label set size 99th	max path size	heap size 99th percentile

	percentile	percentile		
100	99	96	101	439
200	198	190	102	512
300	296	284	103	553
400	395	379	103	585
500	493	473	104	609
600	591	566	104	628
700	689	660	105	645
800	787	753	105	661
900	885	847	105	675
1000	983	940	105	688

The following tables help us know the ratio of actual number of temporary labels created to the maximum available which is Km in the single directional version. The next column has the ratio of number of permanent labels created to the maximum available which is Kn. The next column is the ratio of cycle checks done to the maximum that can be done which is Km. Last column has the failed cycle check ratio.

Barcelona Network (Nodes - 1020; Arcs - 2522):

K	Temporary label creation ratio	Permanent label creation ratio	cycle check ratio
100	23.89601507	50.3744281	26.51695083
200	24.22536677	51.02871732	27.15440788
300	24.39556899	51.33190359	27.46916579
400	24.4818555	51.48104575	27.64657679
500	24.55611948	51.59728758	27.78836902
600	24.637614	51.73448393	27.9270971
700	24.6941864	51.81364963	28.02856057
800	24.73663428	51.87833946	28.10788189
900	24.777563	51.945502	28.18401401

1000	24.81496332	51.99566585	28.25204699

Winnipeg Network (Nodes - 1052; Arcs - 2836):

K	Temporary label creation ratio	Permanent label creation ratio	cycle check ratio
100	22.30337094	51.17391635	29.38222849
200	22.6692031	51.93096958	30.36827221
300	23.04941702	52.75680608	31.14344617
400	23.60891396	54.05953422	32.06038787
500	23.94798307	54.78701901	32.60297602
600	24.72510343	56.94686312	33.7295134
700	24.76746121	57.01900598	33.93001813
800	24.79351551	57.07114068	34.02252292
900	23.9508729	54.72791719	32.95856762
1000	24.15729055	55.20990114	33.25500423

Austin Network (Nodes : 7388; Arcs: 18961)

K	Temporary label creation ratio	Permanent label creation ratio	cycle check ratio
100	22.20059491	50.43779101	24.71530826
200	22.20951005	50.4766811	24.94019197
300	22.20435983	50.4787403	25.04951216
400	22.20226676	50.48640363	25.12835083
500	22.19707484	50.48722144	25.18074785
600	22.19121179	50.4842754	25.21874479
700	22.18848674	50.4872233	25.25254334
800	22.18638521	50.48925217	25.28192421
900	22.18477202	50.49234314	25.30798915
1000	22.18429323	50.49532052	25.33126481

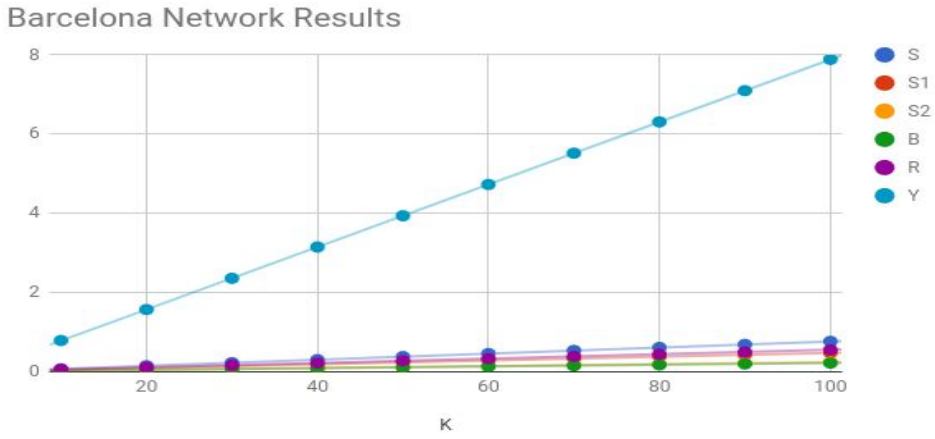
Summary

The path size in real world networks had a maximum size of 3*√n. So the cycle checking practically has a complexity of O(3√n) and done a maximum of 0.4*Km times. Rearranging the minmax heap and the heap, H is performed around 0.25Km times after arc processing. Also rearranging the minmax heap and the heap, H is performed around 0.6Kn times after node selection. So the overall complexity of the single directional version is parameterized as $O(0.25Km^*(log(n^*K) + 1.2Km^*\sqrt{n^*}) + 0.6n^*Klog(n^*K))$.

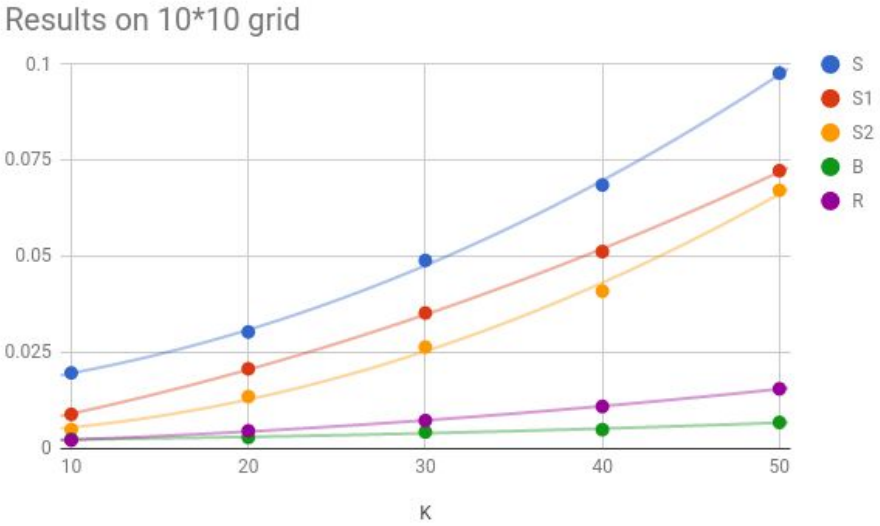
7.4 Experiments on Computational time

Now the algorithm is tested on grid networks of different sizes and test networks to arrive at the practical running time. The algorithm that uses minmax heap alone along with the D.Shier's algorithm is run first. Then the algorithm that checks for the divisibility of primes for cycle check alone is used along with D.Shier's Algorithm is run. Then both the changes are implemented together and run. Then the bidirectional version of the Shier's algorithm with minmax heap is run. It is observed to show good results for small values of Kn. Then the bidirectional version along with network reduction is run. The computational running time (in seconds) were obtained as follows. S will represent Shier's Algorithm. S1 will represent Shier with minmax heap and cycle check using prime number divisibility. S2 will represent Shier with minmax heap alone. B represents Bidirectional version of S2. R represents version of B after network reduction. Y represents Yen's Algorithm. Since Yen's algorithm shows a very high computational time for large Kn values, it is not computed for that

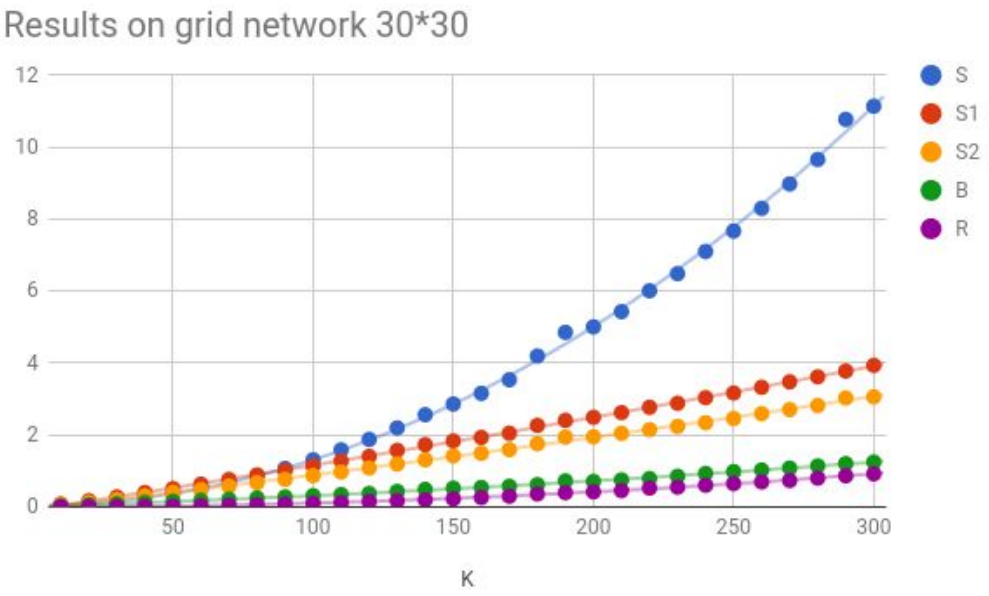
Barcelona Network (1020 nodes, 2522 arcs)



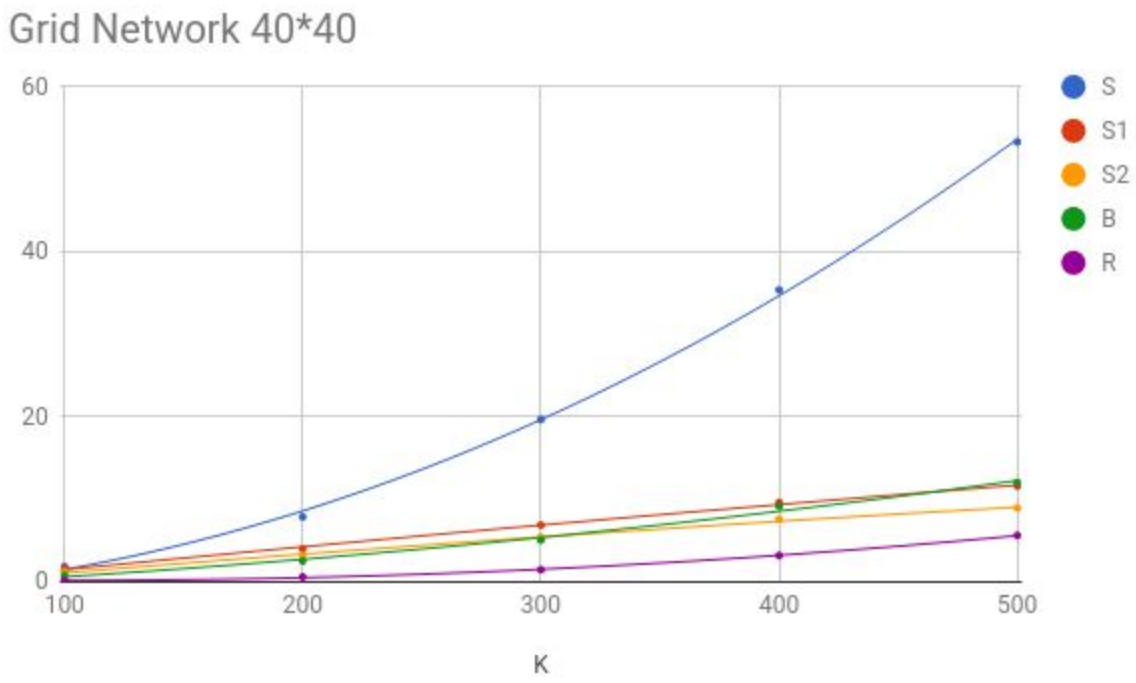
Grid network : 10*10



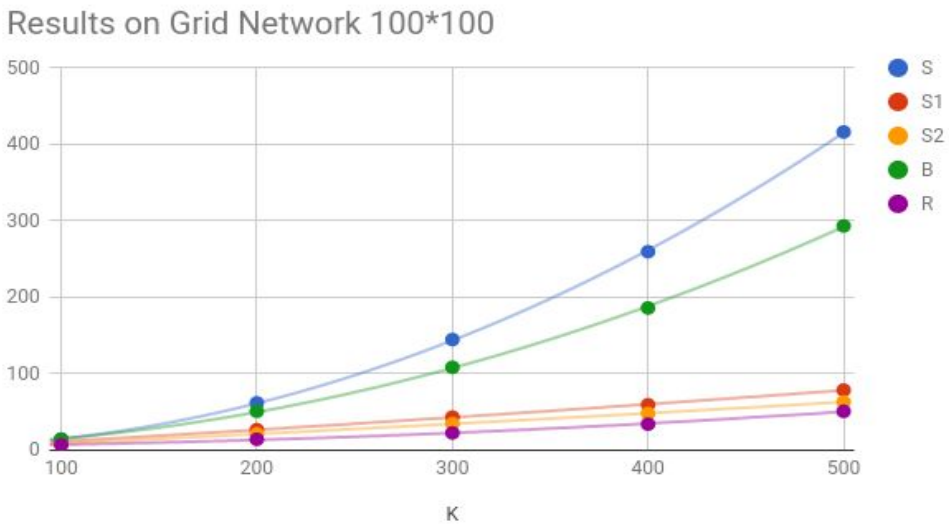
Grid network : 30*30



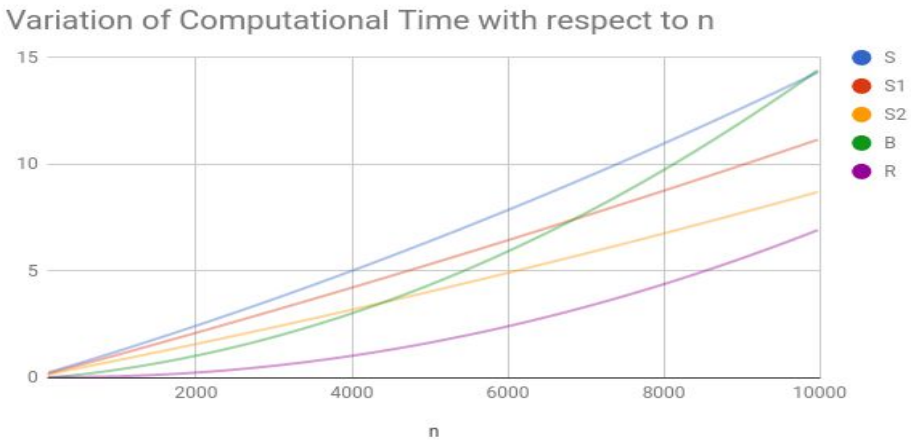
Grid network 40*40



Grid network : 100*100



Computation Time vs Number of Nodes in the Network



Summary

From the obtained results it is clear that bidirectional version of the algorithm is suited for small Kn values and its running time shoots up for larger values. Network reduction is seen as an effective way of improving running time from the obtained results. But the network reduction suffers from the limitation that it cannot be used for large K values (K>n/2). In the single directional versions, it is clear that the temporary label set modification done to Shier's algorithm is more effective and gives better practical results than the cycle check modification.

CHAPTER - 8

RESULTS AND CONCLUSION

8.1 Relevance of Large K in Multiobjective Optimization Problem

As observed in the results, the relevance of the change in data structure to store the temporary labels of nodes comes into picture only when the number of temporary labels is above a threshold which is around 125. This implies that the value of K should be large enough such that the temporary label set size reaches the threshold in a large number of iterations. This makes it important to explain the relevance of such large K values in real world applications.

Multiobjective optimization is an area of multi-criteria decision making that is concerned with problems that involve multiple objective functions. A few examples include the tradeoff between the comfort and price while buying a car and the need to minimize travel time while minimizing fuel usage and pollution. For a non trivial multiobjective optimization problem, a set of non dominated or pareto optimal solutions exist. A non dominated or pareto optimal set is one in which any solution in the set does not dominate another solution in the set but it dominates every solution outside the set. It is an NP hard problem whose solution set can go upto infinity. In a network optimization problem, the solution set can possibly include all the feasible paths in the network. The value of K in this problem is not known prehand and is found out only after solving it. So, the restriction in the number of labels that can be stored per node is removed when we solve the problem unlike the demonstrated algorithm in which there can be a maximum of K labels that can be stored per node. The problem is solved until a dominated label is obtained for the sink node. The computational complexity of the single direction version of the algorithm with respect to K is of the order of KlogK against the K^2 in the previous algorithm. So this helps in solving the problem more effectively.

8.2 Summary of the Research Work

The different solution methods as present in renowned literature were discussed in the beginning of the study. Then the data used for the research work was discussed. Example data was presented and the method of generating grid networks was explained. The network

representation used in the algorithm and the other notations and proofs were discussed in the fourth chapter. The fifth chapter involved the theoretical improvements in solving the problem. The complexity of finding the best temporary label was made logarithmic with a double ended priority queue. The improvement was observed to be relevant for large K values. The complexity when the cycle check was carried out linearly was observed to be $O(Km(logK + n)$ and when the cycle check was carried out by checking the divisibility of the prime with the product of primes, it was observed to be $O(Km(log(nK) + C)$. The prime number divisibility method of cycle checking is observed to show better results only when the nodes in the paths are equally distributed across the range. After performing the computational experiments, the average number of iterations performed were observed. This helped us parameterize the complexity and tighten it to a smaller expression. Performing the experiments on grid networks ensured topological uniformity to understand the role of n on the computational complexity term.

8.3 Possible Future Work

There are a number of places where this algorithm can be used. Two ways were discussed in this study. The multiobjective optimization problem involves successively finding the shortest paths before a dominated path is observed. Though the NP hardness of the problem is not resolved, the complexity involved in arriving at the solution decreases. The problem of finding the shortest path in a time dependent network can be solved by solving the K shortest paths algorithm for a given K and a suitable static network and finding the best path among these paths for every time period.

REFERENCES

[1] J.Y. Yen. (1971) “Finding the k shortest loopless paths in a network”, *<http://people.csail.mit.edu/minilek/yen_kth_shortest.pdf>*

[2] D. Shier. (1974) “Computational experience with an algorithm for finding k shortest paths in a network” Journal of the Research of the NBS, 78

[3] John Leo (1993) “Time-varying K shortest paths algorithm” Matsushita Communication Industrial Ltd.,

<https://pdfs.semanticscholar.org/c39f/7ff4e09c5b49573c5db9c3a2b044083d0e51.pdf>

[4] D. Eppstein (1999) “Finding the k shortest paths,” SIAM J. Comput., vol. 28, no. 2, pp.652-673,

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.39.3901&rep=rep1&type=pdf>

[5] Myrna Palmgren and Di Yuan "A Short Summary on K Shortest Path: Algorithms and Applications",

<http://homepages.engineering.auckland.ac.nz/~amas008/Courses/LinkopingColGen99/kth.pdf>

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. "Introduction to Algorithms, Third Edition. The MIT Press, 3rd edition, 2009"

[7] Ravindra K. Ahuja, Thomas L.Magnanti, James B.Orlin "Network Flows – Theory, Algorithms and Applications"

[8] E.Q.V. Martins, M.M.B. Pascoal and J.L.E. Santos (1999) "Labelling algorithms for ranking shortest paths"

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.46.7725&rep=rep1&type=pdf>

[9] Bock, F., Kantner, H. and Haynes, J., An Algorithm (The rh Best Path Algorithm) for Finding and Ranking Paths Through a Network, Research Report, Armour Research Foundation, Chicago, Illinois, November 15, 1957

[10] Pollack, M., "The kth Best Route Through a Network," Opns. Res., Vol. 9, No. 4 (1961), pp. 578

[11] Clarke, S., Krikorian, A. and Rausan, J., "Computing the N Best Loopless Paths in a Network," J. of Siam, Vol. 11, No. 4 (December 1963), pp. 1096-1102

[12] Sakarovitch, M., The k Shortest Routes and the k Shortest Chains in a Graph, Operations Research Center, University of California, Berkeley, Report ORC-32, October 1966

[13] Athanasiosis K. Ziliaskopoulos and Hani S. Mahmassani Time-Dependent Shortest Path Algorithm for Real Time Intelligent Vehicle Highway System Applications

[14] Ben Stabler, Hillel Bar-Gera, Elizabeth Sall "Transportation Networks Repository, Github", 2016 <*https://github.com/bstabler/TransportationNetworks*>

[15] Richard Bellman. On a routing problem. Quarterly Applied Mathematics, XVI(1):87– 90, 1958

[16] Floyd, Robert W "Algorithm 97: Shortest Path" Communications of the ACM (June 1962) <*https://doi.org/10.1145%2F367766.368168*>