B.Sc. in Computer Science and Engineering Thesis

A New Approach to Calculate All Pairs Shortest Path

Submitted by

Namrata Saha
201414025

Sadia Akter
201414035

Saborni Shernaj Binte Elahi
201414039

Supervised by

Dr. Mohammad Kaykobad

Professor

Department of Computer Science and Engineering

Bangladesh University Of Engineering and Technology

Department of Computer Science and Engineering
Military Institute of Science and Technology

December 2017

CERTIFICATION

This thesis paper titled **"A New Approach to Calculate All Pairs Shortest Path"**, submitted by the group as mentioned below has been accepted as satisfactory in partial fulfillment of the requirements for the degree B.Sc. in Computer Science and Engineering in December 2017.

Group Members:

1. **Namrata Saha**
2. **Sadia Akter**
3. **Saborni Shernaj Binte Elahi**

Supervisor:

Dr. Mohammad Kaykobad
Professor
Department of Computer Science and Engineering
Bangladesh University Of Engineering and Technology

CANDIDATES' DECLARATION

This is to certify that the work presented in this thesis paper, titled, "A New Approach to Calculate All Pairs Shortest Path", is the outcome of the investigation and research carried out by the following students under the supervision of Dr. Mohammad Kaykobad, Professor, Department of Computer Science and Engineering, Bangladesh University Of Engineering and Technology.

It is also declared that neither this thesis paper nor any part thereof has been submitted anywhere else for the award of any degree, diploma or other qualifications.

Namrata Saha
201414025

Sadia Akter
201414035

Saborni Shernaj Binte Elahi
201414039

ACKNOWLEDGEMENT

We are thankful to Almighty Allah for his blessings for the successful completion of our thesis. Our heartiest gratitude, profound indebtedness and deep respect go to our supervisor, **Dr. Mohammad Kaykobad**, Professor, Department of Computer Science & Engineering, Bangladesh University of Engineering & Technology, for his constant supervision, affectionate guidance and great encouragement and motivation. His keen interest on the topic and valuable advice throughout the study was of great help in completing thesis.

We are especially grateful to the Department of Computer Science and Engineering (CSE) of Military Institute of Science and Technology (MIST) for providing their all out support during the thesis work.

Finally, we would like to thank our families and our course mates for their appreciable assistance, patience and suggestions during the course of our thesis.

Dhaka
December 2017.

1. Namrata Saha
2. Sadia Akter
3. Saborni Shernaj Binte Elahi

ABSTRACT

The issue of the thesis work is All pairs shortest path problem. If it is considered simple and widely used algorithm in the field of all pairs shortest path problem,Floyd Warshalls Algorithm is the most famous one, which is to compute shortest path between all pairs of nodes in a directed weighted graph. Researchers have given many other approaches to reduce the time and space complexity of this problem. To determine the shortest distances in a given graph between every pair of vertices, is called the all-pairs shortest path problem. The solution of the problem can be developed mainly by using the Floyd-Warshall algorithm. By using the major applications of Dijkstra's algorithm, this all pairs shortest path problem can be explained. All pairs shortest path is one of the most important topics in the field of graph theory, computational complexity and algorithm design. Our research aim is to build an efficient algorithm for all pair shortest path problem in a graph that will provide improvement in the computation and run time analysis. This is experimented with some examples of directed graphs.

Contents

List of Figures

List of Tables

LIST OF ABBREVIATION

SG : Simple Graph
WG : Weighted Graph
MST : Minimum Spanning Tree
APSP: All Pairs Shortest Path
BFS : Breadth First Search

CHAPTER 1
INTRODUCTION

1.1 Overview

The objective of this thesis paper is to find a new sufficient approach for the determination of all pairs shortest path in a directed graph. In order to solve the all-pairs shortest path problem,it is to be determined the shortest graph distances between every pair of vertices in a given graph. The solution of the problem can be developed with the help of using the Floyd-Warshall algorithm. This algorithm works in a weighted graph means where there is a certain weight for each path. In a matrix it is calculated all distances between all pairs of vertices. It is called the graph distance matrix. In that case it can be called the all-pairs shortest path matrix.

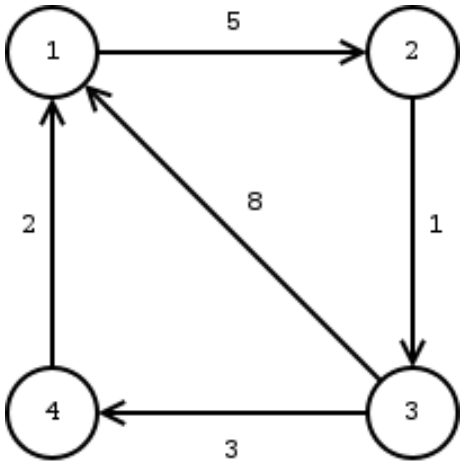


Figure 1.1: Directed graph to find all pairs shortest path

1.2 Basic Definitions and Notations

Before describing some of the relevant algorithms of all pairs shortest path finding problems, we need to introduce and define some of the useful and important notations that will be used throughout this paper.

Definition. A graph G consists of a set of vertices V and a set of edges E.

So it can easily be said that $G = (V,E)$.

Definition. Edges are the connectors between two nodes or vertices.
Vertices of a graph will be represented by lower case English letters possibly with subscripts like the following: v, u_3, v_i. Edges will be represented as the concatenation of the names of the two vertices it connects. For example *vu* would all represent edges in a graph. It should be noted that since we are dealing with directed graphs the order of the concatenation matters a lot, therefore vu is not equal to uv unless their weights are same.

Definition. A graph is **simple** if it has no loop or multiple edges.

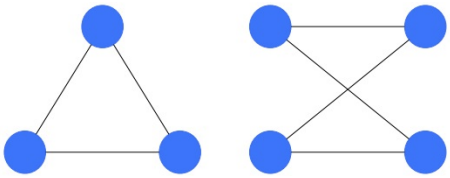


Figure 1.2: Simple graph

Definition. A graph is **directed** graph (or set of nodes connected) if all the edges are directed from one vertex to another.
Definition. A graph is **weighted** if it has certain weight for all edges.

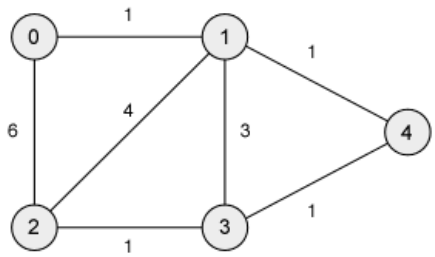


Figure 1.3: Weighted graph

Definition. In **all pairs shortest path** theorem to find the shortest path from one vertex to another it is needed to declare a source vertex *s* to a target vertex *t* in a directed graph. To get the shortest path from *s* to every possible target or from every possible source to *t* by constructing a shortest path tree.
The shortest path tree specifies two information for each node *v* in the graph:
1. dist(*v*) is the length of the shortest path (if any) from *s* to *v*;
2. pred(*v*) is the second-to-last vertex (if any) the shortest path (if any) from *s* to *v*.
The main goal of all pairs shortest path problem is to find the shortest path from every possible source to every possible destination.
Specifically, for every pair of vertices *u* and *v*, it is needed to compute the following information:

1. dist(*u*, *v*) is the length of the shortest path (if any) from *u* to *v*;
2. pred(*u*, *v*) is the second-to-last vertex (if any) on the shortest path (if any) from *u* to *v*.

These are some basic notations and definitions related to all pairs shortest path problem solving theory.

1.3 Literature Review

From history, we acknowledged the FloydWarshall algorithm. It is an example of dynamic programming. It and was published in its currently recognized form by Robert Floyd in 1962. [9]

Theorem 1 [11]: A new algorithm was invented (1991-1992) for the all pairs shortest path problem. Complexity of that invention is O(n^3(log log n/log n)$^{1/2}$) time on a uniform cost RAM. This is approach is an improvement which is basically based on Fredman's result for this problem. This was done by a factor of (log n/log log n)$^{1/6}$. On a **PRAM-EREW**, a parallel algorithm for this problem is also presented.

Theorem 2 [12]: Another approach was tested in 1991. To analyze the scalability of several parallel algorithms, they used the isoefiiency metric. It was done for finding shortest paths between all pairs of nodes in a directed connected graph. It was demonstrated the isoefficiency functions as a compact and useful predictor of performance correctly. The classic trade-offs of hardware cost vs time and memory vs time were found to be represented as trade-offs of hardware cost vs scalability and memory vs scalability.

Theorem 3 [13]: A new approach was detected by giving an algorithm of time $O(n^v log^3 n), v = (3 + \omega)/2$, for the case of edge lengths in {-1, 0, 1}. Here in 1968 the upper bound on the exponent, ω, of matrix multiplication over a ring which was 3. This was reduced for being times.Again 1986 it had been reduced on 2.376. Also,for the all pairs shortest path problem, the exponent of the algorithms stayed at three. These were for the very special case of directed graphs which were with uniform edge lengths. In case of integer edge lengths with absolute value bounded above by M, where v less than 2.688, the time bound is $O((Mn)^v log^3 n)$ and the exponent is less than 3 for M= $O(n^\alpha)$, for $\alpha < 0.116$ and the current bound on ω.

Theorem 4 [10]: Shortest path finding problem is a fundamental problem that has a large amount of applications in areas like computer science, network analysis, network routing

and operations research.
In an approach it was found that by some simple modification with a fast algorithm which use the pre-calculated results to accelerate the latter calculation, the classic Dijkstras algorithm can be improved.
In this experiment the experimental results show that by a factor relating to the connection probability in random networks of ER model, the average running time of this new algorithm is lower than the Dijkstras algorithm.
The time complexity of this approach is reduced to about O($n^{2.4}$) in scalefree complex networks.
At first in 2005 Chan [18] obtained an algorithm which has time complexity O(n^3/ log n). In a directed graph, this is the best-known result for the APSP problem. Using a smaller table, Han [19] proposed an algorithm with the same time complexity later.

Theorem 5 [14]: Another approach gave a tight distributed algorithm for computing the all pairs shortest paths (APSP) when there is given an unweighted and undirected graph, under situation of synchronous communications and the CONGEST (B) model.
Previously the best results for computing all pairs shortest path need O(N + D) time. Here N is the number of nodes and D is the diameter [1, 2].
In order to close the gap happened between the computing result, a multiplexing technique was proposed to push the parallelization of distributed BFS tree constructions to the limit such that the APSP problem can be solved in O(N/B+D) time which meets the lower bound. O(N/B + D) is the time complexities in order to compute these two graph properties.

Theorem 6 [15]: To solve the distance version of all pairs shortest path problem, another approach was invented. It was basically for undirected graph with complexity O(M(n) log n), where M(n) denotes the time necessary to multiply two n $\times$ n matrices of small integers. This is currently known to be o($n^{2.376}$).

Theorem 7 [16]: Besides of computing the all pairs shortest path distance, the Floyd Warshalls Algorithm can be used to detect the presence of negative cycles.
Another technique for finding shortest path based on Floyd Warshalls algorithm which improved time complexity was presented as an $O(n^{3-\epsilon})$ which was O(n^3) before.

Theorem 8 [17]: To solve the problem of finding all pairs shortest path distance, the complexity needed was O(n^2 log n)time. Another approach was made to eliminate the logarithm factor mainly and also obtain the first (slightly) subquadratic algorithm for the problem, with run time O(n^2(log log n/log n)$^{1/2}$).
An implicit representation of all the shortest paths is computed here.

1.4 Problem Definition

Let there be two nodes i and j. Distance from i to j is d_{ij}.Suppose in another path there is another node k in between i and j. Distance from i to k is $d_i^{(k-1)}$ and distance from k to j is $d_j^{(k-1)}$.So considering k in middle of i and j, distance is $d_i^{(k-1)} + d_j(k-1)$. We intended to find minimum path distance of these two nodes.
So, $d_{ij}^k = min\{d_{ij}^k, d_i^{(k-1)} + d_j^{(k-1)}\}$

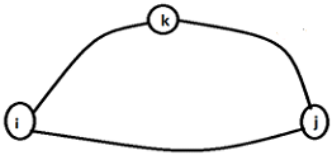


Figure 1.4: Problem definition

1.5 Scopes and objectives of the thesis

Objective of our thesis work is to build an efficient algorithm for solving all pair shortest path problem in a weighted directed graph that will minimize the computation and run time. In finding the shortest path between all the pairs of a directed and weighted graph, it is to declare a source node and a target node at first. Then it is to find the shortest distances from source node to all other nodes. Also similar way from other nodes to a target node. In this way the shortest distance is detected between two nodes.
This problem can be solved by using Floyd Warshall algorithm mainly and efficiently by considering a directed and weighted graph with positive or negative weights but no negative cycle.
All pairs shortest path problem has many real life applications.In networking this approach can be seen as a network node map of a system which is closed. To choose the efficient paths are necessary for data passing. Data usually travel from one point to another suppose point A to point B, but in middle data can appear at any given nodes which are between A and B. Moreover data need to travel to any other node from one node. An analysis of the enitre system, regarding this data passing, can create a great cost efficiency analysis in the whole network to show where the weak or hidden spots in the network may exist and also which path is the shortest and most cost efficient. Also can be detected the easiest path as well.
Not only in networking, it can be used in route searching process as well. So here goal of our

thesis work is to improve the approach to solve the problem in an efficient way to minimize the computation and calculation.

1.6 Thesis Organization

In Chapter 2 related major algorithms of All Pairs Shortest Paths in a graph is discussed. Besides that, existing sufficient conditions and methods are described that helped to establish our new approach.
In chapter 3 the new approach for finding all pairs shortest path theorem that we established is discussed with proof of correctness.We worked with the weighted graph with no negative cycle.
In chapter 4 conclusion and recommendation for future work is described.
Here we tried to derive the findings of our approach and tried to figure out future possibilities of this approach.

CHAPTER 2
RELATED MAJOR ALGORITHMS

2.1 Floyd-Warshall

In the field of computer science, the FloydWarshall algorithm is an algorithm for determination of shortest paths where there is a weighted graph with positive or negative edge weights. But it deals with no negative cycle.
The FloydWarshall algorithm basically works with the comparison of all possible paths through the graph between each pair of vertices. It is able to do this with the complexity of O(n^3) comparisons in a graph. This criteria considers that there may be up to O(n^2) edges in the graph, and testing is done with every combination of edges. It is done by it by improving in an incremental way of an estimation on the shortest path between two vertices. It is done until the estimate is optimal.
The FloydWarshall algorithm is an example of dynamic programming. In 1962, it was published in its currently known form by Robert Floyd. [1] Moreover, it is in an essential way that the same as algorithms which were previously published by Bernard Roy in 1959 [2] and also by Stephen Warshall in 1962 [3]. It was for finding the transitive shutdown of a graph, [4] and is almost related to Kleene's algorithm which was published in 1956 to convert a deterministic finite automaton into a regular expression. [5] Also in 1962 the modern creation of the algorithm as three nested for-loops was first described by Peter Ingerman. [6] This algorithm is known as **Floyd's algorithm**, **the RoyWarshall algorithm**, **the RoyFloyd algorithm**, or **the WFI algorithm**.

2.1.1 Floyd-Warshall Algorithm

The FloydWarshall algorithm compares all possible paths through the graph between each pair of vertices. It is able to do this with O(V^3)comparisons in a graph. This is remarkable considering that there may be up to Ω (V^2) edges in the graph, and every combination of edges is tested. It does so by incrementally improving an estimate on the shortest path between two vertices, until the estimate is optimal.

Consideingr a graph G with vertices V numbered 1 through N. Further consider a function

shortestPath (i,j,k) that returns the shortest possible path from i to j using vertices only from the set {1,2,....,k} as intermediate points along the way. Now, given this function, our goal is to find the shortest path from each i to each j using only vertices in {1,2,....,N}.

For each of these pairs of vertices, the shortestPath (i,j,k) could be either
(1) a path that doesn't go through k (only uses vertices in the set {1,.... ,k-1}.)
or
(2) a path that does go through k (from i to k and then from k to j, both only using intermediate vertices in {1,....,k-1})

We know that the best path from i to j that only uses vertices 1 through k-1 is defined by shortestPath (i,j,k-1), and it is clear that if there were a better path from i to k to j, then the length of this path would be the concatenation of the shortest path from i to k (only using intermediate vertices in {1,....,k-1}) and the shortest path from k to j (only using intermediate vertices in {1,...,k-1}).

If w(i,j) is the weight of the edge between vertices i and j, we can define shortestPath (i,j,k) in terms of the following recursive formula: the base case is
shortestPath (i,j,0)=w(i,j)
and the recursive case is
shortestPath (i,j,k)= min(shortestPath (i,j,k-1),shortestPath (i,k,k-1)+shortestPath (k,j,k-1)).
[20]
This formula is the heart of the FloydWarshall algorithm. The algorithm works by first computing shortestPath (i,j,k) for all (i,j) pairs for k=1, then k=2, etc. This process continues until k=N, and we have found the shortest path for all (i,j) pairs using any intermediate vertices.

Here is the basic version of Floyd-Warshall algorithm:

Algorithm 1 Floyd-Warshall Algorithm

1.**let** dist be a V $\times$V array of minimum distances initialized to $\infty(infinity)$
2. **for** each vertex v
3. dist[v][v] $\leftarrow 0$
4. **for** each edge (u,v)
5. dist[u][v] $\leftarrow$ w(u,v) // the weight of the edge (u,v)
6 . **for** k from 1 to V
7. **for** i from 1 to V
8. **for** j from 1 to V
9 . **if** dist[i][j] >dist[i][k] + dist[k][j]
10. dist[i][j] $\leftarrow dist[i][k] + dist[k][j]$
11. **end if**

2.1.2 Floyd-Warshall Complexity

Let n be V, the number of vertices. To find all O(n^2) of shortestPath (i,j,k) (for all i and j) from those of shortestPath (i,j,k-1) requires $(2n^2)$ operations. Since it begins with shortestPath (i,j,0)= edgeCost (i,j) and compute the sequence of n matrices shortestPath (i,j,1),shortestPath (i,j,2),....., shortestPath (i,j,n), the total number of operations used is n. $2n^2 = 2n^3$.

Therefore, the complexity of the algorithm is Θ (n^3).1001[20]

2.1.3 Floyd-Warshall Example

The algorithm above is executed on the graph on the left below:

Prior to the first iteration of the outer loop, labeled k = 0 above, the only known paths

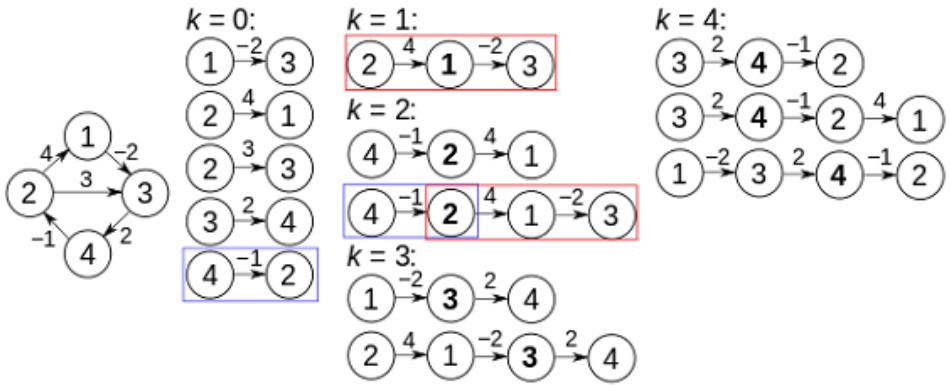


Figure 2.1: Example graph of Floyd-Warshall Algorithm

correspond to the single edges in the graph. At k = 1, paths that go through the vertex 1

are found: in particular, the path [2,1,3] is found, replacing the path [2,3] which has fewer edges but is longer (in terms of weight). At k = 2, paths going through the vertices {1,2} are found. The red and blue boxes show how the path [4,2,1,3] is assembled from the two known paths [4,2] and [2,1,3] encountered in previous iterations, with 2 in the intersection. The path [4,2,3] is not considered, because [2,1,3] is the shortest path encountered so far from 2 to 3. At k = 3, paths going through the vertices {1,2,3} are found. Finally, at k = 4, all shortest paths are found.

The distance matrix at each iteration of k, with the updated distances in **bold**, will be:

$k=0$		j			
		1	**2**	**3**	**4**
i	**1**	0	∞	−2	∞
	2	4	0	3	∞
	3	∞	∞	0	2
	4	∞	−1	∞	0

$k=1$		j			
		1	**2**	**3**	**4**
i	**1**	0	∞	−2	∞
	2	4	0	**2**	∞
	3	∞	∞	0	2
	4	∞	−1	∞	0

Figure 2.2: Example matrix of Floyd-Warshall Algorithm (k=0,1)

$k=2$		j			
		1	**2**	**3**	**4**
i	**1**	0	∞	−2	∞
	2	4	0	2	∞
	3	∞	∞	0	2
	4	**3**	−1	**1**	0

$k=3$		j			
		1	**2**	**3**	**4**
i	**1**	0	∞	−2	**0**
	2	4	0	2	**4**
	3	∞	∞	0	2
	4	3	−1	1	0

$k=4$		j			
		1	**2**	**3**	**4**
i	**1**	0	**−1**	−2	0
	2	4	0	2	4
	3	**5**	**1**	0	2
	4	3	−1	1	0

Figure 2.3: Example matrix of Floyd-Warshall Algorithm (k=2,3,4)

A negative cycle is a cycle whose edges sum to a negative value. There is no shortest path between any pair of vertices i,j which form part of a negative cycle, because path-lengths from i to j can be arbitrarily small (negative). For numerically meaningful output, the FloydWarshall algorithm assumes that there are no negative cycles. Nevertheless, if there are negative cycles, the FloydWarshall algorithm can be used to detect them. The intuition is,
1.The FloydWarshall algorithm iteratively revises path lengths between all pairs of vertices (i,j), including where i=j;
2.Initially, the length of the path (i,i) is zero;
3.A path [i,k,.....,i] can only improve upon this if it has length less than zero, i.e. denotes a negative cycle;
4.Thus, after the algorithm, (i,i) will be negative if there exists a negative-length path from i back to i.
Hence, to detect negative cycles using the FloydWarshall algorithm, one can inspect the diagonal of the path matrix, and the presence of a negative number indicates that the graph contains at least one negative cycle. [7] To avoid numerical problems one should check for negative numbers on the diagonal of the path matrix within the inner for loop of the algorithm. [8] Obviously, in an undirected graph a negative edge creates a negative cycle (i.e., a closed walk) involving its incident vertices. Considering all edges of the above example graph as undirected, e.g. the vertex sequence 4 2 4 is a cycle with weight sum 2. [20]
But we are not considering any negative cycle here.

2.2 Johnson's Algorithm

When there is a situation that the problem is to find shortest paths between every pair of vertices in a given weighted directed Graph and weights may be negative. Already it has been discussed Floyd Warshall Algorithm for this problem. Time complexity of Floyd Warshall Algorithm is O(V^3). Using Johnsons algorithm, all pair shortest paths solution can be found in O(V^2log V + VE) time. Johnsons algorithm uses both Dijkstra and Bellman-Ford as subroutines.

If it is applied the Dijkstras Single Source shortest path algorithm for every vertex, considering every vertex as source, it can be found all pair shortest paths in O(V*VLogV) time. So using Dijkstras single source shortest path seems to be a better option than Floyd Warshell, but the problem with Dijkstras algorithm is, it doesnt work for negative weight edge. The idea of Johnsons algorithm is to re-weight all edges and make them all positive, then apply Dijkstras algorithm for every vertex.

2.2.1 Algorithm

Here is the algorithm:

Algorithm 2 Johnson's Algorithm

---

1. Let the given graph be **G**. Addition of a new vertex **s** to the graph, addion of edges from new vertex to all vertices of **G**. Let the modified graph be **G**.

2. Run of Bellman-Ford algorithm on **G** with **s** as source. Let the distances calculated by Bellman-Ford be h[0], h[1], .. h[V-1]. If a negative weight cycle is found, then return.
Note that the negative weight cycle cannot be created by new vertex **s** as there is no edge to **s**. All edges are from **s**.

3. Reweight the edges of original graph. For each edge (u, v), assign the new weight as original weight + h[u] h[v].

4. Remove the added vertex s and run Dijkstras algorithm for every vertex.

When it is for the following property, this is always true about h[] values as they are shortest distances.

h[v] <= h[u] + w(u, v)
The property means, shortest distance from s to v must be smaller than or equal to shortest distance from s to u plus weight of edge (u, v). The new weights are w(u, v) + h[u] - h[v]. The value of the new weights must be greater than or equal to zero because of the inequality "h[v] <= h[u] + w(u, v)". [21]

2.2.2 Complexity

Here Bellman Ford Algorithm is called once and Dijkstra called V times. These are the main steps. Moreover, time complexity of Bellman Ford is O(VE) and time complexity of Dijkstra is O(VLogV). So overall time complexity is O(V^2log V + VE). [21]
When the graphs is complete, the time complexity of Johnson's algorithm becomes same as Floyd Warshell algorithm. But the algorithm performs much better than Floyd Warshell algorithm for sparse graphs.

2.3 Min Plus & Matrix Multiplication

Min-plus matrix multiplication algorithm is the process which is also known as the **distance product**. This is basically an operation on matrices.

Given two **n** $\times$**n** matrices **A**=(a_{ij}) and **B**=(b_{ij}), their distance product C=(c_{ij}) $= A\star$ B is defined as an n$\times$ n matrix such that $c_{ij} = \min_{k=1}^{n}\{a_{ik} + b_{kj}\}$. This is standard matrix multiplication for the semi-ring of tropical numbers in the min convention.

The operation of min plus matrix multiplication is nearly related to the shortest path problem. If **W** is an **n** $\times$**n** matrix containing the edge weights of a graph, then W^k gives the distances between vertices using paths of length at most **k** edges, and W^n is the distance matrix of the graph.

CHAPTER 3
A NEW APPROACH FOR ALL PAIRS SHORTEST PATH

3.1 Overview of the Theorem and Algorithm

We have tried to find a new approach to detect the shortest paths of all nodes but selecting path between a source and a target node at one time in a weighted graph.

3.1.1 Overview of the Theorem

Floyd-Warshall algorithm [1] uses all the vertices of a graph in each iteration. So, the run time of Floyd is $\mathcal{O}(n^3)$ in the best case and the worst case. One of the most important thing is that the run time does not change depending on the input. So we thought of a new approach which considers this and the run time of our new approach is $\mathcal{O}(n^2 \log n)$.

The new algorithm limits the vertices to be explored. Firstly, the source finds the nearest vertex with the lowest cost and search whether there is a path to the destination or not. Same thing is done for the destination. So we get two vertices- one is the nearest from the source and another is the nearest from the destination. So, this becomes the limit and next the path from source to destination will be updated for the vertices in this limit and only if the cost of the path is less than the previous path. The iteration will occur $\log(n)$ times to find the path up to n length.

the algorithm is illustrated in 4 stages to clear the concept. Firstly, a simple parallel algorithm and run time analysis is given and then the correctness proof has been shown and then examples are given with scenarios. After that the result and time complexity of the proposed algorithm is compared with Floyd-Warshall.

3.1.2 Algorithm

Here is the algorithm of our approach:

Algorithm 3 Our Algorithm with new approach

```
1. Let Ri be an array of indices to which distance of i is in increasing order
2. Let Ci be an array of indices to which distance of j is in increasing order
3. for L=1 to log2n
4.         for i = 1 to n
5.                 for j= 1 to n
6.                         if (i==j)
7.                                 continue
8.                         end
9.                         p=R(i,2);
10.                        q=C(j,2);
11.                        a=graph(i,p)+graph (p,j)
12.                        b=graph(i,q)+graph(q,j)
13.                        graph (i,j)=min (a,b,graph(i,j))
14.                        for k=3 to n
15.                                if graph(i,q)≤(i,R(i,k))
16.                                        break
17.                                end
18.                                d=graph(i,R(i,k))+graph (R(i,k),j);
19.                                if (d < graph(i,j))
20.                                        graph(i,j)=d
21.                                end
22.                        end

23.                        for k=3 to n
24.                                if graph(p,j)≤graph (C(j,k),j)
25.                                        break
26.                                end
27.                                d=graph(i,C(j,k))+graph (C(j,k),j);
28.                                if (d < graph(i,j))
29.                                        graph(i,j)=d
30.                                end
31.                        end
32.                end
33.        end
34.end
```

3.1.3 Run time Analysis

Line 1 sorts the row of the matrix in increasing order and line 2 sorts the column in increasing order. So, line 1-2 need $n^2 \log n$ times. Line 3 iterates $\log_2 n + 1$ times. Line 4 iterates $(n+1)(n^2 \log n)$ times. Line 5 iterates $n(n+1) \log n$ times. Line 6- 13 need $n^2 \log n$ iterations. Line 14 iterates $(m+1)n^2 \log n$ times. Line 15-22 iterate $mn^2 \log n$ where in worst case $m = n - 2$. Line 23 iterates $(l+1)n^2 \log n$ times. Line 24-31 iterate $ln^2 \log n$

where in worst case $l = n - 2$. Line 32,33 and 34 need $n^2 \log n$, $n \log n$ and $\log n$ times respectively. Let, the cost of line 1 to 34 are $c_1, c_2, c_3 \ldots\ldots c_{34}$ respectively.

Total run time =$(c_1 + c_2)n^2 \log n + c_3(\log n + 1) + c_4(n + 1) \log n + c_5 n(n + 1) \log n + An^2 \log n + c_{14}(m+1)n^2 \log n + Bmn^2 \log n + c_{23}(l+1)n^2 \log n + Cln^2 \log n + c_{32}n^2 \log n + c_{33}n \log n + c_{34} \log n$
= $A_1 n^2 \log n + B_1 n \log n + D_1 mn^2 \log n + E_1 ln^2 \log n + c_3$
= $\mathcal{O}(mn^2 \log n + ln^2 \log n)$
= $\mathcal{O}((m + l)n^2 \log n)$
Here, $A = c_7 + c_8 + ... + c_{13}$
$B = c_{15} + c_{16} + ... + c_{22}$
$C = c_{24} + c_{25} + ... + c_{31}$
$A_1 = A + c_1 + c_2 + c_5 + c_{23} + c_{32}$
$B_1 = c_4 + c_5 + c_{33}$
$D_1 = c_{14} + B$
$E_1 = c_{23} + C$
$F_1 = c_3 + c_4 + c_{34}$

3.2 Correctness

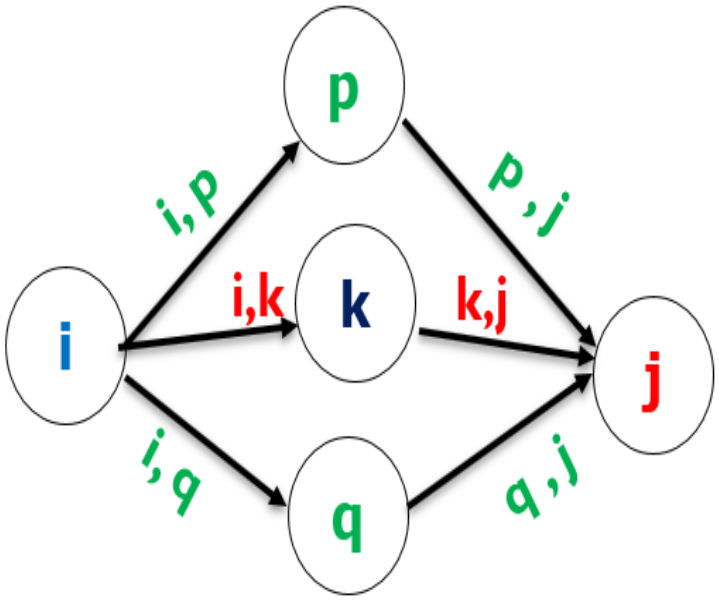


Figure 3.1: A directed graph G where i is the source and j is the destination

Let, there be a path from i to j through k,

$$d(i, j)_{\text{through k}} = d(i, k) + d(k, j)$$

where d(i,j)is the path from i to j through vertex k.
Let, p be the nearest vertex of i,

$$d(i,j)_{\text{through p}} = d(i,p) + d(p,j)$$

where d(i,j)is the path from i to j through vertex p.

if $d(k,j) > d(p,j)$,then $d(i,k) + d(k,j) > d(i,p) + d(p,j)$.
as $d(i,p)$ is minimum and so $d(i,k) > d(i,p)$.

Let, q be the nearest vertex of j,

$$d(i,j)_{\text{through q}} = d(i,q) + d(q,j)$$

where d(i,j)is the path from i to j through vertex q.

if $d(i,k) > d(i,q)$,then $d(i,k) + d(k,j) > d(i,q) + d(q,j)$.
as $d(q,j)$ is minimum and so $d(k,j) > d(q,j)$. The path from i to j through k can be the shortest path either
The path from i to j through k can be the shortest path. If either

$$d(i,p) < d(i,k) < d(i,q)$$

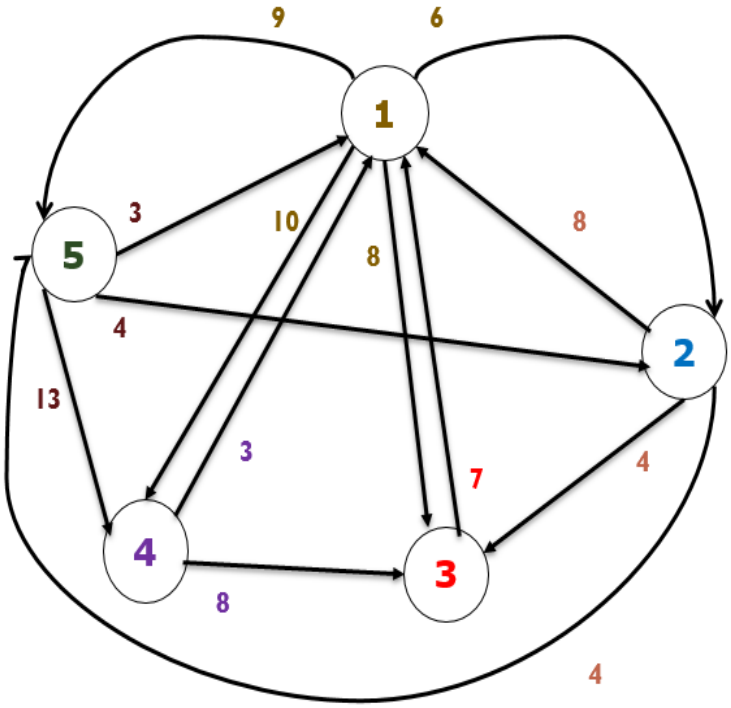
or,

$$d(q,j) < d(k,j) < d(i,q)$$

conditions are satisfied.
$d[i,j]$ becomes the best of four alternatives- its old value or the $d[i,j]_{\text{though p}}$ or the $d[i,j]_{\text{though k}}$ or the $d[i,j]_{\text{though q}}$. The outer loop will update in each iteration up to 2^l where, $l = 1$ to $\log_2 n$. This shows the correctness of our algorithm.

3.3 Examples and Scenarios

According to our new approach first we have to sort indices for each row and each column separately to get the sorted row matrix R and sorted column matrix C. For the graph



$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & 6 & 8 & 10 & 9 \\ 8 & 0 & 4 & \infty & 4 \\ 7 & \infty & 0 & \infty & \infty \\ 3 & \infty & 8 & 0 & \infty \\ 3 & 4 & \infty & 13 & 0 \end{pmatrix} \end{array}$$

Figure 3.2: A directed graph G having 5 vertices and 13 edges and the adjacency-matrix representation of G.

mentioned in figure 3.2 we get,

$$\text{Sorted Row, } R = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 1 & 2 & 3 & 5 & 4 \\ 2 & 3 & 5 & 1 & 4 \\ 3 & 1 & 2 & 4 & 5 \\ 4 & 1 & 3 & 2 & 5 \\ 5 & 1 & 2 & 4 & 3 \end{pmatrix} \end{array}$$

First row of matrix R represents an array of indices to which distance of vertex 1 is in increasing order. That means, the nearest vertex from 1 is 2 (first column is negligible as distance node 1 to node 1 is not considered) the second nearest vertex 3 third nearest vertex

is 5 and so on.

$$\text{Sorted Column, } C = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 1 & 4 & 5 & 3 & 2 \\ 2 & 5 & 1 & 3 & 4 \\ 3 & 2 & 1 & 4 & 5 \\ 4 & 1 & 5 & 2 & 3 \\ 5 & 2 & 1 & 3 & 4 \end{pmatrix} \end{array}$$

$$\text{Output} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & 6 & 8 & 10 & 9 \\ 7 & 0 & 4 & 17 & 4 \\ 7 & 13 & 0 & 17 & 16 \\ 3 & 9 & 8 & 0 & 12 \\ 3 & 4 & 8 & 13 & 0 \end{pmatrix} \end{array}$$

Output matrix is computed by our new approach to calculating all pairs shortest path algorithm 3.

3.3.1 Example 1

Suppose, source is vertex 2 and destination is vertex 3. From sorted row matrix, R we get the nearest vertex of 2 is vertex 3 and from sorted column we get the nearest vertex of 3 is 2. In graph 3.2 the distance between 2 and 3 is 4. So in output matrix the shortest between vertex 2 and 3 is 4.

$$\text{Sorted Row, } R = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 1 & 2 & 3 & 5 & 4 \\ 2 & 3 & 5 & 1 & 4 \\ 3 & 1 & 2 & 4 & 5 \\ 4 & 1 & 3 & 2 & 5 \\ 5 & 1 & 2 & 4 & 3 \end{pmatrix} \end{array}$$

$$\text{Sorted Column, } C = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 1 & 4 & 5 & 3 & 2 \\ 2 & 5 & 1 & 3 & 4 \\ 3 & 2 & 1 & 4 & 5 \\ 4 & 1 & 5 & 2 & 3 \\ 5 & 2 & 1 & 3 & 4 \end{pmatrix} \end{array}$$

$$\text{Output} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & 6 & 8 & 10 & 9 \\ 7 & 0 & 4 & 17 & 4 \\ 7 & 13 & 0 & 17 & 16 \\ 3 & 9 & 8 & 0 & 12 \\ 3 & 4 & 8 & 13 & 0 \end{pmatrix} \end{array}$$

3.3.2 Example 2

Suppose, source is vertex 3 and destination is vertex 4. From sorted row matrix, R we get the nearest vertex of 3 is vertex 1 and from sorted column matrix C we get the nearest vertex of 4 is 1. There is no direct path from vertex 3 to vertex 4 in graph 3.2. But there is a path via vertex 1.
Distance from vertex 3 to vertex 1 = 7
Distance from vertex 1 to vertex 4 = 10
Distance from vertex 3 to vertex 4 = $10 + 7 = 17$

So, output matrix is updated from infinity to 17.

$$\text{Sorted Row, } R = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 1 & 2 & 3 & 5 & 4 \\ 2 & 3 & 5 & 1 & 4 \\ 3 & 1 & 2 & 4 & 5 \\ 4 & 1 & 3 & 2 & 5 \\ 5 & 1 & 2 & 4 & 3 \end{pmatrix} \end{array}$$

$$\text{Sorted Column, } C = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 1 & 4 & 5 & 3 & 2 \\ 2 & 5 & 1 & 3 & 4 \\ 3 & 2 & 1 & 4 & 5 \\ 4 & 1 & 5 & 2 & 3 \\ 5 & 2 & 1 & 3 & 4 \end{pmatrix} \end{array}$$

$$\text{Input} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & 6 & 8 & 10 & 9 \\ 8 & 0 & 4 & \infty & 4 \\ 7 & \infty & 0 & \infty & \infty \\ 3 & \infty & 8 & 0 & \infty \\ 3 & 4 & \infty & 13 & 0 \end{pmatrix} \end{array} \text{Output} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & 6 & 8 & 10 & 9 \\ 7 & 0 & 4 & 17 & 4 \\ 7 & 13 & 0 & 17 & 16 \\ 3 & 9 & 8 & 0 & 12 \\ 3 & 4 & 8 & 13 & 0 \end{pmatrix} \end{array}$$

3.3.3 Example 3

Suppose, source is vertex 1 and destination is vertex 2. From sorted row matrix, R we get the nearest vertex of 1 is vertex 2 and from sorted column matrix C we get the nearest vertex of 2 is 5.There is a direct path from vertex 1 to vertex 2 in graph 3.2. But there is also a path via vertex 5.

Distance from vertex 1 to vertex 2 = 6

Distance from vertex 1 to vertex 4 via vertex 5 = $9 + 4 = 13$

So in output matrix the shortest between vertex 1 and 2 is 6.

$$\text{Sorted Row, } R = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 1 & 2 & 3 & 5 & 4 \\ 2 & 3 & 5 & 1 & 4 \\ 3 & 1 & 2 & 4 & 5 \\ 4 & 1 & 3 & 2 & 5 \\ 5 & 1 & 2 & 4 & 3 \end{pmatrix} \end{array}$$

$$\text{Sorted Column, } C = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 1 & 4 & 5 & 3 & 2 \\ 2 & 5 & 1 & 3 & 4 \\ 3 & 2 & 1 & 4 & 5 \\ 4 & 1 & 5 & 2 & 3 \\ 5 & 2 & 1 & 3 & 4 \end{pmatrix} \end{array}$$

$$\text{Output} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & 6 & 8 & 10 & 9 \\ 7 & 0 & 4 & 17 & 4 \\ 7 & 13 & 0 & 17 & 16 \\ 3 & 9 & 8 & 0 & 12 \\ 3 & 4 & 8 & 13 & 0 \end{pmatrix} \end{array}$$

3.3.4 Example 4

Let, source is vertex 2 and destination is vertex 4. From sorted row matrix, R we get the nearest vertex of 2 is vertex 3 and from sorted column matrix C we get the nearest vertex of 4 is 1. There is no direct path from vertex 2 to vertex 4 in graph 3.2.

$$\text{Sorted Row, } R = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 1 & 2 & 3 & 5 & 4 \\ 2 & 3 & 5 & 1 & 4 \\ 3 & 1 & 2 & 4 & 5 \\ 4 & 1 & 3 & 2 & 5 \\ 5 & 1 & 2 & 4 & 3 \end{pmatrix} \end{array}$$

$$\text{Sorted Column, } C = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 1 & 4 & 5 & 3 & 2 \\ 2 & 5 & 1 & 3 & 4 \\ 3 & 2 & 1 & 4 & 5 \\ 4 & 1 & 5 & 2 & 3 \\ 5 & 2 & 1 & 3 & 4 \end{pmatrix} \end{array}$$

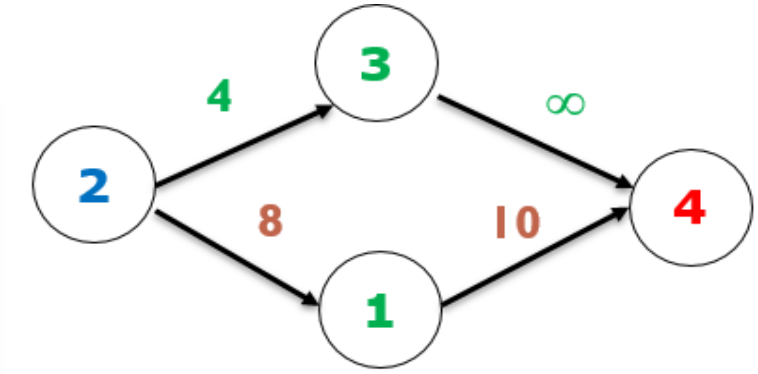


Figure 3.3: Path from vertex 3 to vertex 4 via vertex 1

$d_{24via1} = d_{21} + d_{14} = 8 + 10 = 18$. According to our algorithm, it will be checked if there is any vertex (say k) for which $d_{23} <= d_{2k} <= d_{2_1}$ or $d_{14} <= d_{k4} <= d_{34}$. To find that k check the second nearest vertex from vertex 2 in sorted row matrix which is 5. Also, check the vertex from which the distance is nearest to vertex 4. For this check the second nearest vertex from vertex 4 in sorted column matrix which is also 5.

$$\text{Sorted Row, } R = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 1 & 2 & 3 & 5 & 4 \\ 2 & 3 & 5 & 1 & 4 \\ 3 & 1 & 2 & 4 & 5 \\ 4 & 1 & 3 & 2 & 5 \\ 5 & 1 & 2 & 4 & 3 \end{pmatrix} \end{array}$$

$$\text{Sorted Column, } C = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 1 & 4 & 5 & 3 & 2 \\ 2 & 5 & 1 & 3 & 4 \\ 3 & 2 & 1 & 4 & 5 \\ 4 & 1 & 5 & 2 & 3 \\ 5 & 2 & 1 & 3 & 4 \end{pmatrix} \end{array}$$

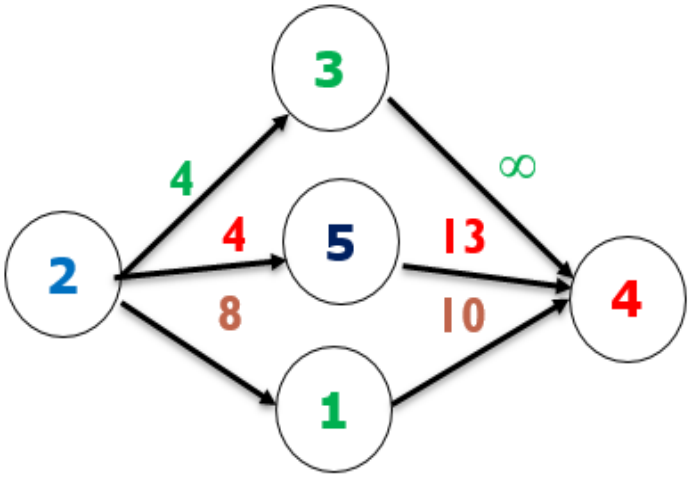


Figure 3.4: Path from vertex 3 to vertex 4 via vertex 5

$d_{23} <= d_{25} <= d_{2_1} = 17$

It is minimum. So, output matrix is updated from infinity to 17.

$$\text{Input} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & 6 & 8 & 10 & 9 \\ 8 & 0 & 4 & \infty & 4 \\ 7 & \infty & 0 & \infty & \infty \\ 3 & \infty & 8 & 0 & \infty \\ 3 & 4 & \infty & 13 & 0 \end{pmatrix} \end{array} \text{Output} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & 6 & 8 & 10 & 9 \\ 7 & 0 & 4 & 17 & 4 \\ 7 & 13 & 0 & 17 & 16 \\ 3 & 9 & 8 & 0 & 12 \\ 3 & 4 & 8 & 13 & 0 \end{pmatrix} \end{array}$$

3.4 Experimental Data

We have tested our proposed algorithm with various directed graphs. Code has been written in Matlab. Experimental results are shown in following table 3.1:

Table 3.1: Experimental results for various graphs

Serial	Size of Matrix	Output matched with Floyd Warshall
1.	5×5	Matched
2.	10×10	Matched
3.	20×20	Matched
4.	50×50	Matched
5.	100×100	Matched
6.	500×500	Matched

3.5 Comparison with Floyd Warshall Algorithm

The time complexity of the Floyd-Warshall algorithm $\mathcal{O}(n^3)$ and space complexity is $\mathcal{O}(n^2)$ [1]. On the other hand, the time complexity of our new approach is $\mathcal{O}((m+l)n^2 \log n)$ and space complexity is $\mathcal{O}(n^2)$. In best case, its order is $\mathcal{O}(n^2 \log n)$ where, $m+l$ becomes 1 because none of the inner loop are executed and for Floyd-Warshall, order remains same.

A graph is plotted considering the time complexity and number of vertices which are $y-axis$ and $x-axis$ respectively. In the graph, Floyd-Warshall and the best case of the proposed algorithm are compared.

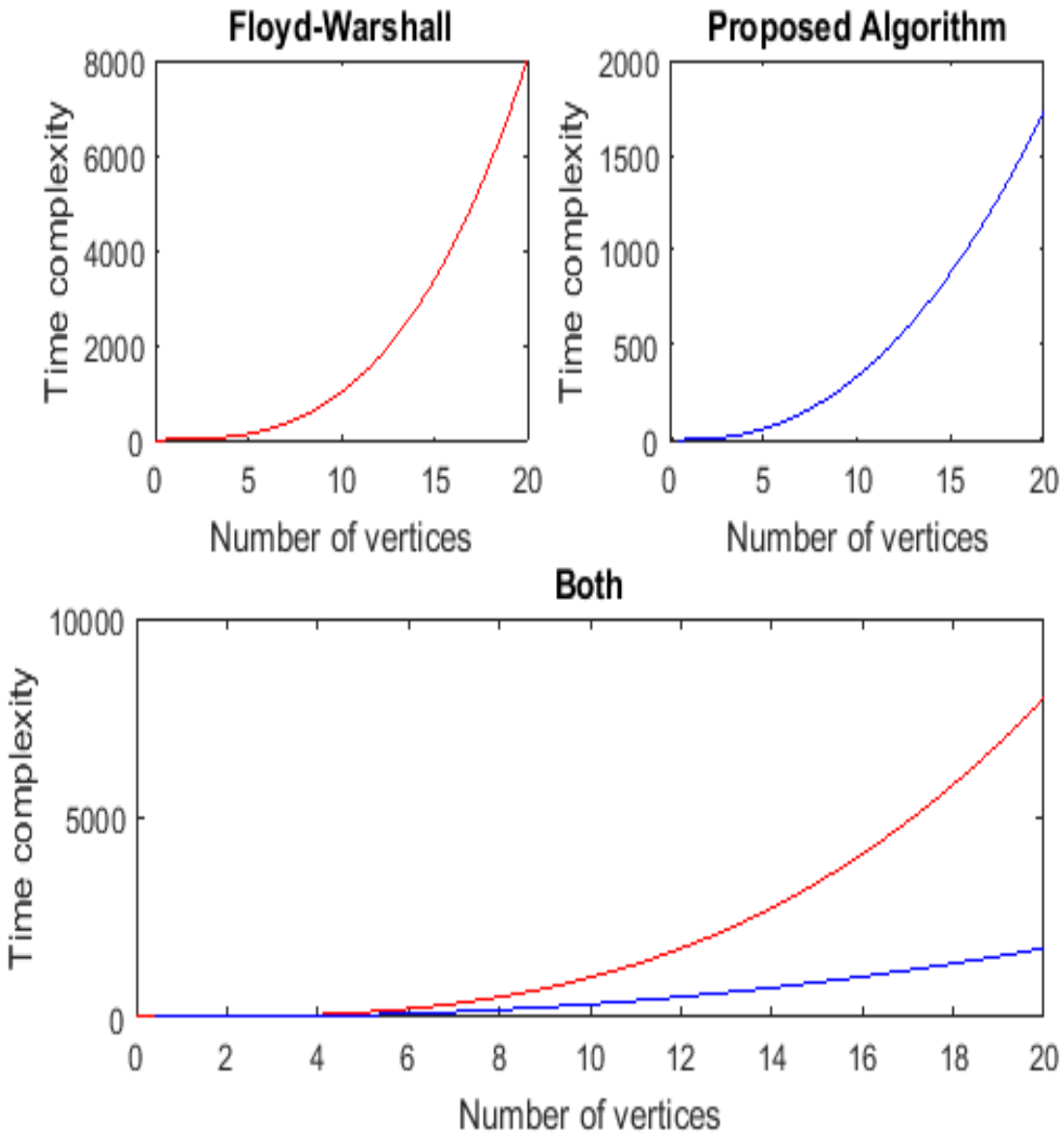


Figure 3.5: A comparison between Floyd-Warshall and the proposed algorithm (20×20)

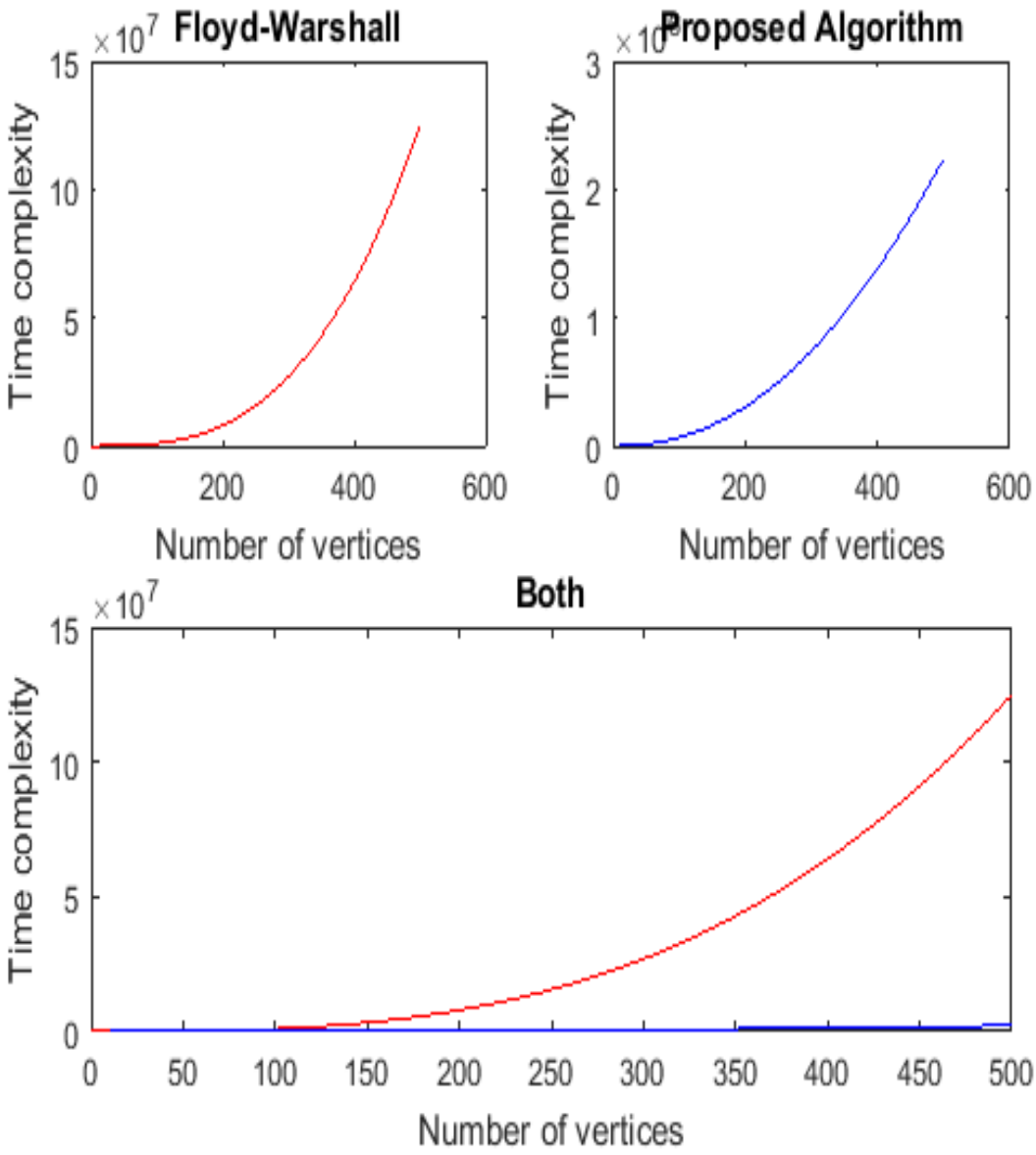


Figure 3.6: A comparison between Floyd-Warshall and the proposed algorithm (500×500)

CHAPTER 4
CONCLUSION AND RECOMMENDATION

4.1 Conclusion

All pairs shortest path is an important topic in the area of graph theory,algorithm and complexity theory. Also we can say, all pair shortest path is widely discussed topic and if it is possible to improve it by minimizing the run time it can be of great use. So, we are tried to improve it using different approach. We tried to improve Floyd-Warshall algorithm, MST, least common ancestor and tried to establish new algorithm. These approaches didn't bring us success. And then we are tried another approach to make a solution of that problem. Our goal of the research is to detect a new condition of finding shortest path between all pairs in a weighted graph. In chapter 1, we have seen the overview and history of all pairs shortest path , some basic definition and notations , scope and objective of the thesis. In chapter 2, we have seen some major algorithms of all pairs shortest path and their complexities. In the chapter 3, we propose a new condition of finding all pairs shortest path between all nodes of a weighted graph. So, the theorem become more efficient than before in best cases.

4.2 Recommendation

Here we have tried to find a new approach to find the shortest path of all pairs in a weighted graph. We have tried to check the theorem for $4\times4, 5 \times 5, 10 \times 10, 500 \times 500$ and so on matrices. We have used the row and column sorting formula here. If it is possible to minimize the sorting part or replace that part with another similar approach with less time complexity, then the condition will be more efficient. In future we will try to increase the efficiency of the theorem of our new approach of finding all pairs shortest path by decreasing the sorting time.

REFERENCES

[1] Floyd, Robert W. (June 1962). ”Algorithm 97: Shortest Path”. Communications of the ACM. 5 (6): 345. doi:10.1145/367766.368168.

[2] Roy, Bernard (1959). ”Transitivit et connexit”. C. R. Acad. Sci. Paris. 249: 216218.

[3] Warshall, Stephen (January 1962). ”A theorem on Boolean matrices”. Journal of the ACM. 9 (1): 1112. doi:10.1145/321105.321107.

[4] Weisstein, Eric W. ”Floyd-Warshall Algorithm”. MathWorld.

[5] Kleene, S. C. (1956). ”Representation of events in nerve nets and finite automata”. In C. E. Shannon and J. McCarthy. Automata Studies. Princeton University Press. pp. 342.

[6] Ingerman, Peter Z. (November 1962). ”Algorithm 141: Path Matrix”. Communications of the ACM. 5 (11): 556. doi:10.1145/368996.369016.

[7] Hochbaum, Dorit (2014). ”Section 8.9: Floyd-Warshall algorithm for all pairs shortest paths” (PDF). Lecture Notes for IEOR 266: Graph Algorithms and Network Flows. Department of Industrial Engineering and Operations Research, University of California, Berkeley.

[8] Stefan Hougardy (April 2010). ”The FloydWarshall algorithm on graphs with negative cycles”. Information Processing Letters. 110 (8-9): 279281. doi:10.1016/j.ipl.2010.02.001.

[9] Floyd, Robert W. (June 1962). ”Algorithm 97: Shortest Path”. Communications of the ACM. 5 (6): 345. doi:10.1145/367766.368168.

[10] Wei Peng, Xiaofeng Hu, Feng Zhao, Jinshu Su. *”A Fast Algorithm to Find All-Pairs Shortest Paths in Complex Networks”*. https://doi.org/10.1016/j.procs.2012.04.060

[11] Tadao Takaoka. *”A new upper bound on the complexity of the all pairs shortest path problem”*. https://doi.org/10.1016/0020-0190(92)90200-F

[12] Vipin Kumar, Vineet Singh. *”Scalability of Parallel Algorithms for the All-Pairs Shortest-Path Problem”*. https://doi.org/10.1016/0743-7315(91)90083-L

[13] Noga Alon, Zvi Galil, Oded Margalit. *”On the Exponent of the All Pairs Shortest Path Problem”*. https://doi.org/10.1006/jcss.1997.1388

[14] Qiang-Sheng Hua,Haoqiang Fan,Lixiang Qian, Ming Ai, Yangyang Li, Xuanhua Shi, Hai Jin. *"Brief Announcement: A Tight Distributed Algorithm for All Pairs Shortest Paths and Applications"*. https://doi.org/10.1145/2935764.2935812

[15] Seidel. R. *"On the All-Pairs-Shortest-Path Problem in Unweighted Undirected Graphs"*. https://doi.org/10.1006/jcss.1995.1078

[16] Himanshu Garg, Paramjeet Rawat. *"An Improved Algorithm for Finding All Pair Shortest Path"*. https://doi.org/10.5120/7539-0492

[17] Timothy M. Chan, Dimitrios Skrepetos. *"All-Pairs Shortest Paths in Unit-Disk Graphs in Slightly Subquadratic Time"*. https://doi.org/10.4230/LIPIcs.ISAAC.2016.24

[18] T. M. Chan. All-pairs shortest paths with real weights in O(n^3/ log n) time. in: Proc. 9th Workshop Algorithms Data Structures, in: Lecture Notes in Computer Science, vol.3608, Springer, Berlin, 2005, pp.318-324.

[19] Y. Han. A note of an O(n^3/ log n) time algorithm for all pairs shortest paths. Information Processing Letters, vol.105, no.3, pp. 114-116, 2008.

[20] https://en.wikipedia.org/wiki/Floyd

[21] https://www.geeksforgeeks.org/johnsons-algorithm/

APPENDIX A
Thesis.sh File

APPENDIX B
code_mssql.rc File

APPENDIX C
code_tomcat.rc File

APPENDIX D
code_autopwn.rc File