International Journal of Advanced Computer Technology (IJACT)

ISSN:2319-7900

An Efficient algorithm to compute all pairs shortest path using DNA sequence

P. Swarnambigai[1], Research Scholar, Dr Ambedkar Govt. Arts College, Vyasarpadi, Chennai-39;
Dr A. Murugan[2], Associate Professor, Dr Ambedkar Govt. Arts College, Vyasarpadi, Chennai-39

Abstract

In this paper, we present an algorithm to compute all pairs optimized shortest paths using DNA sequencing [19]. Researchers have given many approaches for finding all pair shortest path problem but the proposed algorithm is used to reduce time complexity to compute shortest path. In the existing algorithm takes $O(n^3)$ time to find the path and in the proposed algorithm, it will take n!(n-r)! /r! where, *n* denotes the number of nodes and *r* denotes the intermediate node during the calculation of shortest path between nodes. Floyd Warshall's Algorithm is used to compare to the proposed algorithm. This concept is implemented in the DNA sequencing concept of Bioinformatics [18].

Keywords: Floyd Warshall's Algorithm, DNA Sequencing, Bioinformatics

1. Introduction

The shortest path problem [4] is the problem of finding path between two vertices (or nodes) in a graph, such that the sum of the weights of its constituent edges is minimized. An example of it can be, finding the quickest way to get from one location to another on a road map. In this case, the vertices represent locations and the edges represent segments of road and are weighted by the time needed to travel to that location. The single source shortest path is one of the oldest classical problems in algorithm theory. Given a positively weighted directed graph 'G', with a source vertex s, this problem ask for finding the shortest path from 'S' to all other vertices. So it can be considered the mother of all routing problems. Given a weighted directed graph G = (V, E) with two special vertices, source's' and a target 't', and the problem is to find the shortest directed path from's' to't'. In other words, we have to find the path 'P' starting at's' and ending at 't' minimizing the function:

$$w(p) = \sum_{e \in p} w(e)$$

Specifically, for every pair of vertices 'u' and 'v', we need to compute the following information:

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

Given a weighted digraph with a weight function w : E→ R, R is the set of real numbers that determine the length of the shortest path between all pair of vertices in G. Given an input, n*n matrix, 'W' represent the edge weights of n vertices; i.e., W= (w_{ij}), where

$$w_{ij} = \begin{cases} 0 & if\ (i = j) \\ w(i,j)\ if\ (i \neq j)\ and\ (i,j) \in E \\ \infty & if\ (i \neq j)\ and\ (i,j) \notin E \end{cases}$$

2. Literature Survey

The algorithm compares all possible paths through the graph between each pair of vertices. Consider 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 1 to *k* + 1, which is explained in Fig 1. If the weight of the edge between vertices i and j, we can define shortestPath(i, j, k +1) in terms of the following recursive formula:

ShortestPath(i,j,0) = w(i,j)

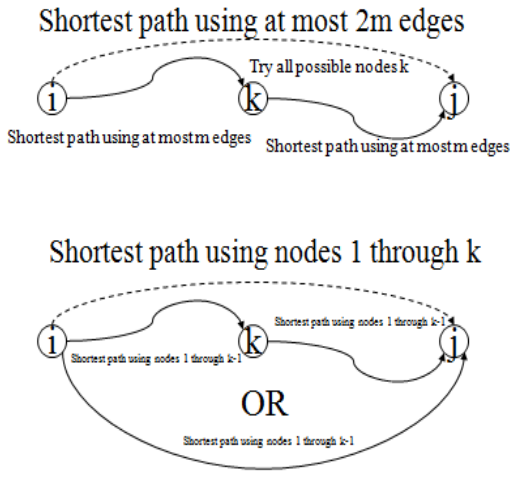

Figure 1: Shortest path between given vertices

ISSN:2319-7900

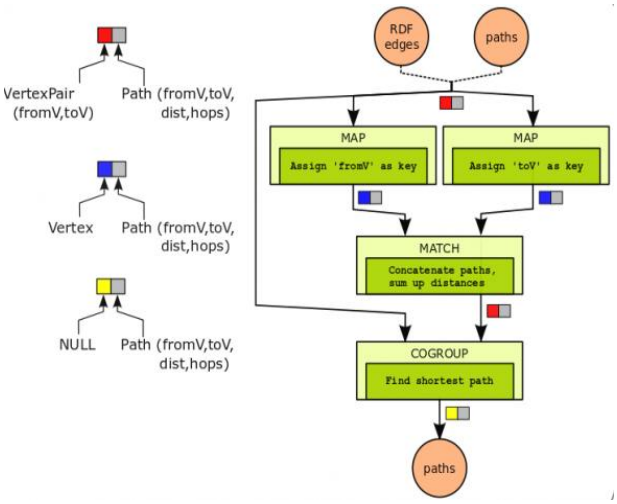

Figure 2 Steps to find shortest path

3. Existing Algorithm

The Floyd–Warshall algorithm [17] (also known as Floyd's algorithm) is a graph analysis algorithm for finding shortest paths in a weighted graph with positive or negative edge weights (but with no negative cycles) and also for finding transitive closure of a relation R. A single execution of the algorithm will find the lengths (sum of weights) of the shortest path between all pairs of vertices, though it does not return details of the paths themselves.

We initialize the solution matrix same as the input graph matrix. Then we update the solution matrix by considering all vertices as an intermediate vertex. The idea is to pick the vertices one by one and update all shortest paths which include the picked vertex as an intermediate vertex in the shortest path. When we pick vertex number k as an intermediate vertex, we already have considered vertices {0, 1, 2, .. k-1} as intermediate vertices. For every pair (i, j) of source and destination vertices respectively, there are two possible cases.

1) k is not an intermediate vertex in shortest path from i to j. We keep the value of dist[i][j] as it is.

2) k is an intermediate vertex in shortest path from i to j. We update the value of dist[i][j] as dist[i][k] + dist[k][j].

The fig. 2 and 3 shows the above optimal substructure property in the all-pairs shortest path problem. The procedure is given bellow.

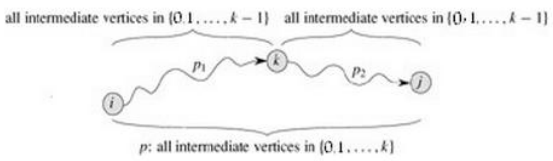

Figure 2: Intermediate Vertices

- Used to enable finding a shortest path
- Initially the array contains 0
- Each time that a shorter path from *i* to *j* is found the *k* that provided the minimum is saved (highest index node on the path from *i* to *j*)
- To print the intermediate nodes on the shortest path a recursive procedure that print the shortest paths from *i* and *k*, and from *k* to *j* can be used

4. Proposed Algorithm using DNA sequence

DNA sequence [19] is the process of determining the precise order of nucleotides within a DNA molecule. The Four DNA bases are adenine (A), guanine (G), cytosine (C), and thymine (T). In the proposed algorithm; we have used the DNA bases Adenine and Thymine. The Weight of the node in the graph has to be initialized as nucleotide after that it has to be converted as binary representation in assumption as A represented as '0' and T represented as '1'. The following algorithm explains how to reduce the time, when find the path and performance will be comparatively fast with the existing algorithm exist to find the shortest path.

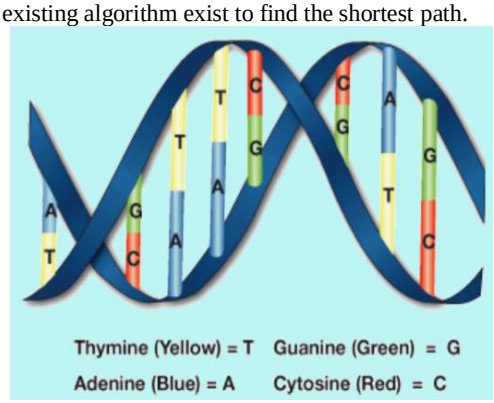

Figure 3: Graphical Representation of DNA Sequence

The result is analyzed by the implementation of the all pair shortest path algorithm using DNA Sequence is done by comparing the Floyd-Warshall algorithm for shortest path. In the proposed algorithm, if the source, intermediate and destination node is same, then it will not find the path between the nodes so this will save the time. The Result is analyzed by the algorithm described and verified for many graphs with different size of the graph. In the following code, read the input from the external file which contains the weight of the graph in the format of genome DNA sequence. First initialize the two 2 dimensional array such that D[][]="0", P[][]=0. Here D[][] ="0" is string array which is used to store the input value. P[][]=0 is an integer array which is used to store the intermediate node. This code get the input as the file Input stream format and pass the value into two dimensional array D[][] for matrix representation of the graph.

```
int getDecimalFromBinary(int binary)
//This procedure is used to convert the binary
//number into decimal number
{
        Int decimal=0;
        Int power =0;
        While (true)
```

ISSN:2319-7900

```
{
        If(binary==0)
        {
                Break;
        }
        else
        {
                Int tmp =binary %10;
                Decimal += tmp
*math.pow(2,power);
                Binary =binary/10;
        }
}
Return decimal;
}
Int BTD (string g1)
// This procedure is used to convert the genome DNA sequence into
//binary number and then using the procedure
getDecimalFromBinary(int binary)
//it is used to convert the binary number into decimal number
{
        Char[] c=character(g1)   ;//convert string value of g1 into
character array
        Int [] b= new int[4]; //declare the integer array of size
        String temp="";
        For (int z=c.length -1;z>=0;z--)
        {
                If(c[z] == 'T')
                        b[z]=1;
                else
                        b[z]=0;
        }
        For (int z=0;z<b.length; z++)
        {
                Temp=temp + b[z];
        }
Return getDecimalFromBinary(integer(temp)); //convert the string
value of temp into integer
}
Void TimeEfficientAllPath (int n, string D[][])
//This procedure is used to calculate the all pair shortest path algo-
rithm with time efficient
// this algorithm read the n number of  vertices, and weight of the
path as genome DNA sequence as // a input.
{
        Read n; //number of vertex
        For(int i=1 ;i<=n;i++)
                For(int j=1 ;j<=n;i++)
                {
                D[i][j]="0"; // initialize the weighted matrix
                P[i][j]=0 ; //  P matrix is used to store the in-
termediate node value
                }
                For(int i=1 ;i<=n;i++)
                {
                For(int j=1 ;j<=n;i++)
                {
                If(i != j) //check source and distination  not
same
                        {
                                Read D[i][j]; //read the weight of
graph in the form of //genome DNA sequence
                        }
                }
        }
//Convert Genome DNA sequence into decimal number
                For(int i=1 ;i<=n;i++)
                   For(int j=1 ;j<=n;i++)
                      If(i != j)
                        {
                          I[i][j] =BTD (D[i][j]) ;
                        }

// Calculation
                For(int k=1 ;k<=n; k++)
                For(int i=1 ;i<=n;i++)
                For(int j=1 ;j<=n;j++)
                {
                //This condition used to check source
                ,destination and //intermediate node should not
                be same
                If((i != j) && (j !=k) && (j !=k))
                   {
                   If(I [i][j] >( I[i][k] + I [k][j]))
                {
                I [i][j] =( I[i][k] + I [k][j]);
                P[i][k] = k; // P matrix used to store the inter-
mediate node
                                }

                        }
                }
        }

        }
}
```

5. Conclusion

The algorithm run on i3 machine with 2 GB RAM and it is implemented in Java JDK 2.0 and it works perfectly. The above algorithm is verified with more the 15 graphs and it generated the correct paths between all the pair of vertices. The above algorithm can also be executed parallel manner if DNA computer exist or if DNA computing concept is implemented using Java threads [20].

References

[1] P K Singh, Rajendra Kumar Member, IACSIT and Vijay Shankar Pandey,"*An Efficient Algorithm for All Pair Shortest Paths*", International Journal of Computer and Electrical Engineering, Vol. 2, No. 6, December, 2010

[2] Stefan Hougardy , "*The Floyd-Warshall Algorithm on Graphs with Negative Cycles* ",Information Processing Letters 110 (2010), 279-281

ISSN:2319-7900

[3] Udaya Kumar Reddy K. R, and K. Viswanathan Iyer, "*All-pairs shortest-paths problem for unweighted graphs in O(n log n) time*", International Journal of Computational and Mathematical Sciences 3:5 2009

[4] Yijie Han, "*An O(n log log n/ log n) time algorithm for all pairs shortest paths*", Manuscript, 2009.

[5] Wikipedia. Floyd-Warshall algorithm — Wikipedia, The Free Encyclope-dia, 2009. [Online; accessed 20-November-2009].

[6] Gary J. Katz1,2 and Joseph T. Kider Jr1 , "*All-Pairs Shortest-Paths for Large Graphs on the GPU*",Graphics Hardware (2008) David Luebke and John D. Owens (Editors)

[7] Timothy M. Chan, "*All-pairs shortest paths with real weights in O(n / log n) Time*", Algorithmica, 50:236–243, 2008.

[8] Yijie Han," *An O(n (log log n/ log n)) time algorithm for all pairs shortest paths*", Algorithmica, 51:428–434, 2008.

[9] Yijie Han, "*A note of an O(n / log n) time algorithm for all pairs shortest paths*", Information Processing Letters, 105:114–116, 2008.

[10] Timothy M. Chan, "*More algorithms for all-pairs shortest paths in weighted Graphs*", In STOC07, pages 590–598, 2007.

[11] Uri Zwick, "*A slightly improved sub-cubic algorithm for the all pairs shortest paths problem with real edge lengths*", Algorithmica, 46:181–192, 2006.

[12] Tadao Takaoka, "*An O(n log log n/ log n) time algorithm for the all-pairs shortest path problem*", Information Processing Letters, 96:155–161, 2005.

[13] Seth Pettie, "*A new approach to all-pairs shortest paths on real-weighted Graphs*", Theoretical Computer Science, 312:47–74, 2004.

[14] Tadao Takaoka, "*A new upper bound on the complexity of the all pairs shortest path problem*", Information Processing Letters, 43:195–199, 1992.

[15] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein, "*Introduction to Algorithms*" MIT. Press, Mcgraw-Hill Book Company, ISBN 0-262-03141-8, 1990.

[16] Michael L. Fredman, "*New bounds on the complexity of the shortest path Problem*" ,SIAM Journal on Computing, 5(1):83–89, 1976.

[17] Tadao Takaoka, "*A faster algorithm for the all-pairs shortest path problem and its application*" In K. -Y. Chwa and J. I. Munro, Springer-Verlag LNCS Vol 3106, pp 278–289.

[18] Barnes, M. a. (2003). *Bioinformatics for Geneticists*.

[19] http://en.wikipedia.org/wiki/DNA_sequencing. (n.d.).

[20] A. Murugan and B.Lavanya and K. Shyamala *A Novel Programming Approach for DNA Computing.* International Journal of Computational Intelligence Research, 7(2):199-209, 2011.