ISSN No. 0976-5697

Volume 8, No. 3, March – April 2017

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Cluster of Optimal Chains of All-Pairs Shortest Paths to Deduce Minimum Cost Tour

Govindaraj Pandith T.G
Department of Computer Applications
AIMS Institute of Higher Education
Bangalore, India
Registered research candidate in Rayalaseema University
Kurnool

Dr. Siddappa M
Department of Computer Science and Engineering
Sri Siddhartha Institute of Technology
Tumkur, India

Abstract: This paper aims at analyzing all-pairs shortest paths algorithm by Floyd to extract optimal path between any pair of vertices of various path length. To form least cost round tour, vertices of an optimal chain are grouped into cluster. These optimal chains are combined by taking an account of both path length as well as magnitude of the path. The chain which has largest path length and optimal distance will be considered first. Tour is constructed by adding up vertices to optimal chains in suitable positions based on nearness.
This approach is a mix of both dynamic and greedy strategies. Illustrations of different variations are focus of the study.

Keywords:Cost Adjacency Matrix, Optimal Chain, Round tour, Path length matrix, predecessor matrix and all-pairs shortest paths

I. INTRODUCTION

There are several algorithms for solving shortest path problem viz., Bellman-Ford algorithm, reaching algorithm, Floyd-Warshall algorithm and Dijkstra's algorithm[1]. Routing algorithms are of prime importance in computer networking. There are three major types of algorithms.

- Single Pair - These algorithms take a specific source and destination. Only one path is required in response, and this is usually assumed to be the most optimal.
- Single Source - In this type algorithm, the path from one node to all the others is required.
- All Pairs - The algorithm must return the shortest paths from every waypoint to all the others. This is naturally the most computationally expensive.

The all-pairs shortest path problem can be considered the most prominent of all routing problems. It aims to compute the shortest path between any pair of vertices v and u. Using Dijkstra's single-source algorithm, it is possible to get a naive implementation of O(n^3) since the O(n^2) algorithm must be iterated for every vertex -- i.e. running an O(n^2) process n times.

Storing all the paths explicitly can be impractical in terms of memory consumption, so these are usually considered as all-pairs shortest distance problems [2], which aim to find just the distance from a vertex to another. The result of this operation is an n x n matrix, which stores estimated distances to the each node.

The pseudo code of the Floyd's algorithm [3-4] all-pairs shortest paths

```
Algorithm AllPairShortestPaths(C[1:n,1:n])
 //Input: Cost adjacency matrix C of the given graph with n vertices
 //Output: n x n distance matrix D of the shortest Paths', pred a n x n
 //matrix of intermediate nodes and a n x n matrix pathlength which
 //contains the length of the optimal path between a pair of vertices.
 {
        for i=1 to n do  //Initialize distance matrix

        for j=1 to n do
        {
                D[i,j]= C[i,j];
                pred[i,j]=0;
                pathlength[i,j]=0;
                if(C[i,j]!=0)
                        pathlength[i,j]=1;
        }
    for k=1 to n do
        for i=1 to n do
          for j=1 to n do
          if(D[i,k]+D[k,j]<D[i,j])
          {
                D[i,j]= D[i,k]+D[k,j];
                pred[i,j]=k;
                pathlength[i,j]=pathlength[i,j]+1;
          }
 }
```

The matrix D[i,j] consisting of all distances from vertex v_i to vertex v_j is known as the all-pairs shortest path matrix, or more simply, the graph distance matrix[5].

Main objectives of this paper are to

i) Analyse Floyd's algorithm to form cluster of chains using predecessor matrix.
ii) Adopt different variations and obtain least cost round tour based on pathlength matrix..

In subsequent sections different variations to obtain optimal chains are depicted

II. DIRECTLY FORMING THE CHAINS BY ADDING MINIMUM COST EDGES

Let G be a n vertex graph represented by cost adjacency matrix. Starting from any arbitrary vertex move to the nearest unvisited vertex till all the vertices are included in the path. Repeat the step for all the vertices. Hence there will be n chains of path length n-1. By connecting the end vertices of these individual chains will result in n Hamiltonian Circuits. The minimum among these tours can be considered as an approximate minimum cost round tour.

For example consider the following instance of weighted graph

$$\begin{pmatrix} 0 & 2 & 3 & 4 & 5 \\ 6 & 0 & 7 & 8 & 9 \\ 9 & 2 & 0 & 3 & 1 \\ 1 & 7 & 8 & 0 & 9 \\ 8 & 9 & 6 & 5 & 0 \end{pmatrix}$$

Path1 --> 1-->2-->3-->5-->4
Cost of this path =2+7+1+5 = 15
Path2 --> 2-->1-->3-->5-->4
Cost of this path =6+3+1+5 = 15
Path3 --> 3-->5-->4-->1-->2
Cost of this path =1+5+1+2 = 9
Path4--> 4-->1-->2-->3-->5
Cost of this path =1+2+7+1 = 11
Path5 --> 5-->4-->1-->2-->3
Cost of this path =5+1+2+7 = 15

Path 1, ... , Path 5 are optimal chains starting from vertices 1,…,5 respectively. Connecting the end vertex to starting vertex of these paths will give round tours. Except for Path 2, remaining paths provide round tours of cost 16. Therefore any one of these tours is the required minimum cost round tour viz., -→5→4→1→2→3→5 with total cost 16.

```
Algorithm OptimalChains(C[1:n,1:n])
// Input:Cost adjacency matrix C[1:n,1:n] of Graph G
//Output: n Optimal Chains
{
    s=1; //  initialize s with vertex 1
    while(s<=n)
    {       start=s; cost=0;k=1;
             p[k]=s;
            for i=1 to n do
            {
               m=1; // To set a vertex and initial minimum value
                while(c[start][m]==0 || visited(m)==1)
                            m++;
                min=c[start][m];   p[++k]=m;
                for i=1 to n do
                {
                            if(start==j)
                               continue;
                            if(visited(j)==1)
                               continue;
                            else {
                                                    if(c[start][j]<=min)
                                                    {
                                                      min=c[start][j];p[k]=j;
                                                    }
                                  }
                }
                start=p[k];
                            if(k==n)
                              break;
            }
            print("Path -->");
            for i=1 to n do
            {
             print(p[i]);
            }
            print(newline);//start a new line
            print("Cost of this path =");
            for i=1 to n do
            {
                 print( c[p[i]][p[i+1]]);
                 cost=cost+ c[p[i]][p[i+1]];
            }
            print(cost);
            print(newline);
            for (i=1;i<=n;i++)
                        p[i]=0;
            s++;
            }
}
```

```
Algorithm  visited( v)
{
     for i=1 to n do
     {
      if(v==p[i])
               return 1;
      }
    return 0;
}
```

The execution time of this algorithm is of order of O(n^4).

III CONSTRUCTION OF CHAIN BY SELECTING DISTANT VERTEX

In this method obtain all-pairs shortest path matrix by Flyod’s algorithm. Extract pair of vertices i and j which are far away from each other. Construct the path using the following procedure.

```
Path (i,j)
{
     if(pred[i,j]=0)
     {     print(i,j); //single edge
           return;
     }
     else  // Compute the two parts of the path
     {
           Path(i,pred[i,j]);
           Path(pred[i,j],j);
     }
}
```

Complete the tour by adding vertices which are not visited using greedy approach by inserting nearest vertices to either of the end points of the chain.

Alternatively two distant vertices can be obtained by Dijkstra’s single source shortest path algorithm [6-8]. Johnson’s algorithm can be used since it is efficient than Floyd’s in the case if a graph is a sparse [9] (Even though TSP focuses on a complete graph some real life situation can be a sparse graph).

For the given example all pairs shortest path matrix is

$$\begin{pmatrix} 0 & 2 & 3 & 4 & 4 \\ 6 & 0 & 7 & 8 & 8 \\ 4 & 2 & 0 & 3 & 1 \\ 1 & 3 & 4 & 0 & 5 \\ 6 & 8 & 6 & 5 & 0 \end{pmatrix}$$

Pair of distant vertices are (2,4), (2,5) and (5,2) extract the path between these pairs by using predecessor matrix and using the procedure Path (specified above). Predecessor matrix for given instance is

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 3 \\ 4 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 3 \\ 4 & 3 & 0 & 0 & 0 \end{pmatrix}$$

Corresponding paths are

path from 2 to 4: 2--->4
path from 2 to 5: 2--->3--->5
path from 5 to 2: 5--->3--->2.

Select the longest among these for example the path between (2,5) 2---> 3---> 5. Now find the optimum chain of remaining vertices (4,1) i.e 4--->1. Join this path to either before to vertex 2 or after the vertex 5. We have two chains

4--->1--->2--->3--->5 and its total cost is 11
2--->3--->5--->4--->1 and its total cost is 14

Similarly select the path between (5,2) 5--->3--->2 and join the optimum chain of remaining vertices 4--->1. Then two chains will be

4--->1--->5--->3--->2 and its cost will be 14
5--->3--->2--->4--->1 and its cost is 17.

Therefore most probably the required optimal round will be →5→4→1→2→3→5 with total cost 16.

III. USING PATH LENGTH MATRIX

First obtain the all-pairs shortest path's path length matrix. Select the pair of vertices i and j which have highest path length. Obtain the path between i and j. Get the optimal chain by adding remaining vertices. Path length matrix for the given instance is

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 2 \\ 1 & 0 & 1 & 1 & 2 \\ 3 & 1 & 0 & 1 & 1 \\ 1 & 2 & 2 & 0 & 3 \\ 2 & 2 & 1 & 1 & 0 \end{pmatrix}$$

It is clear from the above matrix that distant pair of vertices in terms of pathlength are (3,1) and (4,5).

The shortest paths between these pair of vertices are

3--->4--->1 with cost 4 (observe the fact that in path length matrix the entry for vertex 3 and 1 is 3 i.e the length of the path is atmost 3) and

4--->1--->3--->5 with cost 5

Select the second chain since it covers more vertices and insert the remaining vertex 2 in the appropriate position based on proximity. Since vertex pair (1,3) is nearest to 2 insert the vertex between 1 and 3 to get the optimal chain.(Obtain the optimum paths if more than one vertex remains and merge the paths).

4--->1--->2--->3--->5 and its total cost is 11.

Linked list data structure can be used for inserting vertices or merging the optimal paths

Therefore most strongly recommended optimal round tour will be 4--->1--->2--->3--->5--->4 with total cost 16.

IV. RESULTS

TSP is a complex problem of combinatorial graph theory and belongs to category of factorial time complexity for which finding an exact solution is very hard when the given graph contains more number of vertices.

An attempt has been made to obtain most appropriate solution by grouping the vertices of predecessor matrix into optimal chains and also by taking an account of path length. For the given problem by observing different variations the best round tour can be →5→4→1→2→3→5 with total cost 16.

V. CONCLUSION

This study is all about clustering vertices based on all pairs shortest path algorithm and utilizing these clusters to form optimal round tour. This approach can achieve solution with relatively less number of iterations compared to classical algorithms of Branch and Bound or Backtracking.

The major objective behind this study is to form variations to deduce minimum cost tour. Attempts can be made with following variations

i) Starting from closest pair in all pairs shortest path matrix
ii) Union of least cost paths
iii) Forming the chain of vertices by selecting shortest length paths from path length matrix

As a non-exact solver the approach adopted in this study can be used for obtaining feasible solution in polynomial time.

VI. REFERENCES

[1]. Lauschke, Andreas and Weisstein, Eric W. "Shortest Path Problem." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/ShortestPathProblem.html

[2]. Alex J. Champandard "Path-Planning from Start to Finish"

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

[4]. Larson, R. and Odoni, A. "Shortest Paths between All Pairs of Nodes." §6.2.2 in Urban Operations Research. 1981.

[5]. Buckley, F. and Harary, F. Distance in Graphs. Redwood City, CA: Addison-Wesley, 1990.

[6]. Dijkstra, E. W. "A Note on Two Problems in Connection with Graphs." Numerische Math. 1, 269-271, 1959.

[7]. Skiena, S. "Dijkstra's Algorithm." §6.1.1 in Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica. Reading, MA: Addison-Wesley, pp. 225-227, 1990.

[8]. Whiting, P. D. and Hillier, J. A. "A Method for Finding the Shortest Route through a Road Network." Operational Res. Quart. 11, 37-40, 1960.

[9]. Johnson, Donald B. (1977), "Efficient algorithms for shortest paths in sparse networks", Journal of the ACM 24 (1): 1–13, doi:10.1145/321992.321993.

About Authors

Govindaraj Pandith T.G has fourteen years of teaching experience for UG and PG courses for computer applications and is presently working as Assistant Professor in the department of Information Technology, AIMS Institute of Higher Education, Bangalore. He Obtained B.Sc.(Physics, Chemistry, Mathematics) from Sri Bhuvanendra College, Karkala affiliated to Mangalore University in the year 1994. Completed PGDCA from Dr. NSAM First Grade College, Nitte in the year 1995 affiliated to Mangalore University. He completed M.Sc, Mathematics from Mangalagangotri, Mangalore University in the year 1997. He also successfully completed M.Sc, Computer Science from Mangalagangotri, Mangalore University with distinction in the year 2001. His research interests are in the areas of Discrete mathematics, Data structures, Pattern Recognition, Analysis and Design of Algorithms. Currently pursuing PhD from Rayalaseema University, Kurnool under the guidance of Dr. Siddappa M, Head of the department of Computer Science and Engineering, Sri Siddhartha Institute of Technology, Tumkur.

Dr. M.Siddappa received B.E and M.Tech degree in Computer Science & Engineering from University of Mysore, Karnataka, India in 1989 and 1993 respectively. He has completed doctoral degree from Dr.MGR Educational Research Institute Chennai under supervision of Dr.A.S.Manjunatha, CEO, Manvish e-Tech Pvt. Ltd., Bangalore in 2010. He worked as project associate in IISc, Bangalore under Dr.M.P Srinivasan and Dr. V.Rajaraman from 1993 – 1995. He has teaching experience of 29 years and research of 15 years. He published 62 Technical Papers in National, International Conference and Journals. He has citation index of 192 till 2016 and h-index of 5 and i10-index of 4 to his credit. He is a member of IEEE and Life member of ISTE. He is working in the field of data structure and algorithms, Artificial Intelligence, Image processing and Computer networking. He worked as Assistant Professor in Department of Computer Science & Engineering from 1996 to1998 in Sri Siddhartha Institute of Technology, Tumkur. Presently, he is working as Professor and Head, Department of Computer Science & Engineering from 1998 in Sri Siddhartha Institute of Technology, Tumkur. He has visited Louisiana university Baton rouge, California university and Wuhan university China. He bagged “Best Teacher” award from ISTE in the year 2013.