संकुल नवप्रवर्तन केंद्र, दिल्ली विश्वविद्यालय
CLUSTER INNOVATION CENTRE, UNIVERSITY OF DELHI

CITY PLANNING : An Approach through Shortest Path Algorithm

Major Project Report

V.1 Discretizing and understanding real life situations through a mathematical lens

Mentor

Dr. Shashi Aggarwal
Associate Professor, CIC
University of Delhi

Submitted By

Aman Thakur
Anurag Kumar
Nikita Garg
Rahul Yadav

ACKNOWLEDGEMENT

We would like to thank our mentor Dr. Shashi Aggarwal for her constant encouragement and stimulating advice. Without her help the project would not have come to its present state. We thank her for sparing her valuable time to attend our petty doubts. Her notes and documentation enhanced our clarity over the subject. We would also like to thank our friends and family for their unconditional support and love.

-Authors

TABLE OF CONTENTS

1. Abstract

The shortest path problems have always been a highly competitive area of research. Mathematicians have applied various shortest path algorithms in a large number of fields. From routing complexities in the network protocols, to path optimization in air traffic control the utility of finding optimally shortest path provided with respective constraints is of high importance. In the following paper, we have examined the performance of different algorithms for the shortest path problems. The widely used Floyd's algorithm, Dijkstra's Algorithm, the Genetic Algorithm have been considered for study. **We have also presented a new algorithm of our own, for solving the problem of finding the shortest distance. As an implementation, we have considered a kind of city planning with reference to graph theory.**

2. Introduction

Graph theory is a branch of Mathematics and Computer Science used to model various structures in order to bring new inventions and modifications in the existing environment. Various problems such as job scheduling, list colouring, seven bridge problem, shortest path etc. are based on the concepts of Graph theory.

In Computer Science, Graph theory is used in data mining, image segmentation, clustering, networking etc. The visualization of a network in the form of graph provides convenient and efficient method for building and testing of various algorithms.

Graph

A graph represents a relationship between two or more variables. Based upon the direction, graphs are classified into directed and undirected graphs.

Node

A node or a vertex in a graph represents an object or a variable that might be linked to other objects.

Edge

An edge determines the relationship between two or more nodes.

2.a Adjacency Matrix

Adjacency Matrix represents the existence of a relationship between adjacent vertices in a graph. It also represents the existence of a relationship of any vertex with itself.

2.b Weighted Adjacency Matrix

An extension to adjacency matrix, weighted adjacency matrix not only determines the existence but also shows the nature of the relationship between adjacent vertices in terms of the numerical quantity provided to the edge connecting the two nodes.

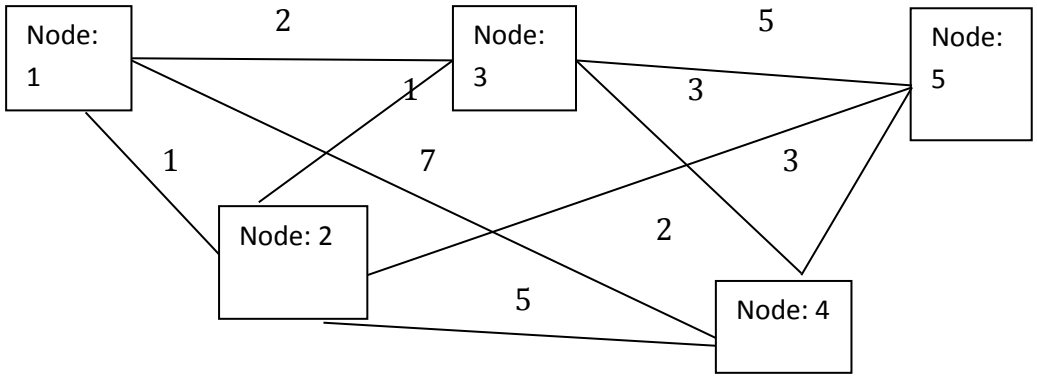

Figure1: Undirected Graph

Adjacency Matrix:

	Node: 1	Node: 2	Node: 3	Node: 4	Node: 5
Node: 1	0	1	1	1	0
Node: 2	1	0	1	1	1
Node: 3	1	1	0	1	1
Node: 4	1	1	1	0	1
Node: 5	0	1	1	1	0

Weighted Adjacency Matrix:

	Node: 1	Node: 2	Node: 3	Node: 4	Node: 5
Node: 1	0	1	2	7	0
Node: 2	1	0	1	5	2
Node: 3	2	1	0	3	5
Node: 4	7	5	3	0	3
Node: 5	0	2	5	3	0

2.c The City Planning Problem

An important issue that concerns urbanization is that most of the cities that are coming up do not actually follow a plan and are a result of utter mismanagement. In this paper, we propose the basic planning of a city mainly in terms of transportation. The user can place the landmarks and other features such as green cover, lakes road networks and see for himself how optimized the map would look like in terms of area and other necessities.

After the map has been designed, the user can traverse throughout the city, travelling the shortest path. Several shortest path algorithms have been studied and their shortcomings analyzed. Based upon these shortcomings, a new algorithm has been proposed, that provides a solution to most of the questions.

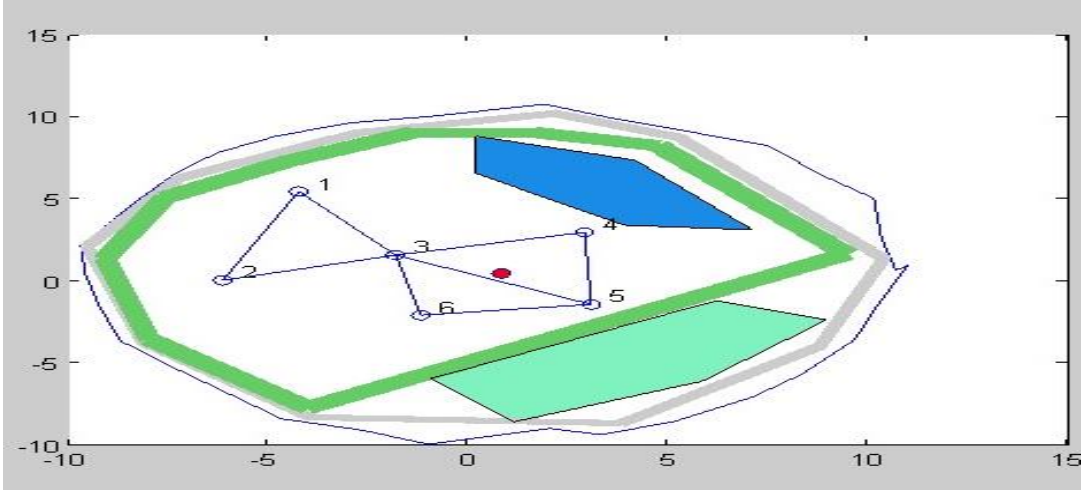

Figure 2: Basic City Plan. Blue region represents water body and the green represents forest cover.

3. Shortest Path Algorithms

3.a Floyd`s Algorithm

The Floyd–Warshall algorithm is a graph analysis algorithm for finding shortest paths in a weighted graph with positive or negative edge weights. A single execution of the algorithm will find the lengths of the shortest paths between all pairs of vertices, though it does not return details of the paths themselves.

The Floyd–Warshall algorithm compares all possible paths through the graph between each pair of vertices. It is able to do this with only $\Theta(|V|^3)$ comparisons in a graph. This is remarkable considering that there may be up to $\Omega(|V|^2)$ 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.

Pseudocode:

```
let dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)
for each vertex v
  dist[v][v] ← 0
for each edge (u,v)
  dist[u][v] ← w(u,v)  // the weight of the edge (u,v)
for k from 1 to |V|
  for i from 1 to |V|
    for j from 1 to |V|
        if dist[i][k] + dist[k][j] < dist[i][j] then
            dist[i][j] ← dist[i][k] + dist[k][j]
```

3.b Dijkstra`s Algorithm

Dijkstra's algorithm is a graph search algorithm that solves the single-source shortest path problem for a graph with non-negative edge path costs, producing a shortest path tree. For a given source vertex (node) in the graph, the algorithm finds the path with lowest cost (i.e. the shortest path) between that vertex and every other vertex. It can also be used for finding costs of shortest paths from a single vertex to a single destination vertex by stopping the algorithm once the shortest path to the destination vertex has been determined. For example, if the vertices of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities. As a result, the shortest path first is widely used in network routing protocols, most notably IS-IS and OSPF (Open Shortest Path First).

Pseudocode:

```
function Dijkstra(Graph, source):
  for each vertex v in Graph:                  // Initializations
      dist[v]     := infinity;                 // Mark distances from source to v as not yet computed
      visited[v]  := false;                    // Mark all nodes as unvisited
      previous[v] := undefined;                // Previous node in optimal path from source
  end for

dist[source] := 0;                             // Distance from source to itself is zero
 insert source into Q;                         // Start off with the source node
while Q is not empty:                          // The main loop
   u := vertex in Q with smallest distance in dist[] and has not been visited;  // Source node in
first case
   remove u from Q;
   visited[u] := true                          // mark this node as visited
   for each neighbor v of u:
     alt := dist[u] + dist_between(u, v);      // accumulate shortest dist from source
    if alt < dist[v] && !visited[v]:
      dist[v] := alt;                          // keep the shortest dist from src to v
      previous[v] := u;
      insert v into Q;                         // Add unvisited v into the Q to be processed
    end if
  end for
 end while
 return dist;
endfunction
```

3.c Genetic Algorithm

Genetic Algorithms are advanced heuristic search algorithms. Their beauty lies in the fact that they closely try to mimic the process of biological evolution of living organisms. Every organism tries to evolve by the process of selection, crossover and mutation which is cyclic. We try to harness this power of evolution of genetic algorithms in our study.

- The individual chromosome is encoded as a path that we need to find for the given problem. It is this chromosome, and many more of its variants that are produced in a prefixed amount during every generation, and evolve. The chromosome is encoded using the first and the last bit as the start point and the destination nodes in the graph.

- Further, to evaluate the performance and closeness to our desired results, we formulate our fitness function. This fitness function is chosen as our objective function - inverse of path length.
- Now to prevent the Genetic search from transforming into a random search, and guiding it by suitable methods, we introduce the selection, crossover and mutation techniques. As a premier, the Selection process takes place to filter out unfit or chromosomes that have low scores on the fitness function. We use Truncation method for this selection process, where a definite amount of best chromosome population, is only selected to reproduce by the crossover process and form children chromosomes.
- The standard Crossover technique is used, which is one point crossover. The nth random gene is selected at random for a random pair of chromosomes of selected population and about that gene as an axis, their genetic material is then swapped about this axis.
- Mutation is done to mimic the process of making errors while copying. This is also deliberately done to take the search out of local minima to the level of global minima by accommodating higher space for a random change in genes. Here we do this by randomly selecting genes and swapping them for a pair of chromosomes.
- Each iteration of this process is repeated till we get desirable minima of path length. This is known as one generation. As this process is not guided for termination till now, we generally end it by either fixing the number of generations or a threshold value for the fitness function.

4. Drawbacks of existing algorithms

The time complexity of Floyd's algorithm has a running time complexity of $O(n^3)$ which is very high especially when we are dealing with large, dense graphs.

Dijkstra's algorithm works with a running time complexity of $O(n^2)$ which is better than the Floyd-Warshell algorithm but the algorithm fails when the weights are negative.

Genetic Algorithm is preferably used to solve complex pareto optimization problems. Its performance on nonlinear optimization problems involving multiple constraints and objectives is pretty acceptable. But where it lacks is the fact that there is no proof for what it obtains as the solution chromosome for finding the desirable maxima or minima is the global maxima or minima. There is no mathematical proof of this fact too. Essentially it depends on our fed genetic parameters such as selection ratio, crossover probability, mutation probability and number of generations that what will be the penetration in the search space, how close to the global optimal is our obtained chromosome.

5. Proposed Algorithm

As mentioned earlier, the existing algorithms have drawbacks in terms of their time complexity and their inefficiency when dealing with negative weights. The proposed algorithm has the same time complexity as that of the Dijkstra's algorithm ($O(n^2)$) but it works equally efficiently on weighted graphs with negative weights also. The algorithm computes the shortest distance of each vertex in the vertex set V of the graph G from the starting vertex and also returns the corresponding path from starting vertex to the terminal vertex.

The time complexity of the proposed algorithm can be improved by eliminating loops and incorporating matrix operations. The number of variables, however, would increase.

Pseudo code

```
function Shortestpath (Graph, source):
 v is the vertex set
define weighted anjacency matrix adjacency //initialization
   for each vertex v in Graph:                    // Initializations
        Dist[v]     := infinity;              // Stores the minimum distance of each vertex from starting  vertex
         Parent[v]   :=null                    // Stores the parent vertex of each vertex
   end for
Dist[Start] :=0                          //Mark distance of start node from start node to be equal to 0
i=start;
for each j from 1 to |v|              //Starting iteration from the source
     if adjacency[i][j]~=0
       if Dist[j]>Dist[i]+adjacency[i][j]
          Dist[j]=Dist[i]+adjacency[i][j]
          Parent[j]=i
       end if
     end if
end for
for each j from 1 to |v|              //Iteration starting from the first node
      if i=start then
          continue
     else
         if adjacency[i][j]~=0
          if Dist[j]>Dist[i]+adjacency[i][j]
             Dist[j]=Dist[i]+adjacency[i][j] //stores the minimum distance
             Parent[j]=i
          end if
        end if
      end if
end for
k=1;
choose terminal vertex as terminal   //User can choose the terminal vertex
while k<|v|
       if (terminal~=start)
          path[1][k]=terminal                 //stores the path in reverse order
          terminal=Parent[terminal]
          k=k+1
      end if
end while
Reverse path and obtain the route from starting node to terminal node
Endfunction Shortestpath
```

6. Future Proposals

The city planning problem discussed in the report can further be extended with the incorporation of other landscape features and structures. For now, the user locates different features on the map manually. We propose to extend the project to a level where different features would be inserted on the map automatically, depending upon the availability of space and necessity. We would also want to compute the least traffic path between different vertices, depending upon the number of crossings in each path.

Moreover, we would also want to work on the proposed algorithm and try and reduce the time complexity.

7. References

1. Comparison of Dijktra's and Floyd's algorithms : http://www.mcs.anl.gov/~itf/dbpp/text/node35.html
2. Floyd's algorithm also known as Floyd–Warshall algorithm : http://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm
3. Dijkstra's algorithm: http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
4. Genetic algorithm: http://en.wikipedia.org/wiki/Genetic_algorithm
5. Johnson's algorithm : http://en.wikipedia.org/wiki/Johnson%27s_algorithm
6. Bellman-Ford algorithm : http://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm
7. http://students.ceid.upatras.gr/~papagel/english/java_docs/allmin.htm

8. Appendix

```
%% Program to plot a map of a city through user input and calculate the
shortest distance between landmarks

%City planning and shortest path algorithm implementation

clear
clc

%plotting the boundary
%--------------------------------------------------------------------------
r=10;
theta=0:pi/15:2*pi;
theta1=zeros(2,31);
theta2=zeros(2,31);
thetaC=zeros(2,31);
thetaC1=zeros(2,31);

for i=1:31
   pp=rand();

   theta1(1,i)=r*cos(theta(1,i))+pp;
   theta1(2,i)=r*sin(theta(1,i))+pp;


end
 %plots the centroid of the random figure
centroidX=sum(theta1(1,:))/31;
centroidY=sum(theta1(2,:))/31;

plot(theta1(1,:),theta1(2,:),'r')
%fnplt(cscvn(theta1),'g',40) ;
hold on

plot(centroidX,centroidY,'ok','MarkerEdgeColor','b','MarkerFaceColor','r')


% connect the first and the last

plot(theta1(1,:),theta1(2,:),'b')
hold on
line( [ theta1(1,length(theta1(1,:))), theta1(1,1)],
[theta1(2,length(theta1(2,:))), theta1(2,1)] ) ;
hold on
plot(centroidX,centroidY,'ok','MarkerEdgeColor','b','MarkerFaceColor','r');
hold on

% Plot Outer Ring Road
```

```
npoints1 = input( ' Input the no. of points to mark for the outer road -- '
);
[x1,y1] = ginput(npoints1);
line(x1,y1 ,'LineWidth',3,'Color',[.8 .8 .8] );
line([x1(1,1) x1(npoints1,1)], [y1(1,1) y1(npoints1,1)]
,'LineWidth',3,'Color',[.8 .8 .8] );


%plot the outer corridor

npoints6 = input( ' Input the no. of points to mark for the corridor -- ' );
[x6,y6] = ginput(npoints1);
line(x6,y6 ,'LineWidth',5,'Color',[.4 .8 .4] );
line([x6(1,1) x6(npoints6,1)], [y6(1,1) y6(npoints6,1)]
,'LineWidth',5,'Color',[.4 .8 .4] );


% Plot green cover

disp(' ') ;
disp(' Plot 5 points for forest cover ' ) ;
npoints2 = 5 ; % input( ' Input the no. of points to mark -- ' );
[x2, y2] = ginput(npoints2) ;
c = [ 0 .9 .5] ;
fill( x2, y2, c, 'FaceAlpha', 0.5 ) ;


% Plot water body

disp(' Plot 5 points water body ' ) ;
npoints3 = 5 ; % input( ' Input the no. of points to mark -- ' );
[x3, y3] = ginput(npoints2) ;
c = [ 0 .5 .9] ;
fill( x3, y3, c, 'FaceAlpha', 0.9 ) ;



%-------------------------------------------------------------------------

%confines the coordinates to within the randomly generated figure
 npoints3=input(' Input the number of landmarks needed-- ');
 [p,q]=ginput(npoints3);
 plot(p,q,'ok','MarkerEdgeColor','b','MarkerSize',6)

%  %Naming the points
%   %-----------------------------------------------------------------------
    count=length(p);
    s=49;
      for n=1:count
        Name(n,1)=char(s);
       s=s+1;
      end
      str=num2str(Name);
```

```
% %   %putting names on the points
% %   %-------------------------------------------------------------------
    for v=1:count
        text(p(:,1)+0.5,q(:,1)+0.5,str);
    end



%Formulate the adjacency matrix--------------------------------------
n=length(p);
adjacency=zeros(n,n);

    Coordinates=[p q];
        point=zeros(n,2);
        disp('Enter the no of connections');
        no=input(' ');
        point=zeros(no,2);
        for i=1:no
           point(i,:)=input('   ');
        end
       for i=1:no


adjacency(point(i,1),point(i,2))=(sqrt((Coordinates(point(i,1),1)-
Coordinates(point(i,2),1))^2+(Coordinates(point(i,1),2)-
Coordinates(point(i,2),2))^2));

adjacency(point(i,2),point(i,1))=(sqrt((Coordinates(point(i,1),1)-
Coordinates(point(i,2),1))^2+(Coordinates(point(i,1),2)-
Coordinates(point(i,2),2))^2));

       end

%Connect the landmarks-----------------------------------------------------


    for i=1:n

        for j=1:n
        if adjacency(i,j)>=1
            plot([Coordinates(i,1) Coordinates(j,1)],[Coordinates(i,2)
Coordinates(j,2)])

        end
       end
    end

duplicate=adjacency;

%finding the shortest distance between any two points----------------------

DistMat=zeros(1,n);%Stores the value of shortest distance
ParentMat=zeros(1,n);%Stores the parent of each node in the sequence
disp('Enter the starting node');
```

```
start=input(' ');
for i=1:n
    DistMat(1,i)=Inf;
end
DistMat(1,start)=0;
i=start;

    for j=1:n
        if (adjacency(i,j)>0)
            if(DistMat(1,j)>(DistMat(1,i)+adjacency(i,j)))
                DistMat(1,j)=DistMat(1,i)+adjacency(i,j);
                ParentMat(1,j)=i;
            end
        end
    end

    i=1;

while i<n
    if(i==start)

    else
    for j=1:n
        if (adjacency(i,j)>0)
            if(DistMat(1,j)>(DistMat(1,i)+adjacency(i,j)))
                DistMat(1,j)=DistMat(1,i)+adjacency(i,j);
                ParentMat(1,j)=i;
            end
        end
    end
    end
    i=i+1;
end


 k=1;
 disp('Enter the terminal node');
 terminal=input(' ');%User input for the terminal vertex
 while (k<n)
     if(terminal~=start)
         path(1,k)=terminal;
         terminal=ParentMat(terminal);

     end
     k=k+1;
 end
pathNew=[path start];
len=length(pathNew);
FinalPath=zeros(1,len);%stores the path from starting to terminating vertex
for o=1:len
    FinalPath(1,o)=pathNew(1,len+1-o);
end
```