DESIGN AND ANALYSIS OF ALGORITHMS

CS5592
PROJECT

(DUE DATE: 4/17/2017)

Maninandan Kumar Manusani-16233134
Sai Teja Makani – 16227902
Shajee Uddin Zain Mohammed - 16230179

Deliverables – Spring Semester 2017

Aim to write a report that you could proudly incorporate in a portfolio of accomplishments that potential future employers might ask for.

Feel free to use outside sources and material that are free and publicly available; just make sure to cite them appropriately. All figures should have captions and clearly labeled axes. Equations are preferably typeset with a proper equation editor.

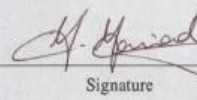
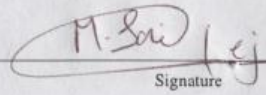
You need to explore and perhaps expand the algorithms for finding the information as outlined above. You should feel free to explore additional heuristics, or additional sensitivity questions, or algorithm/programming ideas that you feel are appropriate. The goal is to determine whether any heuristic provides an alternative to the optimal computation, and on how robust this conclusion is under various scenarios. You can write separate algorithms/programs for the separate parts, or use the same one, whichever is most appropriate for the problem. You need to write a report in which you present your findings and conclusions, supported by the data you have collected. Your analysis is expected to have appropriate rigor and communicated clearly, so that any student, who is planning to take CS5592 next semester, is able to follow your derivation. In your narrative, highlight any key insight or neat design feature that is particularly helpful for efficiency in this project, and those that were particularly enlightening to you. Your report has the following sections:

Project report

0. (0%) Cover Page, with Title, Name of Author, and Author's Declaration (see below for details).

CS5592, Spring Semester 2017: Dijkstra's Shortest Uncertain Path Algorithm
Team name: AVENGERS

I understand and have adhered to the rules regarding student conduct. In particular, any and all material, including algorithms and programs, have been produced and written by myself. Any outside sources that I have consulted are free, publicly available, and have been appropriately cited. I understand that a violation of the code of conduct will result in a zero (0) for this assignment, and that the situation will be discussed and forwarded to the Academic Dean of the School for any follow up action. It could result in being expelled from the university.

1: MANINANDAN KUMAR MANUSANI		16233134
First & Last Name	Signature	UMKC-Id
2: SAI TEJA MAKANI		16227902
First & Last Name	Signature	UMKC-Id
3: SHAJEE UDDIN ZAIN MOHAMMED		16230179
First & Last Name	Signature	UMKC-Id

Please print and sign this page. Subsequently scan it back in and attach it to your report.

Contents

1.INTRODUCTION:

Every thing in our day to day has been modeled as graphs, whether it might be the road maps, airspaces, computers, communication. When we plan a road trip from one city to another city, the different cities are considered as nodes, or intersections, or server node and the links connect them. We try to find the shortest path to reach our destination. There are various algorithms to find the shortest path, but there are various more cars on the road which will cause interference and extra delay because we share the road with more and more others. The delay along an edge is rarely a constant value, but rather fluctuates according to a random variable for each edge.

We have been given a number of different input scenarios. Each scenario will have

1. The size n of the system , together with the index of two cities, ‘a’ and ‘b’.

2. An adjacency matrix E with edge-weights. The (i, j) th entry E[i, j] represents the main characteristics of the random variable representing the traveling time on the edge between the two specific nodes, i and j, assuming that there are no other cars on the edge that might slow you down. The main characteristics we provide are T, the type of distribution, together with two more parameters, α and β.

2.EXPERIMENTAL DESIGN:

We can design a matrix E with the two nodes a and b which contains the variations of Dijkstra's Shortest Path algorithm. We use different criteria to calculate the "shortest path" for given inputs.

In our project we consider five criteria's, which are explained below:

1) **Mean value**: This is used to calculate the distance of the path whose total expected value is smallest among all the paths.

   Mean value = $\Sigma x / n$ (where Σx is the sum of the all the distances from source to destination and n= total number of nodes)

   ```
   Mean_Value(float expected value)

   return expected_value;
   ```

2) **Optimist**: The path is used such that it's "expected value - standard deviation" (as long as positive) is smallest from among all paths.

   To calculate optimist_weight(float mean, float standard_deviation)

   ```
   float edge_weight <- mean - standard_deviation;

   if edge_weight > 0 then

   return edge_weight;

   else

   return mean;
   ```

3) **Pessimist**: The path whose "expected value + standard deviation" is smallest from among all paths.
   To calculate pessimist_weight(float expected value, float standard_deviation)

   ```
   return mean + standard_deviation;
   ```

4) **Double Pessimist**: The path whose "expected value + 2 standard deviation" is smallest from among all paths.

To calculate double_pessimist_weight(float expected value, float standard_deviation)

return (mean + (2 * standard_deviation));

5) **Stable**: The path whose "squared coefficient of variation" is smallest from among all paths.

The coefficient of variation is defined as

$c^2 = V[X] / E[X]^2$

where E [X] is for the random variable on the link for edge.

V[X] is for the random variable on the link for vertex.

6) **Own_Value**: The path whose value is (expected value*square coefficient of variation). The reason to choose this criteria is , when we observe lage amount of data with large values. The coefficients are mostly smaller when compared with U and standard deviation. So by multiplying U with coefficient of variation we may get smallest path and find the efficient path.

Own Value(float expected value, float square_coefficient_variation)

return mean* square_coefficient_variation

Time Complexity:

→in our design the core part dijkistra's time complexity id O(ElogV) . where E is number of edges and V is number of vertices. Because each vertex connected to many edges and all needed to be processed before marking is as visited =true

→Creating graph also takes O(ElogV) as it should traverse on all edges and vertices to connected them.

→Proceessing Input wont take more than O(logE).as number of edges to be copied and processes to standard format.

→Print results is just a single loop to show all results so O(logE).as number of edfes is more than number of nodes.

→So overall time complexity wont exceed more than O(ElogV).

3.EXPERIMENTAL IMPLEMENTATION:

Implementation Details:

Development Language: Java

We used java object oriented programming to implement the algorithm because it is designed to develop algorithms in a easy way, we have the flexibility of n-D Matrices, dynamic memory allocation, in java and the most important point to be mentioned is that we are comfortable with using object oriented programming.

Mathematical implementation:

1. Say, Total number of nodes as N
2. Input Matrix as I[K][5]
3. Intermediate Matrix as IM[K][5]
4. Edge Matrix as E[K][3]
5. The number of hops would be h.

In the first iteration we calculate mean, variance, c2 using alpha and beta from input i.e I[i][3], I[i][4]. Suppose if I[1][2]==1 then mean equals I[i][3] and Variance and c2 as 0. Similarly as above using the table we calculate the mean variance and c2 in each criteria.

So for i equals 1 to K we calculate the mean, variance and c2 using the table given,

So we get an intermediate matrix IM[K][5] which consists of nodes of edges and their mean, variance and c2 values. Now this intermediate matrix is used to calculate edge lengths in each criterion.

For Mean Value criteria, E[K][3]= IM[K][2]

For Optimist value criteria, E[K][3]= IM [K][2]- sq.root(IM[K][3])

For pessimist value criteria, E[K][3] = IM[K][2] + IM[K][3]

For doubly pessimist value criteria, E[K][3] = IM[K][2] + 2*IM[K][3]

For stable value criteria, E[K][3] = IM[K][4]

For own value criteria, E[K][3]=IM[K][2]*IM[K][3]

So we get the edge lengths in each criteria and in each criteria we give this Edge Matrix as input to Dijkstra's algorithm to find shortest path possible.

Shortest path = Dijkstra_algorithm(E[K][3])

Now we are going to see how the core logic execute for the below example and how our program logic satisfies critical conditions of the algorithmic behavior.

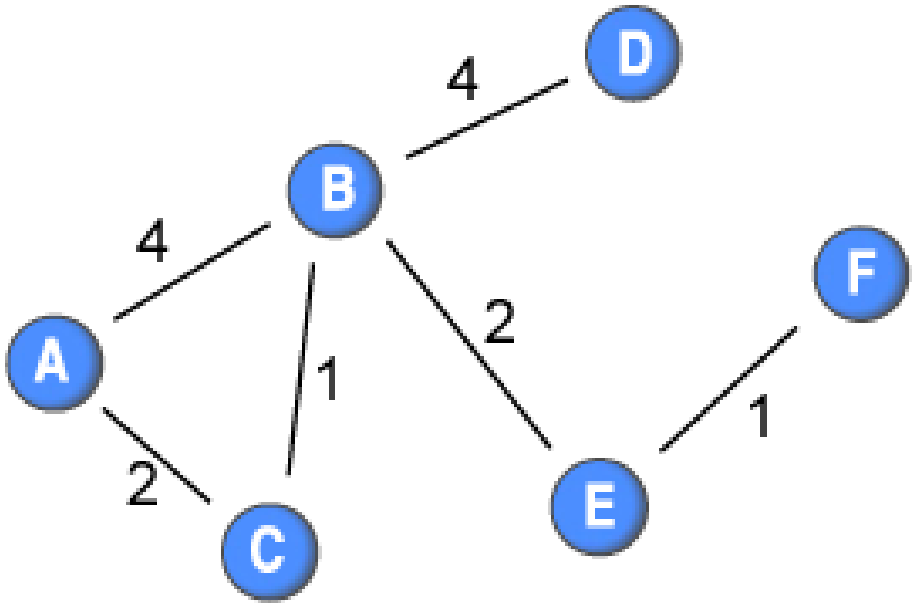


Figure 4.1 Example

Scenario: Lets take an example of the above graph and find the shortest distance between A and E.

Now we will see the execution pattern for the above senarion.

1. Total Number of nodes here N = 6

2. As we are explaining the core part omitting conversion of inputs into standard Dijkstra input format

3. As we are explaining the core part omitting conversion of inputs into standard Dijkstra input

Format(refer section 2)

4. Edge input would be {{A,C,2}, {A, B ,4},{C,B,1}, {B,E,2}, {B,D,4}, {E,F,1} }

5. Number of hopes defined by H.

Referring code sample part:

//Core part responsible for finding short distance

```
Step 1→for(int i=0;i<this.nodes.length;i++){
            ArrayList<Edge>              currentNodeEdges              =
this.nodes[nextNode].getEdges();
            //getting all the edges connecting to a particular node

            for(int joinedEdge=0 ; joinedEdge < currentNodeEdges.size()
;    joinedEdge++){
Step 2→    int                                                neighbourIndex
=currentNodeEdges.get(joinedEdge).getNeighbourIndex
(nextNode);

Step 3→    if(!this.nodes[neighbourIndex].isVisited()){
Step 4→    int                                                  tentative=
this.nodes[nextNode].getDistanceFromSource()+currentNodeEdges.
                                        get(joinedEdge).getLength();
```

```
Step 5→    if(tentative<
nodes[neighbourIndex].getDistanceFromSource()){
Step 6→    nodes[neighbourIndex].setDistanceFromSource(tentative);
                        }
                    }

                }
```

//Core part responsible for finding short distance

Lets say starting node is A destination is D .Identified 6 key steps in the core part lets execute our example for these steps in recursion cycles

Precondition distance form starting node to itself is 0.

And distance from neighboring node is set to Integer.MAX_VALUE.

Cycle1: Starts at A

Step1: Takes the current edges connecting to a node to currentNodeEdges array. So for A we have 2 edges it takes 2 Edges.

Step 2 to step 6 executes recursively until all edges connected to current node becomes null.

Step2: gets the vertex of each edge connected to current Node . now neighbor index = B

Step3:node B is not yet visited so condition satisfies

Step4: here we will find the distance from source here A to B is tentative= 4

Step:5 checking weather the tentative is less than current distance from source

Step6: If less we will assign the tentative distance as the shortest distance from source here A to B = 4

Step2: . now neighbor index = c

Step3: node c is not yet visited so condition satisfies

Step4: here we will find the distance from source here A to C is tentative2

Step:5checking weather the tentative is less than current distance from source

Step6: If less we will assign the tentative distance as the shortest distance from source here A to C = 2

Cycle2: now we will move to C and repeat the steps again

Step1:now we will load the next node i.e C into currentNode

Step2: gets the vertex of each edge connected to current Node . now neighbor index = B

Step3:node is not yet visited so condition not satisfies

Step4: here we will find the distance from source here A to B is tentative= 2+1

Step:5 checking weather the tentative is less than current distance from source

Step6: If less we will assign the tentative distance as the shortest distance from source here A to B= 3 (updated)

If you see on the above cycle2_--> step6 the distance from source of the current node always updated if it is less than tentative distance.

So our algorithm updated the shortest distance from each node and try to improve the shortest distance as when it visits the node.

4.DATA COLLECTION AND INTERPRETATION OF RESULTS:

1) In the input data structure given is of the format

   input_edges[][]=[FROM NODE, TO NODE, INPUT SCENARIO, ALPHA, BHETA]

   input_edges[1][]=[1,2,5,25,1]

2) Later depending on the input scenario format we will generate the **input_intermediate** array matrix which is in the format of [FROM NODE, TO NODE, EXPECTED VALUE, VARIATION, squared coefficient of variance.

   Here it is of type 5, So Expected Value =25, Variation = 1, Squared coefficient= 0.0016

   So input_intermediate= [1,2,25,1,0.0016]

   Similarly we get all outputs intermediate matrix of all input matrix

   So now these input_intermediate results are given as inputs to 6 different functions like mean_value, optimistic, pessimistic, double pessimistic, stable and own_value.

   For above example, mean= expected value, So the resultant edge matrix is

E[1][]=[1,2,25], Similarly for other criterions are calculated

This gives output as edge matrices of each criterion which consists of the standard Dijkstra's format like [start node, end node, edge weight]
In each of the functions above we are going to execute Dijkstra algorithm and fetch the shorted path and shortest distance.
By analyzing the shortest path followed we calculate number of hops and path distance, we find the best path among the 6 criterions.

Performance Tables:

For Input 3:

	μ - σ	μ	μ + σ	μ + 2*σ	c ²	μ * c ²
Mean Value	77.381195	111.04555	144.7099	178.37424	2.051538	68.34762
Optimistic	77.381195	111.04555	144.7099	178.37424	2.051538	22.44322
Pessimistic	102.48119	117.145454	131.80971	146.47397	1.66	0.045454
Double pess	140.1	141.1	142.1	143.1	0.0020661156	0.045454
Stable_value	140.1	141.1	142.1	143.1	0.0020661156	0.045454
Own_value	140.1	141.1	142.1	143.1	0.0020661156	0.045454

For Input 4:

	μ - σ	μ	μ + σ	μ + 2*σ	c ²	μ * c ²
Mean Value	28.9999	136.9997	244.9944	352.99915	5.0	107.9972
Optimistic	77.9889	145.99994	214.01099	241.784	3.57438	87.13631
Pessimistic	140.95856	152.99998	165.04143	177.08287	0.5770911	12.209692
Double pess	140.95856	152.99998	165.04143	177.08287	0.5770911	12.209692
Stable_value	140.95856	152.99998	165.04143	177.08287	0.5770911	12.209692
Own_value	140.95856	152.99998	165.04143	177.08287	0.5770911	12.209692

5.CONCLUSION:

We finally got the outputs of different criterion, i.e the shortest paths calculated and number of hops in each case. Considering the outputs the following conclusions are drawn

In mean value criterion though the shortest path calculated is larger but the number of hops is similar to other nodes. The edge length formula used to calculate results in larger value of shortest distance.

In optimistic value criterion, the shortest path criterion reached is less than mean value but is greater than other criterion, But the number of hops is same as previous criterion. Here all the edge lengths are lesser or equal when compared to mean value criterion.

In Pessimistic value criterion, each edge length calculated is larger than mean value and optimistic value. So, the path calculated is larger value but it is due to larger edge lengths. Similarly Doubly pessimistic value also results in larger values of shortest paths than pessimistic.

In stable value criterion edge lengths calculated are of shorter lengths than other values and similarly it results in shorter path value as low compared to other criterions.

Analyzing the own value criterion, it has the next shortest distance to stable value criterion and gives the best optimal result computing the expected value and squared coefficient of variation.

So stable value criterion results in shortest path to travel from initial node to final node.

Coming to the Time Complexity, For 12 nodes and 20 edges time taken is 15 milliseconds which is less and even if we increase the nodes and edges in graph to a 3 digit number the run time would reach milliseconds which is very small. The conclusion we can draw from the algorithm is that we get the paths in all criterion with the best actual time.

6.EPILOGUE:

Unforeseen Situations:

1) When we were working on this project initially, we were unclear about the actual and predicted paths, adjustment of flows. But, after working through it for several days, we got new ideas of how to proceed with it and implemented it in a better approach.
2) We stored the path in the adjacency matrix and string for the shortest path. It was not efficient so we used multidimensional array.
3) We got the shortest distance but to find the path through which shortest distance is calculated is problem we faced, but we resolved it by taking separate a path array for each node which stores the previous nodes through which node is reached.

Lessons Learnt:

1) Considering our last experience, this time we started working on the project from the day it was assigned to us.

2) We learned different types of data structures. But there is no any one specific data structure that is best for all the cases.

One more opportunity:

If we got one more opportunity to do this project again, we would try our best to optimize the output. Beyond that, we would go a step ahead and design a features that would be helpful in real world, like load balancing given enough time to do so. Also, we would think of an approach better than this and try to make it differently, as we always want to make our solution better than the previous one.

Lessons for future students:

Start early have fun, clarify all doubts as early as possible.

7.APPENDIX:

Program Listing:

The input is given in the format:

First line:<<no. of vertices, start_node, end_node>
All other lines:<<edge, x, y, type, alpha, beta>>

Input 3:

```
12, 1, 12
E,1,2,1,24.000000,22.000000,
E,1,3,4,0.100000,13.000000,
E,2,4,1,25.000000,13.000000,
E,2,5,1,30.000000,13.000000,
E,3,4,4,0.111111,11.000000,
E,3,5,2,16.000000,28.000000,
E,4,6,4,0.100000,15.000000,
```

```
E,4,7,1,25.000000,15.000000,
E,5,6,2,19.000000,37.000000,
E,5,7,3,0.047619,9.000000,
E,6,8,5,22.000000,1.000000,
E,6,9,5,23.000000,1.000000,
E,7,8,3,0.045455,5.000000,
E,7,9,5,22.000000,1.000000,
E,8,10,2,12.000000,28.000000,
E,8,11,5,28.000000,1.000000,
E,9,10,1,22.000000,1.000000,
E,9,11,1,30.000000,1.000000,
E,10,12,1,23.000000,1.000000,
E,11,12,5,28.000000,1.000000,
```

Output 3:

```
Number Of Nodes:12
Number Of Edges:20

mean_value criteria(Expected Value): from node 1 to node 12 is :111.04555       Path: ->1->3->4->6->8->10->12
no.of hops is 6
participating Edges:(1,3)(3,4)(4,6)(6,8)(8,10)(10,12)

Optimist_value criteria: from node 1 to node 12 is :77.381195   Path: ->1->3->4->6->8->10->12
no.of hops is 6
participating Edges:(1,3)(3,4)(4,6)(6,8)(8,10)(10,12)

Pessimistic_value criteria: from node 1 to node 12 is :131.80971        Path: ->1->2->4->6->8->10->12
no.of hops is 6
participating Edges:(1,2)(2,4)(4,6)(6,8)(8,10)(10,12)

Double Pessimist_value criteria: from node 1 to node 12 is :143.1       Path: ->1->2->4->7->9->10->12
no.of hops is 6
participating Edges:(1,2)(2,4)(4,7)(7,9)(9,10)(10,12)

stable_value criteria: from node 1 to node 12 is :0.0020661156  Path: ->1->2->4->7->9->10->12
no.of hops is 6
participating Edges:(1,2)(2,4)(4,7)(7,9)(9,10)(10,12)

own_value criteria: from node 1 to node 12 is :0.045454543      Path: ->1->2->4->7->9->10->12
no.of hops is 6
participating Edges:(1,2)(2,4)(4,7)(7,9)(9,10)(10,12)
=============================================================================================================
U - SD       |         U     |        U + SD      |      U + 2*SD     |      C^2       |  OWN U*C^2   |
=============================================================================================================
77.381195    |    111.04555    |    144.7099    |    178.37424    |    2.051538    |    22.443344
77.381195    |    111.04555    |    144.7099    |    178.37424    |    2.051538    |    22.443344
102.48119    |    117.145454    |    131.80971    |    146.47397    |    1.66    |    14.045454
140.1    |    141.1    |    142.1    |    143.1    |    0.0020661156    |    0.045454543
140.1    |    141.1    |    142.1    |    143.1    |    0.0020661156    |    0.045454543
140.1    |    141.1    |    142.1    |    143.1    |    0.0020661156    |    0.045454543

Eexecution Time: 14 ms
```

Input 4:

```
12, 1, 12
E,1,2,5,25.000000,1.000000,
E,1,3,3,0.043478,10.000000,
E,2,4,5,30.000000,1.000000,
E,2,5,5,29.000000,1.000000,
E,3,4,2,13.000000,29.000000,
E,3,5,3,0.045455,5.000000,
E,4,6,2,14.000000,28.000000,
E,4,7,2,19.000000,37.000000,
```

```
E,5,6,3,0.050000,10.000000,
E,5,7,5,26.000000,1.000000,
E,6,8,3,0.045455,9.000000,
E,6,9,4,0.166667,16.000000,
E,7,8,3,0.050000,5.000000,
E,7,9,3,0.041667,9.000000,
E,8,10,2,23.000000,35.000000,
E,8,11,3,0.047619,8.000000,
E,9,10,2,19.000000,37.000000,
E,9,11,1,26.000000,37.000000,
E,10,12,4,0.166667,16.000000,
E,11,12,1,29.000000,16.000000,
```

Output 4:

```
Number Of Nodes:12
Number Of Edges:20

mean_value criteria(Expected Value): from node 1 to node 12 is :136.99973      Path: ->1->3->5->6->8->11->12
no.of hops is 6
participating Edges:(1,3)(3,5)(5,6)(6,8)(8,11)(11,12)

Optimist_value criteria: from node 1 to node 12 is :114.6077    Path: ->1->2->4->7->9->10->12
no.of hops is 6
participating Edges:(1,2)(2,4)(4,7)(7,9)(9,10)(10,12)

Pessimistic_value criteria: from node 1 to node 12 is :165.04143       Path: ->1->2->4->6->9->11->12
no.of hops is 6
participating Edges:(1,2)(2,4)(4,6)(6,9)(9,11)(11,12)

Double Pessimist_value criteria: from node 1 to node 12 is :177.08287   Path: ->1->2->4->6->9->11->12
no.of hops is 6
participating Edges:(1,2)(2,4)(4,6)(6,9)(9,11)(11,12)

stable_value criteria: from node 1 to node 12 is :0.5770911     Path: ->1->2->4->6->9->11->12
no.of hops is 6
participating Edges:(1,2)(2,4)(4,6)(6,9)(9,11)(11,12)

own_value criteria: from node 1 to node 12 is :12.209692        Path: ->1->2->4->6->9->11->12
no.of hops is 6
participating Edges:(1,2)(2,4)(4,6)(6,9)(9,11)(11,12)
```

U - SD	U	U + SD	U + 2*SD	C^2	OWN U*C^2
28.999996	136.99973	244.99944	352.99915	5.0	107.99972
114.6077	156.9998	199.3919	241.784	2.077091	53.7095
140.95856	152.99998	165.04143	177.08287	0.5770911	12.209692
140.95856	152.99998	165.04143	177.08287	0.5770911	12.209692
140.95856	152.99998	165.04143	177.08287	0.5770911	12.209692
140.95856	152.99998	165.04143	177.08287	0.5770911	12.209692

```
Execution Time: 13 ms
```

REFERENCES:

https://www.tutorialspoint.com/java/java_innerclasses.htm

https://rosettacode.org/wiki/Dijkstra%27s_algorithm#Java

http://www.vogella.com/tutorials/JavaAlgorithmsDijkstra/article.html

https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm