Predicting Shortest Path with congestion

Introduction:

All of the transport including air, road and the computer communications networks are modeled as graphs. The best examples for this case, networks, the usage of google maps to find the paths. In these type of graphs, nodes are the places or cities, edges are the distances between the nodes or the time taken to travel between any two nodes. This method of using locations as nodes in the graph is used for various applications. One such primary application in finding the shortest path between the nodes. Now, for example, when planning a car trip, say from city **a** to a city **b,** people would prefer to take the shortest path. The shortest path by definition means the least time taken to travel from city **a** to city **b.** But, in real world the roads include certain traffic, or there may be traffic jam in the actual shortest path. Since, the roads are filled with traffic i.e. there is congestion in the path, there is a delay in the path, the predicted path should take into account the flow in the path and also the capacity of all the edges.

Assumptions:

Here the Edge matrix 'E' containing all the edge weights, E[i,j] represents the time taken to travel between any to nodes. The matrix 'C' is the capacity matrix, C[i,j] represents the maximum number of cars that can flow through an edge. The matrix 'F' is the flow matrix, F[i,j] which is the number of cars per hour on the entire path from i to j nodes. The matrix 'L' is the load matrix, each entry in the **L** matrix represents the total load on each edge. The load matrix is calculated by taking the Flow matrix **F** and the shortest path which is calculated by running Floyd warshall algorithm on the Edge matrix **E**. The time delay matrix 'G' is calculated by

G[i,j] = $\left(\frac{C[i,]+1}{C[i,j]+1-L[i,j]}\right)$ E[i,j] for **L[i,j] <= C[i,j]** and else **∞**

The actual and the predicted path lengths are to be calculated. The actual path length is the path without any congestion.

Implementation:

The programming language used is **Java**. The reason that **Java** is used for the implementation is the advantage that it has. **Java** is platform independent. **Java** is more explicit and better in indentation.

Approach 1: wait or reroute technique

The approach deals with the rerouting technique. The algorithm is designed to implement a better way rather than just going for the shortest path technique. The approach is designed in a way when the path is congested either choose a wait time (let other vehicles pass) or reroute through another path whichever is less. The basic algorithm is Floyd warshall algorithm. The change here is, whenever the load L exceeds the Capacity C at an edge E[i,j], we compute delay with the load equivalent to the capacity of that edge and then calculate delay for the remaining load with two different capacities this is called L-C delay time. One is the residual capacity and other is the full capacity of the system. The delay time obtained for additional load with residual capacity is calculated and Floyd Warshall is applied implemented on this delay matrix to obtain the alternate routes and respective delays called alternate route delay time. Now we calculate delay time for the additional load on full capacity of system. This is called additional wait time. We compare the alternate route delay time with sum of additional wait time and L-C delay time. We replace infinity with the least of the two times and calculate path delays.

Approach 2: Iterate G

This approach uses the same techniques in the calculation of the Load and the delay time matrix G. The approach also uses the basic algorithm of Floyd warshall to calculate the shortest paths. The shortest paths, load are calculated from the given edge matrix E and the Flow matrix F. The time delay matrix is G is calculated in the similar fashion with the same condition. We then use the obtained G matrix and then again run the Floyd Warshall algorithm using the already computed G matrix in place of E matrix. We now get the new alternate shortest paths for indices where Load L[i,j] > C[i,j]. Using the new alternate shortest paths, we calculate the alternate load and the delay times.

Approach 3: Absolute shortest path

This approach finds absolute shortest path, the shortest path keeping in mind about only one car choosing the shortest path. Not taking into consideration, the congestion in the path and suffer the consequences method as suggested in the project material.

Approach 4: Shortest path when source and destination are given

This is the approach that is designed but not implemented due to time limitations. The approach uses the same Edge and Flow matrices. The approach uses **Dijkstra's** algorithm to find the shortest path. The shortest path is then used to find the Load and the time delay G. If the path is filled with congestion, alternate path i.e. rerouting is done at the path where there is congestion.

Methods/routines/data structures used:

Double Ended queue (Deque) [1] – when modelling any kind of real world waiting line, especially entities like cars which differ from the start to end of line. So these data structures are used for these.

Array[][]- for defining matrices

ArrayList<point> - returning the index values.

ArrayList<float[][]> - returning the matrices of two dimensions from different functions.

Validation:

We used the example that is provided in the project material and implemented as the initial test case. The results obtained were satisfying and matched with the results provided in the project materials. We tried out various test cases for all the three algorithms. The input files that are provided **conjestedpathinput1.txt, conjestedpathinput2.txt** and also various multi-node input containing 10 to 75 nodes are tested on the three algorithms. Approach 1 and approach 3 handled as many as 75 nodes and hopefully will run for more than 75 nodes. The approach 2 ran smoothly only for nodes <= 35. Here are the screens for the output for the test case input taken from the example provided in the project material:

[1] http://stackoverflow.com/questions/3880254/why-do-we-need-deque-data-structures-in-the-real-world

[2] http://www.gitta.info/Accessibiliti/en/html/Dijkstra_learningObject1.html