High Level Programming Project Report

Group 10

Delay Tolerant Networks

Description:

In several instances, network communications can occur with relevant delays, as for extra-terrestial communications or for supervised monitoring networks. In such network deciding which node can communicate with which other node and when is extremely complicated issue (a np-completed scheduling problem).Therefor in this project we are trying to find the shortest path from a node to another one by using common used algoritims .

Network:

Using the library networkx to create weighted network by the delays as edges we created a simple network to test our algorithms as the following graph :

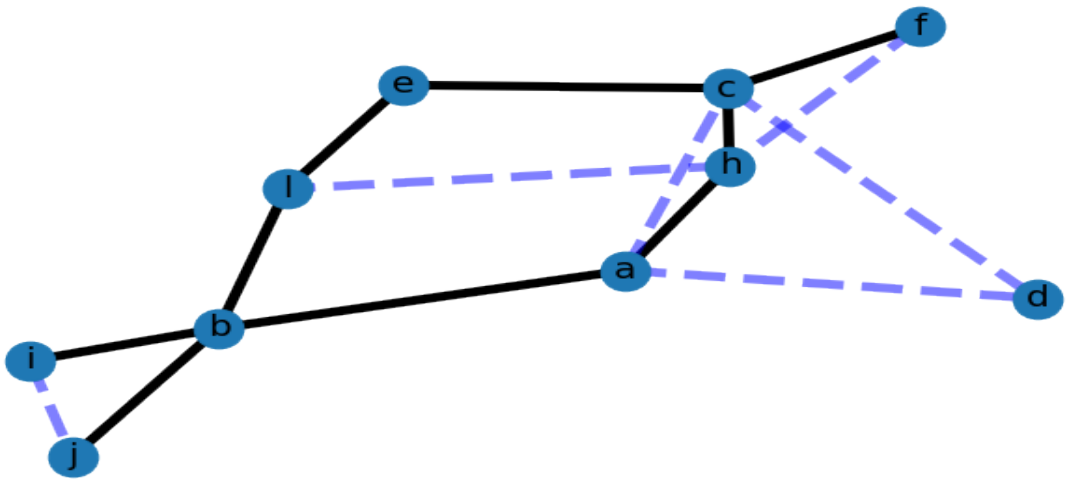

Figure 1: Simple network with wieghted edges

At this point we tested our four following algorithms :

1- Dijkstra.

2- Bell Ford .

3- Johnson .

4- Floyed Warshall.

Dijkstra:

Dijkstra's Algorithm stands out from the rest due to its ability to find the shortest path from one node to every other node within the same graph data structure. This means, that rather than just finding the shortest path from the starting node to another specific node, the algorithm works to find the shortest path to every single reachable node – provided the graph doesn't change.

The algorithm runs until all of the reachable nodes have been visited. Therefore, you would only need to run Dijkstra's algorithm once, and save the results to be used again and again without re-running the algorithm again, unless the graph data structure changed in any way.

In the case of a change in the graph, you would need to rerun the graph to ensure you have the most updated shortest paths for your data structure.

Let's take our network example from above, if you want to go from a node to another one in the shortest way possible, but you know that some ways of transmission are heavily delyed, blocked, and so on, when using Dijkstra, the algorithm will find the shortest path while avoiding any edges with larger weights, thereby finding you the shortest transmision.

Applying this algorithm on our network we get the following result:

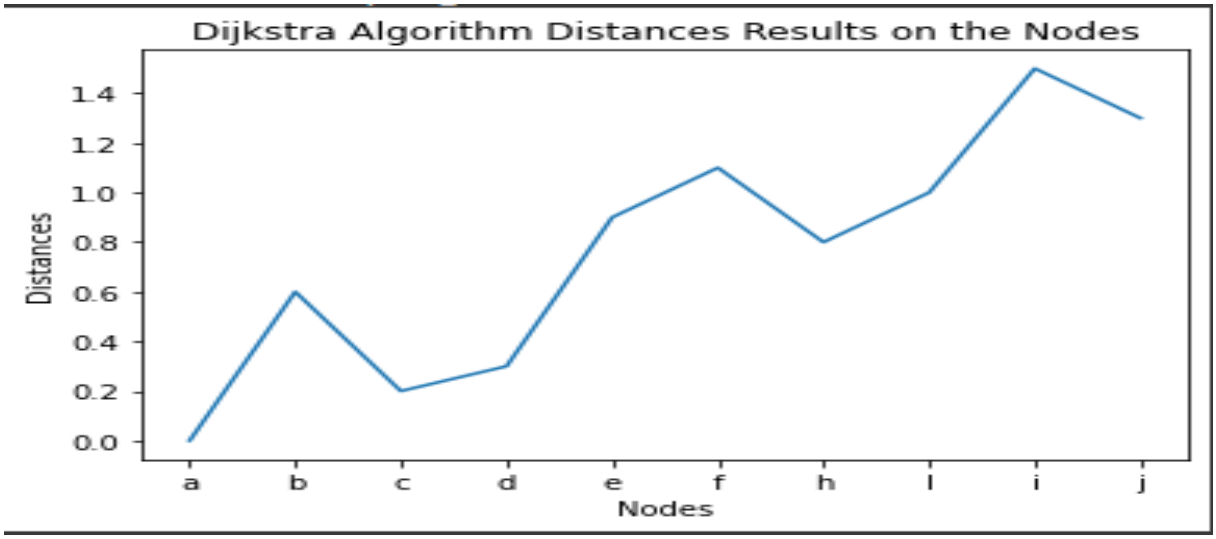

Figure 2: Dijkstra results

With (a) as the starting node for our algorithm and sorting all the other nodes respect this node.At this point we can see the time comlixty that our algorithm needed to calculate the results above. At this point we can see the time comlixty that our algorithm needed to calculate the results above which result in $t = 0.000607s$

Bell Ford:

Similar to Dijkstra's algorithm, the Bellman-Ford algorithm works to find the shortest path between a given node and all other nodes in the graph. Though it is slower than the former, Bellman-Ford makes up for its a disadvantage with its versatility. Unlike Dijkstra's algorithm, Bellman-Ford is capable of handling graphs in which some of the edge weights are negative.

It's important to note that if there is a negative cycle in which the edges sum to a negative value – in the graph, then there is no shortest or cheapest path. Meaning the algorithm is prevented from being able to find the correct route since it terminates on a negative cycle. Bellman-Ford is able to detect negative cycles and report on their existence.

Applying this algorithm on our network we get the following result:

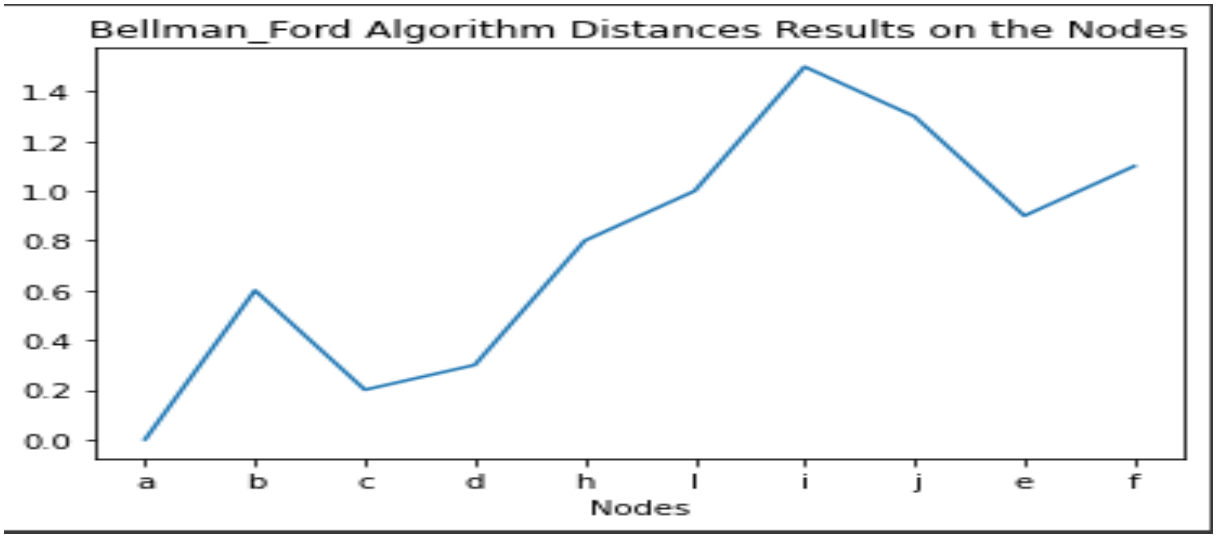

Figure 3: Bell Ford results

Always with (a) as the starting node for our algorithm and sorting all the other nodes respect this node. At this point we can see the time comlixty that our algorithm needed to calculate the results above which result in $t = 0.0013s$

We can easily see that Bell Ford algorithm got the same distances for nodes as the Dijkstra one but with different order due to the algorithm dynamic .

Having the fact the Bell Ford algorithm is much slower than Dijkstra and calculating the same distances. Therefore, it is highly convenient to use it in case there are no negitive delays in the network .

Johnson:

Johnson's algorithm works best with sparse graphs – one with fewer edges, as it's runtime depends on the number of edges. So, the fewer edges, the faster it will generate a route.

This algorithm varies from the rest as it relies on two other algorithms to determine the shortest path. First, it uses Bellman-Ford to detect negative cycles and eliminate any negative edges. Then, with this new graph, it relies on Dijkstra's algorithm to calculate the shortest paths in the original graph that was inputted.

Applying this algorithm on our network we get the following result:

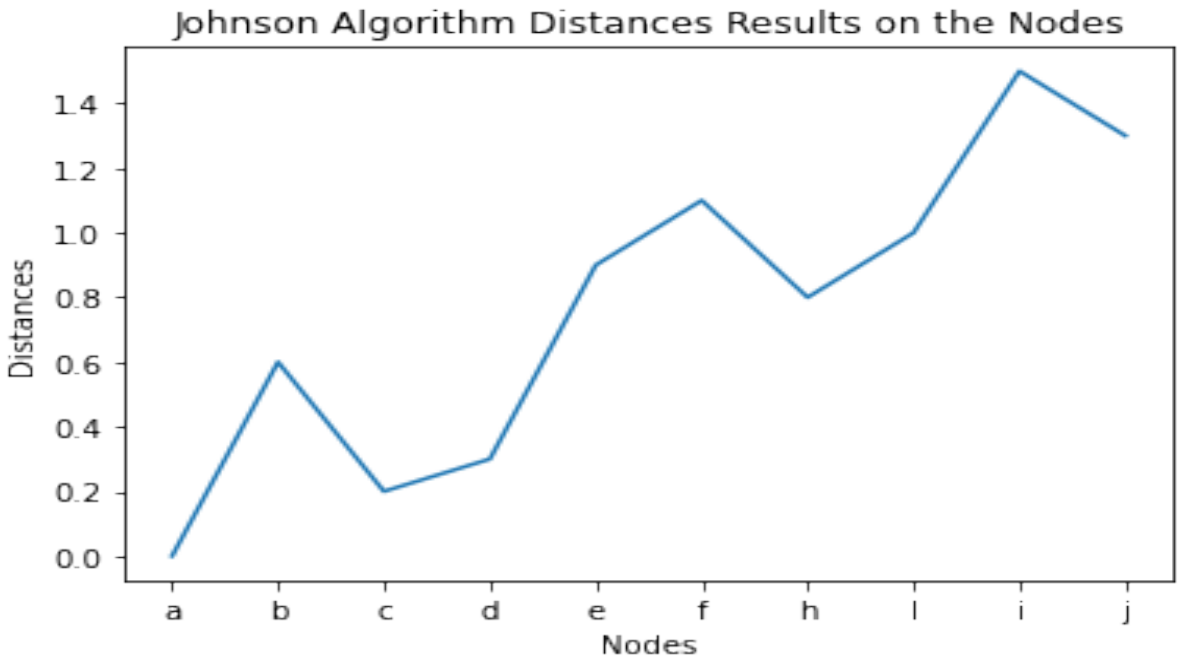

Figure 4: Johnson results

Ploting even just the case of (a) as the starting node for our algorithm and sorting all the other nodes respect this node. At this point we can see the time comlixty that our algorithm needed to calculate the results above which result in $t = 0.01s$.

Floyed Warshall:

The Floyd-Warshall stands out in that unlike the previous two algorithms it is not a single-source algorithm. Meaning, it calculates the shortest distance between every pair of nodes in the graph, rather than only calculating from a single node. It works by breaking the main problem into smaller ones, then combines the answers to solve the main shortest path issue.

Floyd-Warshall is extremely useful when it comes to generating routes for multi-stop trips as it calculates the shortest path between all the relevant nodes. For this reason, many route planning software' will utilize this algorithm as it will provide you with the most optimized route from any given location. Therefore, no matter where you currently are, Floyd-Warshall will determine the fastest way to get to any other node on the graph.

Applying this algorithm on our network we get the following result:

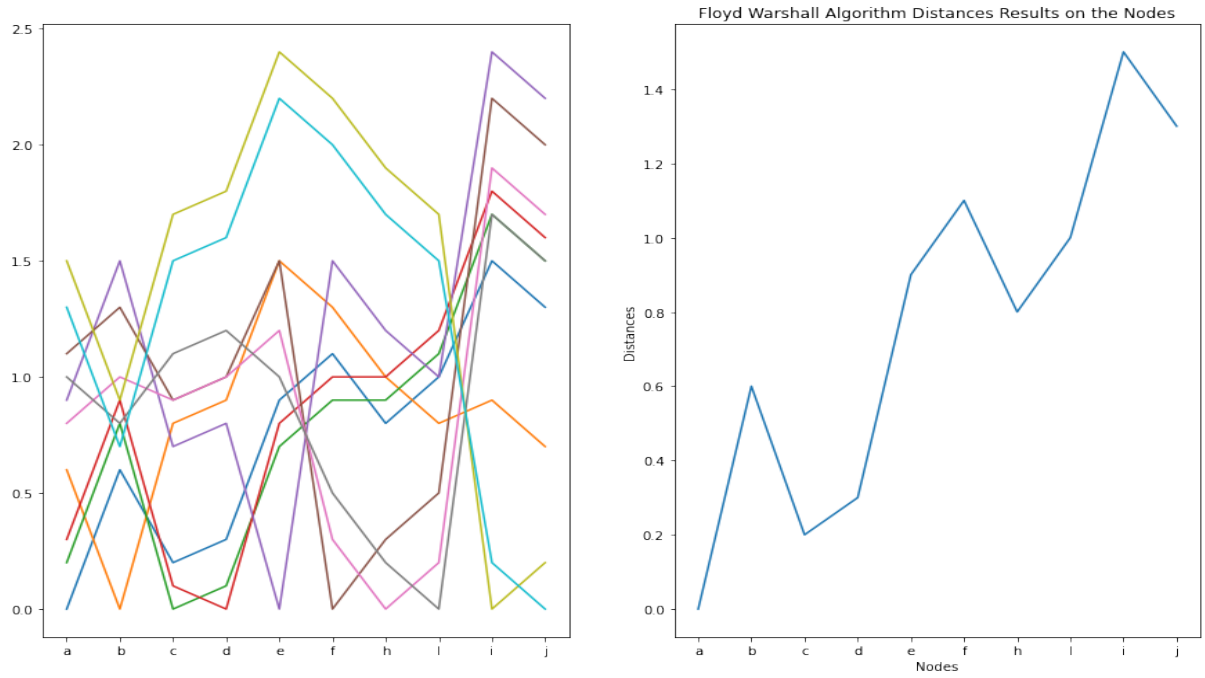

Figure 5: Floyed Warshall results

Ploting even just the case of (a) as the starting node for our algorithm and sorting all the other nodes respect this node. At this point we can see the time comlixty that our algorithm needed to calculate the results above which result in $t = 0.01s$.

Having the fact the Bellman Ford algorithm is much faster than Floyed Warshall and calculating the same distances in case of one node as a starting point. Therefore, it is highly convenient to use Bellman Ford in case there are no negitive cycles in the network .

Optimization:

As we can see from the above grahps and results that we have 3 different kinds of networks, one is with all positive values of delays and one with negitive and poistive values of delays but not negitive cycles and the last one have negitive and poistive values of delays and negitive cylces.

For the first kind of only positive delays we can see easliy as all algorithms calculate almost the same results for distances. Therefore Dijkstra's algorithm is the best as it the most effecient due to the much less time that it takes to calculate the results.

For the second kind of network were we have negitive and poistive values of delays but not negitive cycles we can see can not use Dijkstra's algorithm and for the other 3 we see that Bellman Ford is fastest one if we want to have the results just to respect to one node as input.

For the third kind were we have negitive and poistive values of delays and negitive cylces we can see easily that Johnson algorithm is the only one able to deal with this kind of networks from the 4 algorithms that we used .

Therefore, Our idea was to optimize the algorithm and do a mix that use the best algorithm for each case, Dijkstra's for poistive delays networks and Bellman Ford for negitive delays networks that dose not include negative cycles and Johnson for negative networks that include negative cycles .

Plotting our algorithm against Johnson one in the first case in which we have just positive delays networks we have the following results :

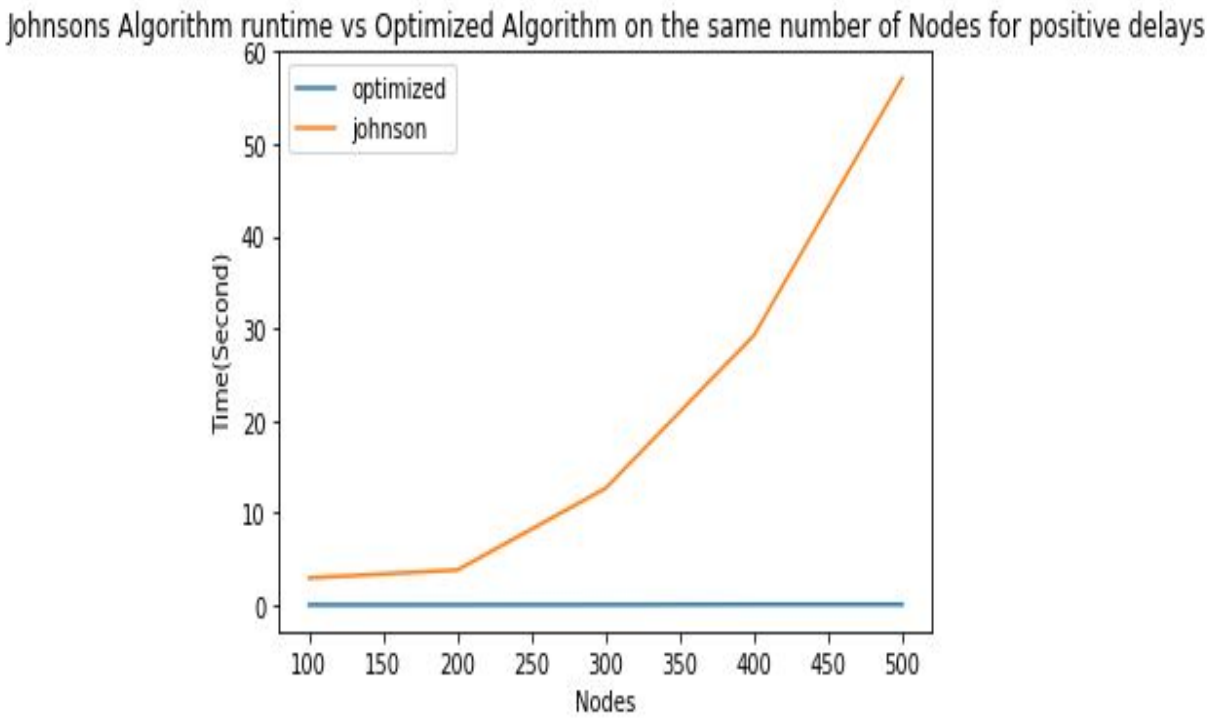

Figure 6: For positive networks delays

And for the ones with negative delays but not negative cycles we have :

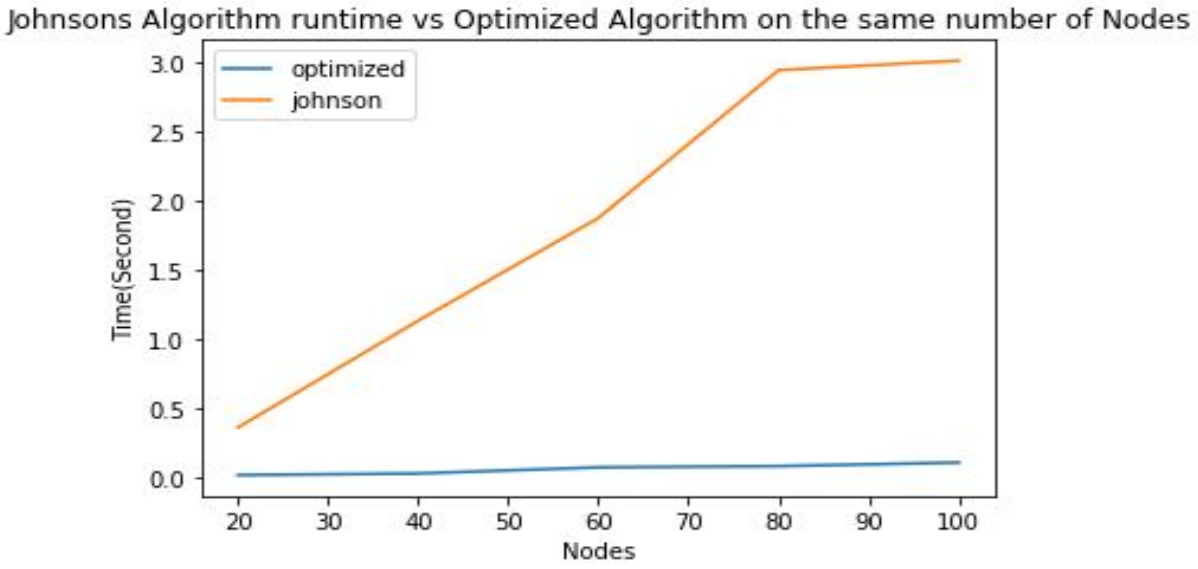

Figure 7: For networks having negative delays

Note that here we used smaller networks as it hard in bigger random networks to have negative delays that not leds to negative cycles .

At the end plotting for the last kind of networks in which we have negative cycles we have the following graph :

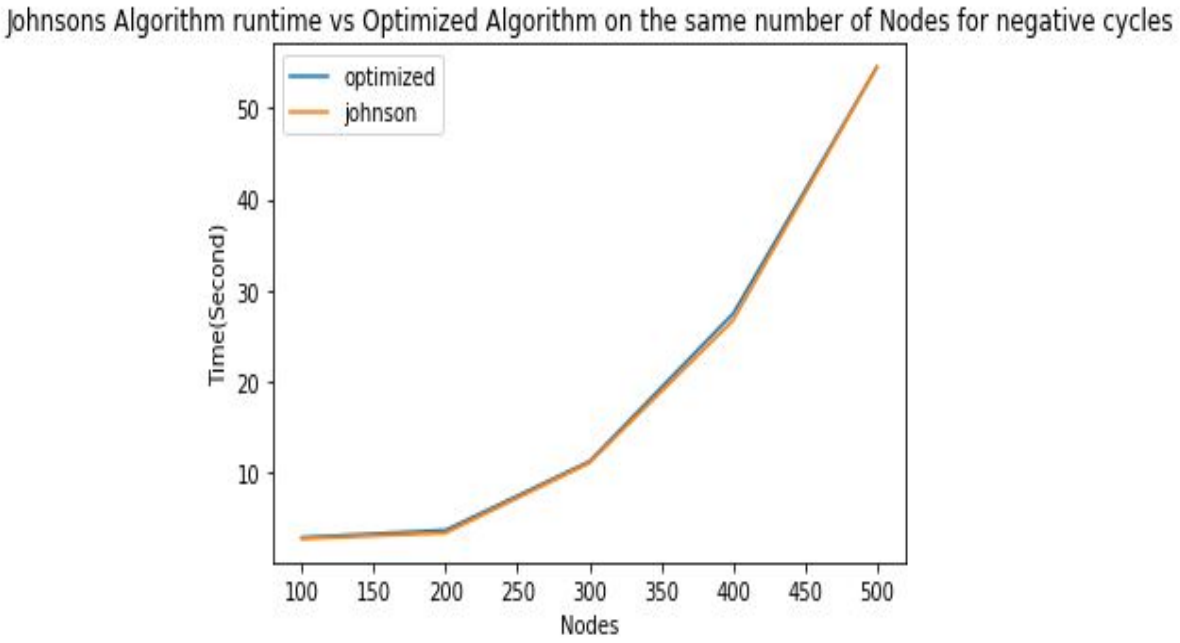

Figure 8: For networks having negative cycles

Conclusion:

We can see easly from the above graphs is using a mix between the algorithms is much more convenient for random networks that can include negative or positive delays and due the big difference in run time between the other 2 algorithms used and the Johnson we can see that the difference between the optimized one and the Johnson in negative cycles can be neglected due the bigger time needed by the Johoson algorithm .