**P****roject Report**

**Implementation and Evaluation of Graph Theory Algorithms**

**Design and Analysis of Algorithms**

**Submitted by:**

Muhammad Mustafa Manga 17K-3795

Muhammad Hassan Ahmed 17K-3654

Section: E

Date of Submission: December 1, 2019

**Submitted to:**

Dr. Muhammad Atif Tahir

# Abstract:

Graphs hold a degree of great importance in Computer Sciences. Several Real-life Problems can be morphed into Graphs Data Structure and their solutions can be computed through Graph Theory. Some Examples include Network Analysis, Geo-Location Navigation Systems etc. This Project Utilizes several Graph Theory Algorithms to calculate Minimum Costs and Clustering Co-effecient on a provided netsim Network Output.

# Introduction:

A graph is a non-linear data structure, which consists of vertices(or nodes) connected by edges(or arcs) where edges may be directed or undirected. Given the provided netsim input files. We utilized the following Algorithms to calculate minimium costs:

## Kruskals:

Kruskal's algorithm is a minimum-spanning-tree algorithm which finds an edge of the least possible weight that connects any two trees in the forest. It is a greedy algorithm in graph theory as it finds a minimum spanning tree for a connected weighted graph adding increasing cost arcs at each step.

## Prims:

Prim's algorithm is a greedy algorithm that finds a minimum spanning tree for a weighted undirected graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized.

## Dijkstra:

Dijkstra's algorithm to find the shortest path between a and b. It picks the unvisited vertex with the low distance, calculates the distance through it to each unvisited neighbor, and updates the neighbor's distance if smaller. Mark visited (set to red) when done with neighbors.

## Bellman Ford:

The Bellman–Ford algorithm is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph.

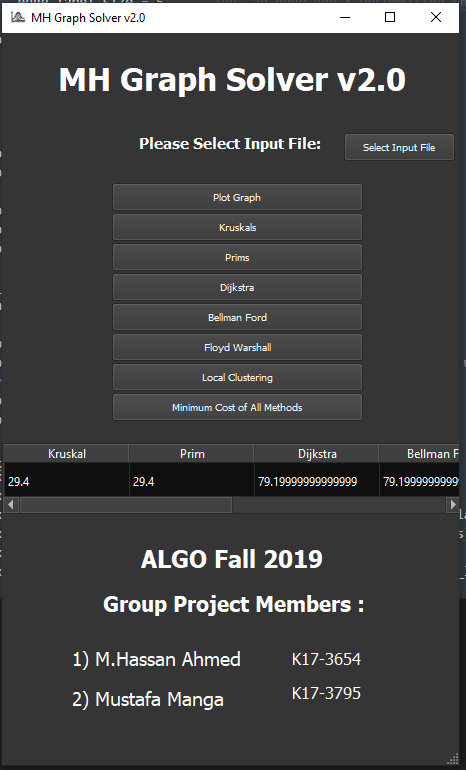
## Floyd Warshall:

the Floyd–Warshall algorithm is an 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 shortest paths between all pairs of vertices.

## Local Clustering Co-effecient:

A clustering coefficient is a measure of the degree to which nodes in a graph tend to cluster together. The local Clustering Co-effecient gives an indication of the embeddedness of single nodes.

# Proposed System:



We used PyQt5, A Python GUI Library, to design the User Interface for our ‘MH Graph Solver’. It takes the provided netsim input files, parses them and plots them using Networkx and Matplotlib.

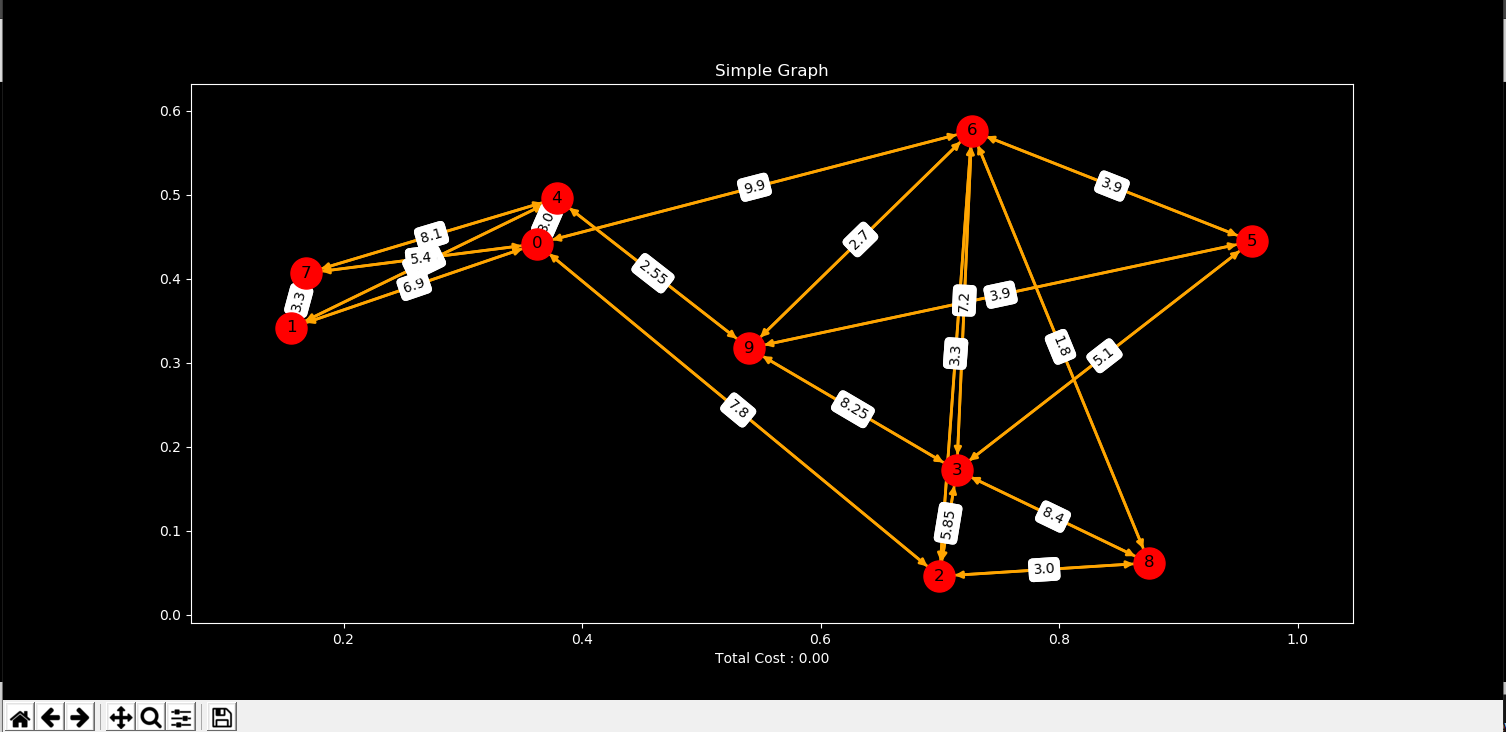
We first Parse the given netsim files and insert into our networkx graph model that will help us manage the Graph Data Structure. The graph object is then utilized by different functions to calculate Minimum Cost based on required Algorithms.

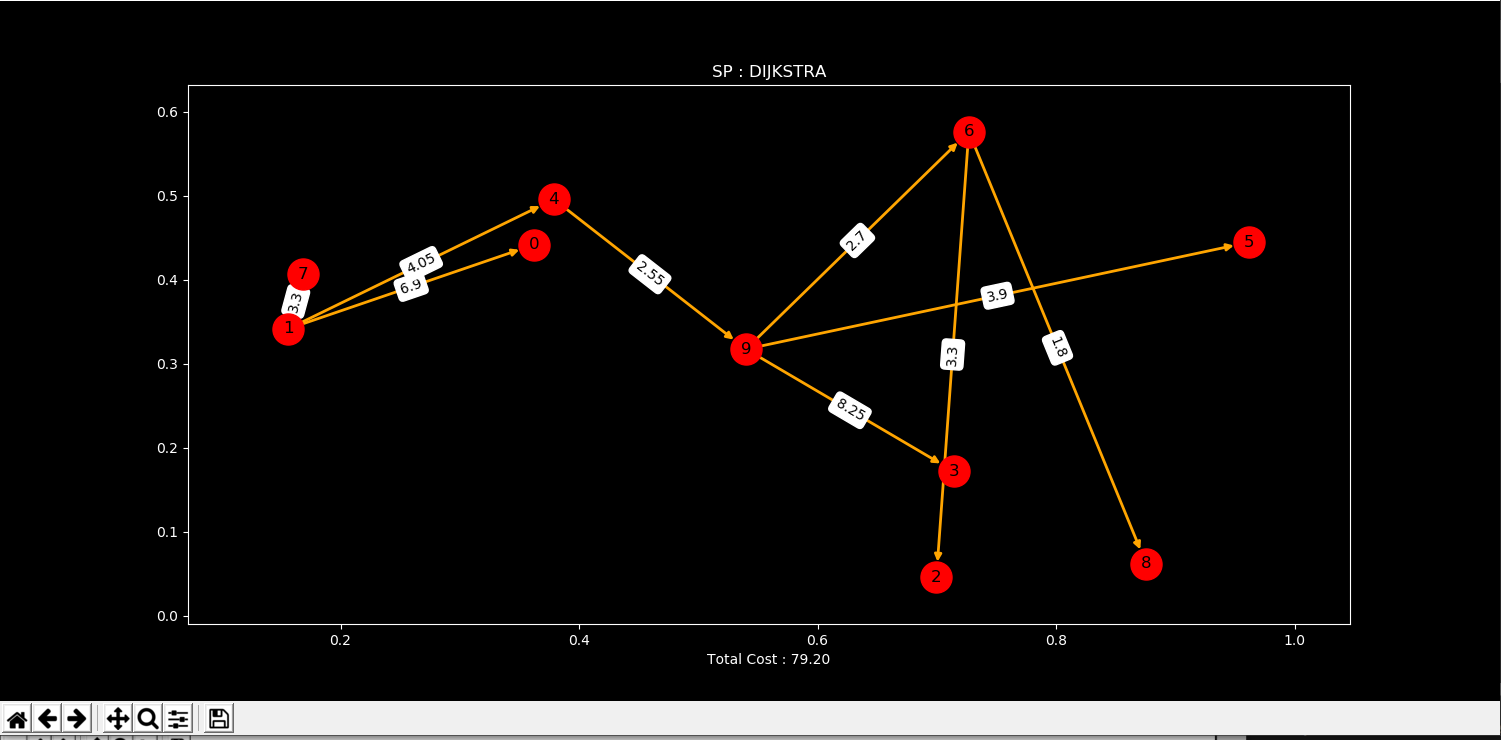
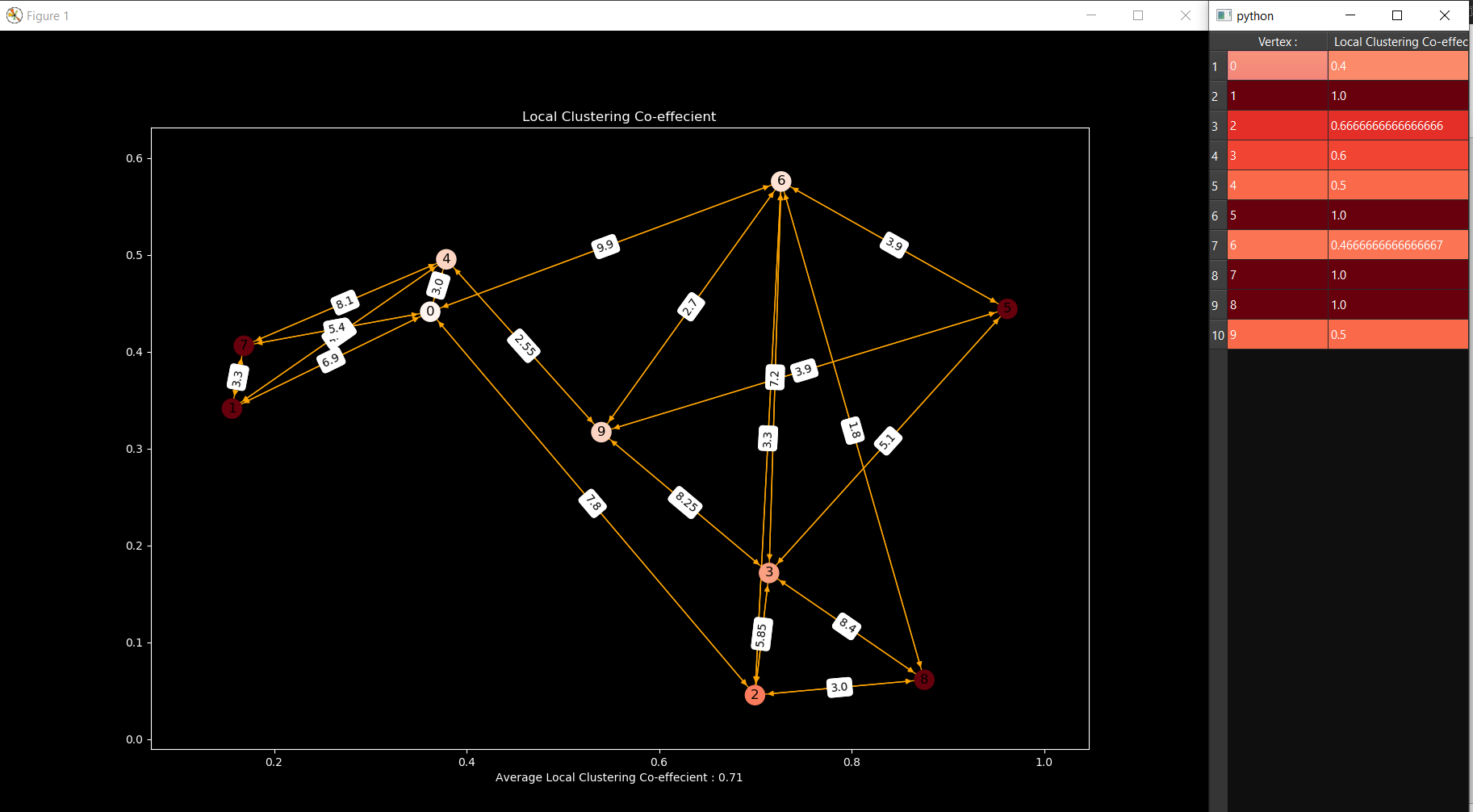
We can display output of all above mentioned Algorithms. We can also Display Total Minimum Cost of all Methods in a simple table.

In case of Local Clustering Co-efficient, We color coded the vertex color in terms of the Local Clustering Co-effecient.

Higher the Local Clustering Co-efficient of vertex, Darker the color of the respective vertex.

A list of Local Clustering Co-efficient for all vertices is also generated.





# Experimental Setup:

We used the given netsim inputs files. Parsed them and inserted into our Graph Data structure. We then used the Implementations of above mentioned Algorithms wo calcualte Minimum Costs and Display their relative Graphs.

The program requires Python to be installed, Run the following Commands to install Dependencies:

pip install pipenv

pipenv install

Run The following command to start the Virtual Environment:

pipenv shell

Run the Following Command to run the program:

python .\MHGraphCalculator.py

The GUI Program will open, Select the input10.txt from benchmarks folder.   
Click on

Input10.txt produces following minimum cost outputs :

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Input 10 | 29.4 | 29.4 | 79.2 | 79.2 | 79.2 | 0.713333 |

# Results and Discussion:

Table 1: Minimum cost algorithm with given source (Vertex 1)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Benchmark | Prims | Kruskal | Dijkstra | Bellman Ford | Floyd Warshall | Clustering Coefficient (Local Clustering) |
| Input 10 | 29.4 | 29.4 | 79.2 | 79.2 | 79.2 | 0.713333 |
| Input 20 | 69.75 | 69.75 | 182.4 | 182.4 | 182.4 | 0.207857 |
| Input 30 | 102.3 | 102.3 | 234.0 | 234.0 | 234.0 | 0.212777 |
| Input 40 | 161.55 | 161.55 | 441.9 | 441.9 | 441.9 | 0.137936 |
| Input 50 | 163.65 | 163.65 | 539.55 | 539.55 | 539.55 | 0.089142 |
| Input 60 | 231.0 | 231.0 | 1144.95 | 1144.95 | 1144.95 | 0.118779 |
| Input 70 | 247.65 | 247.65 | 950.55 | 950.55 | 950.55 | 0.047040 |
| Input 80 | 294.15 | 294.15 | 807.6 | 807.6 | 807.6 | 0.058749 |
| Input 90 | 342.6 | 342.6 | 1284.9 | 1284.9 | 1284.9 | 0.085634 |
| Input 100 | 364.2 | 364.2 | 1261.2 | 1261.2 | 1261.2 | 0.051601 |

# Conclusion:

Hence our designed program MH Graph Solver, produces solutions for all provided input files. Handles invalid Files and produces minimum cost as required.

# References :

* [https://en.wikipedia.org/wiki/Kruskal%27s\_algorithm](https://en.wikipedia.org/wiki/Kruskal's_algorithm)
* <https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/>
* [https://en.wikipedia.org/wiki/Prim%27s\_algorithm](https://en.wikipedia.org/wiki/Prim's_algorithm)
* <https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/>
* [https://en.wikipedia.org/wiki/Dijkstra%27s\_algorithm](https://en.wikipedia.org/wiki/Dijkstra's_algorithm)
* <https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/>
* [https://en.wikipedia.org/wiki/Bellman%E2%80%93Ford\_algorithm](https://en.wikipedia.org/wiki/Bellman–Ford_algorithm)
* <https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/>
* [https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall\_algorithm](https://en.wikipedia.org/wiki/Floyd–Warshall_algorithm)
* <https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/>
* <https://en.wikipedia.org/wiki/Clustering_coefficient>
* <https://networkx.github.io/documentation/networkx-1.9/index.html>
* <https://matplotlib.org/>
* http://www.centiserver.org/?q1=centrality&q2=Local\_Clustering\_Coefficient
* <https://towardsdatascience.com/graph-algorithms-part-2-dce0b2734a1d>
* [https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithms.cluster.clustering.html#networkx.algorithms.cluster.clustering](https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithms.cluster.clustering.html" \l "networkx.algorithms.cluster.clustering)
* <https://github.com/kfoynt/LocalGraphClustering>