

HEURISTIC ALGORITHMS FOR LOW-COST NETWORK TOPOLOGY DESIGN

Krishna Mohan Pitchikala
Graduate Student
University of Texas at Dallas
Texas, USA

Abstract

This paper focuses on designing a low-cost network using an undirected graph with n nodes. There are two main requirements: 1) each node must be connected to at least three other nodes, and 2) the longest distance between any two nodes must be more than four hops. The cost of the network is calculated based on the actual distances between the nodes, and the goal is to make the total length of all the connections as short as possible. To solve this problem, two algorithms (Degree based and Cost based) are created to find good, practical solutions. These algorithms are then tested on randomly generated sets of nodes. After the tests, the results are compared by looking at how well each algorithm minimizes the network cost, how quickly they run, and how many steps they take to finish. The results are provided in graphical form, and the graphical comparison is made to emphasize the merits and demerits of each approach. This helps with showing which algorithm performs better in which scenarios

Index Terms – Network Topology Design, Undirected Graph, Graph Theory, Random Graph Generation

I. INTRODUCTION

Designing efficient network topologies is a critical task in a wide range of applications, including telecommunications, transportation, and distributed computing. Efficient network topologies mean that the nodes in a network are connected in such a way that the network remains functional, efficient, and cost-effective [1]. We will implement two different heuristic algorithms for this network topology design problem, and experiment with them. Let's start with defining the problem statement.

A. Problem Statement

The aim is to create a network topology which is represented by the undirected graph with n nodes, such that it has the below mentioned properties.

1. It contains all the given nodes.
2. The degree of each vertex in the graph is at least 3, that is, each node is connected to at least 3 other nodes.
3. The diameter of the graph is no more than 4. This means that you can reach any node from any other node in at most 4 steps or hops. You can check this using a shortest path algorithm or by running a breadth-first search from each node. Here, "diameter" refers to

the number of hops between nodes, not the physical distance. Also, this limit means the graph must be fully connected.

4. The total cost of the network topology is as low as possible, where the cost is measured by the total geometric length of all links. This means, you have compute how long each link is geometrically (that is, how far apart are its end-nodes), and then sum it up for all links that exist in the network. This sum represents the total cost that we would like to minimize, under the constraints described above in items 1,2,3. Note: do not confuse the geometric distance with the hop-distance, used in the previous point.

B. Overview

A heuristic is a problem-solving method that uses a practical approach to find a solution that is good enough to solve a problem quickly and meet immediate goals, even if it's not the perfect solution. This approach is often referred to as a "rule of thumb" or "best practice". The major advantage of using a heuristic approach is that it provides a quick and simple solution that is easy to understand and use. Heuristic algorithms are practical and effective for finding fast, workable solutions to planning and scheduling problems [6].

The main downside of the heuristic approach is that it is in most cases unable to deliver an optimal solution to a planning and scheduling problem.

There are many proposed Heuristic algorithms to optimize the solutions for the Network Design problem. Some of them are Branch and Bound, Local Search, Cost based, and Degree based. For this project we concentrate on Cost based and Degree based Heuristic Algorithms. Cost based algorithm is based cost of the connections (a greedy approach that processes edges based on their cost) and Degree based algorithm is based on the degree of the node being encountered.

For both these approaches, we start with a possible solution and improve the solution by removing unwanted edges to make it an optimal solution by using these both algorithms. The initial network topology is obtained as follows:

1. Choose the number of nodes (n) in range [15, 40] in steps of 5.
2. Choose random number in the range [0, n] for both x and y coordinate of the node and add the node to the graph.
3. Add edges to the nodes in the graph to make it a complete graph (all nodes are connected).
4. For each node if the degree is not at least 3, add new edges randomly to make the degree of each node to be 3.
5. For each node remove the extra edges based on the degree or cost of the edge connecting those nodes.
6. For the processed edge, we perform a check to find if there are any nodes that are at depth greater than 4 from that node. In such cases, we add the edge back between these vertices to the graph as the constraint was broken because of such removal. In this way a random initial solution is obtained.
7. The cost of the network obtained is calculated as sum of the distances between each point of the edges in graph. This is simply the geometric distance.

Also, we note the time just before running the algorithm and right after the algorithm is finished

and the difference between these two times gives us the run time of the algorithm. The total cost of the network is calculated by taking each edge and getting the start node and end node from the edge which are nothing but the point in plane represented by x and y coordinate. We find the geometric distance between these two nodes, likewise we do this for all edges and adding up the sum gives us the total cost of the network.

II. ALGORITHM AND PSEUDO CODE

A. Degree Based Algorithm

The edges in the graph generated initially are randomized and hence there is a high chance that nodes have a degree greater than 3. The Degree based algorithm employs a concept according to which in an optimized network nearly all nodes would have a degree 3. Hence, we focus on removing the extra edges on the nodes to see if the graph still holds good under the given constraints.

We start by taking each node and getting all the edges and start by deleting extra edge, while deleting the edges if any of the given constraints fails (degree of each node is at least 3 and diameter of the graph is at most 4), we add the edge back to the network and continue like this until every node in the graph is visited.

Algorithm:

1. Check whether the graph is connected or not using `nx.is_connected()` function
2. Get the list of nodes in the network
3. For each unvisited node if the degree of the node is greater than 3 delete the extra edges
4. Check whether the network after deleting the extra edge still holds all the given degree and diameter constraints.
5. If the given constraints fail when we delete an edge add back the edge to the network.
6. Continue the same process for all edges of a given node.
7. Repeat this until all nodes are visited

Pseudo Code:

```
#check the initial graph is connected or not
nx.is_connected(G)
#Take the initial graph
G = nx.Graph()
#Initialize the visited nodes to an empty list, add the visited nodes to this list
visited_nodes = []
#loop through the graph until every node is visited
For node, degree in graph:
    #If degree is greater than 3 and it is still not visited:
    If degree > 3 and node not in visited_nodes:
        #Mark the node as visited in the
        visited_nodes.append(node)
```

```

#Consider all the edges
edges_list = list(G.edges([node]))
#For each edge perform the removal of the edge
For edge in edge_list.:
    #Remove the edges
    G.remove(edge)
    #Check whether the required constraints are satisfying or not
    If diameter ==4 and degree of each node is atleast 3:
        Continue the process
    #Else if removing the edge resulted in breaking the given constraints
    add the edge back to the network
    G.add_edge(edge)

```

B. Cost Based Algorithm

This algorithm uses greedy approach to minimize the cost. To the initial graph generated, we take all the edges in the descending order of the costs and process each edge such that if removing the edge from the graph does not affect the constraints of the network, we remove the edge and go on like this till all the edges are visited. If removing the edge results in defying the degree and diameter constraints, we add the edge back and move on to the next edge. Thus, in this method the edges with maximum cost are removed first and hence reducing the cost of the graph.

Algorithm:

1. Check whether the graph is connected or not using `nx.is_connected()` function
2. Get the cost/weight of all the edges in the network
3. Sort the edges in the decreasing order of their costs.
4. For each edge in the sorted list of edges remove the edge from the graph and check whether the given diameter and degree constraints still hold good.
5. If the constraints hold good, remove the edge from the graph and continue with next edge in the sorted edge list.
6. If the given constraints fail when we delete an edge add back the edge to the network.
7. Continue the same process for all edges in the sorted list.
8. Repeat this until all edges are visited

Pseudo Code:

```

#check the initial graph is connected or not
nx.is_connected(G)
#Take the initial graph
G = nx.Graph()
#Initialize the visited nodes to an empty list, add the visited nodes to this list
visited_nodes = []
#get the list of edges in descending order of the costs

```

```

costs = nx.get_edge_attributes(G, 'weight')
sorted_costs = dict(sorted(costs.items(), key=operator.itemgetter(1), reverse=True))
sorted_edges_list = sorted(sorted_costs)
#loop through the sorted edges list
For edge in sorted_edges_list:
    #remove the edge for the network
    G.remove_edge(edge)
    #check if the given degree and diameter constraints still hold
    If diameter ==4 and degree of each node is atleast 3:
        continue
    Else:
        #Add back the edge and continue
        G.add_edge(edge)

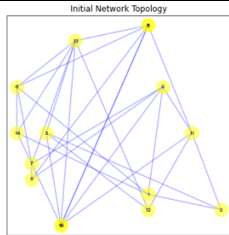
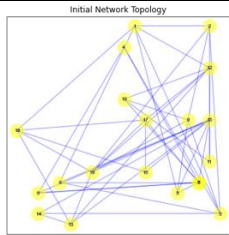
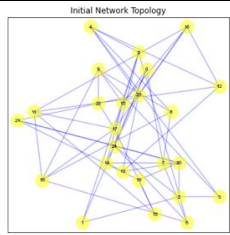
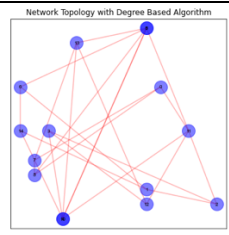
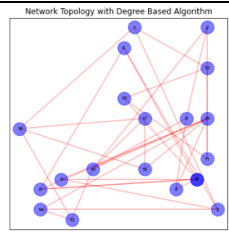

```

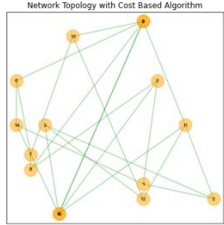
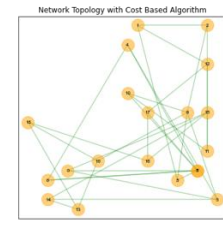
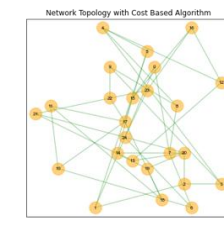
III. RESULTS AND ANALYSIS

A. Results

a. Graphs with n nodes

TABLE - I. Network topology graphs

Number of nodes		15	20	25
Initial network topology				
	Number of edges	30	47	55
	Total cost	250.29	553.95	627.40
Network topology with degree-based algorithm				
	Number of edges	24	34	42
	Total cost	208.19	399.20	476.65

Network topology with cost-based algorithm				
	Number of edges	23	32	42
	Total cost	194.41	345.00	435.89

b. Cost comparison of algorithm-1 and algorithm-2

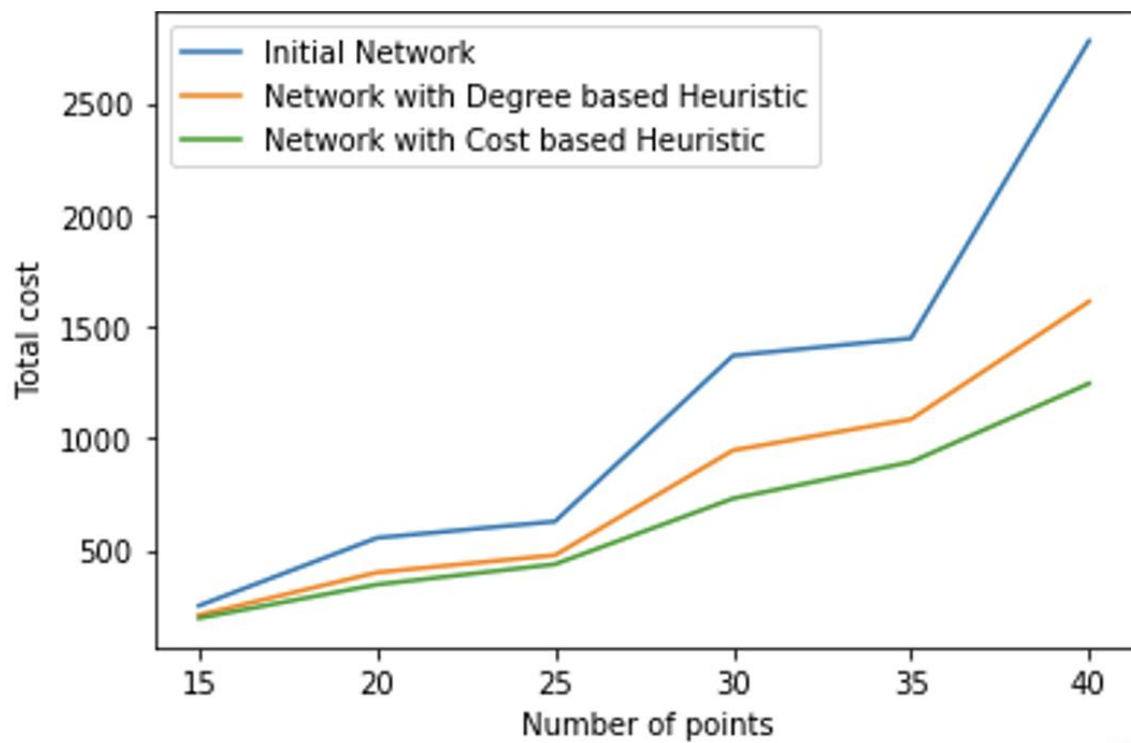


Fig. 1. Cost comparison of Algorithm-1 and Algorithm-2

c. Run time comparison of algorithm-1 and algorithm-2

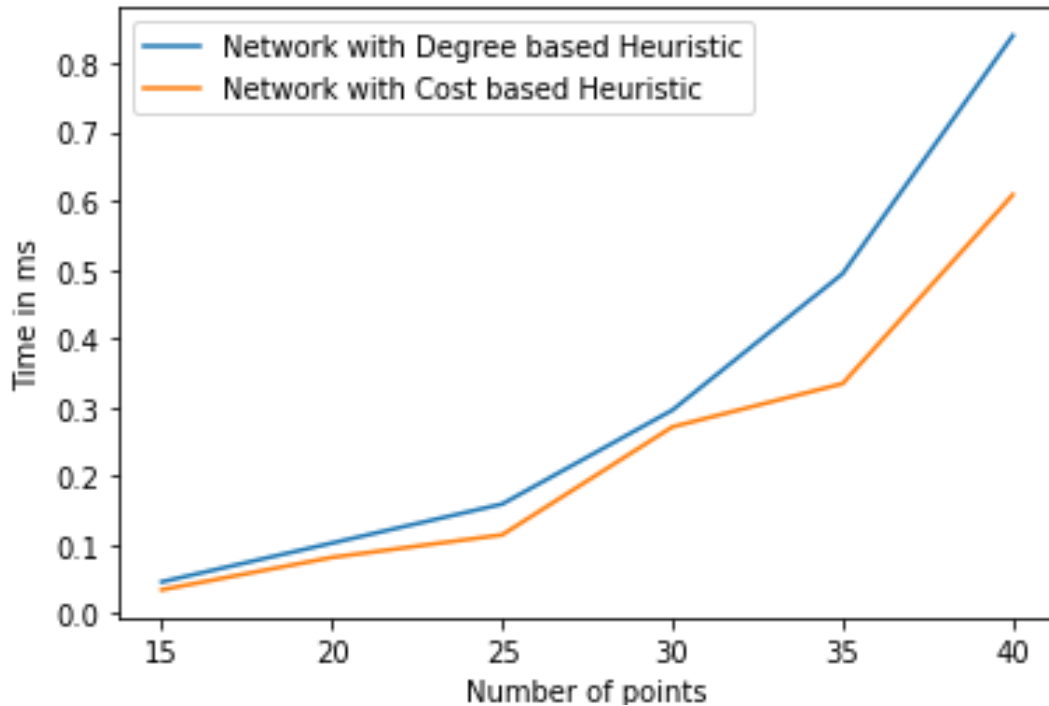


Fig. 2. Run Time comparison of Algorithm-1 and Algorithm-2

B. Analysis

- From Fig-1 it is observed that both degree-based algorithm and cost-based algorithm reduce the total cost by considerable margin.
- Among those two it is seen that the cost based heuristic algorithm has lower total cost compared to degree based heuristic algorithm. This can be supported by the fact that when removing edges in degree-based algorithm, we do not focus on the cost but in case of cost-based algorithm we prioritize cost value of each edge while eliminating the edges in descending order of costs there by keeping low-cost edges.
- Thus, edges having higher cost and do not affect the degree of the node and diameter of the graph when deleted are removed first. Hence, the final graph obtained after removing this kind of edges is the resultant of all the low-cost edges with satisfying the given degree and diameter constraints.

Total Cost of Degree-based heuristic algorithm > Cost based heuristic algorithm

- From Figure-2, it is clear that most of the time degree-based heuristic has less time of execution compared to cost-based heuristic.

Run Time of Degree-based heuristic algorithm > Cost based heuristic algorithm

- Also, with increase in number of nodes in the graph the total cost of the graph and the runtime increases as the number of edges also increase and time taken to process these edges also increase.

Total Cost \propto Number of Nodes in the Graph
Run Time \propto Number of Nodes in the graph

- Considering, Runtime and Total cost of the network we prefer cost-based algorithm over degree-based algorithm to reduce the total cost of the network.

IV. CONCLUSION

The following conclusions are drawn based on the analysis and results:

- Both degree-based and cost-based heuristic algorithms enhance network cost.
- The total cost is consistently lower in the cost-based algorithm compared to the degree-based algorithm.
- The cost-based algorithm is efficient in maintaining a lower overall cost.
- The degree-based algorithm has shorter execution times.
- As the number of nodes in the graph increases:
 - The total cost of both algorithms increases.
 - The runtime of both algorithms increases due to a higher number of edges and additional processing.
- In summary:
 - The degree-based algorithm is faster.
 - The cost-based algorithm is preferred for reducing overall network cost.
 - The cost-based algorithm is the better choice when minimizing network expenses is the primary goal.

REFERENCES

1. <https://www.dnsstuff.com/what-is-network-topology>
2. https://optimization.cbe.cornell.edu/index.php?title=Heuristic_algorithms
3. <https://blog.boot.dev/computer-science/examples-of-heuristics-in-computer-science/>
4. <https://www.sciencedirect.com/science/article/abs/pii/S1084804517303995>
5. https://networkx.org/documentation/latest/_downloads/networkx_reference.pdf
6. https://icrontech.com/blog_item/optimization-vs-heuristics-which-is-the-right-approach-for-your-business/
7. Liu, D., Jing, Y., Zhao, J. *et al.* A Fast and Efficient Algorithm for Mining Top-k Nodes in Complex Networks. *Sci Rep* **7**, 43330 (2017).
8. <https://www.sciencedirect.com/science/article/pii/S2212017312005695>