1 Introduction

Minimum Steiner Tree Problem is an NP-hard optimisation problem in which a connected undirected graph $G = (V, E)$ is given with a set of cost c_e for each edge $e \in E$ and a set of nodes $T \subseteq V$ known as the terminals. The goal is to find a subtree S of G, such that each terminal node in T is included in S and the total cost of tree S is as small as possible. Simply put, the output is a Steiner tree spanning terminals set T.

From the problem formulation, it is interesting to observe that this is a Minimum spanning tree problem if $T = V$, meaning all nodes in G are terminals and hence should be spanned. On the other hand, if $|T| = 2$, this becomes a Shortest Path problem, where the source node and destination node make the two terminals.

2 Approximation Algorithm

Algorithm 1: Basic Distance Network Heuristic

```
Step1: Let $H = (V, E')$ be the metric completion of G
Step 2: Let $H[T]$ be the subgraph of $H$ induced by $T$
Step 3: Return the minimum spanning tree of $H[T]$
Step 4: Find Steiner tree $G' = (V', S)$ by replacing edges of $MST(H[T])$ by the original shortest
  paths in $G$
```

3 Implementation

3.1 Approach

- Step 1:
  The metric completion of G is constructed to be the complete graph of nodes set V with a newly-defined distance function $d : V\text{x}V \to \mathbb{R}$ such that $d(u, v)$ is the distance of the shortest path from u to v in G with respect to the cost function $c(e)$.

- Step 2:
  Instead of computing the full metric H, we directly construct subgraph $H[T]$ containing only terminal nodes in T. In order to do so, we memorise the shortest path data only for paths coming out from a terminal t to another terminal t'.
  Dijkstra algorithm provides the shortest path from a node $v \in V$ to the rest of the $(|V| - 1)$ nodes in G. Hence, we compute Dijkstra for each terminal node $t \in T$, and only save the shortest distance as well as the corresponding paths from t to all terminal $t' \in T$. Note that some of these paths may overlap.

- Step 3:
  Prim's algorithm is used to compute the minimum spanning tree of $H[T]$.

- Step 4:
  Since $H[T]$ is defined with edges that may not exist in G, with respect to a different set of cost d. We need to convert the outputted MST back into a tree in G by replacing its edges with the paths derived by Dijkstra algorithm.

Consider all the paths p_e for each edge $e \in MST(H[T])$. As this original tree is connected and if there is an edge $e_u v$ in the MST then there is a corresponding path connecting u to v in S. Note that the total edge cost of S is therefore smaller or equal the cost of the minimum spanning tree in $H[T]$.

3.2 Data Structure

Only one class *SteinerTree* is used, with information of the given graph instance G. Number of nodes, edges, and terminals are saved upon reading the data file. Details about connected nodes and edge costs are under the form of an adjacency matrix of size $|V| \times |V|$. Indices of the terminal nodes are collected in an array list, which makes the search if one node is a terminal or not more convenient.
After processing the data, the attained adjacency matrix is immediately used when computing Dijkstra shortest path, in order to construct the reduced metric completion. Consider $|T|$ iterations for terminal nodes $t_1, \ldots, t_{|T|}$

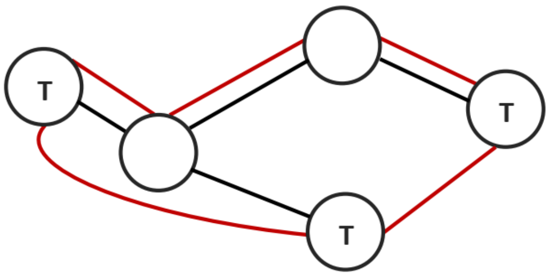
- In each iteration, Dijkstra computes shortest distances from t_i to $v \in G, v \neq t_i$
- Corresponding shortest path can be obtained using recursive tracking method
- The output of Dijkstra method will be of the form ArrayList < ArrayList < Integer >>, in which arrays containing information regarding paths from node t_i to node t_j, $j \in \{t+1, \ldots |T|\}$ are recorded.
  Array list for each t_j is an Array list itself, containing the shortest distance value as the first element, followed by the paths from t_i to t_j.

Outputted data is processed into an adjacency matrix again, cost values are obtained with respect to the shortest distances. Positions are also recorded in order to later track corresponding shortest paths in the array.
This adjacency matrix is inputted to run Prim's algorithm. After the minimum spanning tree is found, edges are replaced by original shortest path. All steps are operated directly on the adjacency matrix.

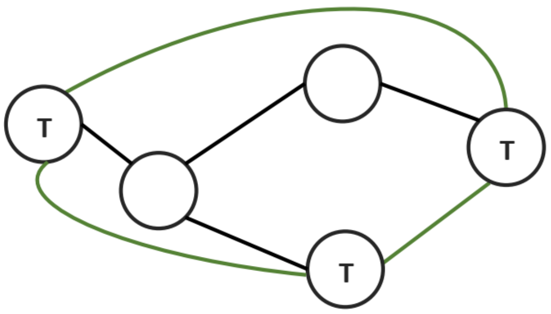
4 Result Analysis

4.1 Solution Quality

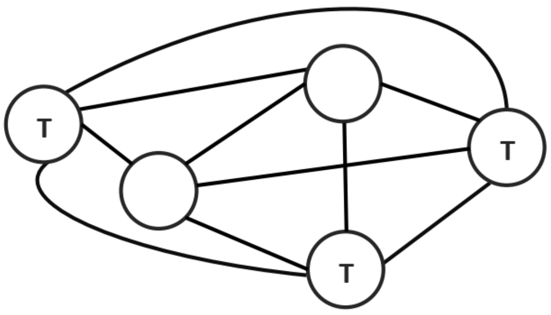
First we consider the optimal solution of the Steiner tree problem. Now doubling all these edges gives this a total cost of 2 times the optimal value. We can use the process of short-cutting by starting at any node and adding the edge to the next node, skipping all nodes that are already visited by previously adding edges. This method can be seen in the image below. Indicated in red.



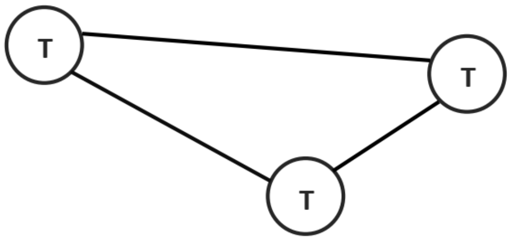
Now we shortcut again by not including all of the non-terminal nodes. This cycle is shown in green in the image below.



Any green or red edge that did not exist previously between 2 nodes is the shortest path between these two nodes. Since the red cycle is shorter than 2 times the optimal solution and the green cycle is shorter than the red cycle, we can say by transitivity that the green cycle is smaller than 2 times the optimal solution.



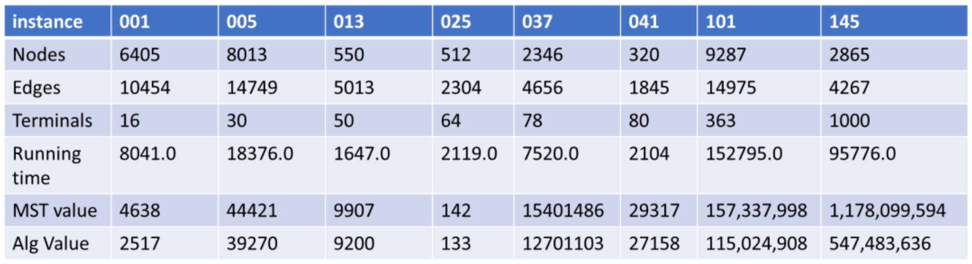
Now we will look at the graph G[R] as described in the question. After connecting all edges using the triangle inequality we get the following graph. All edges in this graph have the value of the shortest path between the nodes it connects.



Now we consider the minimum spanning tree in any graph and show that this is smaller than 2 times the optimal solution of the corresponding minimum path of the Steiner tree. Now any minimum spanning tree in this graph induced by the terminal nodes contains all possible green cycles. Therefor the minimum spanning tree in this graph is smaller or equal to any green cycle minus any of its edges. Since the minimum spanning tree in this graph is smaller or equal to any of the green cycles minus any of it's edges, the minimum spanning tree of this graph is smaller or equal to 2 times the optimal solution of the Steiner tree problem.

- Now that we have found that this minimum spanning tree is a 2-approximation. The result of our MST-value in our program is also a 2-approximation. Since in our program, we backtrack all shortest paths in the metric complete graph and only count the edges used multiple times once, we get an even better approximation. This is still a 2 (specifically $(2 - \frac{2}{|T|}) - approximation$). However we can use the MST value and the fact that this is a 2 approximation to decrease the approximation of our optimal solution.
- Example:
  Consider instance001. Here the value of the MST is 4638. The value of the solution we return is 2517. We know that the MST is a 2 approximation, so the optimal solution must be bigger or equal to $4638/2 = 2319$.

Therefore we can say that our solution is at most $(2517 - 2319)/2319 \cdot 100\% = 8.54\%$ higher than the optimal solution and therefor is a $1.0854 - approxiation$ instead of a 2 approximation.



instance	001	005	013	025	037	041	101	145
Nodes	6405	8013	550	512	2346	320	9287	2865
Edges	10454	14749	5013	2304	4656	1845	14975	4267
Terminals	16	30	50	64	78	80	363	1000
Running time	8041.0	18376.0	1647.0	2119.0	7520.0	2104	152795.0	95776.0
MST value	4638	44421	9907	142	15401486	29317	157,337,998	1,178,099,594
Alg Value	2517	39270	9200	133	12701103	27158	115,024,908	547,483,636

4.2 Running Time

The running time is calculated in the following manner. Our implementation of Dijkstra using the adjacency matrix takes $O(V^2 \cdot log(V))$. In order to calculate the whole metric complete graph, we use Dijkstra for each of the terminals, increasing the running time to $O(T \cdot (V^2 \cdot log(V))$, where V is the number of nodes and T is the number of terminals.

.

After The multiple implementations of Dijkstra we use Prim's algorithm in order to find the shortest path. Prim runs in $O(T^2)$, since we run Prim after Dijkstra and $V \geq T$, we can say that the running time depends only on our implementation of Dijkstra and not on Prims.

5 Further Improvement

- The algorithm is neatly implemented with a simple and efficient code structure. We observe that the programme runs rather quickly, especially for smaller instances. However, the heap memory is not enough to deal with certain large-scale problems. This is due to the huge memory size of the adjacency matrix that we choose to store the data.
  For improvement, it is beneficial to adapt the code structure for big instances. A node class may be implemented with data for adjacent nodes and edges, the information of shortest path may also be included. This way, the huge size $|V|$x$|V|$ of the adjacency matrix may be reduced to one-dimensional array without losing the depth of the data.

- We also notice another aspect of our program that needs improvement. After converting the minimum spanning tree to original edges in graph G, it is possible that some cycles are included. These cycles ought to be removed. To do so, it is possible that we again run the Prim's algorithm on the outputted subgraph to eliminate unwanted edges. Additionally, we may consider the degree of each node afterward and remove non-terminal nodes of degree one.

- We have been careful in the implementation process to obtain good efficiency, however, it is true that our program still has multiple shortcomings. We should work further on implementing the above-mentioned points, as well as further optimising the data structure to improve the running time.