Minimising telephone cable length to connect N hubs in N towns to one main hub

Eimantas Misevicius; Ollie Jaques; David Birley; Alfred Brown

November 18, 2020

Abstract

In our ever increasingly connected world that we live in, it is becoming crucial that one remains in the loop to stay relevant. However telephone cables connecting our towns are expensive, unsightly and made from finite materials. From this comes the problem, what is the most efficient way of connecting N towns to each other, by keeping the length of cables between them to a minimum. This may seem like a modern question, however the root of this problem can be traced back as far as Ulysses Round Trip. This is because finding the shortest path between different places will always be a relevant problem. Here we show three methods for answering this problem, the first being Prim's algorithm, the second being an augmented version of Prim's algorithm and the third using Steiner trees. We found that all three solutions to this problem work efficiently and work similarly well when N is small ($N<10$), however Prim's algorithm would become less effective compared to the Steiner tree as N became larger ($N>100$). This is somewhat to be expected, as Prim's algorithm can only consider two nodes (towns) when finding a shortest path, whereas Steiner Trees can consider up to four nodes, using junctions in the cables to minimise length. This is why the methods work similarly well with small N, as the optimization of Steiner trees does not appear to make much difference till lots of little optimisations have started to add up. Furthermore, we observe that the positioning of the N hubs play a key role and that the various models mentioned above have varying effectiveness depending on the particular geography of the hubs.

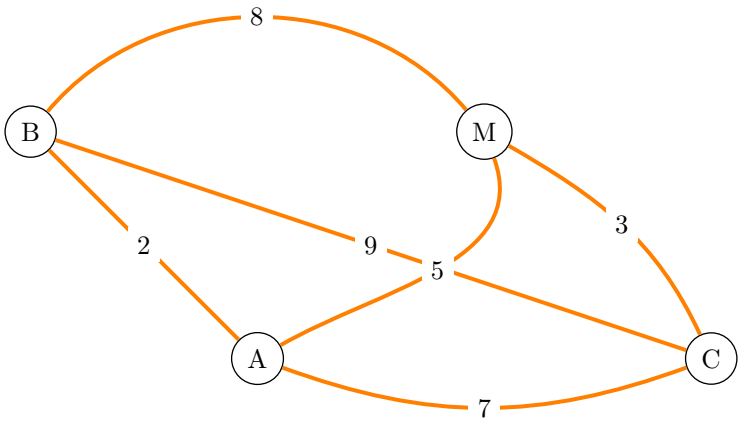
1 Introduction

For centuries mathematicians have tried to optimise the distances needed to get from one place to another. Famous mathematical models have been devised as early as Euler's Seven Bridges of Königsberg paper in 1736 to more recent models such as Prim's Algorithm of the late 1950s. This paper will investigate some of these models and other models found in literature in order to optimise the least amount of cable needed to connect N telephone hubs to one main hub. Furthermore, the performance of each model will be analysed with respect to number and position of the telephone hubs.

While these models may give us optimal solutions on paper, we must not forget that these are just models and cannot necessarily be applied to real life. If one were to generate an optimal model for a real world situation, one would need to take into consideration a whole variety of different factors, like rivers, farm land, nature reserves, town building regulations etc...

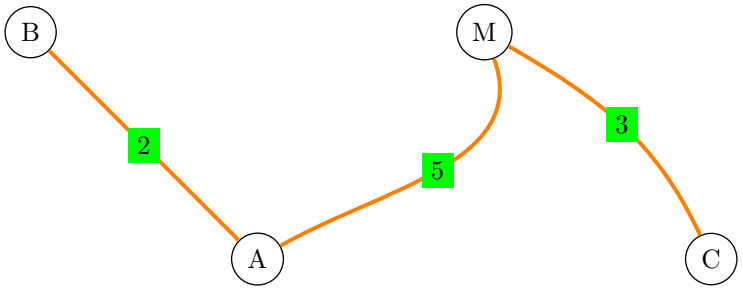
2 Prim's Algorithm

Let us imagine a county with four towns. We chose one of the towns to be the main hub and centre it within a coordinate system. The other towns are now displaced around this centre. Now, as one could do practically, the distances between each town and the main hub is measured and recorded in a table. This is shown in the figure below. Note that we consider the distance between a hub and itself as zero and we would, if say two towns cannot be connected by a link, consider their shared distance as infinite.



	M	A	B	C
M	0	5	8	3
A	5	0	2	7
B	8	2	0	9
C	3	7	9	0

Now we are in the position to follow Prim's famous algorithm. We start from the main hub and find the shortest cable length, which in this case is *Town C*, ignoring all zeros. We now jump to row C and find the shortest length from C, however, since all connecting cable lengths from C are longer than those from M, we return to M and find the next shortest length, which is to *Town A*. Jumping to row A we then follow the same pattern by finding the shortest length in row A, which in this case is to *Town B*. Since all towns are now connected, the algorithm stops. This so-called "minimum spanning tree" (MST) can be seen below by the highlighted line in the figure below.



To generalise this method, we can represent Prim's algorithm using pseudo-code, as shown below [Vis27].

```
T = {M} #T is the MST, consisting now only of the source (the main hub)

Enqueue (add) nodes and link length connected to M into set PQ

while (!PQ.isEmpty)
#while the set of nodes and link length is not empty

  if (node n linked with link length e = PQ.remove ∉ T)
  #Go to node and remove node and link length from set PQ
  #Then check if node in set T

    T = T ∪ {n, e}, enqueue links connected to n
  # If it is not, then add link length to T
  # Also, add all connecting nodes and link lengths to the set PQ

  else ignore e
  #If link length already in T then ignore
```

```
#Repeat until PQ is filled with all nodes then exit
MST = T
#Finally, store T as minimum spanning tree
```

We can now implement this code and make observations on how well Prim's algorithm works for different values of N (the number of hubs) and the positioning of the hubs. There is a fantastic website [Vis27], cited in the bibliography below, that allows one to do precisely this. It simply applies Prim's algorithm to any graph one creates. In the figures below, we show some of the snapshots taken while increasing the number of hubs and a bar graph representing the minimum spanning lengths (weighting), generated by the programme.

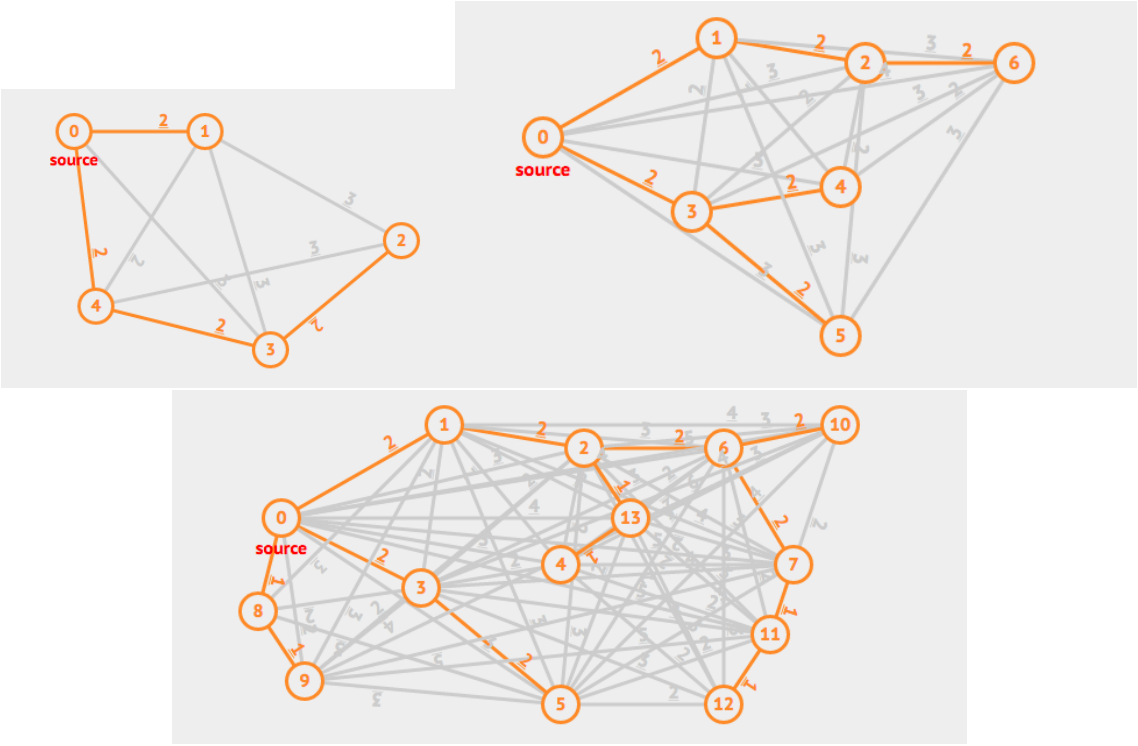


Figure 1: 3 minimum spanning trees using Prim's Algorithm

As the number of nodes increases, the positioning of the nodes become more important. Notice that the first two graphs (top left and top right) are simple extensions of each other. In other words, the nodes that are being added are being added to the surrounding areas of the previously generated spanning tree, thus just increasing the weighting by the length to the nearest node. (This is done for all nodes up to 12). For a spanning tree with more than 12 nodes, the new nodes are placed within the network, as is shown in the bottom graph with node 13. As we can see, the minimum spanning has changed, node 4 is no longer connected to node 3 but instead to node 13. We can see what effect this has by plotting the number of hubs (N) against the weighting of the minimum spanning tree, as shown on the next page.

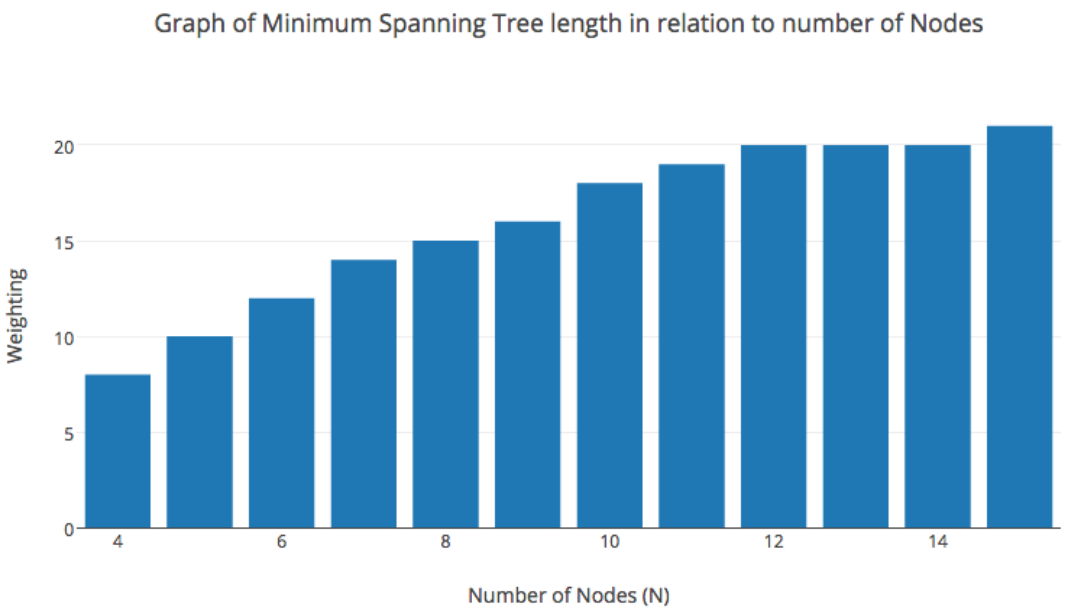
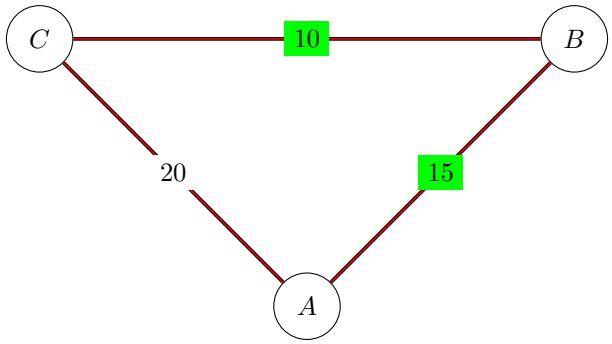


Figure 2: Showing MST weighting for graphs above

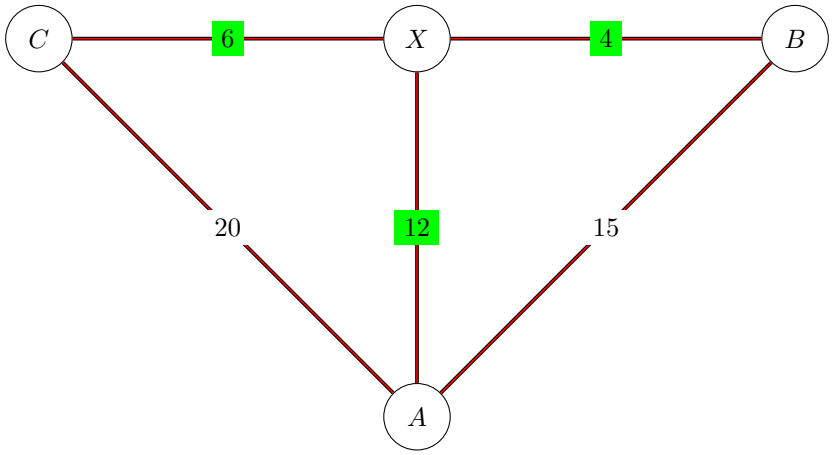
We observe that for nodes that are added externally to the already given network (here starting at 4 nodes) there seems to exist an almost linear relationship between the weighting and the number of nodes. However, when we start introducing nodes within the network, we observe that the weighting does not necessarily increase, but rather seems to remain constant. This is the case for nodes 12 to 14. Node 15 increases the weighting by 1 link length, even though it was placed within the network. What we can infer from this investigation, is that by placing nodes within a network it allows for Prim's Algorithm to find different and perhaps shorter spanning trees than it would if nodes are being added externally, thus leading us to believe that there is a more efficient way to decrease the total weighting of a network. Which in turn means there is perhaps a better method for more efficient usage of telephone cable. This is investigated in the next model below.

3 Prim's with added nodes

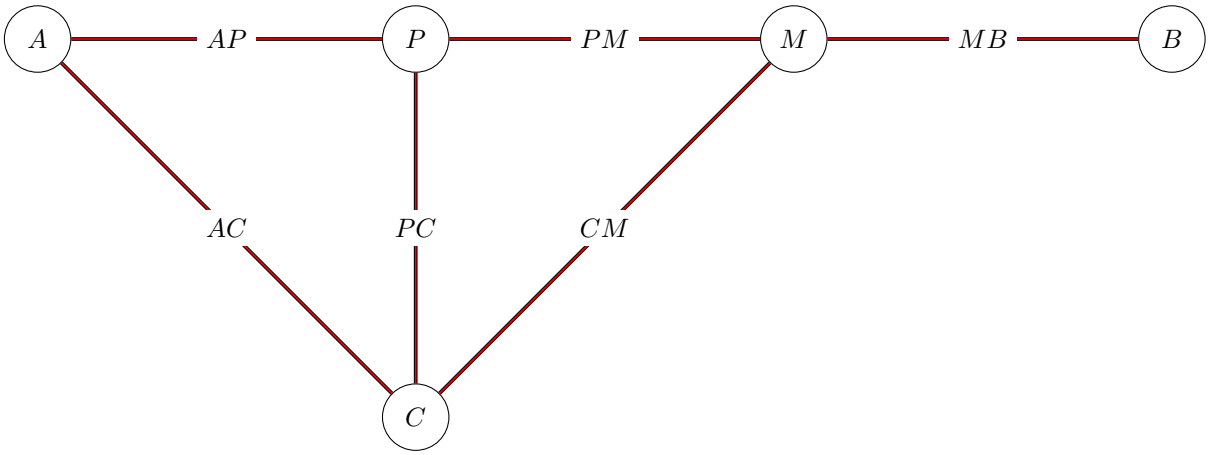
A big issue with Prim's algorithm and many others used to solve the minimum spanning tree problem is that all the arcs must start and finish at the nodes defined at the beginning. Very often the total arc lengths can be reduced if some arcs simply meet other arcs already made between nodes. One way in which you can improve upon Prim's is if you take any 3 nodes and then connect the 2 nodes closest together, if you can draw an arc from the 3rd node to meet the first arc such that they are perpendicular, this will always be much shorter than if the arcs just went between the nodes (unless where the the lines are perpenticular is at one of the nodes). This is in effect acting as though each arc is made up of many fake nodes throughout, with each one able to receive arcs from other nodes.



The highlighted values show the shortest route going from $A - B - C$. But the length can be reduced with an extra fake node (X) on the arc CB, creating the new tree $A - X, C - X - B$.



This is very difficult to implement as each new fake node added creates many possible new arcs to each real node, especially for large numbers of nodes. Meaning that increasingly more distances are needed to be calculated and therefore it takes a lot longer to find a solution. Perhaps a good compromise is to just create one fake node in the middle of each arc. Obviously only using one extra node is less effective and it may at first seem difficult to see in which cases it would be a shorter route to the fake node as opposed to the 2 nodes either side of it. However, we can use simple geometry to find this. If we connect any 2 nodes, A and B, and find their midpoint M. For any 3rd node C, such that an arc can be drawn from it perpendicular to the arc AB and meet it at the point P, using Pythagoras' theorem of $a^2 + b^2 = c^2$, you can deduce that when $PM < AP$, $CM < AC$. This means that any point within the middle 50% of the arc will have a shorter distance to M than A or B.



For example, if you take the simple case of an $A = (0, 0)$ and a $B = (0, 40)$, creating an $M = (0, 20)$. Any points (C) with an x within the bounds $10 < x < 30$ will have a shorter distance to M than either A or B.

This can be used to implement an augmented Prim's, following the usual process but with each new cable laid connecting two towns, a new fake town can be thought to be created in the middle of the cable. These new towns don't need any extra cable to join them to the rest as they already reside on the cables between other real towns, due to being defined as such. Instead they just create extra opportunities to cut down the total length. It will only be a small amount if at all for small numbers of towns, but for much larger numbers of towns the benefits become apparent. To further extend the idea of an augmented Prim's algorithm we now turn to Rectilinear Steiner Trees, which do not confine placing additional fake nodes in between nodes, but rather at a point within a plane of nodes.

4 Rectilinear Steiner Trees

The problem of connecting a main hub to N other hubs can be described in essence as a Steiner problem, which is an umbrella term for a class of problems in combinatorial optimization. Using Euclidean Steiner trees this problem can be seen as a generalization of two other combinatorial optimization problems: the shortest path problem and the minimum spanning tree problem. So if the problem consists of exactly two nodes then the Steiner tree reduces into finding the shortest path. However if there are three or more nodes then the Steiner tree is equivalent to the minimum spanning tree.At first glance, Steiner Trees may not appear to have too much of an advantage over Prim's Algorithm. Indeed for $N = 2$, the solutions are exactly same. However the Steiner trees start to optimise the shortest path better when N is equal to three or above. They do this by trying to make an imaginary point in between three points. This Steiner point must have a degree of 3, as well as three edges incident to such a point must form three angles of 120 degree or less. This is also known as a Fermat point. If these conditions can be met, then an optimization of the shortest path can be made by creating an intersection of cables at the Fermat point. More than this Steiner Trees can further optimize when $N = 4$, as it can join between two Fermat points. This is shown below.

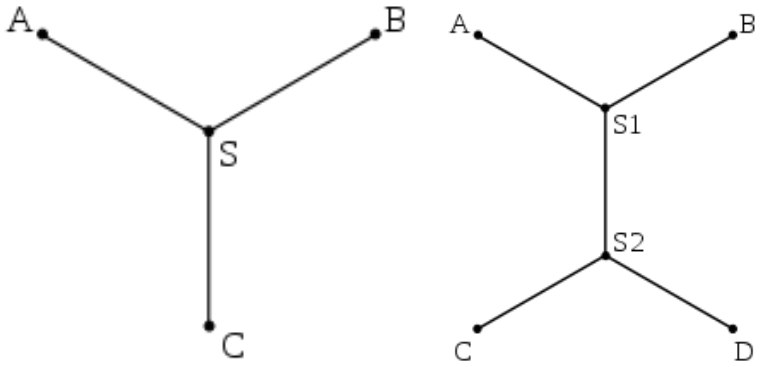


Figure 3: Steiner Tree optimizations for $N = 3$ (left), and $N = 4$ (right)[wik01a]

Euclidean Steiner Trees however are classified as NP-hard, and so it is not clear whether there is an optimal solution. Because of this we will be instead be using an approximation, Rectilinear Steiner Trees. This variant of the geometric Euclidean Steiner Tree, is less efficient as it replaces the Euclidean distances with Rectilinear distances, and so locks the wire connecting towns to only horizontal and vertical movement. However it is easier to program and this technique is useful when the problem already constrains to vertical/horizontal movement e.g. wiring in circuit. This can be done by drawing horizontal and vertical lines through each vertex to construct a Hanan grid, as shown below.

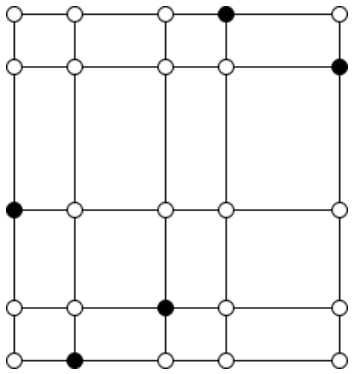


Figure 4: Hanan Grid for 5 vertexes[wik01b]

The following pseudo-code is to create the minimum spanning tree:

```
Algorithm Rectilinear Spanning Graph (RSG)
```

```
for(i = 0; i < 2; i + +) {
  if (i == 0) sort points according to x + y;
  else sort points according to x − y;
  A[1] = A[2] = ∅;
    for each point p in the order {
      find points in A[1], A[2] such that p is in their R2 i+1 and R2 i+2 regions, respectively;
      connect p with points in each subset;
      delete the subsets from A[1], A[2], respectively;
      add p to A[1], A[2];
    }
  }
```

[Zho04]

Meanwhile this is the pseudo-code for making the optimization to a Stiener tree, by checking the nodes, and creating Steiner points when it can reduce the overall length:

```
Algorithm Rectilinear Steiner Tree (RST)
  T = ∅;
  Generate the spanning graph G by RSG algorithm;
  for (each edge(u,v) ∈ G in non-decreasing length) {
    s1 = find set(u); s2 = find set(v);
    if (s1 != s2){
      add(u,v) in tree T;
      for(each neighbor w of u,v in G)
        if(s1 == find set(w))
          lca_add_query(w,u,(u,v));
        else lca_add_query(w,v,(u,v));
      lca_tree_edge((u,v), s1.edge);
      lca_tree_edge((u,v), s2.edge);
      s = union set(s1, s2); s.edge =(u,v);
    }
  }
  generate point-edge pairs by lca_answer_queries;
  for (each pair(p, (a,b), (c,d)) in non-increasing positive gains)
    if((a,b), (c,d) has not been deleted from T) {
      connect p to (a,b) by adding three edges to T;
      delete(a,b), (c,d) from T;
    }
```

[Zho04]

Using this code we can now create spanning trees for nodes, and then from that we can find the optimised Steiner tree. As shown below for 500 hubs:

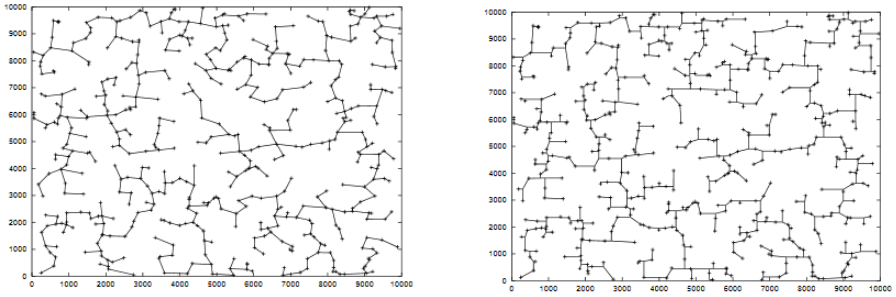


Figure 5: Minimum Spanning Tree for $N = 500$ (left), and Rectilinear Steiner Tree for $N = 500$ (right)[Zho04]

5 Testing and Results

It is surprisingly difficult to test which of the three models produce the least usage of cable, because one cannot simply fix the positioning of the nodes for different values of N and then apply the previously mentioned pseudo-code for each model. Instead one has to compare an average weighting generated by each model separately to give a rough guide as to which model is most effective. The values of N chosen to test were 2, 5, 10, 50, 100 and 500, and they are shown in the table below. Since Prim's Algorithm with added nodes was devised as an alternative extended algorithm, the tests were only generated for Prim's algorithm and Steiner Trees.

N	Average Length for Prim's Algorithm	Average Length for Steiner Trees
2	1.42	1.42
5	6.23	6.02
10	9.78	9.10
50	51.35	49.56
100	103.45	98.47
500	495.32	469.95

As we can see, Prim's Algorithm and Steiner Trees generate roughly the same weighting for N up to 50. At 100, we start to see a slight difference in weighting, showing Steiner Trees as more efficient. For high N, such as 500 we find that the difference in efficiency of Steiner Trees becomes even more apparent. We might thus conclude from this experiment that Steiner Trees are a better solution for larger N, but again, it is very important to mention that the positioning of the nodes in this experiment have not been taken into consideration, and thus depending on the actual situation of the real-life case, Steiner Trees may not necessarily present the best solution.

6 Conclusion

The advantages of using Prim's Algorithm is that it works well for any real world situation, because all it requires are the lengths between the nodes (towns) that can be connected by cable and nothing more. Complications, such as rivers can easily be taken into consideration, by simply ignoring cables that in theory would have to cross the river and simply using lengths of alternative routes (even if they are not the most efficient).

The disadvantages of using Prim's algorithm as a model for this kind of problem is that it is clearly not necessarily the optimal usage of cable. This is where Prim's with added nodes trumps Prims, as the cables can split and don't have to start and finish at pre-set nodes, meaning the lengths can be reduced greatly. For small amounts of towns/nodes this model will often have the exact same length but it will never be longer, so in terms of length it will always at least match Prim's. However, the process to calculate the extra lengths as each new fake node is added means it takes much more time to generate than Prim's despite sometimes giving the same result, which could be considered a waste. Also Prim's with added nodes only works really well in specific arrangements of nodes, when they create cross-like patterns, and it really doesn't help if the nodes are in a line.

In addition realistically, using this method would mean lots of different cables would have to be laid, many of which would be very small. from a practical standpoint using less cables with a slightly larger length could be thought to be easier to actually lay. Steiner Trees which clearly seem to be the optimal model for situations containing a large number of towns, is in practice also very hard to do. As we have seen, we can only generate a program for rectilinear Steiner Trees, meaning the cables would have to run on a grid format, which could be very tedious to do in the real-world. Nevertheless, if one can design a method for applying Euclidean Steiner Trees (which is definitely possible, but beyond the scope of this investigation) then this would definitely provide the most optimal solution, as it uses the the least amount of cable.

References

[Vis27] Visualgo. https://visualgo.net/mst, (Accessed: 2017-02-27).

[wik01a] Wikipedia. https://en.wikipedia.org/wiki/Steiner_tree_problem, (Accessed: 2017-03-01).

[wik01b] Wikipedia. https://en.wikipedia.org/wiki/Rectilinear_Steiner_tree, (Accessed: 2017-03-01).

[Zho04] Hai Zhou. Efficient steiner tree construction based on spanning graphs. *CiteSeerX*, pages 1–6, 2004.