Cost optimization system for fiber-optic backhaul routes based on geographic data

Author: José María Rodríguez Cano de Santayana

Director: Ignacio Ríos Calvo

Rivergo Advisors and Universidad Pontificia de Comillas - ICAI

February 2021

Abstract

The purpose of this paper is to document the process of developing and implementing a tool capable of optimizing the cost of connections in optical fiber deployment projects. This problem can be reduced to the Steiner Tree problem in graphs in its optimization version. To generate a graph that represents all available deployment means, geospatial data will be ingested and processed from different sources. The system will have an interactive interface implemented as a web dashboard, allowing both running the optimization algorithm and analyzing results dynamically.

1. Introduction

When trying to connect with optical fiber a series of locations to an existing backbone, the problem arises of how to trace these connections so that the cost is minimal. To decide where the fiber is deployed, different alternatives will be considered. Some examples would be pipelines, gas pipelines or power lines, as well as the complete roadmap in Spain, including all

available transport routes: highways, streets, roads, and paths. In these deployment projects establishing an investment limit per branch is a common practice. Therefore, it will be necessary to determine which points are viable, thus optimizing the estimated benefit. This computation will be done considering the costs and revenue that connecting each point is expected to report.

This problem, as with many other similar problems in other types of infrastructures, can be reduced to the minimum Steiner Tree in graphs. This is a famous NP-hard problem so most practical applications implement an approximation algorithm capable of giving a solution in polynomial time within a given approximation ratio of the optimum solution.

2. The minimum Steiner tree problem in graphs

The Steiner tree problem is one of the most famous NP-hard problems, especially regarding graph theory because of its practical use in many real-world applications. Given a graph $G = (V, E)$ comprised of a set of vertices V and a set of edges E of order $n = |V|$, edge costs $c \in \mathbb{R}_+^E$ and a subset of terminal vertices $S \subseteq V$ of order $k = |S|$, the objective is to find a minimum cost subtree $T = (V', E')$ of G spanning all terminal nodes.

Since the Steiner minimal tree problem is known to be an NP-hard problem, assuming $P \neq NP$, only approximate solutions can be found in polynomial time. The best solution found to date is the one proposed by Robins and Zelykovsky [1], showing an approximation ratio of 1.55. That means that the solutions found are, in the worst case, 55% worse than the real optimum. However, many of the solutions that achieve low approximation ratios also involve high temporal complexity. Therefore, each algorithm offers a different point of balance between the approximation ratio and the execution time. Practical solutions may involve from non-polynomial algorithms that find an optimal solution such as [2], to the simplest solution, reduction of the problem to the MST problem (Minimum Spanning Tree).

It is important to highlight that in most practical applications it is not feasible to use an optimal algorithm [3], since they are not very scalable due to their non-polynomial temporal complexity.

The approximation algorithms used in practice could be classified in three main categories.

- **Heuristics** (shortest path). These are the easiest to implement and the ones that were first discovered. They are based on shortest path computations between two vertices of a graph following some heuristic criteria to improve the solution found over the simplest possible solution (MST). Some examples of this type of algorithm are developed in more detail in [4] [5].

- **Contraction**. There is a great variety of algorithms of this type since for many applications they are the balance point between complexity, execution times, approximation ratio and scalability. These algorithms attempt to approximate a solution by finding local optimum solutions for sets of $k \leq |S|$ terminal nodes. Some examples of this type of algorithm are [6] [1].

- **LP** (Linear programming). These are the most complex and the ones that generally achieve the best approximation ratios, however, they do not find much practical use since they are not very scalable, and their execution times are greater than those of the other two categories. They find themselves at a midpoint between the rest of the approximation algorithms and the exact algorithms in which it is less frequent to find practical applications. Some examples of this type of algorithm are [7] [8].

3. Construction of the graph G to represent means of deployment

This graph will be constructed with geospatial data from many different sources, representing pipelines, gas pipelines, power lines, as well as the complete roadmap in Spain, that is, all available transport routes: highways, streets, roads, and paths.

Given that the geometries of the transport and distribution infrastructures contemplated in this project are represented as sets of lines in two dimensions on a projection, it is critical that, in the data cleaning stage the coordinates of these geometries match exactly to correctly represent intersections. To transform geospatial data into a graph, it is sufficient to identify these coincidence points, assign them as vertices in a graph and join them through an edge if they are part of the same line. These lines that connect different nodes of the total network will be called sections of the total network.

To recover the geographical information lost in this process, a unique identifier will be assigned to each node and line, so that once the optimal connections have been calculated, it is possible to recover exactly the corresponding geospatial objects. As the geometric features will be lost in this step, it is necessary to carry out the corresponding measurements before finishing the transformation. A series of attributes will be assigned to the edges of the graph to be used later in the optimization algorithm.

- **Origin / type**. For each section, an attribute will be assigned to the corresponding axis of the graph that specifies what type of infrastructure it represents. That is: transport network (roads), gas pipeline, overhead lines, etc.

- **Subtype**. For each section, the specific characteristics that may affect the cost of deployment. In the case of public roads, the exact type will be specified (urban road, highway, road, path, etc.); if it is a pipeline, the type: (PVC, corrugated, etc.). This will be useful later for several reasons. First, it is necessary to calculate the cost of deployment in this section. Additionally, this information will be used to analyze the calculated connections and to apply filters if deemed necessary, for example: filter the highways and highways so that the algorithm does not consider them since they usually involve a long delay in the project due to the time needed to obtain the necessary licenses.

- **Owner**. For each section, the owner of the infrastructure will be specified. It can be public (in the case of roads for example), owned by the client company or owner by another company. This may affect the deployment price, it can be used as a filter, and it will allow the analysis of the solution found.

- **Distance**. The distance of each section will be measured in meters. Although the algorithm will not consider the distance but the cost when optimizing, this measure is useful to later analyze the characteristics of the solution.

- **Cost**. This is the most important parameter. It will be added as an attribute to each edge of the graph. The cost of deploying fiber in each section will be calculated considering its distance and the cost (per meter) based on the characteristics of the section: type, subtype and owner.

A series of features will also be specified for the vertices. Each vertex may fall in one of the following categories:

- **Points to connect**: these vertices will represent the offices, towers, or any other location that is to be connected with fiber. Although the geographic data does not initially match exactly with any object in the total network, it will be pre-processed to intersect the road.

- **Splice**: these vertices represent the existing splices in the fiber optic network of the client company. The final purpose of the algorithm will be to calculate the routes between the points to be connected and any splice (the one considered optimal in each case).

- **Normal nodes**: all nodes that are not in any of the previous categories will be considered a normal node. As already mentioned, these nodes represent intersections and points of coincidence between different sections of the total network.

A single virtual vertex will be created that abstractly represents the existing network of the client company. This vertex and the edges that connect it will not have a geographic equivalent or cost. This virtual node (this "-1" as a identifier) will be connected to all existing splices in the graph with edge cost 0.

4. Approximation algorithm

Analyzing the operation of any algorithm to calculate the MST, either Prim's or Kruskal's, a lemma can be assumed that will simplify and significantly reduce the number of necessary calculations. Note that this lemma will also be valid for any Steiner tree algorithm. It is important to note that in the case of the MST approximation algorithm the number of edges of the resulting Steiner tree will be equal to the number of vertices minus one $|E'| = |V'| - 1$. This is also the minimum number of edges a Steiner tree can have.

Lemma: No edge in the resulting Steiner tree will cost more than any of the shortest paths between the nodes it connects and the virtual node -1.

Therefore, just by calculating the shortest path from each terminal node to the virtual node -1, we will have already computed all the possible paths that can be contained in the Steiner tree.

The algorithm finally implemented in this project is the one proposed by H. Takahashi and A. Matsuyama in [4]. Since no specific name is assigned to it, from now on, it will be referred to as TM80.

This algorithm offers an approximation factor of $2 \cdot (1 - \frac{1}{k})$, where k is the number of terminal nodes. The approximation factor will converge to 2, similarly to the basic MST approximation algorithm. However, in practice, better solutions can often be found. In fact, the solutions are always strictly better than MST.

This algorithm follows an additive process adding routes definitively to what will eventually be a complete Steiner Tree. One of the great advantages of this algorithm is its great speed, presenting a temporal complexity of $O(kn^2)$, with k being the number of terminal vertices and n the total number of vertices in the graph. However, taking advantage of the particularities of the problem addressed in this project, the performance of the algorithm can be considerably improved.

Unlike MST, TM80 allows us to obtain Steiner trees with Steiner nodes of order higher than 2. This means that it can considerably reduce the distance and cost of deployed fiber by creating intermediate nodes in which several routes converge. The operation of the algorithm is detailed below as pseudo-code.

```
function T&M(Graph G, Terminal nodes S)
create empty set of spanned terminal nodes st
create empty graph T
add one arbitrary terminal node to st
while st ≠ S
      create empty array best_edges
      min_cost ← -∞
      for each node v in (S − st)
            cost, edges ← shortest path in G between
            v and any node in T
            if cost < min_cost
                  min_cost ← cost
                  best_edges ← edges
      T = T ∪ best_edges
return T
```

One of the additional functionalities offered by the tool developed in this project is the ability to calculate which terminal nodes and branches are viable according to a given CAPEX limit. The deployments can be optimized so that, the more points are evaluated at the same time, the greater return it will be possible to obtain on average for each connected point.

Viability of a point: Any branch is defined as viable if the total cost of the branch (subtree from the closest splice) is less than the CAPEX limit times the total number of viable points connected. If a branch is determined not viable, the leaf vertex with maximum connection cost will be eliminated. The minimum Steiner Tree will be recalculated without the removed vertex. This is because, by suppressing the need to connect a node that has already been determined not

viable, the connections for the rest of the nodes calculated by the algorithm can be different. If they are, it will be at a strictly lower cost than the previous calculation.

Figure 1 Pruning process in a branch. Vertices with an "X" mean the branch is not viable yet. The lightness of edges denotes their cost, the lightest edge will be removed if the branch is not viable yet.

5. Visualization interface

The final part of this project is the development and implementation of an interactive visualization dashboard from where you can analyze the points you want to evaluate in real time. It allows to modify the different input parameters dynamically to evaluate a project. An MVP has been implemented that allows reading a .shp file located in the same directory as the Python script and shows through a web interface what is shown in Figure 2.

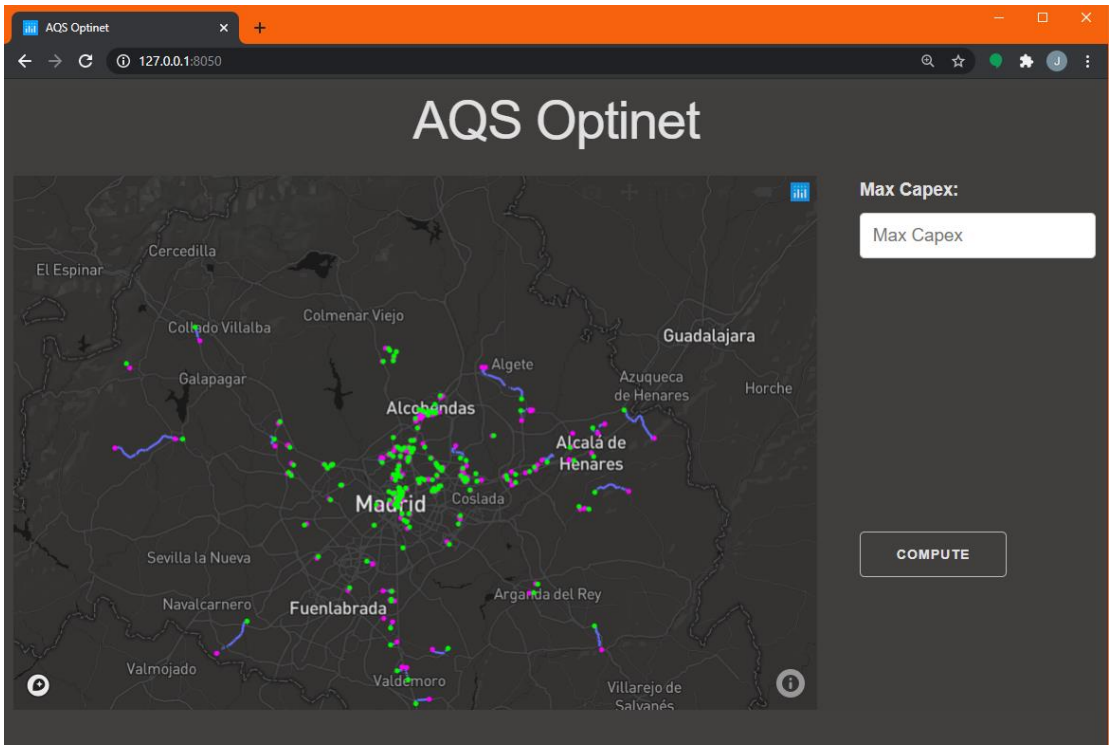

Figure 2 Web dashboard interactive visualization tool

6. Conclusions

A viable platform has been developed and implemented capable of evaluating fiber optic deployment projects. The viability under the conditions detailed in this document has been demonstrated, as well as the great scalability that the tool offers for future extensions that may be desired. Undoubtedly, the use of this tool will be a great advantage when evaluating projects, since it allows considering large amounts of points to connect. As detailed in this document, the more points are evaluated simultaneously, the better solution the algorithm can find. The developed tool offers a simple and intuitive interface that allows these evaluations to be carried out quickly and in a more dynamic way.

7. References

[1] G. Robins and A. Zelikovsky. *Improved Steiner tree approximation in graphs*. SODA 00, 2000. 770-779

[2] S. E. Dreyfus and R. A. Wagner. *The Steiner problem in graphs*, Networks 1, 1972. 195-207.

[3] M. Chimani and M. Woste. *Contraction-Based Steiner Tree Approximations in Practice.* Algorithms and Computation, 2011. 40-49

[4] H. Takahashi and A. Matsuyama. *An approximate solution for the Steiner problem in graphs*. Math. Japonica 24, 1980. 573-577.

[5] Winter and J. MacGregor Smith. *Path-distance heuristics for the Steiner problem in undirected networks*. Algorithmica 7, 1992. 309-327.

[6] A. Zelikovsky. An 11/6-aproximation algorithm for the Steiner tree problem in graphs. Information Processing Letters 46, 1993. 79-83.

[7] L. G. Khachiyan. *A polynomial algorithm for linear programming.* Soviet Math. Doklady 20, 1979. 191-194.

[8] J. Byrka, F. Grandoni, T. Rothvoß, and L. Sanità. *An Improved LP-based Approximation for Steiner Tree*. Proc. 42nd STOC, 2010.

[9] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner Tree Problem,* North-Holland, 1992.

[10] F. K. Hwang and D. S. Richards. *Steiner tree problems*. Networks 22, 1992. 55-89.

[11] A. B. Kahng and G. Robins, *On Optimal Interconnections for VSLI,* Kluwer publishers 1995.

[12] P. Winter and J. MacGregor Smith. *Path-distance heuristics for the Steiner problem in undirected networks*. Algorithmica 7, 1992. 309-327.