Efficient computation of fiber optic networks

van den Boogaart, A.A.M.

Award date:
2019

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computer Science
Algorithms Group

Efficient computation of fiber optic networks

Master Thesis

Arjan van den Boogaart

Supervisors:
dr. Bart M.P. Jansen
ir. Flip van der Valk

Nieuwkuijk, November 2018

Abstract

The network planning involved for fiber networks gives a combination of many different known algorithmic problems. We aim to automate the planning of fiber networks by determining which customers exist in an area, placing distribution points nearby and connecting each customer to a distribution point, all while minimizing the expected construction and material costs of the plan. In this work we define these minimization problems and use both existing solutions and new approaches to convert the geographic data to a graph, compute a near-optimal Steiner tree on the graph and place capacitated medians within the Steiner tree.

We develop multiple models that convert geographic data into a graph with suitable locations for cable ditches, weighted by the expected cost of digging. We apply the SCIP-Jack Steiner tree solver on our graph to find a near-optimal Steiner tree between the customers, aiming to minimize digging costs for a fiber network. Though the connection of a customer to the road is often not placed correctly, the rest of the network is similar to manually drawn networks. Preliminary results show that the costs of our generated networks are significantly lower than manually drawn networks for the same area.

We present a new optimal facility location algorithm on trees to place distribution points with maximum capacity k to serve customers within the network, which minimizes the sum of cable costs to the facility and the facility costs in running time $O(k^2 \cdot n)$ when we do not allow multiple facilities in a connected component for a tree with n nodes. When we allow networks to overlap, the running time increases to $O(n \cdot D \log D + n \cdot D \cdot k)$, where D is the sum of integer demands given by customers.

Executing the algorithms sequentially gives cost-efficient network plans for customers to distribution points within 10 minutes for (sub-)urban areas with up to thousands of customers. Our approach can replace a majority of the manual labor required in fiber network planning.

Contents

Chapter 1

Introduction

The use of the internet has been greatly increasing in recent years, which has created a need for faster internet connections for consumers. The fastest internet connection currently available for consumers uses fiber optics. In this thesis, we will consider the placement of underground infrastructure required for such networks.
Fiber optic networks have multiple different layers, starting from the customers up to the internet backbone connecting different cities.
A customer receives a cable from a nearby *distribution point* (DP) with 2 fiber strands going into the customer access point (CA) inside their fuse box. Only one fiber strand is currently used, the other is reserved for possible future needs. A DP is a large weather resistant box that is placed underground. It can typically connect up to 48 customers using 96 fibers. Each customer receives a distinct cable from the DP, which is placed underground and is typically located beside the road. Once it reaches the customer's property, it will pass through it to the front door, where it then connects to the fuse box.
Inside of a DP, each fiber strand of the outgoing cables is welded to the incoming 96 fiber cable, forming a passive connection. The 96 fiber cable connects the DP to an *Area Point of Presence* (AP).
The AP is a small building that receives the cables for up to 960 customers, or around 20 full DPs. It extracts each of the individual fibers again, but forms an active connection that takes care of routing incoming packets to the correct customer, and routes packets from customers closer to the required destination.
The *City Point of Presence* (CP) is on the highest level of the hierarchy. It receives the outgoing cables from APs and forms the so-called *backbone* of the internet. The combined network allows the customers to send data to anyone on the internet.
ThePeopleGroup is a company active in underground infrastructure planning and provide software

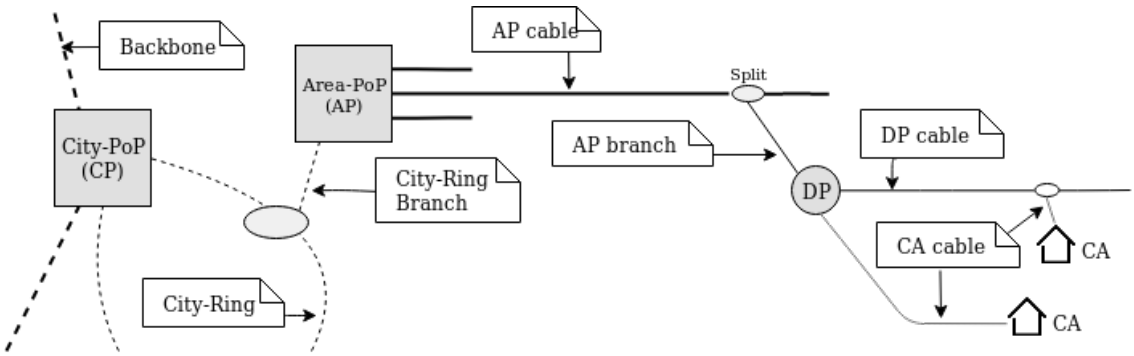


Figure 1.1: High-level fiber network architecture

to aid in the design of these networks.
They define multiple different types of network planning projects. A High-Level design will take into account how to place CPs and APs to cover the demands from cities. Low-level designs typically already have a set location for an AP, and only require DP and cable placements for a set of customers. We can also distinguish between existing and new neighborhoods, called brownfielding and greenfielding respectively.
The focus of this thesis is on the planning of low-level designs for existing neighborhoods.
The largest cost involved in these projects is the digging cost. Digging a ditch costs around €10/m, though there are typically extra costs when breaking up paving for the extra work required and municipal fees.
Breaking up an asphalt road and restoring it is the most expensive at €110/m. Luckily, digging up the road is not the only option: It is also possible to shoot a metal tube through the ground to place cables in. This is called drilling and costs between €35/m and €60/m. Drilling is not just used when digging costs are too high. Its also used to place cables under trees or when the cable needs to pass through someone's garden.
Each type of area and coverage then has its own cost associated to it. Digging through private property is usually only done to place that customer's connection, because it is difficult to get the required permissions and because it costs more than municipal property. Railroads and large waterways are generally avoided for similar reasons.

The other large costs in a project are distribution points (∼€1000 per DP) and cables (∼€1/m). Some other material costs are cable ducts and protective covers around welds and cable branches.

Currently, the network planning is a manual process performed by network engineers. They use CAD tools to place customer access points, divide the customers into clusters for Distribution points, place the DPs somewhere on the network and then draw the lines, called traces, from each access point to its distribution point.
The current process has 2 main issues. The first issue is the high design cost. A lot of customer connections are very simple, but still require design time from the engineer.
The second issue with a manual approach is the inability to precisely measure the expected costs and efficiently evaluate alternatives. This can lead to network plans that look good, but that have higher construction costs than alternative plans for the same area.

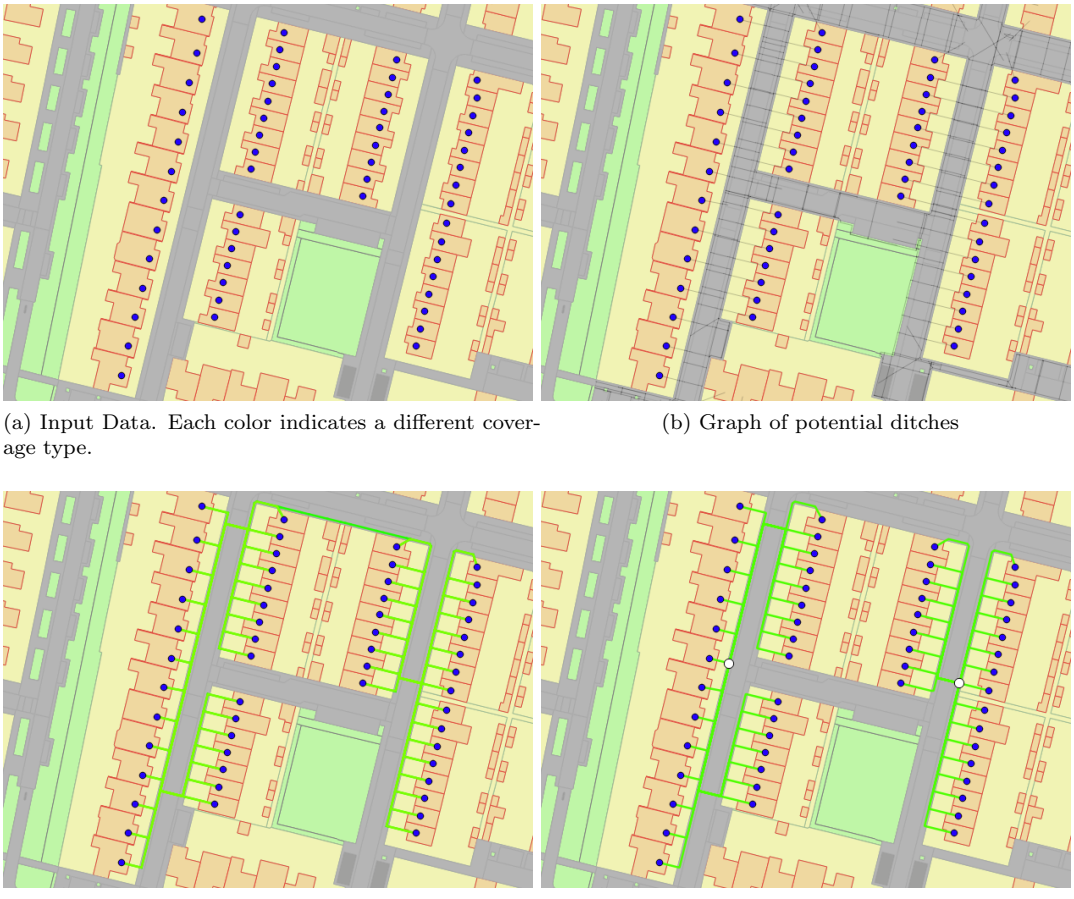
1.1 Problem statement and approach

Our aim is to automate the process of placing customer access points, placing DPs, and creating each of the connections from customers to a DP for existing neighborhoods. The goal is to reduce the required design time and reduce the expected construction costs of network plans compared to a manual approach.
Our approach consists of 2 phases. During the first phase, we create a weighted graph containing edges representing potential locations for ditches and drillings based on geographic datasets of the project area.
For any spanning tree on that graph, the cost of an edge of this tree is similar to the cost of digging a ditch or drilling at the location of that edge for the given length.
Similarly, for any network in the real world, there exists a spanning tree in the graph with similar structure and similar costs.

In the second phase we connect each customer to a distribution point while minimizing the total cost of this network.
The costs involved can be split into 2 known graph problems: The digging cost describes the Steiner Tree problem, in which we aim to find the minimal total edge cost to create a tree containing all customers. The cable and DP costs describe a capacitated facility location problem, in which we want to minimize costs by finding central locations for the distribution points. The cable cost for a customer is given by the distance to its distribution point.
While both problems are NP-hard on graphs, they both have practical solutions that provide quick near-optimal results. The combined problem however does not have any significant solutions.

Since our digging costs are at least an order of magnitude higher than the cable costs, we decided to find a Steiner Tree solution first, after which we place facilities optimally on the tree.

Results We develop a heuristics based approach to generate potential ditch networks that contain similar solutions to those found in manually constructed networks.
We then evaluate the performance of the SCIP-Jack[1] Steiner Tree solver on our datasets, which shows great performance, even for datasets containing thousands of customers.
We present a new optimal algorithm for the Capacitated Facility Location problem on trees and give various modifications to have solutions conform to the fiber network requirements and improve running times.
The result is a set of algorithms that when computed one after the other gives a valid fiber network with low costs. The algorithm behaves well in urban and sub-urban neighborhoods, such that the majority of the network can be directly used in a network plan, while other sections only require minor modifications, such as moving customer connections around, to obtain good solutions. We have not considered rural areas in our evaluation, as these have a very different structure from urban areas and will require additional modifications for good solutions. Figure 1.2 shows each step of our approach starting from an input dataset.



(a) Input Data. Each color indicates a different coverage type.

(b) Graph of potential ditches

(c) Steiner Tree Solution

(d) Facility Location Solution

Figure 1.2: Example output of each phase. The selected customers are indicated using blue dots, the cable placement is indicated in green and the DPs are shown as white dots. Each background color indicates a different type of area, such as buildings, roads, vegetation or private property.

1.2 Organization

The remaining chapters are organized as follows: Chapter 2 gives a thorough overview of related work, showing other fiber network approaches, an overview of the Steiner Tree problem and of the Facility Location problem.
Chapter 3 then describes the constraints and costs of fiber networks more formally, followed by a description of the available datasets and a formal problem description. We then describe the operations on polygons that we use during our geographic phase.
Chapter 4 covers the geographic phase, in which we create a graph based on geographic datasets. We present our model to create a representative graph and several modifications to improve running times and costs. We show the effect of each modification and how the complete model compares to manually constructed networks.
In chapter 5 we evaluate the performance of the SCIP-Jack [1] Steiner tree solver on our datasets.
In chapter 6 we present a new algorithm for capacitated facility location in trees and prove its optimality. We then show some modifications to the algorithm required for the fiber network. We show the computational results of this algorithm compared to similar facility location algorithms.
Chapter 7 combines each of the algorithms into an algorithm for the fiber network planning problem. We discuss the advantages and disadvantages of our approach and evaluate the performance as a whole.
Finally, in chapter 8 we summarize and discuss our findings, give a conclusion and give several options for extensions and future work.

Chapter 2

Related Work

In this chapter we discuss related work on the various algorithmic problems treated in this thesis. We first consider other approaches to the fiber network planning problem (Section 2.1), followed by the Steiner tree problem (Section 2.2) and the facility location problem (Section 2.3).

2.1 Fiber Network Planning

There are already software products that create a fiber network using similar constraints. We evaluate some of these approaches and compare them to our method.

Ksavi FTTH Planner The Fiber To The Home (FTTH) Planner by Ksavi[2] takes a divide and conquer approach to the problem. As input it takes the set of customers and set of roads in the area. They first connect customers to the nearest road, after which they use hierarchical clustering to create multiple subproblems. Distribution points are placed at the center of each cluster, which are then connected using the shortest path heuristic algorithm for the Steiner Tree problem [3].
The first issue with this approach is that it only takes the road network into account. A solution to our problem requires a much more detailed network, determining which side of the road is used, and where roads are crossed. The hierarchical clustering also introduces many errors. It creates clusters based on straight-line distance instead of distance in the network, causing customers that aren't nearby to be assigned to the same distribution point.
The solutions given by this process are not detailed enough and can produce solutions with excessively high costs in practice.

ESRI FTTx The ESRI FTTx [4] approach provides more detail by creating a potential ditch on each side of each road, and adding crossings at intersections. They consider the cost of ditches, cables and distribution points.
The ditches are always placed at the same distance from the center of the road, which often causes them to be placed at invalid locations in practice. The coverage type of a road is not considered in their cost function, which means it cannot differentiate between asphalt, pavers and grass. While the solutions are already a lot more detailed than those of the Ksavi planner, the ditch network is not accurate enough and the cost of edges is not always representative of the actual cost.

FiberPlanIT The approach of FiberPlanIT[5] is the most advanced. They select the customers and use polylines as roads. Each road receives a ditch on either side based on the width of the road and crossings are generated where roads intersect and wherever customers are connected to the network.
Based on this network, they use a capacitated clustering algorithm that takes into account the ditch costs and lengths. Afterwards, they generate a Steiner tree with the centers of each cluster

as terminals.
There are still some issues with this approach that cause more expensive solutions than required. The locations of ditches are improved compared to the ESRI approach, but due to the constant width per road, the placement can still be invalid.
While the ditch network differentiates between ditches crossing roads and those beside roads, it does not consider the coverage type when calculating costs. The ditch network also does not differentiate between municipal and privately owned property. While it produces very good results for the available data, the ditch costs are still not always accurate.
The exact clustering and Steiner Tree algorithms used by FiberPlanIT have not been disclosed, so we are limited in the evaluation of their graph optimization phase.

2.2 Steiner Tree

The Steiner Tree problem (STP) is one of the 21 original NP-hard problems discussed by Richard Karp [6]. NP-hardness of STP was later also proved for grid and planar graphs by Garey and Johnson [7].
In the Steiner Tree problem, we are given a graph or space with distance function and a set of terminals. A Steiner tree is a tree of minimal weight that contains all terminals, and may also contain additional vertices, called Steiner points.

Along with the applications in infrastructure, there are many other applications, such as in chip design, modeling molecular structures and in constructing phylogenies. Given the many different applications, many algorithms have been made that are specifically optimized for those applications.

We first consider a problem variant that accounts for the path lengths to a source node along with the edge costs. In this variant, we not only consider the cost of each edge, but also the distance in the tree from each terminal to the source node in the minimization function. The variant has not been studied as thoroughly as the Steiner tree problem in graphs. The approaches often focus on obtaining a Steiner tree first and then use post-processing to decrease path lengths. An example is the greedy algorithm to balance minimum spanning trees (MST) and shortest path trees by Khuller, Raghavachari and Young [8]. This approach takes a MST and performs post-processing steps to transform it into an intermediate tree. For a ratio γ it produces a tree that is a $1 + \sqrt{2} \cdot \gamma$-apx MST and a $1 + \sqrt{2}/\gamma$-apx shortest path tree. Bharath-Kumar and Jaffe [9] investigated the trade-off in path costs and edge costs for Steiner trees for a single root, also by applying post-processing to a Steiner tree. The approach of Khuller et al.[8] was altered for Steiner trees by Lin and Xue[10], obtaining similar results.

The Steiner tree variant in euclidean non-uniform space does not have significant results yet to the best of our knowledge. A rough grid-based approximation using a genetic algorithm is presented by Frommer et al. [11], but does not provide guarantees on the distance from optimal solutions and already takes 15 minutes for an approximate solution for instances with 15 terminals.

The most common variant of the Steiner tree problem is the Steiner tree problem in graphs. Here, we disregard any path and facility costs and only focus on minimizing the sum of edge costs to construct a tree containing each terminal. The Steiner tree problem in graphs has many different approaches and recent improvements.

2.2.1 Exact Algorithms

Exact algorithms for NP-hard problems generally specify the running time using an additional parameter k along with the input size n. A fixed-parameter-tractable algorithm is then an algorithm with running time $O(f(k) \cdot n^{O(1)})$, where f is an arbitrary function depending on k, which is generally exponential. Running time indications of FPT algorithms typically use O^* notation, which hides polynomial factors, or $\tilde{O}$ notation, which hides polylogarithmic factors. The running time $O(2^k \cdot n \log^2 n)$ will then be shown as $\tilde{O}(2^k \cdot n)$ or $O^*(2^k)$.

Steiner Trees are typically parameterized by the amount of terminals k, input nodes n and input edges m. The fastest exact algorithm was for a long time the Dreyfus-Wagner dynamic-programming algorithm [12] with improvements by Erickson, Monma and Veinott [13], which has a running time of $O(3^k \cdot n + 2^k \cdot n \log n + m)$ using exponential space.
In 2006, Mölle, Richter and Rossmanith [14] improved this to $O((2+\epsilon)^k \cdot n^{O(1)})$. While asymptotically faster, the polynomial term is much larger and makes it unpractical in practice. Björklund et al.[15] gave the first $\tilde{O}(2^k \cdot n^2 + nm)$ algorithm by applying fast subset convolution. Finally, Nederlof [16] presented a polynomial space algorithm with running time $O(2^k \cdot n^{O(1)})$ in 2009 under bounded edge-weights.

There are also other parameterizations for the Steiner Tree problem. The amount of Steiner points, which is the amount of non-terminal vertices required to create a Steiner Tree, is sometimes used, but is not applicable in our case. The Treewidth tw of a graph, a measure indicating how much a graph differs from a tree, can be used in a dynamic-programming algorithm to obtain $O(2^{O(tw)} \cdot n^{O(1)})$ and better running times [17]. Because the treewidth of planar graphs is unbounded, this parameterization is also not applicable in our case.
The Steiner Tree problem is also well-studied for unweighted planar graphs. In 2014, a polynomial kernel in the amount of terminals and Steiner points was found by Philipczuk et al. [18] which paved the way for a sub-exponential time algorithm. It was further improved by Suchý [19] by applying additional reduction rules to the problem.
Since we expect our instances to typically require at least one DP with around 48 customers, these exact methods are not feasible for our problem. We must make do with methods that may not give an optimal solution.

2.2.2 Approximations and Mixed Integer Linear Programs

The prevalent approximation algorithm for the Steiner tree problem uses the shortest path heuristic and is based on Prim's algorithm for Minimum Spanning Trees. At each iteration, the minimum cost path between pairs of terminals that are not yet connected is added, which results in a 2-approximation of an optimal Steiner Tree [20].

The current best approximation algorithm for the Steiner tree problem is by Byrka et al.[21] and gives a 1.39-apx. It is NP-hard to find solutions within $\frac{96}{95}$ [22].
Along with the listed exact and approximation approaches, there also exist Mixed-Integer Linear Programming (MILP) approaches. Based on a cost function and a set of constraints, a massive solution search space is created, with the goal of either maximizing or minimizing the cost function. The search space can be reduced by applying *reduction rules*, which remove invalid solutions and parts of the problem that cannot be in optimal solutions. More advanced reduction rules have been developed over the years for the Steiner tree problem, as seen in the paper by Uchoa et al. [23] *Branch and cut* strategies are applied in MILP to find lower bounds for a solution, which can be used to find optimal subproblems. This decreases the search space further.
The 11th DIMACS implementation challenge [24] was dedicated to Steiner tree problems and has led to the creation of multiple MILP based implementations for different variants of the problem. The best solutions for the Steiner Tree problem in Graphs based on their metrics were PUW[25], SCIP-Jack [1] and Mozart/Steynerd [26]. These MILP approaches provided optimal solutions to many large datasets with hundreds of terminals that previously did not have a proven optimal solution. They often provide near-optimal solutions to very large instances within a couple minutes.

2.3 Facility Location

In the facility location problem we try to place an undetermined amount of facilities to service the demand from customers such that the sum of fixed setup costs for the facilities and the variable cost of serving the customers is minimized.
The facility location problem occurs in many different applications, such as placing stores near many customers, finding locations for hazardous materials far away from people, placing fire

stations to minimize response time and many others. It is a broad field with many different objective functions. The predominant objective functions are the *minisum* and *minimax* functions [27]. In a *minisum* problem, we aim to minimize the sum of paths lengths from customers to their facilities. The *minimax* problem instead aims to minimize the maximum distance of customers to facilities.
Since our aim is to minimize the cable lengths, we will only consider the *minisum* problem, which is commonly referred to as the *median* problem.
Kariv and Hakimi [28] first showed that the p-median problem, in which at most p facilities can be placed, is NP-hard for general graphs. They then presented a polynomial time algorithm to solve the p-median problem in trees.
The algorithm was improved by A. Tarim [29] to obtain a running time $O(p \cdot n^2)$ for n nodes and at most p facilities.
The median problem has many existing approaches on general graphs for both capacitated and uncapacitated variants. An overview of the methods is given by J. Reese [30].
The capacitated p-median algorithm of Yaghini, Karimi and Rahbar [31] provides near-optimal solutions to instances with 200 customers and up to 80 facilities in the order of minutes. The algorithm given by Stefanello et al. [32] solves instances with 737 customers and 74-148 medians within 10-20 minutes and solutions to instances with 3038 customers and 600-1000 medians within 30-45 minutes.
To the best of our knowledge, the capacitated median problem in trees has not been studied before.

Chapter 3

Preliminaries

In this chapter we will take a closer look at the constraints and costs of a fiber network. We consider the available datasets and their structure, then define the problem description more formally. We also describe several geometric operations we use when constructing our graph.

3.1 Network Constraints and Costs

Constraints

The precise network properties are defined by the network operators. We will be following the constraints as described in the *ReggeFiberProgramma van Eisen* (PvE).
The relevant constrains are as follows:

CR1	A DP is connected to an AP using a 96 fiber cable.
CR2	A CA is connected to a DP using a 2 fiber cable.
CR3	A DP can facilitate at most 48, 24 or 12 CAs depending on its size.
CR4	Large buildings with at least 48 CAs receive their own DP.
CR5	The customer's cable forms an uninterrupted connection from DP to CA.
CR6	Cables on private property must be placed by drilling, unless the distance is less than 2m.
CR7	Cables from different DPs cannot be placed in the same ditch.
CR8	Cables from AP to DP cannot cross other cables from AP to DP in opposite directions.
CR9	Cables crossing roads require an additional protective duct.
CR10	Paving and road verge must be restored back to original condition after digging.
CR11	Cables crossing asphalt roads must use a drilling.

We also need to consider the width of a ditch. These are typically around $60cm$ wide. A line representing a ditch must then be placed at distance greater than $30cm$ from any buildings and obstructions.

In a network plan, ditches are typically placed at $40cm$, $60cm$, or $80cm$ from obstructions.

Costs

In addition to the constraints, there are many different costs when constructing a fiber optics network.
These can be divided into area, material and labor costs.

Area Costs: The area costs are incurred when digging through non-municipal areas. They require extra permissions or permits. A customer will naturally give permission to dig or drill on their property for their own connection when registering for a connection, but not for other connections. Customers with adjacent front doors sometimes only require a single ditch. Otherwise digging through private property to connect other customers is almost never feasible due to extra cost, time and work required to obtain permission.

Some areas require expensive permits, for example railroads or airfields. These costs are generally incurred per ditch or drilling and are up to thousands of euros.

Labor Costs:

- Digging: Digging a ditch involves creating a long 60cm deep ditch where cables can be placed. Roughly €10 per meter of ditch length.
- Drilling: Drilling involves mechanically pushing a metal tube through the ground. The tube can later be used to place cables into. Costs vary between €35 and €60 per meter.
- Restoring area: Once cables have been placed, the area must be restored to original condition. First the ditch needs to filled back up, then the original coverage must be placed back. The costs differ greatly between coverage type. For grass, it only requires the sods to be placed back, whereas paving requires more work. Most paving costs around €4 per meter.
- Placing Cables: Each cable needs to be placed in the ditch network from DP to its CA. It costs around €0.50 per meter of cable.
- Placing Ducts: Cables crossing roads must be placed inside ducts for protection. Ducts are also required when trees are nearby, to protect the cables from the expanding roots. This costs approximately €1.50 /m.
- Placing Distribution Points: This requires a large hole to be dug, all the relevant cables to be connected and the hole to be restored afterwards. Since they are placed inside ditches, we can mostly ignore digging and restoration costs. The cost of placing a DP is then around €150.

Material costs:

- Distribution Points: The cost depends on the capacity. For 48 customers around €800, followed by ∼€400 for 24 and ∼€200 for 12.
- Customer Access Points: Around €25.
- Cables: The cables are very cheap and mostly consist of the plastic protection around the fibers. A 2 fiber and 96 fiber cable don't actually differ much on the outside and are about the same width. Both cost around €0.15/m.
- Ducts: €1/m.
- Cable welds: When fiber cables need to be joined, a protective enclosure needs to be applied to the weld. It costs around €0.20.

We combine the costs into the following categories: Ditch cost, DP cost, Cable cost and CA cost.
For the ditch cost, we can add up everything that has a per meter cost that is incurred once per ditch. This combines the digging, drilling, restoring, duct placing and duct costs into a single per meter cost depending on the coverage type. If a given ditch is inside an area with extra costs, we can add those extra costs here as well.
The DP cost depends only on the placing cost and material cost of the DP. We note that larger DPs are cheaper per customer connection, but require longer cables to reach each customer. DPs with capacity less than 48 customers are generally only used in rural areas when the distance between customers is relatively large.
The cable cost is given by the material cost for the cables and cost of placing cables.
CA cost is given by the material cost of the access point.

These costs form our main metric in the creation and evaluation of solutions. Given accurate cost specifications for each item on the list, we can accurately calculate the cost of a proposed network, as well as try to minimize these costs when designing networks. These costs will therefore form the minimization objective function for our algorithm.

3.2 Dataset Specification

To be able to create accurate and representative networks we require accurate geographic data of an area. The Netherlands' Cadastre collects and registers such data and provides the *Basisregistratie Grootschalige Topografie*(BGT) [33] and *Basisregistratie Adressen en Gebouwen* (BAG) [34] as open datasets. The BGT provides the geographical locations of physical objects such as buildings, roads, green space, waterways and railroads in a consistent format. The BAG provides the locations for all addresses in the Netherlands.
Along with the BGT and BAG, we will use the Inspire RoadLink (RL) road dataset to determine the names of roads in the BGT.

We will make use of the following features from the BAG, BGT and RL datasets:

Name	**Type**	**Description**
Adres	Point	Location and information of addresses
Wegdeel	Polygon	Polygons for each distinct road section
Ondersteunend wegdeel	Polygon	Polygons for road verge and traffic islands
Onbegroeid Terreindeel	Polygon	Paved areas that are not roads, including private property
Begroeid terreindeel	Polygon	Areas with vegetation, such as road verge, forest and fields
Waterdeel	Polygon	Waterways and bodies of water
Ondersteunend Waterdeel	Polygon	Areas beside or sloping down to water areas.
Pand	Polygon	Buildings
RoadLink	Polyline	Major roads represented by lines
Ruimtelabel	Point	Contains name labels drawn on maps

Along with coordinates, each object in a feature has attributes to convey additional information.
The *adres* feature contains attributes such as streetname, house number and postal code. It also contains a *pandid* to relate it to a building in the 'pand' feature. There is also an attribute for the function of the building, such as *residential*, *industrial*, *office*, *education* or other functions.

Each of the polygonal features has an attribute *bgt_status*, indicating whether the object exists, is planned to be constructed or is under construction.
We only consider existing structures.

The *pand* feature contains additional attributes such as *id*, and *year of construction*.

The *wegdeel* features carry information about the *function* of the object and the *physical appearance*. These attributes are mandatory and listed in the Object Manual[35]. Some examples of the function are *bicycle lane*, *local road*, *airplane runway* or *railroad*.
The possible physical appearances are *closed paving*, such as asphalt or concrete, *open paving*, such as pavers and tiles, *half-paved*, like gravel, *unpaved*, and *vegetation*, such as grass, plants or trees.
There are 2 additional optional attributes named *plus function* and *plus physical appearance*, providing more detailed information on the function and appearance of objects. It can for instance specify whether a road segment with closed paving is made of asphalt or concrete.

The *terreindeel* features contain the same attributes as the *wegdeel* feature, except for the function. For private property, the physical appearance is *erf*. Otherwise, it's one of the options listed above.

The *waterdeel* and *ondersteunend waterdeel* features carry similar information on the type of the area, indicating whether the object is a river, lake or river bank for example.

There is an additional attribute for all polygons except buildings called *relative height*. In cases where objects share the same area, the upper one will have a higher value than the lower one.

The *RoadLink* feature has attributes indicating the street name and municipality. It also contains information on the house numbers associated to it, though this does not correspond exactly to the numbers of the *adres* feature. Each object also has a start and end node, such that adjacent roads can easily be identified.

Finally, the *Ruimtelabel* feature also provides some street names. These are normally used to indicate the name of the road when visualizing the map to indicate the street names. The amount of labels is very limited and most road segments do not have a label.

3.3 Problem Description

Now that all constraints, costs and usable data have been described, we can more formally describe our problem of finding the cheapest way of connecting each customer to a DP and connecting the DPs together.
Since we aim to connect all customers, the cost of customer access points is the same in any valid solution. We therefore don't include these costs in our definition.
Minimizing the ditch cost in the graph while keeping all customers connected gives the Steiner tree problem.
Minimizing DP cost and cable costs for a given graph where a DP has limited capacity gives a capacitated median problem.

As shown in chapter 2, the Steiner tree problem does not have significant results for euclidean space with a non-uniform metric. Since our problem involves a more complicated cost function, we think a direct solution on the plane is not feasible.
Therefore, we divide our problem into 2 separate problems: A geographic problem, in which the datasets are used to generate a representative graph of the area containing potential ditches with a weight, and a minimization problem on graphs.

Geographic Problem The geographic problem involves transforming the input data from euclidean space to an edge-weighted graph.
For any Steiner tree on the graph, the cost of its edges must closely represent the digging or drilling cost of that ditch in the real world, such that the total construction cost of a Steiner tree closely approximates the total cost of constructing that network in the real world.
For any efficient network in the real world, we want the constructed graph for that area to contain a network with similar structure and costs.
To capture this more formally, we compare the cost of an optimal Steiner tree on the graph to the cost on the weighted plane.
Consider a set of points P in a weighted polygonal subdivision. The cost of a path within a simple polygon is defined as the length of the path multiplied by the weight of the polygon. The cost of a path in general, is the sum of the costs of its segments in each polygon.
A network G can be constructed with vertices $V \supseteq P$ and edges E, where the cost of an edge is determined by the straight line path between its vertices. We define the dilation of a pair of points $u, v \in P$ with $u \neq v$ as the ratio between the lowest cost weighted path in the polygonal subdivision $\pi(u, v)$ and the lowest cost path in the network $dist(u, v)$, giving $\Delta_{(u,v)} = \frac{dist(u,v)}{\pi(u,v)}$.
The dilation of a network $G = (V, E)$ is given by the maximum edge dilation in the graph. $\Delta_G = \max_{\substack{u,v \in V \\ u \neq v}} \frac{dist(u,v)}{pi(u,v)}$.
Since a graph G containing a vertex for each terminal and the shortest weighted path to all other terminals gives $\Delta_G = 1$, there always exists a 2-approximation of the weighted euclidean Steiner Tree in this graph [20].

However, this dilation metric doesn't account for Steiner points and therefore can't be directly related to the quality of our graph, as shown in figure 3.1.

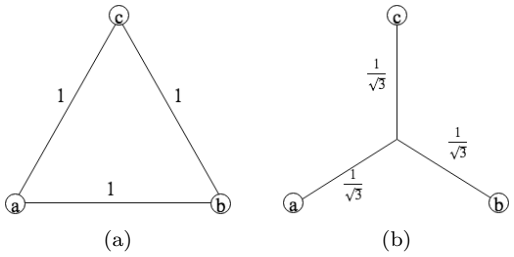


Figure 3.1: (a) has $\Delta_G = 1$ and a Steiner tree with cost 2 whereas (b) has $\Delta_G = \frac{2}{\sqrt{3}}$ but a Steiner tree with cost $\sqrt{3}$

We define the Steiner dilation with terminals T as the ratio between the cost of a minimum-weight Steiner tree on the weighted subdivision and the minimum cost of a Steiner tree on a graph embedded on the subdivision.
Given a weighted polygonal subdivision and terminals T, we want to construct a graph $G = (V, E)$ with terminals $T \subseteq V$ embedded on the polygonal subdivision where the cost of an edge equals the cost of the line segment in the subdivision, such that there exists a Steiner tree F' in G minimizing $\Delta_G^S = \frac{cost(F')}{cost(F)}$ for any minimum cost Steiner tree F on the subdivision.
The size of the graph should be limited based on the size of size of the subdivision to keep the graph minimization problem manageable. More specifically: For a subdivision with n line segments, we want $|E| = O(n + |T|)$.

The output consists of a weighted graph $G = (V, E)$ of potential ditches and a set of terminals T. Each edge e has a *weight* $w(e)$ indicating the cost of creating that ditch in that area, and a *length* $l(e)$ giving the length in meters of that edge. The weight of an edge within a simple polygon is defined by the length of the edge multiplied by the weight of the subdivision and the one-time costs. For an edge in general, it is defined by the sum of costs of its segments in each polygon and

the one-time costs.
Each vertex $v \in V$ has a *demand* $d(v) \in \mathbb{N}_0$, indicating the amount of customer access points on that vertex. The set of terminals T is then defined as each vertex with a positive demand: $T = \{v \in V | d(v) > 0\}$.

Graph Problem The second phase involves a graph minimization problem based on the graph from the geographic problem and the cost categories defined in section 3.1.
The cost categories described earlier now need to be minimized. Along with the graph G, edge weights $w(e)$, lengths $l(e)$ and vertex demands $d(v)$ defined above, we are given a DP cost C_d, maximum DP capacity k and cable cost C_c.
In our formal definition we will assign demand from each terminal to a specific DP placed on a vertex. We define the path $\pi(u, v)$ of an assignment $f(v, s)$ for terminal v and DP s as the shortest path in the output graph F, where the distance $dist(u, v)$ is determined by the sum of lengths of its edges.

We then define our graph problem as follows:

STEINER TREE CAPACITATED MEDIAN PROBLEM

Input: Graph $G = (V, E)$.
Weight and length functions $w(e) \geq 0$, $l(e) \geq 0$ for all edges.
Node demand $d(v) \in \mathbb{N}_0$ for all nodes.
Maximum facility capacity $k \in \mathbb{N}^+$.
Facility cost $C_d > 0$ per facility.
Cable cost $C_c > 0$ per meter.

Let terminals $T = \{v \in V \mid d(v) > 0\}$ and let total demand $D = \sum_{v \in T} d(v)$

Solution: Tree $F = (V', E')$ with $V' \subseteq V$, $E' \subseteq E$ and $T \subseteq V'$.
Set of facilities S with their location $loc(s) = v \in V'$.
For each node $v \in T$ and facility $s \in S$ an assignment $f(v, s) \in \mathbb{N}_0$ assigning that amount of demand from v to $s \in S$, such that $\forall v \in T : \sum_{s \in S} f(v, s) = d(v)$ and $\forall s \in S : \sum_{v \in T} f(v, s) \leq k$.

Minimizing: $\sum_{e \in E'} w(e) + |S| \cdot C_d + \sum_{v \in T} \sum_{s \in S} f(v, s) \cdot dist(v, s) \cdot C_c$

In our setting, we expect the largest cost in our solution to be the total ditch cost. We also note that there are many high-performance solvers for the Steiner tree problem, while there are no solvers when combined with the facility location problem to the best of our knowledge. Therefore, we decided to focus on obtaining a near-optimal Steiner tree and deciding on the placement of facilities afterwards. While the cost of cables can theoretically increase by a factor $O(k)$ [9] for each DP network without significantly increasing ditch cost, we expect that post-processing algorithms [10][9] applied to each DP network can be used to bound the distance to an optimal solution.

For the facility location problem, we do not consider the weight of edges, but only the length. Since the DP cost and facility cost are both constant, we can scale both costs by $\frac{1}{C_c}$ to obtain C'_d and C'_c and get an equivalent solution. Since $C'_c = 1$, we can remove this factor in our minimization function. Let $C = C'_d$.

We then split our graph problem into the following 2 problems:

STEINER TREE PROBLEM

Input: Graph $G = (V, E)$.
Weight function $w(e) \geq 0$ for all edges.
Node demand $d(v) \in \mathbb{N}_0$ for all nodes.

Let terminals $T = \{v \in V \mid d(v) > 0\}$

Solution: Tree $F = (V', E')$ with $V' \subseteq V$, $E' \subseteq E$ and $T \subseteq V'$.

Minimizing: $\sum_{e \in E'} w(e)$

Capacitated Median Problem

Input: Tree $G = (V, E)$.
Length function $l(e) \geq 0$ for all edges.
Node demand $d(v) \in \mathbb{N}_0$ for all nodes.
Maximum facility capacity $k \in \mathbb{N}^+$.
Facility cost $C > 0$ per facility.

Let terminals $T = \{v \in V \mid d(v) > 0\}$ and let total demand $D = \sum_{v \in T} d(v)$

Solution: Set of facilities S with their location $loc(s) = v \in V$.
For each node $v \in T$ and each facility $s \in S$ an assignment $f(v, s) \in \mathbb{N}_0$ assigning that amount of demand from v to $s \in S$, such that $\forall v \in T : \sum_{s \in S} f(v, s) = d(v)$ and $\forall s \in S : \sum_{v \in T} f(v, s) \leq k$.

Minimizing: $|S| \cdot C + \sum_{v \in T} \sum_{s \in S} f(v, s) \cdot dist(v, s)$

Applying the output of a Steiner tree algorithm to a capacitated median algorithm will then produce a valid heuristic solution to our graph minimization problem.

3.4 Geometric Operations

To solve our geographic problem, we use several common geometric algorithms and operations on points, line segments, polylines, polygons and subdivisions.

A *point* contains x and y coordinates, which are real numbers. A *line segment* is defined by 2 points and contains every point that line, including both endpoints. We then define a *polyline* as a connected series of line segments, defined by a sequence of points. A *polygon* is an area bounded by a polyline in which the start and endpoints coincide. In this work, we only consider simple polygons without holes or self-intersections.
A *subdivision* is a structure induced by a set of line segments only intersecting in common endpoints. It contains vertices at the endpoints of segments, edges are the interiors of the line segments, and faces being the interiors of connected 2-dimensional regions not containing any point or line segment.
Each of the geometric types can also store *attribute* information. The faces in a subdivision can for instance store the coverage type of that location.

The Computational Geometry book by M. de Berg, O. Cheong, M. van Kreveld and M. Overmars [36] describes efficient and easy to understand algorithms for most of our operations. While these algorithms are not necessarily the most efficient solution available, they already provide a good bound on the running times.
We use the implementations provided by the Feature Manipulation Engine (FME)[37]. Unfortunately, we discovered that FME does not use an efficient algorithm for some of the operations, causing running times in practice that are much longer than required.

Line Intersection

Problem: Given a set of line segments S with closed endpoints in the plane, report all intersections between segments and split segments at the intersection points.

Running Time: $O(n \log n + I \log n)$ for n line segments and I intersection points in S. [36, Thm. 2.3]

OVERLAY [36, Ch 2.3]

Problem: Given two planar subdivisons S_1, S_2 with complexities n_1, n_2, compute the subdivision S induced by the edges of S_1 and S_2.
The complexity is given by the sum of edges, vertices and faces of a subdivision.

Running Time: $O(n \log n + k \log n)$ where $n = n_1 + n_2$ and k is the complexity of the overlay. [36, Thm. 2.6]

The overlay between a subdivision S_1 and set of line segments S_2 can be computed similarly.

We can use the overlay to create boolean operations between polygons P_1, P_2. The UNION $P_1 \cup P_2$ gives the faces labeled with P_1 or P_2. The INTERSECTION $P_1 \cap P_2$ gives the faces labeled with P_1 and P_2. The DIFFERENCE $P_1 \backslash P_2$ gives the faces labeled with P_1 and not P_2.

NEAREST NEIGHBOR

Problem: Given a set of points P and polygonal subdivision S, give the closest line segment for each point in P within radius r.

Running Time: R-trees [38] give Linear storage and linear worst-case query time. The Query time is much better in practice [39].

OFFSET

Problem: Given a polygon P with n edges and distance d, give the area within distance d of P. This is similar to the Minkowski sum of a polygon and a circle with radius d.

Running Time: We can use the convolution method to obtain $O(n)$ running time [40].

Instead of using the exact circular arcs given by the offset method, we allow the gaps between offset segments to be closed by extending the lines until they intersect, or to be connected directly if the angle is large. This does not produce the exact offset, but simplifies the geometry and suffices in our setting.

DISSOLVE

Problem: Given a subdivision S with complexity n, merge adjacent faces with the same label l.

Running Time: Requires removing edges with the same label on both faces. $O(n)$.

SIMPLIFY

Problem: Given a polyline P with n segments and threshold ϵ, produce a minimum link polyline P' that is always within distance ϵ of P.

Running Time: $O(n^2)$ using Imai-Ira algorithm with improvements from Chan & Chin [41].

Chapter 4

Geographic conversion

In this chapter we show how we transform the geographic data into a representative graph. We first describe a base model and show how it is related to the Steiner dilation. Later, we modify the base model to decrease the graph size and improve the quality of the graph in practice.
While we cannot provide an exact bound on the Steiner dilation of the graph, we show for each part of our process how the constructed edges relate to similar bounded Steiner tree solutions.

4.1 Model

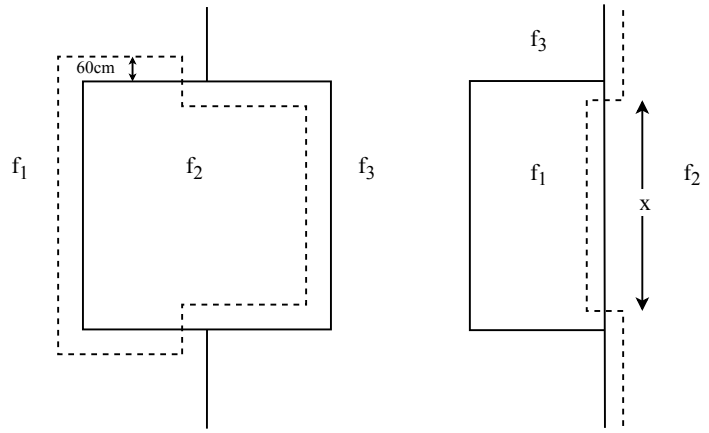
In our model we construct a graph based on a set of potential ditches D. A potential ditch is a line segment given by 2 points and weighted by the sum of the length of the line segment multiplied by the weight of the polygon it is in for each polygon it passes through. We divide the creation of our set of ditches D into three parts: Generating offset lines for all polygons D_O, creating crossings D_C, and connecting customer access points D_A.

We will first reorder the features given by the BGT dataset. Let BGT_R be the set of *wegdeel* polygons that are not railroads or airstrips, and the set of *ondersteunendwegdeel* polygons that are traffic islands.
Let BGT_V be the set of *begroeidterreindeel* polygons, and the *ondersteunendwegdeel* polygons that are road verge.
Let BGT_P be the set of *pand* polygons, and the set of *onbegroeidterreindeel* polygons that have the physical appearance attribute erf.
BGT_U is the set of polygons that are typically avoided. It contains the railroad and airstrip polygons from the *wegdeel* feature and the polygons of the *waterdeel* and *ondersteunendwaterdeel* features.
Each polygon also receives a per meter *ditch cost* attribute based on the coverage type and function, as described in section 3.1, given by $c_d(p) \in \mathbb{R}_{>0}$ for a polygon p.

4.1.1 Offset Lines

With our offset lines we aim to create a set of lines for each polygon giving a cheap way of digging a ditch traveling along the perimeter of the polygon, based on the adjacent polygons. If the area outside the polygon is cheaper to dig in, we place a ditch 60cm outside of the perimeter of the polygon. Otherwise, we place it at 60cm inside the polygon. Different adjacent polygons may be encountered while traveling around the perimeter, so it may be required to switch from the outer line to the inner line or vice versa, as shown in figure 4.1a.
For small polygons or adjacent polygons with similar costs, the added cost of switching to the outside and back to the inside may exceed the costs of placing the ditch inside the polygon, even

though the ditch cost inside is larger.
The difference between switching to the outer line and staying on the inner line as in figure 4.1b depends on the length of x and the difference in ditch cost between f_1 and f_2. Let $k = \frac{d_c(f_2)}{d_c(f_1)}$, then if $1.2k + 1.2 + x \leq x \cdot k$, switching to the outside will give a cheaper offset line than staying inside. For example, $k = 1.5$ will require $x \geq 6$ for the switch to be cost effective.



(a) Offset line traveling around the perimeter of f_2

(b) Switching may lead to higher costs when x is small

Figure 4.1: Examples of offset lines and switching when the ditch cost of $f_1 < f_2 < f_3$

We can create the offset lines by generating the offset at $60cm$ for each polygon and creating subsets by ditch cost. Then we repeatedly take the difference between the cheaper subset and the union of the more expensive subsets.
Let the set of ditch costs be given by $C_d = \{c_1, \ldots, c_n\}$ such that $\forall p \in BGT, c_d(p) \in C_d$ and $\forall c_i, c_j \in C_d : i < j \implies c_i < c_j$. The subsets by ditch cost are then given by $P_i = \{p \in BGT \mid c_d(p) = c_i \in C_d\}$.
For each P_i, we compute its offset at 60cm, giving P'_i.
The offset lines for P'_i are given by $D_{O,i} = P'_i \setminus \bigcup_{\substack{c_j \in C_d \\ j>i}} P'_j$.
The set of offset lines is then given by $D_O = \bigcup_{c_i \in C_d} D_{O,i}$.

We then have that each polygon has an outer line wherever the cost outside the polygon is lower, and an inside line, created by the adjacent polygon, wherever the cost is lower inside. When adjacent polygons have the same cost, they will have both the inner and outer lines, which intersect near the boundary to other polygons.
Computing the overlay results in a fully connected graph where we can find a cheap ditch between any 2 points on the graph that travel along the boundary of the polygons, apart from a few uncommon cases where polygons are contained in other polygons. Most of these cases are covered when creating crossing ditches.

We can compute D_O efficiently by computing the overlay of all offsets together, then dissolving faces when they have a label with the same polygon *id* and the highest ditch cost label is the same. If we assume the minimum width of all polygons exceeds $1.2m$, the complexity of the overlay increases by a constant factor of the amount of edges in the BGT subdivision, as the offset will then only intersect adjacent polygons.
In practice the minimum width is quite often less than $1.2m$, but the amount of other offsets of polygons encountered per edge is still expected to be a low constant.

For many coverage types, the ditch cost is very similar. In these cases, the cost of switching

between inner and outer line will almost always exceed the cost of the line traversing through the more expensive coverage type. Instead of creating subsets P_i for each different ditch cost c_i, we will only create the subsets P for BGT_V, BGT_R and $BGT_P \cup BGT_U$, and apply the same process as above. This will produce a much larger graph, but will also produce better solutions when minimizing the cost on the graph.
We decrease the size of the graph by dissolving all offsets given by $BGT_P \cup BGT_U$ first. Since the ditch cost of $BGT_P \cup BGT_U$ is the highest, a line in the resulting set of offset lines will never be placed inside any of these high-cost polygons. In practice, crossing a polygon from this set is only very rarely required, for example when a river or railroad separates the project area and there is no nearby bridge or crossing. We consider these cases more specifically in section 4.1.3.

4.1.2 Customer Connections

Customer access points given by the address points are located at an arbitrary point inside a building, which is in turn usually located on private property. Our goal is to connect the access point through the front door of a building straight to a road. The cable from a customer should only pass through their private property and municipal areas, since a large cost will be incurred when passing through other property.
While the access points all contain an address, none of the polygonal features contain any information that allows us to relate an address to a specific road or a private property to a specific building. It also does not provide a notion of a front door. For our base model, we will connect each address point with a straight line to the nearest neighbor given by offset lines within or produced by a road segment, giving D_A.

4.1.3 Crossings

While the offset lines and customer connections allow us to connect all customers, only traveling around the boundary of polygons will lead to very expensive solutions. Crossing a polygon is often required to obtain efficient solutions.
However, there are also cases where crossing a polygon is never cheaper than traversing the boundary.
For any polygon with an outer offset line of length l, minimum width w, where the ditch cost of the polygon is k times the ditch cost of its adjacent polygons, we state as a rule of thumb that crossings are only required if $w < \frac{l}{2k}$. Otherwise, for any crossing, we expect there exists a lower cost path around the boundary of the polygon.
The ditch cost of the polygons in $BGT_P \cup BGT_U$ is typically at least 5 times higher than roads or vegetated areas, so crossings are only viable for long and narrow polygons in this set. These long and narrow polygons are common for railroads and waterways, though these don't usually exist within a neighborhood, and most fiber network projects do not contain these, so they were not included in this model.
On the other hand, when k is small, or whenever an inner offset line is available, the graph should contain crossings to allow for efficient Steiner tree solutions.
We create crossings wherever a customer connects to the network by extending the line given by the customer access point and the nearest neighbor until that line touches an edge of the offset lines for $BGT_P \cup BGT_U$ or BGT_V. The resulting line is split whenever it intersects other offset lines.

For a rectangular road polygon with customers connected to its boundary, the result of this operation is similar to the Hanan grid [42].
The Hanan grid is constructed by creating horizontal and vertical lines through each terminal, where each intersection creates a vertex. The Hanan grid is known to contain an optimal rectilinear Steiner tree.
Our process produces a similar graph when the road polygon is rectangular. The main differences being that the terminals are connected to the road segment first, and that the distance metric is not uniform.

As described above, digging through private property for another customer is not feasible, so the straight line connection between terminals through private property and buildings is not required. Since our terminal is then always a leaf, we can contract its edge because it is always required in an optimal Steiner tree solution [23]. Applying this leaf reduction rule then places our terminals on the boundary of the road segment.
A graph constructed using our model for a rectangular road polygon will then contain an optimal rectilinear Steiner tree with a Steiner dilation of $\sqrt{2}$. For large portions of our input data this bound applies in practice. Outside of road intersections the roads are often rectangular.

There are many polygons in BGT_R that have no customers nearby, but that are required for an efficient Steiner tree. Because these polygons will not have any crossing lines formed by the addresses, the only way to pass them is around the boundary. To resolve this, we add additional crossing lines in the following way:
Each of the polygons in BGT_R has its offset lines chopped into l meter segments. For each segment, we construct a line perpendicular to the first edge of the segment that lies within the area given by the offset and ends at the boundary. The length l should be set such that for any 2 points on the boundary of the polygon's offset, the edge dilation is bounded by some constant.
In figure 4.2, the dilation between any 2 points on the boundary for the rectangle is bounded by $\frac{d+l}{d}$, since there is always a crossing available within $l/2$ meter. For roads in residential areas, the minimum width of a road is given by $4.5m$, and is typically accompanied by footpaths with a width of at least $1.5m$. Setting $l \leq 6m$ will result in a dilation ≤ 2 for rectangular areas. We cannot provide this bound on the dilation for all polygons in the dataset, but note that in practice the model has produced very good results.
Because ditches crossing roads require extra protection, the per meter cost of the crossings is increased by the cost of protective covers.

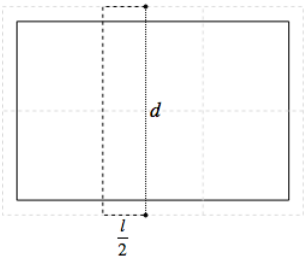


Figure 4.2: Example of crossings for a rectangular polygon. The shortest euclidean line is compared to the shortest path in the graph.

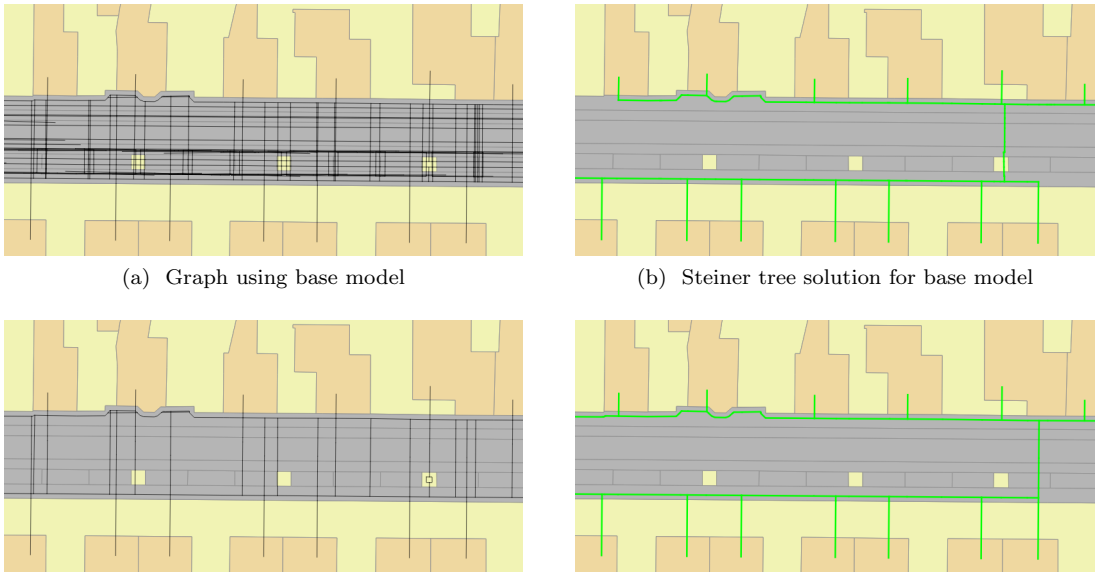
The graph given by the offset lines, customer connections, and crossing lines then gives a connected graph, in which we can either travel around areas or cross them to reach any other customer access point. In practice, the graph contains sections similar to the Hanan grid.

4.2 Adaptations to base model

The base model presented often produces very large graphs that take long to construct which in turn require a lot of time to compute a Steiner tree on. Inspecting the results of a Steiner tree solution shows us there are many redundant edges in our graph. We also note several issues at road intersections and with the customer connections.
We apply several adaptations to our base model to add missing edges where required, but also remove a significant amount of edges that are not required. The result is a smaller graph that allows for faster computation with similar costs.

4.2.1 Removing redundant offset lines

Inspection of our graph and a resulting Steiner tree solution shows that a majority of the offset lines is never required. Whenever a road has adjacent footpaths or parking spaces, the base model will produce offset lines inside and outside each of those polygons, though an efficient Steiner tree solution will not use these, as shown in figure 4.3 (a) and (b).
Given a road that has a footpath and parking lanes on both sides, a line crossing that road will intersect 12 offset lines. A Steiner tree solution typically only uses the outermost 2 lines.
This follows from the properties of a Steiner tree. An efficient solution picks a line on both sides of the road to connect all customers on that side efficiently, then crosses the road to connect both lines.
Picking a line that is further from the customers would require each customer to cover that distance, while only slightly decreasing the length of the crossing. Picking the outermost line is always cheaper when customers are close together and the coverage types are the same.
When the coverage types are different, the outermost line often has the cheapest coverage type. In The Netherlands, the sidewalk is almost always open paving, and road verge is cheaper still.
We could opt to use more ditch cost subsets when creating offset lines, resulting in a decrease of offset lines up to 50% by producing at most 1 offset line per polygon. However, dissolving all roads before producing offset lines will always give just the outermost lines. The effect of this reduction is shown in figure 4.3(c) and (d). Experimental shows that the size of the graph is reduced by over 70% while the cost of a solution only shows an increase of up to 1.4%.



(a) Graph using base model

(b) Steiner tree solution for base model

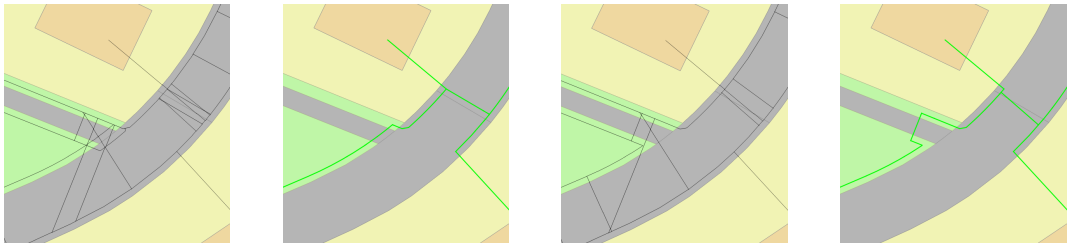
(c) Graph after dissolving all roads

(d) Steiner tree solution for dissolved roads

Figure 4.3: Effect of dissolving all roads on graph size and solution

The increase in cost has 2 reasons. Firstly, the offset lines that happened to cross roads in the base model did not include the costs of a protective duct. This causes Steiner tree solutions on the base model to cross roads more often due to these incorrect costs. The dissolving of all roads removes these incorrectly priced offset lines.
Secondly, while it was previously possible to cross roads at intersections, the removal of the offset lines between roads has also removed many crossings near road intersections. An example is shown in figure 4.4, where the quality of the solution is decreased because of these missing offset lines between roads.
Experimental evaluation shows that the first reason only accounts for 0.1 to 0.2 percent point,

indicating that missing crossings lead to the majority of the increase in cost.



(a) Graph using base model

(b) Steiner tree solution for base model

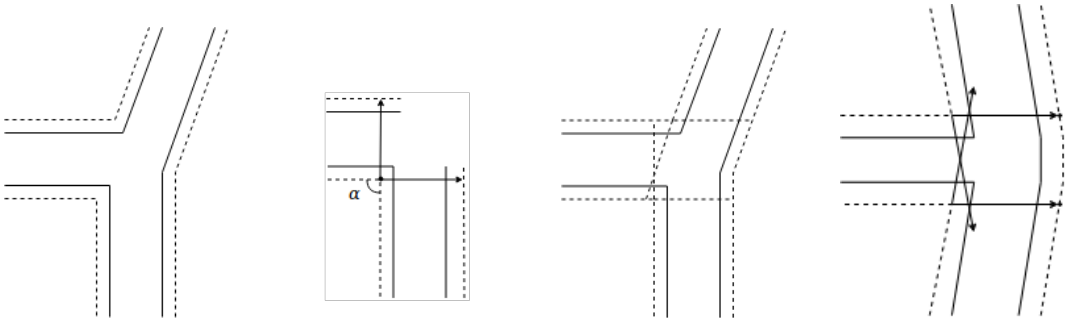
(c) Graph after dissolving roads

(d) Solution after dissolving roads

Figure 4.4: Dissolving roads leads to increased cost near road intersections

4.2.2 Road Intersections

We aim to recreate the crossings at intersections that were removed when dissolving the roads. It is often the case that the offset line given by the dissolved roads takes a sharp turn at road intersections. Our approach involves detecting these cases, and creating a crossing line when it occurs. Given an offset line, we detect where the angle between the current segment and the next segment exceeds α. Both line segments are then extended until they touch another segment of the offset lines, as long as the resulting line has distance $\leq d$. The process is visualized in figure 4.5. There are many intersections where the offset line smoothly changes its heading. We set $\alpha = 22.5$, such that any significant change in heading will allow us to check whether extending the line is viable. We then only create extensions for the original line segments before and after the turns when the length of that segment exceeds a minimum length, such that not every small notch on the boundary leads to additional edges. The maximum length of the crossing $d = 15m$, based on the expected maximum width of a road. The width of a municipal road with cycle lanes is at least $6m$. The addition of parking spaces and footpaths often bring the total width to $10m$, and may be wider near intersections to provide smooth turns. Setting the distance to $15m$ allows most roads to be crossed at a slight angle.
The addition of these crossings produces solutions with costs between 101.1% and 100.5% of the base model. The graph size is still significantly smaller at 24% to 30% of the graph produced by the base model and still allows for quick computation of a near optimal Steiner tree.



(a) Offset lines

(b) Extending lines at sharp turns

(c) Result of extending lines

(d) Missing crossing due to slanted line segments

Figure 4.5: Example of extending lines to recreate crossings.

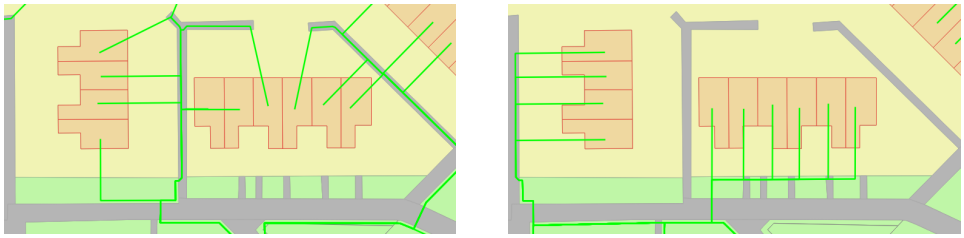
We note there are still be crossings missing, for instance when the straight line given by

extending a segment does not intersect an offset line within the required distance. In figure 4.5(d), the straight extension of some offset lines will not create a crossing for the road coming from the left, due to the slanted offset lines.
It is likely that the approach can be further improved to deal with these cases by allowing small deviations from the heading given by the line segment.

Improving Customer connections

While our aim is to connect a customer by a line from their front door, through their property, to the road, the data given by the BGT does not easily allow for this. The address point is placed at an arbitrary location within the building, the polygons indicated as private property often contain many different customers, buildings have no notion of a front door and road segments often have no relation to addresses.
Connecting customers by a straight line from the address point to the set of offset lines works reasonably well for a majority of cases. There are however many cases where a customer's connection crosses another building or connects to an alleyway instead of the main road. Instead of connecting to the nearest road segment, we improve our model by determining street names of the road segments using additional data, then connecting customers to a nearby road segment based on the address information.

The most accurate indicator for the name of a road polygon is given by the *ruimtelabel* feature. These can be directly linked to a road polygon by finding the polygon containing it.
For the other segments we use the *RoadLink* dataset. For each road segment, we determine using an overlay which line segments from the *RoadLink* dataset it contains. The line segments within a polygon are grouped by street name. The polygon is then labeled with the address given by the set of lines with the highest total length.
The resulting labels on road polygons allow us to connect the customer access points to the road indicated by their address. We look for a road segment within $25m$ with the same name when connecting customers. If we cannot find any, we revert to our previous approach and connect to the nearest road segment.
The result is that connections to alleyways are almost completely eliminated, as shown in figure 4.6.

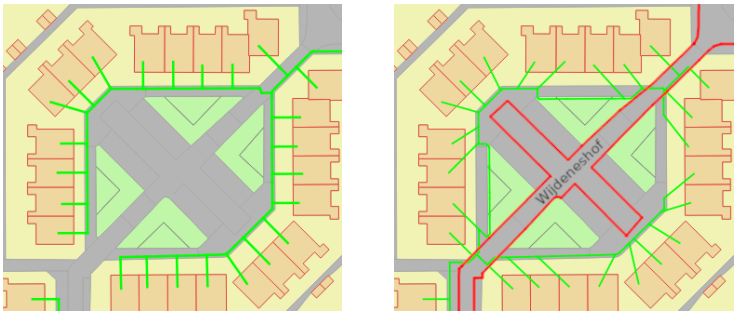


(a) (b)

Figure 4.6: Connecting customers to roads in the base model (a) compared to connecting them to roads with the same street name (b)

The maximum distance was determined by the highest distance of an address point connected to the wrong road to the correct road, which was around $23m$. A higher value is not required, and may lead to customers being connected to an incorrectly named road segment more often.
The *RoadLink* dataset does not line up directly with the BGT, so the street name is not always correct. Besides that, the line segments typically only intersect the main lane of a road. Footpaths, cycle paths, parking spaces and driveways are not able to be labeled in this manner. While connecting to a named road segment eliminates customers being connected to alleyways, it also causes many connections to have weird angles to the network when connecting to the main lane instead of a footpath, as shown in figure 4.7.

In other cases, a wrongly labeled road segment can also lead to many issues. Manual inspection of the results shows that the amount of wrongly connected customers in the altered model is much lower than in the base model. On the other hand, the wrongly named roads lead to large errors for a small amount of customers.
While the use of road names does not always lead to good results, we consider connecting to the correct road to be more important than having a straight connection.
In our test cases, the result of using the road names have varied. Because the cost through private property is high and the new connection is at least the distance of the old method, the total cost has gone up in all cases to 106.4% to 108, 1% of the base model. However, because we consider that correctly connecting customers is more important than the length of their connection, since a short path through walls is not better than a long path from a front door, we also look at the cost of the network when ignoring the customer connections. This is indicated by 'NC Cost' in table 4.3. The cost of the rest of the network is then reduced to between 96.9% and 99.8% of the base model without customer connections.



(a) Customer are connected to the nearest road segment

(b) Customers are connected to the named road segment indicated in red

Figure 4.7: In some cases the base model (a) results in better connections than the altered model (b) for customer connections

4.2.3 Green Space

Green space such as road verge, parks or other vegetated areas are not as limited in the allowed directions for passing through them in practice, since there is no added cost for crossing them.
The base model only contains the offset lines given by the polygons, whereas adding additional line segments may create additional low cost paths between terminals. The current approach is to simplify the boundary of the polygons with $\epsilon = 0.05m$ and then add the edges given by a Delaunay triangulation of the polygon to the graph. The Delaunay triangulation for a set of points is known to contain a minimum spanning tree (MST) on those points [36, Ch. 9.6]. While the triangulation and MST are not directly related to the Steiner tree, since the terminals can be connected to any point on the boundary of the polygon, the added edges still provide shorter paths through vegetation that decrease the total solution cost by 0.3% to 0.8% compared to graphs without the added paths. When not considering the cost of customer connections, the cost is decreased by 1.1% to 2.6%.

4.2.4 Cost Mapping

We use the integer weights as given by ThePeopleGroup to network engineers when creating networks. These do not give the actual costs in €per meter, but are normally used as rule of thumb when designing a network. Unfortunately, the actual costs per meter could not be obtained

at the time of writing for each of the coverage types. The cost function we use produces networks that are visually very similar to manually constructed networks.

The base weight of a polygon is determined by the *bgt_physical* attribute. When available, the *plus_physical* attribute can be used to alter the base weight. The weight is then increased based on $bgt\_function$ of that polygon. Ditches are priced based on the polygon they are in and receive additional weight if they are in the road crossings set, to account for the cost of a protective duct.

bgt_physical	weight
default	10
closed paving	10
open paving	2
Half-paved	1
unpaved	1
vegetation	1

bgt_function	weight
local road	+2
regional road	+2
rail crossing	+20

plus_physical	
ornamental paving	4
grass paver	2

ditch_type	
crossing	+1

Table 4.1: Weight given to ditches base on polygon and ditch attributes

4.3 Experimental Evaluation

In this section we will be testing our geographic conversion on 4 input sets of differing sizes. The input sets are defined by a bounding box or bounding polygon resulting in the set of polygons, polylines and points within that area from each dataset. We define the size of an input set by the total amount of line segments within the area, the size of the area in km^2 and the amount of customer access points.

We picked towns and neighborhoods of various sizes throughout The Netherlands. For Den Bosch (Empel) and Heerhugowaard, we obtained a dataset of manually constructed networks for Coax, which we will use to compare our results to. Unfortunately, the quality of this dataset is low. Polylines often have their coordinates incorrectly ordered, or entire areas are shifted. For Heerhugowaard, we picked an additional neighborhood HHW2 for which the manually constructed network was more accurate. For HHW2, we only computed the complete model.

Table 4.2: Specification of input datasets

Name	Area (km^2)	Customers	Line Segments
Gilze	2.32	2722	196002
Emmeloord	0.74	1315	63678
Empel	1.54	2239	254046
Heerhugowaard (HHW)	0.07	157	8399
HHW2	0.14	365	25029

The output is evaluated based on the size of the graph, and the cost and running time of the Steiner tree approximation on that graph as given by SCIP-Jack.

We evaluate the effect of each modification to the base model cumulatively, indicating the modifications by D for dissolve, E for extending offset lines, C for the altered customer connections and T for the triangulation of the vegetated areas.

Since an optimal Steiner tree is often not feasible, we give a time limit of 10 minutes, excluding pre-processing time, or stop once a solution with a gap $\leq 0.1\%$ has been found.

We also list the computation time for our graph and the cost of a solution when ignoring the ditch from the road to the customer access point. Due to the inefficient implementation of some

geometric operations in FME, the running times are far from an optimal implementation. We did not include the time required for retrieving the data in our graph creation time.
Since the connections from the customers to the road have a very high weight, they can skew the results considerably. The cost of these is not as relevant, since we are more concerned with the correctness of the connection and the cost of the tree in municipal areas. Therefore, we also show the cost of a solution without these connections, indicated by the 'NC Cost'. This shows the effect of our modifications more clearly on the cost of solutions on the graph.

4.3.1 Results

Table 4.3: Detailed results for geographic conversion

Name	GC [s]	$\|V\|$	$\|E\|$	% of base	STP [s]	Gap	Cost	% of base	NC Cost	% of base
Gilze	438	255208	398676	-	681,5	12,4%	250829	-	73982	-
Gil. D	132	64592	78566	19,71%	4,3	0,02%	254324	101,39%	77717	105,05%
Gil. DE	162	75567	94521	23,71%	8,7	0,02%	253668	101,13%	76950	104,01%
Gil. CDE	134	75237	93789	23,53%	4,4	0,01%	271052	108,06%	73843	99,81%
Gil. CDET	190	85133	111345	27,93%	13,34	0,01%	269208	107,33%	72998	98,67%
Emmeloord	171	105549	163357	-	331,2	0,06%	138801	-	35270	-
Emm. D	49	30275	37121	22,72%	1,5	0,02%	139799	100,72%	36269	102,83%
Emm. DE	53	37794	48360	29,60%	3,1	0,03%	139569	100,55%	36043	102,19%
Emm. CDE	50	37993	48489	29,68%	2,8	0,01%	148943	107,31%	33554	95,13%
Emm. CDET	59	42353	56752	34,74%	4,4	0,01%	148533	107,01%	33172	94,05%
Empel	552	239584	367433	-	669,5	12,7%	209432	-	62909	-
Emp. D	215	74424	91447	24,89%	8,9	0,04%	212377	101,41%	66082	105,04%
Emp. DE	238	87179	110223	30,00%	13,5	0,04%	211785	101,12%	65233	103,69%
Emp. CDE	286	88928	112694	30,67%	10,3	0,01%	222801	106,38%	60945	96,88%
Emp. CDET	288	102574	137003	37,29%	23,6	0,02%	221007	105,53%	59348	94,34%
HHW	13	10102	15192	-	6,6	0,09%	19034	-	3861	-
HHW D	2	3638	4522	29,77%	0,2	-	19223	101,00%	4050	104,91%
HHW DE	4	4586	5913	38,92%	0,2	0,02%	19165	100,69%	3992	103,40%
HHW CDE	4	4524	5778	38,03%	0,1	-	22403	117,70%	3361	87,04%
HHW CDET	11	4921	6470	42,59%	0,3	-	22315	117,24%	3279	84,94%
HHW2 CDET	20	13861	18859	-	1,5	-	60569	-	7933	-

Table 4.3 shows the results for each of the applied modifications. In terms of total Steiner tree cost, the base model always produces the best results. The downsides of the base mode are the size of the graph, which requires a much longer computation time when computing the Steiner tree, and the incorrect cost on crossings.
Dissolving is then a very important step that takes care of both these problems with the base model. As explained in section 4.2.1, the increase is caused by 2 reasons. To identify which portion of the cost difference is caused by each part, we computed which ditches in the Steiner tree crossed roads but were not labeled as a crossing for each of the base instances. This set was computed by dissolving all polygons in BGT_R labeled *local road* or *regional road*, then computing the offset at $-0.7m$. Any ditch in the Steiner tree part of the offset lines within or intersecting this area was then wrongly classified.
The length of this set for each graph using the base model is given in 4.4. Since the added cost for a crossing is given by 1, we simply add the length to the STP cost of the base solution.

The added cost of using the dissolve model is then slightly less, decreasing the cost compared to the base model by 0.1 to 0.2 percent point, or between 0.5 and 0.7 percent point for the NC Cost.
The addition of the extend modification further reduces the cost, resulting in solutions with cost at most 101% of the base model. For large input sets, the decreased running time of the dissolve and extend modifications vastly outweighs the increase in cost. For small input sets, the modifications still have significant results on the running time, but the base model can still be used for a quick solution.

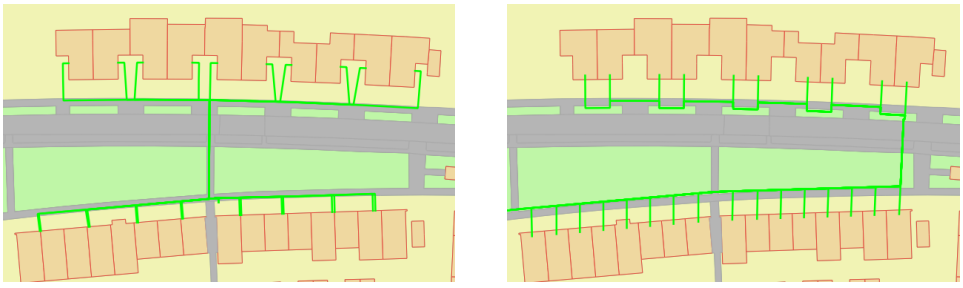
Name	Length (m)
Gilze	524
Emmeloord	182
Empel	294
HHW	9.3

Table 4.4: Missing length of crossing cost

The green space modification only adds additional options and generally gives improvements between 0.2 and 0.8 percent point of the cost. We think there is still much to gain, for example by applying more appropriate methods for decomposing polygons.

Altering the customer connections has varying results. The Emmeloord and Heerhugowaard input sets have many alleyways and require the more advanced approach to generate valid results. In Empel, some roads did not receive a street name, while others received the wrong street name, causing around 75 customers to be connected to the wrong road segment. Similar problems exist in Gilze, where around 20 customers are not connected to the correct road.
Compared to the base model, the amount of wrongly connected customers is much lower. The resulting graph represents the real world more accurately and results in much lower costs when not accounting for the connection from customer access point to the road network.
There is still a lot that can be improved when connecting customers. Determining the front of a building is the biggest issue of our geographic conversion. Some human interaction is still required to place the customer access points at the correct location within a building, and some roads require the street name to be set manually to produce correct results.

Manually constructed networks When comparing to manually constructed networks, it becomes clear how significantly our customer connections differ from a connection to the front door. A section of both networks is compared in figure 4.8. We notice that adjacent houses can sometimes share a ditch for their cables when the front doors are adjacent. Since we have no information on the location of a front door, we cannot assume adjacent houses have adjacent front doors in our model. Another significant difference in figure 4.8 is the placement of the ditch on the south side. Where our network tries to keep to the grass, the manually constructed network places the ditch as close to the buildings as possible.

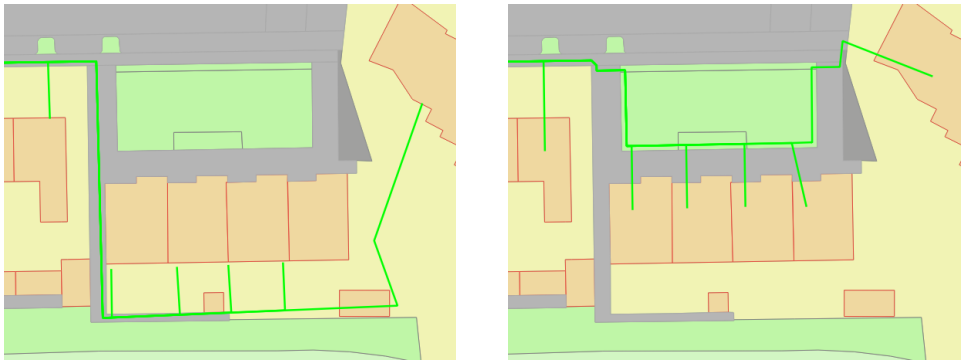


(a) Manual Network (b) Our Network

Figure 4.8: Manual network compared to our calculated network.

As shown in figure 4.9, even when a customer connection may look visually correct, the front door can be in a completely different location.

To compare our computational results to the manual dataset, we applied the same cost function to the ditches given by the manually constructed network. Because the manual network does not contain any lines within buildings, we remove any line within a building to give a fair comparison. We also compare the costs when removing all lines in private property, indicated by NC Cost. Table 4.5 shows these results.



(a) Manual Network (b) Our Network

Figure 4.9: Incorrectly assuming front of building based on road network.

	Manual			Computational		
Name	Length [m]	Cost	NC Cost	Length [m]	Cost	NC Cost
Empel	41299	206535	64000	41277	164456	59348
HHW2	6125	34665	9273	6178	31542	7933

Table 4.5: Comparison of our solution to manually constructed networks

Domain experts found that the constructed network compares well to a manual network in many areas. The major issues lie with connecting customers to the correct road and with the constructed network following the boundary of polygons too closely, leading to jagged lines. The constructed network saved a significant amount of time compared to the manual process.

4.3.2 Conclusion

Graphs constructed using our model allow for efficient Steiner trees that represent the real world well. Using the complete model, the size of our network does not exceed the size of the input polygonal subdivision, and allows for quick computation of near-optimal Steiner trees.
The model can be improved further by dealing with missing crossings as in figure 4.5(d), more accurately determining which road to connect a customer to and by using a better polygonal decomposition method for the green space.
Even though the relation of the cost function to the real world costs is weak, the Steiner tree on the network still relates well to manual solutions. The Steiner tree on the network is visually similar to manually constructed networks and has similar total length. The cost of the Steiner tree is significantly smaller than our cost function projected onto the manually constructed network. Since our cost function does not accurately represent the real-world cost, we cannot draw a strong conclusion on the effect of our algorithmic approach on the costs compared to a manual approach. We think using a more accurate cost function will lead to similar results, and lower costs than manually constructed networks.
The networks we have constructed still require many manual improvements. The biggest issue lies with connecting the customers, since the accuracy and availability of the required data is low. Another common issue is that ditches follow the perimeter of a polygon too closely producing a jagged line. Post-processing could be used to remove part of this issue.

Chapter 5

Steiner Tree

In this chapter, we take a closer look at the Steiner tree problem in graphs. We compare the performance of the reduction rules presented in [43] and the SCIP-Jack solver on graphs created using our models from chapter 4 with existing Steiner tree instances from SteinLib [44] and a dataset constructed similarly to our Steiner tree instances for Austrian towns[45].

SteinLib [44] is a collection of Steiner tree problems in graphs for many different variants and provides information about their characteristics and origins. Many of the given test sets have been solved already, though there are still some large instances for which an optimal solution has not been proved. These test sets are often randomly generated, or generated in such a way that they defy pre-processing.
The test sets we considered most similar from SteinLib were from VLSI derived problems giving grid graphs with holes, which use a rectilinear metric. We've selected several instances from these sets for which the graph has a size similar to our instances from chapter 4, which are given by the *alue*, *msm* and *taq* identifiers.

The test set constructed by Markus Prossegger [45] is much more similar to our problem. They use a very similar process to construct a graph, using a geographic dataset of polygons with coverage types as well. They also aim to find an efficient ditch network for fiber cables. We select several instances with size similar to our instances, given by the I identifier.

SCIP-Jack is a Steiner tree solving application built on the SCIP framework, which was developed for the 11th DIMACS challenge [24]. It is a generic Steiner tree solver that is able to solve various variants of the Steiner tree problem.
SCIP-Jack uses many different polynomial time pre-processing and reduction techniques described in [43] to significantly reduce the size of the input graphs. Application of these techniques can find optimal solutions to some instances already. Pre-processing reduces the amount of edges on average by 78%[1]. After pre-processing, it computes an initial solution using the shortest path heuristic. The core of their approach uses branch-and-cut procedures to compute lower bounds and prove optimality.

5.1 Experimental Evaluation

Our experiments were performed on an Intel Core i7-7700HQ CPU with 2.8GHz and 16GB RAM running Ubuntu 18.04. We used the release version of SCIP 5.0.1 with SoPlex LP solver version 3.1.1. The total run time of each instance was limited to 10 minutes. Most of our instances were solved optimally within the time limit. If an instance was not solved optimally, we give the gap between the primal bound (pb), giving the cost of the best found solution, and the dual bound (db), giving a lower bound on the cost, defined by $\frac{|pb-db|}{\max(|pb|,|db|)}$.
We also show the effect of the pre-processing steps on each dataset, along with the pre-processing time and the percentage of remaining edges, denoted by *R[%]*.

From the results in table 5.1 we immediately see the importance of the pre-processing steps

Table 5.1: Computational results for Steiner tree instances

				Pre-processing				Time			
Instance	$\|V\|$	$\|E\|$	$\|T\|$	$\|V\|$	$\|E\|$	$\|T\|$	R [%]	Pre	Sol	Tot	Gap
alue7065	34046	54841	544	28193	48733	512	89%	77,8	600,0	677,8	2,4%
alue7080	34479	55494	2344	27631	47769	2002	86%	78,3	600,0	678,3	1,6%
msm2846	3263	5783	89	1763	3097	87	54%	44,5	328,1	372,6	-
taq0377	6836	11715	136	6096	10873	136	93%	73,1	600,0	673,1	3,0%
taq0903	6163	10490	130	5723	10035	118	96%	46,4	600,0	646,4	2,0%
I006	31754	52875	2202	11750	17726	1856	34%	80,2	600,0	680,2	0,0%
I039	8755	28898	347	2466	3820	313	13%	3,6	29,4	33,0	-
I050	43073	142552	2868	14714	22446	2226	16%	55,1	600,0	655,1	0,0%
I058	23527	79256	1256	5892	9149	1007	12%	5,9	45,0	50,9	-
I065	3898	12712	144	710	1101	116	9%	1,2	1,5	2,7	-
Gilze	255208	398676	2722	94070	174770	2462	44%	91,6	600,0	691,6	12%
Gil. D	64592	78566	2722	3952	6330	861	8%	2,6	18,2	20,8	-
Gil. DE	75567	94521	2722	5539	9294	946	10%	3,6	26,2	29,8	-
Gil. CDE	75237	93789	2722	4193	7045	799	8%	3,3	12,3	15,7	-
Gil. CDET	85133	111345	2722	6733	12429	913	11%	11,9	22,1	34,0	-
Emmeloord	105549	163357	1315	31593	61299	1061	38%	115,5	600,0	715,5	0,1%
Emm. D	30275	37121	1315	1210	1891	357	5%	0,9	5,0	5,9	-
Emm. DE	37794	48360	1315	3080	5436	501	11%	2,1	12,6	14,7	-
Emm. CDE	37993	48489	1315	1112	1919	275	4%	0,5	3,9	4,4	-
Emm. CDET	42353	56752	1315	1195	2085	320	4%	0,8	4,7	5,5	-
Empel	239584	367433	2239	90376	170850	2050	46%	78,4	600,0	678,4	13%
Emp. D	74424	91447	2239	8734	13934	1053	15%	33,6	165,7	199,3	-
Emp. DE	87179	110223	2239	11623	19730	1141	18%	42,2	380,5	422,6	-
Emp. CDE	88928	112694	2239	6943	12095	998	11%	6,5	32,0	38,5	-
Emp. CDET	102574	137003	2239	13328	24582	1188	18%	15,5	136,2	151,7	-
HHW	10102	15192	157	1439	2639	93	17%	2,6	10,1	12,6	-
HHW D	3638	4522	157	0	0	0	0%	0,1	0,1	0,2	-
HHW DE	4586	5913	157	115	180	46	3%	0,1	0,2	0,4	-
HHW CDE	4524	5778	157	0	0	0	0%	0,1	0,1	0,1	-
HHW CDET	4921	6470	157	0	0	0	0%	0,1	0,1	0,2	-
HHW2 CDET	13861	18859	364	201	293	94	2%	0,2	1,4	1,6	-

on our dataset. It removed at least 50% of the edges in our instances, often reaching around 90%. Some of the smaller instances could be solved using pre-processing alone. It shows our instances differ significantly from similarly sized rectilinear grid graphs from SteinLib. The pre-processing on these instances instead removed at most 50%, and generally only around 10%.
As expected, our results are similar to the I test set. These instances are between our base model and our other models in terms of computation time and size.
Our improved models all allow for very large parts of the instances to be reduced and give rise to quick optimal solutions.
An interesting aspect to note is that our model with the altered customer connections performs better than the model without it. We think this may be thanks to the terminals being closer to each other in the graph, which allows for more alternatives to be eliminated.

The instances using the base model are often too large or complex to compute a provably efficient solution for. The cost of each solution is however very similar to the cost given by the adapted models, indicating that the reported gap is much larger than the gap to an actual optimal solution. We found that the initial solution given by SCIP-Jack using the shortest path heuristic is often already very close to an optimal solution.
SCIP-Jack performs exceptionally well on our adapted models. In large instances, we found the pre-processing time to be quite large. Since our Steiner tree results regularly require manual input to correct wrongly connected customers, fix jagged edges or add important additional crossings, we think it is often more important to give a quick near-optimal solution rather than a slow optimal solution. In our settings, we used a lower reduction setting, which results in less pre-processing techniques being applied, and we report a solution once the gap is within 0.1%. The pre-processing

time is reduced quite significantly for the larger instances, while typically still producing a near-optimal solution within 10 seconds. For smaller instances, using the full set of pre-processing techniques and waiting for an optimal solutions is still viable.

Chapter 6

Facility Location

In this chapter we discuss the facility location problem and show 3 approaches for facility location in trees. As seen in chapter 2, the predominant objective functions are *minisum* and *minimax*, giving the median and center problems respectively.

In our setting, we want to connect each customer to a DP, where we can place a DP on any vertex of the network. The cost of a solution is given by the cost of cables from each customer to its DP and the cost of placing that amount of DPs. Minimizing these costs then gives us the median problem in trees. Our tree is the Steiner tree constructed earlier giving the set of ditches that can be used to place cables in. The median problem involves placing facilities in such a way that the sum of distances to the nearest facility from each customer node is minimized along with the cost of placing the facilities. We start with a problem definition and give several properties of the location of optimally placed facilities.

Our initial approach involved the implementation of the p-median algorithm by A. Tamir [29], but this did not give valid solutions for our problem setting. We then present a greedy algorithm to minimize the amount of facilities required when facilities have a maximum allowed capacity. Next, we propose an optimal algorithm that finds the locations of capacitated facilities in time $O(D \log D \cdot n + k \cdot D \cdot n)$ for total demand D, maximum capacity k for trees with n nodes.

We show how to modify the algorithm to ensure facilities won't cover overlapping areas and options to obtain a better running time if optimality does not need to be guaranteed. Finally, we evaluate how the approaches compare in terms of running time and cost.
We define the median problem as follows:

> Median Problem in Trees
>
> **Input**: Tree $G = (V, E)$
> Weight function $w(e) \geq 0$ for all edges.
> Node demand $d(v) \in \mathbb{N}_0$ for all nodes.
> Facility cost $C > 0$ per facility.
>
> Let terminals $T = \{v \in V \mid d(v) > 0\}$ and let total demand $D = \sum_{v \in T} d(v)$
>
> **Solution**: Set of facilities S with their location $loc(s) = v \in V$.
> For each terminal $v \in T$ an assignment $f(v) = s \in S$
>
> **Minimizing**: $|S| \cdot C + \sum_{v \in T} d(v) \cdot dist(v, f(v))$

The function $dist(u, v)$ gives the cost of the shortest simple path between nodes u and v. To simplify the notation, we define $dist(u, m) = dist(u, loc(m))$ for node u and facility m, since a facility is always located on a vertex.
Trees with $D = 0$ are solved trivially by not placing any medians.

Recall that in our fiber network a DP can facilitate at most 48 customers.
The above definition can lead to solutions where some facilities service a high total demand,

whereas others service low demand. Even though the amount of facilities provides enough capacity in total, the allocation of terminals to facilities can give an incorrect solution where facilities have too much demand assigned to it.
For this reason we need an additional constraint limiting the amount of demand assigned to a facility to the maximum capacity k.
We call this the Capacitated Median Problem in Trees.

Capacitated Median Problem in Trees

Input: Tree $G = (V, E)$.
Weight function $w(e) \geq 0$ for all edges.
Node demand $d(v) \in \mathbb{N}_0$ for all nodes.
Maximum facility capacity $k \in \mathbb{N}^+$.
Facility cost $C > 0$ per facility.

Let terminals $T = \{v \in V \mid d(v) > 0\}$ and let total demand $D = \sum_{v \in T} d(v)$

Solution: Set of facilities S with their location $loc(s) = v \in V$.
For each node $v \in T$ and $s \in S$ an assignment $f(v, s) \in \mathbb{N}_0$ assigning that amount of demand from v to s, such that $\forall v \in T : \sum_{s \in S} f(v, s) = d(v)$ and $\forall s \in S : \sum_{v \in T} f(v, s) \leq k$.

Minimizing: $|S| \cdot C + \sum_{v \in T} \sum_{s \in S} f(v, s) \cdot dist(v, s)$

We need to take extra care for terminals v with $d(v) > 1$. It may be the case that 2 facilities are both almost full and can't fit the entirety of the demand of v, so the assignment function can assign the demand of a terminal to multiple facilities. Note that we allow solutions in which multiple facilities are located at the same vertex.

The definition above requires facilities to be placed at vertices. This restriction is without loss of generality: If a facility is placed somewhere along an edge, then moving the facility to the vertex in the direction of the majority of the clients assigned to it does not increase cost.

Lemma 6.1. *An optimal facility placement exists where facilities are only placed on vertices with positive demand or degree* ≥ 3.

Proof. Consider an instance (G, w, d, k, C) of the capacitated facility location problem in trees. Let S be a set of solution facilities with assignment function f. Suppose there exists a facility m where $v = loc(m)$ is a vertex with demand 0 and degree at most 2. We show how to obtain a new optimal solution in which each facility is placed on a different vertex.

Case $degree(v) = 0$, then $|V| = 1$ since we have a tree. Since v is a vertex with 0 demand, it follows that each vertex of G has demand 0. Hence there is a valid solution of cost 0 that places no facilities, contradicting the assumption there exists a facility m.

Case $degree(v) = 1$.
Let v_p be the parent of v.
Then the cost OPT of solution (S, f) is given by:

$$\begin{aligned} OPT &= |S| \cdot C + \sum_{u \in T} \sum_{s \in S} f(u, s) \cdot dist(u, s) \\ &= |S| \cdot C + \sum_{u \in T} \sum_{s \in S \setminus \{m\}} f(u, s) \cdot dist(u, s) + \sum_{v \in T} f(u, m) \cdot dist(u, m) \\ &= |S| \cdot C + \sum_{u \in T} \sum_{s \in S \setminus \{m\}} f(u, s) \cdot dist(u, s) + \sum_{u \in T} f(u, m) \cdot dist(u, v) \end{aligned} \tag{6.1}$$

We alter the solution (S, f) by setting $loc(m) = v_p$ to obtain a solution (S', f') with cost OPT'. The cost of OPT' of solution (S, f) is then given by:

$$OPT' = |S| \cdot C + \sum_{u \in T} \sum_{s \in S \setminus \{m\}} f(u, s) \cdot dist(u, s) + \sum_{u \in T} f(u, m) \cdot dist(u, v_p) \tag{6.2}$$

Given that $d(v) = 0$ and $degree(v) = 1$, we have that $\forall_{v \in T} : dist(u, v) \geq dist(u, v_p)$.
Thus, $OPT' \leq OPT$

Case $degree(v) = 2$. Let v_l, v_r be the neighbors of v. Let G_l be the tree in the forest $G - \{v\}$ that contains v_l and let G_r be the tree in the forest that contains v_r. Let T_l, T_r be the terminals in G_l, G_r respectively. Let $d_l = \sum_{u \in T_l} f(u, m)$ and $d_r = \sum_{u \in T_r} f(u, m)$.

Then,

$$\begin{aligned} OPT &= |S| \cdot C + \sum_{u \in T} \sum_{s \in S \setminus \{m\}} f(u, s) \cdot dist(u, s) \\ &\quad + \sum_{u \in T_l} \big(f(u, m) \cdot (dist(u, v_l) + w(v_l, v))\big) + \sum_{u \in T_r} \big(f(u, m) \cdot (dist(u, v_r) + w(v_r, v))\big) \\ &= |S| \cdot C + \sum_{u \in T} \sum_{s \in S \setminus \{m\}} f(u, s) \cdot dist(u, s) \\ &\quad + \sum_{u \in T_l} \big(f(u, m) \cdot dist(u, v_l)\big) + \sum_{u \in T_r} \big(f(u, m) \cdot (dist(u, v_r)\big) + d_l \cdot w(v_l, v) + d_r \cdot w(v_r, v) \end{aligned} \tag{6.3}$$

Suppose $d_l \geq d_r$, then placing m on v_l instead of v gives solution (S', f') with cost OPT':

$$\begin{aligned} OPT' &= |S| \cdot C + \sum_{u \in T} \sum_{s \in S \setminus \{m\}} f(u, s) \cdot dist(u, s) \\ &\quad + \sum_{u \in T_l} \big(f(u, m) \cdot dist(u, v_l)\big) + \sum_{u \in T_r} \big(f(u, m) \cdot (dist(u, v_r)\big) + d_r \cdot (w(v_r, v) + w(v_l, v)) \\ &\quad \leq OPT \end{aligned} \tag{6.4}$$

The case where $d_l \leq d_r$ follows from symmetry.
Then, whenever a facility is placed on a vertex v with degree 2 and demand 0, there is an alternative solution that is at least as good where that facility is placed on another vertex. Along a path of degree-2 demand-0 vertices, we can repeat this process to move the facility in a direction that either had degree ≥ 3 or a positive demand, while maintaining optimality. This process can be applied independently for each facility. □

Corollary: Any node v with $d(v) = 0$ and $deg(v) = 1$ can be removed without increasing the cost of the optimal solution.
Any node v with $d(v) = 0$, $deg(v) = 2$ and neighbors v_l, v_r can be contracted by removing v and adding edge (v_l, v_r) between its neighbours with $w(v_l, v_r) = w(v_l, v) + w(v, v_r)$ without increasing the cost of an optimal solution.
These reduction rules reduce the size of the problem instance to a smaller equivalent problem. We can exhaustively apply these rules to remove all non-terminal nodes with degree ≤ 2.

We define 2 paths as crossing paths, when one path contains an edge (u, v) and the other contains an edge (v, u) for any $u, v \in V$.

Lemma 6.2. *There is an optimal solution (S, f) such that for any pair of facilities $m_1, m_2 \in S$ and terminals $u_1, u_2 \in T$ with $f(u_1, m_1) > 0$ and $f(u_2, m_2) > 0$, the path from u_1 to m_1 does not cross the path from u_2 to m_2.*

Proof. Suppose there is an optimal solution (S, f) with cost OPT with terminals u_1, u_2 assigned to facilities m_1, m_2 respectively, where the simple path from u_1 to m_1 contains edge (v, w) and the simple path from u_2 to m_2 contains edge (w, v). Let $d_1 = f(u_1, m_1)$, $d_2 = f(u_2, m_2)$ and

$d_3 = f(u_1, m_2)$, $d_4 = f(u_2, m_1)$.
The cost of this solution is given by:

$$\begin{aligned} OPT =& |S| \cdot C + \sum_{u \in T} \sum_{s \in S} f(u,s) \cdot dist(u,s) \\ =& |S| \cdot C + \sum_{u \in T} \sum_{\substack{s \in S \\ \wedge \neg(u=u_1 \wedge (s=m_1 \vee s=m_2)) \\ \wedge \neg(u=u_2 \wedge (s=m_1 \vee s=m_2))}} f(u,s) \cdot dist(u,s) \\ &+ d_1 \cdot dist(u_1, m_1) + d_2 \cdot dist(u_2, m_2) \\ &+ d_3 \cdot dist(u_1, m_2) + d_4 \cdot dist(u_2, m_1) \end{aligned} \tag{6.5}$$

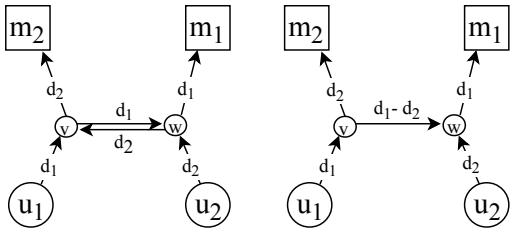
We split the cost of the solution into two parts:
$BASE = |S| \cdot C + \sum_{u \in T} \sum_{\substack{s \in S \\ \wedge \neg(u=u_1 \wedge (s=m_1 \vee s=m_2)) \\ \wedge \neg(u=u_2 \wedge (s=m_1 \vee s=m_2))}} f(u,s) \cdot dist(u,s)$
$ASG = d_1 \cdot dist(u_1, m_1) + d_2 \cdot dist(u_2, m_2) + d_3 \cdot dist(u_1, m_2) + d_4 \cdot dist(u_2, m_1)$
Then $OPT = BASE + ASG$.

$$\begin{aligned} ASG =& d_1 \cdot dist(u_1, m_1) + d_2 \cdot dist(u_2, m_2) \\ &+ d_3 \cdot dist(u_1, m_2) + d_4 \cdot dist(u_2, m_1) \\ =& d_1 \cdot (dist(u_1, v) + w(v,w) + dist(w, m_1)) + d_2 \cdot (dist(u_2, w) + w(w,v) + dist(v, m_2)) \\ &+ d_3 \cdot dist(u_1, m_2) + d_4 \cdot dist(u_2, m_1) \\ =& d_1 \cdot (dist(u_1, v) + dist(w, m_1)) + d_2 \cdot (dist(u_2, w) + dist(v, m_2)) + w(v,w) \cdot (d_1 + d_2) \\ &+ d_3 \cdot dist(u_1, m_2) + d_4 \cdot dist(u_2, m_1) \end{aligned} \tag{6.6}$$

Suppose $d_1 \geq d_2$. We can construct an alternative solution (S, f') by altering f. We set $f'(u_1, m_1) = d_1 - d_2$, $f'(u_1, m_2) = d_2 + d_3$, $f'(u_2, m_1) = d_2 + d_4$ and $f'(u_2, m_2) = 0$, which does not alter the total demand assigned to each facility.
The cost of this assignment is given by $OPT' = BASE' + ASG'$. Since the assignment is only changed for terminals u_1, u_2 to facilities m_1, m_2, $BASE' = BASE$.
Our new assignment cost is then given by:

$$\begin{aligned} ASG' =& (d_1 - d_2) \cdot dist(u_1, m_1) + d_2 \cdot dist(u_2, m_1) + d_2 \cdot dist(u_1, m_2) \\ &+ d_3 \cdot dist(u_1, m_2) + d_4 \cdot dist(u_2, m_1) \\ \leq& (d_1 - d_2) \cdot (dist(u_1, v) + w(v,w) + dist(w, m_1)) \\ &+ d_2 \cdot (dist(u_2, w) + dist(w, m_1)) + d_2 \cdot (dist(u_1, v) + dist(v, m_2)) \\ &+ d_3 \cdot dist(u_1, m_2) + d_4 \cdot dist(u_2, m_1) \\ =& d_1 \cdot (dist(u_1, v) + dist(w, m_1)) + d_2 \cdot (dist(u_2, w) + dist(v, m_2)) + w(v,w) \cdot (d_1 - d_2) \\ &+ d_3 \cdot dist(u_1, m_2) + d_4 \cdot dist(u_2, m_1) \\ \leq& ASG \end{aligned} \tag{6.7}$$

Then, $OPT' = BASE + ASG' \leq BASE + ASG$.
The case $d_2 \geq d_1$ follows from symmetry.
An example with $d_3, d_4 = 0$ is shown in figure 6.1.
We have shown that any optimal solution (S, f) with cost OPT where the paths from terminals to facilities cross can be modified to an alternative valid solution without that crossing path with cost $OPT' \leq OPT$. We can exhaustively apply the modification until no such crossing paths exist. Therefore, an optimal cost solution must exist without any crossing paths. □



(a) Original Allocation. (b) Altered Allocation.

Figure 6.1: Reassigning terminals to remove crossing paths.

Another property of the median problem is that the demand assigned to a facility over a single edge does not exceed the other demands assigned to that facility, which we state as follows:

Lemma 6.3. *Let (G, w, d, k, C) be an input to the Capacitated Median problem on trees. Let $v \in V(G)$ with facility m placed on v, let u be a neighbor of v in G, and let G_u be the tree in the forest $G-\{v\}$ that contains u. Then there is an optimal solution (S, f) such that $\sum_{w \in G_u} f(w, m) \leq \sum_{w \in V \setminus G_u} f(w, m)$.*

Proof. Let (S, f) be an optimal solution to the Capacitated Median Problem on trees for input (G, w, d, k, c), where $u, v \in V(G)$ with $m \in S$ and $loc(m) = v$ and u is a neighbor of v, and let G_u be the tree in the forest $G - \{v\}$ that contains v, such that $d_1 = \sum_{w \in G_u} f(w, m) > \sum_{w \in V \setminus G_u} f(w, m) = d_2$.
The cost OPT of (S, f) is as follows:

$$\begin{aligned} OPT =& |S| \cdot C + \sum_{u \in T} \sum_{s \in S \setminus \{m\}} f(u, s) \cdot dist(u, s) \\ &+ \sum_{w \in G_u} f(w, m) \cdot dist(w, u) + d_1 \cdot w(u, v) \\ &+ \sum_{w \in V \setminus G_u} f(w, m) \cdot dist(w, v) \end{aligned} \tag{6.8}$$

We can construct an alternative solution by setting $loc(m) = u$ to obtain solution (S', f') with cost OPT'.

$$\begin{aligned} OPT' =& |S| \cdot C + \sum_{u \in T} \sum_{s \in S \setminus \{m\}} f(u, s) \cdot dist(u, s) \\ &+ \sum_{w \in G_u} f(w, m) \cdot dist(w, u) \\ &+ \sum_{w \in V \setminus G_u} f(w, m) \cdot dist(w, v) + d_2 \cdot w(v, u) \\ \leq& OPT \end{aligned} \tag{6.9}$$

So there always exists an optimal cost solution where the demand of the subtree G_u for a neighbor u of v assigned to the facility placed on v does not exceed the other demand assigned to the facility on v. □

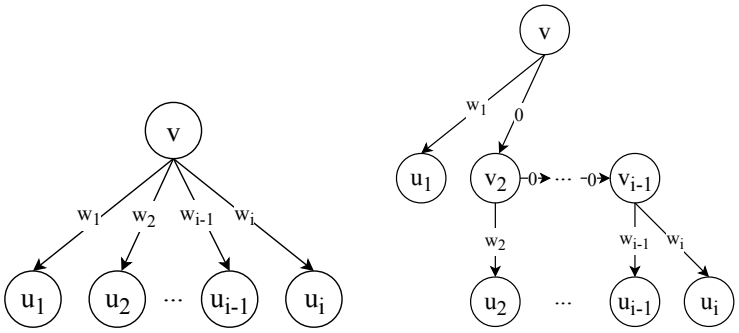
6.1 Transformations and reductions

The p-median algorithm and capacitated median algorithm both use a transformation from a tree into an equivalent rooted binary tree to simplify the algorithm. We use the transformation as

shown in the p-median paper by A. Tamir [29] that works as follows:
We first pick an arbitrary node as root of the tree.

Definition 6.4. *We define a subtree V_v as the subtree of G rooted at node v that includes all descendants of v. A node u is a descendant of v if v lies on the unique path from u to the root of G.*
A node u is a child of node v if u is a descendant of v with an edge (u, v)

Then, for each node v with children $U = \{u_1 \dots u_i\}$ where $|U| \geq 3$, we create an additional set of nodes $U' = \{v_2 \cdots v_{i-1}\}$. Then $|U'| = |U| - 2$. For each $2 \leq j \leq i-2$ we change (v, u_j) to (v_j, u_j) and add an edge (v_j, v_{j+1}) with weight 0.
Finally, we change (v, u_{i-1}) to (v_{i-1}, u_{i-1}) and (v, u_i) to (v_{i-1}, u_i) and add the edge (v, v_2) with weight 0.



(a) Original tree. (b) Equivalent binary tree.

Figure 6.2: Transformation of nodes with 3 or more children

For nodes v with 1 child, we add a new node v_1 and an edge (v, v_1) with weight 0. Any new node is given a demand of 0. The amount of nodes in the transformed tree is at most $2n - 3$ [29]. Since the weight of the edge is 0, placing a median on a new node is equivalent to placing it on its parent.

The capacitated median algorithm uses binary trees in which only the leaves can be terminals.

For a general tree, we can transform it in the following way such that all terminals are leaves: We add a child u to internal nodes v with $d(v) > 0$. Set $d(u) = d(v)$ and then $d(v) = 0$. The added edge then has weight $w(u, v) = 0$. For a tree with n nodes where every internal node has a positive demand, the transformed tree has at most $2n - 2$ nodes, since at least 2 nodes are leaves in any tree.

After this, we transform this new tree to a binary tree as before, with the root being an arbitrary internal node, and obtain a rooted binary tree where only leaves are terminals.
The amount of nodes is then easily bounded by $2(2n - 2) - 3 = 4n - 7$, which remains a linear increase.

Note that for our application in Fiber Networks, the graph we created in Chapter 3 will usually not have internal nodes with demand. In the same vein, the amount of internal nodes with degree greater than 3 is small within Steiner trees. Along with the application of the reductions following from Lemma 6.1, the resulting amount of nodes will be $2 \cdot |T| - 1$, given that it is a binary tree with $|T|$ leaves.

Definition 6.5. *The children of a node v in a binary tree G are given by v_l, v_r.*

6.2 p-median Algorithm

The p-median problem on trees is a variation on the Median Problem in Trees in which at most p facilities can be placed. A 'leaves to root' dynamic programming algorithm for the problem was

first presented by Kariv and Hakimi [28] with a running time of $O(p^2 \cdot n^2)$. The algorithm was then improved by Tamir [29] to obtain a running time $O(p \cdot n^2)$.
We implemented this algorithm to gauge the performance on our datasets and to determine whether it could be adapted in such a way that the capacity constraints always hold.
Implementing the algorithm was challenging and required some work to obtain correct and optimal solutions for the p-median problem. We found that the algorithm could not be easily adapted to satisfy the capacity constraints.

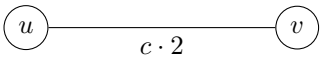
6.3 Greedy capacitated facility location

We show a simple greedy $O(n)$ algorithm for the capacitated median problem in trees intended to provide a quick valid solution.
During pre-processing, the demand of terminals v with $d(v) \geq k$ is set to $d(v)\%k$ and $\lfloor d(v)/k \rfloor$ medians are added with location v. We then have for any node v that $d(v) \in \{0 \dots k-1\}$.

The algorithm uses a tree rooted at an arbitrary node and computes a function $g(v)$ for each node v starting from the leaves.
For leaves, $g(v) = d(v)$.
For nodes v, it depends only on $d(v)$ and the children of v, given by $N(v)$.
If $d(v) + \sum_{u \in N(v)} d(u) \leq k$, we set $g(v)$ to that value. Otherwise, we repeatedly remove an edge from v to one of its children until $d(v) + \sum_{u \in N(v)} d(u) \leq k$ and set $g(v)$ to that value.
This procedure produces a forest in which each connected component has a total demand of at most k.
For each connected component we then find its 1-median. Finding the 1-median is easily done in linear time in the number of nodes, for example using the algorithm described by Goldman[46].
Each terminal in a connected component is then assigned to the median point of that component.

Lemma 6.6. *The greedy algorithm is not a c-approximation for the capacitated median problem for any factor c.*

Proof. We can easily prove this by counterexample for arbitrary approximation factor $c \geq 1$. Let $k = 2$ and $C = 1$. G has 2 nodes u, v with $d(u) = d(v) = 1$ and a edge (u, v) with $w(u, v) = c \cdot 2$.



A possible valid solution places a facility on both u and v to always obtain cost 2. The greedy algorithm will however always include (u, v) because $d(u) + d(v) \leq k$, giving a solution with cost $c \cdot 2 + 1$. □

6.4 An optimal algorithm for capacitated facility location

We now show a dynamic programming algorithm that solves the Capacitated Median Problem in Trees to optimality with running time $O(D \log D \cdot n + k \cdot D \cdot n)$ for a tree with n nodes, maximum capacity k and total demand $D = \sum_{v \in T} d(v)$.

To simplify the algorithm we assume without loss of generality that all terminals in G are leaves and that G is a rooted binary tree where internal nodes have exactly 2 children. The transformation from any tree to the required type is shown in Section 6.1.

We will use subproblems indexed by (v, p), where v is a node and p a supply/demand value with $-D \leq p \leq D$ for the subtree V_v.

The value of p indicates the amount of leftover supply within a subtree V_v that is available from node v. A negative value of p indicates that a total of $|p|$ demand inside V_v has not been satisfied and must be supplied to v.
A large positive value for p then means that the facilities in V_v send a lot of their excess capacity to the root, such that the root can act as facility with p available capacity.
On the other hand, a large negative value of p indicates that many terminals aren't assigned to a facility in V_v. Instead, they are assigned to the root v. In that case, v acts as a terminal with demand $|p|$.

We define a subproblem (v, p) as follows:

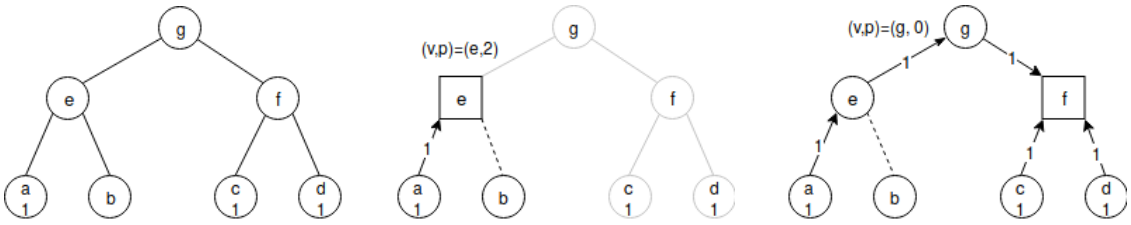
Input: Binary Tree $G = (V, E)$.
Root node $v \in V$.
Supply/demand value $p \in \{-D, \dots, D\}$
Weight function $w(e) \geq 0$ for all edges.
Node demand $d(u) \in \mathbb{N}_0$ for all nodes.
Maximum facility capacity $k \in \mathbb{N}^+$.
Facility cost $C > 0$ per facility.

Let $T_v = \{v \in T \cap V_v\}$

Output: Set of facilities S with their location $loc(s) \in V_v$.
An assignment function $f : (T_v \cup \{v\}) \times (S \cup \{v\}) \to \mathbb{N}_0$ such that $\forall_{u \in T_v} : \sum_{s \in S \cup \{v\}} f(u, s) = d(u)$ and $\forall_{s \in S} : \sum_{u \in T_v \cup \{v\}} f(u, s) \leq k$.
Where $p = \sum_{s \in S} f(v, s) - \sum_{u \in T_v} f(u, v)$.

If $p = 0$, we have $\sum_{u \in T_v} f(u, v) = \sum_{s \in S} f(v, s)$: any extra demand assigned to v is canceled out by the extra supply assigned to v. We can reassign this extra demand to the medians providing additional supply.
We define the cost of a solution as follows: $|S| \cdot C + \sum_{u \in T_v \cup \{s\}} \sum_{s \in S \cup \{v\}} f(u, s) \cdot dist(u, s)$.

It follows that a minimum cost solution to the subproblem given by $(root, 0)$ is a solution for the Capacitated Median Problem in Trees.



(a) Input (b) Solution for subproblem (e, 2) (c) Solution for subproblem (g, 0)

Figure 6.3: Example input with unit edge weights, $k = 3$, $C = 3$ and demand 1 on vertices a, c, d. Facilities are shown as squares, and assignments as an arrow to the facility with the allocated demand.

The example in figure 6.3 gives an input and the solutions for 2 different subproblems. The cost of the solution to $(e, 2)$ is given by: $1 \cdot C + 1 \cdot dist(a, e) = 4$. Here, the facility placed on e can facilitate another 2 demand.
Figure 6.3c shows a solution for the root of the tree with $p = 0$ with cost $C + dist(a, f) + dist(c, f) + dist(d, f) = 8$. In 6.3c we also see that we only show the amount of demand or supply

that passes through each edge. We later show how to construct an assignment function based on this information. However, this requires that there is not a positive supply and a positive demand passing through any edge, which we state as follows:

Lemma 6.7. *There exists a minimum cost solution (S, f) with cost OPT to the subproblem (v, p) where either $p = \sum_{s \in S} f(v, s)$ or $p = -\sum_{u \in T_v} f(u, v)$.*

Proof. Suppose there exists a solution with $\sum_{s \in S} f(v, s) > 0$ and $\sum_{u \in T_v} f(u, v) > 0$.
Then there is some terminal u with $d_1 = f(u, v) > 0$ and some facility m with $d_2 = f(v, m) > 0$ and $d_3 = f(u, m) \geq 0$.
We split OPT into a base and assignment cost again, where the base cost contains all costs except the cost of the assignments listed above.

$$\begin{aligned} OPT =& |S| \cdot C + \sum_{w \in T} \sum_{s \in S} f(w, s) \cdot dist(w, s)[(w, s) \notin \{(u, v), (v, m), (u, m)\}] \\ &+ dist(u, v) \cdot d_1 + dist(v, m) \cdot d_2 + dist(u, m) \cdot d_3 \\ =& BASE + dist(u, v) \cdot d_1 + dist(v, m) \cdot d_2 + dist(u, m) \cdot d_3 \end{aligned} \tag{6.10}$$

If $d_1 \geq d_2$, we can construct an alternative solution (S, f') and set $f'(u, v) = d_1 - d_2$, $f'(u, m) = d_2 + d_3$ and set $f'(v, m) = 0$ without breaking capacity constraints to obtain an assignment with cost:

$$\begin{aligned} OPT^* &= BASE + dist(u, v) \cdot (d_1 - d_2) + dist(u, m) \cdot (d_2 + d_3) \\ &\leq OPT \end{aligned} \tag{6.11}$$

Since $dist(u, m) \leq dist(u, v) + dist(v, m)$.
If $d_2 \geq d_1$, we can similarly reassign all demand of u to use a part of the supply of m.

This reallocation produces a new solution whose cost is not larger, but in which $|\sum_{s \in S} f(v, s)| + |\sum_{u \in T_v} f(u, v)|$ has decreased. Since such an allocation can be done as long as $\sum_{s \in S} f(v, s) > 0$ and $\sum_{t \in T_v} f(u, v) > 0$, it follows that after a finite amount of reallocations we obtain an optimal solution that has the desired form.

Then, for any solution with $\sum_{s \in S} f(v, s) > 0$ and $\sum_{u \in T_v} f(u, v) > 0$ there is an equivalent or cheaper solution with $\sum_{s \in S} f(v, s) = 0$ or $\sum_{u \in T_v} f(u, v) = 0$. □

We define $F(v, p)$ as the cost of a minimum cost solution to the subproblem (v, p). Next, we present a recursive formula based on the subproblems given by the children of v and prove that it is equal to $F(v, p)$. For a leaf, the cost of a solution for a value p is given by the minimum amount of facilities required to obtain that value of p. For internal nodes, we distinguish 2 cases: Either we allocate previously unallocated demands or supplies from child subtrees to the current node, or we use a previous solution with a lower value for p at the current node and add a facility to obtain p.

Theorem 6.8.
For a leaf v:

$$F(v, p) = \begin{cases} \lceil (p + d(v))/k \rceil \cdot C & \text{if } p + d(v) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{6.12}$$

For an internal node v with children v_l, v_r

$$F(v, p) = \min \begin{cases} \min_{q_1 + q_2 = p} \big(F(v_l, q_1) + F(v_r, q_2) + |q_1| \cdot w(v_l, v) + |q_2| \cdot w(v_r, v) \big) \\ \min\limits_{\substack{0 < q \leq k \\ p - q \geq -D}} \big(F(v, p - q) + C \big) \end{cases} \tag{6.13}$$

Proof.

Leaf v: If v is a leaf, then V_v consists of only v.
If (S, f) is a solution to subproblem (v, p), then we can re-write the cost of a solution as follows:

$$\begin{aligned} &|S| \cdot C + \sum_{u \in T_v \cup \{s\}} \sum_{s \in S \cup \{v\}} f(u,s) \cdot dist(u,s) \\ =&|S| \cdot C + \sum_{s \in S \cup \{v\}} f(v,s) \cdot dist(v,s) \\ =&|S| \cdot C + \sum_{s \in S \cup \{v\}} f(v,s) \cdot 0 \\ =&|S| \cdot C \end{aligned} \tag{6.14}$$

Hence, the cost of the solution reduces to the number of facilities it places. An optimal solution places the minimum number of facilities needed for a given value of p.
If $p \leq -d(v)$, then we can construct a solution by assigning $f(v, v) = d(v)$. Setting $S = \emptyset$ does not violate the capacity constraints, giving a cost of 0 which is the minimum of any solution.
If $p > -d(v)$, we need to provide for a total capacity of at least $p + d(v)$ demand. The minimum amount of facilities required is given by $\lceil (p + d(v))/k \rceil$.
A solution with that amount of facilities costs $\lceil (p + d(v))/k \rceil \cdot C$.

Node v: Here we will prove that $F(v, p) =$ our minimization function $\min(\dots)$ by proving $F(v, p) \leq \min(\dots)$ and $F(v, p) \geq \min(\dots)$

F(v, p) $\leq$ min(. . .) For all $q_1 \in [\max(-D, -D + p), \min(D, D + p)] \cap \mathbb{Z}$ and $q_2 = -q_1 + p$ there exist optimal solutions with cost $F(v_l, q_1), F(v_r, q_2)$.
We have $\max(-D, -D + p) \leq q_1 \leq \min(D, D + p)$ such that q_2 is always within $-D$ and D, since $-D \leq p \leq D$.
The values $|q_1|, |q_2|$ indicate there is a demand of a terminal or leftover supply of that amount assigned to v_l, v_r respectively in the optimal solution for that subproblem.
Let (S_l, f_l) and (S_r, f_r) be optimal solutions of the subproblems (v_l, q_1) and (v_r, q_2) respectively.

We can construct a set of facilities for $F(v, p)$ from the union of S_l and S_r. Since $V_l \subset V_v$ and $V_r \subset V_v$, each facility is inside V_v.
We construct the assignment function as follows:

We take over any assignment for which v_l or v_r is not the source or destination.
For any terminal $u \in T_l$ in (v_l, q_1) with $d_u = f_l(u, v_l) > 0$, we set $f(u, v) = d_u$ and $f(u, v_l) = 0$ for the solution (S, f).
For any facility m with $d_m = f_l(v_l, m) > 0$, we set $f(v, m) = d_m$ and $f(v_l, m) = 0$ for the solution (S, f).
We move the supplies and demands of the solution (S_r, f_r) from v_r to v similarly.

The cost of this solution is given by:

$$
\begin{aligned}
F(v,p) \leq & F(v_l,q_1) + F(v_r,q_2) \\
& - \big(\sum_{s\in S_l} f_l(v_l,s)\cdot dist(v_l,s)\big) - \big(\sum_{u\in T_l} f_l(u,v_l)\cdot dist(u,v_l)\big) \\
& + \big(\sum_{s\in S_l} f(v,s)\cdot dist(v,s)\big) + \big(\sum_{u\in T_l} f(u,v)\cdot dist(u,v)\big) \\
& - \big(\sum_{s\in S_r} f_r(v_r,s)\cdot dist(v_r,s)\big) - \big(\sum_{u\in T_r} f_r(u,v_r)\cdot dist(u,v_r)\big) \\
& + \big(\sum_{s\in S_r} f(v,s)\cdot dist(v,s)\big) + \big(\sum_{u\in T_r} f(u,v)\cdot dist(u,v)\big) \\
= & F(v_l,q_1) + F(v_r,q_2) \\
& + \big(\sum_{s\in S_l} f(v,s)\cdot w(v_l,v)\big) + \big(\sum_{u\in T_l} f(u,v)\cdot w(v_l,v)\big) \\
& + \big(\sum_{s\in S_r} f(v,s)\cdot w(v_r,v)\big) + \big(\sum_{u\in T_r} f(u,v)\cdot w(v_r,v)\big) \\
= & F(v_l,q_1) + F(v_r,q_2) + |q_1|\cdot w(v_l,v) + |q_2|\cdot w(v_r,v)
\end{aligned}
\tag{6.15}
$$

In the last step of the equality, we assume the solutions (S_l, f_l) and (S_r, f_r) have $\big(\sum_{s\in S_l} f(v,s)\cdot w(v_l,v)\big) + \big(\sum_{u\in T_l} f(u,v)\cdot w(v_l,v) = |q_1|$ and similarly for the right subtree, which holds because there exist optimal cost solutions to (v_l, q_1), (v_r, q_2) with this property (Lemma 6.7).
Finally, we will balance the solution by canceling out supply and demand assigned to the root, similarly to Lemma 6.7.
Let $p_1 = \sum_{u\in T_v} f(u,v)$ and $p_2 = \sum_{s\in S} f(v,s)$.

Suppose $p_1, p_2 > 0$, then there is some facility m with $d_m = f(v,m) > 0$ and some terminal u with $d_u = f(u,v) > 0$. By Lemma 6.7 we have that either $m \in V_l \wedge u \in V_r$, or $m \in V_r \wedge u \in V_l$.
Then, because V_v is a tree, the simple path from u to m must go through v, so $dist(u,m) = dist(u,v) + dist(v,m)$.
The cost of this assignment is: $d_u \cdot dist(u,v) + d_m \cdot dist(v,m)$.
We reassign $\min(d_u, d_m)$ demand from $f(u,v)$ to $f(u,m)$ to obtain a new assignment cost:

$$
\begin{aligned}
& (d_u - min(d_u,d_m))\cdot dist(u,v) + (d_m - min(d_u,d_m))\cdot dist(v,m) + min(d_u,d_m)\cdot dist(u,m) \\
= & d_u\cdot dist(u,v) + d_m\cdot dist(v,m)
\end{aligned}
\tag{6.16}
$$

We repeat reassigning until $p_1 = 0 \vee p_2 = 0$ to obtain a valid solution conforming to Lemma 6.7.

This proves that $F(v,p)$ is at most the value described by the top line of equation 6.13.
Next, we show that $F(v,p)$ is at most the value described by the bottom line of equation 6.13.

For all $q \in \{1,\ldots,k\}$ with $-q \geq -D + p$ there exists a solution (S_q, f') to the subproblem $(v, p-q)$ with cost $F(v,p-q)$.
We can use this to create a solution for subproblem (v,p).

The set of medians is then given by $S = S_q \cup \{m\}$, where $loc(m) = v$. We assign $f(v,m) = q$.
Given that $dist(v,m) = 0$, we can balance out supply and demand as before without changing assignment costs.
As we are creating q additional supply, we need $\lceil q/k \rceil$ facilities. Given that $0 < q \leq k$, we have $\lceil q/k \rceil = 1$.
The cost of this solution is then given by $F(v,p-q) + C \geq F(v,p)$.
We then have that:

$$
F(v,p) \leq min \begin{cases} \min_{q_1+q_2=p} \big(F(v_l,q_1) + F(v_r,q_2) + |q_1|\cdot w(v_l,v) + |q_2|\cdot w(v_r,v)\big) \\ \min\limits_{\substack{0<q\leq k \\ p-q\geq -D}} \big(F(v,p-q) + C\big) \end{cases}
$$

F(v, p) ≥ min(. . .) Let OPT be a solution (S, f) for the subproblem given by (v, p) with cost $F(v, p)$.
We assume there is no allocation of demands from the left subtree to supply in the right subtree and vice versa, which can always be achieved by sending and receiving from the root. Additionally, we assume a subtree does not simultaneously send and receive from the root, following Lemma 6.7.
We can then distinguish 5 cases for how v can be used in OPT:

1. OPT has a median m with $loc(m) = v$ that can facilitate q' demand.
2. There is some demand $q'_1 = \sum_{u \in T_l} f(u, v) \geq 0$ and $q'_2 = \sum_{u \in T_r} f(u, v) \geq 0$ placed on v in OPT, where the supplies from subtrees on v are $\sum_{\substack{m \in S \\ loc(m) \in V_l \cup V_r}} f(v, m) = 0$.
3. There is some supply $q'_1 = \sum_{\substack{m \in S \\ loc(m) \in V_l}} f(v, m) > 0$ and $q'_2 = \sum_{\substack{m \in S \\ loc(m) \in V_r}} f(v, m) > 0$ from facilities in subtrees available at v.
4. There is some allocation between subtrees $q'_3 = \sum_{\substack{u \in T_l \\ m \in S \\ loc(m) \in V_r}} f(u, m)$, demand moving out of left subtree $q'_1 = q'_3 + \sum_{u \in T_l} f(u, v) \geq 0$ and supply available from right subtree $q'_2 = q'_3 + \sum_{\substack{m \in S \\ loc(m) \in V_r}} f(v, m) > 0$
5. There is some allocation between subtrees $q'_3 = \sum_{\substack{u \in T_r \\ m \in S \\ loc(m) \in V_l}} f(u, m)$, demand moving out of right subtree $q'_1 = q'_3 + \sum_{u \ in T_r} f(u, v) \geq 0$ and supply available from left subtree $q'_2 = q'_3 + \sum_{\substack{m \in S \\ loc(m) \in V_l}} f(v, m) > 0$

Case 1 Given that OPT is a solution where m facilitates demand $q' = \sum_{u \in T_v} f(u, m)$ with $0 < q' \leq k$, then there exists a solution (S', f') for subproblem $(v, p - q')$ with $S' = S \setminus \{m\}$ where f' is obtained by placing the demand on m in f on v in f'. Since (S', f') is a solution for subproblem $(v, p - q')$, the cost of (S', f') is at least $F(v, p - q')$. The cost of solution (S, f) is then at least $F(v, p - q') + C$.

$$\begin{aligned} F(v, p) \geq & F(v, p - q') + C \\ \geq & min_{\substack{0 < q \leq k \\ p - q \geq -D}} F(v, p - q) + C \end{aligned} \tag{6.17}$$

Case 2 Here, $p \leq 0$ since we have a leftover demand of $|p|$ at v. Given that v receives a demand of q'_1 from terminals inside the left subtree and a demand q'_2 from terminals in the right subtree, there exists some solution (S_l, f_l) to $(v_l, -q'_1)$ with $\sum_{u \in T_{v_l}} f'(u, v_l) = q'_1$.
We can obtain this solution (S_l, f_l) by assigning the demand from terminals in the left subtree on v to v_l instead. We can do this similarly for (S_r, f_r). Because v_l, v_r are children of v, we have that $\forall u \in V_l$, $dist(u, v) = dist(u, v_l) + w(v_l, v)$ and $\forall u \in V_r$, $dist(u, v) = dist(u, v_r) + w(v_r, v)$.
Then, $F(v, p) =$ cost of (S_l, f_l) + cost of $(S_r, f_r) + |q'_1| \cdot w(v, v_l) + |q'_2| \cdot w(v, v_r)$.
By the definition of F, the cost of (S_l, f_l) is at least $F(v_l, -q'_1)$.
For a solution (S_r, f_r) to $(v_r, -q'_2)$ with $\sum_{u \in T_r} f(u, v_r) = q'_2$, we know the cost is at least $F(v_r, -q'_2)$.

$$\begin{aligned} F(v, p) \geq & F(v_l, -q'_1) + F(v_r, -q'_2) + q'_1 \cdot w(v_l, v) + q'_2 \cdot w(v_r, v) \\ \geq & min_{q_1 + q_2 = p} \big(F(v_l, q_1) + F(v_r, q_2) + |q_1| \cdot w(v, v_l) + |q_2| \cdot w(v, v_r) \big) \end{aligned} \tag{6.18}$$

Case 3 Here, $p > 0$ since we have a leftover supply of p at v. Given that OPT receives a supply of q_1' from facilities inside the left subtree and a supply of q_2' from facilities in the right subtree, there exists some solution (S_l, f_l) to (v_l, q_1') with $\sum_{m \in S_l} f_l(v_l, m) = q_1'$ and cost at least $F(v_l, q_1')$. A similar solution (S_r, f_r) exists for (v_r, q_2').
We can obtain the solution (S_l, f_l) by assigning the leftover supply from facilities in (S, f) placed on v from facilities in V_l on v_l instead, and similarly for the solution (S_r, f_r).

$$\begin{aligned} F(v,p) \geq & F(v_l, q_1') + F(v_r, q_2') + q_1' \cdot w(v, v_l) + q_2' \cdot w(v, v_r) \\ \geq & min_{q_1+q_2=p}\big(F(v_l, q_1) + F(v_r, q_2) + q_1 \cdot w(v, v_l) + q_2 \cdot w(v, v_r)\big) \end{aligned} \tag{6.19}$$

Case 4 In our solution OPT, we have a demand q_3' from terminals in T_l that is assigned to facilities in V_r. Let such assignments $f_3 = \{(u, m) \mid f(u, m) > 0 \wedge u \in T_l \wedge m \in S \wedge loc(m) \in V_r\}$.
Then $\forall (u, m) \in f_3 : dist(u, m) = dist(u, v_l) + w(v_l, v) + w(v, v_r) + dist(v_r, m)$.
Similarly, let $f_1 = \{(u, v) \mid f(u, v) > 0 \wedge u \in T_l\} \cup f_3$.
Then $\forall (u, m) \in f_1 : dist(u, v) = dist(u, v_l) + w(v_l, v)$.
Let $f_2 = \{(v, m) \mid f(v, m) > 0 \wedge m \in S \wedge loc(m) \in V_r\} \cup f_3$.
Then $\forall (u, m) \in f_2 : dist(v, m) = dist(v_r, m) + w(v_r, v)$.

Given OPT has demands q_1' for which the assigned facility is outside V_l, there is a solution to $(v_l, -q_1')$ with $\sum_{u \in T_l} f(u, v_l) = q_1'$ and cost at least $F(v, -q_1')$.
Given that OPT has q_2' supply that is provided by facilities in V_r to a vertex outside of V_r, there is a solution (v_r, q_2') with facilities S_r and $\sum_{m \in S_r} f(v_r, m) = q_2'$ with cost at least $F(v_r, q_2')$.
We then have:

$$\begin{aligned} F(v,p) \geq & F(v_l, -q_1') + F(v_r, q_2') + (q_1' - q_3') \cdot w(v_l, v) + q_3' \cdot (w(v_l, v) + w(v_r, v)) + (q_2' - q_3') \cdot w(v_r, v) \\ = & F(v_l, -q_1') + F(v_r, q_2') + q_1' \cdot w(v_l, v) + q_2' \cdot w(v_r, v) \\ \geq & min_{q_1, q_2=p}\big(F(v_l, q_1) + F(v_r, q_2) + |q_1| \cdot w(v, v_l) + |q_2| \cdot w(v, v_r)\big) \end{aligned} \tag{6.20}$$

Case 5 Is proved by symmetry with case 4

Then for each case, we have that either $F(v,p) \geq \min_{0<q\leq k} \big(F(v, p-q) + C\big)$ where $p - q \geq -D$ or $F(v,p) \geq min_{q_1,q_2=p}\big(F(v_l, q_1) + F(v_r, q_2) + |q_1| \cdot w(v, v_l) + |q_2| \cdot w(v, v_r)\big)$.

Then

$$F(v,p) \geq min \begin{cases} \min_{q_1+q_2=p} \big(F(v_l, q_1) + F(v_r, q_2) + |q_1| \cdot w(v_l, v) + |q_2| \cdot w(v_r, v)\big) \\ \min\limits_{\substack{0<q\leq k \\ p-q\geq -D}} \big(F(v, p-q) + C\big) \end{cases}$$

Now that we've proved that the minimum cost for a subproblem at a leaf equals the cost given by equation 6.12, and that the minimum cost for an internal node v for the subproblem (v, p) is at least and at most the cost given by the equation 6.13, we can conclude the value given by our recursive formula equals the minimum cost of a solution to the subproblem (v, p). □

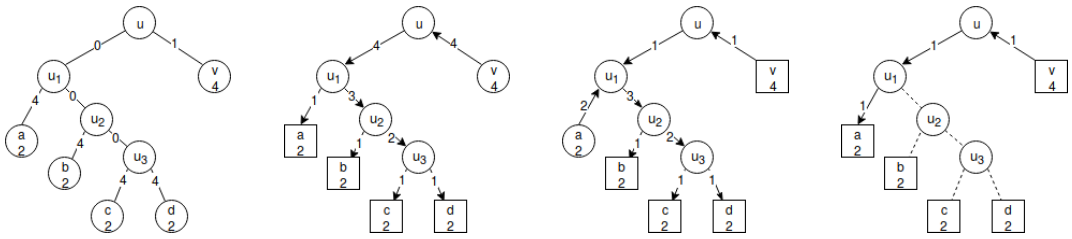
6.4.1 Complexity and Solution

The recurrence translates to a dynamic programming algorithm in the standard way, as described in Chapter 15 of Cormen et al. [47]
Computing F for subproblem (v, p) requires at most D steps to determine the minimum solution using all possible values of q_1 and q_2 for the left and right subtree, and at most k steps to determine the minimum of a solution (v, q) with $p - k \leq q \leq p$.
We can state that $k = min(D, k)$, because a value $k > D$ places no capacity constraint on the problem instance and will produce an identical solution to $k = D$.

Computing F at v for all relevant values p then takes $O(D^2)$.
For a tree with n vertices, the total running time is then $O(D^2 \cdot n)$.

Using the table of values given by the dynamic programming algorithm, we can find a solution (S, f) by starting at the root node with $p = 0$ and then picking the minimum cost subproblems using the recursive formula. This table however only provides information on the placement of facilities and the amount of demand or supply going into or out of subtrees. We can use this information to create a forest. We add a directed edge with the capacity q_1 or q_2 and direction from terminal to facility to a new graph, along with the nodes encountered during our traversal through the solution table. Similarly, nodes receive an attribute with the amount of facilities placed there.

We then compute the assignment function by following the paths from each terminals to a facility in this forest. Splitting the demand of a terminal may be required when there is a node with out-degree 2, or when there is a facility at that location and a positive out-degree.



(a) Input with edge weights on edges.

(b) Optimal solution with cost 84.

(c) Suboptimal solution with cost 85.

(d) Suboptimal solution with cost 85.

Figure 6.4: Example input and output with $k = 3$ and $C = 16$

There are generally many different cost-equivalent assignments functions for an optimal solution.
Consider the example in figure 6.4c, here we can assign $f(v, b) = 1$, $f(a, c) = 1$ and $f(a, d) = 1$.
Another valid and cost-equivalent assignment is $f(v, d) = 1$, $f(a, b) = 1$ and $f(a, c) = 1$.
We found the demand of a terminal is regularly split in practice between different facilities, since a terminal with high demand may not fit in its entirety on a single facility.

We can reduce the running time by using Lemma 6.3. Given a facility m placed on v facilitating $|q_1|, |q_2|$ demands from subtrees V_{v_l}, V_{v_r}, we can limit the leftover supply of m given by $f(v, m)$ to $min(|q_1| + |q_2|, k - (|q_1| + |q_2|))$, since there exists an optimal solution where $f(v, m)$ does not exceed $|q_1| + |q_2|$.
Since the networks of facilities can overlap, we cannot limit the value of p in a subproblem (v, p) by k. However, the demand from the left and right subtrees placed on facilities located on v is at most the total demand of those subtrees. Thus, there exists an optimal solution where the value of p does not exceed $D_v = \sum_{u \in T_v} d(u)$.
The demand in subtree V_v is given by D_v. If no demand in the subtree is satisfied, the value of p at v is given by $-D_v$, so p can never be lower than $-D_v$. Then, there always is an optimal solution for an input instance composed of solutions of subproblems (v, p) that satisfy $-D_v \leq p \leq D_v$.
This then also holds for the subtrees V_l, V_r with demands D_{v_l}, D_{v_r} for the values of q_1 and q_2.
Then, for any value of $p = q_1 + q_2$, the amount of possible values for q_1 is bounded by $2 \cdot D_{v_l}$ and similarly for q_2, because values of q_1 and q_2 outside that range are not required and therefore need not be defined for the subtrees given by V_l, V_r. Thus, for each value of p in (v, p), there are only $2 \cdot \min(D_{v_l}, D_{v_r})$ combinations to obtain that value of p.
The amount of computations then does not depend on the subtree size of V_v itself for a particular value of p, but instead depends on the size of the subtree with the smallest total demand.

Lemma 6.9. *For a binary tree $T = (V, E)$ with n nodes, $\sum_{x \in V} \min(|V_l(x)|, |V_r(x)|) = O(n \log n)$ where $V_l(x), V_r(x)$ are the nodes in the subtrees given by the left and right children of x respectively.*

Proof. We will prove this by proving that for any $x \in V$, at most $\log n$ ancestors of x will have x in the smaller of their 2 subtrees.
Let $Y = \{y_1, y_2, \ldots, y_k\}$ be the ancestors of x, such that the path from x to the root visits them in the order $y_1, y_2, \ldots, y_k$, where $y_i \in Y$ and for each y_i, x belongs to the minimum-size child subtree of y_i.
For any $y_{i+1} \in Y$, the subtree containing x given by V_{y_i} can be at most as large as the subtree not containing x for the smallest subtree of y_{i+1} to contain x. Thus, the size of the subtree $V_{y_{i+1}} \geq 2 \cdot V_{y_i}$.
Then, for a $y_i \in Y$ to contain x in the smallest subtree, its subtree must be at least double the size of the smallest subtree given by its children.
Since the total amount of vertices is n, we can at most double the size of the subtree $\log n$ times before it exceeds n. Thus, the amount of ancestors of x that contain x in their smallest subtree is at most $\log n$.

Since the amount of occurrences of each node in the smallest subtree of an ancestor is at most $\log n$, we conclude $\sum_{x \in V} \min(|V_l(x)|, |V_r(x)|) = O(n \log n)$. □

We can apply Lemma 6.9 to our setting as well, where each demand given by a node occurs at most $\log D$ times in the recursive calls. This results in a lower running time.

Theorem 6.10. *The running time of the capacitated median algorithm is given by* $O(n \cdot D \log D + n \cdot k \cdot D)$.

Proof. The application of Lemma 6.3 reduces the time required at each node v for all values of p to $O(D \log D)$ when checking the values given by the subtrees. We still require $O(k \cdot D)$ time when checking lower values of p to determine whether to place a DP for all values of p at node v. The running time for all nodes in the tree is then given by $O(n \cdot D \log D + n \cdot k \cdot D)$. □

In practice, it may also be be useful to limit the range of p to a constant factor of k when k is relatively small and D is large. This reduces the running time to $O(k^2 \cdot n)$, but can generate suboptimal results when an optimal solution would have more than k demand traversing an edge. An example is shown in figure 6.4, where (b) is an optimal solution containing an edge that carries more than k demand. The solutions in figures 6.4c and 6.4d show possible solutions when we do not allow $|p| > |k|$.

6.4.2 Modifications for Fiber Networks

In our fiber-network setting, we do not want the cable from a customer connected to a DP to cross or be placed in the same ditch as a cable from another customer going to a different DP. We also always want all demand of a customer to be facilitated by the same DP.
In other words: In the forest given by the directed edges retrieved from a solution, a facility cannot have a path to another facility.

This requirement results in problems when customers have demand larger than k. The *Programma van Eisen*(design brief) provided by *ReggeFiber*[48] states that such customers receive their own DP within the building.
We therefore pre-process the input tree such that nodes with $d(v) > k$ are removed along with their edges. We keep track of the nodes removed, and add an additional facility at those nodes with capacity equal to $d(v)$.
Other problem settings may require different pre-processing to handle such terminals.

We can alter our algorithm to only produce solutions in which there is exactly 1 facility reachable from any terminal fairly easily: We do not allow any node to have multiple outgoing edges, which go from terminals to facilities, in our solution.

Each possible combination is shown in figure 6.5. The cases with out-degree greater than 1 are (a), (e), (f) and (h). The cases that we need to allow are then given by $(p < 0 \wedge q_1 \leq 0 \wedge q_2 \leq 0) \vee (p \geq 0) \wedge (q_1 \leq 0 \vee q_2 \leq 0)$.

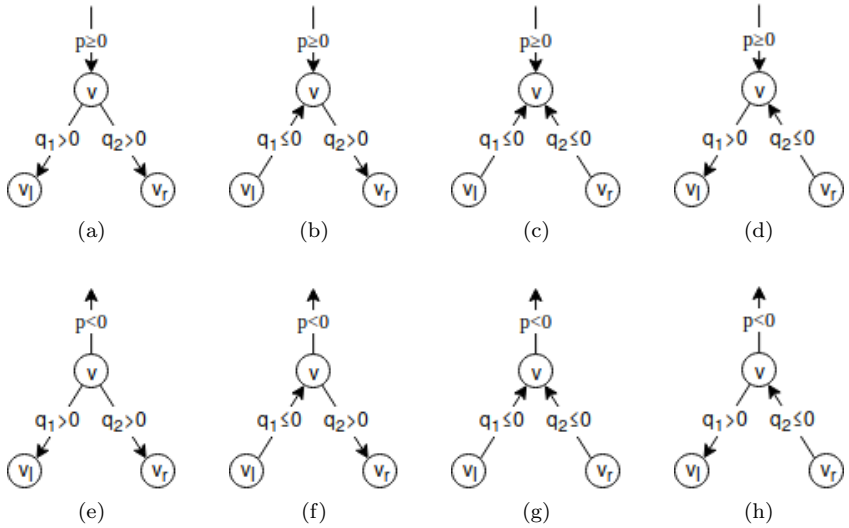


Figure 6.5: All combinations of q_1, q_2 and p values for a vertex v.

We can now also bound p, q_1 and q_2 by $\{-k, \ldots, k\}$, because having the absolute value of any greater than $|k|$ will always require splitting the demand to multiple facilities.

After applying this check on the cases, there may still be some edge cases in which networks can overlap. For example when placing a facility on one of the new nodes added during the transformation to a binary tree.
We add an infinite cost $pc(v) = \infty$ for placing facilities on new nodes v, such that the adjacent original node u with $pc(u) = 0$ will be used instead when a facility is required.

Since a facility is not explicitly forced to facilitate all demand, it may also be the case that part of the incoming demand is passed on. Since the facility will always have enough capacity available, given $|p| \leq |k|$, passing on demand to a next edge is always more expensive if that edge has positive weight.
Therefore, nodes in the original dataset with a 0 weight edge between them should be contracted. We can still allow new nodes during the transformation with a 0 weight edge because the placement cost on new nodes is infinite.
The solution shown in figure 6.4d is then an optimal solution for non-overlapping networks.

The new recursive formula is then given by:

For a leaf v:

$$F(v,p) = \begin{cases} \lceil (p + d(v))/k \rceil \cdot (C + pc(v)) & \text{if } p + d(v) > 0 \\ 0 & \text{otherwise} \end{cases}$$

For an internal node v with children v_l, v_r

$$F(v,p) = \min \begin{cases} \min\limits_{\substack{q_1+q_2=p \\ (p<0 \wedge q_1 \leq 0 \wedge q_2 \leq 0) \\ \vee (p \geq 0 \wedge (q_1 \leq 0 \vee q_2 \leq 0))}} \big(F(v_l, q_1) + F(v_r, q_2) + |q_1| \cdot w(v_l, v) + |q_2| \cdot w(v_r, v)\big) \\ \min\limits_{\substack{0<q\leq k \\ p-q \geq -k}} \big(F(v, p-q) + C + pc(v)\big) \end{cases}$$

Because we no longer depend on the total demand, $-k \leq p \leq k$, so the running time for this variant is $O(k^2 \cdot n)$.

6.5 Experimental Evaluation

In this section, we perform a set of experiments on the given facility location algorithms.
We first measure the effect of the reduction rule presented in Lemma 6.1 and the transformations from section 6.1 on the tree size.
We then compare the running times and cost of solutions between our optimal capacitated median algorithm, the greedy algorithm and the p-median algorithm.

The experiments were performed on the same device as listed in section 5.1. The algorithms were implemented in Python using the NetworkX graph library [49].
We set $k = 48$, $C = 1000$ and, when evaluating the p-median algorithm, p equal to the amount of facilities reported by the capacitated median algorithm plus one. For our capacitated algorithm, we do not allow overlapping networks. The modification of section 6.4.1 based on Lemma 6.3 was not implemented.

The input is created by computing the complete geographic model for an area, then reporting the first Steiner tree solution within 0.01%. Along with the areas used before, we add a section of Middenbeemster (MB) with 700 customers.

Instance	$\|T\|$	D	Input $\|V\|$	Reduced $\|V\|$	Transformed $\|V\|$
HHW	157	157	944	306	313
HHW2	364	365	2883	714	727
MB	700	765	4655	1375	1399
Emmeloord	1315	1431	7794	2578	2629
Empel	2239	2361	18397	4356	4477
Gilze	2721	3705	18788	5333	5441

Table 6.1: Result of reduction and transformation on tree size.

The results of the reduction and transformation into a binary tree shown in table 6.1 immediately highlights the importance of applying the reduction rule. The reduction rule results in a size decrease of 66% to 76% on our instances. The effect of the transformation to a binary tree afterwards is very small, only increasing the size by 2% to 3%.
We can base the size of the reduced and transformed tree completely on the value of $|T|$. Since each terminal is a leaf, and there are no other leaves, we can conclude that the size of the binary tree is given by $2 \cdot |T| - 1$. We note that in the Gilze instance, 1 terminal has demand greater than k and was given its own DP and removed from the tree beforehand, leading to a different amount of terminals compared to earlier computational results.

			Capacitated Median				Greedy				p-median			
Instance	D	$\|V\|$	t[s]	L [km]	$\|S\|$	Cost	t[s]	L [km]	$\|S\|$	Cost	t[s]	L [km]	$\|S\|$	Cost
HHW	157	313	0,83	8,79	6	14789,9	0,06	14,64	4	18640,8	1,70	8,79	6	14789,9
HHW2	365	727	2,07	21,22	14	35224,7	0,18	34,30	9	43303,6	16,87	21,22	14	35224,7
MB	765	1399	5,28	45,02	29	74018,3	0,5	74,91	19	93905,1	143,45	45,01	29	74008,4
Emm.	1431	2629	8,82	71,57	56	127573,7	0,46	120,81	37	157808,3	Out of memory			
Empel	2361	4477	13,1	132,92	101	233915,4	1,43	243,06	57	300064,4	Out of memory			
Gilze	3705	5441	16,01	157,30	128	285302,2	1,11	276,72	74	350717,9	Out of memory			

Table 6.2: Computational results of different facility location algorithms

The results in table 6.2 show a clear linear relation between the running time and input size for the capacitated median algorithm. The results for the greedy algorithm also show this linear relationship, but not as clearly. On the other hand, the limited amount of results for the p-median algorithm shows a quadratic relationship. This confirms the expectations given by the running times.

The cost of solutions given by the greedy algorithm is a lot higher than the optimal value, whereas the cost of p-median solutions is almost identical to the optimal solution for the capacitated problem. As we can see, using the minimum amount of distribution points is not a good

strategy; most facilities in an optimal solution are not filled to their maximum capacity. We think this results in the solutions for the p-median problem to be nearly identical.

We expect that increasing C will result in the cost of solutions for the greedy algorithm to be closer to the optimal cost, while decreasing the similarity to the solutions given by the p-median algorithm, as this should lead to facilities being filled to their capacity more often.

Chapter 7

Results and Discussion

In the previous chapters, we've developed, used and evaluated different algorithms for each problem separately. We now combine the algorithms into a usable algorithm for the fiber network planning problem and discuss the advantages and shortcomings of our approach.

7.1 Results

We first perform an additional set of tests on the combined algorithms. We measure the total running time of our approach and evaluate the effect of applying the facility location algorithm after the Steiner tree algorithm.
Finally, we compare our complete approach with manually constructed networks.

In our tests, we use the same areas as before. We apply the complete geographic model, use the first solution from SCIP-Jack that is provably within 0.1% of an optimal solution and apply our capacitated median algorithm with $k = 48$, $C = 1000$ and do not allow networks in the solution to overlap. Our experiments were performed on an Intel Core i7-7700HQ CPU with 2.7 GHz and 16GB RAM running Ubuntu 18.04. We used FME Desktop version 2018.0.1.0 for our graph construction (GC), SCIP version 5.0.1 with SoPlex LP solver version 3.1.1 as Steiner tree solver (STP) and Python 3.6 along with NetworkX version 2.1 for the Capacitated Median algorithm (CM). The running time for each part is shown in table 7.1.
When comparing to manually constructed networks, we list the STP cost, total cable length and amount of DPs. We remove cables and ditches within private property.

Instance	Area $[km^2]$	$\|V\|$	$\|E\|$	$\|T\|$	GC [s]	STP [s]	CM [s]	Total [s]
HHW	0,07	4921	6470	157	10,9	0,3	0,8	12
HHW2	0,14	13861	18859	364	19,5	1,5	2,1	23
MB	0,30	20799	56260	700	73,5	6,1	5,3	85
Emmeloord	0,74	42353	56752	1315	59,7	4,4	8,8	73
Empel	1,54	102574	137003	2239	288,3	23,6	16,0	328
Gilze	2,32	85133	111345	2722	190,0	13,3	13,1	216

Table 7.1: Total running times for network planning

The running time clearly shows that the graph construction is the bottleneck of our approach. As we've noted before, we think this is due to the implementation of the geometric operations in FME. Intersecting lines in FME often shows quadratic running time behaviour, while we do not expect a quadratic number of intersections given our construction method. We expect that improving the implementation of the geometric operations will decrease the running time of this part significantly.

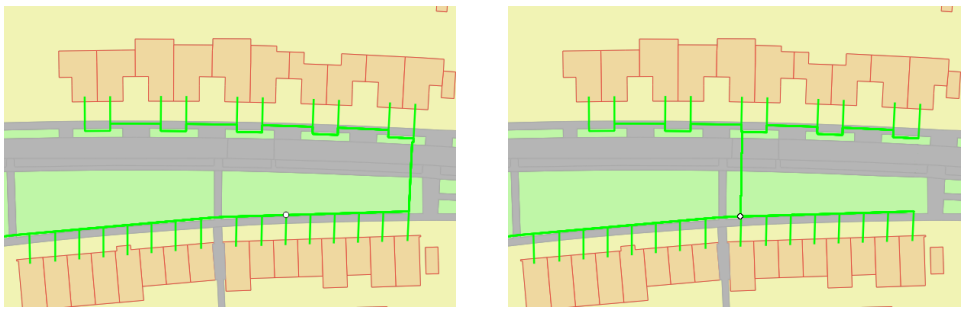
SCIP-Jack performs very well on our complete model, though the base model for some of our large datasets was around the limit of its abilities. Additional testing with larger input sets will be required to accurately determine the limit of SCIP-Jack on our complete model. We think the capacitated median algorithm will not be a bottleneck in our system. The size of the tree is always $2 \cdot |T|$, providing running times linear to the amount of terminals, given that $k \leq 48$.
It's interesting to note that the running time of the capacitated median algorithm is very similar to the SCIP-Jack running time. STP is a NP-Hard problem, while the capacitated facility location problem in trees is not. We think the running time of the capacitated median could be significantly reduced by an implementation in C or C++ instead of Python, as algorithms implemented in C++ are often over 10 times faster than similar implementations in Python [50]. Additionally, the initial heuristic solution given by the STP solver is already very close to an optimal solution on our input. SCIP-Jack often shows a linear relationship between $|E|$ and the running time on our input.

	Manual			Computational		
	STP NC Cost	Cable Length	#DPs	STP NC Cost	Cable Length	#DPs
HHW2	9273	18609	15	7933	15897	13
Empel	64000	104064	116	59348	111923	101

Table 7.2: Cost comparison to manual networks

Comparing the manually constructed networks to our results shows our approach requires a smaller amount of DPs and cables. Our results show a decrease in cost between 3% and 14% when placing facilities. While the results so far are very promising, we think more data is required to be able to draw a strong conclusion.

One of the choices we made initially is that we apply the facility location algorithm after the Steiner tree algorithm. While our facility location results are optimal on the tree, we commonly see suboptimal placements when we look at the results on the graph.
One such case is shown in figure 7.1(a). Due to the distance of a crossing ditch being larger near the center of the figure, a Steiner tree will contain the edge on the right. For a slightly larger ditch cost, we can decrease the cable length by hundreds of meters in this case, as shown in figure 7.1(b). While the shown example is one of the most extreme cases, we think the cost can be significantly decreased by altering the tree based on the original graph.



(a) Facility Location solution (b) Possible Improvement

Figure 7.1: Suboptimal facility location solution.

7.2 Discussion

In chapter 4 we presented a series of geometric operations that we can apply to the geographic input to obtain a representative graph. The size of the constructed graphs was smaller than the input data and allowed for quick computation of near-optimal Steiner trees, while also representing

the real world well.
There are several issues remaining with the construction of the graph. The inability to determine the front door of a building introduces many errors in the network. This would require a better heuristic method when picking a road segment to connect to, or additional data, such as street names or more accurate placed address points. Until then, manual corrections need to be applied to obtain a valid network.
Manually correcting this issue is quite simple: Placing an additional labeling point with the correct street name on a road segment will cause the customer connection to connect to the correct road, solving most problems. In other cases, the address point could be moved or the customer connection could be drawn by hand.
The second issue is that ditches often follow the perimeter of a polygon too closely, which leads to jagged lines in our graph. These cases could be solved during future post-processing steps or additional steps during graph creation to find straighter paths. For now, these issues also require manual corrections.
Finally, there are still some missing viable crossings, which leads to higher cost Steiner trees. Solving this would require some method of determining where road intersections are given the road polygons.

A big advantage of creating a graph first, is that it can be modified and extended further. New edges can be added manually, for instance when a crossing was not present, or edges can be manually modified if the cost is not accurate. Modifying the cost function to give more accurate weights can further improve solutions. The cost function can also be easily modified using additional datasets, for instance using a dataset giving the locations of trees or a dataset listing contaminated areas. There also exists data about existing ducts, which can often be used to place additional cables in without much digging. Adding these as edges with low cost and connecting them to the rest of the network can improve solutions further.
The graph can however not easily represent one-time costs, such as the cost of obtaining a permit for placing a ditch in an area. We are limited to the average cost of a ditch in such an area.
Another possible issue is that the cost of a ditch for a customer connection could have a different cost when passing though another customer's property. The current model will typically connect directly to the road, and not consider shortcuts through private property, but possible extensions that add these shortcuts will need to define these costs very clearly and find a way to represent it in the graph.
Another issue that may arise is the placement of intersections within coverage types that require a drilling. It's not realistic to perform multiple drillings that intersect in the middle of an asphalt road, though this may be possible in our graph. Because the cost of a drilling is often much higher than digging in the surrounding area, we expect a Steiner tree solution to minimize the distance through asphalt by picking intersecting points besides it. In our instances, we have not found any cases where there is a node in the Steiner tree with degree 3 or higher within an asphalt road segment. The addition of a tree dataset may however introduce these errors. If these issue present themselves, we should modify the model by not allowing intersection points to be added when intersecting lines or overlaying if those points would be placed within such areas, and removing any nodes with degree 3 or higher within these areas.
We compared our solutions to manually constructed networks multiple times, but the results on that are not conclusive at this point. The cost function on the edges is not directly related to the actual cost, so the manually constructed networks may in fact be cheaper when using a more accurate cost function. The manual data has also been known to contain inaccuracies. A conclusive comparison would require an accurate cost function, more areas to compare to and a more robust dataset.

We think the construction of a graph is necessary for a quick solution at this time. A rasterization approach, as in Frommer et al. [11], was shown to produce extremely large graphs and requires long computation times, and to the best of our knowledge, there are no significant results for Steiner tree problems on weighted subdivisions.
The generation of the graph based on the polygons instead of on the center lines of roads provides

a more accurate graph. This reduces the cost of a network further and also decreases the amount of manual labor required when correcting the network. Currently, the graph creation takes up the largest amount of time. The use of more efficient algorithms can likely decrease this time significantly.

Splitting the graph minimization problem into 2 separate problems, the Steiner tree and facility location problem respectively, allowed us to generate valid solutions for very large areas. Recent capacitated median algorithms such as the algorithm by Stefanello et al. [32] show long computation times for instances with thousands of customers and are limited to placing facilities on customers. Recent Steiner tree solvers have been able to deal with much larger instances quickly. Considering that the Steiner tree costs in our setting are much higher than the facility location costs, we think solving for a near-optimal Steiner tree first has lead to quicker and more cost-efficient results. We expect that combining the cost functions of the median problem and Steiner tree problem will invalidate a large amount of reduction rules for the Steiner tree problem. Therefore, we do not expect a mixed-integer approach to the combined problem to be able to solve instances of our size at this time. Additionally, we believe the cost added due to separating the problems can be mitigated using post-processing techniques based on the approach of Lin and Xue [10].

Chapter 8

Conclusion

In this thesis we have covered a wide variety of algorithmic problems that occur when designing fiber networks. We've split the problem into 3 separate parts, a geographic problem, a Steiner tree problem and a facility location problem.
In chapter 4 we have shown how common geometric operations can be applied to transform a geographic dataset into a representative graph to obtain practical solutions to our geographic problem.
We applied several modifications to decrease the running time and improve the solutions further. For the Steiner tree problem, we used the SCIP-Jack [1] solver to very quickly obtain near optimal solutions. After improving the base model from chapter 4 by dissolving roads, we obtained optimal solutions within 10 minutes for all input areas.
We then developed a new optimal algorithm for the Capacitated Median problem in trees. By using the constraint in fiber networks that the network of one facility should not contain cables going to other facilities, we were able to bound the running time by $O(k^2 \cdot n)$ for maximum capacity k for a tree with n nodes.

By combining the solutions of each problem, we end up with an algorithm that can automatically find customers in an area, connect each customer to the rest of the network and place distribution points, all while minimizing the expected construction costs.
The approach can be easily extended using additional datasets and allows manual corrections at various point. Our approach produces accurate networks for large sections of the input area, though many manual corrections are still required. Our algorithm has already seen use by network engineers and has saved days of work. In the limited amount of areas for which manually constructed data was available, the expected cost of our solution was lower compared to the manually constructed network, both in terms of ditch costs, and DP and cable costs.
The approach works very well in (sub-)urban areas and is able to calculate fiber network plans within 10 minutes for thousands of customers.

8.1 Future Work

Many extensions and improvements can still be applied to further improve the fiber network planning.
We can use additional datasets, such as trees, contaminated areas or existing ducts to improve the cost function and add additional edges.
The customer connections could potentially be improved using the cadastral map, which is a polygonal subdivision dividing the land by owner. The cadastral map often indicates alleyways as private property, allowing those to be removed from the set of roads a customer can connect to. Additionally, it provides additional labels with street names.
As noted before, we can apply post-processing on each DP network to reduce cable costs based on the algorithms from Lin and Xue [10]. The capacitated median algorithm could be repeated

on the resulting tree to possibly assign customers to a different DP and further decrease cost.
We have listed costs for different sizes of DPs before, but we've only considered the 48 capacity DPs in chapter 6. The capacitated median algorithm on trees can likely be extended easily to allow for facilities with different capacities and costs. We think this will be an important addition in rural areas, because the distance between buildings will lead to higher cable costs, which in turn will lead to less DPs being filled to capacity.
The implementation of the adaptation for the capacitated median algorithm given in section 6.4.1 using Lemma 6.3 has not been experimentally evaluated yet. We think it will significantly decrease the running time for the variant in which we allow networks to overlap, and will also have an effect on the variant in which network cannot overlap. We think it is likely the $O(k \cdot D)$ running time required at each node checking for facility placement on that node can be further reduced as well, which could decrease the running time for the capacitated median algorithm to $O(n \cdot D \log D)$.
Finally, we think a more efficient implementations of geometric operations and of the capacitated median algorithm will further decrease the running time and increase the usefulness of our approach.

Bibliography

[1] Gerald Gamrath, Thorsten Koch, Stephen J Maher, Daniel Rehfeldt, and Yuji Shinano. SCIP-Jack—a solver for STP and variants with parallelization extensions. *Mathematical Programming Computation*, 9(2):231–296, 2017. 3, 4, 7, 29, 54

[2] KSavi. FTTx / FTTH network planning and design software based on QGIS, 2017. `https://ksavinetworkinventory.com/free-network-design-software/`. 5

[3] Kurt Mehlhorn. A faster approximation algorithm for the Steiner problem in graphs. *Information Processing Letters*, 27(3):125–128, 1988. 5

[4] Christoph Zonsius. Automated and cost-optimized strategic fiber planning with the ArcGIS platform, 2016. 5

[5] Comsof NV. Fiberplanit - strategic FTTx network planning, 2016. `https://fiberplanit.com/`. 5

[6] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972. 6

[7] Michael R Garey, Ronald L Graham, and David S Johnson. The complexity of computing Steiner minimal trees. *SIAM journal on applied mathematics*, 32(4):835–859, 1977. 6

[8] Samir Khuller, Balaji Raghavachari, and Neal Young. Balancing minimum spanning trees and shortest-path trees. *Algorithmica*, 14(4):305–321, 1995. 6

[9] K. Bharath-Kumar and J. Jaffe. Routing to multiple destinations in computer networks. *IEEE Transactions on Communications*, 31(3):343–351, 1983. 6, 14

[10] Guo-Hui Lin and Guoliang Xue. Balancing Steiner minimum trees and shortest-path trees in the rectilinear plane. In *Circuits and Systems, 1999. ISCAS'99. Proceedings of the 1999 IEEE International Symposium on*, volume 6, pages 117–120. IEEE, 1999. 6, 14, 53, 54

[11] Ian Frommer, Bruce Golden, and Guruprasad Pundoor. Heuristic methods for solving euclidean non-uniform steiner tree problems. In *Genetic and Evolutionary Computation Conference*, pages 392–393. Springer, 2004. 6, 52

[12] Stuart E Dreyfus and Robert A Wagner. The Steiner problem in graphs. *Networks*, 1(3):195–207, 1971. 7

[13] Ranel E Erickson, Clyde L Monma, and Arthur F Veinott Jr. Send-and-split method for minimum-concave-cost network flows. *Mathematics of Operations Research*, 12(4):634–664, 1987. 7

[14] Daniel Mölle, Stefan Richter, and Peter Rossmanith. A faster algorithm for the Steiner tree problem. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 561–570. Springer, 2006. 7

[15] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets möbius: fast subset convolution. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 67–74. ACM, 2007. 7

[16] Jesper Nederlof. Fast polynomial-space algorithms using möbius inversion: Improving on Steiner tree and related problems. In *International Colloquium on Automata, Languages, and Programming*, pages 713–725. Springer, 2009. 7

[17] Stefan Fafianie, Hans L. Bodlaender, and Jesper Nederlof. Speeding up dynamic programming with representative sets: An experimental evaluation of algorithms for steiner tree on tree decompositions. *Algorithmica*, 71(3):636–660, Mar 2015. 7

[18] Marcin Pilipczuk, Michal Pilipczuk, Piotr Sankowski, and Erik Jan van Leeuwen. Network sparsification for steiner problems on planar and bounded-genus graphs. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 276–285. IEEE, 2014. 7

[19] Ondřej Suchỳ. Extending the kernel for planar steiner tree to the number of steiner vertices. *Algorithmica*, 79(1):189–210, 2017. 7

[20] L. Berman L. Kou, G. Markowsky. A fast algorithm for Steiner trees. *Acta Informatica*, 15:141–145, 1981. 7, 13

[21] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. An improved LP-based approximation for steiner tree. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 583–592. ACM, 2010. 7

[22] Miroslav Chlebík and Janka Chlebíková. The steiner tree problem on graphs: Inapproximability results. *Theoretical Computer Science*, 406(3):207–214, 2008. 7

[23] Eduardo Uchoa, Marcus Poggi de Aragão, and Celso C Ribeiro. Preprocessing Steiner problems from VLSI layout. *Networks*, 40(1):38–50, 2002. 7, 20

[24] DIMACS. 11th DIMACS implementation challenge in collaboration with ICERM: Steiner tree problems, 2014. `http://dimacs11.zib.de/`. 7, 29

[25] Thomas Pajor, Eduardo Uchoa, and Renato F Werneck. A robust and scalable algorithm for the steiner problem in graphs. *Mathematical Programming Computation*, 10(1):69–118, 2018. 7

[26] Matteo Fischetti, Markus Leitner, Ivana Ljubić, Martin Luipersbeck, Michele Monaci, Max Resch, Domenico Salvagnin, and Markus Sinnl. Thinning out steiner trees: a node-based model for uniform edge costs. *Mathematical Programming Computation*, 9(2):203–229, 2017. 7

[27] C.R. Moberg T.S. Hale. Location science research: A review. *Annals of Operations Research*, 123:21–35, 1971. 8

[28] S.L. Hakimi O. Kariv. An algorithmic approach to network location problems. II: The p-medians. *SIAM Journal on Applied Mathematics*, 37(3):539–560, 1979. 8, 38

[29] A. Tamir. An $O(pn^2)$ algorithm for the p-median and related problems on tree graphs. *Operations Research Letters*, 19(2):59–64, 1996. 8, 32, 37, 38

[30] J. Reese. Solution methods for the p-median problem: An annotated bibliography. *Networks*, 48:125–142, 2006. 8

[31] M. Rahbar M. Yaghini, M. Karimi. A hybrid metaheuristic approach for the capacitated p-median problem. *Applied Soft Computing Journal*, 13:3922–3930, 2013. 8

[32] Fernando Stefanello, Olinto C. B. de Araújo, and Felipe M. Müller. Matheuristics for the capacitated p-median problem. *International Transactions in Operational Research*, 22(1):149–167, 2015. 8, 53

[33] Kadaster. Basisregistratie grootschalige topografie, 2017. https://www.kadaster.nl/bgt. 11

[34] Kadaster. Basisregistratie adressen en gebouwen, 2018. https://www.kadaster.nl/bag. 11

[35] Ministerie van Infrastructuur en Milieu. Objectenhandboek BGT, 2016. http://imgeo.geostandaarden.nl/. 12

[36] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, 3rd ed. edition, 2008. 15, 16, 24

[37] Safe Software. FME feature manipulation engine. https://www.safe.com/, 2018. 15

[38] Nick Roussopoulos, Stephen Kelley, and Frédéric Vincent. Nearest neighbor queries. *SIGMOD Rec.*, 24(2):71–79, 1995. 16

[39] Cha S.K. Lee B.S. Hwang S., Kwon K. Performance evaluation of main-memory R-tree variants. *International Symposium on Spatial and Temporal Databases*, pages 10–27, 2003. 16

[40] Xiaorui Chen and Sara McMains. Polygon offsetting by computing winding numbers. *Design Automation Conference*, 2:565–575, 2005. 16

[41] W. S. Chan and F. Chin. Approximation of polygonal curves with minimum number of line segments. In *Algorithms and Computation*, pages 378–387. Springer Berlin Heidelberg, 1992. 16

[42] M. Hanan. On Steiner's problem with rectilinear distance. *J. SIAM Appl. Math.*, 1966. 19

[43] D. Rehfeldt. A generic approach to solving the Steiner tree problem and variants. Master's thesis, Technische Universität Berlin, 2015. 29

[44] Thorsten Koch, Alexander Martin, and Stefan Voß. Steinlib: An updated library on Steiner tree problems in graphs. In *Steiner trees in industry*, pages 285–325. Springer, 2001. 29

[45] M. Prossegger. Generation of a weighted network graph based on hybrid spatial data. In *Proceedings of the Fifth international conference on advanced Geographic Information Systems, Applications, and Services*, pages 120,124, 2013. 29

[46] A. J. Goldman. Optimal center location in simple networks. *Transportation Science*, 5(2):212–221, 1971. 38

[47] Ronald L. Rivest Clifford Stein Thomas H. Cormen, Charles E. Leiserson. *Introduction to Algorithms (2nd ed.)*. MIT Press and McGraw-Hill, 2001. 44

[48] ReggeFiber. Programma van Eisen versie 6.0, 2016. 46

[49] Daniel A. Schult Aric A. Hagberg and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, 2008. https://networkx.github.io. 48

[50] Isaac Gouy. The computer language benchmarks game, 2018. https://benchmarksgame-team.pages.debian.net/benchmarksgame/. 51