Heinrich-Heine-Universität Düsseldorf
Department of Computer Science
Algorithmic Bioinformatics

Optimisation problems for the reinforcement of telecommunication networks

Adrian Prinz

Master's Thesis

Submission:	September 16, 2022
Supervisor:	Prof. Dr. Gunnar W. Klau
Second Accessor:	Dr. Rolf Bardeli

Acknowledgements

I would like to thank my supervisors Prof. Gunnar W. Klau and Dr. Rolf Bardeli for their guidance and advice inside and outside our regular meetings from the beginning to the end of the thesis. Furthermore, I would like to thank Dr. Richard Schmied, who was often able to help me with his expertise. Without them, this thesis would not have been possible.
I would like to thank Vodafone GmbH for their cooperation. Lastly I thank Philipp Spohr and all the students of the Algorithmic Bioinformatics department for the regular discussions and tips.

Computational infrastructure and support were provided by the Centre for Information and Media Technology at Heinrich-Heine-University Düsseldorf.

Declaration

I hereby confirm that this thesis is my own work, and that I have only used the sources and materials specified in my thesis.

Düsseldorf, September 14, 2022

[illegible]

Adrian Prinz

Abstract

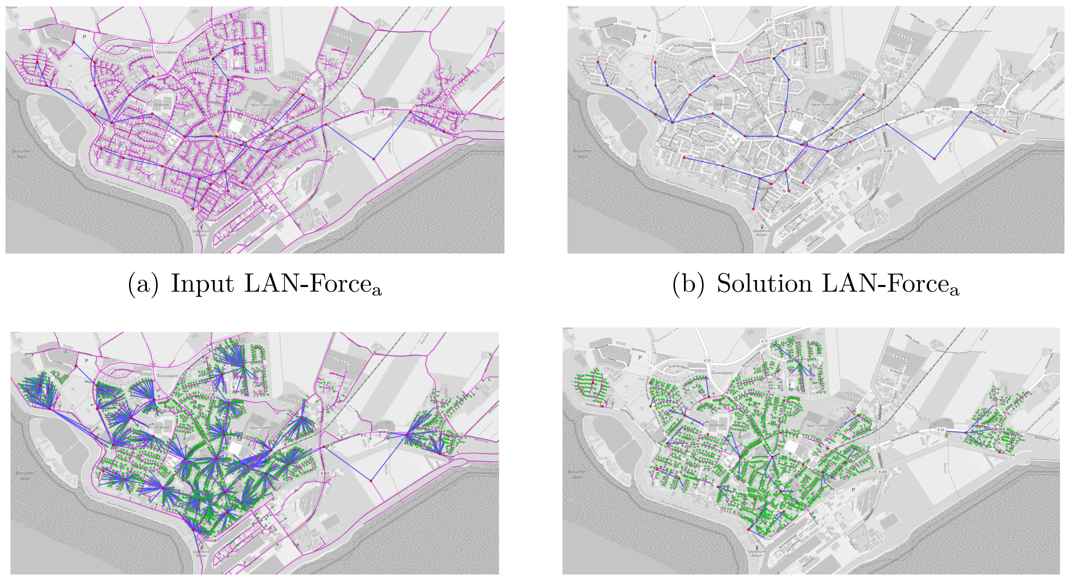
This thesis presents two new combinatorial optimisation problems that can be used to model the expansion of a coaxial telecommunications network using fibre optic cables. In these capacitated network design problems, it is necessary to install fibre optic cables in order to route enough traffic from different access points to the customers. The first problem deals with routing traffic from the access points to so-called amplifier points. Each amplifier point is already connected to the existing coaxial cable network. However, some amplifier points require more traffic than can be routed to them via coaxial cable. The goal of this problem is to reinforce the already existing coaxial cable network with fibre optic cables so that there is enough capacity in the network to sufficiently supply all amplifier points. For this reason, the problem is called Local Access Network Reinforcement problem for amplifier points (LAN-Force$_{\mathrm{a}}$). The second problem deals with the supply of traffic to the customers. The customers must be connected to the amplifier points via existing coaxial cables or via new fibre optic cables. Unlike the LAN-Force$_{\mathrm{a}}$ problem, not every customer has to be connected to the network. A customer is connected to the network if its demand for traffic can be satisfied. It gives a profit depending on whether it is connected to the network by fibre or by coax. The problem is called the Local Access Network Reinforcement problem for households (LAN-Force$_{\mathrm{h}}$).
We prove the NP-completeness of the two problems and propose different methods to solve them. We first reduce the instances without losing the optimal solution in order to speed up the overall solution process. We develop two different integer linear programs to solve the problems. Furthermore, we describe a heuristic for both problems in order to quickly find approximate results.
We perform computational experiments on real-world graphs from Vodafone GmbH. For smaller and densely populated instances we often find an optimal solution. For large instances, we are able to find approximate solutions, which were on average between 3 and 12 percent worse than the respective exact results.

Contents

1 Introduction

In today's world, it is important to be connected and up to date with others. The world is globalising, both in the private and public sector, individuals want to be informed and communicate. For this, a connection to the internet, as well as a telephone and TV connection is indispensable. In order to take advantage of the internet, telephone and TV, the private customers and companies receiving it must be connected to a cable network provided by telecommunications companies.
The data stream is routed from various access points via cables to so-called amplifier points. These amplifier points forward the data stream to the individual customers, who may be business buildings or private households. The cable network used for the supply consists of different types of cable. While in the past mainly the more inefficient coaxial cable consisting of copper was used, today's efforts to expand the network with more efficient fibre optic cables are significant. This process is called fibre rollout. Telecommunication companies are interested in a cost-efficient solution so that profit-paying customers are connected to the network as cheaply as possible. The planning of such a network is highly complex and contains various problems on graphs, see for example Bley et al. [1], Grötschel et al. [2], Voß et al. [3].
In telecommunications infrastructure planning, the goal is to optimally design the fibre rollout for single planning units. The planning units are counties, cities or whole municipalities. We can represent the planning units as a graph consisting of vertices and edges. We represent the access points, amplifier points, customers and street intersections (through which cables can be laid) as vertices and the cables connecting the different components of the network as edges of the graph.
This thesis deals with the optimal reinforcement of an existing coaxial cable infrastructure with fibre optic cables. The two types of cable differ in cost of cable laying and capacity. The capacity determines how much data can be routed over the cable and thus implies how many customers can be supplied via the respective cable. Each customer has its own demand, which corresponds to the number of flats or offices in the building to be supplied. The profit of connecting a building to the network depends on the demand.
For the optimal planning of the fibre roll-out, the Prize-collecting Local Access Network problem (PC-LAN) has already been defined in the literature [4]. The objective of the PC-LAN problem is to perform the most profitable cable laying by balancing the cost of cable laying against the profit of the connected customers. At each edge of the graph it is possible to choose one of many cables, which all differ in cost and capacity. A customer only pays the profit if it is connected to the network via cable and its demand is satisfied. If the cost of the network connection exceeds the customer's profit, the customer does not necessarily have to be connected to the network. However, there is a so-called overbuilding risk, which states that a certain percentage of all customers must be connected to the network. If the telecommunications companies do not take care to connect this percentage of all customers

to their own network, there is a risk that these customers will be served by rival companies and thus market power will diminish.
In this thesis we divide the process of supplying customers into two subproblems. This division into two problems makes sense in order to be able to formulate different optimisation functions and to deal with special specifications. The goal of the first problem is to provide all amplifier points with sufficient traffic. In the second problem, we will connect the customers to the amplifier points as profitably as possible. We call the first problem Local Access Network Reinforcement problem for amplifier points (LAN-Force$_a$) and the second problem Local Access Network Reinforcement problem for households (LAN-Force$_h$). While LAN-Force$_a$ is a special case of the PC-LAN problem LAN-Force$_h$ is only a variant. Both problems are NP-complete. Figure 1 shows an example input graph for both problems, together with a solution.



(a) Input LAN-Force$_a$

(b) Solution LAN-Force$_a$

(c) Input LAN-Force$_h$

(d) Solution LAN-Force$_h$

Figure 1: Input and solution of the problems LAN-Force$_a$ and LAN-Force$_h$. In the LAN-Force$_a$ we reinforce the existing blue coax network with pink fibre cables. For this, we connect all red amplifier points whose demand is not satisfied by the coax network to the yellow fibre access points via fibre cables.
The goal of the LAN-Force$_h$ problem is to connect the green households as profitably as possible via fibre and coaxial cable to the red amplifier points.

For the LAN-Force$_a$ we assign a demand to each amplifier point. The demand of an amplifier point corresponds to the cumulative demand of all customers connected to the respective amplifier points in the course of the LAN-Force$_h$ problem. In contrast to the PC-LAN problem, there are only exactly two cable types, coaxial cable and fibre optic cable, whereby only exactly one of the two types can be used at

each edge. Since the demand of all amplifier points has to be satisfied mandatorily, LAN-Force$_\mathrm{a}$, unlike PC-LAN, is not prize-collecting and there is no overbuilding risk. The amplifier points are already connected to the respective access points by coaxial cables, but the demand of the amplifier points can exceed the capacity of the coaxial cables, so that a fibre roll-out is necessary.
For LAN-Force$_\mathrm{h}$ we assume that the demand of all amplifier points is already satisfied by applying LAN-Force$_\mathrm{a}$. The objective of LAN-Force$_\mathrm{h}$ is to maximise the total profit by connecting actual customers to the respective amplifier points. The profit is composed of the cost of laying cables and the profit of connecting the customers to the network. Customers pay a profit based on the type of connection. Since a fibre connection is more reliable and faster, the profit paid by customers for a fibre connection is greater than for a coaxial connection. As already described for LAN-Force$_\mathrm{a}$, we only distinguish between two cable types. Since the objective is to maximise the total profit, the problem is prize-collecting. However, unlike the PC-LAN problem, there is no overbuilding risk.

Our contributions: In this thesis, we define the two problems LAN-Force$_\mathrm{a}$ and LAN-Force$_\mathrm{h}$. In Chapter 3 we prove the NP-completeness of the problems PC-LAN, LAN-Force$_\mathrm{a}$ and LAN-Force$_\mathrm{h}$. We focus in Chapter 4 on the solution of LAN-Force$_\mathrm{a}$. For this, we describe how we can divide a given planning unit into smaller graphs and how we subsequently delete vertices and edges of these graphs using reduction techniques. Furthermore, we explain two Integer Linear Programs (ILPs) for optimally solving the reduced LAN-Force$_\mathrm{a}$ instances and find a heuristic that approximates an optimal solution. In Chapter 5 we deal in detail with the LAN-Force$_\mathrm{h}$ problem. We adapt the reduction techniques we have found, ILPs and the heuristic for LAN-Force$_\mathrm{a}$ to LAN-Force$_\mathrm{h}$. We then test our methods on real instances provided by Vodafone GmbH.

2 Preliminaries

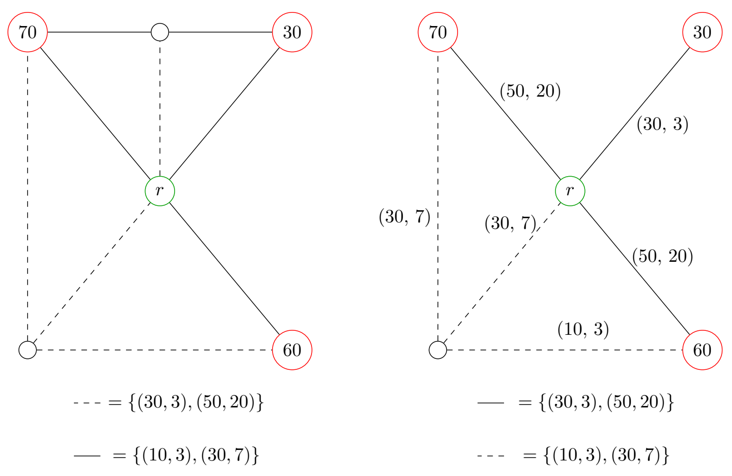
2.1 Problem statements

In the following we define the most important problems for this thesis. This is the PC-LAN problem already known from the literature, introduced by Ljubić et al. in [4]. Then we define the problems LAN-Force$_\mathrm{a}$ and LAN-Force$_\mathrm{h}$, which are new to our knowledge.

2.1.1 PC-LAN

Let $G = (V, E)$ be an undirected graph, where V represents the vertex set and E the edge set. A specific vertex $r \in V$ serves as the root vertex. We associate each vertex

$v \in V \setminus \{r\}$ with a demand d_v and a profit p_v. We can divide the vertices into two disjoint sets $K \subset V, I \subset V$ with $K \cap I = \emptyset$ and $K \cup I = V \setminus \{r\}$. The vertex set I represents street intersections, the vertex set K represents potential customers. We assign to each vertex $v \in I$ a demand and a profit of 0, we refer to these vertices as intersection vertices throughout the report. All vertices $v \in K$, on the other hand, are assigned a positive demand $d_v > 0$ and a positive profit $p_v > 0$. We call these vertices customer vertices. A customer only pays profit p_k if he is connected to an access point and his demand d_k is satisfied.
Each edge $e \in E$ is associated with a number of modules $N_e = \{n_1, n_2, \dots, n_{|N_e|}\}$, where each module $n_i = (u_{e,n_i}, c_{e,n_i})$ consists of the capacity u_{e,n_i} and the cost c_{e,n_i}. Without loss of generality, we can assume that the modules are sorted such that $u_{e,n_i} \leq u_{e,n_{i+1}}$. The individual modules represent the different types of cables that can be used to connect the potential customers to the network. There is an additional condition that we have to collect a minimum customer prize p_0. It is a percentage α of the total possible profit, generated by including all customers to the solution, that is $p_0 = \alpha \sum_{k \in K} p_k$. This condition is called overbuilding risk.



(a) Input PC-LAN (b) Solution PC-LAN

Figure 2: Example input graph for the PC-LAN problem, together with the optimal solution. The green vertex represents the root. Red vertices are customers. Their demands equal their profits and are written at the corresponding labels. There are two different edge types, which are composed of different modules. At each edge, you can choose exactly one module consisting of edge capacity and edge cost. In the solution, the installed modules are next to the respective edge.

A full ILP model for this problem will be given in the Appendix A. The objective of the optimisation problem is to maximise the profit, which is composed of the sum of the profits of all vertices included in the solution reduced by the cost of all installed edge modules in the solution. The mathematically formal optimisation function can thus be expressed by $\max \sum_{k \in K} p_k y_k - \sum_{e \in E} \sum_{n \in N_e} c_{e,n} x_{e,n}$, where $x_{e,n}$ and y_v are binary variables that induce whether the respective module n_e or vertex v is part of the solution. For each edge, a maximum of one module may be

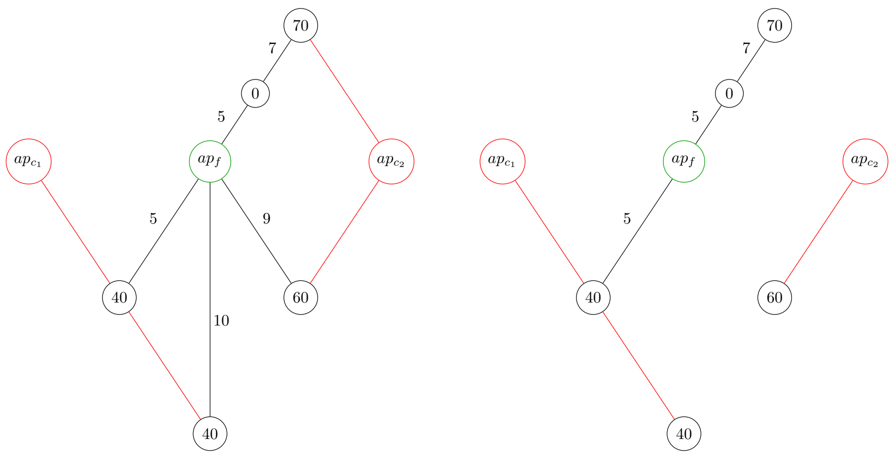
installed. To add a potential customer $v \in K$ to the solution, its specific demand d_v of traffic must be satisfied. The connection from the root r to the individual customers can be seen as a flow problem, where the flow is directed from the root to the vertices in the solution. On each edge the flow must not exceed the capacity of the installed module. When the flow reaches a vertex v connected to the solution, it consumes d_v units of flow. To satisfy the demand of a vertex, we can use the flow of different incident edges. Therefore, different than in other shortest network problems, the result does not necessarily have to be a tree [4]. Figure 2 shows an example instance together with a solution for the PC-LAN problem.
The problem is NP-complete, as we prove in Chapter 3.1.

In our next problem, the LAN-Force$_\mathrm{a}$ problem, there is no choice between different cable types at each edge. Instead, there are only two cable types, with only one of the two cables eligible at each edge. Instead of one root, there are multiple access points. Since the goal of the LAN-Force$_\mathrm{a}$ problem is to supply all amplifier points, the problem is not prize-collecting.

2.1.2 LAN-Force$_\mathrm{a}$

Let $G = (V, E)$ be an undirected graph. There are multiple vertices that can be used to route traffic into the network. These connection vertices $AP \subset V$ are referred to as access points. The set of access points can be divided into fibre access points (AP_f) and coaxial access points (AP_c), where: $AP_f \subset AP, AP_c \subset AP$ with $AP_f \cap AP_c = \emptyset$ and $AP_f \cup AP_c = AP$.
As described for the PC-LAN problem, we can divide the vertices $V \setminus AP$ into two disjoint sets K and I. The vertices $k \in K$ are called amplifier points, they have a positive demand $d_k \in \mathbb{Q}^+$. We assign to all other vertices $i \in I$ a demand of 0.
Each edge has exactly one module ($|N_e| = 1$). For simplicity, we assign exactly one positive capacity u_e with positive cost c_e to each edge $e \in E$. We can divide the edges into two subsets $E_f, E_c \subseteq E$ with $E_f \cap E_c = \emptyset$ and $E_f \cup E_c = E$. We assign to each edge $e \in E_c$ the capacity $u_e = u$ with $u \in \mathbb{N}$ and costs of $c_e = 0$. For each edge $e \in E_f$ we assign the capacity $u_e = \infty$ and the costs $c_e = c$ with $c \in \mathbb{N}$. We interpret the different edge types E_f and E_c as fiber optic cables or coaxial cables respectively.
For each $AP \in AP_c$ there is an already existing coaxial cable network connecting only amplifier vertices $k \in K$ to the AP. This network has a tree structure, where the respective access point is the root of the tree. We call this tree a net cascade. There are no amplifier points k that are not connected to a net cascade. The edges in E_f are street segments where it is possible to lay a fiber optic cable. Laying a fiber optic cable involves costs depending on the length of the street segments.
The goal is to connect every amplifier point $k \in K$ to the network at minimum cost. An amplifier point can only be part of the solution if the traffic reaching it

from an access point is greater than or equal to its demand d_k. For this reason, it is necessary to install fibre optic cables from E_f whenever not enough traffic can be routed to the amplifier points through the edges of the already existing coaxial cable network E_c. Figure 3 shows an example instance of the LAN-Force$_\mathrm{a}$ problem.



(a) Input LAN-Force$_\mathrm{a}$ (b) Solution LAN-Force$_\mathrm{a}$

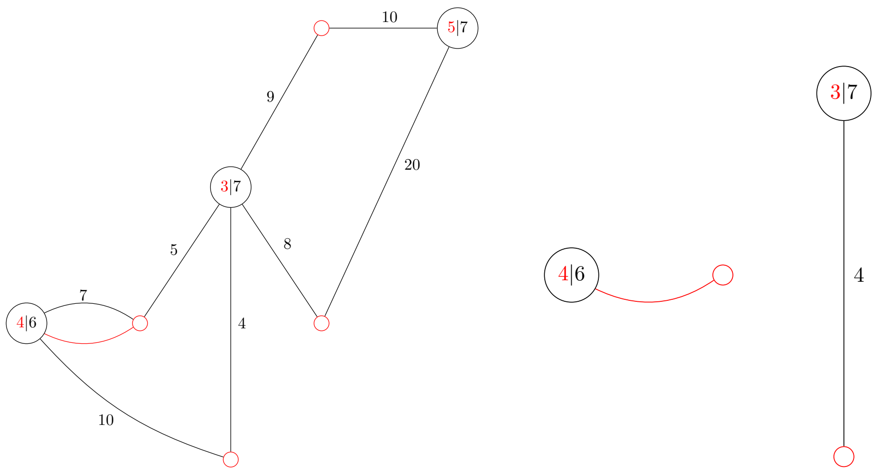
Figure 3: Example input graph for the LAN-Force$_\mathrm{a}$ problem, together with the optimal solution. The red edges represent the net cascade consisting of coaxial cables with a capacity of $u = 64$. The black edges represent fibre edges with their respective costs. In each vertex, we have recorded the respective vertex demand.

On the connection from access point to amplifier point, after using a coaxial cable edge, we are not allowed to use a fibre edge. Overall, we try to route the flow to the customers via the net cascades at no cost, and then connect all customers that could not be connected to the net cascade via coaxial cable as cheaply as possible.
We call this problem Local Access Reinforcement Problem, as the goal is to reinforce the capacity existing through the net cascades with the help of fibre roll-out. We prove the NP-completeness of the problem in Chapter 3.2. A full ILP model for this problem will be given in Chapter 4.3.
The LAN-Force$_\mathrm{a}$ problem is a special case of the PC-LAN problem, since we can transform the LAN-Force$_\mathrm{a}$ problem into the PC-LAN problem. For this, we create an artificial root by connecting a new vertex r to all $a \in AP$ by an edge $e_{ra} = \{r, a\}$. These new edges have costs of 0 and capacities of infinity. Furthermore, for each edge e we create a module $n_e = (u_e, c_e)$ from the capacity u_e and the cost c_e. The result is a PC-LAN instance where each edge has exactly one module.

2.1.3 LAN-Force$_\mathrm{h}$

The LAN-Force$_\mathrm{h}$ problem is a variant of the PC-LAN problem, which we can derive from the LAN-Force$_\mathrm{a}$ problem. The goal of the LAN-Force$_\mathrm{h}$ problem is to connect paying customers to the amplifier points as profitable as possible. The amplifier points were already supplied as part of the LAN-Force$_\mathrm{a}$ problem and thus serve as root vertices in the LAN-Force$_\mathrm{h}$ problem. Below we give a formally definition of the LAN-Force$_\mathrm{h}$ problem.

Let $G = (V, E)$ be an undirected graph. There are multiple vertices called amplifier vertices $V_{AMP} \in V$ to route traffic into the network. We can further divide the vertices of the graph $V \setminus AMP$ into two disjoint vertex sets K and I. The vertices $k \in K$ represent paying customers. They have two different positive profits $p_{k_c} > 0$ and $p_{k_f} > 0$ and a positive demand $d_k > 0$. We assign to all other vertices $i \in I$ a profit and a demand of 0. The edges of the LAN-Force$_\mathrm{h}$ problem are defined analogously to the edges of the LAN-Force$_\mathrm{a}$ problem. Unlike the LAN-Force$_\mathrm{a}$ problem, the coaxial cables do not occur in a tree structure around the individual amplifier points. Instead, there are selected clients that have a direct coaxial cable connection to an amplifier point. Figure 4 shows an example instance.



(a) Input LAN-Force$_\mathrm{h}$ (b) Solution LAN-Force$_\mathrm{h}$

Figure 4: Example input graph for the LAN-Force$_\mathrm{h}$ problem, together with the optimal solution. The red vertices are amplifier points. The red edges represent coaxial cables with a capacity of $u = 64$. The black edges represent fibre edges with their respective costs. The coax and fibre profit are in first and second place respectively as labels in the customers. The demand corresponds to the coax profit.

The goal of the LAN-Force$_\mathrm{h}$ problem is to maximise the total profit. The profit consists of the profit of connecting customers reduced by the cost of cable expansion.

Customers pay a profit according to the nature of their cable connection. If there is a coaxial cable edge on the path of connectivity, customer k pays profit p_{k_c}. If the path consists entirely of fibre, the customer pays a profit of p_{k_f}. A customer only pays the profit if its demand d_k is fully satisfied. Since, analogous to the LAN-Force$_{\text{a}}$ problem, no fibre edge may follow a coaxial edge on the root-to-customer connection, we only need to consider the last edge of the connection for the nature of the customer's profit. A customer may never be connected to the network via both coaxial and fibre cables. We prove the NP-completeness of the problem in Chapter 3.3. A full ILP model for this problem will be given in the Appendix B.
The LAN-Force$_{\text{h}}$ problem is a variant and not a special case of the PC-LAN problem, since it is not possible to transform the LAN-Force$_{\text{h}}$ problem into the PC-LAN problem. It is not possible to model the different customer profits p_{k_f} and p_{k_c} according to the PC-LAN problem.

2.2 Terms and notation

In this section we explain some basic terms. Both the LAN-Force$_{\text{a}}$ problem and the LAN-Force$_{\text{h}}$ problem have, instead of a single root vertex r, a set of access points AP that supply the network with traffic. However, some techniques we describe require an explicit root vertex. Whenever we talk about a root vertex r in the context of the LAN-Force$_{\text{a}}$ or LAN-Force$_{\text{h}}$ problem, we create an artificial root by connecting the new vertex r to all $a \in AP$ by an edge $e_{ra} = \{r, a\}$. The new edge has cost $c_{ra} = 0$ and capacity $u_{ra} = \infty$. With the help of this artificial root, it is possible to supply all access points with enough traffic for free.
When we speak of a graph without a net cascade G_f, we mean the graph that results from G when we delete all coaxial cable edges $e \in E_c$. After applying the techniques that require G_f as input, we can add the coaxial cable edges back in.
We often use the Steiner tree problem (STP) [5] and the prize-collecting Steiner tree problem (PCSTP) [6]. The input of the STP is a graph $G = (V, E)$. All edges $e \in E$ have associated costs $c_e \in \mathbb{Q}^+$. The vertices are divided into so-called terminal vertices $T \subset V$ and non-terminal vertices $I \subset V \setminus T$. All vertices of the subset $T \subset V$ must be included in the solution. The goal of the STP is to connect all terminal vertices $t \in T$ with each other via edges and thereby minimise the cost of all edges in the solution S. The PCSTP is an extension of the STP. In this problem, each terminal vertex $t \in T$ is associated with a positive profit $p_t \in \mathbb{Q}^+$. The terminal vertices do not necessarily have to be part of the solution, instead the objective is to maximise the total profit consisting of all customer profits reduced by all edge costs in the solution ($\max \sum_{t \in S} p_t - \sum_{e \in S} c_e$). When speaking of the rooted version of the problem (RSTP or RPCSTP), there is a designated root vertex r which must be part of the solution.
In an undirected graph, we denote the set of all incident edges of a vertex $v \in V$ by $\delta(v)$. In the directed case, we make a distinction between the incoming edges $\delta^-(v)$

and the outgoing edges $\delta^+(v)$. Similarly, we define the set of all edges incident to a vertex set $V' \subset V$ with $\delta(V')$. In the directed case we denote the edges entering V' with $\delta^-(V')$ and the outgoing edges with $\delta^+(V')$.
We denote the length of the shortest path between the two vertices $v_i \in V$ and $v_j \in V$ by $d(v_i, v_j)$. If we do not specify anything extra, we use the Dijkstra method for calculation of the shortest path [7], using the edge costs as underlying costs.
For simplicity, we define branches of a tree. Given a rooted tree $T = (V, E)$ with root r. If we delete each edge incident to the root vertex $e_{r,v} \in \delta(r)$, we call all resulting subgraphs except the root vertex itself a branch of the tree T. Figure 5 shows an example.

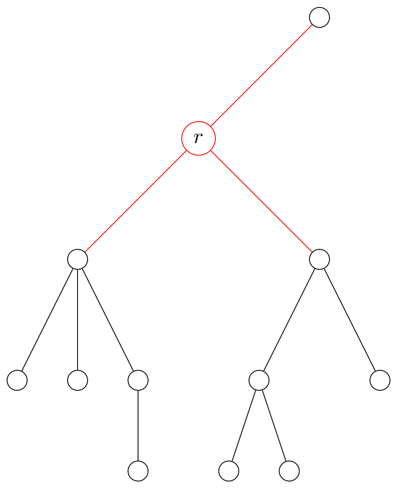


Figure 5: Branches of a tree: This tree has three branches because the root is connected to three subgraphs. The respective branches are the black subgraphs.

3 Proof of NP-completeness

In the following we prove the NP-completeness of the problems PC-LAN, LAN-Force$_\mathrm{a}$ and LAN-Force$_\mathrm{h}$. A problem is NP-complete if it is in NP and is NP-hard. For our proofs we first define the decision variant of the respective problem. Then we prove that the problem is in NP by showing that we can verify a solution instance of the problem in polynomial time. Finally we show that the problem is NP-hard by transforming each instance of an already known NP-complete problem in polynomial time into an instance of the problem to be proved and give an equivalence proof.

3.1 PC-LAN

The decision variant of the PC-LAN problem is as follows:

Given an instance of the PC-LAN problem and a parameter $k \in \mathbb{Q}^+$. Is there a vertex set $V' \subset V$, an edge set $E' \subset E$ and for each edge a module n_i such that all the constraints of the PC-LAN problem are satisfied and $\sum\limits_{v \in V'} p(v) - \sum\limits_{e \in E'} c_{e,n_i} \geq k$?

We want to show that PC-LAN is NP-complete. To do this, we will show that it is in NP and that it is NP-hard.

PC-LAN is in NP: We show that we can verify a solution in polynomial time. Given a set of vertices $V' \subset V$ and edges $E' \subset E$ together with the respective installed modules $n_i \; \forall e \in E'$, which is a solution. To check whether the total profit is greater than k, we must sum up the profits of all vertices in V' and subtract the sum of all installed module costs. This can be done in $\mathcal{O}(V+E)$. To check whether the solution is connected, we apply a depth-first search in $\mathcal{O}(V+E)$.
To test whether the demand of each vertex in the solution is satisfied, we create an artificial sink vertex t. We connect each vertex $v \in V'$ to t, giving the respective edge a capacity of d_v. We then apply the Edmonds–Karp algorithm [8] from source r to sink t on the resulting graph. The Edmonds–Karp algorithm has a runtime of $\mathcal{O}(VE^2)$. Since we can check the given solution in polynomial time, PC-LAN is in NP.

PC-LAN is NP-hard: We show that PC-LAN is at least as hard as the NP-complete problem RPCSTP by reducing RPCSTP to PC-LAN. Let I be an instance of RPCSTP consisting of a graph $G = (V, E)$ with a root vertex $r \in V$. Each vertex $v \in V$ is assigned a profit $p_v \geq 0$, each edge $e \in E$ is assigned a cost $c_e \geq 0$. We define the instance I' for the PC-LAN problem as follows: We copy the graph G together with the root vertex r. Each vertex $v \in V$ keeps its profit p_v. In addition, each vertex is assigned a demand $d_v = 0$. Each edge $e \in E$ with cost c_e is assigned a module $n_0 = (0, c_e)$. Since there is no overbuilding risk in the RPCSTP, we set $p_0 = 0$. This reformulation of instance I into instance I' is in $\mathcal{O}(V+E)$ and thus obviously in polynomial time. Finally, it must be shown that the RPCSTP with instance I and the PC-LAN problem with instance I' are equivalent.

'$\Rightarrow$': Assuming I is a 'Yes'-instance of the RPCSTP, we need to show that I' is a 'Yes'-instance of the PC-LAN problem. Since I is a 'Yes' instance, there is a vertex set $V' \subset V$ and an edge set $E' \subset E$ connected to the root for which $\sum_{v \in V'} p_v - \sum_{e \in E'} c_e \geq k$ holds. In I', the vertex profits and edge costs are identical to those in I. Since each vertex v has a demand of 0, it is possible to connect any vertex connected by edges to the network and take its profit.

'$\Leftarrow$': Assuming I' is a 'Yes'-instance of the PC-LAN problem, we need to show that I is a 'Yes'-instance of RPCSTP. There is a vertex set $V' \subset V$ and an edge set $E' \subset E$ which is connected to the root and for which holds: $\sum\limits_{v \in V'} p(v) - \sum\limits_{e \in E'} c_e, n_0 \geq k$. Furthermore, for each vertex $v \in V'$, its demand d_v is satisfied. The sets V' and E' also form an optimal solution in I, since the underlying graph consists of the same vertices and edges with the same weights.

We have now shown that PC-LAN lies in NP and is NP-hard. It follows that PC-LAN is NP-complete.

3.2 LAN-Force$_\mathrm{a}$

The decision variant of the LAN-Force$_\mathrm{a}$ problem is as follows:

Given an instance of the LAN-Force$_\mathrm{a}$ problem and a parameter $k \in \mathbb{Q}^+$. Is there an edge set $E' \subset E$ such that all the constraints of the LAN-Force$_\mathrm{a}$ problem are satisfied and $\sum\limits_{e \in E'} c_e \leq k$?

LAN-Force$_\mathrm{a}$ is in NP: We show how to verify a solution in polynomial time. Let $E' \subset E$ be a solution for the LAN-Force$_\mathrm{a}$ problem. To check whether the total cost is smaller than k, we sum up the costs of all edges in E' and compare the sum to k. This can be done in $\mathcal{O}(E)$.
To verify that the solution is connected and that the demands d_v of the vertices $v \in V'$ are satisfied, we perform a depth-first search analogous to the proof of the PC-LAN problem, followed by execution of the Edmonds–Karp algorithm in $\mathcal{O}(V + E + VE^2)$.

LAN-Force$_\mathrm{a}$ is NP-hard: We show that LAN-Force$_\mathrm{a}$ is NP-hard by using the RSTP for the reduction. Let I be an instance of RSTP consisting of a graph $G = (V, E)$ with a root vertex $r \in V$. Each edge $e \in E$ is assigned a cost $c_e \geq 0$. We define the instance I' for the LAN-Force$_\mathrm{a}$ problem as follows:
We copy the graph G together with the root vertex r. The set of fibre access points contains only the root vertex $AP_f = \{r\}$, the set of coax access points is empty $AP_c = 0$. We assign to each vertex a demand $d_v = 0$. The terminal vertices T of the RSTP become the amplifier points K of the LAN-Force$_\mathrm{a}$ problem. We assign to each edge $e \in E$ with cost c_e a a capacity of infinity. The resulting instance is

an instance of the LAN-Force$_\mathrm{a}$ problem with only a single access point and no net cascade. The whole reformulation of instance I into instance I' is in $\mathcal{O}(V+E)$ and thus obviously in polynomial time. Finally we have to show that the RSTP with instance I and the LAN-Force$_\mathrm{a}$ problem with instance I' are equivalent.

'$\Rightarrow$': Assuming I is a 'Yes'-instance of the RSTP, we need to show that I' is a 'Yes'-instance of the LAN-Force$_\mathrm{a}$ problem. Since I is a 'Yes'-instance, there is a solution $E' \in E$ for which $\sum\limits_{e \in E'} c_e \leq k$ holds. In I' the edge costs are identical to those in I. Since each vertex v has a demand of 0, it is possible to connect any vertex to the network.

'$\Leftarrow$': Assuming I' is a 'Yes'-instance of the LAN-Force$_\mathrm{a}$ problem, we need to show that I is a 'Yes'-instance of RSTP. There is a solution $E' \in E$ for I' for which holds: $\sum\limits_{e \in E'} c_e \leq k$. Furthermore, for each vertex v in the solution, its demand d_v is satisfied. The set E' also forms an optimal solution in I, since the underlying graph consists of the same vertices and edges with the same weights. Since the demands of every vertex are 0 the optimal solution has a tree structure.

3.3 LAN-Force$_\mathrm{h}$

The decision variant of the LAN-Force$_\mathrm{h}$ problem is as follows:

Given an instance of the LAN-Force$_\mathrm{h}$ problem and a parameter $k \in \mathbb{N}$. Is there a vertex set $V' \subset V$ and an edge set $E' \subset E$ such that all the constraints of the LAN-Force$_\mathrm{h}$ problem are satisfied and $\sum_{k \in K} \sum_{e_f \in \delta(k) \subset E'_f} p_{k_f} + \sum_{k \in K} \sum_{e_c \in \delta(k) \subset E'_c} p_{k_c} - \sum_{e \in E'} c_e \geq k$?

LAN-Force$_\mathrm{h}$ is in NP: We show how to verify a solution in polynomial time. Let a set of vertices $V' \subset V$ and edges $E' \subset E$ ve a solution for the LAN-Force$_\mathrm{h}$ problem. To check whether the total profit is bigger than k, we must sum up all profits of the customers according to their connection and subtract the costs of all edges in E'. This can be done in $\mathcal{O}(V+E)$.
To verify that the solution is connected and that the demands d_v of the vertices $v \in V'$ are satisfied, we perform a depth-first search analogous to the proof of the PC-LAN problem, followed by execution of the Edmonds–Karp algorithm in $\mathcal{O}(V+E+VE^2)$.

LAN-Force$_\mathrm{h}$ is NP-hard: We show that LAN-Force$_\mathrm{h}$ is NP-hard by using the

RPCSTP for the reduction. Let I be an instance of RPCSTP consisting of a graph $G = (V, E)$ with a root vertex $r \in V$. Each vertex $v \in V$ is assigned a profit $p_v \geq 0$, each edge $e \in E$ is assigned a cost $c_e \geq 0$. We define the instance I' for the LAN-Force$_\mathrm{h}$ problem as follows:
We copy the graph G together with the root vertex r. The set of fibre access points contains only the root vertex $AP_f = \{r\}$, the set of coax access points is empty $AP_c = 0$. Each vertex $v \in I$ keeps its profit p_v. We assign to each vertex $v \in K$ the two profits $p_f = p_v$ and $p_c = p_v$ and to each vertex $v \in V$ a demand $d_v = 0$. In addition we assign to each edge $e \in E$ with cost c_e a capacity $u_e = \infty$. The resulting instance is a single access point instance of the LAN-Force$_\mathrm{h}$ problem, with customers paying the same for a fibre or coax connection. The whole reformulation of instance I into instance I' is in $\mathcal{O}(V + E)$ and thus obviously in polynomial time. Finally, we have to show that the RPCSTP with instance I and the LAN-Force$_\mathrm{h}$ problem with instance I' are equivalent.

'$\Rightarrow$': Assuming I is a 'Yes'-instance of the RPCSTP, we need to show that I' is a 'Yes'-instance of the LAN-Force$_\mathrm{h}$ problem. Since I is a 'Yes'-instance, there is a vertex set $V' \subset V$ and an edge set $E' \subset E$ connected to the root for which $\sum_{v \in V'} p_v - \sum_{e \in E'} c_e \geq k$ holds. In I', the vertex profits and edge costs are identical to those in I. Since each vertex v has a demand of 0, it is possible to connect any vertex connected by edges to the network and take its profit.

'$\Leftarrow$': Assuming I' is a 'Yes'-instance of the LAN-Force$_\mathrm{h}$ problem, we need to show that I is a 'Yes'-instance of RPCSTP. So there is a vertex set $V' \subset V$ and an edge set $E' \subset E$ which is connected to r and for which holds: $\sum_{k \in K} \sum_{e_f \in \delta(k) \subset E'_f} p_{k_f} + \sum_{k \in K} \sum_{e_c \in \delta(k) \subset E'_c} p_{k_c} - \sum_{e \in E'} c_e \geq k$. Furthermore, for each vertex $v \in V'$, its demand d_v is satisfied. The sets V' and E' also form an optimal solution in I, since the underlying graph consists of the same vertices and edges with the same weights. Since the demands of every vertex are 0 the optimal solution is a tree.

4 LAN-Force$_\mathrm{a}$

In this chapter we describe an algorithmic process to find an optimal and an approximate solution for the LAN-Force$_\mathrm{a}$ problem for a given input graph. Since the input graphs can be very large, we deal with a division of the graph into smaller subgraphs in Chapter 4.1. We can solve the different subgraphs separately. Then, in Chapter 4.2 we describe different reduction techniques to delete vertices and edges in the respective subgraphs and thus further reduce the size of the instance. We then discuss two different Integer Linear Programs (ILPs) to find an optimal solution on the graphs. Finally, we describe a heuristic that makes it possible to split the input graph into smaller subgraphs and then approximate an optimal solution.

4.1 Focus on net cascades

One objective of the LAN-Force$_\mathrm{a}$ problem is to satisfy the demand d_k of all the amplifier points $k \in K$ in the graph. We can satisfy the demand by connecting the respective amplifier point to an access point, whereby the edges of the path need to have sufficient capacity. We describe in Chapter 4.1.1 the focusing on individual net cascades to reduce the input graph. We create a subgraph G_{sub} for each net cascade. The resulting subgraph G_{sub} contains the respective net cascade along with any fibre cables and access points that may be required to optimally serve the net cascade.
The resulting subgraphs provide a good starting point for further reductions. However, by focusing on individual net cascades, we may lose optimal solutions, as it may be cheaper to connect two net cascades via shared fibre access points than via their own. For this reason, we describe in Chapter 4.1.2 how we can algorithmically find a union of related net cascades so that we do not lose optimal solutions.

4.1.1 Focus on single net cascades

For each net cascade in the input graph G we compute a separate subgraph G_{sub}. In addition to the net cascade, G_{sub} contains all the fibre cables $E' \subseteq E_f$ and fibre access points $AP' \subseteq AP_f$ potentially required in an optimal solution. We compute an upper bound ub of the optimal solution to find out which edges E' and access points AP' could be required for an optimal solution. This upper bound indicates the *maximal* cost of an optimal solution satisfying the demands of all the amplifier points in the net cascade. Afterwards we search for all reachable vertices within a maximum distance of ub using a breadth-first search starting from each amplifier point. All fibre cables, fibre access points and intersection vertices that lie within this radius together with the net cascade form G_{sub}. In the following, we explain different methods to algorithmically determine such an upper bound.

Shortest path from every vertex (UB_1): In the worst case each amplifier point must be connected individually via a shortest path to a fibre access point. The upper bound thus corresponds to the sum of all shortest paths over fibre edges from all amplifier points to their next fibre access point. We only consider shortest paths in the graph without net cascades G_f, as we cannot guarantee that we can sufficiently supply the amplifier points due to the limited capacity of the coaxial cables.
This upper bound can be further improved by adding the cost of edges that occur

multiple times on shortest paths to ub only once. Furthermore, we can supply all the amplifier points of a branch of the net cascade for free if their accumulated demand does not exceed the capacity of the coaxial cables. In this case, it is not necessary to add up the shortest paths of these amplifier points to ub. Finally, we consider the case where some but not all amplifier points of a branch can be supplied by the net cascade. In this case, it is advantageous to supply those amplifier points via the net cascade that would incur the greatest costs through a connection to fibre. For this reason, we calculate the shortest path to the nearest fibre access point for all amplifier points of this branch. We sort the vertices in descending order of the length of the shortest paths and add the shortest paths to ub only until the cumulative demand of the remaining vertices is less than the coaxial cable capacity.
Figure 6 shows an example. In the example we can disregard vertex 1 in the calculation of ub because its demand can be completely satisfied by the net cascade. Vertex 3 and vertex 4 are on a branch of the net cascade and have a cumulative demand of 80. Since the capacity of the coaxial cables is 64, the net cascade cannot supply both vertices together. Since $d(3, ap_{f_2}) = 12$ and $d(4, ap_{f_2}) = 10$, we only need to add the shortest path from vertex 4 to the next fibre access point to ub. The edge $(6, ap_{f_2})$ lies on both the shortest path $d(5, ap_{f_2})$ and $d(4, ap_{f_2})$ and thus only needs to be included once. This results in an upper bound of $ub = d(2, ap_{f_1}) + d(4, ap_{f_2}) + d(5, ap_{f_2}) - d(6, ap_{f_2}) = 5 + 10 + 15 - 5 = 25$.

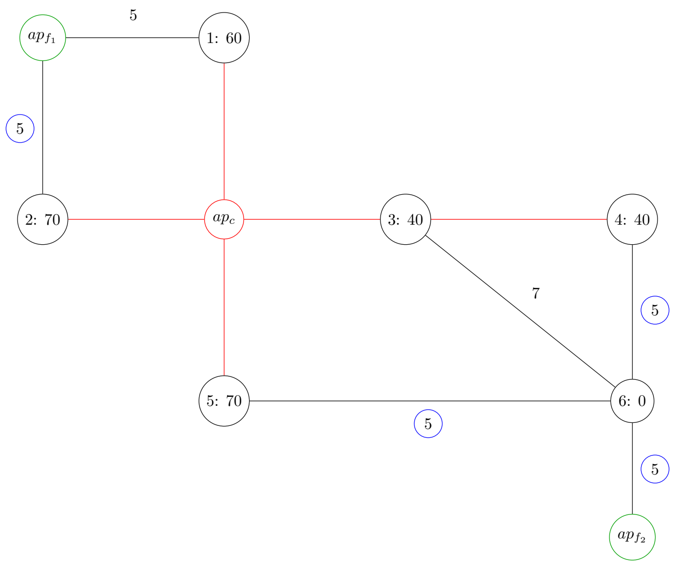


Figure 6: UB_1**:** The red edges represent the net cascade consisting of coaxial cables with a capacity of $u = 64$. The black edges represent fibre edges with their respective costs. In each vertex $v \in V \setminus AP$, we have recorded the vertex ID along with the respective vertex demand. We use all fiber edges whose costs are annotated with a blue circle to calculate the upper bound.

The algorithm UB_1 finds very poor upper bounds when the distance between amplifier points and fibre access points is very large, since in the worst case very long edge-disjoint paths are found from each amplifier point to the next fibre access point. As a result, ub becomes very large, resulting in a very large subgraph. For this reason, we explain below another method for finding an upper bound that addresses this problem.

Shortest path from one vertex (UB_2): As described for UB_1, we must connect all amplifier points to a fibre access point via fibre cables in order to calculate an upper bound. Instead of connecting each amplifier point individually to the next fibre access point, we only add the distance of the amplifier point k with the shortest path to the next fibre access point to the upper bound. We connect all other amplifier points to the same fibre access point via shortest paths to k.
We can apply the same improvements as described in UB_1: Edges that occur in multiple shortest paths do not need to be added to ub multiple times. In addition, we do not have to add the distances of the amplifier points that can be supplied by the net cascade to ub. Finally, on the shortest paths, we have to add up multiple edges to ub only once.
We calculate an example upper bound using the graph in Figure 7. First, we calculate all the shortest paths from each amplifier point to its nearest fibre access point. The shortest distance of 20 is between vertex 4 and the access point ap_{f_4}. We need to connect all further amplifier points to vertex 4 via shortest paths. This results in an upper bound of $ub = d(4, ap_{f_4}) + d(1, 4) + d(2, 4) + d(3, 4) - d(3, 4) = 20 + 2 + 2 + 4 - 2 = 26$. Note that this upper bound is far smaller than the bound calculated with UB_1, which is $ub = d(1, ap_{f_1}) + d(2, ap_{f_2}) + d(3, ap_{f_3}) + d(4, ap_{f_4}) = 21 + 21 + 21 + 20 = 83$.

Pcst-fast (UB_3): For this upper bound computation, we use the heuristic pcst-fast [9]. Pcst-fast determines an upper bound for the RPCSTP.
To use pcst-fast we need to create a RPCSTP instance from the given LAN-Force$_\text{a}$ instance. For this we use the graph without net cascades G_f. The remaining edges of the graph are then exclusively fibre edges with a capacity of ∞. Since the network flow does not play a role in RPCSTP, we can disregard the demand of the amplifier points and the capacity of the edges. We temporarily introduce profits for all amplifier points which we set to ∞ to guarantee that pcst-fast includes all amplifier points in the solution,. The resulting graph is then suitable for the application of pcst-fast. Furthermore we define a root vertex r. As already described for the calculation of UB_1 and UB_2, the amplifier points that can be supplied by the net cascade do not have to contribute to the calculation of the upper bound. We set the prize of all these amplifier points to 0 so that the cost of the upper bound is not driven up by the connection of these amplifier points.

Depending on the input instance, the goodness of the upper bounds calculated by

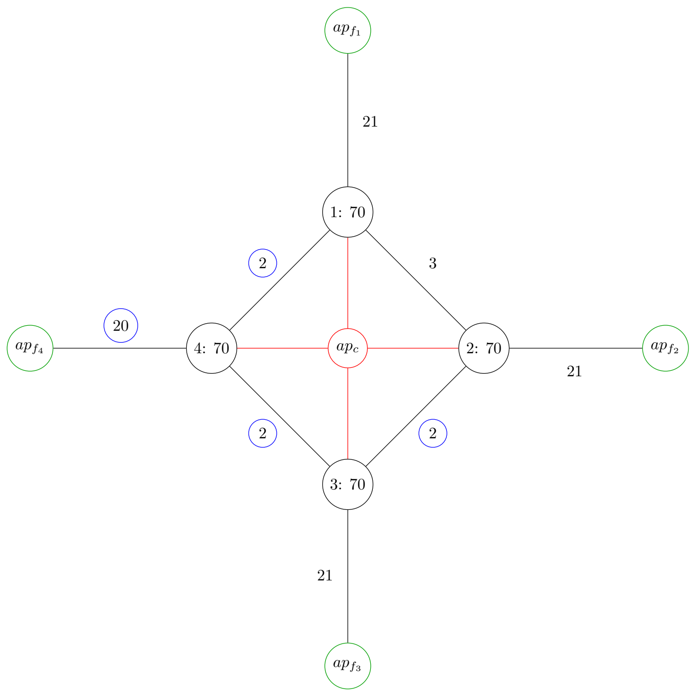


Figure 7: UB_2**:** The red edges represent the net cascade consisting of coaxial cables with a capacity of $u = 64$. The black edges represent fibre edges with their respective costs. In each vertex $v \in V \setminus AP$, we have recorded the vertex ID along with the respective vertex demand. We use all fiber edges whose costs are annotated with a blue circle to calculate the upper bound.

the methods UB_1, UB_2 and UB_3 varies. For this reason, we use all three methods in succession and then select the smallest of the calculated upper bounds. This upper bound ub serves as the radius, which indicates the maximum distance of the breadth-first search from each amplifier point of the net cascade.
A vertex $i \in I$ in G_{sub} can only be part of an optimal solution if the sum of its distance to the nearest fibre access point and its distance to the nearest amplifier point in G_{sub} is less than or equal to the upper bound ub. It is thus possible to further reduce G_{sub} by deleting every vertex i for which this is not true.

4.1.2 Merging net cascades

By focusing on individual net cascades, it is possible that we lose optimal solutions. Figure 8 shows an example. In the optimal case we supply the two amplifier points 1 and 3 by the fibre access point ap_{f_3} at a cost of 8. However, focusing on the two net cascades individually, we calculate an upper bound of $ub = 5$ for each. The fibre access point ap_{f_2} is not included in the respective subgraphs so we calculate a total cost of 10. To avoid this, we describe below a method to form clusters of coherent net cascades in order to focus on the clusters afterwards and not lose optimal solutions.

First we calculate the upper bound ub for each net cascade using the method described in Chapter 4.1.1. Then we determine the graph G_{sub} for every net cascade using a breadth-first search starting from each amplifier point. If the subgraphs of two net cascades G_{sub_1} and G_{sub_2} are not vertex-disjoint, it is possible that both net cascades are supplied together in an optimal solution. For this reason we merge all net cascades with overlapping subgraphs into a single net cascade. We create a new access point, which is connected to all net cascade access points to be merged. The connecting edges between the new net cascade access point and the old net cascade access points have a capacity of ∞ and a cost of 0.
Afterwards it is possible to find a subgraph G_{sub} for this new net cascade using the method described in Chapter 4.1.1. In the example from Figure 8, vertex 2 is contained in both the subgraph of ap_{c_1} and the subgraph of ap_{c_2}. Since the two subgraphs are not vertex disjoint, we merge the two net cascades into a common net cascade.

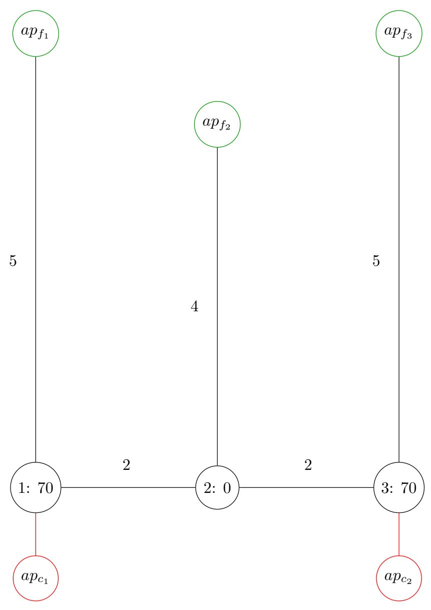


Figure 8: The red edges represent the net cascade consisting of coaxial cables with a capacity of $u = 64$. The black edges represent fibre edges with their respective costs. In the non-amplifier points, the respective vertex ID can be seen together with the vertex edge.

4.2 Reduction techniques

In this chapter we describe a variety of reduction techniques to reduce a given LAN-Force$_a$ instance without losing the optimal solution. Most of the reduction techniques described were originally developed for the reduction of the (R)PCSTP. We describe how to adapt the respective reduction techniques in order to use them for the LAN-Force$_\mathrm{a}$ problem as well. Furthermore, we explain our own reduction technique Collapse net cascades.

4.2.1 Basic Reductions

The Basic Reductions for the PCSTP originally come from [10] and are an effective method to reduce a given LAN-Force$_\mathrm{a}$ instance in linear time. From the original five different reductions, we were able to reformulate three for the LAN-Force$_\mathrm{a}$ problem. While the first two reduction rules focus on deleting intersection vertices, the third rule attempts to delete unconnected components.

Proposition 1 (Adapted from *NTD$_1$* in [10])**.** *A vertex $v \in I$ of degree 1 and its incident edge $\{u, v\}$ can be removed.*

A vertex $v \in I$ must connect at least two terminals to be part of the optimal solution, since it does not contribute to the profit itself. If v has vertex degree 1, it is impossible for it to connect two terminals and thus cannot be part of the optimal solution.

Proposition 2 (Adapted from *NTD$_2$* in [10])**.** *A vertex $v \in I$ of degree 2 and its incident edges $\{u, v\}$, $\{v, w\}$ can be substituted by a single edge $\{u, w\}$ with $c_{\{u,w\}} = c_{\{u,v\}} + c_{\{v,w\}}$ and $u_{\{u,w\}} = \min(u_{\{u,v\}} + u_{\{v,w\}})$. In case of two parallel edges, the one of lowest cost is retained.*

Each intersection must be connected to at least two terminals as explained earlier. If an intersection has degree 2 and is part of the optimal solution, then both incident edges must also be part of the optimal solution. Since only the edges increase costs, we can replace all components by a single edge.

Proposition 3 (*UDV* in [10])**.** *Each vertex $v \in V$ that is not connected to the root vertex r can be deleted along with all incident edges.*

With this reduction rule it is possible to delete all components which do not belong to the optimal solution.

4.2.2 Collapse net cascades

The goal of the reduction technique Collapse net cascade is to reduce the number of amplifier points of the graph:

Proposition 4 (Collapse net cascades)**.** *Let $G' = (V', E')$ be a branch of a net cascade such that $\sum_{v \in V'} d_v \leq u$ holds. All vertices in V' are replaced by the new vertex v' with $d_{v'} = \sum_{v \in V'} d_v$. All edges $e = \{j, k\} \in E$ connected to a vertex $k \in V'$ can be replaced by the edge $e' = \{j, v'\}$ with $c_{e'} = c_e$ and $u_{e'} = u_e$.*

We can reduce all vertices of a branch of a net cascade, which obviously can be supplied by the net cascade for free, to a single vertex v'. It is not possible for us to delete the entire branch, as we may need to be able to route traffic over amplifier points to amplifier points in other branches.

4.2.3 Least Cost

Proposition 5 (Least Cost [11])**.** *In the graph without net cascade $G_f = (V', E')$, an edge $e_{uv} = \{u, v\} \in E'$ can be deleted if $d(u, v) < c_{uv}$.*

Since all edges in G_f have infinite capacity, it is sufficient to make a single connection from an access point to an amplifier point to satisfy the demand of the amplifier point. Thus, if there is a cheaper connection between vertices u and v, it can be chosen. The reduction technique only works on the graph without net cascade G_f, as Figure 9 shows. In the example we cannot delete the edge $e = \{2, 3\}$ even though $d(2, 3) = 3 < 5 = c_e$, because in this case we could not satisfy the demand of vertex 2.

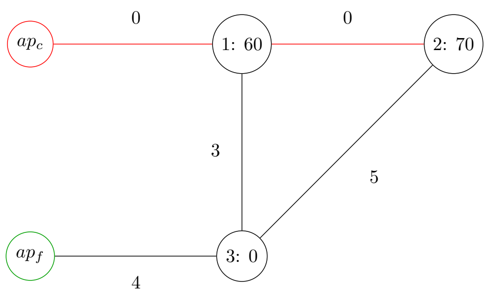


Figure 9: The red edges represent the net cascade consisting of coaxial cables with a capacity of $u = 64$. The black edges represent fibre edges with their respective costs.

4.2.4 Dual Ascent

Dual Ascent (DA) is a complex reduction technique that reduces a given input graph $G = (V, E)$ using upper and lower bounds. DA originates from [12] and was originally developed for the RPCSTP. We were able to reformulate the reduction technique for the RSTP. Since the graph without net cascade G_f with artificial root r is very similar to the RSTP, it is possible to apply DA to the LAN-Force$_a$

problem as well, although we need to adapt the reduction technique to the LAN-Force$_a$ problem. In this chapter, we first describe the DA for the RSTP. We then explain the cases in which DA for the RSTP may delete optimal solutions of the LAN-Force$_a$ problem. We describe our found adaptations of DA for the LAN-Force$_a$ problem and prove their correctness.

Dual Ascent for RSTP

Preprocessing DA only works on directed graphs. For this reason, we transform the given input instance $G = (V, E)$ into a directed instance $G' = (V, A)$ by translating each edge $e_{ij} = \{i, j\} \in E$ with a cost of $c_{e_{ij}}$ into edges $e_{ij} = (i, j)$ and $e_{ji} = (j, i)$ with respective costs of $c_{(i,j)} = c_{(j,i)} = c_{e_{ij}}$. For the algorithm we assume that the terminals have only one incoming edge and no outgoing edge. For this we create a new vertex v_i for each terminal $t_i \in T$. We connect every vertex v_i to the respective vertex t_i via an incoming edge $e_{t_i v_i} = (t_i, v_i)$ with a cost of 0. We set the profit of v_i to p_{t_i} and the profit of t_i to 0. The resulting graph satisfies the conditions required for DA and can be easily transformed back after applying the reduction rules.

Finding a lower bound The goal of DA is to delete vertices and edges of the input instance with the help of upper and lower bounds. We use the heuristic pcst-fast [9] for finding an upper bound. We use the routine from Algorithm 1 to calculate the general lower bound LB.

Algorithm 1: Dual ascent routine for lower bound

```
   Data: RSTP instance $G = (V, A), c, p, T, r$
   Result: Lower bound $LB$, reduced costs $\tilde{c}$
1  $LB = 0$
2  $\tilde{c}_{ij} = c_{ij}$                              $\forall (i, j) \in A$
3  $T_a = T \setminus r$
4  while $T_a \neq \emptyset$ do
5      $t = chooseActiveTerminal(T_a)$
6      $W = W(t)$                                   \\found with BFS
7      $\Delta = min_{(i,j) \in \delta^-(W)} \tilde{c}_{ij}$
8      $\tilde{c}_{ij} = \tilde{c}_{ij} - \Delta$                   $\forall (i, j) \in \delta^-(W)$
9      $LB = LB + \Delta$
10 return $LB$, $\tilde{c}$
```

We show an example of how the algorithm works in the Appendix. We define some basic terms beforehand to better understand the algorithm. In addition to the normal cost $c_{(i,j)}$ of an edge (i, j), we introduce reduced costs $\tilde{c}_{(i,j)}$. The saturation

graph $G_S \subseteq G$ contains all edges from G whose reduced costs are zero. We describe a path between two vertices i, j on a graph G by $P_H(i, j)$. By active terminals we mean all terminals which we cannot reach from the root in the saturation graph $T_a = \{t \in T \setminus r : \nexists P_{G_s}(r, k)$. An active component $W(t)$ is rooted at an active terminal t. This is the set of vertices that can be reached from t in the saturation graph $(W(t) = \{i \in V : \exists P_{G_s}(i, t)\})$.
Algorithm 1 is a greedy algorithm. At the beginning in lines 1 - 3 we initialise lower bound, reduced costs and the set of active terminals. The reduced costs correspond to the original costs of each edge.
The idea of the DA algorithm is to iteratively try to connect the terminals to the root in the saturation graph. For this purpose, we select in line 5 an active terminal in each iteration of the main loop. Leitner et al. propose in [12] an algorithm for the choice of the active terminal. Due to time constraints, this choice is random in our implementation.
Starting from this terminal we search for the active component $W(t)$ by using a breadth-first-search. The search considers only the reduced costs of all incoming edges. If it encounters an edge (i, j) with $\tilde{c}_{(i,j)} > 0$, the search is aborted and j is added to $W(t)$.
After finding the active component, Δ stores the minimum reduced cost of all incoming edges in $W(t)$. Thus, Δ is the minimum cost to extend the active component $W(t)$ by at least one vertex. We then reduce all the reduced costs of the edges entering $W(t)$ by Δ. Since Δ corresponds to the costs added in this iteration, the lower bound is increased by Δ and the search moves to the next iteration.
We delete a terminal from the set of active terminals as soon as the breadth-first-search encounters another active terminal or the root. In this case the active terminal is connected to the rest of the solution.
In summary, the algorithm greedily tries to find the lowest cost to connect a terminal $t \in T \setminus r$ to other terminals or the root, respectively. Since the algorithm is greedy, it is only a bound and not an optimal solution (see Figure 10 for a example). For implementation details, the reader is referred to Pajor et al. in conjunction with Ljubić et al. [13, 12].

Reduction rules For the RPCSTP there are originally three reduction rules [12]. While two of these reduction rules focus on the deletion of vertices and edges, the third reduction rule can only be applied in the prize-collecting case. Since LAN-Force$_\mathrm{a}$ is not prize-collecting, we only describe the other two reduction rules here.
The reduction rules compare a local lower bound L with a global upper bound U. Since we are considering a minimisation problem, we can exclude a solution if $L > U$ holds. In the following, we denote by $\tilde{d}(i, j)$ the cost of the shortest path from $i \in V$ to $j \in V$, using the reduced cost $\tilde{c}$ as the underlying cost.

Proposition 6 (*Test 1* in [12])**.** *An arc $(i, j) \in A$ can be removed if $LB + \tilde{d}(r, i) + \tilde{c}_{ij} + min_{t \in T \setminus r} \tilde{d}(j, t) > UB$.*

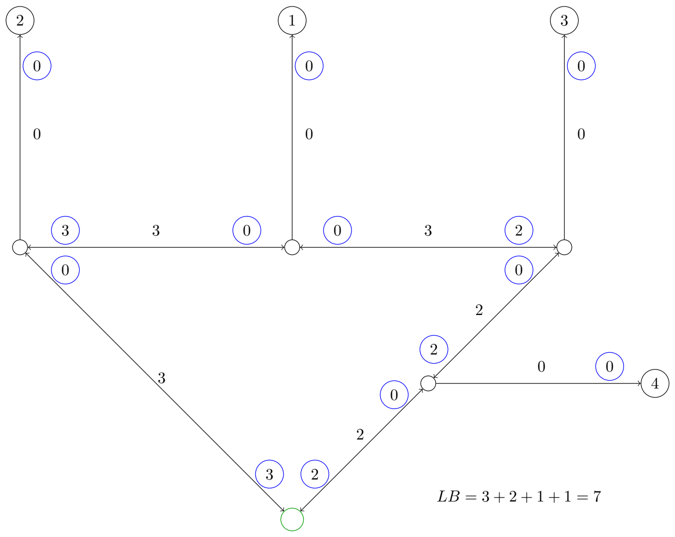


Figure 10: DA instance after applying Algorithm 1. The vertices were selected in ascending order of their ID. The black numbers next to the edges represent the original costs. We show the reduced costs in blue circles. The lower bound of 7 is smaller than the optimal cost of 9.

LB denotes the minimum cost of an optimal solution. If the edge (i, j) is part of the optimal solution, it must connect the root to at least one other terminal, otherwise it would not belong to the optimal solution. The starting point is the solution found in the preprocessing step. Thus, the reduced costs $\tilde{c}$ can be considered as the path costs.

Proposition 7 (*Test 2* in [12])**.** *A vertex $i \in V \setminus r$ can be removed if $LB + \tilde{d}(r, i) + min_{t \in T \setminus r} \tilde{d}(i, t) > UB$.*

The reasoning is similar to Proposition 10. If i is a terminal, it must be connected to the root, the path length to the next terminal $min_{t \in T \setminus r} \tilde{d}(i, t)$ is equal to 0, since it is itself the next terminal. If i is a non-terminal vertex, it must be connected to both the root and at least one other terminal to be part of the optimal solution.

DA for LAN-Force$_\mathbf{a}$

Preprocessing First, we compute the graph without net cascade $G_f = (V, E')$ from the input instance $G = (V, E)$. We also append the artificial root r to G_f. We can use the resulting instance as input to the DA algorithm for the RSTP described above.

Incorrect edge deletion for LAN-Force$_\mathbf{a}$ When applying the reduction rules described above to LAN-Force$_\mathrm{a}$, it is possible to delete vertices and edges which would actually be part of an optimal solution. The reason for this is that not all amplifier points considered by DA necessarily have to be connected to the fibre network, but some can also be supplied by the coax network. However, DA may delete shortest paths between amplifier point v_i and root, which are part of an optimal solution for LAN-Force$_\mathrm{a}$. This happens when there is an amplifier point v_j such that the common connection of v_i and v_j to the root is edge-disjoint with the shortest path from v_i to the root and the cost of connecting them together is cheaper than connecting them individually. However, if it is possible to connect v_j to the coaxial network, the shortest path from v_i to r would possibly be part of the optimal solution. Figure 11 shows an example.

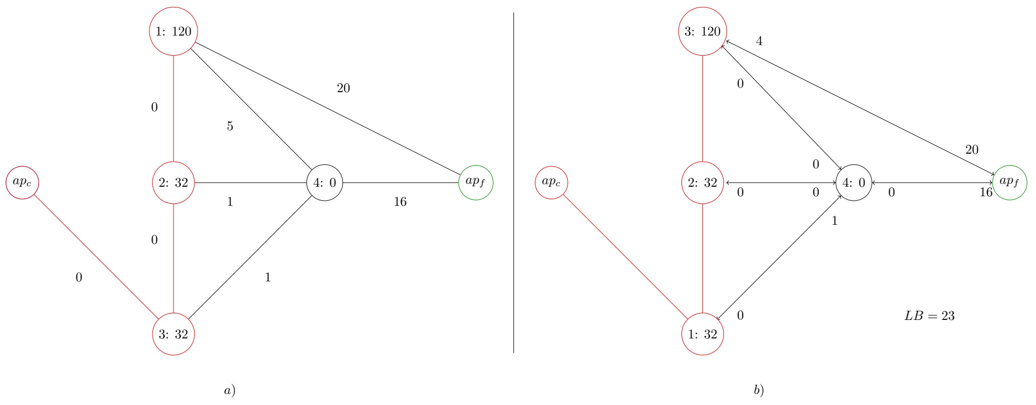


Figure 11: *a*) LAN-Force$_\mathrm{a}$ instance to which we apply DA. *b*) LAN-Force$_\mathrm{a}$ instance after applying the DA routine to calculate LB (for a derivation of $LB = 23$ and the reduced cost, see the Appendix C). Let the upper bound (and optimal solution) of the Steiner problem be $U = 23$. Applying *Test 1* to the edge $e = (3, ap_f)$ deletes it, since the condition $LB + \tilde{d}(r,i) + \tilde{c}_{ij} + min_{t \in T \setminus r} \tilde{d}(j,t) > UB$ is satisfied with $23 + 0 + 4 + 0 > 23$. However, since in the LAN-Force$_\mathrm{a}$ problem vertices 1 and 2 can be supplied via the net cascade, this edge is included in the optimal solution.

Solving the problem of incorrect edge deletion We found an additional preprocessing step to prohibit the deletion of edges and vertices of the optimal solution. For this we divide the amplifier points V_{AMP} into the two disjoint groups V_{AMP_1} and V_{AMP_2} such that $V_{AMP_1} \cap V_{AMP_2} = \emptyset$ and $V_{AMP_1} \cup V_{AMP_2} = V_{AMP}$. All $v \in V_{AMP_1}$ have a demand greater than the coaxial cable capacity u and thus must be connected to the fibre network. All other vertices whose demand is smaller than u form the set V_{AMP_2}. As a result we may be able to connect all vertices in V_{AMP_2} to the coaxial network. We temporarily set the demand of all amplifier points in V_{AMP_2} to 0, thus declaring them to be normal intersection vertices. All vertices in V_{AMP_1} accordingly represent the terminal vertices for the subsequent execution of the DA algorithm. We use the resulting instance as input to DA. When calculating the upper bound, we use pcst-fast on the unmodified instance to include the vertices from V_{AMP_2}. The larger V_{AMP_1} and the smaller V_{AMP_2}, the more effective DA's reduction techniques are. If $V_{AMP} = V_{AMP_1}$ holds, then DA can be applied without modification. After applying the DA routine for calculating LB and the reduced costs, we set the profits of all vertices in V_{AMP_2} to their original value and continue with the reduction tests.

Proof In the following, we prove that DA does not delete any vertices and edges required for an optimal solution using our additional preprocessing step. We distinguish between three cases and prove that in all three cases we do not delete edges and vertices of an optimal solution.

- **Case 1:** $V_{AMP} = V_{AMP_1}$: All amplifier points v have a demand of $d_v > u$. We cannot connect a single one of them to the net cascade because the capacity of the net cascade is not sufficient to satisfy even one of them. Since all customers must thus be connected to the fibre network in the best possible way, the RSTP problem arises and we can apply DA.

- **Case 2:** $V_{AMP} = V_{AMP_2}$: All customers have a demand of $d_v \leq u$. Thus, no single customer has to be connected to the fibre network. It is unclear which customers should be connected to fibre and which customers should be connected to coaxial. Since there are no more amplifier points in the modified instance, the lower bound also corresponds to $L = 0$. The reduced cost of each edge is equal to the original cost. The upper bound U corresponds to the maximum cost to connect all customers to fibre. We thus only delete edges if there is certainly a cheaper way to connect each amplifier point to fibre. (This case corresponds to the deletion of vertices, which we already perform when focusing on net cascades in Chapter 4.1).

- **Case 3:** $V_{AMP_1} \neq \emptyset$ and $V_{AMP_2} \neq \emptyset$. U corresponds to the maximum cost to connect all amplifier points to fibre. LB corresponds to the minimum cost to connect all amplifier points from V_{AMP_1} to fibre. $U - LB = B$ corresponds to the maximum cost to connect all vertices from V_{AMP_2} to the fibre network if all amplifier points from V_{AMP_1} are already connected. B is therefore an upper bound for the connection of V_{AMP_2}. Now, in the optimal solution, if we connect the set $N \subseteq V_{AMP_2}$ to the net cascade and $G \subseteq V_{AMP_2}$ is served by fibre, we only need to connect the customers from G to r. Since B is an upper bound for *all* vertices from V_{AMP_2}, all paths of the connection for G are also included. Since any reduced edge costs due to clients from V_{AMP_1} only further reduce the cost of the reduction test, we cannot delete edges that would be part of an optimal solution.

Since the optimal solution is preserved in all three cases, we can use the reduction technique.

4.2.5 Voronoi diagram tests

The Voronoi diagram tests use geometric properties of the input graph to transform it into a smaller graph. As for DA, we calculate a global upper bound U and local lower bounds L. Then U is compared to L. If for an explicit instance $L > U$ holds, the instance does not represent an optimal solution and it is possible to delete vertices or edges. The Voronoi reduction techniques were originally described for the PCSTP [10]. In this paper we first describe our adapted version for the STP. Since, as for DA, optimal solutions could possibly be deleted by applying these reduction techniques, we use the adjustment already described for DA to guarantee the preservation of all optimal solutions. We prove that this adjustment does not delete any optimal solutions in the Voronoi diagram tests either.

Voronoi diagram tests for STP

Preprocessing A Voronoi diagram of a graph $G = (V, E)$ is a division of all vertices in V into vertex-disjoint partitions, so that for each terminal $t \in T$ there is a separate partition. The individual vertices are each assigned to the terminal t closest to it. Formally, a Voronoi diagram is a partition $\{N(t) \mid t \in T\}$ of V such that:

$$v \in N(t) \Rightarrow d(v,t) < d(v,t')\ \forall t' \in T.$$

In the case of equality of the distances $d(v,t)$ and $d(v,t')$ we add v randomly to partition $N(t)$ or $N(t')$. Each one of these partitions is also called Voronoi region, we call the respective terminals t_i base of the partition ($t_i = base(v_i)$). The radius $radius(t_i)$ of a partition $N(t_i)$ is defined as the minimum cost to leave the partition $N(t_i)$ from the terminal t_i. The sum over all radii $\sum_{t \in T} radius(t)$ is a lower bound of the optimal solution. It is only a lower bound because we cannot guarantee that we can connect every terminal t_i to the solution via the cheapest cost $radius(t_i)$. Figure 12 shows an exemplary division of a graph into its partitions in addition to the respective radii.

Reduction rules In the following, we describe two reduction techniques that use the previously calculated Voronoi diagrams to delete both vertices and edges of the input graph. For the application of the reduction rules it is necessary to calculate an upper bound U. We use the heuristic pcst-fast again for this purpose. Without loss of generality we sort the terminals in ascending order according to the value of their radius. Furthermore, $\underline{d}(v_i, v_j)$ denotes the distance between vertices v_i and v_j avoiding intermediate terminals. With $v_{i,1}$ and $v_{i,2}$ we denote the terminals with the shortest and second shortest distance to v respectively.

Proposition 8 (*Test 1* [10]). *Let $v_i \in V \backslash T$. If a minimum Steiner Tree $S = (V_S, E_S)$ with $v_i \in V_S$ exists, then $\underline{d}(v_i, v_{i,1}) + \underline{d}(v_i, v_{i,2}) + \sum_{q=1}^{s-2} radius(t_q)$ is a lower bound on the cost of S.*

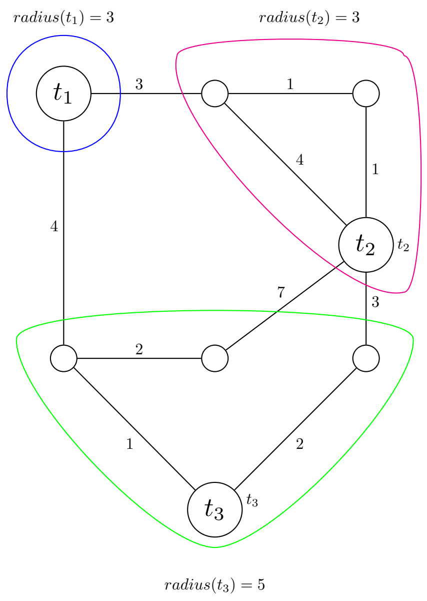


Figure 12: A graph with three terminals divided into its Voronoi diagram. Each terminal has its own Voronoi region. Each vertex belongs to exactly one Voronoi region.

With the help of this reduction technique it is possible to delete single vertices, which certainly cannot be part of an optimal solution. A vertex v_i is only part of an optimal solution if it is connected to at least two terminals. Since we are looking for a lower bound, we can assume that v_i lies on a path that connects the two terminals with the highest connection cost and therefore with the biggest radius. We can calculate a lower bound for the entire instance from the sum over all radii. Since we assume that v_i connects the two most expensive terminals, we only need to sum up the radii of all terminals except the two most expensive ones. Then we have to add up the costs of the paths from v_i to the next two terminals. If the resulting lower bound L is greater than the previously calculated upper bound U, v_i cannot be part of an optimal solution and can therefore be deleted.

Proposition 9 (*Test 2* [10]). *Let $\{v_i, v_j\} \in E$. If there is a minimum Steiner tree $S = (V_S, E_S)$ such that $\{v_i, v_j\} \in E_S$, then L defined by*

$$L = c_{\{v_i,v_j\}} + \underline{d}(v_i, v_{i,1}) + \underline{d}(v_j, v_{j,1}) + \sum_{q=1}^{s-2} radius(t_q)$$

if $base(v_i) \neq base(v_j)$ and

$$L = c_{\{v_i,v_j\}} + min\{\underline{d}(v_i, v_{i,1}) + \underline{d}(v_j, v_{j,2}), \underline{d}(v_i, v_{i,2}) + \underline{d}(v_j, v_{j,1})\}+$$

$$\sum_{q=1}^{s-2} radius(t_q)$$

otherwise, is a lower bound on the cost of S.

The goal of test 2 is to delete edges $\{v_i, v_j\}$ which cannot be part of the optimal solution. Both v_i and v_j must be connected to at least one terminal to be part of the optimal solution. We make a distinction between two cases: If both vertices of the edge are in different Voronoi regions, the shortest path to the nearest terminal is summed to L for both vertices. However, if $base(v_i) = base(v_j) = t_i$, the nearest terminal for both vertices is t_i. Since the edge must connect two different terminals, the shortest path to another terminal in another Voronoi region is searched. In both cases, we estimate the cost of the remaining terminals using the radii.

Voronoi diagram tests for LAN-Force$_{\mathbf{a}}$ If we apply Voronoi reduction techniques to the LAN-Force$_a$ problem without further adaptation, we may delete vertices and edges necessary for the optimal solution. As already described for DA, this problem occurs when we consider individual amplifier points in the Voronoi calculation for fibre cables but subsequently connect them to the net cascade. We use the same preprocessing step described for DA by splitting the amplifier points into the sets AMP_1 and AMP_2 and then computing the lower bound for AMP_1 only. The proof can be done analogously.

4.3 Optimisation

After splitting the input instance into individual subgraphs and reducing these subgraphs, we search for an optimal solution on each reduced subgraph. To solve the NP-complete LAN-Force$_a$ problem we use Integer Linear Programming (ILP) formulations. An ILP attempts to optimise a given linear function $c^T x$ by assigning $x \in \mathbb{R}^n$ the optimal values. The optimisation is subject to various constraints, which are specified by inequalities. Moreover, each $x_i, (i = 1, \ldots, n)$ must be an integer. To solve the ILPs, we use commercial software that uses a branch-and-bound method to find an optimal solution [14].
We use two ILPs which are originally formulated in [4] for the PC-LAN problem. In the following, we explain the ILPs Single Commodity Flow (SCF) model and cut-set (CUT) model along with the adaptations we devised to find an optimal solution to the LAN-Force$_a$ problem. While the SCF model models a network flow from the root to the amplifier points, the CUT model works with exponentially many constraints that aim at graph connectivity. Our adaptations take the following properties of the LAN-Force$_a$ problem into account, which are not part of the PC-LAN problem:

- Instead of a single root, there are multiple access points that can serve the network.

- LAN-Force$_\mathrm{a}$ is not prize-collecting, we must include all amplifier points in the solution.
- Each edge has exactly one capacity and one cost. There are not multiple modules on each edge.
- There is no overbuilding risk.
- On a path from access point to amplifier point we are not allowed to use a fibre cable after using a coaxial cable.

4.3.1 SCF model

Before applying the ILP, we create an artificial root r. Since the SCF model works on directed graphs $G' = (V, A)$ we transform all undirected edges $e = \{i, j\} \in E$ of the input graph $G = (V, E)$ into the directed edges $\overrightarrow{a} = (i, j)$ and $\overleftarrow{a} = (j, i)$. The costs and capacities of the new edges $c_{\overleftarrow{a}}, c_{\overrightarrow{a}}, u_{\overleftarrow{a}}, u_{\overrightarrow{a}}$ correspond respectively to the costs and capacities of the old edges c_e and u_e. We use the binary variables y_v for the vertices and x_a for the edges to decide which vertices and edges are included in the solution. We introduce the continuous flow variables f_{ij} for each edge $(i, j) \in A$. These flow variables represent the bandwidth that we direct from the root to the amplifier points via the edge (i, j). We distinguish the direct edges $a \in A$ into fibre edges a_f and coaxial edges a_c. We define the SCF model as follows:

$$\min \sum_{a \in A} c_a x_a \tag{1}$$

subject to

$$\sum_{a \in \delta^+(v)} f_a - \sum_{a \in \delta^-(v)} f_a = \begin{cases} -d_v y_v, & \text{if } v \in K \\ \sum_{k \in K} d_k, & \text{if } v = r \\ 0, & \text{otherwise} \end{cases} \qquad \forall v \in V \tag{2}$$

$$\sum_{a_f \in \delta^+(v)} f_{a_f} - \sum_{a_f \in \delta^-(v)} f_{a_f} \leq 0 \qquad \forall v \in V \setminus \{r\} \tag{3}$$

$$0 \leq f_a \leq u_a x_a \qquad \forall a \in A \tag{4}$$

$$x_a \leq f_a \qquad \forall a \in A \tag{5}$$

$$0 \leq (\sum_{k \in K} d_k) x_a - f_a \qquad \forall a \in A \tag{6}$$

$$y_k = 1 \qquad \forall k \in K \tag{7}$$

$$f_a \in \mathbb{R}^+ \qquad \forall a \in A \tag{8}$$

$$x_a \in \{0, 1\} \qquad \forall a \in A \tag{9}$$

$$y_v \in \{0, 1\} \qquad \forall v \in V \tag{10}$$

The objective in (1) minimises the cost of the solution, which is the sum of the edge costs of all edges in the solution. The constraints in (2) are called *flow preservation constraints* and come from [4]. The goal of these constraints is to provide the clients with sufficient flow. Thus, each amplifier point k in the solution consumes its demand d_k of flow, the root r emits enough flow that we can supply each customer, and each vertex in the street intersections I forwards all the flow routed through it. We developed the constraints in (3) to prohibit the use of fiber optic cables after coax cables. These constraints make it impossible to forward flow arriving via coaxial cables via fibre optic cables, as the outgoing flow through fibre optic cables must not be greater than the incoming flow through fibre optic cables. While the constraints in (4) guarantee that the flow at each edge is positive but does not exceed the capacity of the edge, the constraints in (5) and (6) say that an edge is only included in the solution if we pass a positive flow over it. Finally, the constraints in (7) ensure that every amplifier point is included in the solution.

4.3.2 CUT model

Unlike the SCF model, the CUT model is not based on explicitly modelling network flow in the graph. Instead, we insert exponentially many constraints to obtain a connected graph. These constraints are dynamically separated in a cutting plane approach, since inserting exponentially many constraints is inefficient when creating the ILP. In addition to the constraints necessary for finding an optimal solution, we explain a variety of other constraints that reduce the search space and thus we can find an optimal solution faster. In the following, we explain the necessary and additional constraints of the branch-and-cut approach. Our CUT model extends the CUT model introduced in [4] for the PC-LAN problem. We have adapted the constraints to the LAN-Force$_a$ problem. Furthermore, we have developed constraints so that routing the data stream from coaxial cable to fibre optic cable is prohibited.

$$\min \sum_{a \in A} c_a x_a \tag{11}$$

subject to

$$\sum_{a \in \delta^-(S)} u_a x_a \geq \sum_{k \in S} d_k \qquad \forall S \subset V \text{ s.t. } S \cap K \neq \emptyset \text{ and } r \notin S \tag{12}$$

$$\sum_{a_f \in \delta^+(v)} x_{a_f} - 1 \geq -M(1 - \delta_v) \qquad \forall v \in V \setminus \{r\} \tag{13}$$

$$\sum_{a_f \in \delta^+(v)} x_{a_f} \leq M(\delta_v) \qquad \forall v \in V \setminus \{r\} \tag{14}$$

$$\sum_{a_f \in \delta^-(v)} x_{a_f} \geq \delta_v \qquad \forall v \in V \setminus \{r\} \tag{15}$$

$$y_k = 1 \qquad \forall k \in K \tag{16}$$

$$x_a \in \{0,1\} \qquad \forall a \in A \tag{17}$$

$$y_k \in \{0,1\} \qquad \forall k \in K \tag{18}$$

$$\delta_v \in \{0,1\} \qquad \forall v \in V \setminus \{r\} \tag{19}$$

A prerequisite for the successful application of the CUT model is an explicit root. For this reason, we create the artificial root r. Furthermore, we transform the undirected input graph into a directed graph, as already described for the SCF model.

The *cut-set inequalities* (12) are exponentially many constraints that are dynamically separated in the context of a cutting plane approach. These constraints state that every subset of vertices S, containing at least one customer and not containing r, must have enough incoming capacity to route the total demand requested inside the set. As a result, the solution is connected, since we can reach any customer-containing vertex set S via incoming edges from the root. We developed the constraints in (13) – (15) to prohibit the use of fiber optic cables after the use of coax cables. We can do this by having at least one incoming fibre edge at each vertex with an outgoing fibre edge in the solution (see Figure 13). For this purpose, we define for each vertex $v \in V \setminus r$ the binary auxiliary variable δ_v. With the help of the inequalities in (13) and in (14), we set δ_v to 1 if at least one outgoing fibre edge is used at vertex v. If we do not use any outgoing fibre edge we set $\delta_v = 0$. M corresponds to a sufficiently large number (e.g. $\max(\{\delta^+(v) | v \in V\})$). The number of fibre edges entering v must be at least as large as δ_v. We express this with the constraints in (15).

These constraints already lead to an optimal solution. By adding the additional constraints described below, it is possible to speed up the process.

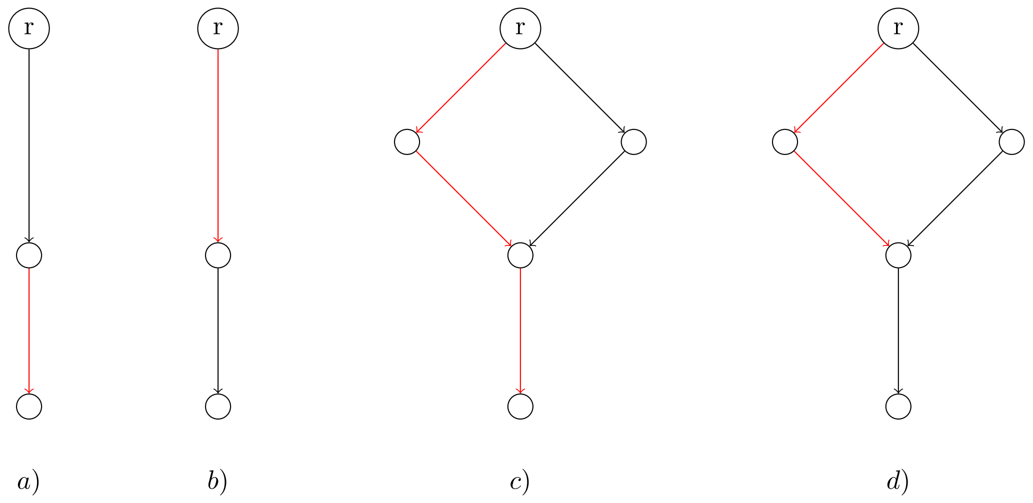


Figure 13: Black edges: coaxial cable, red edges: fibre cable. Apart from *a)*, all structures are permitted. Traffic reaching a vertex over coax cable must not be routed further via fibre cable.

Further inequalities:

$$\sum_{a\in\delta^-(S)} \min(u_a, \sum_{k\in S} d_k) x_a \geq \sum_{k\in S} d_k \qquad \forall S \subseteq V \text{ s.t. } S\cap K \neq \emptyset \text{ and } r\notin S \tag{20}$$

$$\sum_{a\in\delta^-(S)} x_a \geq 1 \qquad \forall S \subseteq V \text{ s.t. } S\cap K \neq \emptyset \text{ and } r\notin S \tag{21}$$

$$\sum_{(l,i)\in A, l\neq j} x_{li} \geq x_{ij} \qquad \forall (i,j)\in A, i\notin K, i\neq r \tag{22}$$

$$\sum_{(j,l)\in A, l\neq i} x_{jl} \geq x_{ij} \qquad \forall (i,j)\in A, i\notin K, j\neq r \tag{23}$$

$$\sum_{a\in\delta^-(i)} x_a \geq 1 \qquad \forall i\in K \tag{24}$$

$$x_{ij} + x_{ji} \leq y_i \qquad \forall (i,j)\in A \tag{25}$$

We can tighten the already known *cut-set inequalities*. For this we check for each edge entering the cut S whether the total flow $\sum_{k\in S} d_k$ required in the cut is smaller than the respective edge capacity. In this case, we can choose the sum over the demands as the upper bound. The *connectivity cuts* constraints in (21) express that each customer-containing subset of vertices without the root r must have at least one incoming edge. Since these are again exponentially many constraints, they are also dynamically separated as part of the cutting plane approach. Each non-customer vertex simply forwards the traffic and can therefore have neither only incoming nor only outgoing edges. This is expressed by the constraints in (22) and in (23). We can reduce the exponentially many *connectivity cut* constraints to linearly many constraints by restricting S to single customer vertices $S = \{k\}$. Since every customer vertex must be included in the solution, we can additionally use the constraints in (24) initially when creating the ILP. Finally, the constraints in (25) make it impossible to choose both directed edges between two vertices i and j.

Apart from the exponentially many constraints from (12), (20) and (21), we use the inequalities described above initially for the ILP. While finding the optimal solution, we perform a cutting plane approach at each vertex of the branch-and-bound tree. We describe the separation of the solution and the insertion of the violated *cut-set* and *connectivity-cut* inequalities below.

Separation

We perform the separation phase at each vertex of the branch-and-bound tree in polynomial time. We try to detect unconnected parts of the graph and insert additional constraints to guarantee graph connectivity. For this we use the *max-flow-min-cut* [15] to detect how much flow can be directed from the root to the amplifier points in the solution. If not all amplifier points are sufficiently supplied, this corresponds to a violated constraint. In detail, the separation phase runs as follows.
Given the fractional solution (x^*, y^*) of the ILP relaxation. We create the directed support graph $G' = (V', A')$, where $V' = V \cap \{t\}$ and $A' = A_1 \cap A_2$ with $A_1 = \{a \in A | x^*_a > 0\}$ and $A_2 = \{(k,t) | k \in K\}$. We set the capacities of all edges $a \in A_1$ to u_a and the capacities of edges $a = (k,t) \in A_2$ to d_k. Subsequently, we can use the *max-flow-min-cut* from the root r to the new sink vertex t. If the maximum flow in G' is less than the total required demand $\sum_{k \in K} d_k y^*_k$, not enough traffic reaches the customer vertices. This is a violated *cut-set* constraint. The total capacity of the vertex sets separated by the cut must be at least as large as the demand of the insufficiently served vertex set.
In addition to the *cut-set* inequalities, we can also separate the *connectivity-cut* inequalities. From the given fractional solution (x^*, y^*) we create the graph $G' = (V, A)$, where the capacity of each edge corresponds to $u_a = x^*_a$. Then, we can invoke the *max-flow-min-cut* problem on G' between the root r and each client vertex $k \in K$ to find violated *connectivity-cut* constraints. In each branch-and-bound vertex we insert the missing inequalities and continue solving the ILP.

4.4 Heuristic

Finding an optimal solution can take a long time on large graphs. For this reason we developed a heuristic that approximates an optimal solution. Our heuristic has variations in finding subgraphs, reducing them and finding an optimal solution. In the following we describe the process for an input instance $G = (V, E)$.

4.4.1 Focus on net cascades

Analogous to the process of finding an optimal solution, we can combine neighbouring net cascades and merge them into one large net cascade. Focusing on the respective net cascade afterwards, we calculate the graph $G_{sub} = (V', e')$ using the three upper bounds $UB_1 - UB_3$ as described in Chapter 4.1. For finding an optimal solution, we further reduced G_{sub} by deleting all vertices $i \in I$ whose sum of shortest distances to the next access point and to the next amplifier point was less than or equal to the smallest upper bound $ub = \min(UB_1, UB_2, UB_3)$. In the heuristic, we instead use the convex hull around all access points and amplifier points (including an ε-distance) in G_{sub} to further reduce G_{sub}. Figure 14 shows an example. Since

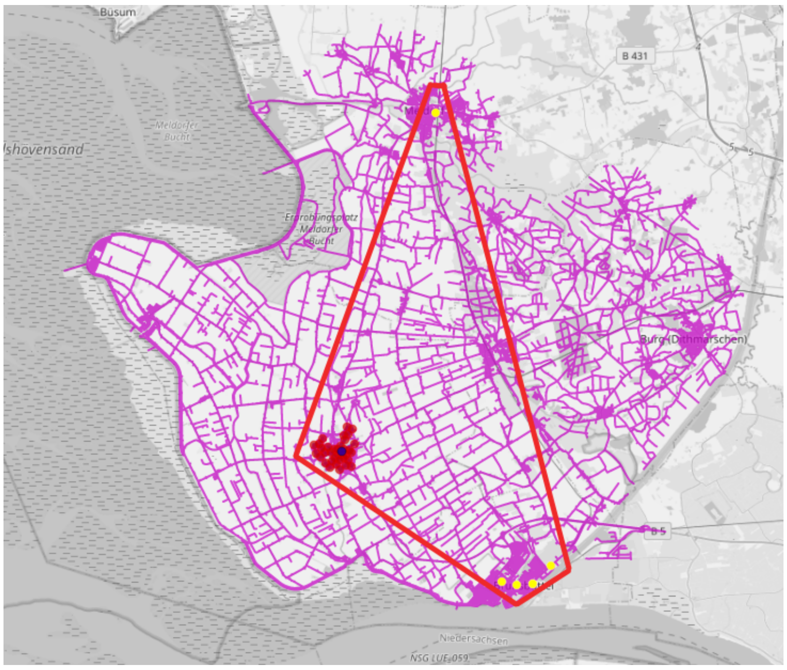


Figure 14: We calculate the convex hull around the fibre access points (yellow), the net cascade access points (blue) and the amplifier points (red). All vertices and edges lying in the convex hull form the subgraph G_{sub}.

the optimal solution is usually completely within the convex hull, this method is a good approximation. However, it is also conceivable that the shortest path between access point and amplifier points lies outside the convex hull.

4.4.2 Reduction techniques

We use all the reduction techniques described in Chapter 4.2 to reduce the input instance. For DA and the Voronoi reduction techniques, we omit the adaptation we found to the LAN-Force$_a$ problem. Instead, we use the original reduction techniques for Steiner trees on the graph without net cascade G_f. It is possible that we lose optimal solutions by doing this, but since the technique described is a heuristic, we can tolerate this.

4.4.3 Finding a solution

After the graph G_{sub} has been further reduced by the reduction techniques, the next step is to find a solution that is as close as possible to the optimal solution. Our strategy for finding an optimal solution is inspired by the strategy found by Ljubić et al. for the PC-LAN problem from [4]. We use the CUT model described in Chapter 4.3.2 to find an initial solution (x^*, y^*). For this, we solve the ILP and add violated inequalities to the LP in the first ten separation phases. We continue to solve the ILP until the eleventh branch-and-bound vertex. We use the fractional solution (x^*, y^*) as input to the following network construction phase. The goal of the network construction phase is to find the approximate solution to the LAN-Force$_a$ problem. We can represent this solution as a vector $z = [z_1, \dots, z_{|E|}]$, where $z_e = 1$ if e is included in the solution and $z_e = 0$ otherwise. The fractional input solution (x^*, y^*) provides the starting point for this by setting z_e to 1 for all edges $e = (i, j)$ if either $x_{ij} = 1$ or $x_{ji} = 1$.
We try to iteratively connect the individual customer vertices $k \in K$ with demand d_k via shortest paths to the root. For the shortest paths, the underlying costs are not the edge costs themselves, but the additional costs w_e incurred if we route the demand d_k via the edge e on which we already forward g_e traffic units. We calculate this cost for every edge as follows:

$$w_e(g_e, z_e, d_k) = \begin{cases} c_e & \text{if } g_e + d_k < u_e \text{ and } z_e = 0 \\ 0 & \text{if } g_e + d_k < u_e \text{ and } z_e = 1 \\ \infty & \text{if } g_e + d_k \geq u_e \end{cases}$$

By calculating the additional costs incurred, it is possible to efficiently connect customers to the root via the cheapest possible routes. Algorithm 2 shows the detailed procedure for finding a solution.

We initialise the algorithm in lines $1 - 3$. At the beginning, only the edges e of the fractional solution x^* are included in the solution by setting $z_e = 1$. We create a demand vector $b \in \mathbb{R}^{|V|}_{\geq 0}$, where b_v corresponds to the demand of the respective vertex $v \in V$. Since no traffic is initially routed through the network, the initial undirected flow is $g_e = 0$ for each edge. In line 4 and 5, we start the main loop by selecting a random vertex v with positive demand. We search for a shortest path from the root r to the vertex v, choosing the extended cost w as the underlying cost. For the LAN-Force$_a$ problem, since traffic routed through fibre is forbidden

Algorithm 2: Network Construction

```
Data: Customers $K$, minimum required capacities $g^*$, input Graph $G = (V, E)$
Result: Network design $z$
1  vertex demand $b$: $b_k \leftarrow d_k \ \forall k \in K$ and $b_v = 0 \ \forall v \in V \setminus K$;
2  Network design $z$: $z_e \leftarrow 1 \ \forall e = \{i, j\} \in E$ if $x_{ij} + x_{ji} > 0$, $z_e \leftarrow 0$ otherwise;
3  Undirected flow $g_e \leftarrow 0 \ \forall e \in E$;
4  while $\exists v \in V : b_v > 0$ do
5      pick random vertex $v \in V : b_v > 0$;
6      $b = b_v$;
7      $b_v = 0$;
8      define edge weight $w \leftarrow w_e(g_e, z_e, b) \forall e \in E$;
9      $length, path \leftarrow shortest\_modified\_path(G, \text{root} = r, \text{target} = v, \text{weight} = w)$;
10     if there is no shortest path then
11         return failed
12     for $e = (i, j) \in path$ do
13         $\bar{u} \leftarrow u_e - g_e$;
14         $\bar{b} \leftarrow \min(b, \bar{u})$;
15         $g_e \leftarrow g_e + \bar{b}$;
16         if $g_e > 0$ then
17             $z_e \leftarrow 1$
18         if $b > \bar{u}$ then
19             $b_i \leftarrow b_i + b + \bar{u}$;
20             $b_j \leftarrow b_j + \bar{u}$;
21             break
22 return $z$
```

to be forwarded over coax, we cannot use Dijkstra's algorithm for shortest paths. Instead, we developed a modified shortest path search, finding only paths where no coax paths follow fibre paths. For a more detailed description of our modified shortest path algorithm, we refer to the Appendix D. We abort the search for a solution in line $10 - 11$ if there is no such shortest path, as we cannot connect the vertex to the root in this case.
If we find a shortest path, we calculate in lines $12 - 15$ how much traffic we can route over the respective edges in addition to g_e. We include each edge over which we route traffic in the solution. If we cannot route all the required traffic b_v over an edge of the shortest path, we increase the demands of the vertices of the corresponding edge. After we have connected all vertices $k \in K$ via shortest paths with r, the algorithm terminates and returns the solution vector z, which induces the final solution.

5 LAN-Force$_\text{h}$

In this section, we describe finding an optimal solution to the NP-complete LAN-Force$_\text{h}$ problem. Given an input graph $G = (V, E)$. The graph contains the amplifier points served by the LAN-Force$_\text{a}$ problem and the customers to be connected to each amplifier point. Since G contains the environment of individual net cascades, it is not possible to split G into smaller subgraphs as described for the LAN-Force$_\text{a}$ problem. Therefore, in the following we directly start describing reduction techniques to reduce G to a smaller graph without losing the optimal solution.

5.1 Reduction techniques

To reduce a given instance of the LAN-Force$_\text{h}$ problem we again use the least cost tests and the basic reduction techniques described in Chapter 4.2. Since LAN-Force$_\text{h}$ is prize-collecting we can add another reduction technique.

Proposition 10 (TD_1 adapted from [10])**.** *Let $k \in K$ be a customer of degree 1 and set $e = \delta(k)$. If $e \in E_f$ and $p_{k_f} \leq c_e$ or $e \in E_c$ and $p_{k_c} \leq c_e$, then k and e can be discarded.*

It is only worthwhile to include a customer k in the solution if its profit exceeds the cost of the connection. The profit depends on the type of connection. If the incoming edge e is a fiber optic edge, we compare the profit of a fiber optic connection p_{k_f} with the edge cost c_e. Otherwise, we compare the profit of a coax connection p_{k_c} with the edge cost.

5.2 Optimisation

After applying the reduction techniques, we try to find an optimal solution to the LAN-Force$_\mathrm{h}$ problem. We again use ILPs for this purpose. The LAN-Force$_\mathrm{h}$ problem differs from the LAN-Force$_\mathrm{a}$ problem in the following properties:

- LAN-Force$_\mathrm{h}$ is prize-collecting, which means we do not necessarily have to include all customers in the optimal solution.
- Customers give different profits depending on the connection.
- Each customer may be connected to either fiber optic or coax. It is not possible for a customer to be connected by both types of cable.

Therefore, in order for the ILPs to find correct optimal solutions, we modify the ILPs described in Chapter 4.3 as follows.

5.2.1 SCF model

The modified SCF model works very similarly to the SCF model for the LAN-Force$_\mathrm{a}$ problem. We replace the objective(11) $\min \sum_{a \in A} c_a x_a$ with the new objective $\max \sum_{k \in K} \sum_{a_f \in \delta^-(k)} p_{k_f} x_{a_f} + \sum_{k \in K} \sum_{a_c \in \delta^-(k)} p_{k_c} x_{a_c} - \sum_{a \in A} c_a x_a$. Since LAN-Force$_\mathrm{h}$ is prize-collecting, it is no longer possible to minimise only over the edge costs. Instead, the total profit is optimised by maximising over the profit associated with the customers, reduced by the necessary edge costs. The customers give a profit, which depends on their respective connection. If the edge connected to customer k is a fiber optic edge a_f, the customer pays profit p_{k_f}, otherwise p_{k_c}.
Furthermore, we replace the constraints (7) $y_k = 1 \quad \forall k \in K$ with the new constraints $\sum_{a \in \delta^-(k)} x_a \leq 1 \quad \forall k \in K$. As a result, not every customer needs to be part of the solution. However, if a customer is included in the solution, it may only be connected by a single edge.
We adopt all other constraints of the SCF model for the LAN-Force$_\mathrm{a}$ problem. This results in a new ILP which is suitable for finding an optimal solution for the LAN-Force$_\mathrm{h}$ Problem.

5.2.2 Cut Model

We can also use the CUT model from Chapter 4.3.2 to solve the LAN-Force$_\mathrm{h}$ problem. For this we have to adapt the CUT model analogously to the changes for the SCF model. Thus, we replace one more time the objective $\min \sum_{a \in A} c_a x_a$ by the new objective $\max \sum_{k \in K} \sum_{a_f \in \delta^-(k)} p_{k_f} x_{a_f} + \sum_{k \in K} \sum_{a_c \in \delta^-(k)} p_{k_c} x_{a_c} - \sum_{a \in A} c_a x_a$. Furthermore we replace the constraints $y_k = 1 \quad \forall k \in K$ by the new constraints

$\sum_{a\in\delta^-(k)} x_a \leq 1 \quad \forall k \in K$. Since the LAN-Force$_\mathrm{h}$ problem is prize-collecting not all client vertices have to be part of the solution. For this reason, we modify the constraints (24) so that a customer must have an incoming edge only if it is also in the solution ($\sum_{a\in\delta^-(i)} x_a \geq y_i \quad \forall i \in K$). Furthermore, we need to change the initial connectivity-cut constraints in (21) from $\sum_{a\in\delta^-(i)} x_a \geq 1 \quad \forall i \in K$ to $\sum_{a\in\delta^-(i)} x_a \geq y_i \quad \forall i \in K$ to keep the problem prize-collecting. We adopt all other necessary and additional constraints of the CUT model for the LAN-Force$_\mathrm{a}$ problem without modification for the LAN-Force$_\mathrm{h}$ problem.

5.3 Heuristic

The heuristic described in this chapter finds an approximate solution to the LAN-Force$_\mathrm{h}$ problem. We directly start reducing the graph. For these reductions, we use all Basic Reductions and the Least Cost tests just as we would for computing an exact solution. The resulting reduced graph serves as input for the subsequent finding of an approximate solution. The strategy is very similar to finding an approximate solution for the LAN-Force$_\mathrm{a}$ problem, but we have to adapt some details. While for the LAN-Force$_\mathrm{a}$ problem we include every customer in the solution, for the LAN-Force$_\mathrm{h}$ problem we try to only add the customers with greater profit as connection costs. For this reason, instead of choosing an arbitrary vertex from all customer vertices in the main loop in each iteration, we make a choice beforehand which customers we want to connect to our solution in the first place. We have developed two approaches for this, which we explain in more detail below.

Approach 1: Using the fractional solution After the ILP has gone through 10 separation phases we get the fractional solution (x^*, y^*). As described for the heuristics for the LAN-Force$_\mathrm{a}$ problem, we use x^* as the initial solution vector z. We use the vertex vector y^* as the selection vector for the clients to be connected. Since we found in experiments that there are far more customers in y^* than in the optimal solution, we try to connect only a subset of the customers in y^* to the solution. For this, we sort the vertices in y^* in descending order of their fractional value and add only the customers with a fractional value greater than a threshold p_0 to the customer vector y'. We try different values for the threshold p_0. This strategy is similar to the one used in [4]. Unfortunately, when collecting our results, we found that the ILP only assigns integer values of 0 and 1 to vertices in y^*, making such a selection of client vertices impossible. For this reason, we developed another strategy that enables selection of customers. We use this second approach in the calculation of all our results in Chapter 6.2.

Approach 2: Using Distances In this approach, we use the solution vector (x^*, y^*) resulting from the separation phase one more time. Instead of sorting the vertices in y^* by their fractional value, we calculate the shortest path from the root vertex r to each of the vertices $y \in y^*$. We then sort the vertices in ascending order by their distance from r. We choose a threshold p_0 with $0 \leq p_0 \leq 1$, which states what percentage of all customers should be connected to the solution. With this threshold and the sorted list of vertices y^* it is now possible to build the list y' of customers to be connected by choosing the first p_0 percent of customers from y^*.

We proceed as described in Algorithm 2 to connect each customer $v \in y'$ via shortest paths to the root. Unlike in the LAN-Force$_\mathrm{a}$ problem, the customers give their profit depending on the particular connection. Let vertex v be a customer for which a coaxial connection via edge $\{u, v\}$ is possible. A connection over coax would cost the connection costs c_{e_c} in addition to the profit loss of a connection over fibre. For this reason, after calculating the additional costs w, we must increase the additional cost of the coax edge e by this profit loss ($w_e = w_e + (p_f - p_c)$). With this modified additional cost, it is now possible to connect the vertex to the root vertex via shortest paths as described in the heuristic for LAN-Force$_\mathrm{a}$. It may happen that a customer is connected to two cables due to the fractional x^* values. For this reason, after connecting all customers from y' to the root, we perform a shortest path search from the root to all customers over fibre. For all customers with two connections, we delete the incident coaxial edge if there is a shortest path, otherwise we delete the incident fibre edge.
With these three changes, we have modified the heuristic to a prize-collecting variant, where clients give a profit depending on their connection and are only connected via coax or fiber optic cable. We thus satisfy the properties of the LAN-Force$_\mathrm{h}$ problem that are not part of the LAN-Force$_\mathrm{a}$ problem and can thus find an approximate solution.

6 Results

In this chapter we discuss the performance of the approaches described in Chapter 4 and 5. We compute exact and approximate solutions for the LAN-Force$_\mathrm{a}$ and LAN-Force$_\mathrm{h}$ problems on various datasets provided by Vodafone GmbH. We implemented all techniques using Python version 3.8.3. To compute the optimal solution of our different ILPs, we used Gurobi version 9.0.2. To easily run our application in parallel for different instances, we use the workflow tool Snakemake version 6.4.0.
We conducted all experiments on the HPC cluster of the Heinrich-Heine University Düsseldorf. We used the Intel Xeon Gold 6136 CPU architecture with 3.00 GHz and 192 GB RAM. We used one core and a maximum of 100GB memory in our experiments. The code of the implementation and the associated data can be found on GitLab.

6.1 LAN-Force$_\mathrm{a}$

We divide the description of our results for the LAN-Force$_\mathrm{a}$ problem into two steps. In the first step, we focus on individual net cascades as described in Chapter 4.1.1 without considering whether a fusion of net cascades would make sense. Since this can lead to non-optimal solutions, we merge in the second step possibly related net cascades as explained in Chapter 4.1.2. We describe the success and runtime of our techniques and compare the exact approach with the approximate results of the heuristic.

6.1.1 Data

Vodafone GmbH provided us with different real-world graphs as input instances. The individual instances represent the street network, together with customers and all serving net cascades and access points of parts of Germany. Since no edge capacities are given in the instances, we assign a sufficiently large capacity to each possible fibre edge ($u_f = \sum_{k \in K} d_k \quad \forall k \in K$). We assign a cost of 0 and a fixed capacity of 64 to each coax edge. We calculated results for the instances 01051, landkreis_mettmann, landkreis_muenchen, landkreis_nuernberg, stadt_dresden, and stadt_muenchen. Since the conclusions drawn from the results are similar for all instances examined we focus on instance 01051 in the following description of the results. This instance is a representation of the district of Heide, it consists of 262143 vertices and 348280 edges. It is the instance with the fewest net cascades, that is why the description of the results is clearer using this instance. We show the results of the other instances in the extended Appendix on GitLab.

6.1.2 Performance on single net cascades

In order to be able to better assess the instances and follow the results, we show in Table 1 the size of the respective instances in the course of the solution process of the exact approach.

AP	Focused	Reduced	Solved
219878	11476	2971	115
	13970	4935	115
234333	504	81	24
	580	128	23
204993	14814	2084	74
	16671	4047	72
48477	11098	2384	104
	13065	4047	104
51824	5120	610	29
	5632	1055	28
156480	16106	4969	171
	20039	8169	175
10230	54949	18135	394
	69114	29501	397
16611	10535	2922	338
	12851	4803	337
20464	12	6	11
	11	5	10
148815	25135	4757	128
	29223	8059	129
106879	10723	2076	161
	12678	3514	163
1732	14	6	13
	13	5	12
138683	21915	3435	90
	24991	5879	89
54696	25840	5150	227
	30238	8687	229

Table 1: Graph sizes after applying the different steps of the solution process of the exact solution. In each cell we show the number of vertices at the top and the number of edges at the bottom.

Focus on net cascades First we calculate an upper bound for each net cascade as described in Chapter 4.1 using the three algorithms UB_1, UB_2 and UB_3. We show the different upper bounds of the algorithms together with the running time of the calculation in Table 2.

AP	UB_1	UB_2	UB_3
219878	1025827	1054833	**770540**
	0.1209	0.1346	7.8950
234333	**76694**	**76694**	**76694**
	0.0044	2.9087	8.2588
204993	**220104**	256272	228116
	0.0195	0.0376	7.9066
48477	509214	541650	**408716**
	0.0290	0.0519	8.1957
51824	**83556**	143907	112819
	0.0048	0.0088	10.4109
156480	1697983	1710631	**1194456**
	0.2980	0.2873	8.1339
10230	3058602	2390079	**1957420**
	9.7941	0.3392	9.5486
16611	2072594	**1434293**	**1434293**
	0.6483	0.0021	9.0261
20464	**0**	**0**	**0**
	0	0	0
148815	633640	633696	**460099**
	0.1047	0.0655	7.8460
106879	724785	680787	**507038**
	0.1219	0.0704	8.3528
1732	**0**	**0**	**0**
	0	0	0
138683	360271	601629	**315101**
	0.0283	0.1005	8.1121
54696	657620	666326	**496908**
	0.1079	0.1660	7.8372

Table 2: Column AP shows the vertex ID of the respective net cascade vertex. The columns UB_1 - UB_3 show the performance of the three different methods for calculating an upper bound. Each cell shows the upper bound found at the top and the runtime in seconds at the bottom. We print the respective best results in bold.

In general, the lower the upper bound, the more efficient is the subsequent focus on the respective net cascade. We found that UB_3 finds the best upper bound in 12 out of 14 cases. The only exceptions to this are the net cascades with access points 204993 and 51824. In the neighbourhood of these two net cascades there are a large number of fibre access points. This favours finding a good upper bound with the help of UB_1. The algorithm UB_2 did not find a better upper bound than the other two algorithms in any instance. Instances 20464 and 1732 are net cascades whose demand can be completely supplied by the existing coaxial cable network. This makes an execution of the three algorithms redundant. In terms of runtime, UB_1 and UB_2 are far superior to the algorithm UB_3. While the first two algorithms usually have a computation time of less than one second, UB_3 needs about 7 seconds on average to compute an upper bound. However, since the subsequent computation time of the reduction techniques and of finding an optimal solution is far greater on large graphs than on small graphs, the application of UB_3 is worthwhile. Overall, it is therefore worthwhile to use UB_3 to find an upper bound. However, since UB_1 can usually be executed in under a second, it is additionally advisable to apply this algorithm.

Reduction techniques Afterwards we apply the reduction techniques. At the beginning we executed the reduction technique Collapse net cascade to delete redundant amplifier points. We then proceeded with the calculation of the Basic Reductions, iteratively applying the techniques NTD_1 and NTD_2 until no vertices and edges are deleted. We finish the calculation of the Basic Reductions with a single application of UDV. Afterwards we use the reduction technique Least cost. Least cost requires a calculation of all shortest paths. Since such a calculation takes a long time, it is possible to limit the calculation of the shortest paths by determining a cut-off at the maximum edge costs of all edges. We do not lose optimal solutions and can perform Least Cost without restrictions. The reduction techniques Voronoi and Dual Ascent follow, where we use the restricted variant for finding the exact solution and the unrestricted variant for finding the approximate solution. We conclude the reductions by running the Basic Reductions again. Table 3 shows the elimination success and runtime of the individual reduction techniques.

AP	Collapse		Basic		Least		Voronoi		Dual		$\sum$	
	0.01	0.01	74.11	78.17	0.0	0.0	0.0	0.8	0.01	2.33	74.13	81.31
219878	0.01	0.01	64.67	69.17	0.0	0.0	0.0	1.11	0.0	3.17	64.68	73.45
	0.0032	0.0027	0.6750	0.6367	0.2581	0.1762	1.609	0.9788	4.8041	6.3664	7.3495	8.1608
	0.0	0.0	84.52	78.38	0.0	0.0	0.0	11.68	0.79	5.97	85.32	96.03
234333	0.0	0.0	78.45	69.6	0.0	0.0	0.0	16.3	0.0	8.66	78.45	94.57
	0.0002	0.0014	0.0095	0.3738	0.0027	0.0743	0.0374	0.2984	0.0341	0.3786	0.0839	1.1265
	0.0	0.0	86.52	85.26	0.0	0.0	0.6	7.53	1.17	3.65	88.29	96.44
204993	0.0	0.0	79.2	77.78	0.0	0.0	0.92	11.18	1.81	5.64	81.93	94.61
	0.0038	0.0055	0.4827	0.8371	0.1184	0.1957	0.6368	1.1267	1.1924	1.1944	2.4341	3.3594
	0.0	0.0	78.53	78.13	0.0	0.0	0.0	6.67	0.01	4.57	78.54	89.37
48477	0.0	0.0	69.03	68.9	0.0	0.0	0.0	9.46	0.0	6.36	69.03	84.71
	0.0028	0.0032	0.4281	0.9924	0.1871	0.2534	0.8245	1.2262	2.522	3.0851	3.9645	5.5603
	0.02	0.0	88.2	85.1	0.0	0.0	0.0	7.93	0.02	4.72	88.24	97.75
51824	0.02	0.0	81.37	77.54	0.0	0.0	0.0	11.81	0.0	7.3	81.39	96.66
	0.0015	0.0056	0.1004	1.0334	0.0284	0.2098	0.1447	1.1628	0.2959	1.1999	0.5709	3.6115
	0.0	0.0	69.15	74.47	0.0	0.0	0.0	0.19	0.01	0.67	69.16	75.33
156480	0.0	0.0	59.24	64.82	0.0	0.0	0.0	0.25	0.0	0.9	59.24	65.97
	0.004	0.0037	1.2653	1.2951	0.3999	0.2907	3.1508	2.7452	9.5439	15.2413	14.364	19.576
	0.0	0.0	67.08	70.52	0.0	0.0	0.0	4.66	0.02	4.15	67.1	79.33
10230	0.0	0.0	57.42	61.05	0.0	0.0	0.0	6.17	0.02	5.39	57.44	72.61
	0.0138	0.0124	6.4003	5.5158	1.3594	1.0258	11.5564	10.0198	35.3093	467.1767	54.6392	483.7505
	0.1	0.05	72.62	73.37	0.0	0.0	5.75	0.0	0.69	7.32	79.17	80.74
16611	0.09	0.04	63.12	64.3	0.0	0.0	7.72	0.0	0.9	10.02	71.82	74.35
	0.0032	0.0068	0.6283	2.6809	0.1864	0.5082	0.6368	2.6474	1.6301	88.1842	3.0848	94.0275
	58.33	58.33	33.33	33.33	0.0	0.0	0.0	0.0	0.0	0.0	91.67	91.67
20464	63.64	63.64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	63.64	63.64
	0.0005	0.0005	0.0003	0.0003	0.0001	0.0001	0.0268	0.0255	0.0111	0.0112	0.0388	0.0376
	0.0	0.0	81.19	82.09	0.0	0.0	0.0	1.99	0.04	3.88	81.23	87.96
148815	0.0	0.0	72.58	73.73	0.0	0.0	0.0	2.91	0.05	5.67	72.63	82.31
	0.0061	0.0058	1.1509	1.0929	0.2952	0.2699	2.5603	2.3713	6.0872	8.033	10.0997	11.7729
	0.05	0.05	80.74	82.19	0.0	0.0	0.0	1.98	0.01	2.55	80.8	86.77
106879	0.04	0.04	72.43	74.54	0.0	0.0	0.0	2.66	0.0	3.58	72.47	80.83
	0.003	0.0032	0.4887	0.8492	0.1128	0.1312	0.6979	0.87	2.3057	5.4094	3.6081	7.263
	64.29	64.29	28.57	28.57	0.0	0.0	0.0	0.0	0.0	0.0	92.86	92.86
1732	69.23	69.23	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	69.23	69.23
	0.0005	0.0005	0.0002	0.0003	0.0001	0.0002	0.026	0.0266	0.0113	0.0126	0.0381	0.0402
	0.0	0.0	84.6	84.07	0.0	0.0	0.3	6.22	0.06	4.55	84.96	94.84
138683	0.0	0.0	76.84	76.23	0.0	0.0	0.44	9.17	0.1	6.82	77.38	92.22
	0.0054	0.0062	0.9368	1.0324	0.2382	0.3083	1.6559	1.7679	3.3517	3.0187	6.188	6.1335
	0.0	0.0	80.1	83.24	0.0	0.0	0.0	1.29	0.0	7.47	80.1	92.01
54696	0.0	0.0	71.31	75.42	0.0	0.0	0.0	1.85	0.0	10.9	71.31	88.17
	0.0068	0.0063	1.37	1.2143	0.2823	0.2366	3.0014	2.543	6.7898	9.7216	11.4503	13.7218

Table 3: Column AP shows the vertex ID of the respective net cascade vertex. The columns Collapse, Basic, Least, Voronoi and Dual show the performance of the respective reduction techniques. On the left are the results of the exact reductions, on the right the results of the approximate reductions of the heuristics. The individual cells are divided as follows: Top: reduction of vertices in percent, Middle: reduction of edges in percent, Bottom: runtime in seconds.

In the following we will explain the performance of the individual reduction techniques in more detail. The Collapse net cascade method achieved particularly great success in instances 20464 and 1732. In both instances we were able to delete over 50% of all vertices and edges. As already mentioned, these two instances are the self-supplying net cascades, where no fiber rollout is needed. As a result we could reduce each branch of the net cascade to a single vertex. However, since the net cascades are very small compared to the entire subgraph in the other instances, the overall reduction success of Collapse net cascade is very low. With the Basic Reductions we were able to delete a large part of all vertices and edges of the input graphs. Apart from the two instances 20464 and 1732, the Basic Reductions deleted between 57 and 88 percent of all vertices and edges of the different input graphs. Although we applied the Basic Reductions in the same way for the exact solution approach and the heuristic, they differ slightly in success because the input instances are not the same due to different division into subgraphs. However, the difference is never more than 6%, indicating consistent success of the Basic Reductions. The Least Cost reduction technique was not able to delete any vertices or edges in a single instance. Since we are working on real world instances, which have geometric properties, it is unlikely that the direct connection between two vertices is more expensive than an alternative path connecting the two vertices via detours. For the reduction techniques Voronoi reductions and Dual Ascent reductions we find that the success of the unconstrained techniques for heuristics far exceeds our constrained techniques. For example, only in instance 16611 were we able to delete more than 1% of all vertices and edges using the restricted Voronoi method. With the unconstrained variant, we were able to delete more than 5% of all vertices and edges in 6 of the 14 instances. The same goes for the Dual Ascent reduction techniques.

Optimisation We tried to find an exact solution with both the SCF model and the CUT model. Since the creation of the support graph in the separation phase of the CUT model takes a long time, the results of the CUT model were far worse than those of the SCF model. In our experiments the CUT model is not even close to being a good alternative to the SCF model in any instance. We decided to explain only the results of the SCF model in more detail for a better overview. We set a timeout of 1800 seconds for the calculation of an exact solution. Table 4 shows the performance of the SCF model compared with the results of the heuristic.

AP	Objective		Ratio	Runtime		Gap
219878	138230.8	248779.8	79.97	1800.1952	1.4796	29.71
234333	76694.4	85571.99	11.58	0.6492	0.0894	0.0
204993	124026.29	142653.90	15.02	715.7041	0.4722	0.0
48477	144506.1	206295.59	42.76	1090.7401	0.9873	0.0
51824	38170.79	40746.6	6.75	5.7985	0.2571	0.0
156480	498937.20	545828.39	9.4	1801.8885	2.9445	60.77
10230	1415167.50	1494360.40	5.6	1800.4358	10.0994	32.46
16611	1345368.10	1348182.10	0.21	1800.1336	2.8176	26.18
20464	0.0	0.0	0	0.0007	0.0021	0.0
148815	260644.80	282368.39	8.33	1800.1486	2.1916	23.16
106879	281176.99	371744.79	32.21	1800.1799	0.8273	38.19
1732	0.0	0.0	0	0.0007	0.0020	0.0
138683	145429.49	154939.3	6.54	1599.4483	0.8083	0.0
54696	358259.99	408782.79	14.1	1801.2699	1.5771	60.68

Table 4: Column AP shows the vertex ID of the respective net cascade vertex. Column Objective shows the final costs of the fibre rollout, column Runtime the runtime of the solution in seconds. Both columns show the results of the exact strategy on the left and the results of the heuristic on the right. In the column Ratio we indicate by how many percent the solution of the heuristic is worse than the solution found by the SCF model. For the calculation of the exact solution we set a timeout to 1800 seconds. The column Gap shows the Gurobi gap of the exact solution in percent if we could not find an optimal solution before timeout.

Since LAN-Force$_\mathrm{a}$ is a minimisation problem, smaller objective values are better than larger ones. We could not find an optimal result in 7 of the 14 instances before the half-hour timeout occurred. In all instances where we could not find an optimal result, the Gurobi Gap indicates the gap between the lower and upper objective bound. Overall, the heuristic delivers results that are on average 10.43% worse than the results found by the exact method. As we show in Figure 15, the heuristic performs very differently on the respective instances. If the result is only 0.21% worse on instance 16611, it is a whole 79.97% worse on instance 219878. We assume that this is due to the distribution of the fibre access points. If these are rare and far away from the net cascade, many amplifier points are connected via shortest paths and the heuristic delivers good results. However, if there are many fiber access points near the net cascade, the heuristic finds a poor solution. We show in Figure 16 the solutions found by the heuristic and the exact method of these two instances for comparison. In individual cases, our heuristic found extraordinarily poor results that were up to 195798% worse than the solution found by the SCF model. In these cases, the CUT model could not find a reasonable fractional solution, so the solution of the heuristic is the whole input graph. Since these are exceptional cases, we recommend that results with exceptionally high costs be examined more closely

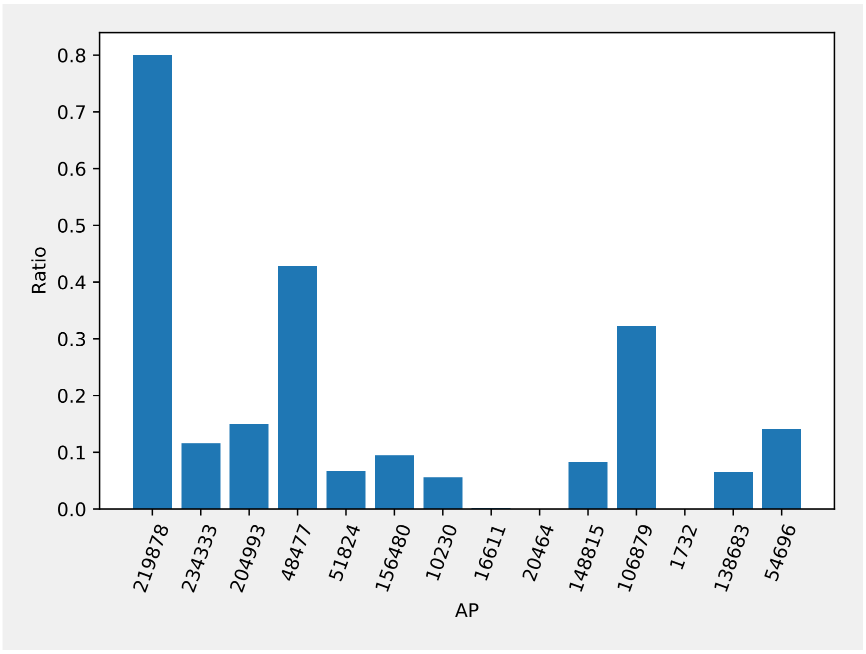
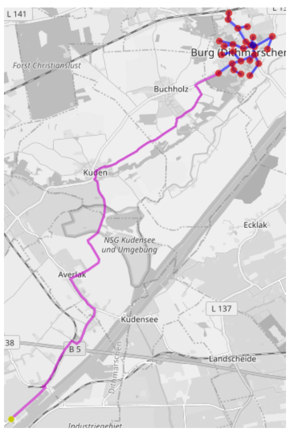


Figure 15: The ratio of how many percent the solution of the heuristic is worse than the solution found by the SCF model.

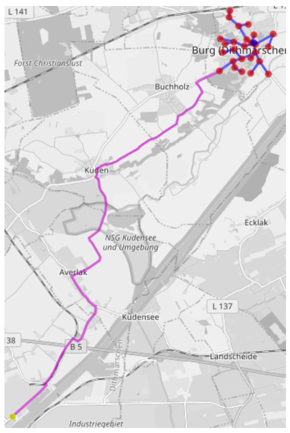
with the SCF model.
Runtime-wise, we were able to find results far faster using the heuristic than we were able to with the exact method. While the heuristic found a result within 10 seconds in all instances, we could not find an optimal result in half of the instances before the half-hour timeout occurred. Overall, the heuristic delivers good results in a very short time. Since the performance on the different instances can vary greatly, we still recommend verifying the results with the exact approach and a short timeout.

6.1.3 Performance on merged net cascades

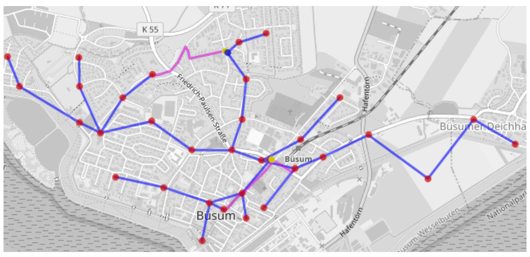
Focus on net cascades We calculated the fusion of the net cascades by first calculating an upper bound for each net cascade and then merging the net cascades where the resulting subgraph would have overlapped. This resulted in five unions of net cascades, which are composed of the original net cascades as follows: 786387: [219878, 48477, 156480, 10230, 16611, 106879], 786388: [234333], 786389: [204993, 51824, 148815, 138683, 54696], 786390: [20464], 786391: [1732]. It is noticeable that there are two large unions of net cascades with the access points 786387 and 786389. The other three new net cascades 786388, 786390 and 786391 each contain only one original net cascade. These are exactly the three original net cascades with the smallest upper bound (see Table 2). Since the upper bounds are very small in these cases, the resulting subgraph is also very small and therefore does not overlap



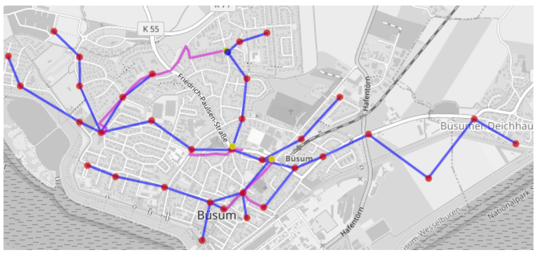
(a) 16611 SCF



(b) 16611 heuristic



(c) 219878 SCF



(d) 219878 heuristic

Figure 16: The exact solution compared to the solution of the heuristic in the net cascades 16611 and 219878. Red vertices and blue edges represent the net cascade with its blue access point. Pink edges show the fibre rollout to the yellow fibre access points.

with the subgraphs of the other net cascades.

AP	UB_1	UB_2	UB_3
786387	8971407 13.0461	14677969 197.7442	**6407960** 10.4311
786388	**76694** 0.0045	**76694** 3.1232	**76694** 8.4430
786389	1907613 0.2362	3202236 4.7866	**1607791** 9.6518
786390	**0** 0	**0** 0	**0** 0
786391	**0** 0	**0** 0	**0** 0

Table 5: Column AP shows the vertex ID of the respective net cascade vertex. The columns UB_1 - UB_3 show the performance of the three different methods for calculating an upper bound. Each cell shows the upper bound found at the top and the runtime in seconds that was needed to find it at the bottom.

In Table 5 we show the results of the calculation of the upper bounds after merging the individual net cascades. The results coincide with the results for individual net cascades. Thus, the performance of method UB_3 far outweighs the performance of the other two methods. Especially for the very large union 786387, the upper bound found by UB_3 is by far the best. Furthermore, UB_3 has the best runtime of 10 seconds in this particularly large instance. It is therefore worth using UB_3 to calculate an upper bound for small graphs as well as for large graphs.

Reduction techniques In Table 6 we show the reduction results. We could see that our observations of the reduction results for the individual net cascades can also be applied to the fusions of net cascades.

AP	**Collapse**		**Basic**		**Least**		**Voronoi**		**Dual**		$\sum$	
	0.01	0.01	56.65	62.87	0.0	0.0	0.0	0.0	0.0	0.0	56.65	62.88
786387	0.01	0.01	47.12	53.03	0.0	0.0	0.0	0.0	0.0	0.0	47.13	53.03
	0.0929	0.0653	61.927	25.1987	9.8821	6.297	387.2763	219.3	1073.0985	6565.347	1532.2768	6816.208
	0.0	0.0	84.52	78.4	0.0	0.0	0.0	11.68	0.79	5.97	85.32	96.05
786388	0.0	0.0	78.45	69.61	0.0	0.0	0.0	16.3	0.0	8.66	78.45	94.58
	0.0002	0.0013	0.0098	0.3121	0.0029	0.062	0.0341	0.2456	0.0371	0.3165	0.0841	0.9375
	0.0	0.0	69.22	64.18	0.0	0.0	0.0	16.76	0.01	7.16	69.23	88.11
786389	0.0	0.0	59.62	54.58	0.0	0.0	0.0	21.01	0.0	8.97	59.62	84.55
	0.0263	0.0549	6.9817	19.5558	1.7216	4.5134	32.3502	80.0488	89.4365	139.1518	130.5163	243.3247
	58.33	58.33	33.33	33.33	0.0	0.0	0.0	0.0	0.0	0.0	91.67	91.67
786390	63.64	63.64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	63.64	63.64
	0.0005	0.0005	0.0002	0.0002	0.0001	0.0001	0.0234	0.0238	0.011	0.0108	0.0352	0.0354
	64.29	64.29	28.57	28.57	0.0	0.0	0.0	0.0	0.0	0.0	92.86	92.86
786391	69.23	69.23	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	69.23	69.23
	0.0005	0.0005	0.0002	0.0003	0.0001	0.0001	0.0243	0.0272	0.0116	0.0111	0.0367	0.0392

Table 6: Column AP shows the vertex ID of the respective net cascade vertex. The columns Collapse, Basic, Least, Voronoi and Dual show the performance of the respective reduction techniques. On the left are the results of the exact reductions, on the right the results of the approximate reductions of the heuristics. The individual cells are divided as follows: Top: reduction of vertices in percent, Middle: reduction of edges in percent, Bottom: runtime in seconds.

The reduction technique Collapse net cascade has particularly great success only in the self-supplying net cascades. The Basic Reductions have the greatest success in all other instances. With them it is possible to delete large parts of the graph in a short time. The three reduction techniques Least Cost, Voronoi Reductions and Dual Ascent have a particularly high runtime in the two large net cascades 786387 and 786389. At the same time, none of the three reduction techniques have a particular reduction success especially in the exact application. Since the computational time of these reduction techniques is very long and the computational success is very low, we did not apply these reduction techniques in our other experiments for LAN-Force$_\mathrm{a}$ in the extended Appendix.

Optimisation Table 7 shows that here, too, the results largely coincide with the results for individual net cascades. In the very large instance 786387, however, we were even able to find a better result with the heuristic than with the exact method. While the exact method still had a gap of 94% after the timeout, the heuristic found a result in 136 seconds. In all other instances, the SCF model found a better result than the heuristic. Overall, the results of the heuristic are even 3% better than the results of the SCF model.

AP	Objective		Ratio	Runtime		Gap
786387	4299400.69	3992215.60	-7.14	1801.8404	135.9335	94.17
786388	76694.4	85571.99	11.58	0.7220	0.0881	0.0
786389	838836.10	980789.30	16.92	1800.5525	18.9793	80.69
786390	0.0	0.0	0	0.0006	0.0018	0.0
786391	0.0	0.0	0	0.0006	0.0020	0.0

Table 7: Column AP shows the vertex ID of the respective net cascade vertex. Column Objective shows the final costs of the fibre rollout, column Runtime the runtime of the solution in seconds. Both columns show the results of the exact strategy on the left and the results of the heuristic on the right. In the Ration column we show how the heuristic performs compared to the exact method in percent. For the calculation of the exact solution we set a timeout to 1800 seconds. The Gap column shows the Gurobi gap in percent of the exact solution if no optimal result could be found before timeout.

6.2 LAN-Force$_\mathrm{h}$

In the following we describe our results for the LAN-Force$_\mathrm{h}$ problem.

6.2.1 Data

As input graphs, we once again use the instances of Vodafone GmbH. In these instances there are no coax connections from the amplifier points to the individual customers. However, in the data some customers have the flag *'uep'*, which stands for transfer point. To offer a choice between coax connectivity and fibre connectivity, we create additional coax edges e with $c_e = 0$ and $u_e = 64$ from all customers marked with the flag *'uep'* to the respective nearest amplifier points. We set the profit p_f and p_c according to the type of connection. For every customer k, we set $p_c = 2000 \cdot d_k$ and $p_f = 3000 \cdot d_k$. To obtain realistic input graphs for the LAN-Force$_\mathrm{h}$ problem we focus on individual net cascades. For each net cascade we compute a rectangle with the respective access point in the centre, which we enlarge until a predefined number of customers α lies within the rectangle. In our examples we use α-values of 2000, 4000 and 8000 customers to get different graph sizes. All vertices and edges within the rectangle then form the input graph for the LAN-Force$_\mathrm{h}$ problem.
As with LAN-Force$_\mathrm{a}$, we have calculated results for the instances 01051, landkreis_mettmann, landkreis_muenchen, landkreis_nuernberg, stadt_dresden, and stadt_muenchen. Since the results of the different instances are very similar, we describe once again mainly the results for instance 01051. We show all the other results in the extended Appendix.

6.2.2 Reduction techniques

We applied the Basic Reductions and Least Cost reduction techniques to our instances. Table 8 shows our reduction success on instance 01051 with 2000, 4000 and 8000 households respectively. We noticed that the Basic Reductions delete about the same number of vertices and edges for each net cascade, regardless of the number of households. While the runtime in the instances with 2000 households is still less than 4 seconds, the Basic Reductions need up to 27 seconds in the case of 8000 households. The runtime thus increases extremely with the size of the input graph. The Basic Reductions were mostly able to delete between 10 and 20 percent of all vertices and edges of the input graph. In the LAN-Force$_\mathrm{a}$ instances, in contrast, they deleted on average about 70%! This is due to the structure of the input graphs. Most customers are only connected to the network by one cable as they only consume the data stream and do not forward it to other customers. In the case of LAN-Force$_\mathrm{a}$, these customers are non-terminals of degree 1. These vertices, along with the incident edges, can be deleted using NTD_1. In the context of LAN-Force$_\mathrm{h}$, these customers are terminals. Since the cost of the connection rarely exceeds the profit of the customer, we cannot delete all these vertices.
The Least Cost reduction technique could not delete vertices or edges in any instance. This suggests a strict Euclidean geometry of the underlying instances. In spite of their low success in elimination, the running time is very high, because we calculate a lot of shortest paths. Since the Least Cost reduction techniques were not successful, we did not use them to calculate the results of the other instances in the extended Appendix.

AP	**2000 hh**		**4000 hh**		**8000 hh**	
	Basic	**Least**	**Basic**	**Least**	**Basic**	**Least**
	12.4	0.0	13.45	0.0	13.1	0.0
219878	9.67	0.0	12.11	0.0	12.51	0.0
	0.5691	17.932	5.4882	82.6322	15.0144	113.2843
	14.5	0.0	12.94	0.0	12.29	0.0
234333	12.26	0.0	11.22	0.0	11.43	0.0
	2.0737	21.8779	8.1869	43.0444	12.18	173.1428
	10.43	0.0	10.21	0.0	9.57	0.0
204993	7.28	0.0	7.26	0.0	7.42	0.0
	0.2582	21.3258	2.4181	75.4042	8.9434	344.6163
	9.53	0.0	10.67	0.0	12.03	0.0
48477	6.84	0.0	8.51	0.0	11.14	0.0
	0.7994	26.2232	4.3464	90.1162	11.7662	144.748
	12.09	0.0	9.69	0.0	10.35	0.0
51824	8.51	0.0	6.67	0.0	8.26	0.0
	0.3762	15.5008	1.4199	91.7088	14.2819	334.313
	10.5	0.0	8.7	0.0	11.44	0.0
156480	7.41	0.0	6.66	0.0	10.44	0.0
	0.7684	11.5424	2.7459	98.8422	15.2344	122.9728
	11.95	0.0	13.21	0.0	13.85	0.0
10230	8.65	0.0	12.39	0.0	13.32	0.0
	0.4312	12.6298	5.1524	29.1031	26.4044	68.5183
	11.09	0.0	17.73	0.0	18.63	0.0
16611	9.79	0.0	17.99	0.0	18.83	0.0
	0.8179	16.9848	10.2718	28.8586	12.7426	93.321
	13.52	0.0	15.85	0.0	14.39	0.0
20464	13.95	0.0	16.06	0.0	14.05	0.0
	1.3414	1.7068	6.2352	9.3582	19.5484	124.4866
	12.74	0.0	11.26	0.0	9.71	0.0
148815	8.74	0.0	7.86	0.0	7.39	0.0
	0.2523	25.9672	2.0971	85.5955	10.809	418.2519
	14.36	0.0	14.56	0.0	15.1	0.0
106879	11.83	0.0	13.67	0.0	14.9	0.0
	0.5743	6.363	5.345	39.5215	13.0266	46.0529
	19.49	0.0	18.79	0.0	16.04	0.0
1732	19.87	0.0	19.04	0.0	16.05	0.0
	3.0148	6.9647	8.453	9.0973	26.1559	80.6593
	12.07	0.0	9.49	0.0	10.21	0.0
138683	8.52	0.0	6.5	0.0	8.1	0.0
	0.3786	15.5248	1.182	64.0788	14.5661	379.5377
	11.73	0.0	10.86	0.0	9.1	0.0
54696	8.05	0.0	7.53	0.0	6.92	0.0
	0.3121	26.4534	2.0595	104.4235	7.9749	312.4983

Table 8: Column AP shows the vertex ID of the respective net cascade vertex. We show the reduction results of the instances with 2000, 4000 and 8000 households in two columns each. The left column shows the reduction results of the Basic Reductions, the right column the results of the Least Cost tests. The individual cells are divided as follows: top: reduction of vertices in percent, middle: reduction of edges in percent, bottom: runtime in seconds.

6.2.3 Optimisation

We tried to find exact and approximate solutions for all instances. As already described for the LAN-Force$_{\mathrm{a}}$ problem, the CUT model had such a poor performance that we do not include it in our results. We set the timeout to 1800 seconds for the SCF model. We calculated the results of the heuristic for different p_0 values. First, we calculated the results for p_0 values of 0, 0.25, 0.5 and 1. We then searched for the best of the values and calculated results with p_0 values of 0.125 and 0.375, 0.375 and 0.625 or 0.625 and 0.876, respectively. In total, we ran the heuristic 6 times to find our approximate result. Table 9 shows our results of the exact and the approximate approach.

hh	AP	Objective		Gap	Runtime		Gap
2000	219878	3798789.1	3656615.0	3.89	965.4257	140.8056	0.0
	234333	4804733.0	4683069.2	2.6	1304.1487	183.3329	0.0
	204993	6839913.2	2007016.2	240.8	1766.9850	53.2285	0.0
	48477	5137034.6	4952403.2	3.73	712.2089	164.5055	0.0
	51824	7970218.3	7842275.0	1.63	588.3427	143.6003	0.0
	156480	3965631.3	3758207.7	5.52	858.1451	131.9956	0.0
	10230	3561872.3	3364533.4	5.87	800.7214	119.4077	0.0
	16611	3007675.3	2869531.1	4.81	1800.1740	170.6053	1.67
	20464	2525999.5	2184067.9	15.66	1800.8304	225.0343	3.98
	148815	7300517.2	7176496.0	1.73	1094.3573	120.5455	0.0
	106879	3578311.6	3376909.7	5.96	1535.1603	114.1877	0.0
	1732	1705823.4	1462496.7	16.64	1801.4009	222.4234	6.54
	138683	7969931.5	7837194.8	1.69	641.0283	144.6620	0.0
	54696	7180230.3	7091946.4	1.24	652.9311	117.1139	0.0
4000	219878	5058297.2	4688613.6	7.88	1800.8703	805.2582	20.59
	234333	7702298.6	7436563.5	3.57	1800.8509	849.4487	9.51
	204993	10823047.5	10454674.3	3.52	1800.3131	620.9123	0.21
	48477	8755004.1	8419537.5	3.98	1800.8039	681.5662	0.2
	51824	14016527.1	13794805.9	1.61	1800.1710	695.7181	0.13
	156480	8689955.4	8331411.6	4.3	1800.8709	746.5275	0.45
	10230	4336059.1	3946246.0	9.88	1800.3794	648.1958	12.08
	16611	3417297.6	2780119.1	22.92	1800.3233	878.3433	31.96
	20464	2870210.8	2431200.5	18.06	1800.7864	6827.3779	40.45
	148815	12697735.5	12367303.1	2.67	1768.8471	656.8002	0.0
	106879	5342016.4	4745208.9	12.58	1800.9360	787.7965	6.57
	1732	2084975.6	1400671.5	48.86	1801.1003	1150.6802	52.04
	138683	13746260.6	13534061.3	1.57	1801.3840	599.0646	0.11
	54696	12875077.1	12549069.1	2.6	1800.6469	696.7401	0.2
8000	219878	7559172.6	6706470.8	12.71	1813.7476	2595.7773	37.54
	234333	12118391.5	9520020.2	27.29	1800.3756	3317.1079	9.56
	204993	21354462.0	20620324.8	3.56	1801.8771	2389.1778	2.46
	48477	11536075.0	9012123.2	28.01	1800.6169	3658.7403	10.61
	51824	21760679.0	20858755.7	4.32	1801.9606	2443.5632	0.62
	156480	11252591.7	8900893.2	26.42	1800.8558	2822.8079	10.97
	10230	7294258.9	6535942.2	11.6	1800.9802	3931.1522	23.5
	16611	7001499.5	6093441.5	14.9	1803.0610	2874.2913	40.35
	20464	7571964.6	6671450.6	13.5	1802.9042	3238.0974	26.88
	148815	21504740.7	20720452.2	3.79	1802.2731	2041.7087	2.67
	106879	6851615.2	4920008.3	39.26	1811.1961	2823.5661	32.0
	1732	7912994.4	6429567.4	23.07	1800.6932	3221.4042	16.77
	138683	21699153.4	20959333.4	3.53	1804.3361	2650.2332	0.9
	54696	21594978.6	20759413.4	4.02	1801.0600	2624.9058	1.15

Table 9: Column AP shows the vertex ID of the respective net cascade vertex. Column Objective shows the final costs of the fibre rollout, column Runtime the runtime of the solution in seconds. Both columns show the results of the exact strategy on the left and the results of the heuristic on the right. For the calculation of the exact solution we set a timeout to 1800 seconds. The Gap column shows the Gurobi Gap of the exact solution if no optimal result could be found before timeout. The results of the instances with 2000, 4000 and 8000 households are one below the other.

The heuristic was not able to find a better objective value than the exact approach in any instance.

The approximate solutions found in the instance with 2000 households were on average 10.21% worse than the exact objective values, in the instances with 4000 households it was 4.92% and with 8000 households 9.79%. We could not find any correlation between input graph size and solution quality. In all other instances, the heuristic was also at most 11% worse overall than the exact approach. While the heuristic consistently delivers good results when considered as a whole, stronger fluctuations are noticeable when considering individual net cascades. For example, the result of the heuristic for net cascade 204993 is 240.8% worse than the exact result. Such bad exceptional cases occur in each of our examined instances. For these instances, no good initial solution could be found in the initial separation phase. Increasing the number of separation phases can remedy this.
We observed a very large increase in the running time of the heuristic. While we needed on average 152 seconds for the instance with 2000 households, it was already 1185 seconds for 4000 households and 2902 seconds for 8000 households. This runtime is composed of all six iterative executions of the heuristic with different p_0 values. Each individual execution of the heuristic here takes longer than in the LAN-Force$_{\text{a}}$ problem, because instead of a few amplifier points we have to connect many customer vertices to the root via shortest paths. Therefore, many more cost-intensive shortest path calculations are performed. It is conceivable to shorten the runtime considerably by parallelisation. The longest runtimes (or largest Gurobi gaps) occurred in both the exact and approximate solutions in instances 20464 and 1732. These are the instances with the fewest amplifier points. We conclude that instances with few amplifier points are particularly difficult to compute because the heuristic takes a longer time to compute the shortest paths.

7 Outlook

In this thesis we have defined the two new problems LAN-Force$_{\text{a}}$ and LAN-Force$_{\text{h}}$ and explained methods to find optimal and approximate solutions for the two problems. In this chapter we want to mention possible approaches to solve the problems more efficiently in the future.
It is conceivable to find other reduction techniques. Apart from the reduction technique Collapse net cascade, we did not find any reductions specifically targeting the problems, but adapted different reductions for other already known graph problems to our problems. It may well be that there are better reduction techniques that specifically use properties of our new problems to eliminate vertices and edges. Due to time restrictions we have adapted the reduction techniques Voronoi and Dual Ascent only for the problem LAN-Force$_{\text{a}}$. An adaptation for the LAN-Force$_{\text{h}}$ should be possible. In the literature there are the reduction techniques Reachability [16],

Degree-l-test [11] and Bottelneck Steiner Distance [17] for the PCSTP. It is conceivable to adapt these reduction techniques to our problem.
For the exact solution of the problem we use the ILPs SCF model and CUT model. The performance of the CUT model was far worse than that of the SCF model. One could further increase the performance of the CUT model by directly adding several violated constraints in each callback instead of just one additional constraint. For this, it might be useful to use the Golberg's maximum flow algorithm [18] instead of the Ford-Fulkerson flow we use.
The heuristic we found gave unsatisfactory results especially for the LAN-Force$_\mathrm{h}$. Unfortunately, we were not able to obtain fractional solutions after the initial separation phases of the CUT model, so we filtered the vertices according to the distance to the next access point. To improve the performance of the heuristic, it might be worthwhile to sort the vertices according to their fractional value. To reduce the runtime of the heuristic, parallelising the execution of the heuristic for different p_0 values may be useful.
Since the problems LAN-Force$_\mathrm{a}$ and LAN-Force$_\mathrm{h}$ are new problems, there is still much to explore. We hope that we have laid a foundation with our work and that more research will be done on the problems in the future to find better solutions faster.

8 Conclusions

In this thesis we have introduced the two new problems on graphs Local Access Network Reinforcement problem for amplifier points and Local Access Network Reinforcement problem for households. The problems are interesting for planning telecommunication infrastructures and are continuations of the already known Prize-collecting Local Access Network problem [4]. We find techniques to divide a given input graph into smaller planning units. We then describe reduction techniques to further reduce the planning units and delete vertices and edges that are not relevant for the solution. We explain two different Integer Linear Programs to find an optimal solution. Since finding an optimal solution on some instances takes a long time, we develop heuristics for finding an approximate solution for both problems. We test our methods on real instances of Vodafone GmbH. Our experiments show that especially the simple Basic Reduction techniques have a great success. Furthermore, our heuristics find good results overall in a short time.

List of Figures

List of Tables

References

[1] Andreas Bley, Ivana Ljubić, and Olaf Maurer. “Lagrangian decompositions for the two-level FTTx network design problem”. In: *EURO Journal on Computational Optimization* 1.3-4 (2013), pp. 221–252.

[2] Martin Grötschel, Christian Raack, and Axel Werner. “Towards optimizing the deployment of optical access networks”. In: *EURO Journal on Computational Optimization* 2.1 (2014), pp. 17–53.

[3] Stefan Voß. “Steiner tree problems in telecommunications”. In: *Handbook of optimization in telecommunications*. Springer, 2006, pp. 459–492.

[4] Ivana Ljubić, Peter Putz, and Juan-José Salazar-González. “A MIP-based approach to solve the prize-collecting local access network design problem”. In: *European Journal of Operational Research* 235.3 (2014), pp. 727–739.

[5] Richard M Karp. “Reducibility among combinatorial problems”. In: *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[6] David S Johnson, Maria Minkoff, and Steven Phillips. “The prize collecting steiner tree problem: theory and practice”. In: *SODA*. Vol. 1. 0.6. 2000, p. 4.

[7] Edsger W Dijkstra. “A note on two problems in connexion with graphs”. In: *Numerische mathematik* 1.1 (1959), pp. 269–271.

[8] EA Dinic. “Algorithm for solution of a problem of maximum flow in a network with power estimation, soviet math. doll. 11 (5), 1277-1280,(1970)”. In: *English translation by RF. Rinehart* (1970).

[9] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. “A nearly-linear time framework for graph-structured sparsity”. In: *International Conference on Machine Learning*. PMLR. 2015, pp. 928–937.

[10] Daniel Rehfeldt, Thorsten Koch, and Stephen J Maher. “Reduction techniques for the prize collecting Steiner tree problem and the maximum-weight connected subgraph problem”. In: *Networks* 73.2 (2019), pp. 206–233.

[11] Ivana Ljubic et al. “Solving the prize-collecting Steiner tree problem to optimality”. In: *ALENEX/ANALCO* 2005 (2005), pp. 68–76.

[12] Markus Leitner et al. “A dual ascent-based branch-and-bound framework for the prize-collecting steiner tree and related problems”. In: *INFORMS journal on computing* 30.2 (2018), pp. 402–420.

[13] Thomas Pajor, Eduardo Uchoa, and Renato F Werneck. “A robust and scalable algorithm for the Steiner problem in graphs”. In: *Mathematical Programming Computation* 10.1 (2018), pp. 69–118.

[14] Laurence A Wolsey and George L Nemhauser. *Integer and combinatorial optimization*. Vol. 55. John Wiley & Sons, 1999.

[15] Peter Elias, Amiel Feinstein, and Claude Shannon. “A note on the maximum flow through a network”. In: *IRE Transactions on Information Theory* 2.4 (1956), pp. 117–119.

[16] Cees Duin. “Preprocessing the Steiner problem in graphs”. In: *Advances in Steiner Trees*. Springer, 2000, pp. 175–233.

[17] Eduardo Uchoa. “Reduction tests for the prize-collecting Steiner problem”. In: *Operations Research Letters* 34.4 (2006), pp. 437–444.

[18] Andrew V Goldberg, Éva Tardos, and Robert Tarjan. *Network flow algorithm*. Tech. rep. Cornell University Operations Research and Industrial Engineering, 1989.

A SCF model for PC-LAN

$$\max \sum_{k\in K} p_k y_k - \sum_{a\in A}\sum_{n\in N_a} c_{a,n} x_{a,n}$$

subject to

$$\begin{aligned}
\sum_{a\in\delta^+(v)} f_a - \sum_{a\in\delta^-(v)} f_a &= \begin{cases} -d_v y_v, & \text{if } v\in K \\ \sum_{k\in K} d_k, & \text{if } v = r \\ 0, & \text{otherwise} \end{cases} && \forall v\in V \\
0 \le f_a &\le \sum_{n\in N_a} u_{a,n} x_{a,n} && \forall a\in A \\
\sum_{k\in K} p_k y_k &\ge p_0 \\
\sum_{n\in N_a} x_{a,n} &\le 1 && \forall a\in A \\
f_a &\in \mathbb{R}^+ && \forall a\in A \\
x_{a,n} &\in \{0,1\} && \forall a\in A, \forall n\in N_a \\
y_v &\in \{0,1\} && \forall v\in V
\end{aligned}$$

In order to describe the network flow correctly, each undirected edge $e = \{u, v\}$ must be converted into the two directed edges $a_{uv} = (u, v)$ and $a_{vu} = (v, u)$. The edges keep their respective modules.

B SCF model for LAN-Force$_\mathrm{h}$

$$\max \sum_{k \in K} \sum_{a_f \in \delta^-(k)} p_{k_f} x_{a_f} + \sum_{k \in K} \sum_{a_c \in \delta^-(k)} p_{k_c} x_{a_c} - \sum_{a \in A} c_a x_a$$

subject to

$$\begin{aligned}
\sum_{a \in \delta^+(v)} f_a - \sum_{a \in \delta^-(v)} f_a &= \begin{cases} -d_v y_v, & \text{if } v \in K \\ \sum_{k \in K} d_k, & \text{if } v = r \\ 0, & \text{otherwise} \end{cases} && \forall v \in V \\
\sum_{a_f \in \delta^+(v)} f_{a_f} - \sum_{a_f \in \delta^-(v)} f_{a_f} &\leq 0 && \forall v \in V \setminus \{r\} \\
0 \leq f_a &\leq u_a x_a && \forall a \in A \\
x_a &\leq f_a && \forall a \in A \\
0 &\leq (\sum_{k \in K} d_k) x_a - f_a && \forall a \in A \\
\sum_{a \in \delta^-(k)} x_a &\leq 1 && \forall k \in K \\
f_a &\in \mathbb{R}^+ && \forall a \in A \\
x_a &\in \{0,1\} && \forall a \in A \\
y_v &\in \{0,1\} && \forall v \in V
\end{aligned}$$

In order to describe the network flow correctly, each undirected edge $e = \{u, v\}$ must be converted into the two directed edges $a_{uv} = (u, v)$ and $a_{vu} = (v, u)$. The edges retain their respective capacities and costs. Furthermore, we have to insert an artificial root r as described in Chapter 2.2.

C Dual Ascent example

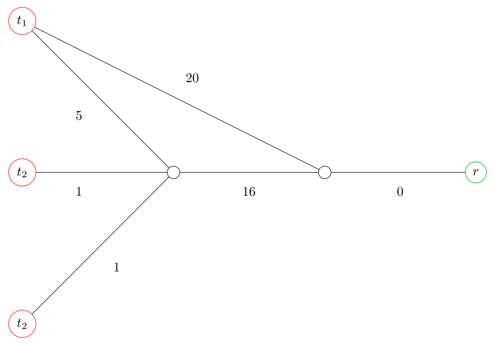


Figure 17: The RSTP input graph

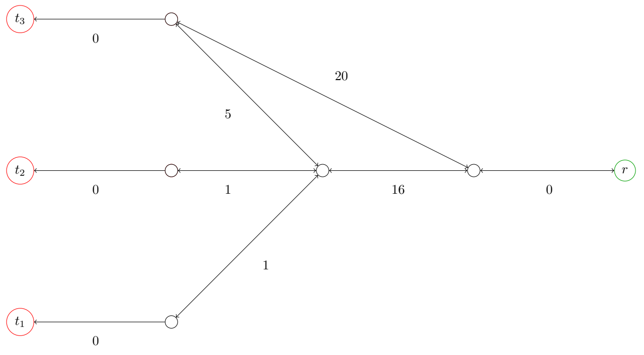


Figure 18: Preprocessing step: the input graph becomes directed. Every terminal is only connected by an incoming edge.

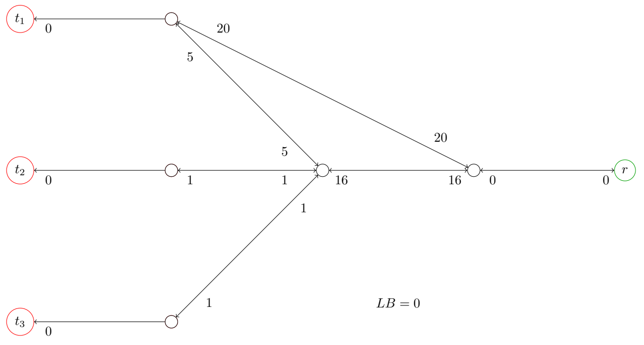


Figure 19: Initialization: the terminals become active terminals. For every edge we introduce reduced costs. We set the lower Bound to 0.

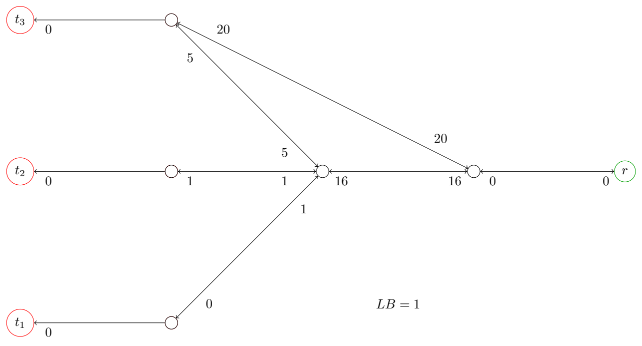


Figure 20: First iteration of the main loop. We try to gradually increase the active component of t_1. An edge with a cost of 1 limits the active component. We set the reduced costs of the edge to 0 and increase LB by 1.

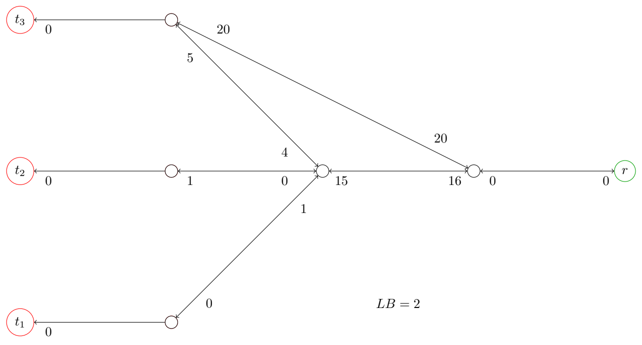


Figure 21: We continue to expand the active component of t_1.

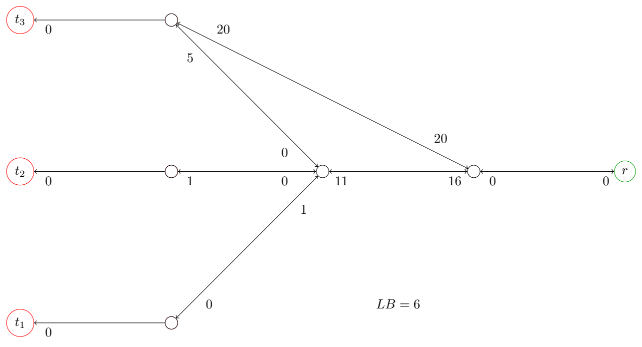


Figure 22: We continue to expand the active component of t_1.

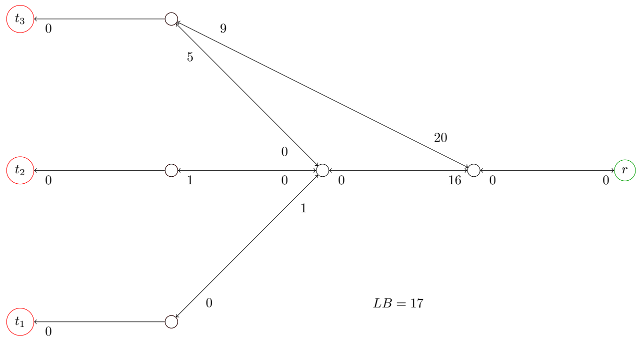


Figure 23: We continue to expand the active component of t_1. After we increase the lower bound by 11, there is a path from t_1 to the root in the saturation graph. Terminal t_1 is thus connected to the root and we can delete it from the list of active terminals.

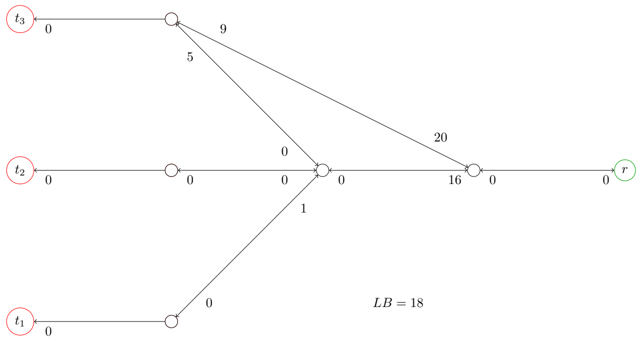


Figure 24: We choose t_2 as the next active terminal and begin to expand its active component. After we have extended the active component by only one edge, it is already possible to reach the root starting from t_2 in the saturation graph. Terminal t_2 therefore becomes inactive and we chose t_3 as the next active terminal.

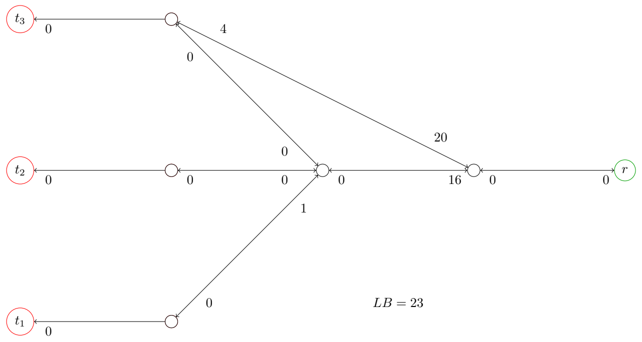


Figure 25: After we have extended the active component by only one edge, it is already possible to reach the root starting from t_3 in the saturation graph. Terminal t_3 therefore becomes inactive. Since there is no active terminal left we end our search. We return our final LB value of 23.

D Modified Dijkstra algorithm

Algorithm 4: We extend the Dijkstra shortest path search. We use for each vertex u the dictionary *coax* to keep track of whether the vertex is part of a coax or fibre link. If u is part of a fibre link, its neighbour v is only considered if there is a fibre edge between u and v.

```
Data: $G = (V, A)$, source
Result: Table with all distances
$dist[source] = 0$
$parent[source] = None$
$coax[source] = True$
$Q.put(source)$
while $Q \neq \emptyset$ do
    $u$ = vertex in $Q$ with smallest cost
    remove $u$ from $Q$
    foreach $v \in neighbours(u)$ do
        if $capacity(u, v) == coaxCapaxity$ then
            if $coax[u] == False$ then
                continue
            $coax[v] = True$
        else
            $coax[v] = False$
        $alt = dist[u] + distance\_between(u, v)$
        if $alt < dist[v]$ then
            $dist[v] = alt$
            $parent[v] = u$
            $Q.put(v)$
return $dist$
```