MASTER

Capacitated Facility Location Problem on Graphs of Bounded Treewidth

Hospel, Roel Alexander

Award date:
2021

Department of Mathematics and Computer Science
Algorithms Group

Capacitated Facility Location Problem on Graphs of Bounded Treewidth

Master Thesis

Roel Alexander Hospel

Supervisors:
dr. Bart MP Jansen
ir. Flip van der Valk
ir. Arjan van den Boogaart

Eindhoven University of Technology, November 2021

Abstract

This thesis looks at the problem of planning fiber-optics networks. The research was done in collaboration with *ThePeopleGroup*, where the company is searching for alternative methods in network calculations in order to deliver better results to their respective customers.

We consider a graph $G = (V, E)$ of potential edges, and a set of users/clients $U \subseteq V(G)$. We wish to place facilities on the vertex set $V(G)$, that can each satisfy up to cap_{DP} user-demand, and to determine what edges $E(G)$ to include such that there exists a path between the appropriate facilities and users.
The cost of such a solution consists of three factors: the cost of constructing facilities, the cost of digging trenches along included edges, and the cost of each cable between facility and user.
A cost-efficient solution balances the cost of digging and equipment cost, so that the total cost is minimized.

The planning of fiber-optics networks is closely related to the Capacitated Facility Location Problem (CFLP), which we know to be NP-hard on general and planar graphs. There exist algorithms to help with automating the planning of these networks.
One such algorithm, which has been developed by A. van den Boogaart[1] in collaboration with *ThePeopleGroup*, involves reducing the solution space to a Steiner tree on the general graph, and then solving CFLP on the resulting Steiner tree in polynomial time. This method may be fast, but the step of selecting which edges to include in the Steiner tree is completely blind to the subsequent cost of constructing facilities and placing cables. This means that we may end up with sub-optimal results in terms of total cost.

The goal of this paper is to seek an alternative method that finds solutions that are more cost-efficient than the algorithm based on Steiner trees. We approach this goal by widening our solution space. Instead of merely considering a tree on the general graph, we can significantly increase cost efficiency if we considered a tree-like graph that includes appropriate additional "edges of interest".

To this end we propose a fixed parameter tractable algorithm for CFLP on graphs of a bounded treewidth k. We have implemented a variant of this algorithm to serve as a proof-of-concept that we can build a solution on the tree decomposition of a graph.
The results of the algorithm in this paper are very promising. However, more work is still needed to improve the implementation of the algorithm, so that it is fast enough to run on larger datasets.

Contents

1 Introduction **1**
- 1.1 Background and Motivation . . . 2
  - 1.1.1 Problem Description . . . 2
  - 1.1.2 Existing Approach using Steiner Trees . . . 3
  - 1.1.3 New Approach on Graphs of Bounded Treewidth . . . 5
- 1.2 Results . . . 5
- 1.3 Related Work . . . 6
  - 1.3.1 Fiber-Optics Network Planning using Steiner Trees . . . 6
  - 1.3.2 Facility-Location Problem . . . 6
  - 1.3.3 Minimal Steiner Tree Problem . . . 7
- 1.4 Organization . . . 8

2 Preliminaries **9**
- 2.1 Graph Theory . . . 9
  - 2.1.1 Basic Definitions . . . 9
  - 2.1.2 Special Graphs . . . 9
- 2.2 Tree Decomposition and Treewidth . . . 10
  - 2.2.1 Tree Decomposition . . . 10
  - 2.2.2 Treewidth . . . 11
- 2.3 Facility Location Problems . . . 12
  - 2.3.1 p-Median Problem . . . 12
  - 2.3.2 Facility Location Problem . . . 13
  - 2.3.3 Facility Location / Network Design Problem . . . 14
- 2.4 Steiner Tree Problem . . . 16

3 Problem Description **17**
- 3.1 Problem Definition . . . 17
- 3.2 Subproblem Definition . . . 18
- 3.3 Elaboration . . . 19

4 Algorithm Description **20**
- 4.1 Description of the Algorithm . . . 20
  - 4.1.1 Overall Strategy . . . 21
  - 4.1.2 ”Introduce Edge”-Node . . . 22
  - 4.1.3 ”Introduce Vertex”-Node . . . 29
  - 4.1.4 ”Forget Vertex”-Node . . . 32
  - 4.1.5 ”Join Sub-Trees”-Node . . . 33
  - 4.1.6 ”Leaf”-Node . . . 37
  - 4.1.7 Examples . . . 38
- 4.2 Trim Function . . . 43
  - 4.2.1 Goal of the Trim Function . . . 43
  - 4.2.2 Trim Function . . . 44

4.2.3 Solution Bound . . . 44
4.2.4 Correctness of the Trim Function . . . 45
4.3 Running Time . . . 46

5 Evaluation & Discussion **47**
5.1 Implementation of the Algorithm . . . 47
5.2 Evaluation of Results . . . 49
5.2.1 Experimental Setup . . . 49
5.2.2 General Synthetic Example . . . 50
5.2.3 Specific Synthetic Examples . . . 52
5.3 Discussion of Algorithm . . . 56
5.3.1 Limitations of Algorithm . . . 57
5.3.2 Future Work . . . 58

6 Conclusion **59**

Appendix **62**

A Linear Program **62**
A.1 Mixed-Integer Linear Programming Definition . . . 62
A.1.1 Shortest Path Problem . . . 62
A.1.2 Shortest Path Between Multiple Sources and Sinks . . . 63
A.1.3 Capacitated Facility-Location Problem . . . 65
A.1.4 Capacitated Facility-Location Problem with Distribution-Facilities . . . 67
A.1.5 Comparison Between Edge Capacity-Based and Path-Based MILP . . . 70

Chapter 1

Introduction

The internet is a global system of interconnected computer networks. Many different mediums can be used to connect up these computer networks with one another, ranging from phone-lines to coaxial cables to cellular networks. Currently the most future-proof of these mediums is the fiber-optic cable, which allows increased speeds and bandwidth compared to its contemporaries.

With the demand for speed and bandwidth rising, internet service providers are rolling out fiber-optic networks to increasingly greater numbers of homes and businesses. Planning out these networks in such a way that they balance out efficiency and cost can overall be a complicated and time-consuming task.

Traditionally these networks would be drawn out by hand, by an engineer. However, since they contain a lot of individual client/user-connections, this means that calculating such a network entirely by hand is rather slow and inefficient. Preferably we'd like to use an algorithm to calculate much of the network, so that the engineer can focus his attention on the more critical aspects of planning out such a network.

ThePeopleGroup is a company that is involved in planning underground infrastructure, and has previously developed algorithms using Steiner trees to aid in the calculation of these fiber-optic networks. While these algorithms are great at quickly calculating decently efficient networks, there still exist some common inefficiencies which need to be manually evaluated and fixed afterwards by the engineer.

The goal of this thesis is to explore alternative methods that could be used for calculating these fiber-optic networks, such that these common inefficiencies would occur less often.

1.1 Background and Motivation

This section we will give a basic overview of the problem we are aiming to solve, followed by a basic description of the pre-existing approach that is currently in use by *ThePeopleGroup*, as well as an explanation why this approach does not necessarily find optimal solutions. We then describe the motivation behind our alternative approach.

1.1.1 Problem Description

Fibre-optics networks consist of several different hierarchical levels, that need to be connected up with one another:

- Fibre Termination Units (FTU):
  FTUs are the places where a fibre-connection enters the building of a user/client (e.g. individual homes and offices).
- Passive Distribution Points (DP):
  DPs are the local, passive distribution points that serve a number of FTUs within a specific neighbourhood. Typically these can serve up to 48 FTUs.
- Active Area Points of Presence (AP):
  APs are larger, active distribution point that serve several DPs within an area. APs are responsible for routing packets. Typically these are capable of serving 40 fully used DPs
- City Points of Presence (CP):
  CPs are the highest level in the network hierarchy, and serve as the connection between APs and the wider internet "backbone".

When planning out a fiber-optics network, typically we already know the location and demand of users/clients, and therefore also their respective FTUs. Our goal is to plan out the location for the DPs, APs (and potentially CPs), as well as the routes that the cables take to connect these components.

Each constructed DP, AP and CP has a construction cost associated to it. The cost of connecting these components with one another is split between the cost of digging trenches and the cost of each individual cable that is run through these edges.
When planning out this network we wish to meet all required capacities, while minimizing the total cost of construction as much as possible.

The problem of planning out fiber-optics networks is part of the family of facility-location problems. More specifically, it is a version of the Capacitated Facility Location Problem (CFLP), which we know to be NP-hard on regular as well as on planar graphs. This means that we cannot realistically solve the problem by comparing all potential solutions, and need to be more deliberate in our approach.

For the scope of this thesis we will focus on the low-level infrastructure, meaning that we limit ourselves to the placement of DPs and FTUs (facilities and users/clients). The higher-level infrastructure of placing APs and CPs falls outside of the scope of this thesis.

1.1.2 Existing Approach using Steiner Trees

The paper "Efficient Computation of Fiber-Optic Networks" by A. van den Boogaart [1] describes the algorithm that is currently used by *ThePeopleGroup* to efficiently calculate such fiber-optics networks. It relies on the property that the facility location problem can be solved in polynomial time if the underlying graph is a tree.

This approach first converts a geographical map to a simpler model graph, where each edge denotes a potential location to dig and place cables.

Calculating the fiber-optics network happens in two main steps:

(1) First, we approximate an optimal Steiner minimal tree on the general graph, with the set of users/clients as terminals. This step must be approximated, since the Steiner tree problem is also an NP-hard problem.

(2) Then we solve the facility location problem on the Steiner tree.
Any edges on the general graph that are not part of the Steiner tree are ignored.

Both of these steps can be solved in polynomial time, leading to an efficient algorithm that is able to generate a decent solution to the facility location problem in $O(k \cdot n^2)$ time, where k denotes the maximum facility capacity.

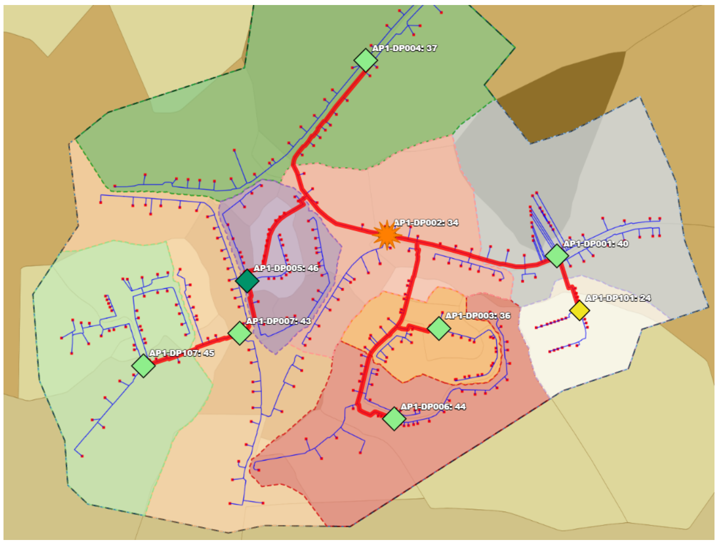


Figure 1.1: Example network generated by the current approach using Steiner trees

Shortcomings of this Approach

While the solution to the facility location problem that is calculated on the Steiner tree is optimal for that Steiner tree, there is no guarantee that that solution is also optimal on the underlying overall graph.

The step in which the Steiner tree is calculated is blind to any decisions that will be made by subsequent facility-location step. The only metric that can be used to determine the quality of the Steiner tree is the cost of digging trenches. The cost of placing facilities, and the cost of laying cables between facility and user/client is ignored.

The facility location solver on trees can only consider edges that are part of that tree, and cannot consider edges that were not included in the Steiner tree, even if they may lead to more optimal solutions when accounting for cable- and facility-cost

There are a couple of common culprits for sub-optimal solutions, that stem from this inability to deviate from the pre-determined underlying Steiner tree.

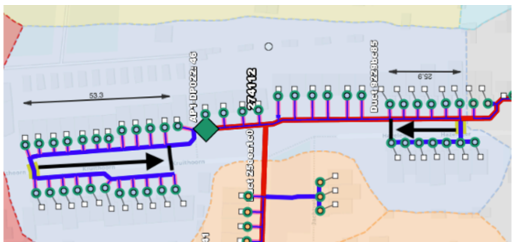


Figure 1.2: Example of a sub-optimal solution using Steiner trees

Figure 1.2 depicts an example of the "inefficient crossing"-scenario, where the Steiner tree has pre-determined a cheap place to cross the street, which is optimal if you account for digging costs, but might not be optimal if you also need to account for the cost of placing facilities and cables.

In this example moving the crossing to the closer side of the street, from the perspective of the facility, means that we save on cable-length by the entire length of the street for each house on the opposite side of the street.

Inefficient Cover of Distribution Points

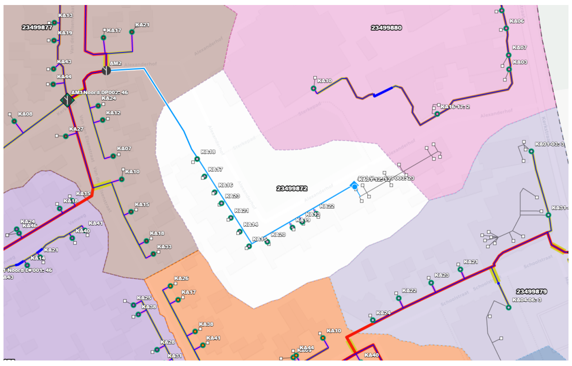


Figure 1.3: Example of a sub-optimal solution using Steiner trees

Figure 1.3 depicts an example of the "inefficient facility mapping"-scenario, where some facilities are not fully utilized to their maximum capacity. This happens because DP coverage areas are not allowed to share edges.

The existing approach requires the graph on which the facility location problem is solved to be a tree. Introducing new potential edges to be included as considerations would introduce cycles into the tree, which breaks the properties of a tree. It would be very useful if we could somehow efficiently solve the facility location problem on certain graphs that are not trees, but are somewhat tree-like.

1.1.3 New Approach on Graphs of Bounded Treewidth

For any graph we can make a tree decomposition, which allows us to solve problems such as the facility location problem on the tree instead of on the underlying graph. However, the efficiency of the tree decomposition algorithm is dependent on the size of the biggest bag in that tree decomposition.

Treewidth is a parameter that helps indicate how tree-like any given graph is. It is commonly used to determine the parameterised complexity of graph algorithms. For a graph of treewidth k there must also exist a tree decomposition with a maximum bag size $k + 1$

This thesis looks into the feasibility of using the concepts of treewidth and tree decomposition to help solve the CFLP problem in polynomial time, on a pre-determined tree-like graph that may be based on the Steiner tree, but includes additional edges of interest.

1.2 Results

In this thesis we present a fixed parameter tractable algorithm to optimally solve CFLP on the tree decomposition of a graph with a given treewidth k. If we assume that the treewidth, and the maximum facility capacity are bounded by a given value, then this algorithm able to find an optimal solution in polynomial time.

Given specific circumstances, our tree decomposition based approach is able to find solutions that are significantly more optimal than the solutions found by the Steiner tree based approach. It is however important to note that these improvements are largely dependent on the additional potential edges that we choose to include in the graph of bounded treewidth.

The worst-case running time of the algorithm can be defined as:

$$\mathrm{O}\left(|V(T)|^{O(1)} \cdot \left(B_{k+1} \cdot (2 \cdot \mathrm{cap}_{\mathrm{DP}} + 1)^{k+1}\right)^2 \cdot (k+1)^2\right)$$

Here $\mathrm{cap}_{\mathrm{DP}}$ denotes the maximum capacity of any facility, and k denotes the treewidth of the input graph. If we assume that both $\mathrm{cap}_{\mathrm{DP}}$ and k are bounded, then the running time is entirely dependent on the size of the decomposition tree.

1.3 Related Work

This section shortly discusses some relevant work in the area of fiber-optics network planning. In subsection 1.3.1 we provide further details on the paper that this thesis is a continuation on. In subsection 1.3.2 we reference some relevant work on the more generalized facility location problem. And in subsection 1.3.3 we list some more relevant work on the area of calculating minimal Steiner trees.

1.3.1 Fiber-Optics Network Planning using Steiner Trees

This thesis is a continuation on the paper ”Efficient Computation of Fiber-Optic Networks” by A. van den Boogaart [1]. The paper describes an approach to generate a fiber-optics networks from publicly available geographic data, using Steiner trees to speed up computation.

Using the available geographic data, a graph representation of potential edges is generated. The fiber-optics network is then generated, on this graph representation, in two main steps:

(1) Approximate a Steiner minimal tree, with the set of users/clients as terminals.

(2) Calculate an optimal fiber-optics network on the Steiner minimal tree.

The paper already discusses some issues with this last step of the fiber-optics network calculation. While the fiber-optics network that is calculated on the Steiner minimal tree is optimal for that Steiner minimal tree, it may lead to suboptimal solutions on the general graph. This is because when calculating the Steiner minimal tree, certain edges that might lead to an optimal solution for the fiber-optics network on the general graph get discarded prematurely.

In this thesis we use tree decomposition to include some of these edges that would otherwise have been discarded, leaving us with a potentially more optimal solution.

1.3.2 Facility-Location Problem

Planning an optimal fiber-optics network, in which we wish to optimally connect a set of users/clients to a set of (potential) distribution points/facilities, has a close similarity to the more general family of facility-location problems.

There exist several versions of the facility location problem, each of which aim to solve slightly different but ultimately similar problems. Some relevant ones are:

- The p-Median problem is at the core of the facility location problem family. It was first described by S.L. Hakimi in 1964-1965 [2][3], and later shown to be NP-hard on general and planar graphs by O. Kariv and S.L. Hakimi in 1979 [4]

- The Facility-Location Problems (FLP) are somewhat extended versions of the p-Median problem, for which there exists both an uncapacitated version (UFLP) and a capacitated version (CFLP). FLP was first described and shown to be NP-hard on general graphs, by G. Cornuejols, G.L. Nemhauser, and L.A. Wolsey in 1983 [5].

- The Facility Location / Network Design Problems (FLNDP) are somewhat extended versions of FLP, for which there exists both an uncapacitated version (UFLNDP) and a capacitated version (CFLNDP). FLNDP was first described and shown to be NP-hard by S. Melkote and M.S. Daskin in 1993, 2001 [6][7]

Each of these versions of the problem have been shortly elaborated in chapter 2, Preliminaries.

Algorithms on General Graphs

The family of facility location problems is known to be NP-hard on general graphs. This means that, if we assume that P $\neq$ NP, there does not exist any algorithm capable of solving the facility location problem on general graphs in polynomial time.

A 2006 bibliography by J. Reese [8] summarizes the literature on solution methods for problems in facility location problem family

Approximation Algorithms A 1997 paper by D.B. Shmoys, É. Tardos, and K. Aardal [9] details a 7-approximation algorithm to the facility location problem. This was later improved to a 5-approximation algorithm in 2004 by R. Levi, D.B. Shmoys, and C. Swamy [10].

Algorithms on Trees

Even though the facility location problem is known to be NP-hard on general graphs, it is possible to give polynomial time algorithms if we can assume certain properties to the underlying graph.

A.J. Goldman proved in 1971 [11] that the p-Median problem on trees is solveable in polynomial time. In 1996, A. Tamir provided an $O(pn^2)$ algorithm to the p-Median problem on trees [12]. The approach proposed by A. van den Boogaart [1] uses this principle to efficiently calculate an fiber-optics network on the pre-calculated Steiner tree.

Algorithms on Graphs of Bounded Treewidth

There has been some prior work on algorithms that aim to solve problems in the facility location family, given a graph of bounded treewidth. In 1989 D. Skorin-Kapov [13] described an $O(n^{k+2})$ algorithm for solving the Single Facility Location Problem on an n-vertex partial k-tree.

1.3.3 Minimal Steiner Tree Problem

The Minimal Steiner Tree problem was described by E.N. Gilbert and H.O. Pollak in 1968 [14], and is among the original set of NP-hard problems are described by R.M. Karp in 1972 [15]. In 1977 it was show by M.R. Garey, R.L. Graham, and D.S. Johnson that [16] that the Minimal Steiner Tree problem is also NP-hard on planar graphs.

The Minimal Steiner Tree problem is shortly elaborated in chapter 2, Preliminaries.

Algorithms on General Graphs

The minimal Steiner tree problem is known to be NP-hard or general graphs. This means that, if we assume that P $\neq$ NP, there does not exist any algorithm capable of solving the facility location problem on general graphs in polynomial time.

Exact Algorithms: For a long time, the fastest exact algorithm for the minimal Steiner tree problem was the algorithm proposed in 1971 by S.E. Dreyfus and R.A. Wagner [17], slightly improved upon by R.E. Erickson, C.L. Monma, and A.F. Veinott Jr. [18]
In 2006 a faster exact algorithm was proposed by D. Mölle, S. Richter, and P. Rossmanith [19].

Approximation Algorithms: L. Berman, L. Kou, and G. Markowsky proposed a fast 2-approximation to the minimal Steiner tree problem in 1981. [20]. It was proven in 2008 by M. Chlebíka and J. Chlebíková [21] that an approximation better than a $\frac{96}{95}$-approximation is NP-hard.

The current best approximation to the minimal Steiner tree problem is a 1.39-approximation as described by J. Byrka, F. Grandoni, T. Rothvoss, and L. Sanità in 2010 [22].

Algorithms on Graphs of Bounded Treewidth

Even though the minimal Steiner tree problem is known to be NP-hard on general graphs, it is possible to produce polynomial time algorithms if we can assume certain properties to the underlying graph.

In 2012 it was shown by M. Chimani, P. Mutzel, and B. Zey [23][24] that the minimal Steiner tree problem can be calculated in polynomial time on graphs of bounded treewidth.

1.4 Organization

The remainder of this thesis document is structured as follows:

In Chapter 2 we give a basic overview of some assumed preliminary knowledge. Specifically, we touch on the subjects of graph theory and tree decomposition, as well as the definitions of the facility location problem and the Steiner minimal tree problem.

In Chapter 3 we give a more exact definition of the problem that we are trying to solve, and how this translates to a sub-problem definition that we can use to build up a solution to the overall problem.

In Chapter 4 we describe in detail how the algorithm builds up towards an optimal solution along each node on a decomposition tree. We also describe how we can use a trim function to speed up the computation of the optimal solution by forgetting about partial solutions that are guaranteed to not be part of any optimal solution.

In Chapter 5 we evaluate the solutions that our algorithm provides on some synthetic as well as some realistic examples, and compare these to the solutions that are provided by the pre-existing approach using Steiner Trees [1]. We also discuss these findings and provide some angles for future work.
In Chapter 6 we summarize the findings of this paper.

As part of preliminary research for this paper, we have also looked into the feasibility of converting the problem to a Mixed Integer Linear Program, and implementing this in a MILP-solver such as SCIP. Sadly, this approach did not bear fruit, and we decided to shelve it and instead focus on the tree decomposition approach.
For the sake of completeness, we have included these MILP definitions as part of Appendix A

Chapter 2

Preliminaries

2.1 Graph Theory

2.1.1 Basic Definitions

An undirected graph G consists of a set of vertices $V(G)$ and a set of edges $E(G)$, where each edge $e \in E(G)$ is an unordered pair of vertices. An edge between u and v will be denoted as (u, v) and is equivalent to the edge (v, u), since the graph is undirected.

A path of length k between two vertices $u, v \in V(G)$ is a sequence of vertices $x = v_0, v_1, \ldots, v_k = y$ such that $(v_i, v_{i+1}) \in E(G)$ for all $0 \leq i < k$. A path is simple if all its vertices are distinct.
A graph is connected if there exists a path between any pair of vertices.

A cycle is a sequence of vertices $v_0, v_1, \ldots, v_k$ such that $v_0 = v_k$ and such that $(v_i, v_{i+1}) \in E(G)$ for all $0 \leq i < k$. A cycle is simple if all vertices are distinct except for $v_0 = v_k$.
A graph is acyclic if it does not contain a simple cycle.

For a vertex set S of a graph G, the sub-graph of G induced by S, denoted $G[S]$, is the graph on vertex set S and edge set $\{(u, v) \in E(G) \mid u \in S \wedge v \in S\}$.
A graph H is a sub-graph of graph G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$.

2.1.2 Special Graphs

A **tree** is a connected acyclic graph.
A **rooted tree** is a tree in which a designated vertex has been chosen as the root. The choice of the root uniquely determines parent/child relations in the tree in the natural way.

Given a set of terminal vertices $T \subseteq V(G)$ in a graph G, a **Steiner tree** on terminal set T is a sub-graph S of G such that S is a tree that contains all vertices of S.

A **planar graph** is a graph that can be embedded in the plane without crossing edges.
This means that there must exist a 2D representation of the graph such that edges only intersect at their respective vertices. This 2D representation is called an "embedding". This concept is particularly useful in geographical maps, or problems related to geographical maps.

An **outer-planar graph** is a stricter version of the planar graph, for which there must exist an embedding such that each vertex is part of the outer face. In other words, there must exist an embedding such that each vertex is reachable from the outside, without crossing over any edges.

2.2 Tree Decomposition and Treewidth

2.2.1 Tree Decomposition

The concept of a tree decomposition is defined as follows (Parameterized Algorithms, p.160 [25]):

A tree decomposition of a graph G is a pair $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$, where T denotes a rooted tree such that every node $t \in T$ is assigned a vertex subset $X_t \subseteq V(G)$, referred to as a bag, such that the following three conditions hold:

(T1) $\bigcup_{t \in V(T)} X_t = V(G)$.
In other words, for every vertex $v \in V(G)$ there exists at least one node $t \in T$ such that $v \in X_t$.

(T2) For every edge $(u, v) \in E(G)$, there exists at least one node $t \in T$ such that $u, v \in X_t$.

(T3) For every vertex $v \in V(G)$ the set $T_v = \{t \in V(T) : v \in X_t\}$ induces a connected sub-graph of T.
In other words, the set of nodes whose corresponding bags contain v, induces a connected sub-graph of T.

A tree decomposition $\mathcal{T}$ is defined to be a "nice" tree decomposition (with "introduce edge"-nodes) if and only if the following conditions are satisfied (Parameterized Algorithms, p.161 & p.168 [25]):

- $X_r = \emptyset$ and $X_l = \emptyset$ for the root r of tree T, and for every leaf l of tree T.
  In other words, all leaves as well as the root contain empty bags.
- Every node $t \in T$ is one of the following five types of nodes:
  - **"Introduce edge"-node**:
    A node $t \in T$ for which hold that:
    - node t is labelled with an edge $(u, v) \in E(G)$ such that $u, v \in X_t$, and
    - node t has exactly one child t', such that $X_t = X_{t'}$.

    We say that edge (u, v) is introduced at node t.
  - **"Introduce vertex"-node**:
    A node $t \in T$ with exactly one child t', such that $X_t = X_{t'} \cup \{v\}$ for some vertex $v \notin X_{t'}$. We say that vertex v is introduced at node t.
  - **"Forget vertex"-node**:
    A node $t \in T$ with exactly one child t', such that $X_t = X_{t'}/\{w\}$ for some vertex $w \in X_{t'}$. We say that vertex w is forgotten at node t.
  - **"Join sub-trees"-node**:
    A node $t \in T$ with exactly two children t_1, t_2, such that $X_t = X_{t_1} = X_{t_2}$
    We say that the sub-trees rooted at nodes t_1, t_2 are joined at node t.
  - **"Leaf"-node**:
    A node $t \in T$ with exactly zero children, such that $X_t = \emptyset$
- For each edge $e \in E(G)$ there exists exactly one node $t \in T$ such that e is introduced at t

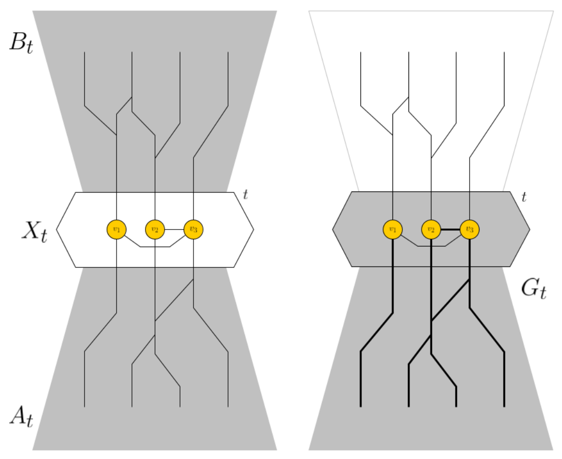


Figure 2.1: Vertex sets A_t, X_t, B_t and sub-graph G_t at some node $t \in T$

We define the tree $T_t \subseteq T$ to be the sub-tree of T rooted at node $t \in T$.
For each node $t \in T$ we can derive two additional vertex sets A_t, B_t, besides the given vertex set X_t, which are defined as follows (see figure 4.17):

- $A_t = \bigcup_{t' \in V(T_t)} \left(X_{t'}\right) \setminus X_t$
  In other words, the vertex set A_t contains all vertices that have been introduced in the sub-tree T_t rooted at node t, excluding the vertices in the vertex set X_t
- $B_t = V(G) - \bigcup_{t' \in V(T_t)} \left(X_{t'}\right)$
  In other words, the vertex set B_t contains all vertices that have not been introduced at any point in the sub-tree T_t rooted at node t.

With each node $t \in T$ of the tree decomposition we associate a sub-graph $G_t \subseteq G$ defined as follows (Parameterized Algorithms, p.168 [25]):

$$G_t = \big(V_t = X_t \cup A_t; E_t = \{e : e \text{ is introduced in the sub-tree rooted at } t\}\big)$$

The example as given in figure 4.17 shows the edges in $E(G_t)$, for some node $t \in T$, as bold.
In this example, at node t, the edge (v_2, v_3) has been introduced in the sub-tree rooted at t, while the edge (v_1, v_3) has not.

2.2.2 Treewidth

The treewidth of a graph is a fundamental tool used in various graph algorithms. It is a given that for any graph of a certain treewidth k, there must exist a tree decomposition such that the maximum bag size X_t for any node $t \in T$ is $k + 1$. [25]

For any graph for which there exists a tree decomposition with a maximum bag size $k + 1$, there also exists a ”nice” tree decomposition with a maximum bag size $k + 1$.

There is no guarantee on the treewidth of a general graph or a planar graph. However, if the graph is considered to be outer-planar, then the treewidth k of that graph is at most 2. [26]

2.3 Facility Location Problems

The family of facility location problems refers to a collection of optimization problems concerned with the optimal placement of facilities to satisfy demand of users/clients. What exactly defines such an optimal placement, and other constraints that may be in place, is largely defined by the specific flavour of facility location problem we are trying to solve.

Some relevant variants of the facility location problem are, but are not limited to:

- The p-Median Problem
- The Facility Location Problem (FLP), which can be split into:
  - Uncapacitated Facility Location Problem (UFLP)
  - Capacitated Facility Location Problem (CFLP)
- The Facility Location / Network Design Problem (FLNDP), which can be split into:
  - Uncapacitated Facility Location / Network Design Problem (UFLNDP)
  - Capacitated Facility Location / Network Design Problem (CFLNDP)

This section provides a short overview of each of these variants.

2.3.1 p-Median Problem

The p-Median problem aims to find a minimum cost solution, such that all user/client-demand has been satisfied by building exactly p facilities. It can be considered the core problem of the family of facility location problems.

Input: We consider two sets:

F	The set of potential facilities that can be constructed
U	The set of users/clients for which we need to satisfy the demand

For each user $u \in U$ we know a value $d(u)$ denoting the total demand that needs to be satisfied to that user. For each pair $f \in F, u \in U$ we know the cost $\text{cost}(f, u)$ of serving exactly one unit of demand from facility f to user/client u.

Goal: Find a minimum-cost solution, such that the demand $d(u)$ of each user $u \in U$ has been satisfied, and exactly p facilities have been built.

$$\begin{array}{lll}
\textbf{Minimize:} & \sum_{f \in F} \sum_{u \in U} (d(u) \cdot \text{cost}(f,u) \cdot x_{f,u}) & \\
\textbf{Subject to:} & \sum_{f \in F} (x_{f,u}) = 1 & \text{For each user } u \in U. \\
& \sum_{f \in F} (y_f) = p & \\
& x_{f,u} - y_f \leq 0 & \text{For each user } u \in U \text{, each facility } f \in F. \\
& x_{f,u} \geq 0 & \text{For each user } u \in U \text{, each facility } f \in F. \\
& x_{f,u} \in \{0,1\} & \text{For each user } u \in U \text{, each facility } f \in F. \\
& y_f \in \{0,1\} & \text{For each facility } f \in F.
\end{array}$$

Decision Variables: We consider the following decision variables:

$x_{f,u}$	Denotes whether user u gets supplied by facility f
y_f	Denotes whether facility f gets constructed

It is important to note that in the p-Median variant of the facility location problem, facilities do not have a cost of construction associated to them, nor do they have a maximum capacity.

2.3.2 Facility Location Problem

The Facility Location Problem (FLP) can be considered an extension on the p-Median problem. Each potential facility that we can construct now has a cost associated to it.

There exist two variants of FLP:

- Uncapacitated Facility Location Problem (UFLP)
- Capacitated Facility Location Problem (CFLP)

Both of these problems are fairly similar. Their main difference is whether a facility can exceed a set maximum capacity.

Input: We consider two sets:

F	The set of potential facilities that can be constructed
U	The set of users/clients for which we need to satisfy the demand

For each facility $f \in F$ we know the cost $\text{cost}_{\text{facility}}(f)$ of constructing that facility. For each pair $f \in F, u \in U$ we know the cost $\text{cost}_{\text{serves}}(f,u)$ of having the demand of user u serviced by facility f.

In the CFLP variant of the problem we also know a demand $d(u)$ for each user $u \in U$, and a maximum capacity $\text{cap}_{\text{facility}}(f)$ for each facility $f \in F$.

Uncapacitated Facility Location Problem

Goal: Find a minimum-cost solution, such that the each user $u \in U$ is served by exactly one facility $f \in F$.

$$\begin{array}{lll}
\textbf{Minimize:} & \sum_{f \in F} (\text{cost}_{\text{facility}}(f) \cdot y_f) + \sum_{f \in F} \sum_{u \in U} (\text{cost}_{\text{serves}}(f,u) \cdot x_{f,u}) & \\
\textbf{Subject to:} & \sum_{f \in F} (x_{f,u}) = 1 & \text{For each user } u \in U. \\
& x_{f,u} - y_f \leq 0 & \text{For each user } u \in U\text{, each facility } f \in F. \\
& x_{f,u} \in \{0,1\} & \text{For each user } u \in U\text{, each facility } f \in F. \\
& y_f \in \{0,1\} & \text{For each facility } f \in F.
\end{array}$$

Decision Variables: We consider the following decision variables:

$x_{f,u}$	Denotes whether user u is serviced by facility f
y_f	Denotes whether facility f gets constructed

Capacitated Facility Location Problem

Goal: Find a minimum-cost solution, such that the demand of each user $u \in U$ has been satisfied, and the maximum capacity of each facility $f \in F$ is not exceeded.

$$
\begin{array}{lll}
\textbf{Minimize:} & \sum\limits_{f \in F} (\mathrm{cost}_{\mathrm{facility}}(f) \cdot y_f) + \sum\limits_{f \in F} \sum\limits_{u \in U} (\mathrm{cost}_{\mathrm{serves}}(f,u) \cdot x_{f,u}) & \\
\textbf{Subject to:} & \sum\limits_{f \in F} (x_{f,u}) = 1 & \text{For each user } u \in U. \\
 & \sum\limits_{u \in U} (d(u) \cdot x_{f,u}) \leq \mathrm{cap}_{\mathrm{facility}}(f) \cdot y_f & \text{For each facility } f \in F. \\
 & x_{f,u} \in \{0,1\} & \text{For each user } u \in U\text{, each facility } f \in F. \\
 & y_f \in \{0,1\} & \text{For each facility } f \in F.
\end{array}
$$

Decision Variables: We consider the following decision variables:

$x_{f,u}$	Denotes whether user u is serviced by facility f
y_f	Denotes whether facility f gets constructed

It is important to note that in the FLP variant of the facility location problem, the cost of satisfying demand to one user is entirely independent to the cost of satisfying demand to another user.

2.3.3 Facility Location / Network Design Problem

The Facility Location / Network Design Problem (FLNDP) can be considered an extension of the Facility Location Problem (FLP). Instead of assuming that a connection between a user and a potential facility is entirely independent from other such connections, these connections are allowed to share edges in an underlying network of potential edges.

There exist two variants of FLNDP:

- Uncapacitated Facility Location / Network Design Problem (UFLNDP)
- Capacitated Facility Location / Network Design Problem (CFLNDP)

Both of these problems are fairly similar. The main difference is whether a facility can exceed a set maximum capacity.

Input: We consider a graph $G = (V, E)$.

For each vertex $v \in V(G)$ we know the cost $\mathrm{cost}_{\mathrm{facility}}(v)$ of constructing a facility on that vertex. We also know the demand $d(v)$ of any user that might be on that vertex $v \in V$

For each edge $(u, v) \in E(G)$ we know the cost $\mathrm{cost}_{\mathrm{dig}}(u, v)$ of including that edge in any path from a facility to a user, as well as the cost $\mathrm{cost}_{\mathrm{unit}}(u, v)$ of supplying a single unit of demand along that edge.

In the CFLP variant of the problem we also know a maximum capacity $\mathrm{cap}_{\mathrm{facility}}(v)$ for each vertex $v \in V(G)$.

Uncapacitated Facility Location / Network Design Problem

Goal: Find a minimum-cost set of edges and facilities on G, such that the demand $d(u)$ of each user $u \in U$ has been satisfied.[6]

Minimize:

$$\sum_{(u,v)\in E(G)} (\text{cost}_{\text{edge}}(u,v) \cdot x_{u,v} + \text{cost}_{\text{unit}}(u,v) \cdot (y_{u,v} + y_{v,u})) + \sum_{v\in V(G)} (\text{cost}_{\text{facility}}(v) \cdot z_v)$$

Subject to:

$\sum_{v\in V(G)} (y_{(v,u)}) + d(u) = \sum_{v\in V(G)} (y_{(u,v)} + w_u)$	For each vertex $u \in V(G)$.
$w_u \leq \sum_{v\in V(G)} (d(v)) \cdot z_u$	For each vertex $u \in V(G)$.
$y_{(u,v)} \leq \sum_{v\in V(G)} (d(v)) \cdot x_{(u,v)}$	For each edge $(u,v) \in E(G)$.
$y_{(v,u)} \leq \sum_{v\in V(G)} (d(v)) \cdot x_{(u,v)}$	For each edge $(u,v) \in E(G)$.
$y_{(u,v)}, y_{(v,u)} \geq 0$	For each edge $(u,v) \in E(G)$.
$y_{(u,v)}, y_{(v,u)} \in \mathbb{Z}$	For each edge $(u,v) \in E(G)$.
$x_{(u,v)} \in \{0,1\}$	For each edge $(u,v) \in E(G)$.
$w_u \geq 0$	For each vertex $u \in V(G)$.
$w_u \in \mathbb{Z}$	For each vertex $u \in V(G)$.
$z_u \in \{0,1\}$	For each vertex $u \in V(G)$.

Decision Variables: We consider the following decision variables:

w_u	Denotes the number of units supplied by the facility on vertex u
$x_{(u,v)}$	Denotes whether the edge (u,v) is included in the solution
$y_{(u,v)}$	Denotes the number of units moved from vertex u to vertex v
z_u	Denotes whether the facility at vertex u is included in the solution.

Capacitated Facility Location / Network Design Problem

Goal: Find a minimum-cost set of edges and facilities on G, such that the demand $d(u)$ of each user $u \in U$ has been satisfied, without exceeding the maximum capacity of any facility.[7]

Minimize:

$$\sum_{(u,v)\in E(G)} (\text{cost}_{\text{edge}}(u,v) \cdot x_{u,v} + \text{cost}_{\text{unit}}(u,v) \cdot (y_{u,v} + y_{v,u})) + \sum_{v\in V(G)} (\text{cost}_{\text{facility}}(v) \cdot z_v)$$

Subject to:

$\sum_{v\in V(G)} (y_{(v,u)}) + d(u) = \sum_{v\in V(G)} (y_{(u,v)} + w(u))$	For each vertex $u \in V(G)$.
$w(u) \leq \text{cap}_{\text{facility}}(u) \cdot z_u$	For each vertex $u \in V(G)$.
$y_{(u,v)} \leq \sum_{v\in V(G)} (d(v)) \cdot x_{(u,v)}$	For each edge $(u,v) \in E(G)$.
$y_{(v,u)} \leq \sum_{v\in V(G)} (d(v)) \cdot x_{(u,v)}$	For each edge $(u,v) \in E(G)$.
$y_{(u,v)}, y_{(v,u)} \geq 0$	For each edge $(u,v) \in E(G)$.
$y_{(u,v)}, y_{(v,u)} \in \mathbb{Z}$	For each edge $(u,v) \in E(G)$.
$x_{(u,v)} \in \{0,1\}$	For each edge $(u,v) \in E(G)$.
$w_u \geq 0$	For each vertex $u \in V(G)$.
$w_u \in \mathbb{Z}$	For each vertex $u \in V(G)$.
$z_u \in \{0,1\}$	For each vertex $u \in V(G)$.

Decision Variables: We consider the following decision variables:

w_u	Denotes the number of units supplied by the facility on vertex u
$x_{(u,v)}$	Denotes whether the edge (u,v) is included in the solution
$y_{(u,v)}$	Denotes the number of units moved from vertex u to vertex v
z_u	Denotes whether the facility at vertex u is included in the solution.

2.4 Steiner Tree Problem

The family of Steiner Tree Problem (STP) refers to a collection of optimization problems concerned with optimally connecting a set of terminals. In this thesis we are mainly concerned with the Steiner Minimal Tree problem on graphs, where we wish to optimally connect a set of terminals along the edges of the graph.

This problem can be defined as follows:

Input: We consider a graph G such that every edge $(u,v) \in E(G)$ has a positive weight weight(u,v) associated to it. We also consider a set of terminals $T \subseteq V(G)$

Goal: Find a minimum-weight connected sub-graph $S \subseteq G$, such that sub-graph S contains all vertices from T.

Chapter 3

Problem Description

In this chapter we will clearly define what the input to our algorithm will look like, and what constitutes a valid solution to CFLP. Furthermore, we define what constitutes a partial solution at any node in the tree decomposition.

3.1 Problem Definition

Input: An undirected connected graph $G = (V, E)$ with a positive digging cost $\text{cost}_{\text{dig}}(e)$ and a cable cost $\text{cost}_{\text{cable}}(e)$ for each edge $e \in E(G)$. A subset of vertices $U \subseteq V(G)$ denoting the locations of users/clients on the graph G, with a positive demand $d(u)$ for each user $u \in U$. Some cost value cost_{DP} and maximum capacity cap_{DP} for any constructed facility.

Feasible Solution: Any valid solution l has the following structure $l = (F, \mu, \Phi, \Pi)$, where:

- F denotes the set of all facilities to be constructed
  Each facility $f \in F$ is associated to some vertex $v(f) \in V(G)$
- $\mu : U \to F$ denotes a mapping, indicating the facility $f \in F$ each user $u \in U$ is served by.
  μ^{-1} is the inverse mapping, defined as $\mu^{-1}(f) = \{u \in U \mid \mu(u) = f\}$, indicating the set of users $u \in U$ that each facility $f \in F$ serves.
  For each facility $f \in F$ it holds that $\sum_{u \in \mu^{-1}(f)}(d(u)) \leq \text{capacity}_{\text{DP}}$.
- $\Phi = \{\Psi_f\}_{f \in F}$ denotes the set "facility trees" associated to their respective facilities.
  Each facility tree $\Psi_f \in \Phi$ is a Steiner tree such that $\Psi_f \subseteq G$, with respective terminals $v(f) \cup \mu^{-1}(f)$.
  For any two facilities $f, f' \in F$ such that $f \neq f'$, it holds that $E(\Psi_f) \cap E(\Psi'_f) = \emptyset$ and that $V(\Psi_f) \cap V(\Psi'_f) = \emptyset$. In other words, $\Psi_f, \Psi_{f'}$ are edge- and vertex-disjoint.
- $\Pi = \{\pi_u\}_{u \in U}$ denotes the set of all constructed paths between users and facilities.
  For each user $u \in U$, the respective path $\pi_u \in \Pi$ runs from u to $v(\mu(u))$ in the tree $\Psi_{\mu(u)}$

Cost Measure: The cost_l of solution l is the sum of the following three items:

$\lvert F \rvert \cdot \text{cost}_{\text{DP}}$	Total cost of constructing all facilities
$\sum_{f \in F} \sum_{e \in \Psi_f}(\text{cost}_{\text{dig}}(e))$	Total cost of digging the trenches for all facility trees
$\sum_{u \in U} \sum_{e \in \pi_u}(\text{cost}_{\text{cable}}(e) \cdot d(u))$	Total cost of cables between facility and user

Constraint:
Users and facilities cannot share a vertex: $U \cap \{v(f) \mid f \in F\} = \emptyset$

3.2 Subproblem Definition

Input: An undirected connected graph $G = (V, E)$ with a positive digging cost $\text{cost}_{\text{dig}}(e)$ and a cable cost $\text{cost}_{\text{cable}}(e)$ for each edge $e \in E(G)$. A subset of vertices $U \subseteq V(G)$ denoting the locations of users/clients on the graph G, with a positive demand $d(u)$ for each user $u \in U$.
Some cost value cost_{DP} and maximum capacity cap_{DP} for any constructed facility.

Additionally we are given a sub-graph $G' \subseteq G$, and a partition of $V(G)$ into the vertex subsets $A \cup X \cup B$, such that:

- $V(G') = A \cup X$
- $V(G) = A \cup X \cup B$
- There exist no edges $(a, b) \in E(G)$ such that $a \in A$ and $b \in B$.

We also define the subset of users $U' \subseteq U$ to be $U' = U \cap (V(G'))$

Partial Solution: Any valid partial solution l' has the following structure $l' = (p, F', \mu, \Phi, \Pi)$:

- $p : X \to \mathbb{Z}$ denotes a mapping, indicating the "leftover capacity"-value for each vertex $x \in X$. For each vertex $x \in X$ it holds that $-\text{capacity}_{\text{DP}} \leq p(x) \leq \text{capacity}_{\text{DP}}$.
  - If $p(x) < 0$ for some $x \in X$, we say that vertex x has "leftover demand"
    $F_P = \{x \,|\, x \in X \wedge p(x) < 0\}$ denotes the set of "pseudo-facilities".
  - If $p(x) > 0$ for some $x \in X$, we say that vertex x has "leftover supply"
    $U_P = \{x \,|\, x \in X \wedge p(x) > 0\}$ denotes the set of "pseudo-users/clients".
    For each pseudo-user $x \in U_P$ define the satisfied demand as $d(x) = \text{ABS}(p(x))$
- F' denotes the set of all facilities to be constructed.
  Each facility $f \in F'$ is associated to some vertex $v(f) \in V(G')$
- $\mu : (U' \cup U_P) \to (F' \cup F_P)$ denotes a mapping of (pseudo-) facilities serving (pseudo-) users. μ^{-1} is the inverse mapping, defined as $\mu^{-1}(f) = \{u \in (U' \cup U_P) \,|\, \mu(u) = f\}$, indicating the set of (pseudo-) users that each (pseudo-) facility serves.
  For each (pseudo-) facility $f \in (F' \cup F_P)$ it holds that $\sum_{u \in \mu'^{-1}(f)}(d(u)) \leq \text{capacity}_{\text{DP}}$.
- $\Phi = \{\Psi_f\}_{f \in (F' \cup F_P)}$ denotes the set "facility trees" associated to their respective facilities. Each facility tree $\Psi_f \in \Phi$ is a Steiner tree such that $\Psi_f \subseteq G'$, with respective terminals $v(f) \cup \mu^{-1}(f)$.
  For any two facilities $f, f' \in (F' \cup F_P)$ such that $f \neq f'$, it holds that $E(\Psi_f) \cap E(\Psi'_f) = \emptyset$ and that $V(\Psi_f) \cap V(\Psi'_f) = \emptyset$. In other words, $\Psi_f, \Psi_{f'}$ are edge- and vertex-disjoint.
- $\Pi = \{\pi_u\}_{u \in (U' \cup U_P)}$ denotes the set of all constructed paths between users and facilities. For each user $u \in (U' \cup U_P)$, the path $\pi_u \in \Pi$ runs from u to $v(\mu(u))$ in the tree $\Psi_{\mu(u)}$

Cost Measure: The $\text{cost}_{l'}$ of solution l' is the sum of the following three items:

• $\lvert F' \rvert \cdot \text{cost}_{\text{DP}}$	Total cost of constructing all facilities
• $\sum_{f \in (F' \cup F_P)} \sum_{e \in \Psi_f}(\text{cost}_{\text{dig}}(e))$	Total cost of digging the trenches for all facility trees
• $\sum_{u \in (U' \cup U_P)} \sum_{e \in \pi_u}(\text{cost}_{\text{cable}}(e) \cdot d(u))$	Total cost of cables between facility and user

Constraint:
Users and facilities cannot share a vertex: $U \cap \{v(f) \,|\, f \in F\} = \emptyset$

3.3 Elaboration

The graph $G = (V, E)$ denotes a predetermined set of potential locations to place facilities and cables. The edge set $E(G)$ denotes the potential locations to dig trenches through which cables can be run. The vertex set $V(G)$ denotes the potential locations where facilities can be built.
The set $U \subseteq V(G)$ denotes the places where users/clients need to be serviced. It is important to note that in a valid solution, facilities and users/clients cannot occupy the same vertex. In other words, it has to hold that $U \cap F = \emptyset$.

Given a valid solution, the set F denotes the set of all facilities that need to be constructed.
Each facility $f \in F$ serves the set of users $u \in \mu^{-1}(f)$, and does so with cables that follow the path π_u, which follows the set of dug trenches denoted by Ψ_f, referred to as the "facility tree".
It is a requirement that any facility tree Ψ_f must be both edge- and vertex-disjoint from any other facility tree Ψ'_f.

In a partial solution we also keep track of a "leftover capacity", which we can use in subsequent nodes to attach additional users/facilities. The best way to visualize this, is by imagining that we leave unconnected cables in the trench, that we can later use to connect user and facilities that we have not yet seen or been able to connect.
There are two types of leftover capacity:

- Pseudo-facilities (with leftover demand) are vertices that pass through capacity in a facility tree, but do not yet have a constructed facility to provide them with that capacity.
  They still require a facility to be hooked in some subsequent step.
- Pseudo-users (with leftover supply) are vertices that are provided capacity to pass through, but do not yet have any users to provide this capacity to.
  They still require users to be hooked in some subsequent step.

Chapter 4

Algorithm Description

4.1 Description of the Algorithm

This section elaborates how the algorithm uses the concept of tree decomposition to solve the problem as defined in the problem definition of section 3.1.

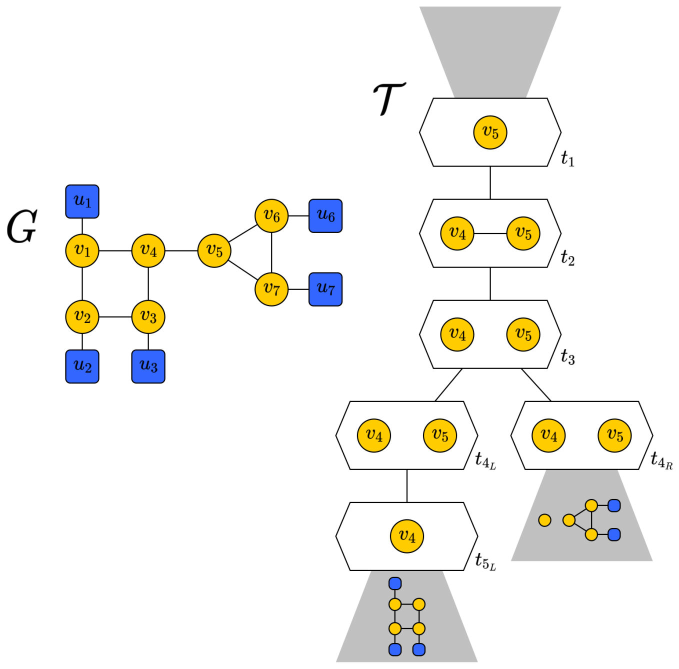


Figure 4.1: An example-graph G, and (part of) some "nice" tree decomposition $\mathcal{T}$ of G

Input: We consider an undirected connected graph $G = (V, E)$ and set of users $U \subseteq V(G)$ as described in the input definition of the problem definition in section 3.1.

Additionally we also consider some "nice" tree decomposition of graph G:

$$\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$$

For the sake of visualizing and clarifying the algorithm, we will occasionally reference the example graph G with the corresponding tree decomposition $\mathcal{T}$, as shown in figure 4.1. The set of users in this example is defined as $U = \{u_1, u_2, u_3, u_6, u_7\}$, depicted in blue, with $d(u) = 1$ for each user $u \in U$.

Goal: Given the graph G and decomposition tree $\mathcal{T}$ as input, we wish to find some solution l_{OPT} that is both a feasible and an optimal solution. This means that there should not exist any solution l'_{OPT} such that l'_{OPT} is valid and $\text{cost}_{l_{\text{OPT}}} > \text{cost}_{l'_{\text{OPT}}}$.

4.1.1 Overall Strategy

We aim to build up such an optimal solution l_{OPT} by iterating over the tree decomposition $\mathcal{T}$ recursively, in a bottom-up fashion.

At each node $t \in T$ we consider the sub-graph G_t, as well as vertex subsets A_t, X_t, B_t, as defined in the tree decomposition definition in section 2.2. These correspond to the sub-graph G' and the vertex subsets A, X, B as defined in the sub-problem definition in section 3.2.
This means that we can consider partial solution l_t, at any node $t \in T$, to be defined as the corresponding partial solution l' in the sub-problem definition.

We define the set L_t, for each node $t \in T$, to be a set of valid partial solutions l_t on sub-graph G_t. It must hold that there exists at least one partial solution $l_{t,\text{OPT}} \in L_t$ a such that $l_{t,\text{OPT}}$ is the partial solution to some optimal solution l_{OPT} on graph G.
The set L_t is referred to as the set of candidate partial solutions on sub-graph G_t.

At each node $t \in T$ we only know about the viability of solutions on the sub-graph G_t. We do not yet know what partial solution will result in the definite optimal solution on graph G. This means we need to keep track of (at least) all candidate partial solutions $l_t \in L_t$ which could potentially result in being the partial solution to some optimal solution l_{OPT}.

At each node $t \in T$ we are able to build L_t by considering the set of solutions $L_{t'}$ for each node $t' \in \text{CHILDREN}(t)$. Since we iterate over the tree T recursively bottom-up, this set $L_{t'}$ has been pre-calculated.

Because $\mathcal{T}$ is a "nice" tree decomposition of G, any node $t \in T$ can be either of five types of nodes:

- "Introduce edge"-node
- "Introduce vertex"-node
- "Forget vertex"-node
- "Join sub-trees"-node
- "Leaf"-node

Sections 4.1.2 through 4.1.6 elaborate how L_t is calculated at each type of node t respectively.
Section 4.1.7 details how the example graph G in figure 4.1 gets solved

4.1.2 "Introduce Edge"-Node

We consider some node $t \in T$ such that:

- node t is labelled with an edge $(u, v) \in E(G)$ such that $u, v \in X_t$, and
- node t has exactly one child t', such that $X_t = X_{t'}$.

We say that edge (u, v) is introduced at node t.

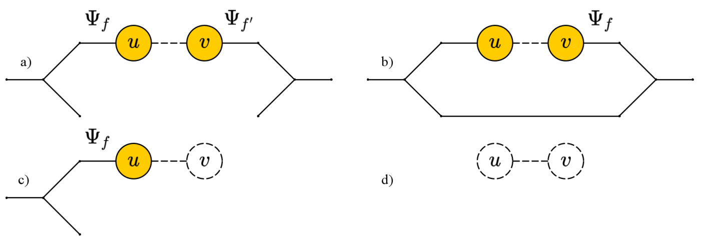


Figure 4.2: Four different scenarios when introducing an edge $(u, v) \in E(G)$

We build up the partial solution set L_t at node t, by updating the partial solution set $L_{t'}$ at node t' such that the resulting set L_t contains the set of candidate partial solutions for G_t.

Given some partial solution $l_{t'} = (p_{t'}, F'_{t'}, \mu_{t'}, \Phi_{t'}, \Pi_{t'})$ such that $l_{t'} \in L_{t'}$, there are four possible scenarios (as illustrated in figure 4.2):

(a) There exist some pair of trees $\Psi_f, \Psi_{f'} \in \Phi_{t'}$ such that $u \in V(\Psi_f), v \in V(\Psi_{f'})$ and $f \neq f'$.
In other words, both vertices u and v are part of the current partial solution $l_{t'}$, and are covered by two distinct facility trees.

(b) There exist some tree $\Psi_f \in \Phi_{t'}$ such that $u, v \in V(\Psi_f)$.
In other words, both vertices u and v are part of the current partial solution $l_{t'}$, and are covered by the same facility tree.

(c) There exist some tree $\Psi_f \in \Phi_{t'}$ such that $u \in V(\Psi_f)$.
There exist **no** tree $\Psi_{f'} \in \Phi_{t'}$ such that $v \in V(\Psi'_f)$ with $f \neq f'$. (or vice versa)
In other words, either vertex either u or v are not part of the current partial solution $l_{t'}$

(d) There exist **no** tree $\Psi_f \in \Phi_{t'}$ such that $u \in V(\Psi_f)$ or $v \in V(\Psi_f)$.
In other words, either vertex u and v are both not part of the current partial solution $l_{t'}$

For each partial solution $l_{t'} \in L_{t'}$ there are two possibilities for what to do with the edge (u, v) introduced at node t:

- We don't include the edge (u, v) in the solution $l_{t'}$, making a set $L_{l_{t'},\text{excl.}}$

  This is possible in all four scenarios (a), (b), (c) and (d)

- We do include the edge (u, v) in the solution $l_{t'}$, making a set $L_{l_{t'},\text{incl.}}$

  This is possible in scenario (a), if leftover supply and demand allow.
  This is also possible in scenarios (c) and (d), if leftover supply or demand allows.

  In scenario (b) including edge (u, v) is impossible. This is because including the edge (u, v) would mean that the resulting facility tree Ψ_f contains a loop, and is therefore no longer a Steiner tree.

Excluding Edge (u, v): Scenarios (a), (b), (c), (d)

Not including the edge (u, v) into a solution $l_{t'}$ does not require much of an explanation:

$$L_{l_{t'},\text{excl.}} \leftarrow \{l'_t\}$$

Including Edge (u, v): Scenario (a)

When including the edge (u, v) into a solution $l_{t'}$ of scenario (a), we aim to merge the facility trees $\Psi_{\mu_{t'}(u)}, \Psi_{\mu_{t'}(v)}$ into a singular facility tree.

There are three sub-scenarios in scenario (a):

(a-1) Vertex u has a leftover capacity $p_{t'}(u) > 0$, and vertex v has a leftover demand $p_{t'}(v) < 0$
In other words, the vertex u is a pseudo-user/client with leftover supply, and the vertex v is a pseudo-facility with leftover demand. (or vice versa)

(a-2) Vertex u has a leftover capacity $p_{t'}(u) < 0$, and vertex v has a leftover demand $p_{t'}(v) < 0$
In other words, both vertex u and v are a pseudo-facility with leftover demand.

(a-3) Neither of the above cases hold

In sub-scenarios (a-1) and (a-2) we look into merging the facility trees $\Psi_{\mu_{t'}(u)}, \Psi_{\mu_{t'}(v)}$ into a singular facility tree with a shared (pseudo-) facility.
In sub-scenario (a-3) there is no such reason to include the edge (u, v).

It is possible that the scenario (a-1) holds in reverse, meaning that $p_{t'}(v) > 0$ and $p_{t'}(u) < 0$. For simplicity in explaining the algorithm we will only describe the scenario such that $p_{t'}(u) > 0$ and $p_{t'}(v) < 0$ holds. However, the reverse scenario is solved in the exact same way by simply swapping the places of u and v

Sub-scenario (a-1)

It is given that the vertex u and v have the leftover capacities $p_{t'}(u) > 0$ and $p_{t'}(v) < 0$.
This means that the vertex u is a pseudo-user/client with leftover supply, and the vertex v is a pseudo-facility with leftover demand. We wish to satisfy the existing leftover demand on v with the existing leftover supply on u, by joining the respective facility trees.

We are only able to merge the respective facility trees $\Psi_{\mu_{t'}(u)}$ and $\Psi_{\mu_t(v)}$ iff the following property holds:

$$\text{ABS}(p_{t'}(u)) \geq \text{ABS}(p_{t'}(v))$$

In other words, vertex v must have a leftover supply, vertex u much have a leftover demand, and the leftover supply at v must be greater than or equal to the leftover demand at u. This is because if supply cannot meet demand, it is impossible for the two trees to be merged by connecting (u, v), given the current solution $l_{t'}$.

Interestingly, since for any vertex x it cannot hold that both $0 > p(x) > 0$, merging facility trees $\Psi_{\mu_{t'}(u)}, \Psi_{\mu_{t'}(v)}$ can always be done in at most one direction.

We do this by extending the facility tree $\Psi_{\mu_{t'}(u)}$ as follows:

$$V(\Psi_{\mu_t(u)}) = V(\Psi_{\mu_{t'}(u)}) \cup V(\Psi_{\mu_{t'}(v)})$$
$$E(\Psi_{\mu_t(u)}) = E(\Psi_{\mu_{t'}(u)}) \cup E(\Psi_{\mu_{t'}(v)}) \cup \{(u,v)\}$$

For moving the supply from vertex v to vertex u, we define an "edge-capacity" c for the edge (u, v). This indicates the number of cables that will be placed on edge (u, v). This capacity cannot exceed the leftover supply at u, but must also at least satisfy demand at v.

Therefor all valid values $c \in \mathbb{Z}$ are defined as follows:

$$\textsc{abs}(p_{t'}(v)) \leq c \leq \textsc{abs}(p_{t'}(u))$$

Moving the pseudo-facility from vertex v to the facility serving u, means that for each user $v' \in \mu_{t'}^{-1}(v)$, i.e. every (pseudo-) user v' that is served by the same (pseudo-) facility on v, we redefine the mapping in l_t as follows:

$$\mu_t(v') \leftarrow \mu_{t'}(u)$$

Consequently we can drop facility $\mu_{t'}(v)$ from F'_t and Φ_t.

We use edge-capacity c to calculate the leftover capacities $p_t(u), p_t(v)$ as follows:

$$p_t(u) \leftarrow p_{t'}(u) - c \qquad p_t(v) \leftarrow c - \textsc{abs}(p_{t'}(v))$$

The set $L_{l_{t'},\text{incl.}}$ contains the generated new solutions l_t for each valid value c:

$$L_{l_{t'},\text{incl.}} \leftarrow \bigcup_{\textsc{abs}(p_{t'}(v)) \leq c \leq \textsc{abs}(p_{t'}(u))} \{l_{t,c}\}$$

Sub-scenario (a-2)

It is given that the vertex u and v both have the leftover capacities $p_{t'}(u) < 0, p_{t'}(v) < 0$. This means that both vertex u and vertex v are a pseudo-facility with leftover demand. We wish to see if by merging the respective facility trees we can eliminate either pseudo-facility u or v

We are only able to merge the respective facility trees Ψ_u and Ψ_v iff the following property holds:

$$\textsc{abs}(p_{t'}(u)) + \textsc{abs}(p_{t'}(v)) \leq \text{cap}_{\text{DP}}$$

In other words, merging the respective facility trees Ψ_u and Ψ_v must not result in a new singular facility leftover supply greater than any facility is allowed to facilitate.

Including the edge (u, v) into a solution l'_t in such a manner, can be done in two directions:

- Placing a pseudo-facility on vertex u and a pseudo-user/client on vertex v.
- Placing a pseudo-facility on vertex v and a pseudo-user/client on vertex u.

For simplicity we only describe the placing a pseudo-facility on vertex u and a pseudo-user/client on vertex v. The inverse works in the exact same way, but by swapping the places of u and v.

We proceed by extending the facility tree $\Psi_{\mu_{t'}(u)}$ as follows:

$$V(\Psi_{\mu_t(u)}) = V(\Psi_{\mu_{t'}(u)}) \cup V(\Psi_{\mu_{t'}(v)})$$
$$E(\Psi_{\mu_t(u)}) = E(\Psi_{\mu_{t'}(u)}) \cup E(\Psi_{\mu_{t'}(v)}) \cup \{(u,v)\}$$

For moving the supply from vertex v to vertex u, we define an "edge-capacity" c for the edge (u, v). This indicates the number of cables that will be placed on edge (u, v). This edge-capacity must at least satisfy demand at v, but cannot allow vertex u to exceed the maximum allowed facility capacity cap_{DP}

Therefor all valid values $c \in \mathbb{Z}$ are defined as follows:

$$\text{ABS}(p_{t'}(v)) \le c \le \text{cap}_{\text{DP}} - \text{ABS}(p_{t'}(u))$$

Moving the pseudo-facility from vertex v to the facility serving u, means that for each user $v' \in \mu_{t'}^{-1}(v)$, i.e. every (pseudo-) user v' that is served by the same (pseudo-) facility on v, we redefine the mapping in l_t as follows:

$$\mu_t(v') \leftarrow \mu_{t'}(u)$$

Consequently we can drop facility $\mu_{t'}(v)$ from F'_t and Φ_t.

We use edge-capacity c to calculate the leftover capacities $p_t(u), p_t(v)$ as follows:

$$p_t(u) \leftarrow p_{t'}(u) - c \qquad p_t(v) \leftarrow c - \text{ABS}(p_{t'}(v))$$

The set $L_{l_{t'},\text{incl.}}$ contains the new solutions $l_{t,(u,v)}, l_{t,(v,u)}$ in both directions for each c:

$$L_{l_{t'},\text{incl.}} \leftarrow \bigcup_{\text{ABS}(p_{t'}(v)) \le c \le \text{cap}_{\text{DP}} - \text{ABS}(p_{t'}(u))} \left\{ l_{t,(u,v)}, l_{t,(v,u)} \right\}$$

Including Edge (u, v): Scenario (c)

When including the edge (u, v) into a solution $l_{t'}$ of scenario (c), we aim to offload any leftover demand or supply on the included vertex u off to the excluded vertex v.

There are three sub-scenarios in scenario (c):

(c-1) The vertex u has a leftover capacity $p(u) > 0$
In other words, the vertex u is a pseudo-user/client, and has leftover supply.

(c-2) The vertex u has a leftover capacity $p(u) < 0$
In other words, the vertex u is a pseudo-facility, and has leftover demand.

(c-3) The vertex u has a leftover capacity $p(u) = 0$
In other words, the vertex u has neither leftover supply or demand.

In sub-scenarios (c-1) and (c-2) we wish to consider including the vertex v into the facility tree $\Psi_{\mu_{t'}(u)}$ to help offload/satisfy some leftover demand or supply from the vertex u.
In sub-scenario (c-3) there is no such reason to include the edge (u, v).

Sub-scenario (c-1)

It is given that the vertex u has a leftover capacity $p(u) > 0$, meaning that the vertex u is a pseudo-user/client and has leftover supply. We wish to consider branching of some or all of this leftover supply at vertex u onto the vertex v.

We do this by extending the facility tree $\Psi_{\mu_{t'}(u)}$ as follows:

$$V(\Psi_{\mu_t(u)}) = V(\Psi_{\mu_{t'}(u)}) \cup \{v\}$$
$$E(\Psi_{\mu_t(u)}) = E(\Psi_{\mu_{t'}(u)}) \cup \{(u, v)\}$$

For branching off the supply to vertex v, we define an "edge-capacity" c to the edge (u, v). This indicates the number of cables that will be placed on edge (u, v). This capacity cannot exceed the leftover supply at u, but must move at least some supply to v.

Therefor all valid values $c \in \mathbb{Z}$ are defined as follows:

$$0 < c \leq p_{t'}(u)$$

We use edge-capacity c to calculate mapping $\mu_t(v)$ and leftover capacities $p_t(u), p_t(v)$ as follows:

$$\mu_t(v) \leftarrow \mu_{t'}(u) \qquad p_t(u) \leftarrow p_{t'}(u) - c \qquad p_t(v) \leftarrow c$$

The set $L_{l_{t'},\text{incl.}}$ contains the generated new solutions l_t for each valid value c:

$$L_{l_{t'},\text{incl.}} \leftarrow \bigcup_{0<c\leq p_{t'}(u)} \{l_{t,c}\}$$

Sub-scenario (c-2)

It is given that the vertex u has a leftover capacity $p(u) < 0$, meaning that the vertex u is a pseudo-facility and has leftover demand. Including the edge (u,v) into a solution $l_{t'}$ as such, can be done in two directions:

- Moving the pseudo-facility located at vertex u onto the vertex v.
- Branching of additional leftover-supply at v, increasing left-over demand at u.

We will describe either method separately below. Both methods combined $L_{l'_t,\text{incl.,move}}, L_{l'_t,\text{incl.,branch}}$ will make up the total set $L_{l_{t'},\text{incl.}}$ as follows:

$$L_{l'_t,\text{incl.}} \leftarrow L_{l'_t,\text{incl.,move}} \cup L_{l'_t,\text{incl.,branch}}$$

Sub-scenario (c-2): *Moving the Pseudo-Facility*

It is given that the vertex u has a leftover capacity $p(u) < 0$, meaning that the vertex u is a pseudo-facility and has leftover demand. We aim to move this pseudo-facility from vertex u to vertex v

We do this by extending the facility tree $\Psi_{\mu_{t'}(u)}$ as follows:

$$V(\Psi_{\mu_t(u)}) = V(\Psi_{\mu_{t'}(u)}) \cup \{v\}$$
$$E(\Psi_{\mu_t(u)}) = E(\Psi_{\mu_{t'}(u)}) \cup \{(u, v)\}$$

For moving the leftover demand from vertex u to vertex v, we define an "edge-capacity" c to the edge (u, v). This indicates the number of cables that will be placed on edge (u, v).
The edge capacity must at least meet the current leftover supply at u and might even need to route back some capacity to u. It can also not exceed the maximum facility capacity cap_{DP}.

Therefor all valid values $c \in \mathbb{Z}$ are defined as follows:

$$\text{ABS}(p_{t'}(u)) \leq c \leq \text{cap}_{\text{DP}}$$

Moving the pseudo-facility from vertex u to vertex v, means that for each user $u' \in \mu_{t'}^{-1}(u)$, i.e. every (pseudo-) user u' that is served by the same (pseudo-) facility on u, we redefine the mapping in l_t as follows:

$$\mu_t(u') \leftarrow v$$

We use edge-capacity c to calculate the leftover capacities $p_t(u), p_t(v)$ as follows:

$$p_t(u) \leftarrow p_{t'}(u) + c \qquad p_t(v) \leftarrow -1 \cdot c$$

The set $L_{l_{t'},\text{incl.,move}}$ contains the generated new solutions l_t for each valid value c:

$$L_{l_{t'},\text{incl.,move}} \leftarrow \bigcup_{\text{ABS}(p_{t'}(u)) \leq c \leq \text{cap}_{\text{DP}}} \{l_{t,c}\}$$

Sub-scenario (c-2): ***Branching off Additional Supply***

It is given that the vertex u has a leftover capacity $p(u) < 0$, meaning that the vertex u is a pseudo-facility and has leftover demand. We aim to branch off some additional supply from vertex u to vertex v

We do this by extending the facility tree $\Psi_{\mu_{t'}(u)}$ as follows:

$$V(\Psi_{\mu_t(u)}) = V(\Psi_{\mu_{t'}(u)}) \cup \{v\}$$
$$E(\Psi_{\mu_t(u)}) = E(\Psi_{\mu_{t'}(u)}) \cup \{(u, v)\}$$

For branching off the supply to vertex v, we define an "edge-capacity" c to the edge (u, v). This indicates the number of cables that will be placed on edge (u, v). This capacity must move at least some supply to v but cannot exceed the maximum supply the facility at u can still supply.

Therefor all valid values $c \in \mathbb{Z}$ are defined as follows:

$$0 < c \leq \text{cap}_{\text{DP}} - \text{ABS}(p_{t'}(u))$$

We use edge-capacity c to calculate mapping $\mu_t(v)$ and leftover capacities $p_t(u), p_t(v)$ as follows:

$$\mu_t(v) \leftarrow \mu_{t'}(u) \qquad p_t(u) \leftarrow p_{t'}(u) - c \qquad p_t(v) \leftarrow c$$

The set $L_{l_{t'},\text{incl.}}$ contains the generated new solutions l_t for each valid value c:

$$L_{l_{t'},\text{incl.,branch}} \leftarrow \bigcup_{0 < c \leq \text{cap}_{\text{DP}} - \text{ABS}(p_{t'}(u))} \{l_{t,c}\}$$

Including Edge (u, v): Scenario (d)

When including the edge (u, v) into a solution $l_{t'}$ of scenario (d), we aim to create a new pair of pseudo-facility and pseudo-user/client.

Including the edge (u, v) into a solution l'_t in such a manner, can be done in two directions:

- Placing a pseudo-facility on vertex u and a pseudo-user/client on vertex v.
- Placing a pseudo-facility on vertex v and a pseudo-user/client on vertex u.

For simplicity we only describe the placing a pseudo-facility on vertex u and a pseudo-user/client on vertex v. The inverse works in the exact same way, but by swapping the places of u and v.

We create a new facility tree Ψ_u which is defined as follows:

$$V(\Psi_u) = \{u, v\} \qquad E(\Psi_f) = \{(u, v)\}$$

We define the mappings $\mu_t(u), \mu_t(v)$ for facility tree Ψ_u as follows:

$$\mu_t(u) \leftarrow u \qquad \mu_t(v) \leftarrow u$$

When building a (pseudo-) facility f, we must determine the "facility-capacity" c before we place it. All valid values $c_{\in}\mathbb{Z}$ are defined as follows:

$$0 < c \leq \text{cap}_{\text{DP}}$$

We use facility-capacity c to calculate the leftover capacities $p_t(u), p_t(v)$ as follows:

$$p_t(u) \leftarrow -1 \cdot c \qquad p_t(v) \leftarrow c$$

The set $L_{l_{t'},\text{incl.}}$ contains the generated new solutions $l_{t,(u,v)}, l_{t,(v,u)}$ in both directions for each valid value c:

$$L_{l_{t'},\text{incl.}} \leftarrow \bigcup_{0<c\leq\text{cap}_{\text{DP}}} \left\{l_{t,(u,v)}, l_{t,(v,u)}\right\}$$

Cost Calculation and Overal Solution Set

These steps allow us to recalculate the cost for each solution l_t as follows:

$$\begin{aligned}\text{cost}_{l_t,\text{excl.}} &\leftarrow \text{cost}_{l'_t}\\ \text{cost}_{l_t,\text{incl.}} &\leftarrow \text{cost}_{l'_t} + \text{cost}_{\text{dig}}(u, v) + c \cdot \text{cost}_{\text{cable}}(u, v)\end{aligned}$$

By calculating $L_{l'_t,\text{excl.}}, L_{l'_t,\text{incl.}}$ for each solution $l'_t \in L'_t$, we can calculate L_t as follows:

$$L_t \leftarrow \bigcup_{l'_t \in L'_t} \left(L_{l'_t,\text{excl.}} \cup L_{l'_t,\text{incl.}}\right)$$

4.1.3 "Introduce Vertex"-Node

We consider some node $t \in T$ with exactly one child t', such that $X_t = X_{t'} \cup \{v\}$ for some $v \notin X_{t'}$. We say that vertex v is introduced at node t.

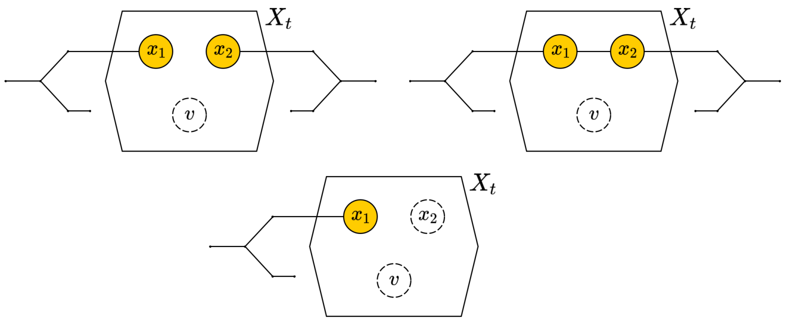


Figure 4.3: Some possible scenarios for $|X_t| = 3$, when introducing a vertex $v \in V(G)$

We build up the partial solution set L_t at node t, by updating the partial solution set $L_{t'}$ at node t' such that the resulting set L_t contains the set of candidate partial solutions for G_t.

By property (T3) of a tree decomposition (see section 2.2) the set of nodes whose corresponding bags contain the introduced vertex v, induces a connected sub-graph of T. Since node t' is the only child of t in T, and since $v \notin X_{t'}$, it must therefore hold that $v \notin V(G_{t'})$.
In other words, prior to node t the introduced vertex v has never before been seen.

This means that for each partial solution $l_{t'} = (p_{t'}, F_{t'}, \mu_{t'}, \Phi_{t'}, \Pi_{t'})$ such that $l_{t'} \in L_{t'}$, there cannot exist a facility tree $\Psi_f \in \Phi'$ such that $v \in V(\Psi_f)$. (as illustrated in figure 4.3)
In other words, the introduced vertex v is by definition not part of any partial solution $l_{t'} \in L_{t'}$.

We can distinguish solution $l_{t'}$ into two distinct scenarios:

(a) The vertex $v \notin U$.
In other words, there exists **no** client/user on the vertex v in G.

(b) The vertex $v \in U$, and has a subsequent demand $d(v)$.
In other words, there exists a client/user on the vertex v in G.

For each partial solution $l_{t'} \in L_{t'}$ there are two possibilities for what to do with the vertex v introduced at node t:

- We don't include the vertex v in the solution $l_{t'}$, making a set $L_{l_{t'},\text{excl.}}$
  This is possible only in scenario (a).
- We do include the vertex v in the solution $l_{t'}$, making a set $L_{l_{t'},\text{incl.}}$
  This is possible in both scenarios (a) and (b).

In either case we will have to calculate the leftover capacity $p_t(v)$. This is because $v \notin X_{t'}$, which means that $p_{t'}(v)$ does not exist, while $p_t(v)$ should exist for the solution to be a valid solution.

Excluding Vertex v: Scenario (a)

Not including the vertex v into a solution $l_{t'}$ does not require much of an explanation:

$$L_{l_{t'},\text{excl.}} \leftarrow \{l_{t'}\} \qquad p_{t,\text{excl.}}(v) \leftarrow 0$$

Including Vertex v: Scenario (a)

When including the vertex v into a solution $l_{t'}$ of scenario (a), we need to automatically also place down a constructed facility. Placing a pseudo-facility on a vertex v with no neighbours and no demand $d(v)$ is not possible, because then the vertex v will have both leftover demand and leftover supply at once.

The newly created facility tree Ψ_v is defined as follows:

$$V(\Psi_v) = \{v\} \qquad E(\Psi_v) = \emptyset$$

When building a facility f on v, we must determine the "facility-capacity" c before we place it. This facility-capacity c must be greater than zero, but must not exceed the maximum facility capacity cap_{DP}.

Therefor all valid values $c \in \mathbb{Z}$ are defined as follows:

$$0 < c \leq \text{cap}_{\text{DP}}$$

We use facility capacity c to calculate mapping $\mu_t(v)$ and leftover capacity $p_t(v)$ as follows:

$$\mu_t(v) \leftarrow v \qquad p_t(v) \leftarrow c$$

The set $L_{l_{t'},\text{incl.}}$ contains the solution l_t for each valid value c:

$$L_{l_{t'},\text{incl.}} \leftarrow \bigcup_{0<c\leq\text{cap}_{\text{DP}}} \{l_{t,c}\}$$

Including Vertex v: Scenario (b)

When including the vertex v into a solution $l_{t'}$ of scenario (b), we need to automatically also place down a pseudo-facility. Placing a constructed facility on a vertex v such that $v \in U$ is not possible, because then the vertex v will have an overlapping facility and user on the same vertex.

The newly created facility tree Ψ_f is defined as follows:

$$V(\Psi_f) = \{v\} \qquad E(\Psi_f) = \emptyset$$

When building a facility f on v, we must determine the "facility-capacity" c before we place it. This facility-capacity c must be equal to the user demand on v.

Therefore we define: $c = d(v)$.

We use facility capacity c to calculate mapping $\mu_t(v)$ and leftover capacity $p_t(v)$ as follows:

$$\mu_t(v) \leftarrow v \qquad p_t(v) \leftarrow -1 \cdot c$$

The set $L_{l_{t'},\text{incl.}}$ contains the generated solution l_t:

$$L_{l_{t'},\text{incl.}} \leftarrow \{l_t\}$$

.

Cost Calculation and Overal Solution Set

These steps allow us to recalculate the cost for each solution l_t as follows:

$$\begin{aligned}\mathrm{cost}_{l_t,\mathrm{excl.}} &\leftarrow \mathrm{cost}_{l'_t}\\ \mathrm{cost}_{l_t,\mathrm{incl.}} &\leftarrow \mathrm{cost}_{l'_t} + \mathrm{cost}_{\mathrm{DP}}\end{aligned}$$

By calculating $L_{l'_t,\mathrm{excl.}}, L_{l'_t,\mathrm{incl.}}$ for each solution $l'_t \in L'_t$, we can calculate L_t as follows:

$$L_t \leftarrow \bigcup_{l'_t \in L'_t} \left(L_{l'_t,\mathrm{excl.}} \cup L_{l'_t,\mathrm{incl.}}\right)$$

4.1.4 "Forget Vertex"-Node

We consider some node $t \in T$ with exactly one child t', such that $X_t = X_{t'}/\{v\}$ for some vertex $v \in X_{t'}$. We say that vertex v is forgotten at node t.

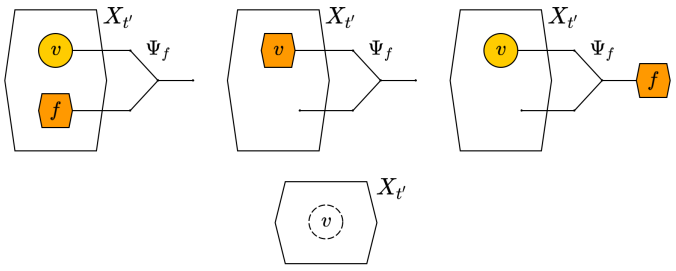


Figure 4.4: Some different scenarios when forgetting a vertex $v \in V(G)$

We build up the partial solution set L_t at node t, by updating the partial solution set $L_{t'}$ at node t' such that the resulting set L_t contains the set of candidate partial solutions for G_t.

When forgetting the vertex v we only want to keep track of solutions $l_{t'} \in L_{t'}$ for which the resulting solution l_t is a valid partial solution to the sub-graph G_t.

For each node $x \in X_t$ we keep track of a mapping $p_t(x)$. This helps us determine the sets F_P of pseudo-facilities and U_P of pseudo-users/clients.
Forgetting the vertex v, means $X_t = X_{t'}/\{v\}$. This means that any solution $l_{t'} \in L_{t'}$ for which the forgotten vertex v has a leftover capacity $p(v) \neq 0$, i.e. v has a pseudo-facility or pseudo-user/client, we ignore that solution $l_{t'}$.

Given some partial solution $l_{t'} = (p_{t'}, F'_{t'}, \mu_{t'}, \Phi_{t'}, \Pi_{t'})$ such that $l_{t'} \in L_{t'}$, there are three possible scenarios (as illustrated in figure 4.4):

(a) There exist some facility tree $\Psi_f \in \Phi_{t'}$ such that $v \in V(\Psi_f)$ with $p_{t'}(v) = 0$.
In other words, the forgotten vertex v is part of the current partial solution $l_{t'}$, and does **not** leave any dangling leftover supply or demand for its respective tree Ψ_f.

(b) There exist some facility tree $\Psi_f \in \Phi_{t'}$ such that $v \in V(\Psi_f)$ with $p_{t'}(v) \neq 0$.
In other words, the forgotten vertex v is part of the current partial solution $l_{t'}$, and does leave some dangling leftover supply or demand for its respective tree Ψ_f.

(c) There exist **no** facility tree $\Psi_f \in \Phi_{t'}$ such that $v \in V(\Psi_f)$.
In other words, the forgotten vertex v is not part of the current partial solution $l_{t'}$

Each solution $l_{t'} \in L_{t'}$ of scenario (a) or (c) we remember by adding $l_{t'}$ to L_t.
Each solution $l_{t'} \in L_{t'}$ of scenario (b) we forget about, since it is incomplete.

The cost calculation of l_t is simple, since we never alter the structure of the solution:

$$\text{cost}_{l_t} \leftarrow \text{cost}_{l'_t}$$

4.1.5 "Join Sub-Trees"-Node

Consider some node $t \in T$ with exactly two children t_1, t_2, such that $X_t = X_{t_1} = X_{t_2}$
We say that the sub-trees rooted at nodes t_1, t_2 are joined at node t.

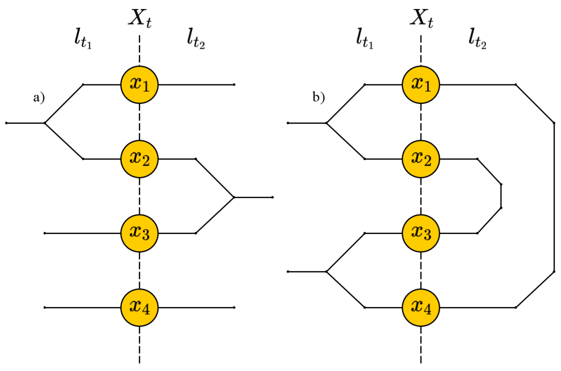


Figure 4.5: Two different scenarios when joining the solutions l_{t_1}, l_{t_2}

We build up the partial solution set L_t at node t, by joining the partial solution sets L_{t_1}, L_{t_2} at nodes t_1, t_2 such that the resulting set L_t contains the sets of candidate partial solutions for G_t.

We build a solution $l_t \in L_t$ by "joining" together any pair of "compatible" solutions l_{t_1}, l_{t_2}.

Property (T3) of a tree decomposition (see section 2.2) states that the set of nodes whose corresponding bags contain the introduced vertex v, induces a connected sub-graph of T.

Considering any vertex $v \notin X_t$, one of the following two properties must hold:

- If vertex $v \in A_{t_1}$, then by property (T3) this means that $v \notin B_{t_1} \cup A_{t_2}$.
- If vertex $v \in A_{t_2}$, then by property (T3) this means that $v \notin B_{t_2} \cup A_{t_1}$.

This means that $A_{t_1} \cap A_{t_2} = \emptyset$.

This means that when joining the partial solutions l_{t_1}, l_{t_2}, we only have to make sure that these solutions are "compatible" (i.e. the resulting joined solution l_t is valid) for facility trees overlapping the vertex set X_t.

Given some pair of partial solutions $l_{t_1} = (p, F'_{t_1}, \mu_{t_1}, \Phi_{t_1}, \Pi_{t_1}), l_{t_2} = (p, F'_{t_2}, \mu_{t_2}, \Phi_{t_2}, \Pi_{t_2})$, there are two possible scenarios (as illustrated in figure 4.5):

(a) The joined graph $\bigcup_{\Psi_f \in (\Phi_{t_1} \cup \Phi_{t_2})} (\Psi_f)$ of all facility trees is acyclic.
In other words, joining the solutions l_{t_1}, l_{t_2} does not create a new facility tree with a cycle.

(b) The joined graph $\bigcup_{\Psi_f \in (\Phi_{t_1} \cup \Phi_{t_2})} (\Psi_f)$ of all facility trees is **not** acyclic.
In other words, joining the solutions l_{t_1}, l_{t_2} creates some new facility tree with a cycle.

Any facility tree Ψ_f, as defined in the problem definition (section 3.1), must be a Steiner tree, and must therefore be acyclic. In the case of scenario (b), joining the respective solutions l_{t_1}, l_{t_2} creates a new facility tree with a cycle, which is not a valid solution.

Thus only scenario (a) can result in a valid joined solution l_t

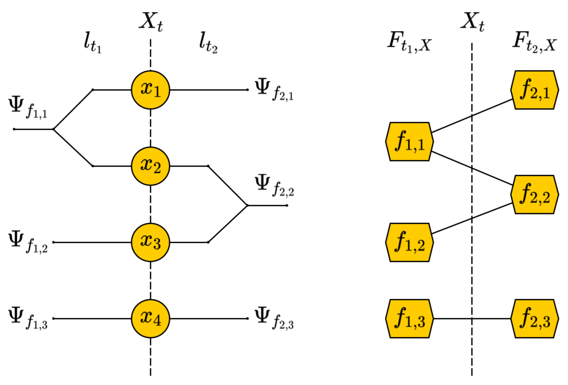


Figure 4.6: Example joining the solutions l_{t_1}, l_{t_2}, and the resulting $\mathcal{F}$

Joining together a pair of solutions l_{t_1}, l_{t_2} of scenario (a) works as follows:

For the solutions l_{t_1}, l_{t_2} we define the sets $F_{t_1,X} \subseteq F_{t_1} \cup F_{P,t_1}$ and $F_{t_2,X} \subseteq F_{t_2} \cup F_{P,t_2}$, of facilities that overlaps the vertex set X_t, as follows:

$$F_{t_1,X} = \{f \mid f \in F_{t_1} \cup F_{P,t_1} \wedge V(\Psi_f) \cap X_t \neq \emptyset\}$$
$$F_{t_2,X} = \{f \mid f \in F_{t_2} \cup F_{P,t_2} \wedge V(\Psi_f) \cap X_t \neq \emptyset\}$$

Using $F_{t_1,X}, F_{t_2,X}$ we define forest of trees $\mathcal{F}$ as follows (example illustrated in figure 4.6):

$$V(\mathcal{F}) = F_{t_1,X} \cup F_{t_2,X}$$
$$E(\mathcal{F}) = \{(f_1, f_2) \mid f_1, f_2 \in (F_{t_1,X} \cup F_{t_2,X}) \wedge V(\Psi_{f_1}) \cap V(\Psi_{f_2}) \neq \emptyset\}$$

For each edge $(f_1, f_2) \in E(\mathcal{F})$ we associate the vertex $\mathrm{v}(f_1, f_2)$ such that:

$$V(\Psi_{f_1}) \cap V(\Psi_{f_2}) = \{\mathrm{v}(f_1, f_2)\}$$

Keep in mind that if it were the case that $|V(\Psi_{f_1}) \cap V(\Psi_{f_2})| > 1$, then the resulting joined facility tree would contain a cycle. This would mean that we are not dealing with a scenario (a) but with a scenario (b) situation, which we have previously determined to result in an invalid solution.

Each connected component in $\mathcal{F}_C \subseteq \mathcal{F}$ denotes the set of facility trees $V(\mathcal{F}_C)$ that are to be merged together into a new facility tree Ψ_C in order to join the two solutions l_{t_1}, l_{t_2}. For each component tree $\mathcal{F}_C$ we approach this merger as follows:

For each facility tree $\Psi_f \in \mathcal{F}_C$ we create a directed version of the component tree $\mathcal{F}'_{C,\Psi_f}$, such that the facility tree Ψ_f is the root of $\mathcal{F}'_{C,\Psi_f}$. This tree $\mathcal{F}'_{C,\Psi_f}$ denotes the direction in which the facility trees are to be merged (see figure 4.7)

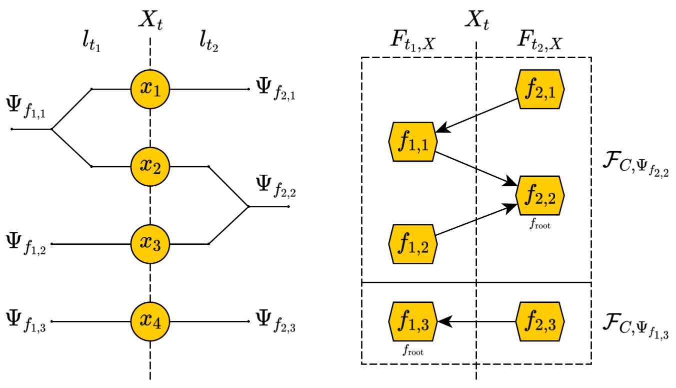


Figure 4.7: Example joining l_{t_1}, l_{t_2}, and some possible set of trees $\{\mathcal{F}'_{C,\Psi_{f_{2,2}}}, \mathcal{F}'_{C,\Psi_{f_{1,3}}}\}$

Joining Connected Components

The solutions l_{t_1}, l_{t_2} can only be joined, if for each connected component $\mathcal{F}_C \subseteq \mathcal{F}$ there exists at least one directed component tree $\mathcal{F}'_{C,\Psi_f}$ that is considered "valid".

We consider a directed component tree $\mathcal{F}'_{C,\Psi_f}$ to be "valid" if for each edge $(f_{\text{child}}, f_{\text{parent}}) \in E(\mathcal{F}'_{C,\Psi_f})$ there exists a valid way of merging the respective facility trees $\Psi_{f_{\text{child}}}, \Psi_{f_{\text{parent}}}$.

For each edge $(f_{\text{child}}, f_{\text{parent}}) \in E(\mathcal{F}'_{C,\Psi_f})$ given the pair of solutions $l_{t_{\text{parent}}}, l_{t_{\text{child}}}$ (these solutions are a re-definition of l_{t_1} and l_{t_2}, depending on whether the directed edge $(f_{\text{child}}, f_{\text{parent}})$ runs from l_{t_1} to l_{t_2}, or l_{t_2} to l_{t_1}), there exist three potential sub-scenarios:

(a-1) For the overlapping vertex $\mathrm{v}(f_{\text{child}}, f_{\text{parent}})$ it holds that:

- $p_{t_{\text{parent}}}(\mathrm{v}(f_{\text{child}}, f_{\text{parent}})) > 0$, and
- $p_{t_{\text{child}}}(\mathrm{v}(f_{\text{child}}, f_{\text{parent}})) < 0$.

In other words, on vertex $\mathrm{v}(f_{\text{child}}, f_{\text{parent}})$, the parent solution $l_{t_{\text{parent}}}$ has a pseudo-user/client with leftover supply, and the child solution $l_{t_{\text{child}}}$ has a pseudo-facility with leftover demand.

(a-2) For the overlapping vertex $\mathrm{v}(f_{\text{child}}, f_{\text{parent}})$ it holds that:

- $p_{t_{\text{parent}}}(\mathrm{v}(f_{\text{child}}, f_{\text{parent}})) < 0$, and
- $p_{t_{\text{child}}}(\mathrm{v}(f_{\text{child}}, f_{\text{parent}})) < 0$.

In other words, on vertex $\mathrm{v}(f_{\text{child}}, f_{\text{parent}})$, the parent solution $l_{t_{\text{parent}}}$ has a pseudo-facility with leftover demand, and the child solution $l_{t_{\text{child}}}$ also has a pseudo-facility with leftover demand.

(a-3) Neither of the above cases hold

In scenarios (a-1) and (a-2) we look into the validity of merging facility trees $\Psi_{f_{\text{child}}}, \Psi_{f_{\text{parent}}}$ into a singular facility tree. In sub-scenario (a-3) there is no such way to merge $\Psi_{f_{\text{child}}}, \Psi_{f_{\text{parent}}}$, so the connected component tree $\mathcal{F}'_{C,\Psi_f}$ is immediately "invalid".

Merging Facility Trees: Scenario (a-1)

When considering the merger of two facility trees of scenario (a-1) on edge $(f_{\text{child}}, f_{\text{parent}}) \in E(\mathcal{F}'_{C,\Psi_f})$, we need to check whether the existing leftover demand on f_{parent} satisfied the existing leftover supply on f_{child}.

The merger is only "valid" if the following holds:

$$\text{ABS}(p_{t_{\text{parent}}}(\text{v}(f_{\text{child}}, f_{\text{parent}}))) \geq \text{ABS}(p_{t_{\text{child}}}(\text{v}(f_{\text{child}}, f_{\text{parent}}))) - d(\text{v}(f_{\text{child}}, f_{\text{parent}}))$$

This check is similar to the leftover capacity check in the "introduce edge"-node, as described in section 4.1.2. The main difference is that we need to account for the user/client-demand $d(\text{v}(f_{\text{child}}, f_{\text{parent}}))$ that would otherwise be counted twice.

For each edge $(f_{\text{child}}, f_{\text{parent}}) \in E(\mathcal{F}'_{C,\Psi_{f_{\text{root}}}})$ of scenario (a-1), such that $\mathcal{F}'_{C,\Psi_{f_{\text{root}}}}$ is a valid directed component tree, we can determine the leftover capacity $p_t(\text{v}(f_{\text{child}}, f_{\text{parent}}))$ as follows:

$$\begin{aligned} p_t(\text{v}(f_{\text{child}}, f_{\text{parent}})) \leftarrow\ & p_{t_{\text{parent}}}(\text{v}(f_{\text{child}}, f_{\text{parent}})) \\ & - (\text{abs}(p_{t_{\text{child}}}(\text{v}(f_{\text{child}}, f_{\text{parent}}))) - d(\text{v}(f_{\text{child}}, f_{\text{parent}}))) \end{aligned}$$

Merging Facility Trees: Scenario (a-2)

When considering the merger of two facility trees of scenario (a-2) on edge $(f_{\text{child}}, f_{\text{parent}}) \in E(\mathcal{F}'_{C,\Psi_f})$, we need to check whether merging the two facility trees $\Psi_{f_{\text{child}}}, \Psi_{f_{\text{parent}}}$ would not exceed the maximum facility capacity cap_{DP}.

The merger is only "valid" if the following holds:

$$\begin{aligned} \text{cap}_{\text{DP}} \geq\ & \text{ABS}(p_{t_{\text{parent}}}(\text{v}(f_{\text{child}}, f_{\text{parent}}))) \\ & + (\text{ABS}(p_{t_{\text{child}}}(\text{v}(f_{\text{child}}, f_{\text{parent}}))) - d(\text{v}(f_{\text{child}}, f_{\text{parent}}))) \end{aligned}$$

This check is similar to the leftover capacity check in the "introduce edge"-node, as described in section 4.1.2. The main difference is that we need to account for the user/client-demand $d(\text{v}(f_{\text{child}}, f_{\text{parent}}))$ that would otherwise be counted twice.

For each edge $(f_{\text{child}}, f_{\text{parent}}) \in E(\mathcal{F}'_{C,\Psi_{f_{\text{root}}}})$ of scenario (a-1), such that $\mathcal{F}'_{C,\Psi_{f_{\text{root}}}}$ is a valid directed component tree, we can determine the leftover capacity $p_t(\text{v}(f_{\text{child}}, f_{\text{parent}}))$ as follows:

$$\begin{aligned} p_t(\text{v}(f_{\text{child}}, f_{\text{parent}})) \leftarrow\ & p_{t_{\text{parent}}}(\text{v}(f_{\text{child}}, f_{\text{parent}})) \\ & + (\text{abs}(p_{t_{\text{child}}}(\text{v}(f_{\text{child}}, f_{\text{parent}}))) - d(\text{v}(f_{\text{child}}, f_{\text{parent}}))) \end{aligned}$$

Finalizing Join

For each facility tree $\Psi_f \in V(\mathcal{F}'_{C,\Psi_{f_{\text{root}}}})$, such that $\mathcal{F}'_{C,\Psi_{f_{\text{root}}}}$ is a valid directed component tree, we define for each user $u' \in \mu_{t'}^{-1}(f)$ the respective mapping as follows:

$$\mu_t(u') \leftarrow f_{\text{root}}$$

For each possible permutation such that we pick exactly one valid directed component tree $\mathcal{F}'_{C,\Psi_f}$ for each connected component $\mathcal{F}_C \subseteq \mathcal{F}$. Joining these directed component tree will result in a set of joined facility trees, which we refer to as $\Phi_{\mathcal{F}}$.

Our ultimate goal at this step, is to create a set $L_{l_{t_1},l_{t_2},\text{valid joins}}$, such that it contains all valid ways of joining the solutions l_{t_1} and l_{t_2}.

We achieve this by expanding $L_{l_{t_1},l_{t_2},\text{valid joins}}$ with a solution l_t for each valid permutation $\Phi_{\mathcal{F}}$. We define the set of facility trees Φ_t for each solution l_t as follows:

$$\Phi_t \leftarrow \Phi_{\mathcal{F}} \cup \{\Psi_f \mid f \in (F_{t_1} \cup F_{P,t_1}) \setminus F_{t_1,X}\} \cup \{\Psi_f \mid f \in (F_{t_2} \cup F_{P,t_2}) \setminus F_{t_2,X}\}$$

These steps allow us to recalculate the cost for each solution l_t as follows:

$$\text{cost}_{l_t} \leftarrow \text{cost}_{l_{t_1}} + \text{cost}_{l_{t_2}}$$

We calculate the set of solutions L_t as follows:

$$L_t \leftarrow \bigcup_{l_{t_1} \in L_{t_1}} \bigcup_{l_{t_2} \in L_{t_2}} \left(L_{l_{t_1},l_{t_2},\text{valid joins}}\right)$$

4.1.6 "Leaf"-Node

For each leaf-node $t \in T$ it is given that:

- $X_t = \emptyset$, and
- The number of children at node t is equal to zero.

Since $X_t = \emptyset$ and the number of children at node t is equal to zero, it holds for the sub-graph $G_t \subseteq G$, that $V(G_t) = \emptyset$ and $E(G_t) = \emptyset$. In other words, sub-graph G_t is an empty graph.

This means that the only valid solution at any leaf-node $t \in T$ is the empty solution.
Therefor L_t for each leaf-node $t \in T$ contains only empty solutions

The cost calculation for each solution $l_t \in L_t$ is trivial:

$$\text{cost}_{l_t} \leftarrow 0$$

4.1.7 Examples

We consider the example graph G with the corresponding tree decomposition $\mathcal{T}$, as shown in both figures 4.1 and 4.8. The set of users in this example is defined as $U = \{u_1, u_2, u_3, u_6, u_7\}$, depicted in blue, with $d(u) = 1$ for each user $u \in U$.

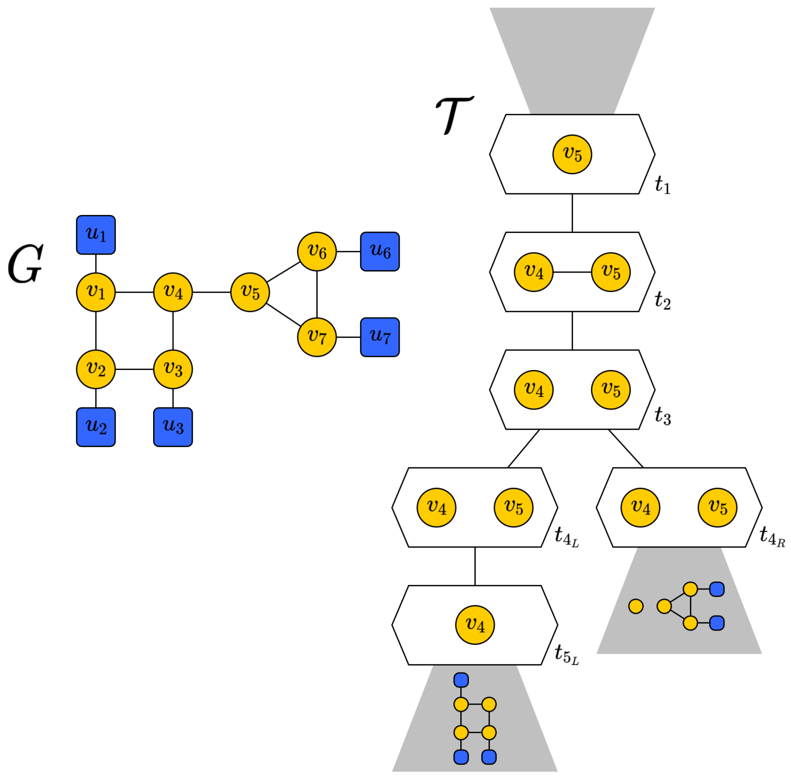


Figure 4.8: An example-graph G, and (part of) some "nice" tree decomposition $\mathcal{T}$ of G

In this section we will show with examples for several nodes $t \in T$, in a bottom-up fashion, how the solutions set L_t is built up given a certain partial solution l'_t (or in the case of the "join sub-trees"-node, partial solutions l_{t_1}, l_{t_2}):

- **Node t_{4_L}:**
  An "introduce vertex"-node, introducing vertex v_5
- **Node t_3:**
  An "join sub-trees"-node, joining the sub-trees rooted at nodes t_{4_L}, t_{4_R}
- **Node t_2:**
  An "introduce edge"-node, introducing edge (v_4, v_5)
- **Node t_1:**
  An "forget vertex"-node, forgetting vertex v_4

Solving the node t_{4_L}

In the example graph G and corresponding decomposition tree $\mathcal{T}$ (see figure 4.8), the node t_{4_L} is considered to be an "introduce vertex"-node. It introduces the vertex v_5.

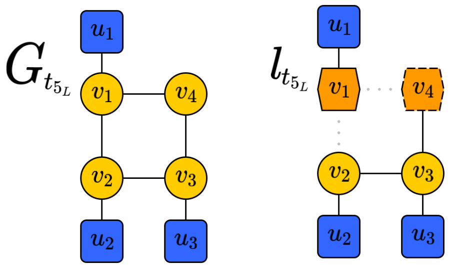


Figure 4.9: Sub-graph $G_{t_{5_L}} \subseteq G$, and some example solution $l_{t_{5_L}} \in L_{t_{5_L}}$

We consider the solution $l_{t_{5_L}} \in L_{t_{5_L}}$ as depicted in figure 4.9. It contains two facility trees $\Phi = \{\Psi_{v_1}, \Psi_{v_4}\}$, such that the set of facilities $F = \{v_1\}$, and the set of pseudo-facilities $F_P = \{v_4\}$.

The goal is to build a list of all valid candidate partial solutions for the sub-graph $G_{t_{4_L}}$ as depicted in figure 4.11, given the solution $l_{t_{5_L}}$

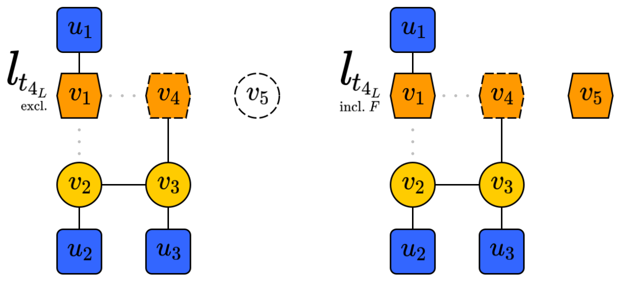


Figure 4.10: The three possible ways of expanding example solution $l_{t_{5_L}} \in L_{t_{5_L}}$

There are two possible ways in which we can extend the solution set $L_{t_{4_L}}$ by updating the solution $l_{t_{5_L}}$, which are depicted in figure 4.10:

- We do not include the vertex v_5, making the solution $l_{t_{4_L},\text{excl.}}$

  This is possible because $v_5 \notin U$, meaning that there is no user/client-demand on $d(v_5)$ that needs to be satisfied directly. If there were a demand $d(v_5)$, not including v_5 in the solution would have cause the solution to be invalid

- We do include the vertex v_5, making the solution $l_{t_{4_L},\text{incl.}}$

  We create such solutions for each facility-capacity $c \in \mathbb{Z}$ such that $0 < c \leq \text{cap}_{\text{DP}}$
  We define the leftover capacity of v_5 to be $p(v_5) \leftarrow c$.

Solving the node t_3

In the example graph G and corresponding decomposition tree $\mathcal{T}$, in figure 4.1, the node t_3 is considered to be an ”join sub-trees”-node. It joins the sub-trees rooted at t_{4_L} and t_{4_R}.

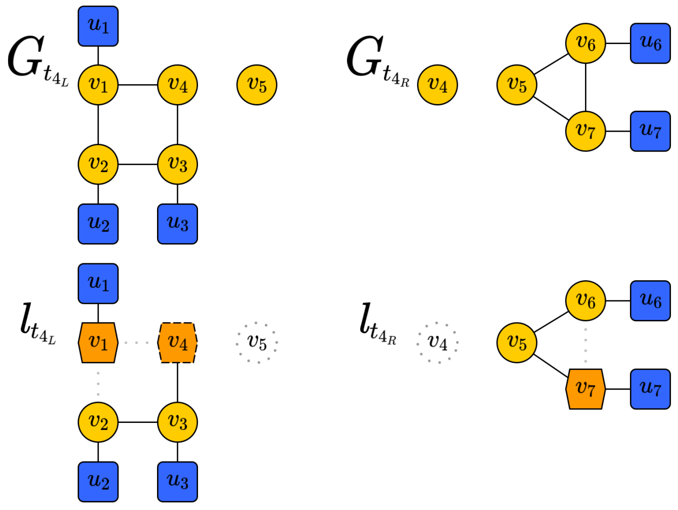


Figure 4.11: Sub-graphs $G_{t_{4_L}}, G_{t_{4_R}} \subseteq G$, and some pair of example solution $l_{t_{4_L}}, l_{t_{4_R}}$

We consider the solution $l_{t_{4_L}} \in L_{t_{4_L}}$ and $l_{t_{4_R}} \in L_{t_{4_R}}$ as depicted in figure 4.11:

- The solution $l_{t_{4_L}}$ contains two facility trees $\Phi_{4_L} = \{\Psi_{v_1}, \Psi_{v_4}\}$, such that the set of facilities $F = \{v_1\}$, and the set of pseudo-facilities $F_P = \{v_4\}$.

  Interesting to note is that this solution $l_{t_{4_L}}$ is the same solution as $l_{t_{4_L},\text{excl.}}$ from the example introducing the vertex v_5 at node t_{4_L}

- The solution $l_{t_{4_R}}$ contains exactly one facility trees $\Phi_{4_R} = \{\Psi_{v_7}\}$, such that the set of facilities $F = \{v_7\}$, and the set of pseudo-facilities $F_P = \emptyset$.

  Interesting to note is that this solution $l_{t_{4_L}}$ has left the vertex v_5 unused.

The goal is to build a list of all valid candidate partial solutions for the sub-graph G_{t_3} as depicted in figure 4.13, given the solutions $l_{t_{4_L}}, l_{t_{4_R}}$

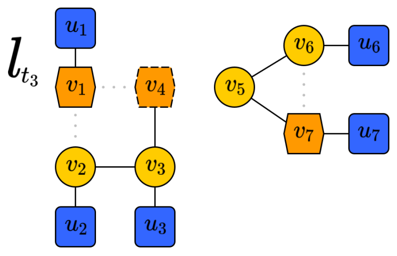


Figure 4.12: The one possible way of joining example solutions $l_{t_{4_L}}, l_{t_{4_R}}$

There is exactly one way in which we can extend the solution set $L_{t_{4_L}}$ by joining the solution $l_{t_{4_L}}, l_{t_{4_R}}$, as depicted in figure 4.12. The two pseudo-facility trees Ψ_{v_4} get merged together.

This merge is only possible when the following properties hold:

- None of the joined trees is $\Phi_3 = \{\Psi_{v_1}, \Psi_{v_3}, \Psi_{v_7}\}$ can contain a cycle.
- At both vertices v_4, v_5 on which the solutions $X_{t_{4_L}}, X_{t_{4_R}}$ overlap, it must either hold that leftover demand matches leftover supply, or it must hold that the combined leftover demand does not exceed the maximum facility capacity cap_{DP}

Solving the node t_2

In the example graph G and corresponding decomposition tree $\mathcal{T}$, in figure 4.8, the node t_2 is considered to be an "introduce edge"-node. It introduces the edge (v_4, v_5).

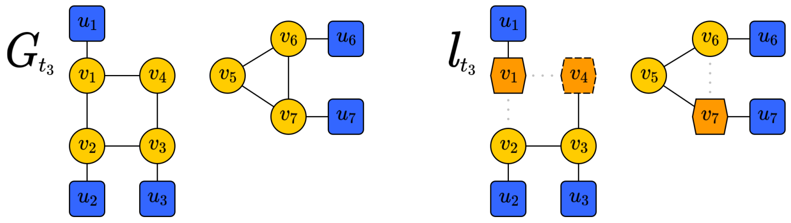


Figure 4.13: Sub-graph $G_{t_3} \subseteq G$, and some example solution $l_{t_3} \in L_{t_3}$

We consider the solution $l_{t_3} \in L_{t_3}$ as depicted in figure 4.13. It contains three facility trees $\Phi = \{\Psi_{v_1}, \Psi_{v_4}, \Phi_{v_7}\}$, such that the set of facilities $F = \{v_1, v_7\}$, and the set of pseudo-facilities $F_P = \{v_4\}$. Interesting to note is that this solution l_{t_3} is the same solution as l_{t_3} from the example joining the sub-trees at node t_3.

The goal is to build a list of all valid candidate partial solutions for the sub-graph G_2 as depicted in figure 4.15, given the solution l_{t_3}

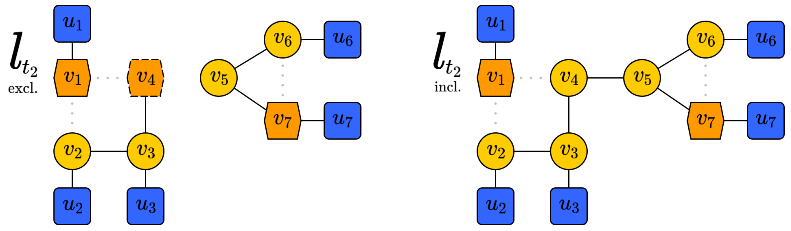


Figure 4.14: The two possible ways of expanding example solution l_{t_3}

There are two possible ways in which we can extend the solution set L_{t_2} by updating the solution l_{t_3}, which are depicted in figure 4.14:

- We do not include the edge (v_4, v_5), making the solution $l_{t_2,\text{excl.}}$
- We do include the edge (v_4, v_5), making the solution $l_{t_2,\text{incl.}}$
  We create such solutions for each facility-capacity $c \in \mathbb{Z}$ such that $\text{ABS}(p(v_4)) \leq c \leq \text{ABS}(p(v_5))$. We redefine the leftover capacities $p(v_4), p(v_5)$ as follows:

$$p_{t_2,\text{incl.}}(v_4) \leftarrow c + p_{t_3}(v_4) \qquad p_{t_2,\text{incl.}}(v_5) \leftarrow p_{t_3}(v_5) - c$$

Solving the node t_1

In the example graph G and corresponding decomposition tree $\mathcal{T}$, in figure 4.1, the node t_1 is considered to be an "forget vertex"-node. It forgets the vertex v_4.

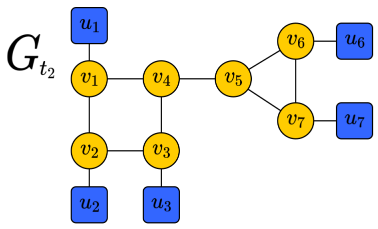


Figure 4.15: Sub-graph $G_{t_2} \subseteq G$, and some example solution $l_{t_2} \in L_{t_2}$

We consider the solution $l_{t_2} \in L_{t_2}$ as depicted in figure 4.15. It contains three facility trees $\Phi = \{\Psi_{v_1}, \Psi_{v_7}\}$, such that the set of facilities $F = \{v_1, v_7\}$, and the set of pseudo-facilities $F_P = \emptyset$. Interesting to note is that this solution l_{t_2} is the same solution as $l_{t_2,\text{incl.}}$ from the example joining the sub-trees at node t_2.

The goal is to build a list of all valid candidate partial solutions for the sub-graph G_{t_1}, given the solution l_{t_2}. We do this by checking whether the solution l_{t_2} is still valid after the vertex v_4 is removed from the vertex set X_{t_2}, making X_{t_1}

The solution l_{t_1} depicted in figure 4.16 is a valid candidate partial solution in L_{t_1} if $p(v_4) = 0$.

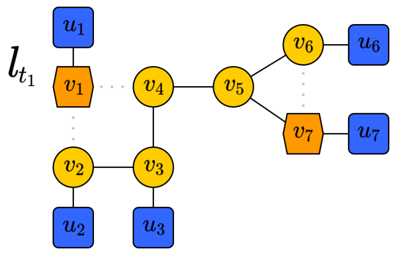


Figure 4.16: Valid candidate partial solution l_{t_1}

4.2 Trim Function

The algorithm as described in section 4.1 is capable of correctly finding an optimal solution to the problem definition as described in section 3.1.

However, in the current form, we keep track of the set of *all* valid partial solutions L_t at every node along the tree t. As we iterate up the tree this becomes rather inefficient in terms of running time, as it is akin to essentially brute-forcing our way to the solution.

For this reason we need to come up with a way of reducing the amount of solutions that we need to keep track of.

4.2.1 Goal of the Trim Function

Given a sub-problem solution $l_t = (p_t, F_t, \mu_t, \Phi_t, \Pi_t)$ for some node $t \in T$, and a full solution $l = (F, \mu, \Phi, \Pi)$ on the corresponding graph G, then l_t is a partial solution to l iff:

- It holds that $F_t \subseteq F$, meaning that any facility $f \in F_t$ (but not necessarily any pseudo-facility $f \in F_P$) is also included in F
- It holds that for each $\Psi'_f \in \Phi_t$ that there exists a $\Psi_f \in \Phi$ such that $\Psi'_f \subseteq \Psi_f$.
- It holds that for each $\mu_t(f) \in \mu_t$ that there exists a $\mu(f) \in \mu$ such that $\mu_t(f) \subseteq \mu(f)$.

It is not needed to keep track of every single possible partial solution in order to build up an optimal solution l_{OPT}. At various points along the tree T we are already able to determine whether some partial solution $l_t \in L_t$ is definitely not going to be partial to any optimal solution l_{OPT}.

We wish to run a trim function after calculating L_t for each node $t \in T$, with which we trim down the number of solutions we keep track of going forward. The resulting set $L_{t,\text{trimmed}} \subseteq L_t$ will be passed on to the next node in tree T.

However, we need to keep in mind that for each node $t \in T$, the set L_t is defined as a set of valid partial solutions l_t on sub-graph G_t, for which it must hold that there exists at least one partial solution $l_{t,\text{OPT}} \in L_t$ such that $l_{t,\text{OPT}}$ is the partial solution to some optimal solution l_{OPT} on graph G.

This means that we need to make sure that we do not "over-trim" in the trim function, such that we forget about solutions that are essential to reaching a valid optimal solution l_{OPT} on graph G. The trick of a good trim function is to find a balance between picking enough solutions to be trimmed off, and not trimming too many solutions.

4.2.2 Trim Function

Defining the Trim IDs

We consider a node $t \in T$ and the corresponding vertex set X_t.
Q_t denotes the set of all possible set partitions of vertex set X_t

We define a set of all possible trim permutations M_t.
Each trim permutation $m_t = (p_t, q_t)$, for $m_t \in M_t$, is defined as follows:

- $p_{m,t} : X_t \to \mathbb{Z}$ denotes a mapping indicating the "leftover capacity" for each vertex $x \in X_t$ For each vertex $x \in X_t$ it holds that $-\text{cap}_{\text{DP}} \leq p(x) \leq \text{cap}_{\text{DP}}$.
- $Q'_t \in Q_t$ denotes a partition of the vertices in X_t into separate trees.

Trimming the Solutions by Trim ID

We define a mapping $\xi : L_t \to M$ which maps each partial solutions $l_t \in L_t$ to exactly one trim-IDs $m \in M$ such that:

- For each vertex $x \in X_t$ it must hold that: $p_{m,t}(x) = p_t(x)$
- For each component $q_t \in Q'_t$ it must hold that there exists some facility $f \in (F \cup F_P)$ such that $V(\Psi_f) \cap X_t = q_t$.

We then define the set of trimmed solutions $L_{t,\text{trimmed}}$ as follows:

$$L_{t,\text{trimmed}} \leftarrow \bigcup_{m_t \in M_t} \left(\text{argmin}_{l_t \in \xi^{-1}(m_t)} (cost_{l_t}) \right)$$

In other words, for each possible trim permutation we only keep track of the lowest cost solution.

4.2.3 Solution Bound

By trimming the solution set L_t for each node $t \in T$ set by all possible trim permutations M_t, we are able to guarantee an upper bound on the size of set $L_{t,\text{trimmed}}$ for node t.

The size of Q_t is defined by the Bell number B_n, which is recursively defined as follows:

$$B_0 = 1 \qquad B_{k+1} = \sum_{i=0}^{k} \binom{k}{i} B_i$$

Since Q_t contains all possible set permutations of X_t, it must hold that $|Q_t| = B_{|X_t|}$
We also know that for each vertex $x \in X_t$ there are $(2 \cdot \text{cap}_{\text{DP}} + 1)$ possible values for $p_{m,t}$

Together this means the bound on the size of solution set $L_{t,\text{trimmed}}$ is:

$$B_{|X_t|} \cdot (2 \cdot \text{cap}_{\text{DP}} + 1)^{|X_t|}$$

As defined by the problem description in chapter 3, we know that the maximum facility capacity $\text{cap}_{\text{DP}} = 48$. Additionally, if we assume that the underlying graph G of tree decomposition $\mathcal{T}$ is an outer planar then there must exist a solution such that $|X_t| \leq 3$ for each node $t \in T$.

Given these limitations, $L_{t,\text{trimmed}}$ can never contain more than:

$$\begin{aligned} B_3 \cdot 97^3 &= 5 \cdot 97^3 \\ &= 4.56 \cdot 10^6 \text{ solutions} \end{aligned}$$

4.2.4 Correctness of the Trim Function

In order for the trim function to be both correct and useful, it is essential that we trim enough solutions to have a meaningful gain in running time, but it is also important that the solutions are not over-trimmed.

Over-trimming the set of solutions means that we forget some partial solution that would have led to a more cost-efficient solution than the most cost-efficient solution we can find with the partial solutions that we keep. In other words, if we forget a partial solution, we must be certain that is could not possibly be part of any optimal solution.

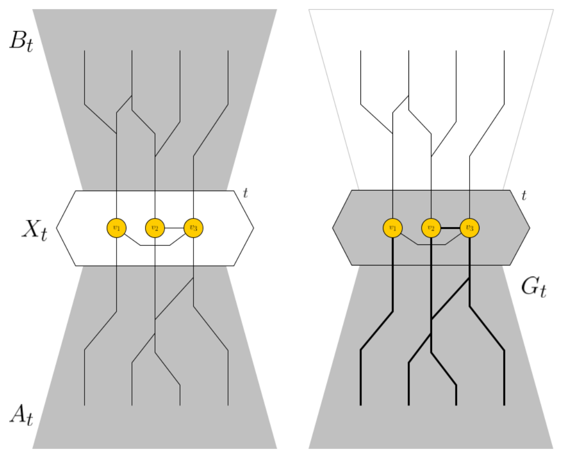


Figure 4.17: Vertex sets A_t, X_t, B_t and sub-graph G_t at some node $t \in T$

We consider a node $t \in T$, with the corresponding vertex sets A_t, X_t, B_t. By the definition of a tree decomposition, there exist no edges between any vertices in set A_t and B_t. This means that for any given solution, each facility tree that contains vertices in A_t and B_t, must also contain vertices in X_t.

The trim function keeps track of a partial solution for each possible way to connect the vertices in X_t, which each possible amount of "leftover capacity". The component that gets trimmed is the structure that these partial solutions have in A_t. Only the cheapest partial solution remains.

We consider a partial solution $l_{\text{OPT}}[A_t \cup X_t]$ that gets selected by the trim function. This solution will lead to the lowest cost solution l_{OPT} on the overall graph, out of all solutions in $L_{t,\text{trimmed}}$. We also consider a different partial solutions $l'[A_t \cup X_t]$ such that $l_{\text{OPT}}[X_t] = l'[X_t]$, but $l_{\text{OPT}}[A_t] \neq l'[A_t]$.

Since $l_{\text{OPT}}[X_t] = l'[X_t]$, and since $l_{\text{OPT}}[A_t \cup X_t]$ is the solution that gets accepted by the trim function, this must mean that $\text{cost}(l_{\text{OPT}}[A_t \cup X_t]) \leq \text{cost}(l'[A_t \cup X_t])$.
Since the solutions $l_{\text{OPT}}[X_t] = l'[X_t]$, it must hold that for any optimal way to solve the remainder on B_t for $l_{G_t,\text{OPT}}$ must also be an optimal way to solve the remainder on B_t for l'_{G_t}. Therefor the $\text{cost}(l_{\text{OPT}}[X_t \cup B_t]) = \text{cost}(l'[X_t \cup B_t])$

This means that any solution that is forgotten by the trim function, cannot be more optimal than the solution of the same trim permutation that is kept track of.

4.3 Running Time

In this section we will shortly elaborate on the running time of the algorithm, given a graph G, a tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ and a maximum capacity cap_{DP}.

Determining the running time, means we have to consider three elements:

- The number of nodes in the tree decomposition,
- The number of solutions (or pairs of solutions) that need to be checked at each node,
- The amount of time spent per (pair of) solutions.

This results in a running time that is build up as follows:

$$\text{O}(\#\text{nodes in tree} \cdot \#\text{solution-(pairs) at node} \cdot \#\text{time per solution-(pair)})$$

We can define the number of nodes in the tree decomposition as follows:

$$\text{O}(V(T)^{\text{O}(1)})$$

For each of the five different types of nodes, we can define the number of (pairs of) solutions, and the number of operations as follows:

Node Type	**Max. #Solution-(pairs)**	**Time/Solution-(pair)**
"Introduce edge"-node	$B_{\lvert X_t \rvert} \cdot (2 \cdot \text{cap}_{\text{DP}} + 1)^{\lvert X_t \rvert}$	$\text{O}(\text{cap}_{\text{DP}})$
"Introduce vertex"-node	$B_{\lvert X_t \rvert} \cdot (2 \cdot \text{cap}_{\text{DP}} + 1)^{\lvert X_t \rvert}$	$\text{O}(\text{cap}_{\text{DP}})$
"Forget edge"-node	$B_{\lvert X_t \rvert} \cdot (2 \cdot \text{cap}_{\text{DP}} + 1)^{\lvert X_t \rvert}$	$\text{O}(1)$
"Join sub-trees"-node	$(B_{\lvert X_t \rvert} \cdot (2 \cdot \text{cap}_{\text{DP}} + 1)^{\lvert X_t \rvert})^2$	$\text{O}(\lvert X_t \rvert^2)$
"Leaf"-node	1	$\text{O}(1)$

This means that the "join sub-trees"-node is by far the most impactful node in terms of running time, which results in the following worst-case running time of our algorithm:

$$\text{O}\left(|V(T)|^{O(1)} \cdot \left(B_{|X_t|} \cdot (2 \cdot \text{cap}_{\text{DP}} + 1)^{|X_t|}\right)^2 \cdot (|X_t|)^2\right)$$

Interestingly if we assume that our tree decomposition is in fact a path-decomposition $\mathcal{P} = (P, \{X_p\}_{p \in V(P)})$, and thus does not contain any "join sub-trees"-nodes, this would lower our running time to be:

$$\text{O}\left(|V(P)|^{O(1)} \cdot \left(B_{|X_p|} \cdot (2 \cdot \text{cap}_{\text{DP}} + 1)^{|X_p|}\right) \cdot \text{cap}_{\text{DP}}\right)$$

As defined by the problem description in chapter 3, we know that the maximum facility capacity $\text{cap}_{\text{DP}} = 48$. Additionally, if we assume that the underlying graph G of tree decomposition $\mathcal{T}$ is outer-planar then there must exist a solution such that $|X_t| \leq 3$ for each node $t \in T$.

This means what we can further simplify our running time to be dependent solely on the size of the tree decomposition:

$$\begin{aligned} &\text{O}\left(|V(T)|^{O(1)} \cdot \left(B_3 \cdot (2 \cdot 48 + 1)^3\right)^2 \cdot (3)^2\right) \\ &= \text{O}\left(|V(T)|^{O(1)} \cdot \left(5 \cdot (97)^3\right)^2 \cdot 9\right) \\ &= \text{O}\left(|V(T)|^{O(1)} \cdot 1.87 \cdot 10^{14}\right) \\ &= \text{O}\left(|V(T)|^{O(1)}\right) \end{aligned}$$

Chapter 5

Evaluation & Discussion

This section elaborates on how the algorithm has been implemented for experimental evaluation, and how the results compare to the old approach using Steiner trees, as detailed by A. van den Boogaart [1]. We will also discuss the implications of as well as the limitations of our approach using tree decomposition.

5.1 Implementation of the Algorithm

For calculating the optimal solution to the capacitated facility location / network design problem on graphs of bounded treewidth, we have implemented a proof-of-concept using Python (version 3.2) with the NetworkX package (version 2.5).

This proof-of-concept requires a pre-calculated graph G of bounded treewidth and a corresponding tree decomposition $\mathcal{T}$ of graph G as input. Synthetic test-cases of graph G and tree decomposition $\mathcal{T}$ can be created externally and loaded in using graphml-files.
If the tree decomposition has not been pre-calculated, it can optionally be generated using the models TREEWIDTH.TREEWIDTH_MIN_DEGREE() or TREEWIDTH.TREEWIDTH_MIN_FILL_IN() from the NetworkX package. These modules approximate a tree decomposition using the "min degree"-heuristic, or the "min fill in"-heuristic respectively.

While the implementation of the algorithm uses the same concepts of tree-decomposition to build up a solution, it is important to note that there is a key discrepancy between the theoretical description of the algorithm from chapter 4 and the implementation in Python. This discrepancy causes the Python implementation to be several orders of magnitude slower than the theoretical algorithm description would be.

The reason for the existence of this discrepancy is that after the implementation was finished, we managed to find a way to significantly speed up the algorithm. The algorithm as described in Chapter 4 follows this improved version of the algorithm. However, due to time constraints we were unfortunately unable to similarly improve the Python implementation.

Despite this discrepancy, we believe that this implementation, while unfortunately significantly slower than it could have been, still functions as a proof-of-concept for the use of tree-decompositions on graphs of bounded treewidth.

Short Explanation of the Discrepancy

Between the algorithm as described in chapter 4, and the algorithm as implemented in Python, there is a difference in the way leftover capacities are kept track of. This difference results in the theoretical definition of the algorithm having a significantly faster running time than version of the algorithm implemented in Python.

The algorithm as described in Chapter 4 keeps track of a singular leftover capacity $p(x)$ for each vertex $x \in X$. This value $p(x)$ denotes the "leftover supply" ($p(x) > 0$) or the "leftover demand" ($p(x) < 0$), but can never denote both at the same time.
The algorithm as implemented in Python keeps track of two separate values $p(x)$ and $q(x)$ for each vertex $x \in X$, where $p(x)$ denotes the "leftover supply", and $q(x)$ denotes the "leftover demand". Since $p(x)$ and $q(x)$ are independent from one another, this means that any vertex $x \in X$ can have both a leftover supply and a leftover demand at the same time.

Similar to the algorithm as described in chapter 4, the trim function in the Python implementation needs to keep track of each unique leftover capacity for each vertex $x \in X$. However, since the leftover capacity is now denoted by two distinct variables $p(x)$ and $q(x)$, this means we double the amount of variables that we need to keep track of at any given node in the tree decomposition.

This significantly increases the bounded size of the set of trimmed solutions $L_{t,\text{trimmed}}$:

$$B_{|X_t|} \cdot ((\text{cap}_{\text{DP}} + 1)^2)^{|X_t|})$$

If we assume $\text{cap}_{\text{DP}} = 48$, and $|X_t| \leq 3$ for each node $t \in T$, then this means the size of trimmed solution set $L_{t,\text{trimmed}}$ can contain up to:

$$\begin{aligned} B_3 \cdot (48^2)^3 &= 5 \cdot (49^2)^3 \\ &= 5 \cdot 2401^3 \\ &= 6.92 \cdot 10^{10} \text{ solutions} \end{aligned}$$

If we compare this solution bound to the solution bound as defined for the theoretical algorithm in chapter 4.2, it is very obvious that the Python implementation of the algorithm is several orders of magnitude less efficient than the theoretical description algorithm.

5.2 Evaluation of Results

In this section we will be experimentally evaluating our approach to solving the facility location network design problem on graphs of bounded treewidth, and comparing this to the solutions that the old approach using Steiner trees, as detailed by A. van den Boogaart [1], provides on those same data sets.

The goal set out for this thesis is to explore approaches that come up with qualitatively better solutions to the facility location problem. To this end we will be focusing our result evaluation on the difference in quality between the solutions provided by the old approach using Steiner trees, and our approach using tree decomposition.
Because the Python implementation is several orders of magnitude slower when compared to the theoretical algorithm as described in Chapter 4, comparing the time it takes to complete these experiments does not accurately reflect the running time of the approach described in this paper.

Section 5.2.1 describes the manner in which the results were obtained. Sections 5.2.2 and 5.2.3 show the results on several example graphs. The former focuses on a complex example graph that is more representative of real world neighbourhoods, while the latter focuses on simple example graphs that are each meant to showcase specific inefficiencies in the Steiner tree approach.

5.2.1 Experimental Setup

The graphs on which we will be comparing the Steiner tree approach to our tree decomposition approach, are synthetic examples designed to showcase situations in which the Steiner tree approach does not provide an optimal result. They are scaled down examples that are modelled after realistic scenarios.

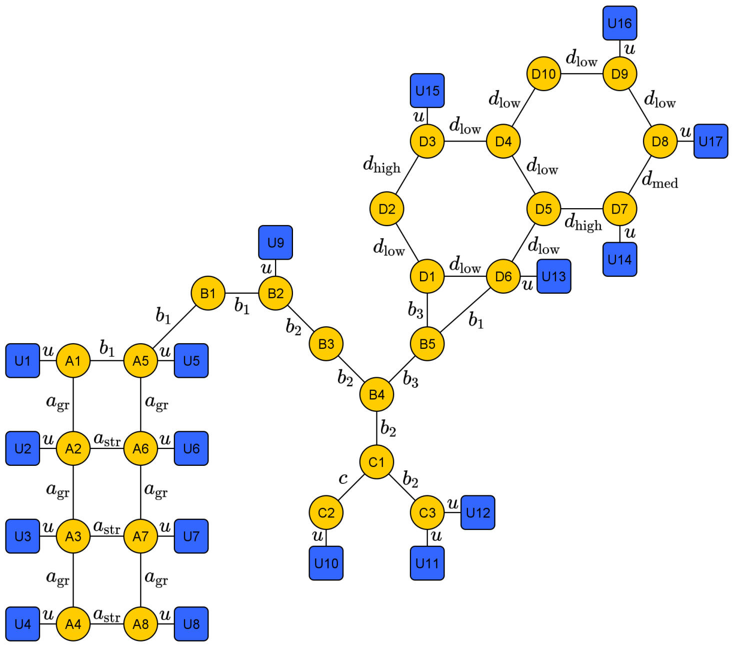
Each graph is designed to be an outer-planar graph, which we know to have a treewidth of 2. This means that there must exist a tree decomposition for this graph which has a maximum bag size of 3.

The graphs are represented as undirected NetworkX graphs, and are fed into the respective implementations based on Steiner trees and tree decomposition. In the case of the tree decomposition approach, a tree decomposition for the example graph is generated using the TREEWIDTH.TREEWIDTH_MIN_DEGREE() function in the NetworkX package.

In order to speed up the calculation process, each example graph has been treated with a simple pre-processing step, which finds all users of degree 1, and moves the user vertex to the neighbouring vertex (with the exception of situations where that would clash with another user). This should not change the solution found by either approach, but does help reduce the number of slow join-nodes that get generated by the default NetworkX package.

5.2.2 General Synthetic Example

We consider the graph of potential edges as depicted in figure 5.1. It is a somewhat complex example graph that is supposed to be a small-scale synthetic representation of what a real-world neighbourhood might look like.



Edge	$\mathrm{cost}_{\mathrm{dig}}$	$\mathrm{cost}_{\mathrm{cable}}$
a_{gr}	2.00	0.50
a_{str}	6.00	1.00
b_1	5.00	1.00
b_2	1.00	1.00
b_3	2.00	1.00
c	5.00	2.00
d_{low}	1.00	0.50
d_{med}	9.00	4.00
d_{high}	10.00	2.00
u	1.00	0.10

Facilities	
$\mathrm{cost}_{\mathrm{DP}}$	1000
$\mathrm{cap}_{\mathrm{DP}}$	6

Figure 5.1: General Synthetic Example

The left table details the specific digging cost $\mathrm{cost}_{\mathrm{dig}}$ and cable cost $\mathrm{cost}_{\mathrm{cable}}$ for each potential edge. The right table details the cost $\mathrm{cost}_{\mathrm{DP}}$ of constructing a singular facility, as well as the maximum capacity $\mathrm{cap}_{\mathrm{DP}}$ that each facility can serve. The blue vertices represent users, each with a demand of $d(u) = 1$.

Figure 5.2 depicts the solutions on this graph, as found by the Steiner tree approach (left) and our tree decomposition approach (right) respectively. Table 5.3 describes the total cost of both of these solutions, broken down by digging cost, cable cost, and facility cost.

The tree decomposition approach results in a solution that is only 75.9% of the cost the solution resulting from the Steiner tree approach. The reason for this reduced cost, is that the tree decomposition tree approach is able to consider edges that have been previously discarded by the Steiner tree approach for having a digging cost that is too high.

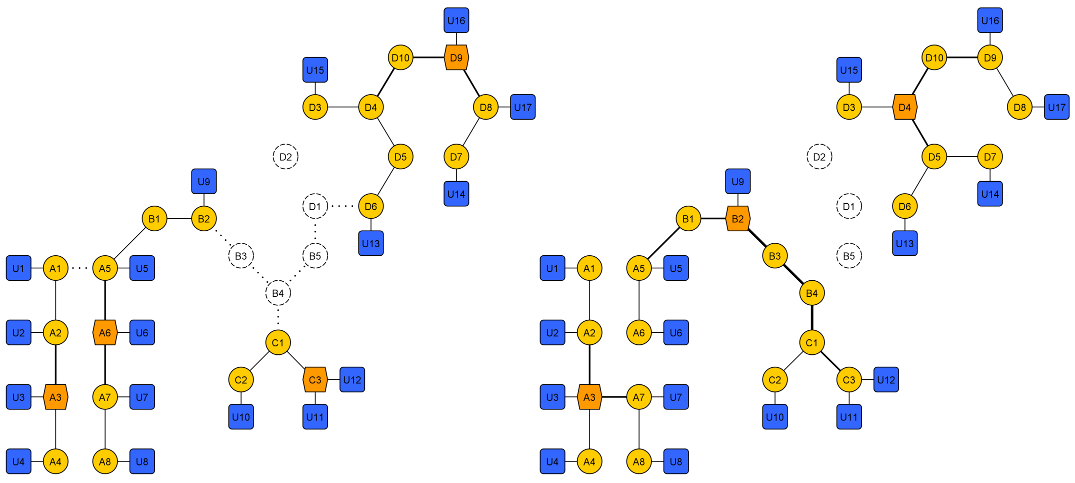


Figure 5.2: Left: Steiner tree solution. Right: Tree decomposition solution

Approach	**Digging Cost**	**Cable Cost**	**Facility Cost**	**Total Cost**
Steiner tree	60.00	19.70	4000	4079.70
Tree decomposition	68.00	30.20	3000	3098.20

Figure 5.3: Cost calculation, broken down by digging-, cable-, and facility-cost

More specifically, there are two points in which the Steiner tree approach has selected an inefficient edge, that ultimately results in a sub-optimal solution:

- In section A of the graph, there are 4 different places to cross the street. The three edges of type a_{str} have a higher digging cost than the edge of type b_1, which means that the Steiner tree approach will always picks the edge of type b_1.

  However, since facility trees are edge- and vertex- disjoint, this means that this locks the Steiner tree approach in an inefficient solution, unless other crossing locations can be considered.

- In section D of the graph, there are two ways to reach user U14. The edge of type d_{med} has a slightly cheaper digging cost compared to the edge of type d_{high}. However, that benefit is negated if you also include the cable cost of even a single cable. Then d_{high} becomes cheaper than d_{med}.

  In the Steiner tree approach, the option of using the edge of type d_{high} is locked out outright, because d_{high} was not part of the underlying Steiner tree.

The Steiner tree approach is a decent approach in most cases, but as shown in this example, there exist scenarios in which significant gains can be made, simply by also considering edges that would result in a more expensive digging cost.

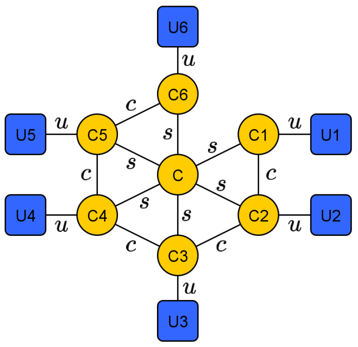
5.2.3 Specific Synthetic Examples

We will look at three smaller and more specific examples, which are designed to show of scenarios in which the Steiner tree approach does not select the most optimal solution.

Using the tree decomposition approach on each example will result in a better solution. It is interesting to note how significant the gains in each example really are.

Wheel and Spoke Example

We consider the graph of potential edges as depicted in figure 5.4. It is a simple example, in which the cable cost along each edge is the same, but the cost of digging each edge along the circumference is only a fraction cheaper than $\frac{6}{5}^{\text{ths}}$ the digging cost of an edge along the spokes.



Edge	cost_{dig}	$\text{cost}_{\text{cable}}$
c	119.00	40.00
s	100.00	40.00
u	1.00	0.10

Facilities	
cost_{DP}	1000
cap_{DP}	6

Figure 5.4: Wheel and Spoke Example

The left table details the specific digging cost cost_{dig} and cable cost $\text{cost}_{\text{cable}}$ for each potential edge. The right table details the cost cost_{DP} of constructing a singular facility, as well as the maximum capacity cap_{DP} that each facility can serve. The blue vertices represent users, each with a demand of $d(u) = 1$.

This is a synthetic example that is designed to showcase that pre-determining which edges have to be dug, could result in a solution where you are wasting more money on cabling cost than you gain on digging cost. It is important to note that when planning realistic fiber-optics networks, such a circular layout would likely never occur

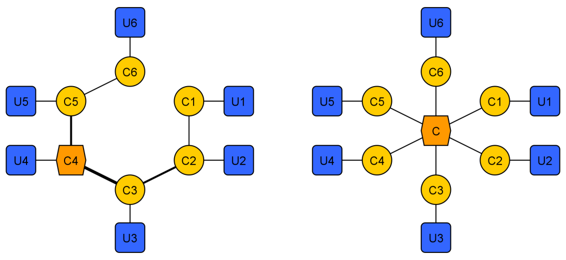


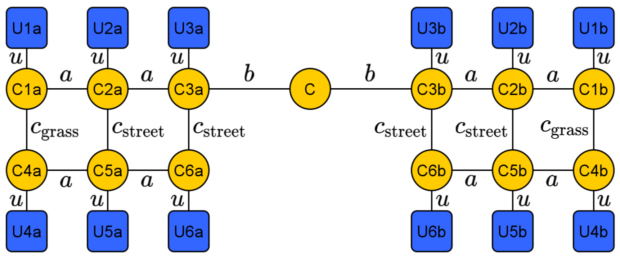
Figure 5.5: Left: Steiner tree solution. Right: Tree decomposition solution

Approach	**Digging Cost**	**Cable Cost**	**Facility Cost**	**Total Cost**
Steiner tree	601.00	360.60	1000	1961.60
Tree decomposition	606.00	240.60	1000	1846.60

Figure 5.6: Cost calculation, broken down by digging-, cable-, and facility-cost

Figure 5.5 depicts the solutions on this graph, as found by the Steiner tree approach (left) and our tree decomposition approach (right) respectively. Table 5.6 describes the total cost of both of these solutions, broken down by digging cost, cable cost, and facility cost.

The tree decomposition approach results in a solution that is 94.1% of the cost the solution resulting from the Steiner tree approach. This is a minor improvement, and may not be significant enough to weigh up against the increased running time.



Edge	$\textbf{cost}_{\textbf{dig}}$	$\textbf{cost}_{\textbf{cable}}$
a	10.00	2.50
b	50.00	5.00
c_{street}	30.00	2.50
c_{grass}	29.00	2.50
u	1.00	0.10

Facilities	
cost_{DP}	1000
cap_{DP}	12

Figure 5.7: Street Crossing Example

Street Crossing Example

We consider the graph of potential edges as depicted in figure 5.7. It is a simple example, which represents two dead-end streets, that can both be covered by exactly one facility. It is a fraction cheaper to dig across the street at the end of the street, but that would involve routing all cables via this edge.

The left table details the specific digging cost cost_{dig} and cable cost $\text{cost}_{\text{cable}}$ for each potential edge. The right table details the cost cost_{DP} of constructing a singular facility, as well as the maximum capacity cap_{DP} that each facility can serve. The blue vertices represent users, each with a demand of $d(u) = 1$.

This example is designed to showcase the real scenario of the Steiner tree approach picking inefficient street-crossing locations.

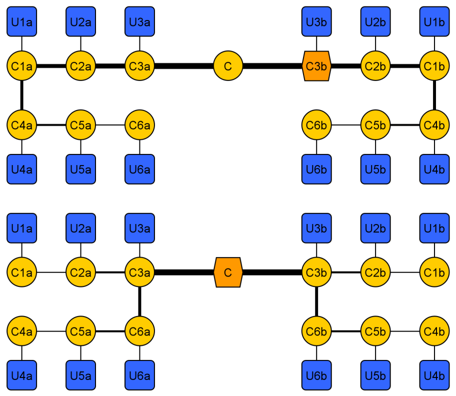


Figure 5.8: Top: Steiner tree solution. Bottom: Tree decomposition solution

Approach	Digging Cost	Cable Cost	Facility Cost	Total Cost
Steiner tree	250.00	136.20	1000	1386.20
Tree decomposition	252.00	106.20	1000	1358.20

Figure 5.9: Cost calculation, broken down by digging-, cable-, and facility-cost

Figure 5.8 depicts the solutions on this graph, as found by the Steiner tree approach (left) and our tree decomposition approach (right) respectively. Table 5.9 describes the total cost of both of these solutions, broken down by digging cost, cable cost, and facility cost.

The tree decomposition approach results in a solution that is 97.9% of the cost the solution resulting from the Steiner tree approach. This is an absolutely minuscule improvement, and in terms of cost gained it might not weigh up against the increased running time.

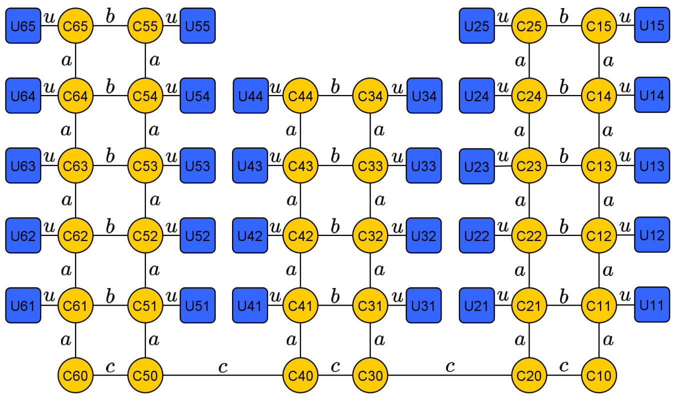
However, it is important to note that this does bear closer resemblance to what engineers would actually like to draw out by hand. Whether that is worth the additional running time is debatable.

Inefficient Facility Mapping Example

We consider the graph of potential edges as depicted in figure 5.10. It is a simple example, which represents three streets laid out in such a way that each of them is just slightly too small to cover a fully used facility. Since multiple crossings cannot be considered, it becomes impossible for the Steiner tree approach to fully make use of each facility

The left table details the specific digging cost cost_{dig} and cable cost $\text{cost}_{\text{cable}}$ for each potential edge. The right table details the cost cost_{DP} of constructing a singular facility, as well as the maximum capacity cap_{DP} that each facility can serve. The blue vertices represent users, each with a demand of $d(u) = 1$.

This example is designed to showcase the real scenario of the Steiner tree approach not always utilizing its facilities to the fullest extent.



Edge	**cost$_{\text{dig}}$**	**cost$_{\text{cable}}$**
a	2.00	0.20
b	3.00	0.20
c	1.00	0.10
u	1.00	0.10

Facilities	
cost$_{\text{DP}}$	1000
cap$_{\text{DP}}$	7

Figure 5.10: Example network generated by the current approach using Steiner trees

Figure 5.11 depicts the solutions on this graph, as found by the Steiner tree approach (left) and our tree decomposition approach (right) respectively. Table 5.12 describes the total cost of both of these solutions, broken down by digging cost, cable cost, and facility cost.

The tree decomposition approach results in a solution that is 67.4% of the cost the solution resulting from the Steiner tree approach. This is a huge improvement, and the clearest practical example of a scenario where the tree decomposition approach could be worth the additional time required to calculate a solution.

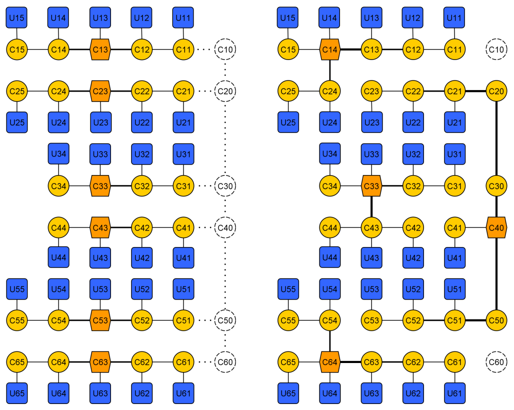


Figure 5.11: Left: Steiner tree solution. Right: Tree decomposition solution

Approach	**Digging Cost**	**Cable Cost**	**Facility Cost**	**Total Cost**
Steiner tree	72.00	9.20	6000	6081.20
Tree decomposition	84.00	12.10	4000	4096.10

Figure 5.12: Cost calculation, broken down by digging-, cable-, and facility-cost

5.3 Discussion of Algorithm

The algorithm proposed by A. van den Boogaart [1] calculates a Steiner tree on the underlying graph, with edge-weight based on the digging cost of each edge, such that the users/clients are its terminals. The solution to CFLP is then calculated in polynomial time on this Steiner tree.

However, the problem with this Steiner tree based-approach is that the step in which the Steiner tree is pre-calculated can only account for the digging cost of the network. It is blind to how the facility- and cable-cost in the subsequent step of solving CFLP will affect the total cost. This causes the algorithm to potentially lock itself in on an inefficient network design, as potential edges which might be part of a more optimal solution to CFLP get discarded in the Steiner tree step.

The key advantage of our tree decomposition algorithm, when compared to the Steiner tree algorithm, is that these potential edges do not always need to be dropped from consideration. Whereas the Steiner tree algorithm requires the underlying graph of potential edges to be a tree, our tree decomposition algorithm is able to efficiently find an optimal solution to CFLP on graphs of bounded treewidth.
This means that our underlying graph is allowed to contain some cycles, which allows us to re-introduce some of these potential edges that would have been dropped by the Steiner tree algorithm.

Being able to deviate from the Steiner tree provides us with two key opportunities to improve the solution:

- We can consider more efficient ways to connect users/clients to their respective facilities.

  Sometimes the cheapest edge to dig, does not result in a solution that efficiently routes the cables from facility to user/client. By routing the cables via an edge (or collection of edges) that has a slightly higher digging cost, we could save enough money in cabling cost to outweigh the additional expense.

  The benefit of this improvement is shown in the ”wheel and spoke” and ”street crossing” examples of section 5.2.3

- We can reduce the total number of facilities that are required.

  For any valid solution it is required that any two facility trees are both vertex- and edge-disjoint. If the underlying tree on which CFLP gets solved is predetermined without taking facility location into account, this can result in situations where facilities simply cannot be utilized to their full capacity.

  Being able to include some additional edges of interest could break us out of this sub-optimal facility distribution, by considering additional different mappings between users and facilities.

  The benefit of this improvement is shown in the ”inefficient facility mapping” example of section 5.2.3

If we can include the appropriate edges in the solution, then we can potentially find significantly cheaper solutions than we could with the Steiner tree algorithm. It is important to note however that this does come at the expense of a greater running time.

5.3.1 Limitations of Algorithm

It is however important to note that the algorithm presented in this thesis is not perfect, and still presents some limitations. In this section we will discuss some of these limitations.

Assumptions about Graph Input

One of the limitations of the algorithm as described in this paper, is that we make assumptions about the graph of potential edges that we assume as an input.

In this paper it is not detailed how real-world geographical data is translated to a graph of potential edges. We assume that this work has already been taken care of, and that the resulting set of potential edges is correct and does not exclude potentially optimal solutions.
Besides the Steiner tree algorithm, the paper by A. van den Boogaart [1] also explains an algorithm for converting such real-life geographical data into a graph representing potential edges.

At minimum including the potential edges of the Steiner tree, that has been determined by the Steiner tree approach, assures that the tree decomposition approach will always find a solution that is at least as cost-efficient as the solution found by the Steiner tree approach. However, picking any additional ”edges of interest” to include, becomes non-trivial when you consider that the resulting graph must have a bounded treewidth.
This paper does not explain how to pick these ”edges of interest”. We just show that including certain edges of interest as additions to the Steiner tree, could lead to more optimal solutions.

A potential metric to look into is the metric of dilation, in which we compare the distance between two points on the Steiner tree, with those same two points on the graph of potential edges. If the difference between these two metrics is high (if the dilation is high), then it might be worth including that edge as an ”edge of interest”.
The applicability and usefulness of this approach does require further research, however.

Building the Tree Decomposition

Another limitation is that in this paper we do not describe how to build the tree decomposition that is used by the algorithm. We assume that the tree decomposition has already been built, and that the bag size is limited.

In the implementation of the algorithm, we utilize the TREEWIDTH.TREEWIDTH_MIN_DEGREE() function from the NetworkX package. This method uses the ”min-degree”-heuristic to approximate an optimal tree decomposition.

We could also apply further pre-processing steps on the generated tree decomposition, which could help speed up the algorithm calculating CFLP on the tree decomposition:

- Reducing the number of complex join nodes from the tree decomposition could be one such pre-processing step.

  The join node is the node type that is by far the most inefficient node to solve in our proposed algorithm. This is because it involves comparing all solutions from the left child with all solutions from the right child.

  As shown in the section on the trim function, the size of these partial solution sets are largely dependent on bag-size at that node. If we could try to reduce join nodes $t \in T$ with bags $|X_t| = 3$, and instead replace them with join nodes with bags $|X_t| = 2$, this could potentially speed up the algorithm by several orders of magnitude.

- Another pre-processing step that could be applied is one we have actually manually applied in the experimental evaluation.

  It involves finding all users in the graph of degree 1, and moving the user to its neighbouring vertex. This will still help find an optimal solution that can be easily converted back to an optimal solution for the original graph, but cuts down significantly on the number of (possibly inefficient) join-nodes when generating the tree decomposition.

Potentially there may exist other additional pre-processing steps that might help improve the time required to find an optimal solution.

Fiber-Optics Network not Complete

It is also important to note that the algorithm as described in this paper does not find the solution to an entire fiber-optics network. The scope has been limited to just calculating the connection between DPs and users/clients. The connections between DPs and APs, and between APs and CPs has been ignored in this algorithm.

These higher-level decision problems are in effect the same problem as CFLP, only one level up. To that end, you could run the same algorithm multiple times to build up the solution layer by layer. This however comes with a similar drawback as the Steiner tree approach has, in that each layer being calculated is blind to any subsequent layers that might need to be calculated. This has a high likelihood for one layer to lock subsequent layers into inefficient solutions.

Another way to tackle this limitation could be by considering the k-Level Facility Location Problem on graphs of bounded treewidth. However, this may have the potential of having an exploding running time, meaning that in practice it may not be quite as useful.
More future research may be required

5.3.2 Future Work

As discussed in the limitations of the algorithm, section 5.3.1, there still exist many extensions, improvements and further additional work that that can be applied to the work presented in this paper, and to the field of fiber-optics network planning.

Some interesting further areas of research might be:

- Speeding up the implementation of the algorithm, so that it can be used on larger networks. This would be very useful for practical applications
- Defining methods to help determine "edges of interest" in such a way that the resulting graph remains of bounded treewidth.
- Improving upon the generation of the tree decomposition, such that the algorithm may be able to run significantly faster.

Ultimately the goal of this thesis is to find a practical solution to the problem of planning fiber-optics networks for *ThePeopleGroup*. To that end, implementing a proper version of the algorithm as presented in this paper, rather than the currently existing sub-optimal solution, is still a task that needs completing. Extending this implementation to include the additional hierarchical layers, such as the AP-to-DP layer and the CP-to-AP layer, are also some desirable additions.

Chapter 6

Conclusion

In this thesis we have set out to improve upon a pre-existing algorithm for designing fiber-optics networks based on Steiner trees, as proposed by A. van den Boogaart [1]. The algorithm proposed in this paper widens the solution space from a Steiner tree to a graph of bounded treewidth.

The problem of designing fiber-optics networks is a form of the capacitated facility location problem. We know this problem to be NP-hard on general and planar graphs. A method to efficiently find a solution to CFLP involves reducing the solution space upon which we are solving it to a point where it can be solved in polynomial time.

In this paper we propose a fixed parameter tractable algorithm to solve CFLP on any graph G, such that the parameter k defined to be equal to the treewidth of graph G. This means that for any graph of bounded treewidth k we can solve CFLP in polynomial time.
This algorithm works by solving the CFLP on a tree decomposition $\mathcal{T}$ of graph G, and relies on the principle that for any graph of a bounded treewidth k, there exists a tree decomposition with a maximum bag-size $k + 1$.

Since our algorithm is able to solve CFLP on graphs containing cycles, whereas the Steiner tree approach is limited to trees, we can selectively reintroduce certain potential edges that got discarded in the Steiner step. This allows us to widen the solution space and potentially find a more cost-efficient solution to CFLP than with the Steiner tree approach.
We do however have to be mindful that any edge that we reintroduce on the Steiner tree does not significantly increase the treewidth of the graph. Outer-planar graphs are an example of such a graph with a low treewidth, namely a treewidth of 2.

We have also implemented a variant of this algorithm using Python and NetworkX. This implementation is not as efficient as the theoretical algorithm described in this paper, but it does serve as a proof of concept that it is possible to use tree decomposition to solve CFLP in polynomial time on graphs of bounded treewidth.

Comparing the results found by our tree decomposition approach to the results found by the Steiner tree approach is very promising. In certain scenarios the tree decomposition approach is able to book results that are up to 24.1% more efficient than the Steiner tree approach.
It is however important to note that the size of this cost-reduction is largely dependent on the specific type of inefficiency that we are attempting to solve, and whether the potential edges that lead to a more optimal solution have been included in the graph of bounded treewidth.

Overall the tree decomposition algorithm is a significant step towards finding more optimal solutions towards designing fiber-optics networks. However, more work is required on the implementation in order to efficiently solve larger data sets.

Bibliography

[1] A. van den Boogaart. Efficient computation of fiber optic networks. 2018. ii, 3, 6, 7, 8, 47, 49, 56, 57, 59

[2] S. L. Hakimi. Optimum locations of switching centers and the absolute centers and medians of a graph. *Operations Research*, 12(3):450–459, 1964. 6

[3] S. L. Hakimi. Optimum distribution of switching centers in a communication network and some related graph theoretic problems. *Operations Research*, 13(3):462–475, 1965. 6

[4] S. L. Hakimi O. Kariv. An algorithmic approach to network location problems II: The p-medians. *SIAM Journal on Applied Mathematics*, 37(3):539–560, 1979. 6

[5] L. A. Wolsey G. Cornuejols, G. L. Nemhauser. The uncapacitated facility location problem. 1983. 6

[6] M. S. Daskin S. Melkote. An integrated model of facility location and transport network design. *Transportation Research Part A: Policy and Practice*, 35(6):515–538, 2001. 6, 15

[7] M. S. Daskin S. Melkote. Capacitated facility location / network design problems. *European Journal of Operational Research*, 129(3):481–495, 2001. 6, 16

[8] J. Reese. Solution methods for the p-median problem:an annotated bibliography. *Networks*, 48(3):125–142, 2006. 7

[9] K. Aardal D. B. Shmoys, É. Tados. Approximation algorithms for facility location problem. *STOC '97*, pages 265–274, 1997. 7

[10] C. Swamy R. Levi, D. B. Shmoys. LP-based approximation algorithms for capacitated facility location. *IPCO 2004. Lecture Notes in Computer Science*, 3064, 2004. 7

[11] A.J. Goldman. Optimal center location in simple networks. *Transportation Science*, 5(2):212–221, 1971. 7

[12] A. Tamir. An $O(pn^2)$ algorithm for the p-median and related problems on tree graphs. *Operations Research Letters*, 19(2):59–64, 1996. 7

[13] D. Skorin-Kapov. On some problems on k-trees and partial k-trees. 1989. 7

[14] E. N. Gilbert and H. O. Pollak. Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 16(1):1–29, 1968. 7

[15] Richard M Karp. Reducibility among combinatorial problems. *Complexity of computer computations*, page 85–103, 1972. 7

[16] D. S. Johnson M. R. Garey, R. L. Graham. The complexity of computing Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 32(4):835–859, 1977. 7

[17] R.A. Wagner S.E. Dreyfus. The Steiner tree problem in graphs. *Networks*, 3:195–207, 1971. 7

[18] A. F. Veinott Jr. R. E. Erickson, C. L. Monma. Send-and-split method for minimum-concave-cost network flows. *Mathematics of Operations Research*, 12(4). 7

[19] P. Rossmanith D. Mölle, S. Richter. A faster algorithm for the Steiner tree problem. *STACS 2006. Lecture Notes in Computer Science*, 3884:561–570, 2006. 7

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

[21] J. Chlebíková M. Chlebík. The Steiner tree problem on graphs: Inapproximability results. *Theoretical Computer Science*, 406(3):207–214, 2008. 7

[22] T. Rothvoß L. Sanità J. Byrka, F. Grandoni. An improved LP-based approximation for Steiner tree. *STOC '10: Proceedings of the forty-second ACM symposium on Theory of computing*, page 583–592, 2010. 7

[23] B. Zey M. Chimani, P. Mutzel. Improved Steiner tree algorithms for bounded treewidth. *Journal of Discrete Algorithms*, 16:67–78, 2012. 8

[24] S. Kratsch J. Nederlof H. L. Bodlaender, M. Cygan. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information and Computation*, 243:86–111, 2013. 8

[25] Marek Cygan, Fedor V. Fomin, Łukasz Kowlik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipscuk, and Saket Saurabh. *Parameterized Algorithms.* Springer International Publishing, 2015. 10, 11

[26] H. L. Bodlaender. Planar graphs with bounded treewidth. *Institute of Information and Computing Sciences*, 1988. 11

Appendix A

Linear Program

A.1 Mixed-Integer Linear Programming Definition

In this section we define the problem statement mathematically.

A.1.1 Shortest Path Problem

At the core of the facility-location problem, we wish to find the shortest (minimal-cost) path between some DP and some FTU. We define the following LP to the shortest path problem:

Given: We consider an undirected graph $G = (V, E)$, with two nodes $s, t \in V$ for which $s \neq t$. For each edge $\langle i, j \rangle \in E$, we know a given weight $c_{\langle i,j \rangle}$.

Goal: Find a minimum-cost path on G, running from s to t.

Minimize: $\displaystyle\sum_{\langle i,j\rangle \in E} c_{\langle i,j \rangle} \cdot x'_{\langle i,j \rangle}$

Subject to:

$x_{\langle i,j \rangle}, x_{\langle j,i \rangle} \in \{0,1\}$ for each edge $\langle i, j \rangle \in E$.

$$\sum_{\substack{j \in V: \\ \exists(\langle i,j\rangle \in E)}} \left(x_{\langle i,j \rangle}\right) - \sum_{\substack{j \in V: \\ \exists(\langle j,i\rangle \in E)}} \left(x_{\langle j,i \rangle}\right) = \begin{cases} 1 & \text{if } i = s \\ -1 & \text{if } i = t \\ 0 & \text{otherwise} \end{cases}$$

for each vertex $i \in V$.

$x'_{\langle i,j \rangle} \cdot 2 \geq (x_{\langle i,j \rangle} + x_{\langle j,i \rangle})$ for each edge $\langle i, j \rangle \in E$

$x'_{\langle i,j \rangle} \in \{0,1\}$ for each edge $\langle i, j \rangle \in E$

Variables: For each edge $\langle i, j \rangle \in E$, we define a two decision-variables $x_{\langle i,j \rangle}, x_{\langle j,i \rangle}$, as well as a decision variable $x'_{\langle i,j \rangle}$.
The variable $x_{\langle i,j \rangle}$ determines whether the edge $\langle i, j \rangle$ is used to from a path from s to t, specifically in the direction i to j. The variable $x'_{\langle i,j \rangle}$ determines whether the edge $\langle i, j \rangle$ is used to from a path from s to t, in either direction i to j, or j to i.

This LP considers the shortest-path problem as a flow-problem, where s is the source generating a flow of 1, and t is the sink releasing a flow of -1. Every other vertex $v \in V/\{s, t\}$ serves as neither a source or a sink, and thus must have the same in-flow as out-flow.

For every edge $\langle i,j\rangle \in E$, if it possible for the flow to run from i to j, from j to i, or in neither direction (or in both directions, but that will never be a minimal-cost solution).

A.1.2 Shortest Path Between Multiple Sources and Sinks

The Capacitated Facility-Location Problem is sadly not as simple as running the shortest path algorithm multiple times between each pair of nodes to find the most optimal paths. That approach would not consider the shared costs between paths, like the cost of constructing edges. For that reason the calculation of the shortest path between multiple sources and sinks is slightly more complex, and can be approach as an LP in multiple different ways. In this section we consider two potential approaches:

- The Edge Capacity-Based Approach
- The Path-Based Approach

Each of these approaches have their respective benefits and disadvantages.

Edge Capacity-Based Approach to Multiple Shortest Path

The capacity-based approach works by considering each edge in the graph individually, and not considering each individual path that might run over this edge. Our responsibility is to make sure that no paths end at a random edge, and we do that by making sure that the total number of paths going in is equal to the total number of paths going out of each individual edge.

This approach introduces a maximum capacity to each edge, but if this maximum capacity is set to the sum of the overall demand, then this capacity does not affect the calculation of the optimal solution.

Given: We consider an undirected graph $G = (V, E)$, with two sets of vertices $S, T \subseteq V$, such that $|S| = |T|$ and $S \cap T = \emptyset$. For each edge $\langle i,j\rangle \in E$, we know a given capacity $s(\langle i,j\rangle)$ and a given weight $c_{\langle i,j\rangle}$.

Goal: Find a minimum-cost set of edges on G, such that there exists some ordering $S = \{s_1, ..., s_k\}, T = \{t_1, ..., t_k\}$ for which there exists a path between each pair of vertices s_l, t_l, for any $1 \leq l \leq k$.

$$\begin{array}{lll}
\textbf{Minimize:} & \displaystyle\sum_{\langle i,j\rangle \in E} c_{\langle i,j\rangle} \cdot x'_{\langle i,j\rangle} & \\
\textbf{Subject to:} & 0 \leq x_{\langle i,j\rangle} + x_{\langle j,i\rangle} \leq s(\langle i,j\rangle) & \text{for every edge } \langle i,j\rangle \in E \\
 & x_{\langle i,j\rangle}, x_{\langle j,i\rangle} \in \mathbb{N}_0 & \text{for every edge } \langle i,j\rangle \in E \\
 & \displaystyle\sum_{\substack{j \in V:\\ \exists(\langle i,j\rangle \in E)}} \left(x_{\langle i,j\rangle}\right) - \sum_{\substack{j \in V:\\ \exists(\langle j,i\rangle \in E)}} \left(x_{\langle j,i\rangle}\right) = & \begin{cases} 1 & \text{if } i \in S \\ -1 & \text{if } i \in T \\ 0 & \text{otherwise} \end{cases} \\
 & & \text{for each vertex } i \in V \\
 & x'_{\langle i,j\rangle} \cdot s(\langle i,j\rangle) \geq x_{\langle i,j\rangle} + x_{\langle j,i\rangle} & \text{for every edge } \langle i,j\rangle \in E \\
 & x'_{\langle i,j\rangle} \in \{0,1\} & \text{for every edge } \langle i,j\rangle \in E
\end{array}$$

Variables: For each edge $\langle i,j\rangle \in E$, we define a two variables $x_{\langle i,j\rangle}, x_{\langle j,i\rangle}$ denoting the total used capacity in either direction along $\langle i,j\rangle$, as well as a decision-variable $x'_{\langle i,j\rangle}$ denoting whether that edge $\langle i,j\rangle$ is used (has a used capacity $x_{\langle i,j\rangle}, x_{\langle j,i\rangle}$ greater than zero).

Path-Based Approach to Multiple Shortest Path

The path based approach works by assigning a decision-variable for each possible path, and a decision variable for each possible pair of path and edge. We decide on a path by path basis whether they are included in the solution, and the used capacity along an edge is determined, not by the flow flowing through it, but by the sum of the paths that choose to use that edge.

We first consider a simplified version of the problem, where we already know a pre-existing mapping between sources and sinks.

Given: We consider an undirected graph $G = (V, E)$, with two sets of vertices $S = \{s_1, ..., s_k\}, T = \{t_1, ...t_k\}$, such that $S, T \subseteq V$ and $S \cap T = \emptyset$. For each edge $\langle i, j\rangle \in E$, we know a given weight $c_{\langle i,j\rangle}$.

Goal: Find a minimum-cost set of edges on G, such that there exists a path between each pair of vertices s_l, t_l, for any $1 \leq l \leq k$.

Minimize: $\displaystyle\sum_{\langle i,j\rangle \in E} c_{\langle i,j\rangle} \cdot x'_{\langle i,j\rangle}$

Subject to: $x_{\langle i,j\rangle,l}, x_{\langle j,i\rangle,l} \in \{0,1\}$ for each edge $\langle i, j\rangle \in E$, and each variable $1 \leq l \leq k$.

$$\sum_{\substack{j \in V: \\ \exists(\langle i,j\rangle \in E)}} \left(x_{\langle i,j\rangle,l}\right) - \sum_{\substack{j \in V: \\ \exists(\langle j,i\rangle \in E)}} \left(x_{\langle j,i\rangle,l}\right) = \begin{cases} 1 & \text{if } i = s_l \\ -1 & \text{if } i = t_l \\ 0 & \text{otherwise} \end{cases}$$

for each vertex $i \in V$, and each variable $1 \leq l \leq k$.

$x'_{\langle i,j\rangle} \cdot 2k \geq \displaystyle\sum_{1 \leq l \leq k} (x_{\langle i,j\rangle,l} + x_{\langle j,i\rangle,l})$ for each edge $\langle i, j\rangle \in E$

$x'_{\langle i,j\rangle} \in \{0,1\}$ for each edge $\langle i, j\rangle \in E$

Variables: For each edge $\langle i, j\rangle \in E$ and each variable $1 \leq l \leq k$, we define a decision-variable $x_{\langle i,j\rangle,l}, x_{\langle j,i\rangle,l}$. These decision variable indicate whether the edge $\langle i, j\rangle$ is used in either direction to from a path from s_l to t_l.
For each edge $\langle i, j\rangle \in E$ we define a decision-variable $x'_{\langle i,j\rangle}$, which indicates whether the edge $\langle i, j\rangle$ is used in any direction, for any path s_l to t_l.

If we want to extend this LP to allow for any pair of source and sink, we need to introduce decision-variables for each potential path, that determine whether these paths are used. The assigning of edges to paths can then only happen if paths are actually being used.

Given: We consider an undirected graph $G = (V, E)$, with two sets of vertices $S, T \subseteq V$, such that $|S| = |T|$ and $S \cap T = \emptyset$. For each edge $\langle i, j\rangle \in E$, we know a given weight $c_{\langle i,j\rangle}$.

Goal: Find a minimum-cost set of edges on G, such that there exists some ordering $S = \{s_1, ..., s_k\}, T = \{t_1, ..., t_k\}$ for which there exists a path between each pair of vertices s_l, t_l, for any $1 \leq l \leq k$.

Minimize: $\sum_{\langle i,j\rangle \in E} c_{\langle i,j\rangle} \cdot x'_{\langle i,j\rangle}$

Subject to:

$\alpha_{s,t} \in \{0,1\}$ for each source $s \in S$, and each sink $t \in T$.

$\sum_{t \in T}(\alpha_{s,t}) = 1$ for each source $s \in S$.

$\sum_{s \in S}(\alpha_{s,t}) = 1$ for each sink $t \in T$.

$x_{\langle i,j\rangle,s,t}, x_{\langle j,i\rangle,s,t} \in \{0,1\}$ for each edge $\langle i,j\rangle \in E$, each source $s \in S$, and each sink $t \in T$.

$x_{\langle i,j\rangle,s,t}, x_{\langle j,i\rangle,s,t} \leq \alpha_{s,t}$ for each edge $\langle i,j\rangle \in E$, each source $s \in S$, and each sink $t \in T$.

$$\sum_{\substack{j \in V: \\ \exists(\langle i,j\rangle \in E)}} \left(x_{\langle i,j\rangle,s,t}\right) - \sum_{\substack{j \in V: \\ \exists(\langle j,i\rangle \in E)}} \left(x_{\langle j,i\rangle,s,t}\right) = \begin{cases} 1 & \text{if } i = s \\ -1 & \text{if } i = t \\ 0 & \text{otherwise} \end{cases}$$

for each vertex $i \in V$, each source $s \in S$, and each sink $t \in T$.

$$x'_{\langle i,j\rangle} \cdot \Big(2 \cdot (|S| \cdot |T|)\Big) \geq \sum_{s \in S, t \in T} \Big(x_{\langle i,j\rangle,s,t} + x_{\langle j,i\rangle,s,t}\Big)$$

for each edge $\langle i,j\rangle \in E$

$x'_{\langle i,j\rangle} \in \{0,1\}$ For each edge $\langle i,j\rangle \in E$

Variables: For each pair of vertices $s \in S, t \in T$, we define a decision variable $\alpha_{s,t}$, indicating whether a path from source s to sink t is considered in the mapping.
For each edge $\langle i,j\rangle \in E$ and each pair s,t, we define a decision-variable $x_{\langle i,j\rangle,s,t}, x_{\langle j,i\rangle,s,t}$. These decision variable indicate whether the edge $\langle i,j\rangle$ is used in either direction to from a path from s to t.

A.1.3 Capacitated Facility-Location Problem

We create an MILP for the Capacitated Facility-Location Problem, by further extending the MILP we have defined for the Multiple Shortest Path Problem.

Edge Capacity-Based Approach to Multiple Shortest Path

Given: We consider an undirected graph $G = (V, E)$. We are given a set of facilities $F \subseteq V$ and a set of consumers/users $U \subseteq V$, such that $F \cap U = \emptyset$ and $\sum_{f \in F}(|s(f)|) \geq \sum_{u \in U}(|d(u)|)$.

For each edge $\langle i,j\rangle \in E$, we know the initial cost of construction $c_{\text{constr.}}(\langle i,j\rangle)$, as well as the cost to increase the capacity by one $c_{\text{cable}}(\langle i,j\rangle)$. For each facility $f \in F$, we know the initial cost of construction $c_{\text{constr.}}(f)$. For each facility $f \in F$, we know a maximum supply/capacity $s(f)$. For each user $u \in U$, we know a given demand $d(u)$

Goal: Find a minimum-cost set of edges and facilities on G, such that the demand $d(u)$ of each user $u \in U$ has to be satisfied, without exceeding the supply $s(f)$ of any facility $f \in F$

Minimize:
$$\sum_{\langle i,j\rangle\in E}\left(\left(c_{\text{constr.}}(\langle i,j\rangle)\cdot x'_{\langle i,j\rangle}\right)+\left(c_{\text{cable}}(\langle i,j\rangle)\cdot\left(x_{\langle i,j\rangle}+x_{\langle j,i\rangle}\right)\right)\right)+\sum_{f\in F}\left(c_{\text{constr.}}(f)\cdot y'_f\right)$$

Subject to:

$$\begin{array}{ll}
0\le x_{\langle i,j\rangle}+x_{\langle j,i\rangle}\le s(\langle i,j\rangle) & \text{for every edge } \langle i,j\rangle\in E\\
x_{\langle i,j\rangle},x_{\langle j,i\rangle}\in\mathbb{N}_0 & \text{for every edge } \langle i,j\rangle\in E\\
0\le y_f\le s(f) & \text{for every facility } f\in F\\
y_f\in\mathbb{N}_0 & \text{for every facility } f\in F\\
\sum\limits_{\substack{j\in V:\\ \exists(\langle i,j\rangle\in E)}}\left(x_{\langle i,j\rangle}\right)-\sum\limits_{\substack{j\in V:\\ \exists(\langle j,i\rangle\in E)}}\left(x_{\langle j,i\rangle}\right)=\begin{cases}y_i & \text{if } i\in F\\ -d(i) & \text{if } i\in U\\ 0 & \text{otherwise}\end{cases} & \\
 & \text{for each vertex } i\in V\\
x'_{\langle i,j\rangle}\cdot s(\langle i,j\rangle)\ge x_{\langle i,j\rangle}+x_{\langle j,i\rangle} & \text{for every edge } \langle i,j\rangle\in E\\
x'_{\langle i,j\rangle}\in\{0,1\} & \text{for every edge } \langle i,j\rangle\in E\\
y'_f\cdot s(f)\ge y_f & \text{for every facility } f\in F\\
y'_f\in\{0,1\} & \text{for every facility } f\in F
\end{array}$$

Variables: For each edge $\langle i,j\rangle\in E$, we define a two variables $x_{\langle i,j\rangle},x_{\langle j,i\rangle}$ denoting the total used capacity in either direction along $\langle i,j\rangle$, and a variable x'_e denoting whether the edge e is used. For each facility $f\in F$, we define a variable y_f denoting the total used capacity of f, and a variable y'_f denoting whether the facility f is used.

Path-Based Approach to Multiple Shortest Path

Given: We consider an undirected graph $G=(V,E)$. We are given a set of facilities $F\subseteq V$ and a set of consumers/users $U\subseteq V$, such that $F\cap U=\emptyset$ and $\sum_{f\in F}(|s(f)|)\ge\sum_{u\in U}(|d(u)|)$.

For each edge $\langle i,j\rangle\in E$, we know the initial cost of construction $c_{\text{constr.}}(\langle i,j\rangle)$, as well as the cost to increase the capacity by one $c_{\text{cable}}(\langle i,j\rangle)$. For each facility $f\in F$, we know the initial cost of construction $c_{\text{constr.}}(f)$. For each facility $f\in F$, we know a maximum supply/capacity $s(f)$. For each user $u\in U$, we know a given demand $d(u)$

Goal: Find a minimum-cost set of edges and facilities on G, such that the demand $d(u)$ of each user $u\in U$ has to be satisfied, without exceeding the supply $s(f)$ of any facility $f\in F$

Minimize:
$$\sum_{\langle i,j\rangle\in E}\left(c_{\text{constr.}}(\langle i,j\rangle)\cdot x'_{\langle i,j\rangle}\right)+\sum_{f\in F}\left(c_{\text{constr.}}(f)\cdot y_f\right)+\sum_{\substack{f\in F,\\ u\in U}}\left(\alpha_{f,u}\cdot\sum_{\langle i,j\rangle\in E}\left(c_{\text{cable}}(\langle i,j\rangle)\cdot\left(x_{\langle i,j\rangle,f,u}+x_{\langle j,i\rangle,f,u}\right)\right)\right)$$

Subject to:

$$\begin{array}{ll}
0\le\alpha_{f,u}\le d(u) & \text{for each facility } f\in F\text{, and each user } u\in U.\\
\alpha_{f,u}\in\mathbb{N}_0 & \text{for each facility } f\in F\text{, and each user } u\in U.\\
\sum\limits_{u\in U}(\alpha_{f,u})\le s(f) & \text{for each facility } f\in F.\\
\sum\limits_{f\in F}(\alpha_{f,u})=d(u) & \text{for each user } u\in U.\\
\alpha'_{f,u}\cdot d(u)\ge\alpha_{f,u} & \text{for each facility } f\in F\text{, and each user } u\in U.\\
\alpha'_{f,u}\in\{0,1\} & \text{for each facility } f\in F\text{, and each user } u\in U.
\end{array}$$

$$x_{\langle i,j\rangle,f,u}, x_{\langle j,i\rangle,f,u} \leq \alpha'_{f,u}$$ for each edge $\langle i,j\rangle \in E$, each facility $f \in F$, and each user $u \in U$.

$$x_{\langle i,j\rangle,f,u}, x_{\langle j,i\rangle,f,u} \in \{0,1\}$$ for each edge $\langle i,j\rangle \in E$, each facility $f \in F$, and each user $u \in U$.

$$\sum_{\substack{j\in V:\\ \exists(\langle i,j\rangle\in E)}} \left(x_{\langle i,j\rangle,f,u}\right) - \sum_{\substack{j\in V:\\ \exists(\langle j,i\rangle\in E)}} \left(x_{\langle j,i\rangle,f,u}\right) = \begin{cases} \alpha'_{f_\mathrm{d},u} & \text{if } i = f \\ -\alpha'_{f_\mathrm{d},u} & \text{if } i = u \\ 0 & \text{otherwise} \end{cases}$$

for each vertex $i \in V$, each facility $f \in F$, and each user $u \in U$.

$$x'_{\langle i,j\rangle} \cdot \Big(2 \cdot (|F| \cdot |U|)\Big) \geq \sum_{f\in F, u\in U} \Big(x_{\langle i,j\rangle,f,u} + x_{\langle j,i\rangle,f,u}\Big)$$ for each edge $\langle i,j\rangle \in E$

$$x'_{\langle i,j\rangle} \in \{0,1\}$$ For each edge $\langle i,j\rangle \in E$

$$y_f \cdot s(f) \geq \sum_{u\in U} (\alpha_{f,u})$$ for each facility $f \in F$.

$$y_f \in \{0,1\}$$ for each facility $f \in F$.

Variables: For each pair of vertices $f \in F, u \in U$, we define a variable $\alpha_{f,u}$ indicating the total capacity supplied from facility f to user u, and a decision variable $\alpha'_{f,u}$ indicating whether facility f supplies any capacity to user u.
For each edge $\langle i,j\rangle \in E$ and each pair f, u, we define a decision-variable $x_{\langle i,j\rangle,f,u}, x_{\langle j,i\rangle,f,u}$ indicating whether the edge $\langle i,j\rangle$ is used in either direction to from a path from f to u.
For each facility $f \in F$ we define a decision-variable y_f, indicating whether facility f gets built.

A.1.4 Capacitated Facility-Location Problem with Distribution-Facilities

The version of the Capacitated Facility-Location Problem that we aim to solve in this paper is slightly more complex than the vanilla Capacitated Facility-Location Problem. We introduce "distribution facilities", which serve as a mandatory step between the actual supply-facility and the consumer.

Edge Capacity-Based Approach to Multiple Shortest Path

Given: We consider an undirected graph $G = (V, E)$. We are given a set of supply-facilities $F_\mathrm{s} \subseteq V$, a set of distribution-facilities $F_\mathrm{d} \subseteq V$ and a set of consumers/users $U \subseteq V$, such that $(F_\mathrm{s} \cup F_\mathrm{d}) \cap U = \emptyset$ and $F_\mathrm{s} \cap F_\mathrm{d} = \emptyset$.
Furthermore it also hold that $\sum_{f_\mathrm{s}\in F_\mathrm{s}} (|s(f_\mathrm{s})|) \geq \sum_{f_\mathrm{d}\in F_\mathrm{d}} (|s(f_\mathrm{d})|) \geq \sum_{u\in U} (|d(u)|)$.

For each edge $\langle i,j\rangle \in E$, we know the initial cost of construction $c_{\mathrm{constr.}}(\langle i,j\rangle)$, as well as the cost to increase the capacity by one $c_{\mathrm{cable}}(\langle i,j\rangle)$. For each facility $f \in F_\mathrm{s}, F_\mathrm{d}$, we know the initial cost of construction $c_{\mathrm{constr.}}(f)$. For each facility $f \in F_\mathrm{s}, F_\mathrm{d}$, we know a maximum supply/capacity $s(f)$. For each user $u \in U$, we know a given demand $d(u)$

Goal: Find a minimum-cost set of edges and facilities on G, such that the demand $d(u)$ of each user $u \in U$ has to be satisfied, without exceeding the supply $s(f)$ of any facility $f \in F_\mathrm{s}, F_\mathrm{d}$.

$$
\textbf{Minimize:}\quad
\begin{aligned}
&\sum_{\langle i,j\rangle\in E}\Bigg(\Big(c_{\text{constr.}}(\langle i,j\rangle)\cdot x'_{\langle i,j\rangle}\Big)\\
&\qquad+\Big(c_{\text{cable},F_{\text{d}}\text{-to-}U}(\langle i,j\rangle)\cdot\big(x_{\langle i,j\rangle,F_{\text{d}}\text{-to-}U}+x_{\langle j,i\rangle,F_{\text{d}}\text{-to-}U}\big)\Big)\\
&\qquad+\Big(c_{\text{cable},F_{\text{s}}\text{-to-}F_{\text{d}}}(\langle i,j\rangle)\cdot\big(x_{\langle i,j\rangle,F_{\text{s}}\text{-to-}F_{\text{d}}}+x_{\langle j,i\rangle,F_{\text{s}}\text{-to-}F_{\text{d}}}\big)\Big)\Bigg)\\
&\qquad+\sum_{f_{\text{d}}\in F_{\text{d}}}\Big(c_{\text{constr.}}(f_{\text{d}})\cdot y'_{f_{\text{d}}}\Big)+\sum_{d_{\text{d}}\in F_{\text{s}}}\Big(c_{\text{constr.}}(f_{\text{s}})\cdot z'_{f_{\text{s}}}\Big)
\end{aligned}
$$

$$
\textbf{Subject to:}\quad
\begin{array}{ll}
\multicolumn{2}{l}{0\le\big(x_{\langle i,j\rangle,F_{\text{d}}\text{-to-}U}+x_{\langle j,i\rangle,F_{\text{d}}\text{-to-}U}\big)+\big(x_{\langle i,j\rangle,F_{\text{s}}\text{-to-}F_{\text{d}}}+x_{\langle j,i\rangle,F_{\text{s}}\text{-to-}F_{\text{d}}}\big)\le s(\langle i,j\rangle)}\\
& \text{for every edge } \langle i,j\rangle\in E\\
\multicolumn{2}{l}{x_{\langle i,j\rangle,F_{\text{d}}\text{-to-}U},x_{\langle j,i\rangle,F_{\text{d}}\text{-to-}U},x_{\langle i,j\rangle,F_{\text{s}}\text{-to-}F_{\text{d}}},x_{\langle j,i\rangle,F_{\text{s}}\text{-to-}F_{\text{d}}}\in\mathbb{N}_0}\\
& \text{for every edge } \langle i,j\rangle\in E\\
0\le y_{f_{\text{d}}}\le s(f_{\text{d}}) & \text{for every distribution-facility } f\in F_{\text{d}}\\
y_{f_{\text{d}}}\in\mathbb{N}_0 & \text{for every distribution-facility } f\in F_{\text{d}}\\
0\le y_{f_{\text{s}}}\le s(f_{\text{s}}) & \text{for every supply-facility } f\in F_{\text{s}}\\
y_{f_{\text{s}}}\in\mathbb{N}_0 & \text{for every supply-facility } f\in F_{\text{s}}\\
\multicolumn{2}{l}{\displaystyle\sum_{\substack{j\in V:\\ \exists(\langle i,j\rangle\in E)}}\big(x_{\langle i,j\rangle,F_{\text{d}}\text{-to-}U}\big)-\sum_{\substack{j\in V:\\ \exists(\langle j,i\rangle\in E)}}\big(x_{\langle j,i\rangle,F_{\text{d}}\text{-to-}U}\big)=\begin{cases}y_{f_{\text{d}}} & \text{if } i\in F_{\text{d}}\\ -d(i) & \text{if } i\in U\\ 0 & \text{otherwise}\end{cases}}\\
& \text{for each vertex } i\in V\\
\multicolumn{2}{l}{\displaystyle\sum_{\substack{j\in V:\\ \exists(\langle i,j\rangle\in E)}}\big(x_{\langle i,j\rangle,F_{\text{s}}\text{-to-}F_{\text{d}}}\big)-\sum_{\substack{j\in V:\\ \exists(\langle j,i\rangle\in E)}}\big(x_{\langle j,i\rangle,F_{\text{s}}\text{-to-}F_{\text{d}}}\big)=\begin{cases}y_{f_{\text{s}}} & \text{if } i\in F_{\text{s}}\\ -y_{f_{\text{d}}} & \text{if } i\in F_{\text{d}}\\ 0 & \text{otherwise}\end{cases}}\\
& \text{for each vertex } i\in V\\
\multicolumn{2}{l}{x'_{\langle i,j\rangle}\cdot s(\langle i,j\rangle)\ge\big(x_{\langle i,j\rangle,F_{\text{d}}\text{-to-}F_U}+x_{\langle j,i\rangle,F_{\text{d}}\text{-to-}F_U}\big)+\big(x_{\langle i,j\rangle,F_{\text{s}}\text{-to-}F_{\text{d}}}+x_{\langle j,i\rangle,F_{\text{s}}\text{-to-}F_{\text{d}}}\big)}\\
& \text{for every edge } \langle i,j\rangle\in E\\
x'_{\langle i,j\rangle}\in\{0,1\} & \text{for every edge } \langle i,j\rangle\in E\\
y'_{f_{\text{d}}}\cdot s(f_{\text{d}})\ge y_{f_{\text{d}}} & \text{for every distribution-facility } f_{\text{d}}\in F_{\text{d}}\\
y'_{f_{\text{d}}}\in\{0,1\} & \text{for every distribution-facility } f_{\text{d}}\in F_{\text{d}}\\
y'_{f_{\text{s}}}\cdot s(f_{\text{s}})\ge y_{f_{\text{s}}} & \text{for every supply-facility } f_{\text{s}}\in F_{\text{s}}\\
y'_{f_{\text{s}}}\in\{0,1\} & \text{for every supply-facility } f_{\text{s}}\in F_{\text{s}}
\end{array}
$$

Variables: For each edge $\langle i,j\rangle\in E$, we define two pairs of variables $x_{\langle i,j\rangle,F_{\text{s}}\text{-to-}F_{\text{d}}},x_{\langle j,i\rangle,F_{\text{s}}\text{-to-}F_{\text{d}}}$, and $x_{\langle i,j\rangle,F_{\text{d}}\text{-to-}U},x_{\langle j,i\rangle,F_{\text{d}}\text{-to-}U}$ denoting the total used capacity in either direction along $\langle i,j\rangle$ for cables between supply-facilities and distribution facilities, and between distribution-facilities and users respectively. Additionally, for each edge we define a variable $x'_{\langle i,j\rangle}$ denoting whether the edge e is used for any capacity in any direction.
For each facility $f\in F_{\text{s}},F_{\text{d}}$, we define a variable y_f denoting the total used capacity of that facility f, and a variable y'_f denoting whether the facility f is used for any capacity.

Path-Based Approach to Multiple Shortest Path

Given: We consider an undirected graph $G=(V,E)$. We are given a set of supply-facilities $F_{\text{s}}\subseteq V$, a set of distribution-facilities $F_{\text{d}}\subseteq V$ and a set of consumers/users $U\subseteq V$, such that $(F_{\text{s}}\cup F_{\text{d}})\cap U=\emptyset$ and $F_{\text{s}}\cap F_{\text{d}}=\emptyset$.
Furthermore it also hold that $\sum_{f_{\text{s}}\in F_{\text{s}}}(|s(f_{\text{s}})|)\ge\sum_{f_{\text{d}}\in F_{\text{d}}}(|s(f_{\text{d}})|)\ge\sum_{u\in U}(|d(u)|)$.

For each edge $\langle i,j\rangle\in E$, we know the initial cost of construction $c_{\text{constr.}}(\langle i,j\rangle)$, as well as the cost to increase the capacity by one $c_{\text{cable}}(\langle i,j\rangle)$. For each facility $f\in F_{\text{s}},F_{\text{d}}$, we know the initial cost of construction $c_{\text{constr.}}(f)$. For each facility $f\in F_{\text{s}},F_{\text{d}}$, we know a maximum supply/capacity $s(f)$. For each user $u\in U$, we know a given demand $d(u)$

Goal: Find a minimum-cost set of edges and facilities on G, such that the demand $d(u)$ of each user $u \in U$ has to be satisfied, without exceeding the supply $s(f)$ of any facility $f \in F_{\mathrm{s}}, F_{\mathrm{d}}$.

Minimize:

$$\sum_{\langle i,j\rangle \in E} \left(c_{\text{constr.}}(\langle i,j\rangle) \cdot x'_{\langle i,j\rangle}\right) + \sum_{f_{\mathrm{s}} \in F_{\mathrm{s}}} \left(c_{\text{constr.}}(f_{\mathrm{s}}) \cdot y_{f_{\mathrm{s}}}\right) + \sum_{f_{\mathrm{d}} \in F_{\mathrm{d}}} \left(c_{\text{constr.}}(f_{\mathrm{d}}) \cdot y_{f_{\mathrm{d}}}\right)$$
$$+ \sum_{\substack{f_{\mathrm{s}} \in F_{\mathrm{s}}, f_{\mathrm{d}} \in F_{\mathrm{d}} \\ \langle i,j\rangle \in E}} \left(c_{\text{cable},F_{\mathrm{s}}\text{-to-}F_{\mathrm{d}}}(\langle i,j\rangle) \cdot \left(x_{\langle i,j\rangle,f_{\mathrm{s}},f_{\mathrm{d}}} + x_{\langle j,i\rangle,f_{\mathrm{s}},f_{\mathrm{d}}}\right)\right)$$
$$+ \sum_{\substack{f_{\mathrm{d}} \in F_{\mathrm{d}}, u \in U \\ \langle i,j\rangle \in E}} \left(c_{\text{cable},F_{\mathrm{d}}\text{-to-}U}(\langle i,j\rangle) \cdot \left(x_{\langle i,j\rangle,f_{\mathrm{d}},u} + x_{\langle j,i\rangle,f_{\mathrm{d}},u}\right)\right)$$

Subject to:

$0 \leq \alpha_{f_{\mathrm{d}},u} \leq d(u)$ — For each distr.-facility $f_{\mathrm{d}} \in F_{\mathrm{d}}$, and each user $u \in U$.

$0 \leq \alpha_{f_{\mathrm{s}},f_{\mathrm{d}}} \leq s(f_{\mathrm{d}})$ — For each supply-facility $f_{\mathrm{s}} \in F_{\mathrm{s}}$, and each distr.-facility $f_{\mathrm{d}} \in F_{\mathrm{d}}$.

$\alpha_{f_{\mathrm{d}},u}, \alpha_{f_{\mathrm{s}},f_{\mathrm{d}}} \in \mathbb{N}_0$ — For each facility $f_{\mathrm{s}} \in F_{\mathrm{s}}$, $f_{\mathrm{d}} \in F_{\mathrm{d}}$, and each user $u \in U$.

$\sum_{u \in U} (\alpha_{f_{\mathrm{d}},u}) \leq s(f_{\mathrm{d}})$ — For each distr.-facility $f_{\mathrm{d}} \in F_{\mathrm{d}}$.

$\sum_{f_{\mathrm{d}} \in F_{\mathrm{d}}} (\alpha_{f_{\mathrm{d}},u}) = d(u)$ — For each user $u \in U$.

$\sum_{f_{\mathrm{d}} \in F_{\mathrm{d}}} (\alpha_{f_{\mathrm{s}},f_{\mathrm{d}}}) \leq s(f_{\mathrm{s}})$ — For each supply-facility $f_{\mathrm{s}} \in F_{\mathrm{s}}$.

$\sum_{f_{\mathrm{s}} \in F_{\mathrm{s}}} (\alpha_{f_{\mathrm{s}},f_{\mathrm{d}}}) = \sum_{u \in U} (\alpha_{f_{\mathrm{d}},u})$ — For each distr.-facility $f_{\mathrm{d}} \in F_{\mathrm{d}}$.

$0 \leq x_{\langle i,j\rangle,f_{\mathrm{d}},u}, x_{\langle j,i\rangle,f_{\mathrm{d}},u} \leq \alpha_{f_{\mathrm{d}},u}$ — For each edge $\langle i,j\rangle \in E$, each facility $f_{\mathrm{d}} \in F_{\mathrm{d}}$, each user $u \in U$.

$0 \leq x_{\langle i,j\rangle,f_{\mathrm{s}},f_{\mathrm{d}}}, x_{\langle j,i\rangle,f_{\mathrm{s}},f_{\mathrm{d}}} \leq \alpha_{f_{\mathrm{s}},f_{\mathrm{d}}}$ — For each edge $\langle i,j\rangle \in E$, each facility $f_{\mathrm{s}} \in F_{\mathrm{s}}$, $f_{\mathrm{d}} \in F_{\mathrm{d}}$.

$x_{\langle i,j\rangle,f_{\mathrm{d}},u}, x_{\langle j,i\rangle,f_{\mathrm{d}},u}, x_{\langle i,j\rangle,f_{\mathrm{s}},f_{\mathrm{d}}}, x_{\langle j,i\rangle,f_{\mathrm{s}},f_{\mathrm{d}}} \in \mathbb{N}_0$ — For each edge $\langle i,j\rangle \in E$, each facility $f_{\mathrm{s}} \in F_{\mathrm{s}}$, $f_{\mathrm{d}} \in F_{\mathrm{d}}$, and each user $u \in U$.

$$\sum_{\substack{j \in V: \\ \exists(\langle i,j\rangle \in E)}} \left(x_{\langle i,j\rangle,f_{\mathrm{d}},u}\right) - \sum_{\substack{j \in V: \\ \exists(\langle j,i\rangle \in E)}} \left(x_{\langle j,i\rangle,f_{\mathrm{d}},u}\right) = \begin{cases} \alpha_{f_{\mathrm{d}},u} & \text{if } i = f_{\mathrm{d}} \\ -\alpha_{f_{\mathrm{d}},u} & \text{if } i = u \\ 0 & \text{otherwise} \end{cases}$$

For each vertex $i \in V$, each facility $f_{\mathrm{d}} \in F_{\mathrm{d}}$, user $u \in U$.

$$\sum_{\substack{j \in V: \\ \exists(\langle i,j\rangle \in E)}} \left(x_{\langle i,j\rangle,f_{\mathrm{s}},f_{\mathrm{d}}}\right) - \sum_{\substack{j \in V: \\ \exists(\langle j,i\rangle \in E)}} \left(x_{\langle j,i\rangle,f_{\mathrm{s}},f_{\mathrm{d}}}\right) = \begin{cases} \alpha_{f_{\mathrm{s}},f_{\mathrm{d}}} & \text{if } i = f_{\mathrm{s}} \\ -\alpha_{f_{\mathrm{s}},f_{\mathrm{d}}} & \text{if } i = f_{\mathrm{d}} \\ 0 & \text{otherwise} \end{cases}$$

For each vertex $i \in V$, each facility $f_{\mathrm{s}} \in F_{\mathrm{s}}$, $f_{\mathrm{d}} \in F_{\mathrm{d}}$.

$$x'_{\langle i,j\rangle} \cdot \Big(2 \cdot (|F_{\mathrm{s}}| \cdot |F_{\mathrm{d}}|) + 2 \cdot (|F_{\mathrm{d}}| \cdot |U|)\Big) \cdot \sum_{u \in U} (d(u)) \geq$$
$$\sum_{\substack{f_{\mathrm{s}} \in F_{\mathrm{s}}, \\ f_{\mathrm{d}} \in F_{\mathrm{d}}}} \left(x_{\langle i,j\rangle,f_{\mathrm{s}},f_{\mathrm{d}}} + x_{\langle j,i\rangle,f_{\mathrm{s}},f_{\mathrm{d}}}\right) \sum_{\substack{f_{\mathrm{d}} \in F_{\mathrm{d}}, \\ u \in U}} \left(x_{\langle i,j\rangle,f_{\mathrm{d}},u} + x_{\langle j,i\rangle,f_{\mathrm{d}},u}\right)$$

for each edge $\langle i,j\rangle \in E$

$x'_{\langle i,j\rangle} \in \{0,1\}$ — For each edge $\langle i,j\rangle \in E$

$$y_{f_\mathrm{d}} \cdot s(f_\mathrm{d}) \geq \sum_{u \in U} (\alpha_{f_\mathrm{d},u}) \qquad \text{For each distr.-facility } f_\mathrm{d} \in F_\mathrm{d}.$$

$$y_{f_\mathrm{s}} \cdot s(f_\mathrm{s}) \geq \sum_{f_\mathrm{d} \in F_\mathrm{d}} (\alpha_{f_\mathrm{s},f_\mathrm{d}}) \qquad \text{For each supply-facility } f_\mathrm{s} \in F_\mathrm{s}.$$

$$y_{f_\mathrm{s}}, y_{f_\mathrm{d}} \in \{0,1\} \qquad \text{For each facility } f_\mathrm{s} \in F_\mathrm{s},\ f_\mathrm{d} \in F_\mathrm{d}.$$

Variables: For each pair of vertices $f_\mathrm{d} \in F_\mathrm{d}, u \in U$, we define a variable $\alpha_{f_\mathrm{d},u}$ indicating the total capacity supplied from distribution-facility f_d to user u.
For each pair of vertices $f_\mathrm{s} \in F_\mathrm{s}, f_\mathrm{d} \in F_\mathrm{d}$, we define a variable $\alpha_{f_\mathrm{s},f_\mathrm{d}}$ indicating the total capacity supplied from supply-facility f_s to distribution-facility f_d.
For each edge $\langle i,j \rangle \in E$ and each pair f_d, u, we define a variable $x_{\langle i,j \rangle, f_\mathrm{d}, u}, x_{\langle j,i \rangle, f_\mathrm{d}, u}$ indicating the total capacity supplied from distribution-facility f_d to user u, along edge $\langle i,j \rangle$ and $\langle j,i \rangle$ respectively.
For each edge $\langle i,j \rangle \in E$ and each pair $f_\mathrm{s}, f_\mathrm{d}$, we define a variable $x_{\langle i,j \rangle, f_\mathrm{s}, f_\mathrm{d}}, x_{\langle j,i \rangle, f_\mathrm{s}, f_\mathrm{d}}$ indicating the total capacity supplied from supply-facility f_s to distribution-facility f_d, along edge $\langle i,j \rangle$ and $\langle j,i \rangle$ respectively.
For each edge $\langle i,j \rangle \in E$, we define a decision-variable $x'_{\langle i,j \rangle}$ indicating whether the edge $\langle i,j \rangle$ is used for any capacity in any direction.
For each facility $f_\mathrm{s} \in F_\mathrm{s}$ we define a decision-variable y_{f_s}, indicating whether facility f_s gets built.
For each facility $f_\mathrm{d} \in F_\mathrm{d}$ we define a decision-variable y_{f_d}, indicating whether facility f_d gets built.

A.1.5 Comparison Between Edge Capacity-Based and Path-Based MILP

Both the aforementioned Edge Capacity-Based and the Path-Based MILP approach to Capacitated Facility-Location Problem will correctly find the solution to the cost-minimization question. However, performance- and usability-wise both approaches do have some differences

Difference in Performance

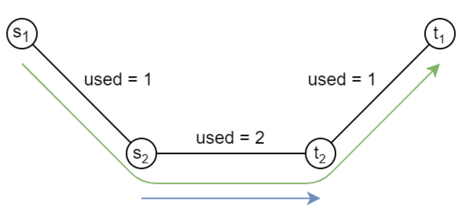
While it is difficult to determine the effective performance for solving any linear program, it is possible to estimate expected performance, based on the amount of decision-variables and the amount of unique constraint-statements that encompass the LP.

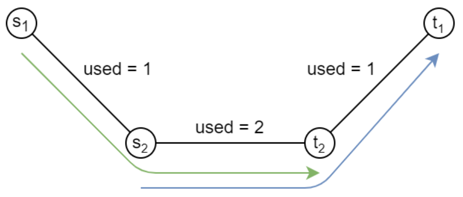
	Edge Capacity Based Approach	**Path Based Approach**
#unique variables	$\mathrm{O}\Big(\lvert F_\mathrm{s}\rvert + \lvert F_\mathrm{d}\rvert + \lvert E\rvert\Big)$	$\mathrm{O}\Big(\lvert E\rvert \cdot \big(\lvert F_\mathrm{d}\rvert \cdot (\lvert F_\mathrm{s}\rvert + \lvert U\rvert)\big) + (\lvert F_\mathrm{d}\rvert + \lvert Fs\rvert)\Big)$
#constraints	$\mathrm{O}\Big(\lvert V\rvert + \lvert E\rvert\Big)$	$\mathrm{O}\Big((\lvert V\rvert + \lvert E\rvert) \cdot \big(\lvert F_\mathrm{d}\rvert \cdot (\lvert F_\mathrm{s}\rvert + \lvert U\rvert)\big)\Big)$
#variables / constraint	$\mathrm{O}\Big(\lvert E\rvert\Big)$	$\mathrm{O}\Big(\lvert E\rvert \cdot \big(\lvert F_\mathrm{d}\rvert \cdot (\lvert F_\mathrm{s}\rvert + \lvert U\rvert)\big)\Big)$

In this table we can see that the path-based approach has a number of constraints and a number of variables both reaching a factor of $\mathrm{O}(n^3)$, whereas the edge capacity-based approach managed achieve a similar result with merely a factor $\mathrm{O}(n)$ constraints and variables.
This likely means that performance-wise, the edge capacity-based approach will be significantly faster than the path-based approach.

Difference in Ambiguity of Result

Even though the capacity-based approach might be faster to calculate, that does not mean that the given solution is ultimately equality unambiguous.





The edge capacity-based approach does not consider individual paths, and rather only considers the total used capacity along each individual edge. There exist specific results where the path of two different facility-to-user pairs, or facility-to-facility pairs, overlap. (See figure above). In some of these cases the lack of distinction between paths might create ambiguity in interpreting the result with regards to mapping facilities to users.
In the path based approach, this will not be the case, since it is always clearly defined which facility is mapped to which user, as well as the specific path that that specific mapping follows.

Difference in Expandability of Problem

There is also a slight difference in the expandability on the problem between the two approaches. Under the path based approach we are able to easy add additional requirements such as a maximum total length of a path, that are not as easily to define in the edge capacity-based approach.

Conclusion

Which of the two approaches is most useful is largely dependent on what qualities you value most.

- if you value performance over unambiguity and expandability, then the edge-capacity based approach might better suit your needs.
- if you value unambiguity and expandability over performance, then the path based approach might better suit your needs.