A Metaheuristic Approach to Solve the FLaMECaST Problem

Bachelor's Thesis of

Simon Kienle

At the Department of Informatics
Institute of Theoretical Informatics (ITI)

Reviewer:	T.T.-Prof. Dr. Thomas Bläsius
Second reviewer:	PD Dr. Torsten Ueckerdt
Advisors:	Wendy Yi
	Max Göttlicher

22 December 2024 – 22 April 2025

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself. I have not used any other than the aids that I have mentioned. I have marked all parts of the thesis that I have included from referenced literature, either in their original wording or paraphrasing their contents. I have followed the by-laws to implement scientific integrity at KIT.

Karlsruhe, 22 April 2025

. . . . . .

(Simon Kienle)

Abstract

The *Flow-weighted Layered Minimal Euclidean Capacitated Steiner Tree Problem* (FLaMECaST) is an optimization problem which aims at finding cost-efficient layouts of hierarchical networks like solar farms. As FLaMECaST is an NP-hard optimization problem, it is necessary to create efficient heuristics to find good solutions for the problem in a feasible amount of time. So within this thesis, an implementation of the metaheuristic Simulated Annealing is proposed to solve the FLaMECaST-problem heuristically.

This work introduces a neighborhood for solutions of the FLaMECaST-problem. Using this neighborhood a method to select feasible candidate neighbors without having to compute the complete neighborhood of solutions is described and applied to the Simulated Annealing procedure. It is shown that the proposed algorithm reaches the best performance by creating initial solutions using a combination of min-cost-max-matching and k-Means. Further experiments find that Simulated Annealing achieves a good reduction of initial costs on FLaMECaST-instances with lower values for the flow-weighting parameter alpha, however as the alpha-values grow the improvement of initial solutions decreases. By calculating solutions for circular instances which can be solved in polynomial time, it is shown that Simulated Annealing can reach good performance as its solutions result in only slightly bigger relative costs than the optimal solutions of these instances.

Zusammenfassung

Das *Flussgewichtete, Geschichtete, Minimal Euklidische Kapazitierte Steiner Baum Problem* (FLaMECaST) ist ein Optimierungsproblem das versucht, eine kosteneffiziente Gestaltung von hierarchischen Netzwerken wie Solarparks zu finden. Da FLaMECaST ein NP-schweres Optimierungsproblem ist, ist es notwendig, effiziente Heuristiken zu entwerfen um gute Lösungen für das Problem mit einer akzeptablen Laufzeit zu finden. Deshalb wird in dieser Arbeit eine Implementierung der Metaheuristik Simulated Annealing für das FLaMECaST-Problem vorgestellt.

Diese Ausarbeitung führt eine Nachbarschaft von Lösungen des FLaMECaST-Problems ein. Auf Grundlage dieser Nachbarschaft wird eine Methode beschrieben, um valide Nachbarn als Kandidaten zu wählen ohne die vollständige Nachbarschaft von Lösungen bestimmen zu müssen. Diese Methode wird im Simulated Annealing Algorithmus angewandt. Es wird gezeigt dass der vorgestellte Algorithmus die beste Leistung zeigt, wenn man zum Erstellen von initialen Lösungen eine Kombination aus kostenminimalen maximalen Zuordnungen und k-Means verwendet. Weitere Experimente stellen fest, dass Simulated Annealing eine gute Reduziereung der anfänglichen Kosten bei FLaMECaST-Instanzen mit kleinen Werten für den Flussgewichtungsparameter alpha erreicht, allerdings nimmt diese Verbesserung der anfänglichen Lösungen mit steigenden alpha-Werten ab. Durch das Berechnen von Lösungen von zirkulären Instanzen, welche in polynomieller Zeit gelöst werden können, wird gezeigt, dass Simulated Annealing gute Ergebnisse liefern kann da die gefundenen Lösungen nur leicht größere Kosten als die optimalen Lösungen dieser Instanzen haben.

Contents

1 Introduction

The global climate change is perceived the biggest threat to the countries of the world [PFG22]. One of the main causes of the climate change is the continuous emission of greenhouse gases like CO_2 or NH_4 into the atmosphere, enhancing the greenhouse effect and leading to global warming [Fil+24]. The Intergovernmental Panel on Climate Change IPCC reports a historic high of anthropogenic greenhouse gas emissions and identifies the energy sector, especially the electricity and heat generation, as the main drivers of greenhouse gas emissions in 2019 [23]. As a consequence a transformation of the current energy system is crucial. This can be done by using renewable energy sources.

Renewable energy sources in power generation are already in action. The International Renewable Energy Agency IRENA reported an increase of global renewable power capacity to 4448 GW at the end of 2024 and declared *solar power* as the largest contributor with 1865 GW [Age25]. Furthermore over three-quarters of the renewable power capacity expansion in 2024 was due to solar power with an increase of 452 GW. Meanwhile utility-scale solar photovoltaic like *solar farms* experienced a massive decline in global weighted average cost from USD 0.460/kWh in 2010 to USD 0.044/kWh in 2023 [Age24].

In the following, the structure of hierarchical energy networks and especially solar farms is closer examined. In the solar farm model described in [GSW22], a solar farm consists of multiple photovoltaic cells generating an electric current. This electric flow then must traverse a fixed number of processing steps before it can enter the power grid. The mentioned processing steps include flow collectors, inverting of direct current (DC) to alternating current (AC) and finally transforming the low-voltage current to high-voltage so it can enter the power grid. As the generated electric current of each photovoltaic cell must pass through exactly six processing steps, a solar farm can be assumed hierarchical with six layers.

In order to improve the benefits of solar power and solar farms as a renewable energy source, the solar farm must be optimally planned to ensure its operability while having minimal costs. One cost factor of solar farms are cable costs which can be optimized with the *Solar Farm Cable Layout Problem* or SoFaCLaP [Age24] [GSW22]. SoFaCLaP aims on finding an optimal cable layout for a solar farm with given positions of all components. SoFaCLaP further supports several different cable types which can connect the components. But it is shown that it is a strongly NP-complete problem to decide whether a SoFaCLaP-instance has a feasible solution and in addition its main downside is that the positions of all components are fixed [GSW22].

To encounter this problem, Henrik Csöre and Elly Schmidt introduce the *Flow-weighted Layered Minimal Euclidean Capacitated Steiner Tree Problem* (FLaMECaST) [CS24]. FLaMECaST is directly inspired by SoFaCLaP and calculates optimal layouts of hierarchical energy networks like solar farms. It provides more flexibility by allowing an arbitrary number and arbitrary positions as well as capacities for the components of the solar farm. Further as more flow through the edges often result in higher cable costs, these flows through the edges are also considered in the cost function which is minimized.

The FLaMECaST-problem described is proven to be an NP-hard optimization problem [CS24]. This motivates the creation of efficient heuristics to find a heuristically good solution of FLaMECaST in a reasonable amount of time. To our knowledge, there is no such algorithm to find a good solution of the FLaMECaST-problem yet. So within this thesis, the metaheuristic Simulated Annealing is used to calculate a heuristic solution of the FLaMECaST-problem.

This thesis is structured as follows. In the next chapter, basic concepts of graph theory and the corresponding notation used in this thesis is described. The third chapter gives a short overview over similar problems to the FLaMECaST-problem that has already been worked on in the past. Chapter 4 describes the exact definition of the FLaMECaST-problem that is then used in Chapter 5 to explain a metaheuristic algorithm to solve the FLaMECaST-problem. In order to evaluate and investigate properties of the created metaheuristic algorithm, the Chapter 6 presents several experiments to test the introduced algorithm in Chapter 5.

2 Preliminaries

This chapter provides basic concepts required to read this thesis. It includes the fundamentals of graph theory, graph embedding as well as flow networks and explains the corresponding notation used in this thesis.

2.1 Graph Theory

A *directed graph* is a tuple $G = (V, E)$ with a set of vertices V and a set of directed edges $E \subseteq V \times V$ that connect the vertices. A directed edge $e = (u, v) \in E$ describes a connection from the vertex u to the vertex v. A directed edge $e \in E$ is called a *multi edge* if it is in E more than once. Given a vertex $v \in V$, the *surrounding edges* of v in G is the set of all edges which start in v or end in v.

A *path* P from $v_0 \in V$ to $v_n \in V$ in a graph G is a tuple $P = (v_0, v_1, \ldots, v_n)$ with the condition $(v_i, v_{i+1}) \in E$ for all $i \in \{0, \ldots, n-1\}$. A *cycle* is a path $C = (v_0, v_1, \ldots, v_n)$ where $v_0 = v_n$. A graph G is called *acyclic* if it does not contain any cycle. A graph G is *simple* if it does not contain any multi edges and does not contain a loop $e = (v, v)$ with $v \in V$.

In the following, a graph G refers to a simple, directed graph.

A *complete, bipartite graph* $G = (V, E)$ is a graph whose set of vertices V can be partitioned into two sets V_1, V_2 so that each edge in E describes a connection from a vertex in V_1 to a vertex in V_2 and every possible edge that could connect one vertex in V_1 to one vertex in V_2 is in E.

An *in-tree* $A = (V, E)$ is a graph that contains a root $r \in V$ such that for every vertex $v \in V \setminus \{r\}$ there is exactly one path P_v from v to the root [KVKV11]. Given two vertices $u, v \in V$, then the *common ancestor* $w \in V$ is defined as the first vertex $w \in V$ which lies on both paths P_u and P_v or is defined as the root if one of the vertices u, v is the root. Further given the unique path P_v of a vertex $v \in V \setminus \{r\}$ to the root, the successor of v on P_v is called the *parent* of v, $parent(v)$. The *children* of v, $children(v)$, is the set of all vertices whose parent is v. The *leaves* of A are all vertices with no children.

A *forest* F is a disjoint union of one or multiple in-trees. A forest $F = (V, E)$ is further called *layered* with k layers if the vertices V can be partitioned into k sets called layers $V_1, V_2, \ldots, V_k$ such that every edge in E connects a vertex of one layer V_i with a vertex of the following layer V_{i+1} for some $i \in \{1, \ldots, k-1\}$. This implies that the parent of a vertex $v \in V_i$ lies in the following layer V_{i+1} and all children of v lie in the preceding layer V_{i-1}, such that $parent(v) \in V_{i+1}$ and $children(v) \subseteq V_{i-1}$. In the following, the set V_i is called the i-th layer of F.

A layered forest F with k layers is called *total layered* if every leaf of F is in the first or the last layer and the roots of the underlying in-trees constitute the last layer. In this case the vertices in V_1 are called *sources* and the vertices in V_k are called *sinks* of F. These definitions imply that a source $s \in V_1$ has no children and a sink $d \in V_k$ does not have a parent, but every other vertex of F does have at least one child and a parent. Further a sink is the only vertex in the forest F that can have none or multiple children.

2.2 Flow in Forests

Given a total layered forest $F = (V, E)$ with k layers, sources $S = \{s_1, \ldots, s_n\} = V_1$ and sinks $D = \{d_1, \ldots, d_m\} = V_k$. Further every source $s_i \in S$ produces a specific flow $f_i \in \mathbb{N}$. Within this thesis, every source outputs a flow $f_i = 1$.

These flow outputs induce a *flow* $f : V \to \mathbb{N}$ in the forest F with the following characteristics:

1. $f(s) = 1$ for all $s \in S$
2. $f(d) = 0$ for all $d \in \{v \mid v \in D, children(v) = \emptyset\}$
3. $f(v) = \sum_{c \in children(v)} f(c)$ for all other vertices $v \in V$

A flow in the forest F can also be represented through its edges with $f : E \to \mathbb{N}$. Given an edge $e = (u, v)$, the flow of e is defined as $f(e) = f(u)$ because a vertex in the forest has exactly one outgoing edge and the vertex therefore passes its complete flow through this edge.

2.3 Graph Embedding

Given a set of vertices V, the *embeddings* of the vertices in V are defined through a function $pos : V \to \mathbb{R}^2$. This embedding function pos assigns every vertex $v \in V$ an embedding in the two-dimensional Euclidean plane. The embedding of v is called *point*. Further given an edge $e = (u, v)$, the *embedding* of e is defined as the line between the point $pos(u)$ and the point $pos(v)$. The length of the edge e with respect to pos, $len_{pos}(e)$, is the Euclidean distance between the point $pos(u)$ and the point $pos(v)$.

Given an embedding function $pos : V \to \mathbb{R}^2$, the *embedding* of a graph $G = (V, E)$ based on pos is defined through the embeddings of the vertices in V and the embeddings of the edges in E. This allows to define the length of the graph G with respect to pos, $len_{pos}(G)$, as the sum of the lengths of all edges of the graph.

3 Related Work

Research on similar problems to FLaMECaST has already been made. Within this chapter, existing work on the FLaMECaST-problem and a selection of related optimization problems with corresponding solution methods are described. This serves to differentiate from previous research as well as giving the opportunity to delve into related problems.

3.1 Existing work on FLaMECaST

The FLaMECaST-problem is firstly introduced in the paper "The Flow-weighted Layered Minimal Euclidean Capacitated Steiner Tree Problem (FLaMECaST)" by Henrik Csöre and Elly Schmidt [CS24]. It provides a proof for its NP-hardness as well as classes of instances which can be solved in polynomial time, like line instances, circular instances, instances with only two layers or instances with the flow-weighting parameter α set to 1. Furthermore it analyzes the number of different topologies of a FLaMECaST-instance and geometric properties of optimal solutions. Finally construction heuristics and a simple edge swap heuristic are created to find heuristic solutions of the FLaMECaST-problem.

This initial paper on FLaMECaST serves as a basis for this thesis. The metaheuristic described in Section 5.3 works with the introduced problem definition of FLaMECaST and uses the proposed construction heuristics to find an initial solution.

3.2 Solar Farm Cable Layout Problem

As already mentioned in Chapter 1, the FLaMECaST-problem is directly inspired by the *Solar Farm Cable Layout Problem* or SoFaCLaP. The main goal of SoFaCLaP is calculating an optimal cable configuration for a solar farm with fixed component positions, transporting the electric current from power generating strings to transformers.

In SoFaCLaP, a fixed set P of candidate points for electric components in the Euclidean plane is given. This set P is partitioned into six disjoint subsets $P_1, \ldots, P_6$ where all points in a set P_i are candidate points for identical components. These points can be represented by a set of vertices V with a corresponding embedding function $pos : V \to \mathbb{R}^2$ mapping the vertices in V to their original points in P.

Further given is a fixed set of edges E which fully connects one set V_i with the following set V_{i+1}, so that $E = \bigcup_{i=1}^{5} V_i \times V_{i+1}$. Additionally a set of cable types C is given. These cables are used to connect two components represented by an edge in E. Every component and cable type owns a certain capacity and every cable type has a specific cost per length unit.

The problem of SoFaCLaP is to define a feasible function $k : E \rightarrow C$ which assigns a cable type for each edge in E. This assignment function should minimize the sum of the weighted lengths given as

$$cost = \sum_{e \in E} cost(k(e)) \cdot len_{pos}(e)$$

while holding all capacity restrictions on components and cables and transporting the flow from the strings in P_1 to the transformers in P_6 [Wah22].

The presented SoFaCLaP is known to be a NP-hard optimization problem, even the decision whether a SoFaCLaP-instance has feasible solutions is strongly NP-complete [GSW22]. This means that efficient heuristics must be found to calculate a good solution in a feasible amount of time. For that purpose, in the Bachelor Thesis "Variable Neighborhood Search for the Solar Farm Cable Layout Problem" by Leonie Wahl a metaheuristic approach to solve SoFaCLaP is introduced [Wah22].

Here, the main difference to the FLaMECaST-problem other than capacity restrictions or costs for cables is the fixed number and the fixed positions of the components in the Euclidean plane. So FLaMECaST allows more flexibility by allowing an arbitrary number and arbitrary positions of points for components in the Euclidean plane.

3.3 Minimum Multi-Source Multi-Sink Steiner Network Problem

The *Minimum Multi-Source Multi-Sink Steiner Network Problem* or MMMSNP is a variation of the Euclidean Steiner Tree Problem [BGTZ14]. Given two sets of distinct points A and B in the Euclidean plane, MMMSNP aims on finding a minimal (A, B)-network connecting each point $a \in A$ with each point $b \in B$. (A, B)-networks are explained in more detail below.

The points in A and B can be represented by two sets of vertices V_A and V_B with $|V_A| = |A|$ and $|V_B| = |B|$. An (A, B)-network is defined as a graph $G = (V, E)$ with $(V_A \cup V_B) \subseteq V$ and a path from each vertex $v_A \in V_A$ to each vertex $v_B \in V_B$. Given an embedding function pos, mapping every vertex in V into the Euclidean plane while mapping the vertices in V_A and V_B to the points in A and B, the length of the (A, B)-network G is defined as $|G| = len_{pos}(G)$. Thereby, the embedding function pos ensures minimal length $|G|$ among all possible embedding functions for G.

A minimal (A, B)-network is an (A, B)-network G with minimal length $|G|$ among all possible (A, B)-networks. So given two sets of points A and B, the Minimal Multi-Source Multi-Sink Steiner Network Problem is finding a minimal (A, B)-network.

The Minimal Multi-Source Multi-Sink Steiner Network Problem is proven to be an NP-hard optimization problem [WBR23]. Despite there is no heuristic algorithm for MMMSNP yet, an algorithm to find an exact solution for the MMMSNP already exists [WBR23].

In contrast to FLaMECaST, MMMSNP does not use a layered structure nor capacities for the vertices in an (A, B)-network. Additionally the length of an (A, B)-network does not consider flows through the edges and as opposed to FLaMECaST, every source vertex $v_A \in V_A$ is connected with every sink vertex $v_B \in V_B$ [CS24].

3.4 Single-Source Capacitated Multi-Facility Weber Problem

An additional related optimization-problem to FLaMECaST is the *Single-Source Capacitated Multi-Facility Weber Problem* or SSCMFW. Similar to the FLaMECaST-problem discussed within this thesis, SSCMFW tries to locate optimal points in the two-dimensional Euclidean plane. The SSCMFW is given a set of customers V_C with its corresponding embeddings in the Euclidean plane $pos_C : V_C \to \mathbb{R}^2$. These customers have a demand $w : V_C \to \mathbb{R}_+$ which is satisfied through exactly one of the facilities V_F with $|V_F| = m$ for a given $m \in \mathbb{N}$. Further a facility owns a certain capacity $q : V_F \to \mathbb{R}$ so that each facility has an upper bound on how much demand they can satisfy. The goal of SSCMFW is to find an embedding $pos_F : V_F \to \mathbb{R}^2$ of the facilities in the Euclidean plane and a mapping $z : V_C \to V_F$, assigning each customer to a facility without exceeding its capacity, which minimizes the Euclidean distances from the customers to the assigned facilities:

$$F(pos_F, z) = \sum_{v_c \in V_C} len_{pos_C, pos_F}(v_c, z(v_c))$$

The presented Single-Source Capacitated Multi-Facility Weber Problem mainly differs from FLaMECaST in having only two layers. In addition the SSCMFW does not consider flow through the edges.

The Single-Source Capacitated Multi-Facility Weber Problem is an NP-hard optimization problem [MTE12]. So heuristics have been developed in order to find a solution for the problem, one of these heuristics is the iterative two-phase algorithm proposed in [MTE12]. One key-method of the algorithm is alternating between mapping the customers to facilities based on a given embedding pos_F and calculating an embedding based on a given mapping of customers to facilities. This procedure is called *location-allocation* and aims on alternately optimizing the mappings z and the embeddings pos_F.

The metaheuristic proposed in Section 5.3 works with a comparable concept. Here, during an iteration step, the embedding pos_t of the current solution is used to heuristically select a candidate neighbor in the neighborhood of the current solution. Then after processing the candidate neighbor, the embedding pos_{t+1} based on the updated solution is calculated for the next iteration, resulting in a related form of location-allocation.

4 The FLaMECaST-Problem

This chapter introduces the optimization problem *FLaMECaST* and its concrete definition that is focused on in this thesis.

The FLaMECaST-problem is related to the Euclidean Steiner Tree Problem and holds some additional constraints [BGTZ14]. The following definition of FLaMECaST is based on the initial problem definition in the paper "The Flow-weighted Layered Minimal Euclidean Capacitated Steiner Tree Problem (FLaMECaST)" by Henrik Csöre and Elly Schmidt [CS24].

4.1 Problem Definition

The FLaMECaST optimization-problem works with multiple input parameters. These input parameters are given as follows.

The first parameter is a set P of points embedded in the two-dimensional Euclidean plane. This set is partitioned into a set S of *sources* and a set D of *sinks* with $S \dot{\cup} D = P$. Further given is a fixed number of *layers* $L \in \mathbb{N}$ with $L \geq 2$, *capacities* $c_2, c_3, .., c_L$ for each layer except the first layer and a fixed number $\alpha \in [0, 1]$. This number α works as a weight in the cost function and is explained more precisely below.

The general goal of FLaMECaST is computing a *total layered forest* $F = (V, E)$ with L layers that satisfies some constraints. Thereby the first layer consists of $|S|$ vertices and the last layer consists of $|D|$ vertices, representing the sources and the sinks in the forest. As a variation of the Euclidean Steiner Tree Problem, an arbitrary number of additional vertices called *Steiner vertices* can be added to the forest. These new vertices form the layers between the first and last layer, resulting in a valid total layered forest with L layers.

For a total layered forest F with L layers to be a *feasible solution* of the FLaMECaST-problem, it must fulfill the following constraints:

1. The forest F consists of $|D|$ in-trees, and each in-tree has exactly one vertex of the last layer as root
2. Every source in the first layer produces an outgoing flow of 1. This induces a flow f in the forest F, transporting the flow of a source to exactly one sink in the last layer
3. Every layer V_i of the forest F, except the first layer V_1, fulfills the *capacity restriction*: the flow going through a vertex $v \in V_i$ does not exceed the upper capacity limit c_i, i. e. $f(v) \leq c_i$

In the following, a *feasible forest* F refers to a total layered forest F with L layers, which is a feasible solution of the FLaMECaST-problem with respect to the presented restrictions.

Given a feasible forest F, an embedding function $pos : V \to \mathbb{R}^2$ for the forest F is *feasible* if the vertices of the first layer V_1 are mapped to the points in S and the vertices of the last layer V_L are mapped to the points in D. A feasible embedding function is further called an *optimal embedding* of F, if it minimizes the cost function

$$cost_{pos}(F) = \sum_{e \in E} len_{pos}(e) \cdot f(e)^{\alpha}$$

among all feasible embedding functions for the forest F, where $f(e)$ is the flow going through an edge e. As a result the *cost* of a feasible forest F is defined as $cost_{pos}(F)$ with regard to its optimal embedding pos.

So an *optimal solution* for the FLaMECaST optimization-problem is defined as the feasible forest F that has the minimal cost among all feasible forests for the FLaMECaST-instance. This means that the objective of FLaMECaST is to calculate a feasible forest F with the smallest cost arising through its embedding.

4.2 Problem Complexity

After introducing the FLaMECaST-problem, it is necessary to take into account a highly important characteristic of every optimization problem: its computational complexity.

Given a general instance of FLaMECaST with an arbitrary number of layers, FLaMECaST is NP-hard. The problem complexity can be proven by reducing the Euclidean Steiner Tree Problem to FLaMECaST. Since FLaMECaST is NP-hard and the number of possible topologies for a feasible forest F grows really fast with a growing problem instance, it is inevitable to find and use efficient heuristics to approach a good solution [CS24].

5 Algorithm

Within this chapter, a FLaMECaST-instance with a set of points $P = S \dot{\cup} D$, a number of layers $L \geq 2$, capacities $c_2, .., c_L$ and $\alpha \in [0, 1]$ is given. This chapter introduces an algorithm to solve the FLaMECaST-problem heuristically. The following is a brief overview of this chapter.

The first section provides basic knowledge of metaheuristics and describes the metaheuristic Simulated Annealing, the main metaheuristic used in this thesis. The second section introduces and explains important components and subroutines of the heuristic algorithm which are put together in the third section to define the final algorithm.

5.1 Simulated Annealing

Metaheuristics are abstract optimization algorithms that can be used to solve complex problems like NP-hard optimization problems in a feasible amount of time. In doing so, a metaheuristic does not necessarily find the global optimum but a heuristically good solution for the problem [TBS23].

Many metaheuristics work by efficiently searching in the *solution space* of an optimization problem for good solutions. The solution space is a set containing all feasible solutions of an optimization problem. The efficient search is accomplished by balancing between exploring the solution space and greedily improving current solutions to gain local optimal solutions.

Through *exploration*, the metaheuristic examines various regions of the solution space in order to discover regions of the solution space which may have good solutions for the optimization problem. In the same time the discovered regions are *analyzed* through local searches to obtain the best solutions in these regions. As a result a metaheuristic works by exploring several regions of the solution space and then analyzing the found regions for best solutions through local searches [TBS23]. But without proper exploration, the metaheuristic may not find the regions with the best solutions and therefore only find local optima.

The metaheuristic used in this thesis is *Simulated Annealing*. It is used on optimization problems that may possess multiple local optima and aims at finding a global optimal solution. Simulated Annealing is inspired by the physical process of cooling. Thereby a solid starts with a certain temperature and gradually cools down while performing changes to the solid so it eventually ends up frozen in a minimal energy configuration.

The idea of Simulated Annealing is described as follows. It starts with an initial solution and searches for an optimal solution by iterating through the solution space of an optimization problem. In each iteration step, Simulated Annealing selects a candidate solution of the solution space based on the current solution. Then it decides with which of the two solutions, the current or the candidate solution, the iteration process is continued. In Simulated Annealing, the candidate solution is used if its cost is lower than the cost of the current solution, otherwise the candidate solution is only chosen with a certain probability [BT93].

In the following, first it is described how Simulated Annealing iterates through the solution space S of an optimization problem. This is done by defining a *neighborhood* $N(s)$ of a solution s in the solution space. Given a solution $s \in S$, then s can be considered as multiple local parts which together build up the solution s. Such a local part can be changed with a neighborhood operation, resulting in a new solution s' called *neighbor* of s. The created neighbor s' is called feasible if s' is a feasible solution of the optimization problem and therefore part of the solution space S. As the neighborhood operation can be applied on all local parts of s, the set of all feasible neighbors of the solution s is the neighborhood $N(s)$ and can be calculated for every solution in the solution space. So Simulated Annealing can iterate through the solution space by repeatedly calculating the neighborhood $N(s)$ of a current solution s, selecting a neighbor $s' \in N(s)$ and continuing the iteration, if accepted, with this neighbor s'.

The method used to determine whether a candidate neighbor $s' \in N(s)$ is accepted or not works as follows. As already mentioned the goal of Simulated Annealing is to find a solution $s^* \in S$ of the optimization problem with minimal costs. For this purpose a *cost function* $H : S \to \mathbb{R}$ is given, assigning each solution $s \in S$ its corresponding cost $H(s)$. This cost function induces a set $S^* \subseteq S$ of global optima with $H(s^*) \leq H(s)$ for all $s^* \in S^*, s \in S$. Further a non-increasing function $T : \mathbb{N} \to (0, \infty)$ called *cooling schedule* is given with $T(t)$ for some $t \in \mathbb{N}$ called temperature at the iteration step t. So if the cost $H(s')$ of the candidate neighbor is lower than the cost $H(s)$ of the current solution, then the candidate neighbor s' is accepted as the new current solution. Otherwise the candidate neighbor s' is only accepted as the new current solution based on the *acceptance probability* $e^{-\frac{H(s')-H(s)}{T(t)}}$ given the temperature $T(t)$ of the current iteration step t.

By using the presented acceptance method, Simulated Annealing is likely to also accept worse solutions than the current solution in the beginning as the temperature $T(t)$ is higher, giving Simulated Annealing the ability to escape local minima. But as the temperature decreases, the acceptance probabilities for worse solutions than the current solution shrink so that Simulated Annealing rather performs a local search, converging to a local optimum that may be a global optimum.

The described iteration procedure continues until a specific termination condition is met. If enough iteration steps are done during the algorithm, it can be proven that, under certain conditions, the described metaheuristic Simulated Annealing converges to a global optimum $s^* \in S^*$ [BT93]. The implementation of the algorithm is given in Algorithm 5.1.

Algorithm 5.1: Simulated Annealing procedure

Input: An initial solution s_0.
Output: Heuristic solution of the optimization problem

```
1 t ⟵ 0
2 while Termination criterion not met do
3 |   s' ⟵ select_candidate_neighbor(s_t)
4 |   if H(s') ≤ H(s_t) then
5 |   |   s_{t+1} ⟵ s'
6 |   else
7 |   |   with probability e^{-(H(s')-H(s_t))/T(t)}: s_{t+1} ⟵ s', otherwise: s_{t+1} ⟵ s_t
8 |   t ⟵ t + 1
9 return s_t
```

5.2 Components of the Algorithm

In the following, four components of the algorithm to solve the FLaMECaST-problem are introduced. These components are used in the final heuristic described in Section 5.3.

In Section 5.2.1 two heuristics to create an initial solution for the FLaMECaST-problem are introduced. After defining the neighborhood of a solution in Section 5.2.2, Section 5.2.3 proposes a method to select a candidate neighbor based on this neighborhood. Section 5.2.4 then describes an approach to calculate the optimal embedding of a solution.

5.2.1 Initial solution

This section describes heuristic methods to construct an initial feasible forest F. The created initial solution is the starting point of the heuristic algorithm in Section 5.3.

Within this thesis, two heuristics to create an initial feasible forest F are described. These heuristics consist of two steps. First each source is assigned to exactly one sink while holding the capacity restriction of the last layer. This is done to guarantee that the capacity of the last layer is not exceeded after the forest is constructed. The assignment step of sources to sinks is the main difference between the two proposed heuristics described in this section. Next the mappings of each source to one sink are used to create an in-tree for each sink with its corresponding sources, so that the union of all in-trees form a feasible forest F.

Assigning sources to sinks As already mentioned each source must be assigned to exactly one sink. This can be done with various methods, like random or with a method resulting in minimal distances among all assignments by solving a min-cost-max-matching-problem.

With random assignment the sources are mapped to the sinks randomly. Here the sources are mapped to the sinks one at a time and randomly with uniform distribution over the sinks whose capacities are not exceeded yet.

The assigning method with min-cost-max-matching aims on creating a mapping of the sources to the sinks with minimal sum of distances between the sources and the assigned sinks. This works by formulating a min-cost-max-matching problem instance that can be solved in polynomial time with $\mathcal{O}(n^3)$ assuming n sources and n sinks [Mun57]. The min-cost-max-matching problem instance is defined as follows: given the sources V_1 and sinks V_L, their embeddings set by the embedding function $pos : V \to \mathbb{R}^2$ and the capacity of the last layer c_L. First the length $len_{pos}(e)$ of each edge $e = (s, d)$ from each source $s \in V_1$ to each sink $d \in V_L$ is calculated. Then a complete bipartite graph $G = (V', E')$ is created with the sources in V_1, each sink in V_L c_L-times and connecting each source with each sink. The weight of an edge $e \in E'$, $w(e)$, is defined as the calculated length between the source and the sink described by e. This problem instance can be solved with the Kuhn-Munkres-algorithm [Mun57]. As every source in a solution of the instance is connected to exactly one sink node because of the construction of G, every source can be assigned to its corresponding sink, resulting in a mapping with minimal sum of distances from the sources to the sinks.

Generating the in-trees The last step created mappings of each source to exactly one sink. These mappings can now be used to construct an in-tree for each sink with its assigned sources and heuristically low costs.

Given a sink node $d \in V_L$ with its corresponding sources $V_S \subseteq V_0$ which are mapped to the sink. To create an in-tree, the initial FLaMECaST-paper introduces the k-means-method which is based on divisive clustering the sources and constructs an in-tree with the sink d as root while holding all capacity restrictions [CS24]. The method uses the widely known k-means-algorithm [WW12]: given a set of points P in the Euclidean plane and a distance metric, like the Euclidean distance between two points. Then the k-means-algorithm tries to calculate k cluster centers z_i and a mapping of each point $p \in P$ to a cluster center so that the sum of the distances between the points in P and the assigned cluster centers are minimal, resulting in k clusters. The main downside of the k-means-algorithm is that it does not guarantee to find the global optimal solution but a local optimal solution [WW12].

The proposed k-means-method begins with the vertex $d \in V_L$ in the last layer and its corresponding sources V_S and recursively generates an in-tree as follows. The idea of the k-means-method in each recursive step is to create an in-tree with the current vertex as root and the assigned sources. Here the assigned sources are clustered and then for each cluster recursively an in-tree is created. The roots of these created in-trees become the children of the current vertex.

Let $u \in V_l$ be the current vertex in the l-th layer, c_{l-1} the capacity of the lower layer and V_u the sources assigned to the current vertex with corresponding positions S_u. First the positions in S_u are clustered with k-means with $k = \lceil \frac{|V_u|}{c_{l-1}} \rceil$, resulting in k clusters $S_1, \ldots, S_k$. If one of the clusters S_i contains more than c_{l-1} sources, then this cluster is re-clustered with k-means because the capacity c_{l-1} of the lower layer would be exceeded. Here the value k is chosen as the minimal number of clusters which must be created so that the clusters do not exceed the capacity c_{l-1}. Then for each cluster V_i with corresponding embeddings S_i a vertex v_i and recursively an in-tree with v_i as root is created. These roots then become the children of the current vertex u. The recursion stops when the second layer is reached, here each assigned source is directly connected to the current vertex u.

This k-means-method is chosen because it showed the best performance while having a relatively low execution time among all heuristics tested by the original FLaMECaST-paper [CS24]. The union of all in-trees created in this step forms a feasible forest F.

With the proposed steps the two heuristics further called *random* and *matching* generate an initial feasible forest F which serves as an initial solution for the FLaMECaST-problem.

5.2.2 Neighborhood of a solution

Within this section, the neighborhood of a solution s is defined. For this purpose a solution s of the FLaMECaST-instance, a feasible forest $F = (V, E)$, is given with its optimal embedding $pos : V \to \mathbb{R}^2$, its flow f and its corresponding cost $H(F) = cost_{pos}(F)$.

In the following, four neighborhood operations are introduced. Each neighborhood operation op induces a neighborhood $N_{op}(s)$, so that the neighborhood $N(s)$ of s is defined as the union of the four neighborhoods induced by the four neighborhood operations.

As described in Section 5.1, a neighborhood operation applies a local change on the solution F. So each proposed neighborhood operation below creates a neighbor by making a local change on the forest F and thus generating a neighbor-forest F'. By doing so only neighbor-forests F' which are *feasible* are part of the neighborhood $N(s)$ of the solution s. So for each neighborhood operation in the following a method to check feasibility is described. In order to illustrate the neighborhood operations, it is assumed that a part of the forest F is the graph shown in Figure 5.1.

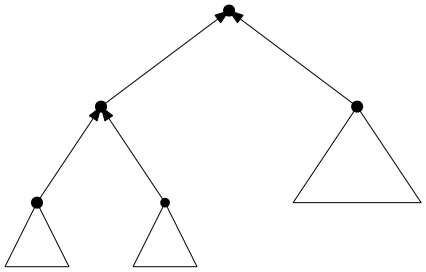


Figure 5.1: This graph is assumed to be part of a feasible forest F and is used to illustrate the neighborhood operations

Recable The idea of the neighborhood operation *Recable* is to change the parent of a vertex in the forest F. It uses a start $v \in V_i$ and a target $t \in V_{i+1}$ in the following layer for some $i \in \{1, \dots, L-1\}$ and changes the outgoing edge e_{old} of v to the new outgoing edge $e_{new} = (v, t)$. The recable-operation is shown in Figure 5.2.

This recable operation can lead to infeasible forests F' as the flow f' in F' changes. Let $w \in V$ be the common ancestor of v and t in the forest F if it exists, otherwise w is the root in the in-tree containing t. Then the flow of each vertex x in the path from t to w increases so that $f'(x) = f(x) + f(v)$. So for these vertices must be verified that the capacity is not exceeded. Furthermore if the old parent of v in F is not a sink, it also must be checked that the old parent has at least one child in the neighbor-forest F', otherwise F' is not a total layered forest and therefore not a feasible neighbor. As a result the feasibility-check for a recable-operation is in $\mathcal{O}(L)$.

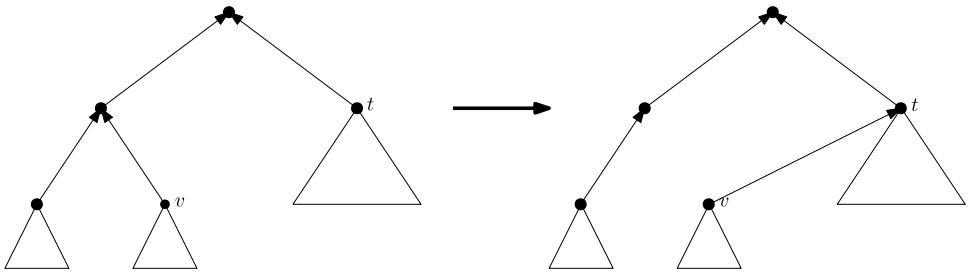


Figure 5.2: Illustration of the recable-operation with the start v and target t

Swap The idea of the neighborhood operation *Swap* is to swap the parents of two vertices in the forest F. Swap uses two vertices $v_1, v_2 \in V_i$ for some $i \in \{1, \dots, L-1\}$ and creates a neighbor-forest F' by swapping their parents, so that the new parent of v_1 is the old parent of v_2 and the new parent of v_2 is the old parent of v_1. The swap-operation is illustrated in Figure 5.3.

Similar to the neighborhood operation recable, the flow f' in F' changes so that the feasibility of the neighbor-forest F' must be verified. Let $w \in V$ be the common ancestor of v_1 and v_2 in the forest F. If this common ancestor exists, then the feasibility of the created neighbor-forest F' can be verified by checking the capacities of each vertex on the path from v_1 to w and each vertex on the path from v_2 to w. Otherwise both vertices lie in different in-trees with roots r_1, r_2, so that each vertex on the path from v_1 to r_1 and each vertex on the path from v_2 to r_2 must be checked. As in the neighborhood operation recable, this feasibility-check is in $\mathcal{O}(L)$.

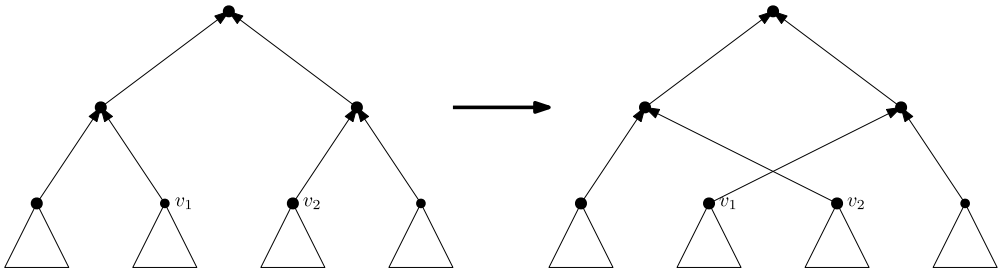


Figure 5.3: Illustration of the swap-operation with the two vertices v_1 and v_2

Merge The idea of the neighborhood operation *Merge* is to combine two vertices to one vertex in the forest F. For that Merge takes two vertices $v_1, v_2 \in V_i$ for some $i \in \{2, \dots, L-1\}$ which have the same parent. These two vertices are merged together by creating a new vertex v with the same parent as v_1 and v_2 and $children(v) = children(v_1) \cup children(v_2)$. Then the old vertices v_1 and v_2 are removed. The merge-operation is shown in Figure 5.4.

The neighborhood operation merge only changes the flow of the newly created vertex v with $f'(v) = f(v_1) + f(v_2)$. This means that the feasibility of the created neighbor-forest F' can be verified by checking the capacity of the new vertex v, so that the feasibility-check is in $\mathcal{O}(1)$.

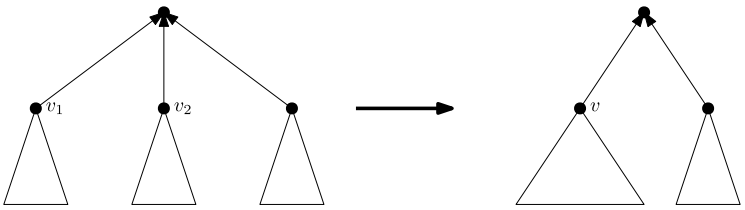


Figure 5.4: Illustration of the merge-operation with the two vertices v_1 and v_2

Split The neighborhood operation *Split* aims on creating two new vertices based on a single vertex in the forest F. It uses a vertex $v \in V_i$ for some $i \in \{2, \dots, L-1\}$ which has at least two children and an arbitrary partition $children(v) = S_1 \dot\cup S_2$ with $S_1, S_2 \neq \emptyset$ of the

children of v. Then new vertices v_1, v_2 are created with the same parent as v, so that the children of v_1 are set to the vertices in S_1 and the children of v_2 are set to the vertices in S_2. Then the old vertex v is removed. The split-operation is illustrated in Figure 5.5.

As only the vertex v is split and therefore its flow split with $f(v) = f'(v_1) + f'(v_2)$, the flows of v_1 and v_2 are smaller than the flow of v with $f'(v_1), f'(v_2) \leq f(v) \leq c_i$, which directly implies that the neighbor-forest F' is feasible.

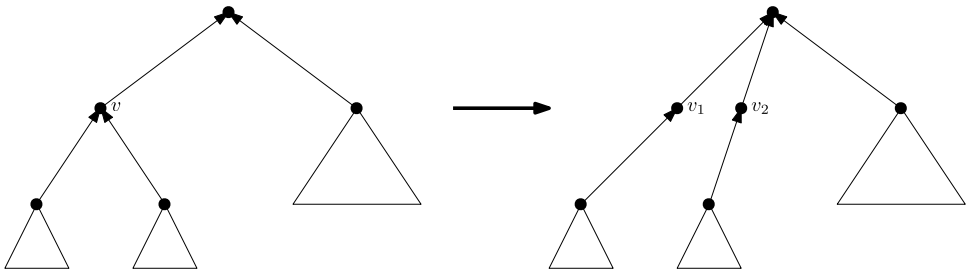


Figure 5.5: Illustration of the split-operation with the vertex v.

5.2.3 Candidate Neighbor Selection

In the following, a method to select a *candidate neighbor* $s' \in N(s)$ in the neighborhood of the current solution is described. As the complete neighborhood of the forest F is too large, the idea here is to only generate a part of the neighborhood of F which is then used to choose a candidate neighbor.

This is done by first selecting a fixed number $N \in \mathbb{N}$ of random vertices $C \subseteq V \setminus V_L$. Then all possible neighborhood operations based on the vertices in C are executed on the forest F, resulting in the neighborhood $N_C(s)$ of the current solution s.

Given a vertex $v \in C$ which lies in the i-th layer V_i, the neighborhood $N_v(s)$ based on v is obtained as follows. First every recable-operation with the start v and a target $t \in V_{i+1}$ is executed on the forest F, resulting in $|V_{i+1}|$ neighbor-forests. Then every swap-operation and every merge-operation, if v is not in the first layer, with $v_1 = v$ and $v_2 \in V_i \setminus \{v\}$ is performed on the forest F, further resulting in $|V_i| - 1$ neighbor-forests F' each. Finally, if v is not in the first layer, a split-operation is executed on the forest F. This split-operation is defined with v and the following partition of the children of v: given the positions P_{pos} of the children of v based on the embedding function pos. These positions are clustered with k-means with $k = 2$, resulting in two clusters whose corresponding vertices form a partition S_1, S_2 of the children of v.

Here only feasible neighbor-forests F' are included in $N_v(s)$. Methods to verify the feasibility of a neighbor-forest F' are described in Section 5.2.2. As a result the neighborhood $N_C(s)$ based on C is the union of all $N_v(s)$ with $v \in C$.

Given the neighborhood $N_C(s)$ based on the randomly selected vertices C, the neighbor $s' \in N_C(s)$ with the lowest heuristic cost $H'(s')$ is chosen to be the candidate neighbor. Here the costs of the neighbors in $N_C(s)$ are calculated heuristically because it is too expensive to embed every neighbor-forest F' with the method described in Section 5.2.4.

The formula for the heuristic cost $H'(s')$ of a neighbor $s' \in N_C(s)$ with the neighbor-forest $F' = (V', E')$ is similar to the actual cost function of F' presented in Section 4.1 and defined as:

$$H'(s') = \sum_{e \in E'} len_{pos'}(e) \cdot f'(e)^{\alpha}$$

Here, f' is the flow in the neighbor-forest F' and pos' is the heuristic embedding of F' mainly built with the positions of the given embedding pos of the forest F. As the proposed neighborhood operations only apply local changes on the forest F, the heuristic cost $H'(s')$ can be calculated by making small changes on the current cost $H(s)$ based on the neighborhood operation which created the neighbor-forest F'.

Recable In case of the neighbor-forest F' was constructed by the neighborhood operation recable, the embedding function pos' is set to the given embedding pos as recable does not create nor delete vertices.

Let $w \in V$ be the common ancestor of the start v and target t in the forest F if it exists. Then the flow f' in the neighbor-forest F' differs from the flow f in the forest F on the paths from v to w and from the old parent of v in the forest F to w, or to their roots if the common ancestor w does not exist. As the flows of the edges on these paths change, the costs of these edges also change. So the heuristic cost of the neighbor-forest F' can be calculated based on $H(s)$ by subtracting the costs of these edges in the forest F and adding the costs of these edges in the neighbor-forest F', e. g. $H'(s') = H(s) - cost_F(edges) + cost_{F'}(edges)$.

Swap The calculation of the heuristic cost of the neighbor-forest F' created by the neighborhood operation swap is similar to the calculation based on the neighborhood operation recable. As in recable before, the embedding function pos' is set to the given embedding pos. Further as in recable the flows on the paths from v_1, v_2 to their common ancestor or to their roots change, also resulting in the heuristic cost $H'(s') = H(s) - cost_F(edges) + cost_{F'}(edges)$.

Merge In case of the neighborhood operation merge created the neighbor-forest F', the embedding function pos' can be mainly built with the positions of the given embedding pos. But a position of the newly created vertex v must be determined. This can be done by creating a new forest F^* with three layers as follows: given the created vertex v as the second layer, then the children of v in F' built up the first layer and the parent of v constitute the last layer, the positions of the parent and the children are given by the embedding pos. Using the SOCP described in Section 5.2.4 on F^*, a position for the vertex v can be calculated. As only the embedding of one vertex must be determined, a solution of the SOCP and therefore the missing position of v can be found fast.

The local change resulting in the neighbor-forest F' was executed on v_1 and v_2 in the forest F. Therefore the heuristic cost of the neighbor-forest F' based on $H(s)$ is calculated by subtracting the costs of the surrounding edges of v_1, v_2 in the forest F and adding the costs of the surrounding edges of v in the neighbor-forest F', so that $H'(s') = H(s) - cost_F(edges) + cost_{F'}(edges)$.

Split If the neighbor-forest F' was created by the neighborhood-operation split, the heuristic cost of F' can be determined similar to the neighborhood-operation merge. Here the embedding function pos' is also mainly built with the positions in pos and the positions of

v_1, v_2 are determined with a SOCP as described for merge above. Then the heuristic cost of the neighbor-forest F' is calculated based on $H(s)$ by subtracting the costs of the surrounding edges of v in the forest F and adding the costs of the surrounding edges of v_1, v_2 in the neighbor-forest F', e. g. $H'(s') = H(s) - cost_F(edges) + cost_{F'}(edges)$.

5.2.4 Embedding

Given a feasible forest $F = (V, E)$ for the FLaMECaST-instance, the optimal embedding of the forest F is an embedding function $pos : V \to \mathbb{R}^2$ minimizing the cost function introduced in Section 4.1:

$$cost_{pos}(F) = \sum_{e \in E} len_{pos}(e) \cdot f(e)^\alpha$$

As the feasible forest F is fixed, the flow function f in the forest F is also fixed, resulting in constant weights $w(e) = f(e)^\alpha$ for all edges $e \in E$. So the problem of finding an optimal embedding function pos in the Euclidean plane can be written as

$$\min_{pos} \sum_{(u,v)=e \in E} w(e) \cdot \|pos(v) - pos(u)\|_2$$

Before solving the presented optimization problem, one notable characteristic of an optimal embedding is described. This result is proven in the original FLaMECaST-paper [CS24].

Given an optimal embedding pos^* of the forest F and a vertex $v \in V \setminus (V_1 \cup V_L)$, then the point $pos^*(v)$ lies at the *Weber point* of its surrounding points given by its surrounding vertices $N = \{parent(v)\} \cup children(v)$. The Weber point is defined as the *weighted geometric median* $p \in \mathbb{R}^2$ minimizing

$$\sum_{u \in N} w(u) \cdot \|pos^*(u) - p\|_2$$

where $w(u) = f(u)^\alpha$ for all $u \in children(v)$ and $w(parent(v)) = f(v)^\alpha$. This means that in an optimal embedding pos^* with fixed positions S, D of the sources and sinks, every point $pos^*(v)$ for all $v \in V \setminus (V_1 \cup V_L)$ is located at the Weber point of its surrounding points.

As there is no explicit formula for finding the Weber point for more than four points and Weber points therefore cannot be used to calculate an optimal embedding, the original FLaMECaST-paper introduces a *second-order cone program (SOCP)* to calculate an optimal embedding for the given forest F [CS24]. In the following, the number of layers is $L \geq 3$ because with $L = 2$ the optimal embedding function pos is trivial as the positions of the sources and sinks are fixed.

Let $N = \sum_{i=2}^{L-1} |V_i|$ be the number of Steiner vertices in the forest F. Without loss of generality, the vertices $v_1, \ldots, v_N$ are the Steiner vertices, the vertices $v_{N+1}, \ldots, v_{N+|S|}$ are the vertices of the first layer and the vertices $v_{N+|S|+1}, \ldots, v_{|V|}$ are the vertices of the last layer. Furthermore let $m_1 = |V_1|$ be the number of the edges starting in the sources and $m_2 = |V_{L-1}|$ be the number of edges ending in the sinks. Then $e_1, \ldots, e_{m_1}$ are the edges starting in sources, $e_{m_1+1}, \ldots, e_{m_1+m_2}$ are the edges ending in sinks and $e_{m_1+m_2+1}, \ldots, e_{|E|}$ are the other edges in F. In the following, an edge $e = (i, j)$ describes an edge starting in the vertex v_i and ending in

the vertex v_j. To find an optimal embedding pos of the forest F, the points $x_1, \ldots, x_N \in \mathbb{R}^2$ to the corresponding Steiner vertices $v_1, \ldots, v_N$ must be calculated based on the source points $S = \{c_{N+1}, \ldots, c_{N+|S|}\}$ and the sink points $D = \{c_{N+|S|+1}, \ldots, c_{|V|}\}$. This can be done with the following second-order cone program:

$$
\begin{aligned}
\min \quad & \sum_{e \in E} w(e) \cdot d_e \\
\text{s. t.} \quad & \|c_i - x_j\|_2 \leq d_e \quad \forall\, (i,j) = e \in \{e_1, \ldots, e_{m_1}\} \\
& \|x_i - c_j\|_2 \leq d_e \quad \forall\, (i,j) = e \in \{e_{m_1+1}, \ldots, e_{m_1+m_2}\} \\
& \|x_i - x_j\|_2 \leq d_e \quad \forall\, (i,j) = e \in \{e_{m_1+m_2+1}, \ldots, e_{|E|}\} \\
& x_i \in \mathbb{R}^2 \quad \forall\, i = 1, \ldots, N \\
& d_e \in \mathbb{R} \quad \forall\, e \in E
\end{aligned}
$$

The proposed SOCP can be solved in $\tilde{\mathcal{O}}(N^\omega \cdot log(\frac{1}{\epsilon}))$ where ω is the matrix multiplication exponent and ϵ is the relative accuracy. Since any feasible embedding for F is a feasible solution for the second-order cone program and the problem is convex, an initial solution to the SOCP exists and the solution $x_1, .., x_N \in \mathbb{R}^2$ is an ϵ-optimal solution [CS24].

As a result, the optimal embedding $pos : V \to \mathbb{R}^2$ of the forest F can be defined as:

- $pos(v_i) = x_i$ for all v_i with $i = 1, \ldots, N$
- $pos(v_i) = c_i$ for all v_i with $i = N+1, \ldots, |V|$

5.3 Algorithm for FLaMECaST

In this section, the algorithm to calculate a heuristic solution for the FLaMECaST-problem is introduced. It puts together the components described in Section 5.2 into the Algorithm 5.2.

The algorithm starts by calculating an initial solution s_0 of the FLaMECaST-instance and its optimal embedding pos_0 with a heuristic introduced in Section 5.2.1. In each iteration of the algorithm, first a candidate neighbor s' with the method described in Section 5.2.3 based on the current solution s_t and embedding pos_t is selected. The candidate neighbor s' is then embedded with the embedding algorithm presented in Section 5.2.4 in order to get its cost $H(s')$. The process of setting the new current solution s_{t+1} works as described in Section 5.1: if the cost of the candidate neighbor $H(s')$ is lower than the cost of the current solution $H(s_t)$, then the candidate neighbor is accepted as the new current solution s_{t+1}. Otherwise the candidate neighbor is used as the new current solution s_{t+1} based on the acceptance probability proposed in Section 5.1. The iteration process terminates after a fixed number of iterations are done. During the iterations the current solutions could get worse as also worse solutions can be accepted, but in every iteration step the best solution found until now is stored. So as the iterations stop, the best solution found whilst iterating through the solution space of the FLaMECaST-instance is returned.

Algorithm 5.2: Simulated Annealing for FLaMECaST

Input: A FLaMECaST-instance
Output: Best solution found

$t \longleftarrow 0$
$s_0, pos_o \longleftarrow heuristic()$
while $t < max\_iteration$ **do**
  $s' \longleftarrow select\_candidate\_neighbor(s_t, pos_t)$
  if $H(s') \leq H(s_t)$ **then**
    $s_{t+1}, pos_{t+1} \longleftarrow s', pos'$
  else
    with probability $e^{-\frac{H(s')-H(s_t)}{T(t)}}$: $s_{t+1}, pos_{t+1} \longleftarrow s', pos'$,
    otherwise: $s_{t+1}, pos_{t+1} \longleftarrow s_t, pos_t$
  $t \longleftarrow t + 1$
return s_{t^*}, pos_{t^*}

6 Evaluation

Within this chapter, the metaheuristic algorithm introduced in Section 5.3 is closer inspected. In doing so, several characteristics of the algorithm are examined by executing the algorithm with various configurations and different FLaMECaST-instances.

The implementation of the algorithm proposed in Section 5.3 is written in Rust 1.84.0 and compiled with cargo 1.84.0. In the following, all experiments are run on Ubuntu 24.04 LTS with a Intel(R) Xeon(R) Gold 6144 and 192 GB of memory in single-threaded mode to ensure comparability.

In order to perform experiments with the introduced algorithm, *benchmark instances* with certain numbers of sources and sinks embedded in the Euclidean plane, fixed numbers of layers and capacities for the corresponding layers are generated. These benchmark instances are divided into three categories: *medium* with six layers, *medium* with ten layers and *large* with six layers. The specified amount of six layers is inspired by the solar farm model with six layers introduced in Chapter 1. The medium instances are created with 600-1000 sources and 20-30 sinks and the large instances have 1400-1800 sources and 20-30 sinks. These nodes are then embedded randomly on a 1 by 1 square in the two-dimensional Euclidean plane. Furthermore the capacities of the layers are generated randomly for each FLaMECaST-instance in ascending order with $\lceil \frac{|sources|}{|sinks|} \rceil$ as the maximal capacity of the sinks to ensure that feasible solutions of the instances exist. These capacities are chosen in ascending order because in a feasible forest F of a FLaMECaST-instance, every vertex $v \in V_i$ in the ith layer passes its flow $f(v)$ to exactly one vertex in the following layer and thus this flow must not exceed the capacity of the following layer, e. g. $f(v) \leq c_{i+1}$.

As described in Section 5.1, Simulated Annealing uses a non-increasing cooling schedule $T : \mathbb{N} \to (0, \infty)$. This cooling schedule leads to also accepting worse solutions in the beginning and rather performing a local search in later iteration steps. A widely used cooling schedule is the *exponential schedule* $T(t) = T_0 \cdot \alpha^t$ where T_0 is the initial temperature at the start of Simulated Annealing and α is the temperature decreasing factor of each iteration [NA98]. In the following, these parameters are selected depending on the initial cost F_0 of a FLaMECaST-instance. The initial temperature T_0 is chosen so that in the beginning, the probability of accepting a neighbor which increases the initial cost by $0.2 \cdot F_0$ is 0.95. Further the decreasing factor α is chosen so that after 95% of the iteration steps are done, the probability of accepting a neighbor which increases the cost of the current solution by $0.0001 \cdot F_0$ is 0.01. This selection of the parameters in combination with the exponential schedule results in the desired behavior of Simulated Annealing described in Section 5.1 as the accepting probabilities of neighbors with worse costs are high in the beginning and low in the end.

6.1 Initial Solution

In this section the effect of the heuristic algorithm chosen to construct an initial solution of the FLaMECaST-problem is analyzed. In the following, it is tried to figure out which of the two heuristics, random or matching, introduced in Section 5.2.1 shows the best performance. As described in Section 5.2.1 the heuristics work by first finding an assignment of the sources to the sinks and then creating in-trees for each sink using the k-means-method. Here the benchmark instances work with $\alpha = 0.1$.

To test the influence of the heuristics on the algorithm, both heuristics are used on all benchmark instances to create initial solutions. These initial solutions then are the starting points for the iterations of Simulated Annealing as described in Section 5.3.

After the iterations finished, the final solutions based on the matching heuristic showed in median 9,2% (quartiles: 7,1% and 11,8%) less costs than the final solutions based on the random heuristic in medium instances with six layers. In parallel medium instances with ten layers resulted in median with 15,6% (quartiles: 13,1% and 19,8%) less costs and large instances in median with 7,3% (quartiles: 6,1% and 8,8%) less costs.

For that Simulated Annealing also needed less time until convergence based on initial solutions created by the matching heuristic as shown in Table 6.1. Within this thesis, convergence means that the cost of the best solution found until a certain iteration step is only 2% higher than the cost of the best solution found during the whole algorithm. In addition using initial solutions created by the random heuristic leaded to convergence in later iteration steps as illustrated in Table 6.2.

As a result using the matching heuristic to create initial solutions for FLaMECaST-instances showed the best performance. But it must be noted that with growing FLaMECaST-instances the matching-heuristic may need too much time as its runtime is cubic.

In the following experiments, the matching-heuristic is used to construct initial solutions.

Table 6.1: Time needed until convergence based on the benchmark instance and heuristic used to create an initial solution.

Time (s)	Matching			Random		
	Lower Quartile	Median	Upper Quartile	Lower Quartile	Median	Upper Quartile
Medium - six layers	48,3	60,8	80,4	60,1	75,8	96,1
Medium - ten layers	90,7	120,0	152,2	113,6	146,2	192,8
Large	210,0	239,6	261,8	225,1	267,1	303,1

Table 6.2: Percentage of total iterations done until convergence took place based on the benchmark instance and heuristic used to create an initial solution.

Iterations done (%)	Matching			Random		
	Lower Quartile	Median	Upper Quartile	Lower Quartile	Median	Upper Quartile
Medium - six layers	56,3	62,3	67,9	80,4	83,7	86,6
Medium - ten layers	58,1	63,8	68,5	81,6	85,0	88,7
Large	64,0	69,6	73,8	83,3	86,0	88,0

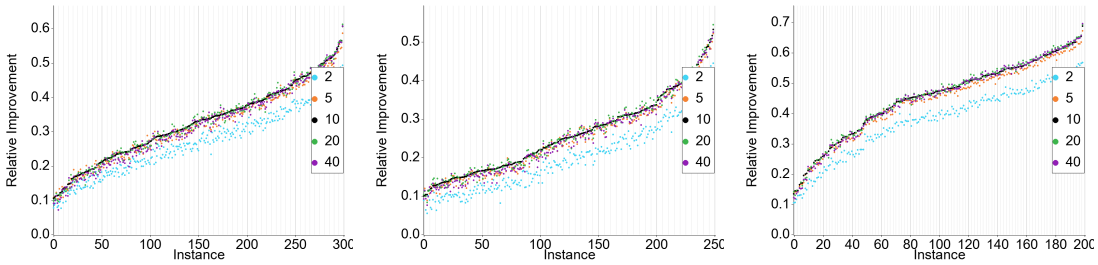
6.2 Number of Random Vertices

In each iteration step, the selection of a candidate neighbor described in Section 5.2.2 works by choosing a fixed number of random vertices in the current solution. This number of vertices can be set arbitrarily. Within this section, the influence of the number of random vertices selected is examined. This is done by defining 2, 5, 10, 20 and then 40 as the number of selected vertices in each iteration step and running experiments with the benchmark instances. Like in Section 6.1 the benchmark instances work with $\alpha = 0.1$.

As described in Section 4.2, the FLaMECaST-problem is a NP-hard problem and therefore the optimal solutions of the created FLaMECaST-instances are not known. So to evaluate the solutions of the experiments, the *relative improvement* of the instances is used. The relative improvement of an instance is the improvement of the cost done from the initial solution to the final solution of Simulated Annealing relative to the initial solution, e. g. $\frac{init\_cost - final\_cost}{init\_cost}$, and describes the improvement of the cost done by Simulated Annealing independently of the absolute cost of an instance.

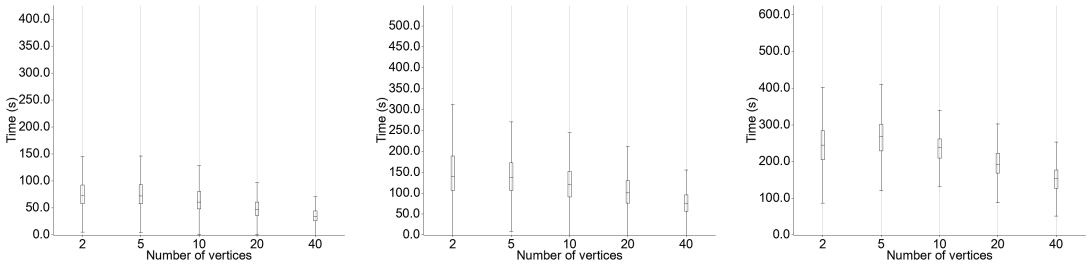
In Figure 6.1 it can be observed that the number of vertices selected in the iteration steps affected the relative improvement done by the algorithm. By only selecting two vertices the improvement was smaller in each benchmark instance type. Although selecting five vertices seems sometimes as good as the other number of vertices, it performed mostly worse. Selecting 10, 20 and 40 random vertices in each iteration step showed the best relative improvements, whereas selecting 40 vertices was often also worse in medium instances.

Furthermore it can be observed that an increasing number of vertices selected in the iteration steps leaded to less time needed until convergence of the algorithm as shown in Figure 6.2. One reason for this behavior is that the convergence took place in earlier iteration steps as illustrated in Figure 6.3, here the algorithm also might not have been fully converged for lower number of vertices. However the results show that a higher number of vertices selected per iteration step leads to better improvements while having earlier convergence of Simulated Annealing, until a certain degree as selecting 40 vertices showed also slightly worse improvements on medium instances.



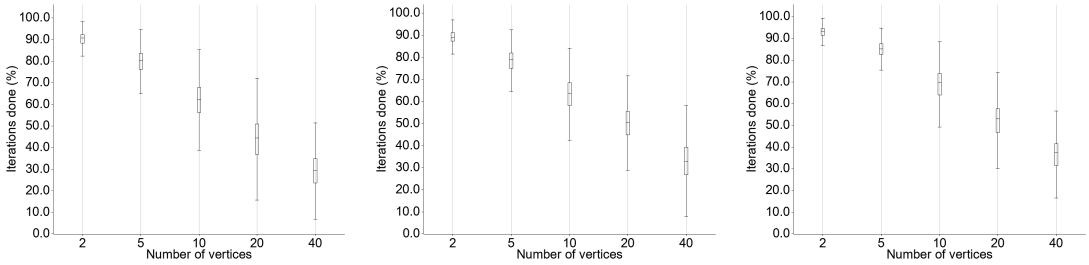
(a) Medium with six layers **(b)** Medium with ten layers **(c)** Large with six layers

Figure 6.1: Plots illustrating the relative improvement done by each benchmark-instance, depending on the number of vertices selected in each iteration step and sorted by 10 vertices for clarity.



(a) Medium with six layers **(b)** Medium with ten layers **(c)** Large with six layers

Figure 6.2: Plots illustrating the time taken until convergence took place for each benchmark instance category, depending on the number of vertices selected in each iteration step.



(a) Medium with six layers **(b)** Medium with ten layers **(c)** Large with six layers

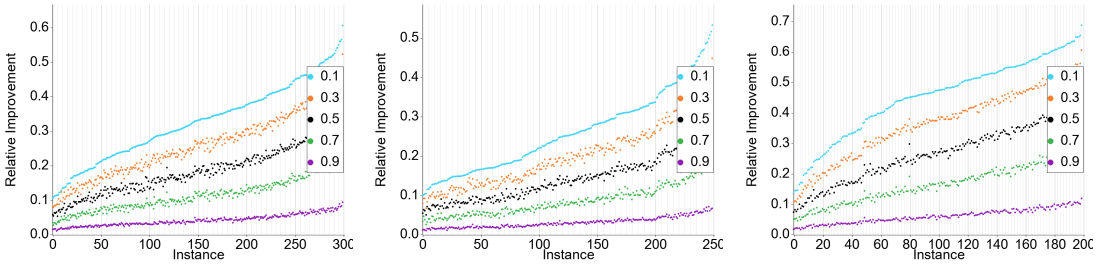
Figure 6.3: Plots illustrating the percentage of iteration steps made before convergence for each benchmark instance category, depending on the number of vertices selected in each iteration step.

6.3 Alpha Value

In the following, the influence of the alpha value α given through a FLaMECaST-instance on the performance of the algorithm described in Section 5.3 is analyzed. Here the alpha values 0.1, 0.3, 0.5, 0.7 and 0.9 are chosen as these values mostly cover the domain of alpha $\alpha \in [0, 1]$ introduced in Section 4.1. Then the proposed algorithm is executed with the benchmark instances paired with each alpha value.

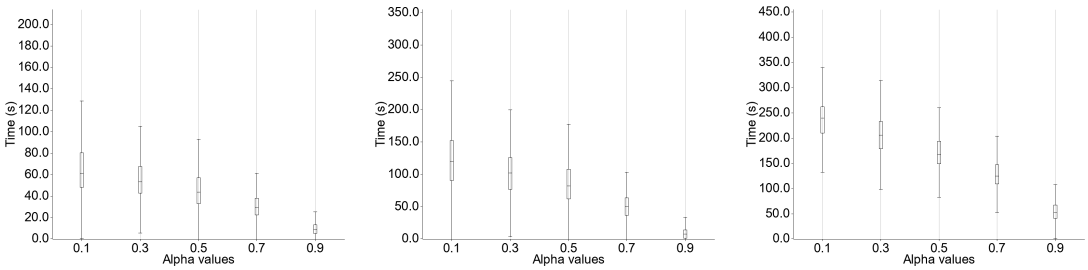
By running the experiments, it can be observed in Figure 6.4 that the relative improvement of the solution quality shrinks with increasing alpha values. In addition instances with higher alpha-value further showed earlier convergence as illustrated in Figure 6.5. This means that Simulated Annealing can lead to better improvements on FLaMECaST-instances with lower alpha values than on FLaMECaST-instances with higher alpha-values. Here the exact reason for this behavior cannot be given as the optimal solutions of the instances are not known. On the one hand, one reason for this behavior could be an insufficient exploration of the solution space, for example by not considering an enough number of vertices in a solution for the instance. With increasing alpha values the cost of an edge as described in Section 4.1, $len(e) \cdot f(e)^{\alpha}$, grows when having a high flow through the edge. So more vertices could split the flow and therefore reduce the cost. But while selecting a candidate neighbor as described in Section 5.2.2, only one split operation per selected random vertex is specified and further adds only one additional vertex to the solution. On the other hand, the small improvement of the costs could also be the result of an already good initial solution given by the matching-heuristic.

In order to get further insights on the solution quality of Simulated Annealing, the following section examines special FLaMECaST-instances whose optimal solutions are given.



(a) Medium with six layers **(b)** Medium with ten layers **(c)** Large with six layers

Figure 6.4: These plots illustrate the relative improvement done for each benchmark instance, depending on the alpha value α.



(a) Medium with six layers **(b)** Medium with ten layers **(c)** Large with six layers

Figure 6.5: These plots illustrate the time needed until convergence for each benchmark instance, depending on the alpha value α.

6.4 Circular Instances

The original paper on FLaMECaST introduced a class of instances which can be solved in polynomial time, the *circular instances* [CS24]. In this class, each instance contains three layers, exactly one sink node and n sources that are placed with equal distances on a circle with radius 1 and the sink node as center. As FLaMECaST-instances in this class can be solved in polynomial time, their optimal costs can be calculated and compared with the costs of the solutions found with the algorithm proposed in Section 5.3. In the following, the number of sources n are chosen in the range between 1 and 10000 and the resulting instances are paired with the alpha values 0.1, 0.3, 0.5, 0.7 and 0.9.

After running Simulated Annealing on the created instances, it can be observed in Figure 6.6 that the solutions found during the iterations only have slightly bigger relative costs than the optimal solutions of the instances. Here solutions of the instances with bigger alpha values are closer to the optimal solution.

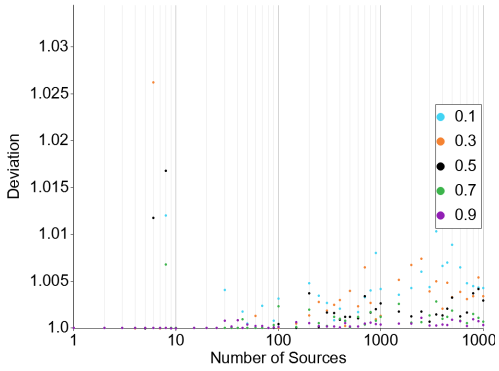


Figure 6.6: Plot illustrating the deviation of the solution found by Simulated Annealing to the optimal solution, measured in $\frac{cost(found)}{cost(optimal)}$.

This observation shows that Simulated Annealing can calculate good solutions of circular instances. However as these experiments only examined one specific class of FLaMECaST-instances, circular instances, further claims on the quality of solutions calculated by Simulated Annealing cannot be made for general instances.

7 Conclusion

In this thesis, an implementation of the metaheuristic Simulated Annealing for the Flow-weighted Layered Minimal Euclidean Capacitated Steiner Tree Problem was proposed. For this purpose, a neighborhood for solutions of the FLaMECaST-problem was introduced. This neighborhood was defined through the four neighborhood operations recable, swap, merge and split. To ensure feasibility of the solutions found by Simulated Annealing during the whole iteration process, methods were described to efficiently check the feasibility of the neighbors created by the neighborhood operations. Using these methods it could be guaranteed that only feasible neighbors were part of the neighborhood of a solution.

Furthermore an approach to select a candidate neighbor in the neighborhood of a current solution was explained. Here only a part of the neighborhood was calculated as the complete neighborhood of a solution is too large. The selection of a candidate neighbor then was derived by formulating methods to assign each neighbor heuristic costs. Here it was analyzed that selecting fixed numbers of vertices like 10 or 20 in each iteration step to calculate a candidate neighbor has proven the best improvement while also having an early convergence.

In addition two heuristics were introduced to construct an initial solution, the random-heuristic and the matching-heuristic. The created initial solutions then served as the starting point for the Simulated Annealing procedure. Experiments showed that Simulated Annealing had the best performance by constructing an initial solution using the matching-heuristic which is based on solving a min-cost-max-matching problem and using the k-means-method.

Finally the described components were put together in order to define the Simulated Annealing routine. It was analyzed that the created algorithm achieved a good reduction of costs on FLaMECaST-instances with lower alpha values, but as the alpha values grew the relative improvement done by Simulated Annealing based on initial solutions decreased. As the optimal solutions of these instances could not be calculated because FLaMECaST is an NP-hard optimization problem, the behavior of Simulated Annealing was further investigated on circular instances, a class of instances whose optimal solutions are known. Here the proposed algorithm showed good performance as it calculated solutions with costs only slightly bigger than the costs of optimal solutions.

7.1 Future Work

Within this thesis, the first approach to implement the metaheuristic Simulated Annealing to solve the FLaMECaST-problem was introduced. This means that in the future further research on this topic can be done based on the contributions of this thesis. For example the definition of the FLaMECaST-problem presented in this work only considered two-dimensional points in the Euclidean Plane, whereas the FLaMECaST-problem can be defined for an arbitrary number of dimensions. Here the concepts of the algorithm proposed within this thesis can be transferred to FLaMECaST-problems with arbitrary dimensions.

Furthermore the components of the algorithm introduced in Section 5.2 can be enhanced. As mentioned in Section 5.2.1 the runtime of the matching-heuristic is cubic so that it might be slow for large instances. Here the performance of additional heuristics could be examined, like a greedy approach, assigning each source sequentially to its nearest sink. Also expansions of the proposed neighborhood operations and its impact on the performance of Simulated Annealing could be analyzed, for example by expanding the merge and split neighborhood operation, making it work with more than two vertices at once.

In addition modifications on the Simulated Annealing procedure defined in Section 5.3 can be done. Here further termination criteria could be examined, like stopping the iterations if no progress in the solution quality was made in a certain number of iterations. Also re-annealing could be added to the Simulated Annealing routine, so that after an iteration process stopped, the iterations restart with the best found solution as the initial solution.

Bibliography

[23] "Emissions Trends and Drivers". In: *Climate Change 2022 - Mitigation of Climate Change: Working Group III Contribution to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change*. Cambridge University Press, 2023, pp. 215–294.

[Age24] International Renewable Energy Agency. *Renewable power generation costs in 2023*. IRENA, 2024.

[Age25] International Renewable Energy Agency. *Renewable Capacity Statistics 2025*. IRENA, 2025.

[BGTZ14] Marcus Brazil, Ronald L Graham, Doreen A Thomas, and Martin Zachariasen. "On the history of the Euclidean Steiner tree problem". In: *Archive for history of exact sciences* Volume 68 (2014), pp. 327–354.

[BT93] Dimitris Bertsimas and John Tsitsiklis. "Simulated annealing". In: *Statistical science* Volume 8 (1993), pp. 10–15.

[CS24] Henrik Csöre and Elly Schmidt. *The Flow-weighted Layered Minimal Euclidean Capacitated Steiner Tree Problem (FLaMECaST)*. 2024.

[Fil+24] Mikalai Filonchyk, Michael P. Peterson, Lifeng Zhang, Volha Hurynovich, and Yi He. "Greenhouse gases emissions and global climate change: Examining the influence of CO2, CH4, and N2O". In: *Science of The Total Environment* Volume 935 (2024), p. 173359. ISSN: 0048-9697. DOI: *https://doi.org/10.1016/j.scitotenv.2024.173359*.

[GSW22] Sascha Gritzbach, Dominik Stampa, and Matthias Wolf. "Solar farm cable layout optimization as a graph problem". In: *Energy Informatics* Volume 5 (2022), p. 25.

[KVKV11] Bernhard H Korte, Jens Vygen, B Korte, and J Vygen. *Combinatorial optimization*. Vol. 1. Springer, 2011.

[MTE12] SMH Manzour-al-Ajdad, S Ali Torabi, and Kourosh Eshghi. "Single-source capacitated multi-facility weber problem—an iterative two phase heuristic algorithm". In: *Computers & Operations Research* Volume 39 (2012), pp. 1465–1476.

[Mun57] James Munkres. "Algorithms for the assignment and transportation problems". In: *Journal of the society for industrial and applied mathematics* Volume 5 (1957), pp. 32–38.

[NA98] Yaghout Nourani and Bjarne Andresen. "A comparison of simulated annealing cooling strategies". In: *Journal of Physics A: Mathematical and General* Volume 31 (Oct. 1998), p. 8373. DOI: *10.1088/0305-4470/31/41/011*.

[PFG22] Jacob Poushter, Moira Fagan, and Sneha Gubbala. "Climate change remains top global threat across 19-country survey". In: *Pew Research Center* (2022).

[TBS23] Vinita Tomar, Mamta Bansal, and Pooja Singh. "Metaheuristic Algorithms for Optimization: A Brief Review". In: *Engineering Proceedings* Volume 59 (2023). ISSN: 2673-4591.

[Wah22] Leonie Wahl. "Variable Neighborhood Search for the Solar Farm Cable Layout Problem". Bachelor's thesis. Karlsruhe Institute of Technology (KIT), 2022.

[WBR23] Alexander Westcott, Marcus Brazil, and Charl Ras. "Structural Properties of Minimum Multi-source Multi-Sink Steiner Networks in the Euclidean Plane". In: *Journal of Optimization Theory and Applications* Volume 197 (2023), pp. 1104–1139.

[WW12] Junjie Wu and Junjie Wu. "Cluster analysis and K-means clustering: an introduction". In: *Advances in K-Means clustering: A data mining thinking* (2012), pp. 1–16.