Karlsruhe Institute of Technology

Planning Geometric Microgrid Cable Layouts

Master's Thesis of

Max Göttlicher

At the Department of Informatics
Institue of Theoretical Informatics

Reviewer:	PD Dr. Torsten Ueckerdt
Second reviewer:	Prof. Dr. Peter Sanders
Advisors:	Matthias Wolf

March 1, 2021 – September 1, 2021

Statement of Authorship

Ich versichere wahrheitsgemäß, die Arbeit selbstständig verfasst, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde sowie die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Karlsruhe, September 1, 2021

Abstract

We study the problem of finding a minimum cost cable layout for microgrids. The cost of a microgrid is determined by the chosen cable types and their lengths. Introducing additional distribution nodes can reduce the total cable length and the associated cost. Previous approaches to cable layout planning rarely include such freely placable Steiner nodes.

Our genetic algorithm integrates planning a cable layout with placing distributions nodes and selecting appropriate cable types. We introduce a set of new genetic operators for geometrically embedded trees with fixed terminals and a variable number of movable Steiner points. Our genetic algorithm can easily be adapted to different problem settings including microgrid cabling, finding the minimum geometric Steiner tree. For microgrid cabling, we introduce a linear time computation of the cable assignment method introduced by Kraft. Our experiments show that our algorithm finds good solutions to the geometric Steiner tree problem on up to 100 terminals and finds a cable layout in a real world microgrid example in less than 30 seconds.

Deutsche Zusammenfassung

Wir untersuchen die kosteneffiziente Verkabelung von Microgrids. Die Kabeltypen und Kabellängen sind bedeutende Kostenfaktoren im Aufbau eines Microgrids. Die Verlegung von Kabeln über zusätzliche Verteilknoten kann die Gesamtlänge und damit die Kosten reduzieren. Bisherige Planungsmethoden berücksichtigen selten frei platzierbare Verteilknoten.

Unser genetischer Algorithmus kombiniert die Planung der Kabeltrassen, Platzierung von zusätzlichen Verteilknoten und die Auswahl der verwendeten Kabel. Dazu stellen wir neue genetische Operatoren für geometrisch eingebettete Bäume vor, die auch Steinerpunkte berücksichtigen. Unser genetischer Algorithmus kann mit verschiedenen Problemstellungen verwendet werden, darunter das geometrische Steinerbaumproblem und Microgridverkabelung. Die Kabelbelegung berechnen wir mit einer neuen Linearzeitumsetzung der von Kraft vorgestellten Methode. In Experimenten konnte unser Algorithmus gute Lösungen für das geometrische Steinerbaumproblem mit bis zu 100 Knoten in weniger als einer Minute und eine Verkabelung für ein reales Microgrid in weniger als 30 Sekunden berechnen.

Contents

1 Introduction

Access to electricity today is ubiquitous in the industrial world. But as of 2021 more than 700 million people still do not have access to electricity [29]. Especially in rural communities in Africa access to public power grids is rare. In this situation microgrids can provide electricity to small industrial workshops and stationary agricultural machines, boosting economic development. Khodayar [33] defines a microgrid as a group of interconnected electric generators and consumers within a clearly defined boundary that acts as a single controllable entity. Microgrids can be set up to increase independence from an unreliable power grid or to provide electricity in an otherwise unconnected area. If the microgrid operates completely separated from public electric infrastructure in island mode it is called an off-grid microgrid. Off-grid systems are not only found in developing countries but also in remote settlements in industrial countries such as Australia and Canada [2]. In this thesis we will always refer to off-grid microgrids.

Such installations can be powered by diesel generators or by renewable energies. Diesel fuel and generator maintenance are more expensive than power from public grids. Renewable energy sources are widely available and can deliver electricity at a lower cost. Hydroelectric plants for example can provide a steady and reliable source of electricity at low operating costs [28]. Other disadvantages of diesel engines include their noise level and environmental hazards [2]. Hydroelectric plants have to be adapted to local geography and cannot always be placed next to the electric loads like diesel generators and photovoltaic systems. Diesel engines can be constructed next to the electric consumers, which is not always possible with hydroelectric plants. Similarly, wind turbines are more efficient if placed on higher ground, which might not be where the consumers are located.

All generation methods need to be connected using cables forming an electric grid. If the distance exceeds some ten meters, cable resistance can reach noticeable levels and the installation requires more careful planning. Cables are a major cost factor in an electric grid. Optimizing the cable layout therefore has a large impact on the final cost [20]. The cable layout involves the routing of the cables and the selection of cable types. Planning of cable routes is still usually done by hand.

1.1 Contribution

In this thesis, we introduce a novel set of genetic operators to be used with problem settings related to Steiner trees. These operators have been developed with a special focus on cable routing in small electric grids. Classical mutation operators are not suited for geometric problems, especially not if the number of points involved varies across the individuals. We introduce mutation operators for edges in geometric problems and for the placement of Steiner nodes. To our knowledge, this approach has not been used on euclidean Steiner trees or similar problems before. We further extend the heuristic cable assignment method by Kraft [34] to be used in the fitness computation of our genetic algorithm by introducing a linear time algorithm to compute the cable assignment.

To test our algorithm's general ability to solve such network optimization problem optimally we perform experiments on Steiner trees. We further test the cable assignment using consumer data from a real-world microgrid. We also provide a proof-of-concept showing that our algorithm can be used with solid obstacles.

1.2 Outline

We first introduce the main concepts on which this work is based in chapter 2. We define the Microgrid Cabling Problem (MCP) in chapter 4. The electric constraints we have to observe yields a subproblem of MCP, the Cable Assignment Problem defined in section 4.2 where we also present an efficient heuristic for Cable Assignment. In chapter 5 we present our genetic algorithm for Steiner tree problems and its application in microgrid cabling section 5.6. We conducted experiments with our algorithm and present the results in chapter 6.

2 Preliminaries

In this chapter, we introduce the formal definitions of the basic concepts used in this thesis. We start with the relevant notions of graph theory in section 2.1 followed by the introduction of the geometrical concepts used in this thesis in section 2.2. Then we give an overview of genetic algorithms in section 2.3 and then introduce the physical laws and rules of electric systems in section 2.4.

2.1 Graph Theory

Electrical grids are composed of nodes which can be consumers, power stations or distribution nodes. These nodes are connected by cables. The natural description for such a network is a graph where the electrical nodes are vertices and the cables are edges. The following definitions are based on "Graph Theory" by Diestel [14].

An *undirected graph* $G = (V, E)$ is a pair of two sets, the set of vertices V and the set of edges $E \subseteq [V]^2$. By $[V]^2$ we denote the set of 2-element subsets of V. We use the shorthand notation uv for undirected edges $\{u, v\} \in E$. We only consider simple graphs, i.e. graphs without multiple edges or loops. Similarly, a *directed graph* is a pair $G = (V, E)$ where $E \subseteq V^2$. Each edge $(u, v) \in E$ is directed from its *tail* u to its *head* v. An undirected graph can be represented as a directed graph by replacing each undirected edge with two opposing directed edges. In this work we will always refer to undirected simple graphs, unless stated otherwise.

Two nodes $u, v \in V$ are *adjacent* if and only if $uv \in E$. In a directed graph v is adjacent to u if $(u, v) \in E$. The neighborhood $\text{neigh}(v) = \{u \in V \mid uv \in E\}$ is the set of all vertices adjacent to v. An edge uv is incident to a vertex x if and only if $x = u$ or $x = v$. A *path* is a sequence of vertices $x_0, \ldots, x_k$ such that $x_i x_{i+1} \in E$ or similarly $(x_i, x_{i+1}) \in E$ in a directed graph. A path is *simple* if all vertices $x_i \neq x_j (i \neq j)$ are distinct. If there exists a path between every pair of vertices $x, y \in V$, the graph is *connected*. A *cycle* is a path $p = (x_0, \ldots, x_k)$ where the first and last vertices x_0 and x_k are the same and all other vertices are distinct.

A *forest* is an undirected graph without cycles. A *tree* is a connected forest.

Equivalently, a tree is a connected graph with $|E| = |V| - 1$. In a *rooted tree* a single vertex is designated as a root r. Edges in a rooted tree have a natural direction away from the root. The *tree-order* is a partial ordering where $x \leq y$ if and only if x is on the path from the root to y.

A *spanning tree* T of a graph $G = (V, E)$ is a tree on V whose edges are a subset of E. In a weighted graph with weights $w : E \to \mathbb{R}_+$ a *minimum spanning tree* has the minimum total weight $\sum_{e \in E} w(e)$. A weighted graph may have multiple minimum spanning trees. It can be efficiently computed in time $\mathcal{O}(|E| \log |V|)$ using Kruskal's [35] or Prim's [46] algorithm. The *geometric spanning tree* of a set of points $V \in \mathbb{R}^n$ is a minimum spanning tree of the complete graph on V where the weights are the Euclidean distances. As explained in the following section it can be computed in $\mathcal{O}(|v| \log |V|)$ time.

2.2 Geometry

Points in $\mathbb{R}^n$ are a very basic concept of geometry. They can be combined to form new points. One such combination is the line $\{at + b | t \in \mathbb{R}\}$ through two points a and b. The concept of linear interpolation between points is generalized by affine combinations.

Definition 2.1 (Affine combination, affine independence [see 11])**.** Let $P = \{p_0, \ldots, p_k\}$ be a set of points in $\mathbb{R}^n$. An *affine combination* $p = \sum_{i=0}^{k} w_i p_i$ is a combination of points p_i with weights w_i where $\sum_{i=0}^{k} w_i = 1$. P is *affinely independent* if no point $p_i \in P$ is an affine combination of $P \setminus \{p_i\}$.

This does not yet yield any notion of being "between" points. A way of conveying the intuitive meaning of this notion is convexity. *Convex combinations* are affine combinations where all weights $w_i > 0$. A point q is between other points $P = \{p_0, \ldots, p_k\}$ if it is a convex combination of the p_i, i.e. it is within the *convex hull* $\mathrm{conv}(P)$.

Definition 2.2 (Convexity, convex hull)**.** A set $P \subset \mathbb{R}^n$ is *convex* if and only if for every $a, b \in P$ the line $\{(1 - t)a + tb | \, t \in [0, 1]\} \subseteq P$. The *convex hull* $\mathrm{conv}(P) \subset \mathbb{R}^n$ of P is the smallest convex set containing P.

The convex hull of a set of a planar points $P \in \mathbb{R}^2$ can efficiently be computed in $\mathcal{O}(n \log n)$ time using Graham scan [25] or the monotone chain algorithm [3]. Both

algorithms operate similarly by first sorting the points and then iterating them in sorting order building the hull along the way. Monotone chain by Andrew [3] sorts the points along one of the Cartesian axes and then constructs two partial hulls, the first of which consists exclusively of right bends and the second only of left bends. Merging the two partial hulls yields the convex hull of P.

A simplex S is the convex hull of an affinely independent set of points which we call its *vertices* V. Simplices resemble a generalization of triangles and can be uniquely described by their extreme points, the vertices. The dimension k of a k-simplex is the dimension of the affine space spanned by its vertices and thus $k = |V| - 1$. A 0-simplex is thus a point, a 1-simplex a line and a 2-simplex a triangle. A *face* F of a simplex is the convex hull of a strict subset $F \subsetneq V$ of its vertices. In case of a tetrahedron the faces are the vertices, edges and triangular faces but not the volume.

A *triangulation* of a finite set of points $P \subset \mathbb{R}^n$ is a partitioning T of the convex hull $\text{conv}(P)$ into non-overlapping simplices. The vertices in T are the points in P. The vertices P together with the edges in the triangulation have a natural interpretation as graph. A triangulation in $\mathbb{R}^2$ is a planar graph.

Definition 2.3 (Triangulation of a point set ([see 11]))**.** Let $P \in \mathbb{R}^n$ be a finite set of points. A triangulation of P is a set T of simplices whose vertices are points in P with the following properties:

1. P is the set of vertices in T, i.e. $\bigcup_{t \in T} = P$
2. each distinct pair of simplices $t_1, t_2 \in T, t_1 \neq t_2$ either does not intersect or its intersection is a common face.
3. the union of all $t \in T$ is the convex hull $\text{conv}(P)$ of P

We call a set $P \subset \mathbb{R}^n$ degenerate if all points in P lie on a hyperplane of dimension $k < n$. In this case the triangulation T is degenerate. The simplices $t \in T$ are not n-simplices but have the lower dimension k. In two dimensions this is the case if all points are collinear.

A *Delaunay triangulation* is a triangulation in which the circumcircle of every triangle does not contain a point in the underlying point set in its interior. In general, there is not a unique Delaunay triangulation for a given set of points [13, p. 97]. Instead the *Delaunay subdivision* is a unique structure. Its cells are not necessarily triangular but can be further subdivided to obtain a Delaunay

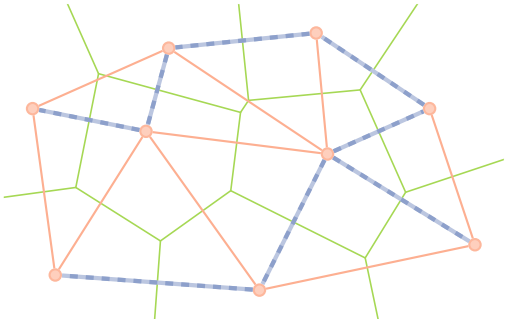


Figure 2.1: Relation of the Voronoi diagram, the Delaunay triangulation and the minimum spanning tree. Green edges are Voronoi edges, blue and red edges belong to the Delaunay triangulation. The thick blue edges are the minimum spanning tree, which is a subgraph of the Delaunay triangulation. Note how the Voronoi edges are the perpendicular bisectors of their corresponding Delaunay edges.

triangulation. The Delaunay subdivision can be characterized by the empty-sphere property:

Definition 2.4 (Delaunay subdivision, Delaunay cell [see 13, p. 97])**.** Let $P \subset \mathbb{R}^2$ be a finite point set and let $B \subset P$ be a subset of its elements. Then B is a *Delaunay cell* if and only if there exists a circle with all the points in B on its boundary and no other points in $P \setminus B$ either on the boundary or in the interior of C. The Delaunay subdivision is the union of all Delaunay cells.

Note that this definition covers a subdivision into lines and vertices. A degenerate planar point set where all vertices are collinear thus yields a simple path through the vertices as its Delaunay subdivision. We also consider point sets with multiple Delaunay triangulations degenerate.

The dual of the Delaunay triangulation is the *Voronoi diagram*. Its vertices are the circumcenters of the Delaunay cells. Each node $p \in P$ is the center of a Voronoi region which covers all points in $\mathbb{R}^n$ closer to p than to any point in P. The boundaries of the Voronoi regions are the perpendicular bisectors of their corresponding Delaunay edges. This can also be seen in fig. 2.1 which illustrates the relation between the Delaunay triangulation and Voronoi diagram. The Duality between Voronoi diagrams and Delaunay triangulations can be exploited to efficiently compute the Delaunay triangulation. There exist algorithms for computing planar Voronoi diagrams running in $\mathcal{O}(|V| \log |V|)$ time [21], which is also the best possible worst-case complexity for computing planar Delaunay triangulations. Other approaches, such as divide-and-conquer can achieve the same optimal run-time complexity and directly compute a Delaunay triangulation [11].

One important property of a Delaunay triangulation is related to minimum spanning trees. Every minimum spanning tree of a point set is a subgraph of the corresponding Delaunay subdivision [13, p. 102]. This is also true for the Delaunay triangulation. We can thus efficiently compute the minimum spanning tree of a planar point-set from its Delaunay triangulation using a spanning-tree algorithm for graphs. A planar Delaunay triangulation is a planar graph with edge count bounded by the Euler characteristic $|E| \leq 3\,|V| - 6$. The Minimum spanning tree of a planar point set can thus be computed with $\mathcal{O}(|V| \log |V|)$ time complexity.

2.2.1 Steiner Trees

The Steiner Tree problem is one of the classical problems of finding a minimal network and has been studied since the early 1800s [9]. While it is commonly stated on graphs, there exist a variety of geometric Steiner tree problems. We are mainly interested in the geometric Steiner tree problem in the (hyper-)plane using the Euclidean distance. A geometric or Euclidean Steiner tree is a minimal network $G = (T \cup S, E)$ connecting a set of *terminals* $T \in \mathbb{R}^2$ using additional Steiner nodes $S \in \mathbb{R}^2$. Despite the name, terminals do not need to be leaves and can have two or more incident edges.

Problem 2.5 (Euclidean Steiner Tree)
Inputs: Terminals $T \in \mathbb{R}^2$

Outputs: Steiner nodes $S \in \mathbb{R}^2$, Connections $E \subseteq [S \cup T]^2$

Properties: $G = (S \cup T, E)$ is a tree, $\sum_{e \in E} \|e\|$ is minimal

Garey, Graham, and Johnson [23] showed that the Euclidean Steiner Tree Problem is NP-hard. They proved NP-completeness when using a discretized Euclidean metric. The NP-completeness of the problem remains unclear since it involves irrational numbers.

Similar to a Steiner Tree a *Fermat point* p is a point that minimizes the euclidean distance to a set of nodes $V \in \mathbb{R}^n$, i.e.

$$\text{minimize} \sum_{v \in V} \|p - v\| \qquad \text{(Fermat Point)}$$

The original Fermat point problem was restricted to triangles $\triangle ABC$ where two cases can occur: either one of the corners has an angle of at least 120° or all angles are strictly less than 120°. In the first case the Fermat point coincides with the

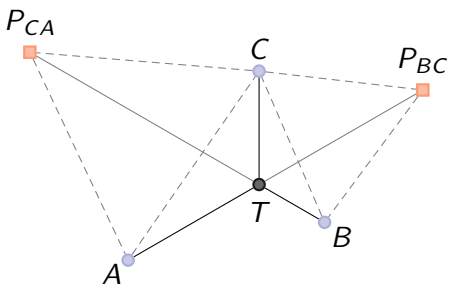


Figure 2.2: Construction of the Fermat-Torricelli point. We construct equilateral triangles on the sides with points P_{BC} and P_{CA} pointing outward. These points are then connected to the opposite terminals. The Fermat point T is located at the intersection of the connecting lines BP_{CA} and AP_{BC}.

obtuse vertex. The second case can be solved geometrically [9]. This is done by constructing an equilateral triangle on each side of the triangle with the additional corner pointing outwards as depicted in fig. 2.2. The Fermat point is located at the intersection of equilateral triangles' circumcircles. Alternatively each of the new vertices in the equilateral triangles can be connected to the opposite point in the original triangle $\triangle ABC$. The latter construction is the method illustrated in the figure. The resulting lines also intersect in the Fermat point.

In the case of three vertices, the Fermat point yields the optimal solution for the Steiner tree problem [9, 24]. Each Steiner point in a Steiner tree is located at the Fermat point of its neighbors.

The Fermat problem can be generalized using weighted distances resulting in the *Weber problem*. Each input point $v_1, \ldots, v_n$ has an associated weight w_i which is multiplied with the distance to the central node. The Weber point p is the point which minimizes the weighted distances, i.e.

$$\text{minimize} \sum_{i=1}^{n} w_i \left\| p - v_i \right\| . \tag{2.1}$$

If $w_i = 1$ for all i this is the same as the Fermat point problem. A numeric solution for the Weber problem was given by Miehle [40] in 1958 which extends the algorithm by Weiszfeld [52] for the Fermat problem with weighted distances. This algorithm was discussed in more detail by Kuhn and Kuenne [36]. In each iteration j performs the following step:

$$p_{j+1} = \left(\sum_{i=1}^{n} \frac{w_i \, p_j}{\left\| v_i - p_j \right\|} \right) \Bigg/ \left(\sum_{i=1}^{n} \frac{w_i}{\left\| v_i - p_j \right\|} \right) . \tag{2.2}$$

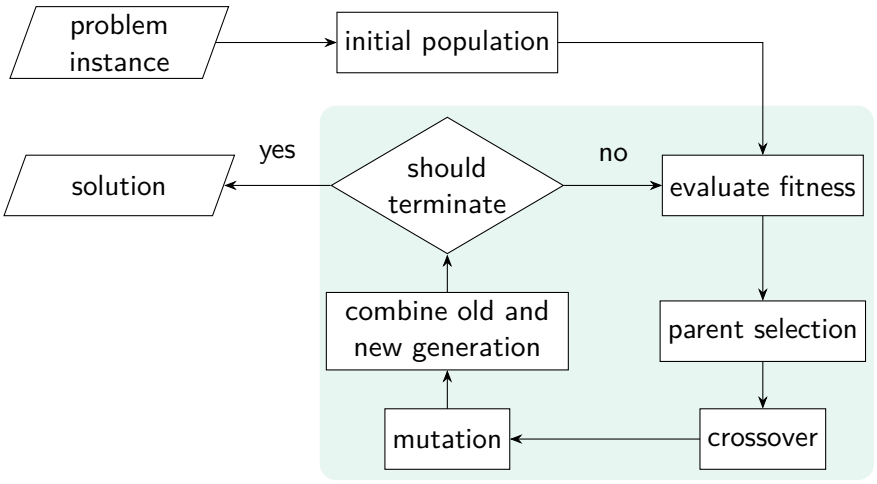


Figure 2.3: Steps of a typical genetic algorithm. The highlighted nodes form the evolutionary loop generating and processing each generation. (based on [22, p. 37])

2.3 Genetic Algorithms

This description of genetic algorithms is based on "Genetic algorithms" by Sivanandam and Deepa [50]. Genetic algorithms are a metaheuristic method inspired by natural evolution. They mimic different aspects of natural evolution to find good solutions to a problem. The solution quality in an optimization problem is measured in terms of its *objective function*. Usually such algorithms are based around a population whose individuals have different genomes, which can be recombined to produce offspring. This is an iterative process. In each iteration a new *generation* of individuals is built from the old population. The whole process is illustrated in fig. 2.3.

The genetic variety within the population helps to escape local minima. To maintain a diverse gene pool and explore new solutions, new individuals are subject to mutation.

In the context of genetic algorithms, an *individual* is a candidate solution to the optimization problem. Each individual has a *fitness* based on the objective function and is represented by its *genome*. The genome is a collection of *genes* which encode features present in the individuals. This representation can be freely chosen to suit the problem. A common representation is a string of individual genes, e.g. a vector.

Each iteration consists of several steps. In the *selection* phase the parent tuples for the next generation are selected. The selection is usually based on the fitness of the individuals, which is determined by a fitness function. Using a fitness function to select individuals creates a selection pressure that removes less optimal genes from the gene pool.

The parents' genomes are then recombined by a *crossover* operator to produce the offspring genomes. New genomes are also subject to random *mutation* which is implemented in a mutation operator. Finally the parents and offspring are mixed by a *reinsertion* operator to form the population of the next generation.

Selection and reinsertion can be implemented independently of genome encoding. They do, however, require an ordering according to the objective value. They exert the selective pressure which favors individuals with higher objective function values. This is often implemented by computing a fitness value which is higher for individuals with better objective values. In a minimization problem this means that the objective value cannot be used as fitness value without applying a transformation. One way of converting the objective function to a fitness function is rank based fitness assignment [4]. In rank-based fitness the individuals are sorted according to the objective function. The fitness value is then calculated only from the individual's rank.

One common selection method which does not rely on an increasing fitness function is *tournament selection*. In each round n individuals are selected uniformly at random from the population and the best one is selected as a parent. The individuals are not removed from the population but can be drawn again in following rounds. In tournament selection without *replacement* the selected parents are not returned and cannot participate in the following rounds. Tournament selection can work with a raw objective function because the absolute value of the objective function is not relevant. Each round is decided by the highest ranking individual which only depends on the ordering of the individuals. Using a separate rank-based fitness assignment is thus not necessary with tournament selection. The number of rounds is determined by the desired number of parents. A tournament of size n is also called n-tournament. A 1-tournament is equivalent to *random selection* where each individual is equally likely to be chosen as parent. If an increasing fitness function is available one can use *roulette wheel selection* where individuals are chosen with a probability proportional to their finite positive fitness value and *truncation selection* where parents are chosen deterministically based on their fitness rank.

To prevent loss of the best individuals when building the next generation, reinsertion schemes combine the parents with the offspring. A straightforward way is *uniform reinsertion* where parents are replaced uniformly at random. This method is not based on the individuals' fitness values and may remove the best performing

individuals from the population. To keep them in the gene pool, an *elitist reinsertion* scheme can be employed. Here, a specified number of best performing individuals of the parent generation is kept while the others are replaced with the offspring. There is no check to ensure parents are replaced by better offspring and thus the average fitness of the population does not necessarily increase.

2.4 Electrical foundations

In electrical power installations we have some constraints set by physics and electrical engineering standards. Different cables are not equally well suited for power transfer because of their internal resistance and current ratings. For a detailed overview of the electrical engineering involved in the design of power grids we refer the reader to Schwab [49]. In this section we give an overview of the relevant concepts of electricity and power grids.

Current is measured in Ampere A, voltage in Volt V and resistance in Ohm Ω. The relation of electric current I, voltage U and resistance R is given by Ohm's law

$$U = R \cdot I \tag{2.3}$$

which applies to most conductors, e.g. metals.

Electric power is the product UI of voltage and current and is measured in Watt W. In an ohmic resistor electric power is dissipated as heat. The losses of power depend on the voltage U across the resistor and the current through the resistor:

$$P = UI \tag{2.4}$$

$$= RI^2. \tag{2.5}$$

The loss of power as heat is undesirable as it is not delivered to consumers but has to be supplied by generators. If the grid suffers excessive power losses its operation may become uneconomical.

Heating of the wires also leads to structural problems. As cooling capabilities depend on many details, cable manufacturers only set a thermal limit which then has to be translated to a current limit. This current rating or *ampacity* is defined by electrical engineering standards and depends among others on the material, the cross section, cable type and installation method. In Germany the thermal current rating is defined by DIN VDE 0298-4 and DIN VDE 0276-626. The ampacity is usually also given in the cable data sheets.

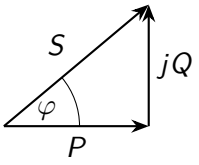
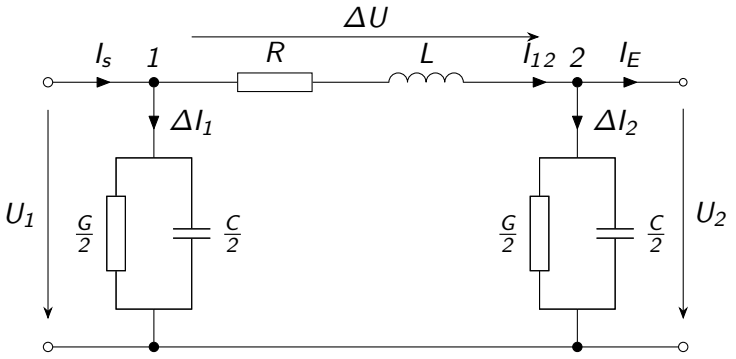


Figure 2.4: Phasor diagram of active power P, reactive power Q and apparent power S offset by the phase angle φ.

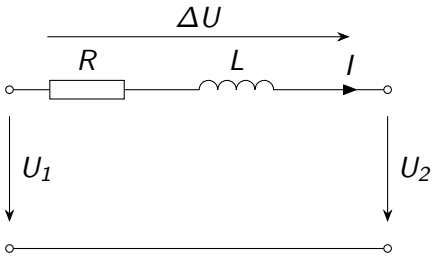
In an alternating current circuit, current and voltage have the same frequency but their phases may be offset by an angle φ [49, pp. 744–748]. An electric load that is purely *resistive*, i.e. behaves just like an ohmic resistor, causes no phase offset. All power flows as *active power* P from the power source to the load. In a purely *reactive* load current and voltage are offset by $\varphi = 90°$ which means that half the time power enters the load while the other half it leaves the load and on average no power is transferred at all between power source and load. Only *reactive power* Q flows between power source and load. Real loads have both active and reactive properties. To account for that AC power is usually given as a complex number $S = P + jQ$ where the real axis corresponds to active power and the imaginary axis corresponds to reactive power. Problems involving complex power can be solved graphically using phasor diagrams such as in fig. 2.4. The magnitude of complex power $|S|$ is called apparent power. It is the value obtained when multiplying the average magnitudes of power and voltage.

Electric grids need to be designed in terms of apparent power because both active and reactive power are transferred through the power lines. With more reactive power higher currents are needed to supply the same active power. Equipment and cables need to withstand the higher currents and power losses increase. A high amount of reactive power is thus undesirable. Some degree of reactive power is, however, needed to operate the grid. Some electric loads require reactive power and it is used to control voltage and frequency in larger grids [49, p. 525]. If a consumer requires active power P and has a load factor of $\cos\varphi$, the connection needs to be designed for an apparent power of $S = P/\cos\varphi$.

Alternating current changes its amplitude over time. Electricity does not travel instantaneously and thus the voltage in an AC cable is not the same everywhere at every instant. If the difference in voltage throughout the cable is negligible we call it *electrically short*. Whether a cable is electrically short or long thus depends on the length l of the cable and the wavelength λ of the voltage. In electrical engineering a cable is considered electrically short if the voltage does not differ by more than 0.5%. The cable length may thus not exceed $\frac{\lambda}{60}$ [49, pp. 346–347]. In a grid with 50Hz and overhead power lines this length is 100km which is far beyond the scale



(a) Equivalent circuit of an electrically short cable [49, p. 362]



(b) Simplified short cable with negligible insulation losses [49, p. 364]

Figure 2.5: Equivalent circuit diagrams of a transmission line. The circuit in (a) models most electrical properties of the transmission line. If insulation losses G and cable capacity C are low they can be neglected leaving only resistance R and inductance L as in (b).

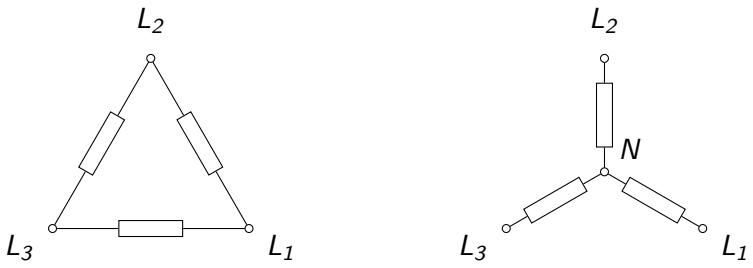


Figure 2.6: Two types of symmetric load connections in a three-phase AC setup. In Delta configuration on the left loads are installed between the phases L_i. In Y-configuration the loads are connected to the neutral N.

of our installations. Our model is thus not affected by effects of wave propagation speed.

Figure 2.5a shows the equivalent circuit of an electrically short transmission line. The electrical resistance R of a wire depends mainly on its material, cross-section and length. For a given material its *resistivity* describes the materials resistance depending on its cross-section and length. Resistivity is measured in $\Omega\,\mathrm{m}$ or equivalently, $\Omega\mathrm{m}^2/\mathrm{m}$. As we know material and cross section for a given type of cable, we can compute its *line resistance* ρ in $\Omega\,\mathrm{km}^{-1}$ depending only on the cable length. When alternating current is used, the cable *inductance* L becomes important, as it adds a reactive component to the cable losses. With conductors close to each other, the cable also has a small electric capacity C. Insulation is not perfect and therefore a small current flows between the conductors, limited by the high insulation resistance G. If G and C are small, we can ignore them resulting in the simplified circuit in fig. 2.5b.

The voltage ΔU across the cable is the voltage drop experienced between the voltage U_1 at the beginning and the voltage U_2 at the end of the cable. It is caused by the cable resistance according to Ohm's law: $\Delta U = RI$. This *voltage drop* is undesirable as it reduces the usable voltage at the consumer end and is thus limited by electrical engineering norms. For a cable of length l, a line resistance of ρ and a current I through the cable the voltage drop is $RI = \rho \cdot l \cdot I$ according to Ohm's law.

The standard to define the quality of service requirements in German and European grids is DIN EN 50160. It defines the maximum deviation from supply voltage at a consumer connection. According to this standard, the voltage should be within $\pm 10\%$ of the regular supply voltage under normal circumstances, i.e. 95% of all times, and may never fall by more than 15%. Voltage deviation is measured over an interval of 10 minutes.

Most AC power systems rely on three-phase transmission which comes with several advantages. By combining three symmetrically loaded phases with a phase offset of 60°, the transferred power is constant in time and does not alternate. Relative to a common reference, voltage and current always add to zero. This is often connected as a fourth neutral wire, simply referred to as the *neutral*. Using a neutral allows mixed use of single-phase and three-phase appliances. Three-phase loads can also be connected in two ways which are depicted in fig. 2.6. They can be placed between the phases or connected to the neutral. *Line voltage* U_r between the phases is $\sqrt{3}$ times the *phase voltage* towards the neutral [49, p. 231]. To avoid ambiguity both values are usually given to describe a grid. In most of Europe for example, the phase voltage is 230 V and the line voltage is 400 V.

If the load is symmetric, i.e. all three phases carry the same load, nominal currents in all three phases are equal. The transmitted active power P of the line is the sum of the powers in each phase [49, p. 338]. It can be calculated from the phase voltage U_p and phase current I_p by

$$\begin{aligned} P &= 3\,U_p\,I_p\,\cos\varphi \\ &= 3\,\frac{U_r}{\sqrt{3}}\,I_p\cos\varphi \\ &= \sqrt{3}\,U_r\,I_p\,\cos\varphi. \end{aligned} \tag{2.6}$$

3 Related Work

Planning a microgrid is a network optimization problem. Minimal tree networks have been studied for centuries in the form of spanning trees and Steiner trees. There are several variants of the Euclidean Steiner Tree Problem (problem 2.5). The Node Weighted Geometric Steiner Tree Problem introduces a cost for adding Steiner nodes and a penalty for not connecting terminals. As a generalization of the Euclidean Steiner Tree Problem it is also NP-hard but polynomial time approximation schemes exist [47]. The Node Weighted Geometric Steiner Tree problem is related to the price-collecting Steiner tree problem on graphs where not connecting a terminal results in a penalty.

Problem 3.1 (Node Weighted Geometric Steiner Tree)
Inputs: Terminals $T \subset \mathbb{R}^d$, penalty function $\pi : T \to \mathbb{R}_+$, Steiner node cost $c_s \in \mathbb{R}_+$

Output: connected terminals $V \subseteq T$, Steiner nodes $S \subset \mathbb{R}^d$, edges E

Objective: minimizes

$$\sum_{e \in E} \|e\| + \sum_{v \in T \setminus V} \pi(v) + |S|\, c_s \tag{3.1}$$

The cost of edges in the plane is not necessarily constant. Some areas might not be accessible or only at a higher cost. The Euclidean Steiner Tree Problem with Obstacles adds polygonal obstacles to the Euclidean Steiner Tree Problem. The cost of an edge through an obstacle o increases by a penalty c_o for each distance unit through the obstacle dist_o. An exact algorithm for the problem with solid obstacles has been proposed by Zachariasen and Winter [53]. Garrote et al. [24] proposed a heuristic that includes soft obstacles.

Problem 3.2 (Geometric Steiner Tree with Obstacles)
Inputs: Terminals $T \subset \mathbb{R}^2$, polygonal obstacles O with associated penalties c_o

Outputs: Steiner nodes $S \in \mathbb{R}^2$, Connections $E \subseteq [S \cup T]^2$

Properties: $G = (S \cup T, E)$ is a tree, $\sum_{e \in E} \left(\|e\| + \sum_{o \in O} c_o \operatorname{dist}_o(e) \right)$ is minimal

One approach to minimal networks combined with flows is the Capacitated Spanning Tree Problem. This problem consists of a set of terminal nodes, a designated sink node and links between the nodes. Each link has a capacity that must not be exceeded and the terminals act as unit sources. A solution is a shortest spanning tree satisfying the capacity constraint. This problem has been shown to be NP-complete by Papadimitriou [43].

Problem 3.3 (Capacitated Spanning Tree)
Inputs: Sources S, sink t, links $E \subseteq (S + t)^2$, capacities $c : E \to \mathbb{N}$, link lengths $l : E \to \mathbb{R}$

Output: Spanning tree T, flow $f : V^2 \to \mathbb{Z}$ where $V = S + t$

Objective: f is a flow that satisfies $f(x, y) \leq c(x, y)$ and $\sum_{e \in T} l(e)$ is minimal

A combination of Steiner trees with flows in electric power grids has been proposed to solve substation placement and wind farm cabling problems. In these instances edges represent cables of different types suited for varying power requirements. The Wind Farm Cabling problem has the objective to minimize the cable cost in offshore and onshore wind farms. In its basic form it can be stated as follows:

Problem 3.4 (Wind Farm Cabling)
Input: Wind Turbines $t \in V_T$ producing power P_t, substations V_0, cable types L with cost c_l and a maximum capacity, possible connections $A \in [V]^2$

Output: Spanning Forest $E \subseteq A$ connecting the turbines to the substations with cables $l_e \in L$ assigned to the edges e inE such that the combined power flow from the turbines to the substations does not exceed the cable capacity

Objective: minimize $\sum_{e \in E} \|e\| \, c_{l_e}$

Fagerfjäll [19] computes not only the cable layout but also wind turbine positions using a mixed integer program. This approach is based on a discrete grid where each vertex has possible edges to vertices in a certain radius. He makes limited use of Steiner trees by increasing the grid density and allowing to connect to vertices without a wind turbine. The inclusion of Steiner points in offshore wind farm layouts has been dismissed by Pillai et al. [44]. They argue that cable junctions

outside turbines and substations are not feasible in an offshore environment and add additional computational complexity to the problem. Dutta and Overbye [18] compute cable layouts that avoid restricted areas using a minimum spanning tree approach. They redirect edge passing through obstacles around the convex hull of the obstacle and the edge endpoints.

The layouts considered by Dutta and Overbye [18], Fagerfjäll [19], and Pillai et al. [44] are tree layouts connecting to central substations. However, Gritzbach, Wagner, and Wolf [26] note that, depending on the available cable types, the optimal layout is not necessarily a tree.

3.1 Genetic Algorithms for Steiner Problems

Several authors have used genetic algorithms to solve Steiner tree problems. One very common approach, used among others by Costa et al. [12] and Jesus, Jesus, and Márquez [30], is to determine the optimum Steiner point positions with an evolutionary algorithm and then compute a minimum spanning tree on the points. This method works on planar Steiner trees as well as in higher dimensions [12].

Several of these approaches use local optimization to move the Steiner points to their optimal positions. Costa et al. [12] and Jesus, Jesus, and Márquez [30] geometrically determine the Fermat points to move Steiner points to better positions. Both compute a minimum spanning tree on the points to obtain the graph topology. Steiner positions can be encoded in different ways. Instead of euclidean coordinates, Jesus, Jesus, and Márquez [30] encode Steiner point positions as convex combinations of the terminals. This ensures that Steiner points are always within the convex hull of the terminals.

Barreiros [6] uses a different approach: Steiner points are moved randomly or towards their neighbors. They also use a different representation of the Graph topology. Instead of storing only Steiner positions and computing the topology based on these positions, they subdivide the nodes and connect the subsets using comb graphs. Such a comb is defined as a path through a number of Steiner nodes, each of which is connected to precisely one terminal. The combs are optimized individually and combined in a separate step to form a Steiner tree on the terminals.

An explicit encoding of the graph is used in other spanning tree problems. Moharam and Morsy [41] use genetic algorithms to find a diameter constrained spanning tree on a given graph. Their work includes tree based crossover and mutation operators. These operators are designed to always yield valid trees.

An genetic algorithm for the geometric Steiner tree problem with hard and soft obstacles has been presented by Rosenberg et al. [48]. Like Costa et al. [12] and Jesus, Jesus, and Márquez [30], they compute a minimum spanning tree on the terminals and Steiner nodes including a subset of the obstacle vertices. They use a mutation operator to add obstacle corners to the tree and another operator to insert new Steiner nodes when an angle of less than 120° is found. Their crossover operator splits the parents along a line and recombines the left and right sides according to the line to form the offspring. Frommer and Golden [22] used a genetic algorithm to approximate the Steiner tree with soft weights from a continuous weight function. They solve the problem on a graph derived from a hexagonal grid on the function.

4 Problem Definition

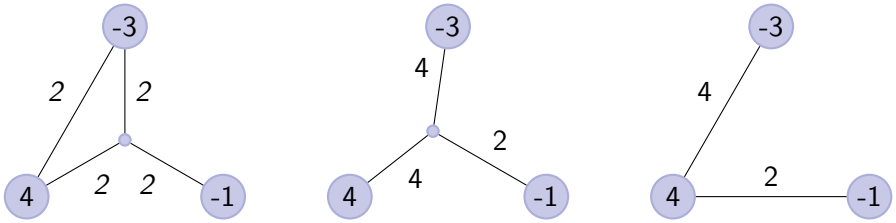
In this chapter we formally introduce the microgrid planning problem. To construct an electrical grid we not only have to determine which nodes to connect but also where to place distribution nodes and what cables to use. Additional constraints are imposed by standards and providing a certain quality of service. This leaves us with two major subproblems of microgrid planning. The first is to find a suitable topology and the second to estimate the cost of an electrical network with that topology. Network cost is determined by the cables and other equipment used. Our method of estimating the network cost is largely based on work by Kraft [34].

The network connects terminals $T = T_g \cup T_c$ which consist of generators T_g and consumers T_c. We allow additional distribution nodes $S = V \setminus T$ to be placed arbitrarily in the plane. We also refer to them as Steiner nodes because the concept of auxiliary nodes to shorten total network length can also be found in the Steiner tree problem. The nodes V are connected using cable connections $E \subseteq [V]^2$. The placement of the Steiner S nodes and the selection of connections E are parts of the problem output. All nodes have a maximum demand $d : V \to \mathbb{R}$. Consumers draw power from the grid, thus all consumers $t \in T_c$ have positive demand $d(t) > 0$. Generators $t \in T_g$ have negative demand $d(t) < 0$ and supply power to the grid. Steiner nodes $t \in V \setminus T$ have a neutral demand $d(t) = 0$. Terminals and Steiner nodes have a position $\text{pos} : T \to \mathbb{R}^2$ in the plane, which is an input in case of Terminals and an output in case of Steiner nodes. An edge $e = uv \in E$ has length $\|e\| = \|\text{pos}(u) - \text{pos}(v)\|$ depending on the positions of its endpoints. Terminals with their positions and demands are part of the input to the planning process. We assume that the connection equipment for terminals has a fixed cost and can therefore be neglected in the optimization.

In this chapter we will use different variables to describe the input and output quantities and other properties of the cost model. To give the reader an overview of the usage of these symbols, we provide the nomenclature in table 4.1.

Table 4.1: Summary of all symbols used in the problem description

	Symbol	Description
Input	c_s	cost of adding a Distribution node
	c_b	cost of a new branch
	c_i	power dependent distribution node cost per kW
	c_p	cost of placing a pole
	s_p	span length, i.e. the maximum distance between two poles
	L	set of available cable types
	I_l	ampacity of cable type l
	c_l	cost of cable type l per meter
	r_l	line resistance of cable type l in $\Omega\,\mathrm{km}^{-1}$
	t_{loss}	power loss tolerance
	t_{drop}	voltage drop tolerance
	U	grid design voltage
	T	terminal nodes, i.e. all generators and consumers: $T = T_g \cup T_c$
	T_c	consumer terminals
	T_g	generator terminals
	$\text{pos}(t)$	terminal positions in the plane
	d	demand and supply of the terminals in kW
	d^+	consumption
	d^-	generation
Output	S	Steiner nodes
	E	undirected connections or edges between the nodes
	l_e	cable type of edge e
	m_e	cable multiplicity of edge e
	P_e	maximum power flow through edge e
	I_e	maximum current in edge e
	C_{poles}	total cost of all poles
	C_{dist}	total cost of all distribution nodes
	C_{cable}	combined cost of all cables
	C	total network construction cost
Other	V	entirety of nodes in the network: $V = T \cup S$
	G	power grid graph $G = (V, E)$
	deg	vertex degree, i.e. number of edges at a vertex v



(a) Cost: ≈ 5.46 **(b)** Cost: ≈ 5.61 **(c)** Cost: 6.0

Figure 4.1: Non-tree optimal cable layout. Uses $2 \cdot \lceil \frac{\text{cap}}{2} \rceil$ as unit cable cost. Nodes are labeled with their supply, edges are labeled with their unit cable cost. (b) is the optimum tree layout using a Steiner node while (c) is the best spanning tree layout.

4.1 Network Topology

Electrical grids can be realized as *mesh* or as *radial* layouts. Mesh topologies are more common in high-voltage grids while low voltage grids use radial layouts [49, p. 395]. They are less complex than mesh layouts and don't require power flow regulation as only one path is available. They are, however, also the least reliable grid topology as no backup path exists in case of a cable failure. When interpreted as a graph, a radial network is a tree. To increase reliability, such networks are often designed as *open loops* with one section intentionally switched off. An additional cable is installed connecting the ends of two branches. This allows the operator to redirect the power for maintenance work and in case of a cable failure.

We aim for a microgrid with little complexity and low cost. This can be realized best by a tree topology. We don't require the redundancy and increased reliability provided by an open loop.

A tree topology has the additional advantage that power flows are easy to control as only one path exists. Power flow in mesh networks is determined by reactive power which therefore needs to be controlled to avoid overloading single power lines. Power flow in mesh networks is more difficult to predict and handle [49, p. 627]. In transmission networks, this requires the use of expensive specialized equipment and careful planning. Generally, each loop in a mesh adds to the complexity of planning and operation [49, pp. 394–396].

Note that a tree topology is not an electrical necessity but a planning decision. The cost of the cables in a mesh network may even be cheaper than in a radial network. An example of such a network is given in fig. 4.1a. In the example, it is less expensive to place an additional direct low capacity cable than to increase the capacity on the path through the distribution node.

All in all a tree layout still seems to be the best option for our purposes. Using a radial network also reduces the algorithmic complexity of some computations, which will be important in the following sections.

4.2 Electric Constraints

We evaluate network topologies by their cost of construction. Each edge needs to be assigned cables that can carry the required current without overheating and at the same time limit power loss and voltage drop in the network. The cable types L need to be supplied by the user. A cable $l \in L$ is represented by a tuple $l = (c_l, I_l, r_l) \in \mathbb{N}^3$ of cost per meter, ampacity and line resistance. Line resistance is usually measured in $\Omega\,\mathrm{km}^{-1}$. We allow multiple cables of the same type l_e to be assigned to a cable. The required number m_e of cables is determined by the ampacity of the cable and its resistance. We limit power losses and voltage drop by computing a lower bound for the maximum resistance. To determine the best cable for an edge $e \in E$ we have to compute the maximum resistance and the maximum current though that edge. In this section we extend the methods used by Kraft [34] to compute the current through all edges as well as estimate their maximum resistances. After that we can compute an assignment of cables l_e with multiplicity m_e to each edge $e \in E$. The total cost of such a cable assignment l_e is

$$C_{\text{cable}} = \sum_{e \in E} m_e c_{l_e} \|e\| . \tag{4.1}$$

4.2.1 Ampacity

If the current trough a cable is too high, the cable overheats and can get damaged. To avoid this we have to compute the minimum ampacity I_e of each edge $e \in E$. A simple conservative approach is to design the network with peak loads for each consumer. This is common practice when designing electrical grids [27, 39]. In an off-grid network with distributed generation as considered in this work, the combined peak demand can exceed the generation capacity. In this case we limit the required cable capacity to the maximum generation. This method is also used by Kraft [34].

Loads on a cable are measured in terms of power which is independent of the voltage level. We want to accommodate every possible flow from generators to consumers in the grid. This approach is unrealistic in larger installations but makes sense in

our small scale network. Due to the limited power generation and small scale the electricity for a consumer may be supplied by the furthest generators.

We denote by P_e the value of the maximum power flow through a connection. The power flow depends on the consumption and generation in the grid. It can be characterized as follows: let $X \subsetneq V, X \neq \emptyset$ be a set of vertices, let $X' = V \setminus X$ and let $E(X, X')$ be the edges in the cut between X and all other vertices X'. We denote by $d^-(X) = \sum_{t \in T_g \cap X} |d(t)|$ the total generation in X and similarly by $d^+(X) = \sum_{t \in T_c \cap X} d(t)$ the total consumption in X. Then the following must hold for the total capacity of the cut $E(X, X')$

$$\sum_{e \in E(X,X')} P_e \geq \min \left\{ d^+(X), d^-(X') \right\} \tag{4.2}$$

where P_e is the maximum power on edge e. Given the voltage U we can solve eq. (2.6) for the maximum current I_e. This current is the minimum ampacity of a cable on that edge. A cable l_e for edge e must fulfill the ampacity constraint

$$I_{l_e} \geq I_e = \frac{P_e}{\sqrt{3}\, U \cos \varphi}. \tag{4.3}$$

We require this for all sets $X \subsetneq V, X \neq \emptyset$. By exchanging X and X' the capacity must also exceed $\min \{d^+(X'), d^-(X)\}$. Note that this formulation requires all terminals to be connected if neither $T_g = \emptyset$ nor $T_c = \emptyset$. As having a separate network component consisting exclusively of Steiner nodes makes no sense, we require the entire network to be connected.

We construct a tree network, therefore every cut consists of a single edge. This yields a straightforward computation of the maximum power P_e through each edge. Using a depth-first search we can compute the generation and consumption. The full algorithm can be found in algorithm 1. We use a post-order search to sum the generation and consumption in a subtree and subtract this sum from the total generation and consumption to find the flow through each edge. The use of a single depth-first search results in linear running time. Our approach differs from Kraft [34] who computed all consumer-producer paths separately.

Algorithm 1: Recursive computation of the edge capacities

```
1  function assign_edge_cap(network, start, visited)
2      (generation, consumption) = (0, 0);
3      if !visited[start] then
4          visited[start] = true;
5          for each (edge, neighbour) in network.neighbours(start) do
6              below = assign_edge_cap (network, neighbour, visited) +
                 (network.generation(neighbour),
                  network.consumption(neighbour)) ;
7              above = (network.total_generation, network.total_consumption)
                 - below;
8              up = min (first(below), second(above)) ;
9              down = min (second(below), first(above)) ;
10             network.capacity(edge) = max (up, down);
11             (generation, consumption) += below;
12     return (generation, consumption)
```

4.2.2 Resistance

We want to use cables that limit losses of voltage and power. In a three-phase cable the voltage drop U_d and power loss P_l can be calculated using eqs. (2.3) and (2.6):

$$U_d = RI = R \cdot \frac{P}{\sqrt{3}\, U \cos\varphi} \tag{4.4}$$

$$P_l = 3\, U_d I = 3R\, I^2 = 3R \left(\frac{P}{\sqrt{3}\, U \cos\varphi} \right)^2 \tag{4.5}$$

where

U_d	absolute voltage drop
P_l	absolute power loss
P	transmitted active power
U	line voltage
R	resistance of a single phase
I	current in a single phase
$\cos\varphi$	load factor.

Power losses are additive throughout the whole network. Thus, we only need to sum the power losses on all edges to obtain the total power loss. Power loss is an economical factor as the additional losses need to be generated elsewhere. What

amount of power loss is bearable depends on the cable installation cost on one the one hand and electricity generation cost on the other hand. Choosing an economically reasonable value is beyond the scope of this work. We therefore leave the maximum tolerated power loss percentage as an input parameter t_{loss}. This leads to the power loss constraint

$$\sum_{e \in E} P_l(e) \leq t_{\text{loss}} \min \left\{ d^+(T), d^-(T) \right\} \tag{4.6}$$

where $d^+(T)$ and $d^-(T)$ are the total generation and demand as defined above and $P_l(e)$ is the power loss on a single edge $e \in E$ calculated as above. Please note that this is not entirely realistic because a higher resistance also causes the voltage to drop which reduces the power losses in other cables. We do not consider cable losses in other cables when computing the power flowing through a cable or to determine cable ampacity.

We want to limit the voltage drop on each generator-consumer path to ensure that consumer voltage does not drop too far below the grid voltage U. Let $U_d(e)$ be the absolute voltage drop on a single edge which always uses one specific type of cable. Voltage drop, too, is additive on a path $p = (v_0, ..., v_l)$, i.e. $U_d(p) = \sum_{i=1}^{l} U_d(v_{i-1}v_i)$. The tolerated voltage drop t_{drop} is the fraction of the nominal voltage by which the voltage at a consumer may drop. According to DIN EN 50160 the voltage at a consumer should not fall by more than $t_{\text{drop}} = 10\%$. We thus have to assign cables that ensure that for each generator $g \in T_g$ and each consumer $c \in T_c$ the voltage on the path $(g = v_1, ..., v_n = c)$ does not drop by more than 10% from the nominal grid voltage U, i.e.

$$U_d(g = v_1, \ldots, v_n = c) \leq t_{\text{loss}} U. \tag{4.7}$$

Problem 4.1 (Cable assignment)
Input: graph $G = (V, E)$, cable types L, demands $d : V \to \mathbb{R}$, grid voltage U, voltage drop factor t_{drop}, power loss factor t_{loss}

Output: cable l_e with multiplicity m_e for each edge $e \in E$

Objective: minimize $\sum_{e \in E} m_e c_{l_e} \|e\|$

Properties:

- voltage drop U_d always below $t_{\text{loss}} U$
- power loss below t_{loss}

We do not know the algorithmic complexity of finding an optimum cable assignment for a given network topology under the voltage drop and power loss constraint. We neither prove NP-hardness nor provide a polynomial algorithm. Nevertheless, we assume finding the optimum cable assignment is NP-complete even if the network is a path. It is related to the weight constrained shortest path problem which is NP-complete, but more general than cable assignment. We could not find an obvious reduction of weight constrained shortest path on cable assignment. We also failed to find a reduction for other similar problem such as knapsack and bin packing. Instead, we present a conservative linear-time heuristic to compute a cable assignment that meets these requirements. This heuristic always yields a valid cable assignment albeit not an optimal one.

Kraft [34] uses a different estimate restricting power loss and voltage drop on each single cable section and not considering the whole network. This approach favors short edges instead of avoiding longs paths and does not yield a conservative estimate for the losses. We found that this leads to distribution nodes being inserted in the middle of long edges connected to a single low capacity terminal which increases the network length and creates long paths.

We use the longest generator-consumer path $P_{\max}(e)$ through an edge e to find a lower bound for the acceptable voltage drop on that edge. We distribute the voltage drop along each such path among the edges of the path by their percentage of the path length. Denote by $\|p\| = \sum_{i=1}^{n-1} \|v_i v_{i+1}\|$ the length of a path $p = (v_1, \ldots, v_n)$. The maximum voltage drop on each edge e is then $U_d(e) \leq \frac{\|e\|}{\|P_{\max}(e)\|} t_{\text{drop}} U$. Then for every path from a generator g to a consumer c

$$U_d(g = v_1, \ldots, v_n = c) \leq \sum_{i=1}^{n} \frac{\|e\|}{\|P_{\max}(e)\|} \cdot t_{\text{drop}} \cdot U \tag{4.8}$$

$$\leq \sum_{i=1}^{n} \frac{\|e\|}{\|(v_1, \ldots, v_n)\|} \cdot t_{\text{drop}} \cdot U \tag{4.9}$$

$$\leq t_{\text{drop}} \cdot U \cdot \sum_{i=1}^{n} \frac{\|e\|}{\|(v_1, \ldots, v_n)\|} \tag{4.10}$$

$$= t_{\text{drop}} \cdot U. \tag{4.11}$$

The voltage drop constraint is thus satisfied. We solve eq. (4.4) for the resistance R and obtain the maximum resistance of the cable e

$$R_d(e) \leq \frac{U_d(e)}{I} = \sqrt{3} \frac{U_d(e)\, U \cos\phi}{P_e} \tag{4.12}$$

$$= \sqrt{3} \frac{\|e\|}{\|P_{\max}(e)\|} t_{\text{drop}} \frac{U^2 \cos\varphi}{P_e}. \tag{4.13}$$

We compute the longest paths through all edges in two passes. The pseudocode can be found in line 2. We use two depth-first search passes to find the longest path lengths. Both passes start at the same arbitrary node. In the first pass we perform a post-order depth-first search to find the distance to the furthest generator and consumer along each edge directed away from the start node. In the second pass we combine the forward distances with a pre-order depth-first search.

Power loss is measured throughout the whole network. We restrict the total power loss in the network in terms of the total network power $P_{\text{total}} = \min\{d^+(T), d^-(T)\}$. We want to assign cables l_e to the edges in such a way that $t_{\text{loss}} P_{\text{total}} \leq \sum_{e \in E} P_l(e)$. Each edge's power loss $P_l(e)$ depends on its maximum current and its resistance which in turn depends on its length $\|e\|$. We therefore estimate the power loss of each edge e as

$$P_l(e) \leq \frac{\|e\| P_e}{\sum_{g \in E} P_g \|g\|} t_{\text{loss}} P_{\text{total}}. \tag{4.14}$$

Together with eq. (4.5) we can solve for the maximum resistance R_l of e

$$R_l(e) \leq \frac{\|e\|}{\sum_{g \in E} P_g \|g\|} t_{\text{loss}} P_{\text{total}} \left(\frac{U \cos \phi}{P_e} \right)^2 \tag{4.15}$$

4.2.3 Assigning Cables

We have a selection of cable types $L = l_1, \ldots, l_n$ which can be used on each edge. To increase the capacity we allow more than one cable on an edge installed in parallel. Unfortunately, ignoring the resistance, the selection of the cheapest cable bundle to meet the ampacity constraint is already NP-complete. The corresponding decision problem whether a given ampacity I can be achieved without exceeding cost C

$$\exists x \in \mathbb{N}^n : \sum_{i=1}^{|E|} x_i c_{l_i} \leq C \wedge \sum_{i=1}^{|E|} x_i I_{l_i} \geq I \tag{4.16}$$

is the same as the decision problem of knapsack [32]. Selecting a cheapest cable bundle for a given ampacity is thus NP-complete.

In real cables ampacities do not necessarily add up when different cables are used. Due to different resistances the power is not split evenly between the cables by their ampacity and thus the bundle ampacity may be lower than the sum of ampacities. To simplify the selection and avoid the NP-complete bundle selection we only create bundles of a single cable type. While cost and ampacity increase with a higher cable multiplicity, the resistance decreases. A cable bundle of m times cable l has a cost of mc_l, an ampacity of mI_l and a line resistance of r_l/m.

Algorithm 2: The procedure `assign_longest_paths` assigns to each edge in a tree the longest path through that edge. The recursive helper functions `forward_sweep` and `collect_totals` perform the actual computation. We use the element-wise maximum max.

```
1  function forward_sweep(graph, start, visited)
2      if !visited[start] then
3          visited[start] = true;
4          c = if graph[start].is_consumer then 0 else −∞;
5          g = if graph[start].is_generator then 0 else −∞;
6          graph[start].longestcg = (c  g);
7          for each (edge, neighbour) in graph[start].neighbours do
8              longest_neighbour = forward_sweep(graph, start, visited) +
                 (edge.length, edge.length);
9              graph[start].longestcg = max(graph[start].longest,
                 longest_neighbour);
10         return graph[start].longestcg
11     else
12         return (−∞  −∞)
13 function collect_totals(graph, start, visited, below = (0  0))
14     if !visited[start] then
15         visited[start] = true;
16         for each (edge, neighbour) in graph[start].neighbours do
17             edge.longest = below + graph[neighbour].longest + edge.length;
18             others = graph[start].neighbours - (edge, neighbour);
19             above = graph[neighbour].longestcg + edge.length;
20             for each (in_edge, in_neigh) in others do
21                 edge.longestcg = max(above +
                     (in_edge.length  in_edge.length) +
                     graph[in_neigh].longestcg, edge.longestcg);
22 function assign_longest_paths(graph)
23     visited = [false for n in graph.nodes];
24     forward_sweep(graph, 0, visited);
25     visited = [false for n in graph.nodes];
26     collect_totals(graph, 0, visited, (0  0));
27     for each edge in graph.edges do
28         (c, g) = edge.longestcg;
29         edge.longest = c + g − edge.length
```

Line resistance is limited by both the voltage drop constraint according to the longest path heuristic and by the power loss constraint. To satisfy both constraints eqs. (4.13) and (4.15) the line resistance r_{l_e} of a single cable in the bundle may not exceed $r_{l_e} \leq \frac{m_e}{\|e\|} \min\{R_d(e), R_l(e)\}$. For each edge we iterate all available cables and compute the minimum multiplicity that has low enough resistance and high enough ampacity. We select the cheapest cable bundle (m_e, l_e) that satisfies the ampacity constraint $I_e \leq m_e I_{l_e}$.

Given a cable selection l_e for each edge $e \in E$ the total cost of the cables is

$$C_{\text{cable}} = \sum_{e \in E} c_{l_e}. \tag{4.17}$$

4.3 Other Cost Factors

Installing an electrical network requires some additional equipment. We adopt the equipment cost model by Kraft [34]. Cables need to be supported by poles or towers. These poles are installed near each terminal and along the cable. A support needs to be installed along the line if the distance to the previous pole exceeds the span length s_p. The cost c_p of installing a single pole is determined by the material cost and labor cost which we assume is constant for every pole. The span length s_p and installation cost c_p are parameters of the planning problem. The total cost of the poles is thus

$$C_{\text{poles}} = c_p \left(|V| + \sum_{e \in E} \left\lfloor \frac{\|e\|}{s_p} \right\rfloor \right). \tag{4.18}$$

Distribution nodes require cable connection equipment. The distribution nodes V_{dist} are all nodes $v \in V$ with node degree $\deg v \geq 3$. In such a branching point special equipment is needed, such as clamps, circuit breakers and switches. We use a simplified cost model which does not differentiate the components. Each distribution point comes with a fixed cost c_s. Each branch comes with additional cost c_b. Finally to account for cost differences in larger installations we add a cost c_i which is multiplied with the maximum power in the node. These three cost parameters are part of the input. We obtain the distribution node cost

$$C_{\text{dist}} = \sum_{v \in V_{\text{dist}}} \left(c_s + c_b \deg v + c_i \max_{e \in E(v)} P_e \right). \tag{4.19}$$

Other electrical equipment such as clamps and connectors is needed at the terminal connections. We assume that the cost for this equipment is independent of the network topology and can thus be neglected for the optimization problem.

4.4 Network Cost

The goal of microgrid cabling is to connect the terminals with minimal total cost. The cable layout has to conform to the constraints introduced above. To reduce the cable length we allow distribution nodes to be placed at additional Steiner nodes S. The edges $E \subseteq [V]^2$ connect all nodes $V = S \cup T$ forming an unrooted tree. Total network cost consists of not only the cost of cables but also supporting poles and equipment in distribution nodes. We obtain

$$C = C_{\text{cable}} + C_{\text{poles}} + C_{\text{dist}}. \tag{4.20}$$

Problem 4.2 (Microgrid-Cabling)
Inputs: The inputs to the problem are

- terminals T and demands $d(t)$
- cable types L
- equipment costs c_s, c_b, c_i and c_p
- voltage drop tolerance t_{drop} and power loss tolerance t_{loss}
- line voltage U

Outputs: Electric network $G = (S \cup T, E)$, Steiner points S, cable assignments l_e

Objective: minimize $C = C_{\text{cable}} + C_{\text{poles}} + C_{\text{dist}}$

Properties: The network satisfies the ampacity constraint eq. (4.3), the power loss constraint eq. (4.6) and the voltage drop constraint eq. (4.7). The network G is connected.

4.5 Complexity

Microgrid Cabling is NP-hard. We show this by reducing the Steiner tree problem which is known to be NP-hard (see section section 2.2.1) to Microgrid cabling. Given a geometric Steiner tree instance $T = \{t_1, \ldots, t_n\}$ with n terminals and respective positions we construct a Microgrid cabling instance with the same terminals. We set the demand $d(t_1) = 1$ and all other demands $d(t_2) = \cdots = d(t_n) = -1$. The

equipment cost is set to $c_s = c_b = c_i = c_p = 0$. We use only a single cable type $l = (c_l = 1, I_l = \infty, r_l = 0)$ with infinite ampacity and no resistance. With a perfect conductor as cable we cannot experience voltage drop and power losses. The tolerance can therefore be set arbitrarily. Similarly, the grid voltage can be set to an arbitrary positive number. This reduction is clearly linear in n.

5 Genetic Model

In this chapter we outline the operators and components we used in out genetic algorithm. The operators are not tailored to electric grids and can be used for a variety of spanning tree and Steiner tree problems. Some related approaches [41] don't operate on a geometric problem but on a given graph. Our method does not require explicit edges in the input and instead considers all connections possible.

One major goal of our operators is to ensure all generated instance are trees. This is particularly important for crossover and mutation operators. Because of this, all operators described prevent the creation of cycles or to disconnect their input graph.

We first introduce how we represent the trees in section 5.1. Then we will introduce methods to generate an initial population of trees in section 5.2. The operators are grouped into crossover operators in section 5.3 and mutation operators in section 5.4. We present several mutation methods which can be combined to suit a variety of tree problems. Common procedures and post processing methods of the operators are described in section 5.5. Finally, we describe how to extend the model to avoid forbidden regions in section 5.7.

5.1 Representation

We store the network as a singly-linked adjacency list. Each edge has a link to the next incoming edge of its target node and to the next outgoing edge of its source node. We list the operations and their time complexity in table 5.1. We do not use a doubly linked list for faster deletions because the average vertex degree in a tree is less than two and thus deletions are fast on average even with a singly-linked list. Terminals are uniquely identified by their index among all individuals while Steiner points are ordered arbitrarily. For each node we store its type, supply and demand as well as its position. Using a graph representation enables us to store, traverse and manipulate trees efficiently. We use the same encoding for genotype and phenotype, saving the decoding step.

Operation	Time	Description
`addNode`	$\mathcal{O}(1)$	add an isolated new node
`addEdge(u, v)`	$\mathcal{O}(1)$	add an edge between u and v
`deleteEdge(u, v)`	$\mathcal{O}(\deg(u) + \deg(v))$	removes and edge between u and v
`deleteNode(v)`	$\mathcal{O}(\sum_{w \in \text{neigh}(v)} \deg(w))$	remove a node v and all incident edges
`neighbors(v)`	$\mathcal{O}(\deg(v))$	find all neighbors of v

Table 5.1: Operations supported by our graph representation and their respective time complexity.

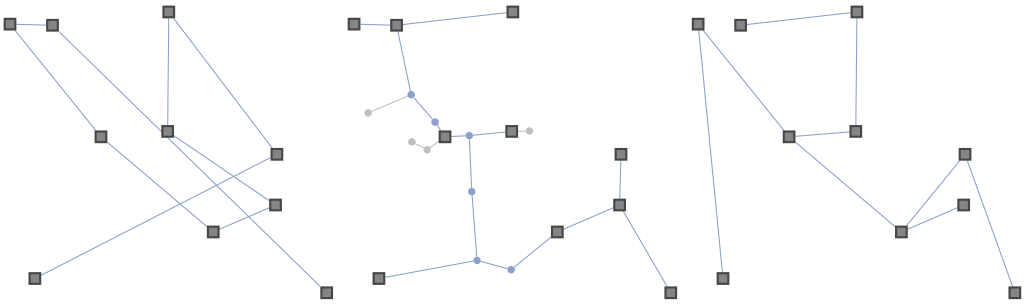
5.2 Initial population

When building an initial population, we have to ensure that all individuals are valid trees. We present three methods to generate trees with different properties. Possible results of the initialization can be found in fig. 5.1.

A simple method of initializing a tree is by constructing a path. For terminals $t_1, \ldots, t_{|T|}$, *path initialization* generates a path by choosing a random permutation π of $\{1, \ldots, |T|\}$. The edges are then given by $E = \left\{t_{\pi(i)} t_{\pi(i+1)} | i = 1, \ldots, |T| - 1\right\}$. This method is very fast as a random permutation can be generated and iterated in linear time. The quality of such an initial solution is, however, usually quite low.

Path initialization often leads to crossing edges and very inefficient paths, which can be undesirable. Instead of a random path, we can use the minimum spanning tree on the terminals and k randomized Steiner nodes S. We first draw a user-supplied number k of points S uniformly at random from the convex hull $\text{conv}(T)$ of the terminals. Then we construct the minimum spanning tree on the nodes $V = T \cup S$. Using additional Steiner nodes increases the genetic variety and avoids initializing a uniform initial population. The minimum spanning tree does not contain crossing edges. Pruning of unnecessary Steiner nodes is, however, necessary to remove redundant Steiner nodes, as can be seen in fig. 5.1b. We call this method *mst initialization*.

A closely related third method is *triangulation initialization*. Instead of building a minimum spanning tree, we construct a random spanning tree from edges of a planar triangulation, such as the Delaunay triangulation. This can be achieved by iterating the Delaunay edges in randomized order and inserting only such edges that



(a) path initialization

(b) MST initialization, $k = 10$ $|T| = 10$ Steiner nodes

(c) triangulation initialization

Figure 5.1: Results of the different initialization methods before further refinement. All three examples were generated with the same $|T| = 10$ randomly placed terminals. The square nodes are the terminals, round nodes are Steiner nodes. The light gray nodes and edges do not receive any power flow and can be pruned. Prunable Steiner nodes of degree 2 are not marked specifically.

don't result in a cycle. Kruskal's MST algorithm follows a similar approach but we don't sort the edges and instead shuffle them randomly. Both MST initialization and triangulation initialization construct a Delaunay triangulation and thus run in $O(|V| \log |V|)$ time.

5.3 Crossover

An important goal of our work was to find a crossover operator that works on euclidean trees and gives meaningful results. Because we move the Steiner nodes, their indices and positions are unrelated in different individuals. If we ignore this fact, most of the geometric information in the parents is lost when breeding offspring. To obtain more meaningful results, we propose several structural crossover operations. All these operations use two parents.

5.3.1 Separate Crossover

Moharam and Morsy [41] proposed a crossover method for edges in graph based spanning tree problem. Their problem does not include Steiner nodes and thus our extension needs a crossover method for Steiner points positions. In *separate crossover*, we first combine the Steiner points of both parents P_1 and P_2. We insert edges in a separate step. Separate crossover is illustrated in fig. 5.2. Figure 5.2a

and fig. 5.2b show the parents and the selected Steiner nodes. Figure 5.2c shows how the Steiner nodes and edges are mapped to edges in the child which is depicted in fig. 5.2d.

To combine the Steiner nodes, we collect the Steiner nodes S_1 and S_2 in both parents. We then choose $k = \max\{|S_1|, |S_2|\}$ child Steiner nodes uniformly at random from $S_1 \cup S_2$. Let $V_c \subset V(P_1) \cup V(P_2) = T \cup S_1 \cup S_2$ be the resulting set of nodes in the child. This has running time $\mathcal{O}(|V(P_1)| + |V(P_2)|)$.

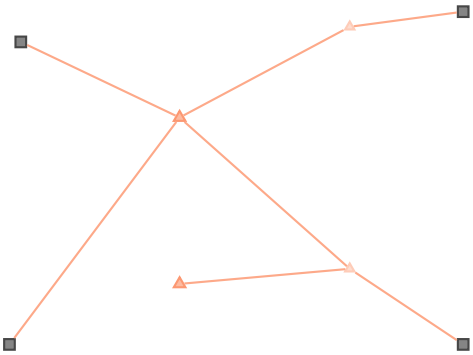
Our edge crossover is based on Moharam and Morsy [41]. They used a similar method in genetic algorithms for constrained balanced trees. We extend their method to graphs with additional Steiner nodes which may change position. Because the Steiner points in the parents are not necessarily present in the child, we need a mapping from parent edges to child edges. We map the endpoints u and v to the nearest vertices in V_c. For faster lookup, we use an R*-tree spatial index structure [7] containing all nodes in V_c. We use OMT bulk loading [38] to insert the nodes into the tree. On average, lookups and insertions take logarithmic time in the number of points with R*-trees but the worst-case running time may be higher. Let E_1 be the edges $E(P_1)$ of the first parent mapped to the closes endpoints in the child. Similarly, let E_2 the mapped edges $E(P_2)$ of the second parent. We first insert all edges in E_1 into the child. Then we determine the edges $E_u = E_2 \setminus E_1$ unique to the second parent. We choose a uniform random number $k \in \{0, \ldots, |E_2|\}$ and insert k edges drawn uniformly at random from E_2 into the child.

To avoid creating cycles when inserting an edge, we always check whether an alternative path between the endpoints exists. If such a path is found, we remove an edge on this path which we draw uniformly at random. Because Steiner nodes and edges are chosen independently, some Steiner nodes in the child might remain disconnected. We use a depth-first search to find all components of the graph and add an edge to a random previously visited node when we enter a new component. This ensures the network is connected.

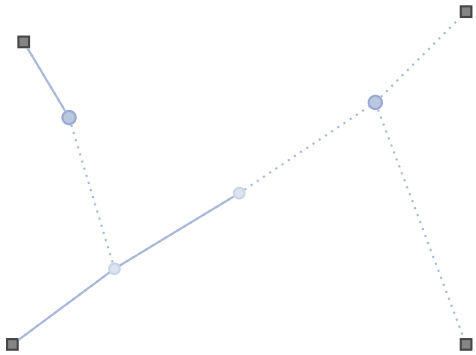
5.3.2 Subtree crossover

Separated crossover does not consider local topology and is thus likely not to preserve locally optimal structures. We propose *subtree crossover*, a new crossover operator that is targeted at such local structures. This method is illustrated in fig. 5.3. By inserting a whole subtree from one parent into the other we keep structures consisting of multiple nodes and edges.

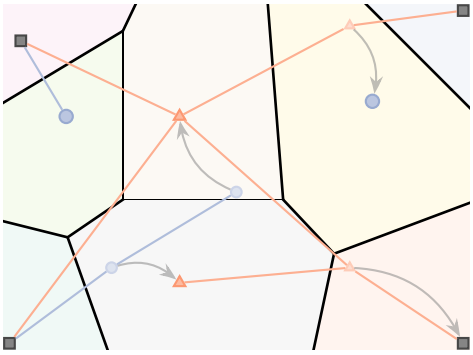
We give the pseudocode for this crossover method in algorithm 3. First, we choose



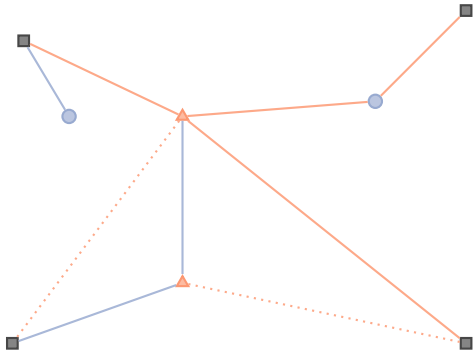
(a) First parent. Triangular nodes are Steiner nodes



(b) Second parent. Steiner nodes are round. Solid edges have been chosen for crossover.

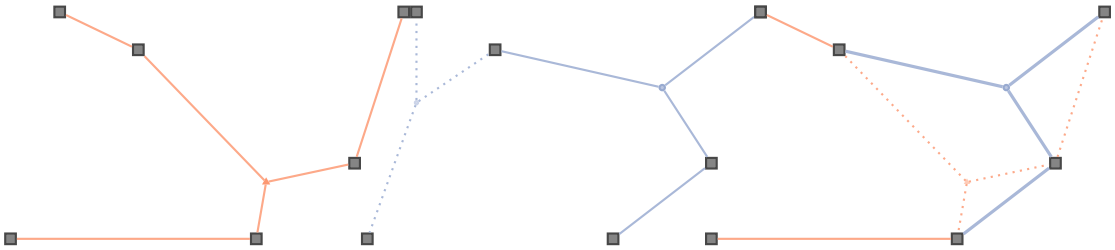


(c) Steiner nodes chosen for crossover. Faint nodes have not chosen for crossover. All endpoints are mapped to the closest node in the child graph, which is the center of each voronoi region.



(d) Edges after crossover. Orange edges are taken from first parent, blue edges are taken from the second parent. Dotted edges have been removed to break cycles.

Figure 5.2: Separate crossover of random edges from the second parent into the first parent. Steiner nodes are chosen randomly from both parents, the discarded nodes the smaller and fainter nodes. Edge endpoints are mapped to the closest points in the offspring network.



(a) First parent network

(b) Second parent. The selected subtree is marked with solid lines.

(c) Child after crossover. The inserted subtree is marked with thick blue lines.

Figure 5.3: In subtree crossover operator we select a subtree from the second parent and then use edges and Steiner nodes from the first parent to connect the remaining nodes in the child network. Orange edges are from the first parent and blue edges from the second parent. The dotted lines in (c) are discarded because their endpoints are already connected by the blue subtree.

an edge in the second parent P_2 uniformly at random. Removing the edge splits the graph into two components, one of which we choose uniformly at random. The chosen component is denoted by U. We initialize the child $C = (T \cup V(U), E(U))$ with the subtree U and the terminals T. Then we iterate the edges in P_1 in randomized order and insert all edges that connect disconnected components. If one of the endpoints is a Steiner node that is not yet present, in the child we also insert the Steiner node. Finally, we prune all redundant Steiner nodes as described in section 5.5.1.

Selecting the subtree takes $\mathcal{O}(|V(P_1)| + |E(P_1)|)$ time with a depth-first search. We use union-find with path splitting [51] to keep track of connected components. Node insertions and connectivity lookup account for a running time of $\mathcal{O}(|(|\,E(P_1))a(V(P_1) + V(P_2)))$ where $a(n)$ is the inverse Ackermann function. We perform up to $|E(U)| + |E(P_1)|$ edge insertions which leads to a total running time of $\mathcal{O}(|V(P_1)| + |V(P_2)| + |E(P_2)| + |E(P_1)|\, a(|V(P_1)| + |V(P_2)|))$.

5.4 Mutation

Mutation has the purpose of introducing new genetic information in the population. In our case this comprises connections and Steiner point positions. Hence, our mutation phase makes use of several independent operators for edges and Steiner node positions, which can be chained in an arbitrary order. An overview of the operators can be found in table 5.2 before the more detailed description in this

Algorithm 3: Subtree crossover

```
1  function subtree_crossover(P1, P2)
2      edge = P2.edges.choose();          /* drawn uniformly at random */
3      subtree = P2.split_at(edge).components.choose() ;          /* random
         component */
4      child = graph(P2.terminals.union(subtree.vertices), subtree.edges);
5      for each edge in P1.edges.shuffle() do
           /* add missing nodes                                          */
6          if edge.source not in child then
7              child.addNode(edge.source)
8          if edge.target not in child then
9              child.addNode(edge.target)
           /* insert edge if it connects disconnected components  */
10         if edge.source not connected to edge.target in child then
11             child.addEdge(edge.source, edge.target)
12     prune_steiner_nodes(child);
13     return child
```

Table 5.2: Overview of our genetic mutation operators and their parameters

Operator	Parameter	Description
Close Endpoint		Move edge endpoints to a different close point
	p_m	Mutation rate. Probability of each edge to be modified.
Nudge		Move Steiner point towards a randomly selected neighbor by a relative distance between 0 and 1.
	p_m	Mutation rate. Probability of each Steiner point to be moved.
Minimize		Reduce the total weighted length of the edges in the network.
	p_m	Probability of reducing the links in the whole network.
	n	number of minimization iterations

section.

Moharam and Morsy [41] proposed an edge mutation method for trees. Their work focuses on diameter constraint spanning trees in a graph. With probability p_m a random edge in the tree is replaced with an edge from the underlying graph that is not in the tree. The resulting cycles that occur are broken by deleting a random edge (see section 5.5.2). The mutation rate p_m is an operator parameter. We do not have an explicit underlying graph but can compensate for this by choosing from all $\binom{|V|}{2}$ possible edges. This previous method has several drawbacks. Many possible edges span across the graph and are unlikely to improve the result. Also, inserting only single edges can make escaping local minima harder.

We propose new method follows a similar but different strategy. *Close endpoint mutation* modifies existing edges by moving their endpoints to different nodes. The operator has a parameter p_m resembling the probability of an edge being mutated. Each single edge is mutated with probability p_m. If an edge uv is mutated we first remove that edge from the graph. We randomly pick one endpoint u and select a different point u' in the graph, where each point u' is chosen with a probability proportional to the inverse of its squared distance $1/\left\|u-u'\right\|^2$ from u. Then we insert (u', v) to the network, breaking newly formed cycles if necessary. If the insertion did cause a cycle, we reinsert (u, v) to ensure the graph remains connected. Close endpoint mutation has expected running time $\mathcal{O}(p_m \left|E\right| \left|V\right|)$ which, in a tree, is quadratic in the number of nodes.

We can enhance close edge mutation to also insert new Steiner nodes. Before edge endpoints are replaced, we insert possible Steiner points by subdividing all edges. Then we perform the edge mutation as described before. The chosen endpoint can now be moved to a Steiner point on an edge. After all new edges have been inserted we prune the redundant Steiner nodes, as described in section 5.5.1. The pruning step also helps to get rid of Steiner points that have become redundant by replacing their incident edges.

For Steiner points we can use the *nudge mutation* operator used by Barreiros [6] who called it *compress*. This operator moves Steiner points towards one of its neighbors. If it is selected, the Steiner node is moved along the edge to a randomly selected neighbor by a relative distance chosen uniformly at random from $[0, 1]$.

Alternatively we use *link-length minimization* as mutation operator. It reduces the total weighted length using the algorithm by Kuhn and Kuenne [36] and Miehle [40]. We weight the edges by their cable cost prior to the minimization. The minimization is parameterized over the number of iterations n and the probability of minimizing the network in the mutation phase p_m. For each child C we generate a random number $s \in [0, 1]$. If $s < p_m$ we perform n iterations of the minimization algorithm.

This is more efficient if we first prune all unnecessary Steiner points.

Using link-length minimization may not yield the optimum result when combined with poles that are placed in a certain interval when moving the Steiner point results in fewer points being placed. To address this we use both link-length minimization and nudge mutation. Nudge mutation helps to explore Steiner positions that are non-optimal in terms of the cable cost but may need fewer support poles.

5.5 Refinement

Some of the operators described above need to refine their outputs to remove artifacts that can be introduced by deleting or inserting edges. In this section we list useful building blocks to handle such artifacts and ensure that outputs are trees.

5.5.1 Steiner Node Pruning

When reconnecting the network, Steiner nodes may become redundant. If the triangle inequality holds in the problem, we can safely remove Steiner nodes of degree 2. The connection through such a node cannot be shorter than the direct connection. Steiner nodes that are not part of a path between terminals can always be removed. We merge coinciding neighbors if at least one of them is a Steiner node.

We identify prunable Steiner nodes using a depth-first search starting at a terminal. We then keep track of whether a terminal has been reached during backtracking. The node can be pruned if no more than two incident edges of a Steiner node are part of a path between terminals. Identifying the prunable nodes takes $\mathcal{O}(|V| + |E|)$ time.

5.5.2 Avoiding Cycles

Some operations insert edges uv into the tree which can result in a cycle if no edge has been removed. We prevent this by removing an edge we chose uniformly at random on the path between u and v before inserting uv.

If we are not interested in a specific edge being inserted, we can instead take a shuffled list of edges and iteratively insert all edges between disconnected components. This

is very similar to Kruskal's algorithm for computing minimum spanning trees. To efficiently find and merge the components the vertices belong to, we use a union-find data structure.

5.5.3 Constant Number of Steiner Nodes

If, for some reason, we want to keep the number of Steiner nodes in the network constant, we can contract Steiner nodes or insert new ones to match the desired number. When contracting a node, we randomly choose one of its neighbors and connect all other neighbours to the selected one. We subdivide edges in randomized order to ensure an even distribution of Steiner nodes.

5.6 Microgrid Cabling

We use the genetic operators presented in this chapter to generate cable layouts for the Microgrid Cabling Problem. The objective function we use is the total network cost eq. (4.20). For each individual we compute a cable assignment l_e using the heuristics presented in section 4.2.2 and combine the cable cost with the cost of distribution nodes and poles to obtain the individual's objective value C. Because we minimize the cost, fitter individuals have lower objective values. We use tournament selection which is rank based and thus works well for both maximization and minimization problems.

The operators can be used without modification. Link length-minimization uses the assigned cable cost as weight. This may lead to an increase in cost if the cable types change. Subsequent mutations will, however, use the new cable type and thus this should not pose any problem. The minimization does not take into account the placement of poles. Hence, an optimum Steiner point position with respect to the cables may still lead to a higher total cost if additional poles need to be placed. We apply the nudging mutation after minimization to explore better Steiner point positions.

5.7 Forbidden Regions

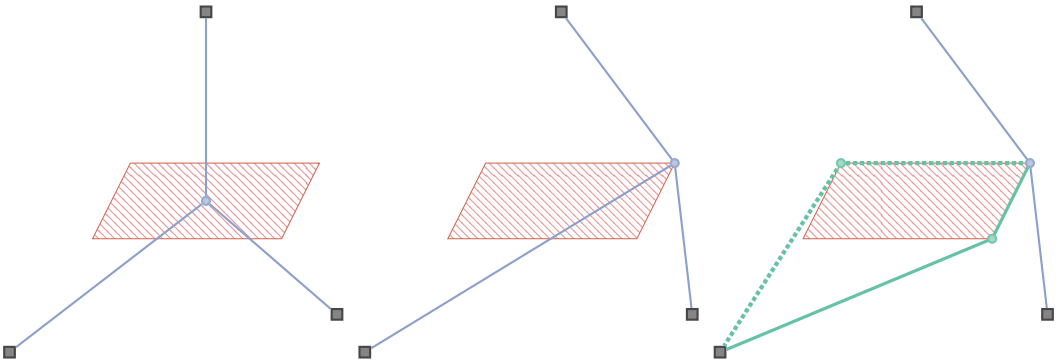
In some parts of the planning area, building new power lines might not be possible. There might be a lake or wider river or a cliff that have to be avoided. We propose

an extension to the model that can handle such regions. It is illustrated in fig. 5.4. Currently, our method is limited to non-overlapping convex polygonal regions, but an extension to other geometries is possible. Also, terminals cannot be placed inside obstacles. The proposed method does not require changes to the existing operators and can be implemented as an additional mutation operation that is executed last. Note that this requires the pruning procedure to respect Steiner points at vertices of the regions which may only be removed if they are isolated or leaves.

Given a set of obstacles O, we have to ensure two things. First, no Steiner node may be located inside an obstacle, and second, no cable may pass through an obstacle. For each obstacle $o \in O$, we identify the Steiner nodes $S_o \subseteq S$ that are inside the obstacle. We then select a corner vertex $v \in o$ uniformly at random and replace all edges to inner Steiner nodes $s \in S_o$ with edges to v. We do not insert the edge if the other endpoint is already connected to v and a new cycle would be formed. The resulting intersection with the obstacles are taken care of in the second step.

We resolve line intersections using the convex hull based bypass algorithm presented by Dutta and Overbye [18]. If an edge intersects an obstacle, we compute the convex hull of the obstacle and the edge endpoints. We then choose uniformly at random one of the two possible paths between the endpoints along the convex hull. This step requires the edge endpoint to be outside the convex hull which is why we need to move the Steiner points first. A line can intersect multiple obstacles. We resolve this by applying the above procedure to one obstacle at a time and checking the edges on the resulting path for collisions separately.

Currently, we compute intersections of edges and obstacles by checking all pairs of an edge and an obstacle line segment for intersections, repeating until no more intersections are found. With m edges and $|O|$ obstacles with $|o|$ sides per obstacle $o \in O$ this results in a running time of $\mathcal{O}(m\,|O| \sum_{o \in O} |o|)$. This could be improved significantly by using a different line intersection algorithm such as the Bentley-Ottman algorithm [8] that could compute all intersections in $\mathcal{O}((m+s+k)\log(m+s))$ where k is the number of intersections. The running time could also be improved by only checking for an intersection if the line intersects the bounding box of the obstacle and by employing spacial data structures to reduce the number of potential obstacles.



(a) Network with Steiner point inside obstacle and three intersecting edges

(b) Steiner point moved to an obstacle vertex, leaving one edge intersecting the obstacle

(c) After replacing the intersecting edge with the solid green path. Also showing the alternative path.

Figure 5.4: Process of resolving collisions with obstacles. Steiner points are first moved out of the obstacle, then the intersecting edges are rerouted along the convex hull of their respective endpoints and intersected obstacles.

6 Experiments

In this chapter we present the results of our algorithm in two problem settings. First, we use the Euclidean Steiner Tree Problem to test the general ability of our algorithm to solve such problems optimally. Then, we examine how our method works on a real world microgrid example.

Our experiments were run on a SuperMicro H8QG6 Server with four AMD Opteron 6172 processors and 256 GB of memory. The machine has a total of 48 CPU cores. Our implementation is written in Rust. We used a modified version of genevo[1] as a framework for running genetic algorithms and as basis for our own operators. The genetic algorithm uses rayon[2] for parallelization.

6.1 Steiner Trees

The Euclidean Steiner Tree Problem is well studied and there are algorithms to compute optimal solutions. As the Steiner tree problem is a subproblem of microgrid cabling (see section 4.5), we expect our approach to yield good solutions. We don't expect the same speed or solution quality as from specialized algorithms such as GeoSteiner [31] due to the more general nature of our problem formulation. In particular, we do not expect our algorithm to find the minimum Steiner tree but only a good approximation.

Our test dataset is the DEG set from the DIMACS 11 implementation challenge [1]. It contains instances of sizes between 10 and 100 terminals which are grouped by size with 1000 randomly generated instances each. The data does not contain information on demand and supply. Our objective in the Euclidean Steiner Tree Problem is to minimize the total length. We create a cabling problem instance using the transformation described in section 4.5.

Our first set of experiments aimed to find a good configuration of the genetic

[1] `https://github.com/innoave/genevo`
[2] `https://github.com/rayon-rs/rayon`

algorithm and to test performance of our new operators on this known problem. The configuration variables we explored were mutation rate, population size and a few selected operator configurations.

We not only measured the time to reach the optimum solution but also the time to reach different solution qualities within several different percentages of the minimal length. Execution time was limited and we counted the fraction of runs to reach the final solution quality before timeout. This yields two quality measurements: The running time and the fraction of runs that did not complete before timeout. We compared configurations by their median running time. The median is more stable than the arithmetic mean, especially when the running time is potentially unbounded.

6.1.1 Mutation Rate

Mutation rate measures the number of random mutations in a recombined genome. With a high mutation rate, beneficial mutations may be overshadowed by detrimental ones. If the mutation rate is too low, however, useful mutations may not be found at all, particularly, if several random changes need to be combined for an improvement. When raising the mutation rate, we thus expect the failure rate to drop at first, reaching a minimum and then to start rising again.

We studied the properties of our algorithm with different mutation rates on several different instance sizes. We computed the minimum Steiner tree using GeoSteiner and repeated the experiment several times per instance to reduce the influence of statistical errors. To detect diverging solutions, we limited running time.

On 50 terminals, we chose a timeout of 90 seconds. We used the 31 first instances of the data set. We ran the experiment 10 times per instance. The population of size 5000 was initialized using the minimum spanning tree method with $2\,|T|$ randomly distributed Steiner points. We increased the timeout to 120s on 100 terminals and only used 21 instances. Each new generation was generated with 25-tournament selection and elitist reinsertion, replacing 95% of the old population with new individuals. We used subtree crossover to recombine the selected parents. We used distance weighted edge mutation and applied link-length minimization to the Steiner point positions. Mutation rate thus refers to the probability of an edge to be replaced by a similar edge. The running time with different mutation rates can be found in fig. 6.1. The plot shows the distributions of running time to reach several multiples of the minimal length as computed by GeoSteiner. We found that when our genetic algorithm found the optimal topology and Steiner point positions, the total length was always slightly lower than the one computed by GeoSteiner.

This is most likely due to floating point precision. The result could be improved by simply reordering the edges, which is a strong indicator of floating point issues.

The median running time on 50 terminals strictly increased in all quality steps for all rates above 0.01. The solution quality was within 10% of the optimal solution in the second iteration, regardless of the mutation rate. Observe that this was the case for all runs, not only the median. The best initial solutions were already within 50% of the optimal solution.

The median convergence time is lowest with low mutation rates. Median convergence time is, however, not the only quantity we employ to measure performance. Figure 6.1 shows the variation of running times. Even though the lowest mutation rates lead to a fast result in the median case, a much higher percentage of the runs takes longer which also results in more runs failing to reach the optimum before timeout. The lowest variance can be found at mutation rates 0.02 and 0.03. While the median running time steadily increases, the mean running time stays approximately the same between 0.02 and 0.05. This is also reflected in the timeout ratios found in fig. 6.2. It shows the percentage of runs not to reach the optimal solution or multiples of it before timeout. The ratio of timeouts stays low between rates 0.02 and 0.09 and starts a sharp rise above 0.09. This increase can be explained with the rise in running time, which almost doubles from a median of 9 seconds for a rate of 0.06 to 17 seconds at a rate of 0.07 and rises further to 60 seconds at a rate of 0.09. The median runs did not reach the optimal solution with rates of 0.1 and above.

This coincides with a rapidly rising running time which leads to less runs reaching the optimal solution in time. Low mutation rates have higher variance and have a higher tendency to get stuck in local minima. The lowest timeout rate of 12% can be found at a mutation rate of 0.05 after which both timeouts and running times increase.

Observe that the timeouts on 50 terminals look different if we don't expect the optimal solution but a solution that exceeds the minimal length by a factor of at most $1 + 10^{-5}$. While for the rates below 0.09, the number of timeouts is almost the same, it drops by a factor of more than 2.5 for $p_m = 0.1$ and $p_m = 0.11$. With mutation rates between 0.02 and 0.08, around 90% of all runs that reached less than $1 + 10^{-3}$ times the optimal solution also reached the minimum. A similar trend can be seen with running times. For the mutation rates below 0.05, the median time to reach the optimal solution was around 1.1 times the time to reach $1 + 10^{-3}$ times the minimal length. Above and including a mutation rate of 0.08 it took more than twice as long, if the solution was found at all. This indicates that too high mutation rates converge slower and have a lower chance of improving an almost optimal solution.

With 100 terminals, the rate of timeouts is much higher. The highest success rate we observed was with mutation rates between 0.01 and 0.02 where only around 35% of all runs failed to find the minimum tree. This was also the range with the lowest median running times for finding the optimal solution.

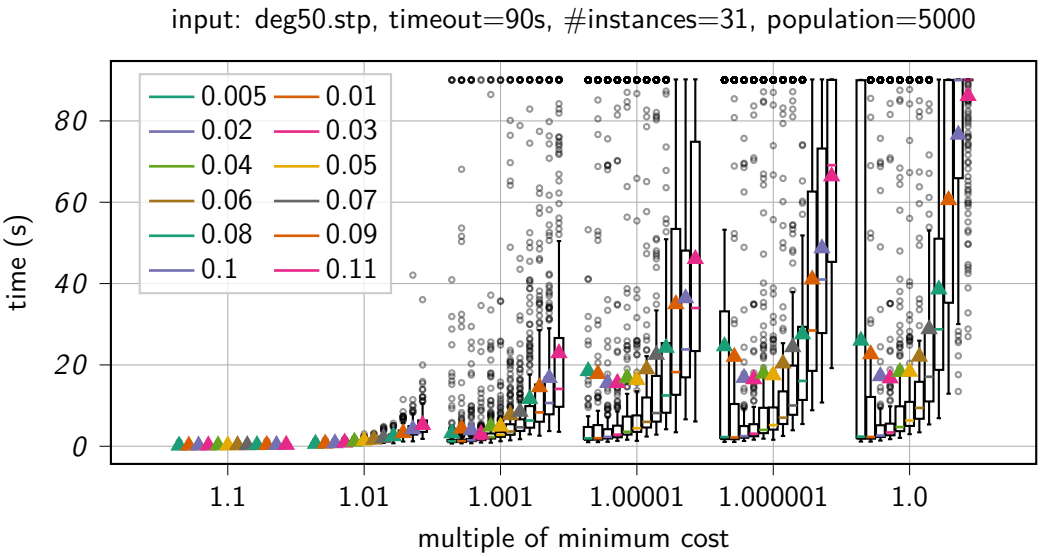
The rising running time at the higher mutation rates is caused by two factors. First, with a higher mutation rate the iterations take longer because more actual mutations are performed. For a mutation rate of 0.01, the time per iteration is about half the time of a rate of 0.1. This is overshadowed by the second contributing factor, the number of iterations. On 50 terminals to a mutation rate of 0.03 the optimal solution is found with a median number of 30 iterations. This increases to 62 iterations for a mutation rate of 0.06 after which it rises further to 420 iterations for a mutation rate of 0.1. With a mutation rate of 0.11, the median number of iterations falls again but this is due to the rising time per iteration.

From this experiment we can already see that our algorithm does not always find the minimum tree. By looking at the instances individually we observed that the performance is not the same on all instances. Figure 6.3 shows the timeout ratio by instance on 50 terminals. While some instances never timed out, others missed the optimum in more than 60% of all runs. One example that was not solved optimally was instance 21, which was the least successful one with only 27% successful runs. With this particular instance, the failure rate was nearly the same at all mutation rates. Nearly all runs found the same locally optimal solution, an extract of which is given in fig. 6.4a. It shows the subset of the terminals that cause the deviation from the minimum Steiner tree. The optimal topology differs from our result in several places. At least five simultaneous mutations are necessary to escape the local minimum. This is especially difficult to escape, because all single mutations on the way are disadvantageous and will thus be selected against. Even though the result is not minimal, its length is only 1.0004 times higher than the optimal. From the figure it is apparent that the highest mutation rates caused more timeouts. In the instances where less than 25% of all runs did not find the optimal solution in time, the rates between 0.01 and 0.07 only accounted for 9% of all timeouts whereas 0.1 and 0.11 combined accounted for 75%.

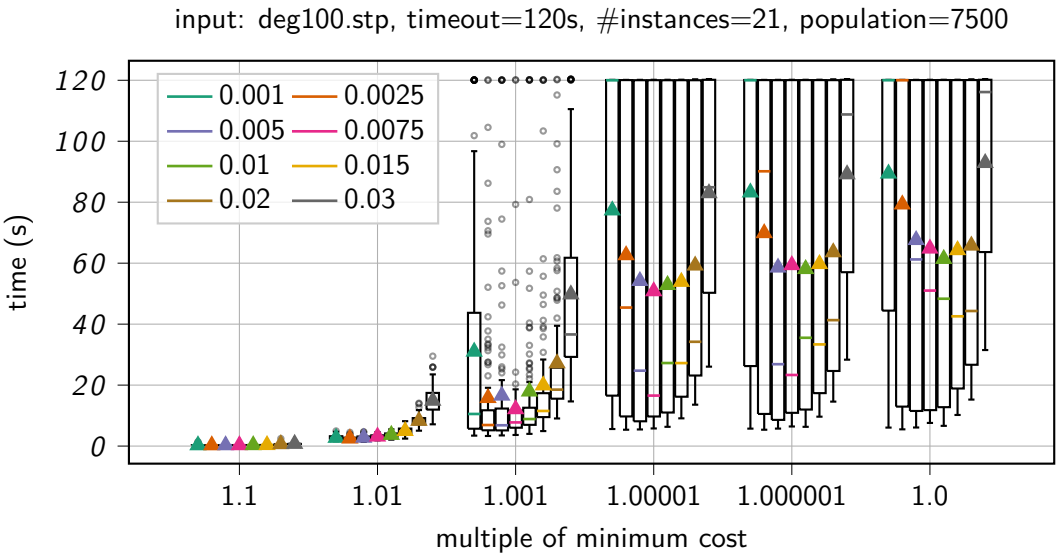
6.1.2 Population

Population size is the second major common parameter of genetic algorithms. We conducted an experiment to find which population sizes yield the best performance. The resulting running times on 50 and 100 terminals can be found in fig. 6.5.

With 50 terminals, we used population sizes between 500 and 60000. From 500 to

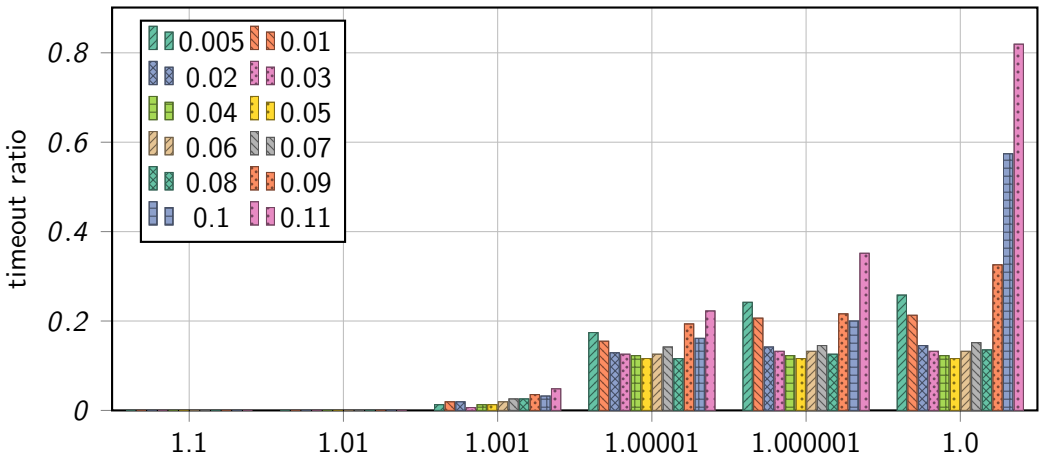


(a) 50 terminals with a timeout of 90 seconds

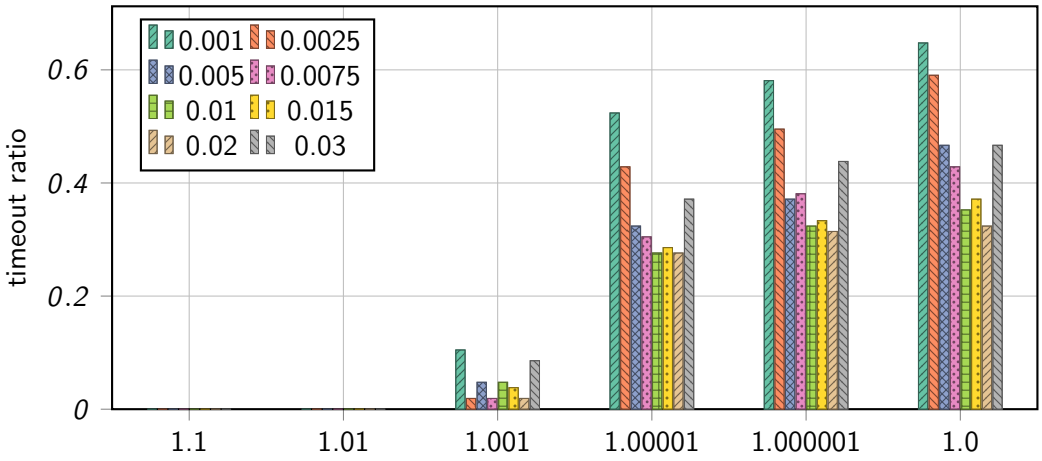


(b) 100 terminals with a timeout of 120 seconds

Figure 6.1: Distribution of running times to reach different quality milestones relative to the optimal solution with varying mutation rates. Colored triangles mark the mean, colored lines mark the median



(a) 50 terminals with a timeout of 90s



(b) 100 terminals with a timeout of 120s

Figure 6.2: Ratio of timeout runs per mutation rate with 50 and 100 terminals.

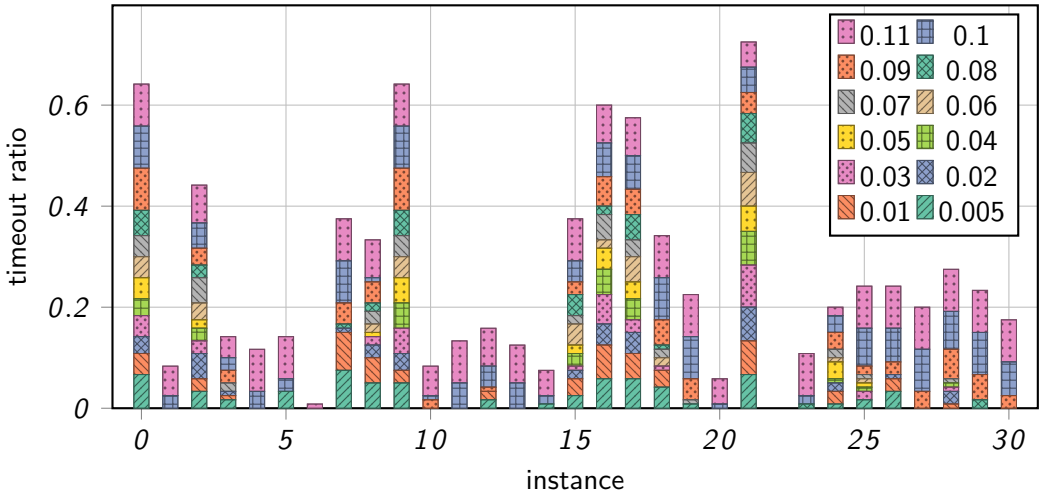
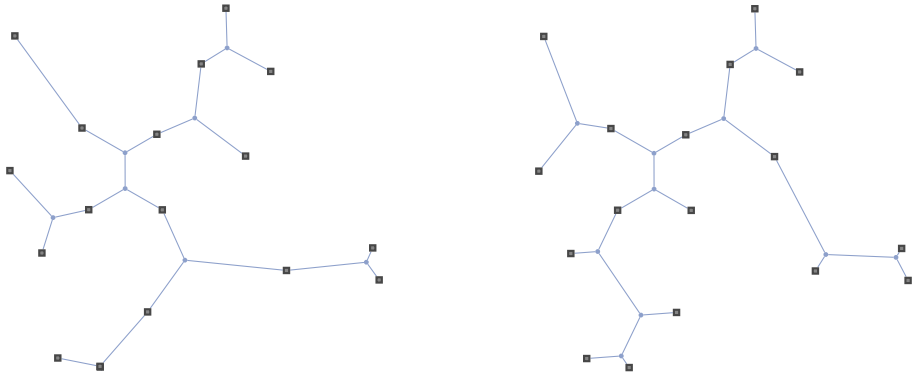


Figure 6.3: Timeout ratio ($t = 90\,\mathrm{s}$, 10 runs each) by 50-terminal instance aggregated over all mutation rates.



(a) Our solution **(b)** Reference solution

Figure 6.4: Our algorithm has problems with local minima that can only be escaped with multiple changes at once. In this example, no step towards the target topology yields a lower cost, only all changes at once are actually lower. Extract from instance 21 of the deg50 set. This instance was particularly prone to local minima.

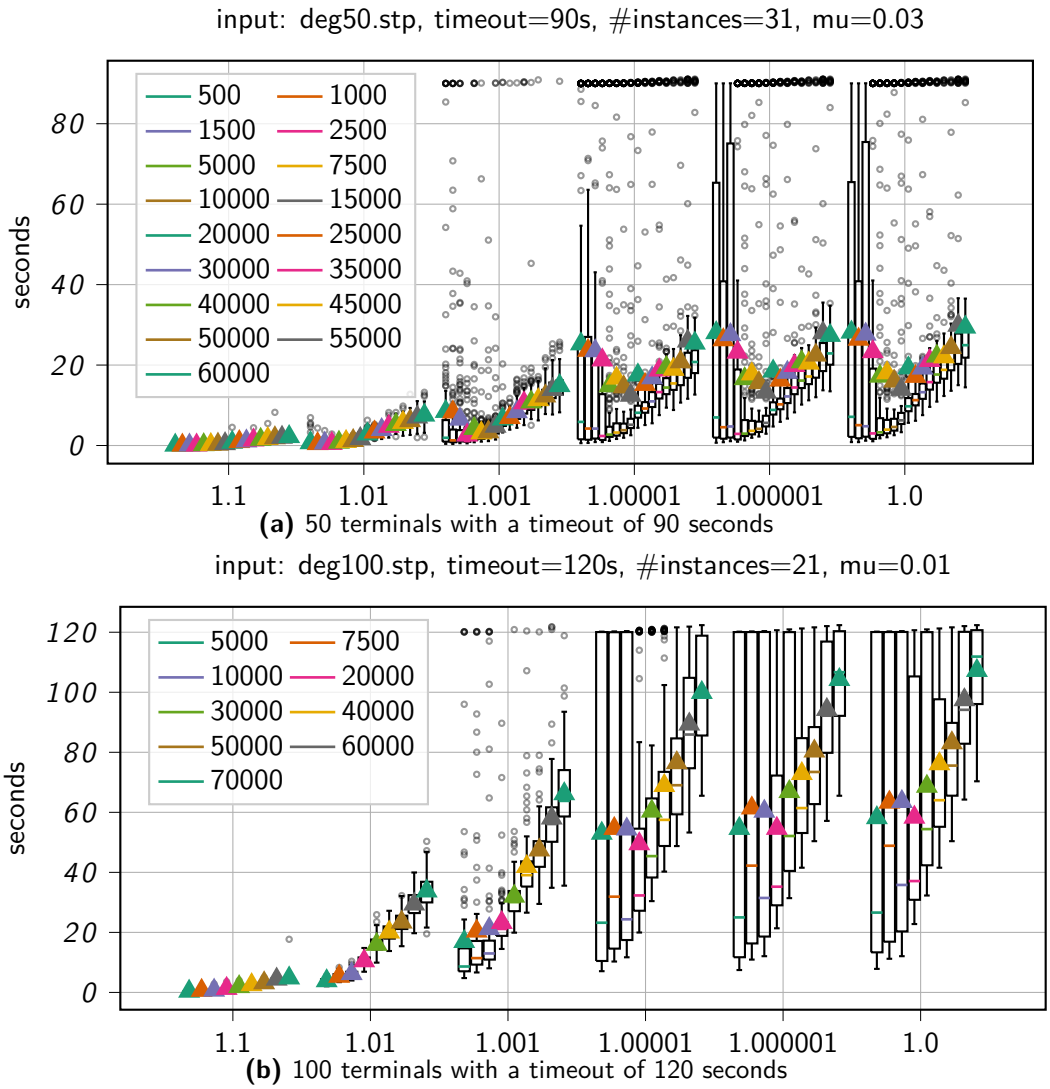


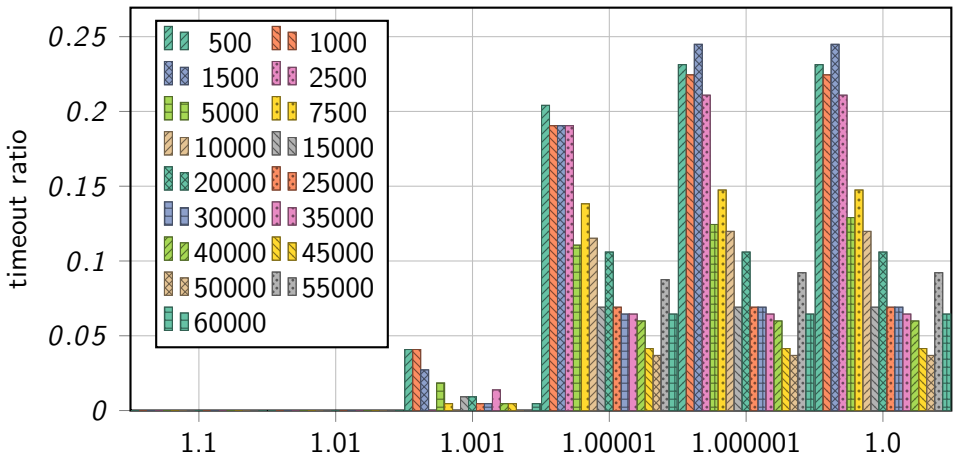
Figure 6.5: Running time distribution to reach multiples of the optimal solution cost with population sizes for 50 and 100 terminals.

2500 individuals, the median running time first falls from 7 seconds to 3 seconds. From 3 seconds at 5000 individuals, it rises to 4.6 seconds at 10000 individuals. The median running time rises approximately linearly to 25 seconds between 10000 and 60000 individuals. A population size below 2500 resulted in a higher median running time. The small population size restricts the genetic variation within the population. Raising the population size decreased the number of timeouts as can be seen in fig. 6.6a We found that small populations below 2500 individuals lead not only to a higher running time but also to more timeouts. Between population sizes of 2500 and 5000, the number of timeouts drops by more than a third from 21% to 12%, as can be seen in fig. 6.6a. It drops even further to 7% at 15000 individuals reaching a minimum of 3.7% at 50000 individuals. The higher genetic variation in a larger population clearly leads to more successful runs at the cost of a higher running time.

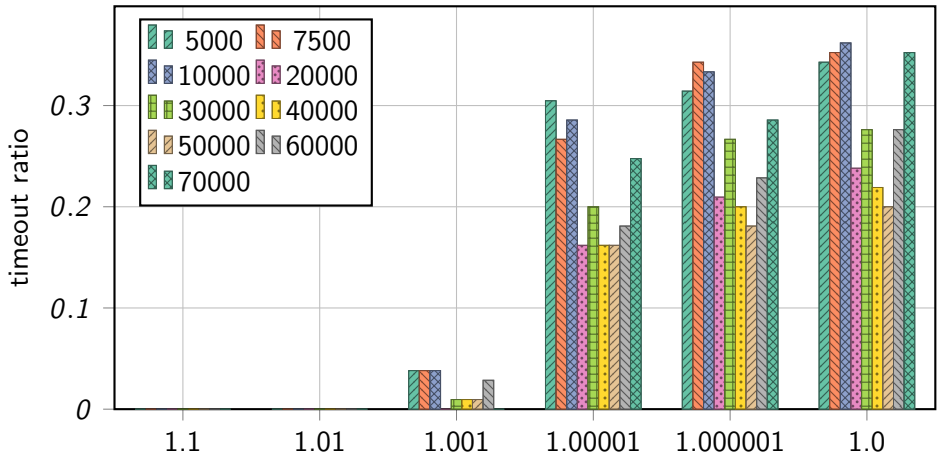
The results are less conclusive with 100 terminals. We chose to test a smaller sample of population sizes and fewer instances due to the much higher running times. Above 10000 individuals, we chose steps of 10000 in the population sizes. In fig. 6.5b we can see a high variance in running times for populations with less than 20000 individuals. With 48 seconds, the median running time for 7500 individuals was higher than for 5000 individuals with 28 seconds and also than for 10000 individuals with 35 seconds. As with 50 terminals, the median running time grows approximately linearly for 20000 or more individuals by around 1.5 seconds per 1000 individuals. The 75% quantile of running times was lowest with 50000 individuals which also had the lowest number of timeouts with 20% non-optimal results. This is despite its median running time of 75 seconds being twice as high as for 10000 individuals. With median running times closer to the timeout, at populations sizes of 60000 and 70000 the number of unsuccessful runs increased again reaching 35% with 70000 individuals.

The best choice of running time and mutation rate depends on the use case. A mutation rate between 0.01 and 0.03 worked well for both 50 and 100 terminals. We found that rates in this range offer a good trade-off between running time and variation in the form of optimally solved runs. Depending on the required solution quality one can adjust the timeout. Reducing the timeout from 90 to 20 seconds on 50 terminals would still yield around 75% optimal solutions in the aforementioned range of mutation rates with a population of just 5000. The success rate is even higher when increasing the population to 25000 or 30000 which are both above 90% at a timeout of 20 seconds with a mutation rate of 0.03.

With 100 terminals the timeout has to be set higher. After 30 seconds we found the best results at rates around 0.01 with a success rate of around 45% with a population of 5000. Until the timeout of 120 seconds this increased to 64% but 0.01 remained a good choice of mutation rate. Increasing the population size Increases



(a) 50 terminals with a timeout of 90 seconds



(b) 100 terminals with a timeout of 120 seconds

Figure 6.6: Timeout ratio before reaching a certain solution quality for different population sizes with 50 terminals, a timeout of 90 seconds and a mutation rate of 0.03.

the running time and thus at a timeout of 30 seconds using only 5000 individuals has the highest success rate. With populations above 30000 no runs reached less than $1 + 10^{-}5$ times the optimal solution. Increasing the timeout to 60 seconds we found only a small difference between the population sizes below and including 30000 which all of which found the optimal solution in around 60% of all cases. An exception to this was 20000 individuals which resulted in a success rate of 70%. With a timeout of 90 seconds this success rate was also reached with 30000 to 50000 individuals.

All in all, using a mutation rate of 0.01 showed good results with both instance sizes. Larger populations increase the solution quality but have a higher running time. To find minimum Steiner trees a population of 30000 showed good results already after a short timeout with 50 terminals. Larger population sizes up to 50000 had a higher success rate with the original timeout but performed worse at reduced timeouts.

For the classical unrestricted Euclidean Steiner Tree Problem and a few variants including the rectilinear Steiner tree problem GeoSteiner remains a much better choice. Our solution does not always find the optimal tree and, if it does, takes much more time and more processing power. The experiments do, however, show that our algorithm is capable of computing minimum Steiner trees.

6.1.3 Using Other Operators

Our operators do not necessarily change the number of Steiner points in the network. We can use this to test the behavior with a fixed number of Steiner points where Steiner nodes in antennas and path nodes are redistributed uniformly at random among the edges but never completely deleted from the Graph. We chose a total of $2\,|T|$ Steiner points where T refers to the terminals. Otherwise, the setup is the same as in section 6.1.1. With 50 terminals, the median running times are lower for all mutation rates except 0.02 and 0.03. At a mutation rate of 0.05 the median time is 5.1 seconds which is 1.2 seconds below the time with dynamic Steiner point count. The difference rises with higher rates. At a rate of 0.08 the median time is 14.8 seconds with fixed Steiner point count compared to 28.8 seconds The main reason for the different running times is the time per iteration. Not subdividing all edges could increase the time per iteration by around one third at rate 0.04 from 0.13 seconds to 0.8 seconds. With higher rates the time per iteration halved. Using a fixed number of Steiner points can therefore produce more generations in the same time, searching more potential solutions. This comes, however, at a cost. The variation is much higher, leading to about twice as many non-optimal solutions after the timeout in the range between rates 0.01 and 0.08. With the lower number of potential Steiner points exiting local minima is less likely when many

non-redundant Steiner points are present in the graph. Only few Steiner points are available for subdividing edges to provide potential Steiner point positions for edge insertion. Using the previous configuration with dynamic number of Steiner points is therefore still recommendable. Some speedup might be achieved in the dynamic configuration by only subdividing a fraction of the edges. We did not investigate further into that direction.

Separate crossover performed very poorly compared to subtree crossover. As expected, it did not preserve locally optimal structures well and optimal Steiner point positions were lost during crossover. We tested separate crossover using the same setup as in section 6.1.1 but chose separate crossover instead of subtree crossover. On 50 terminals, the algorithm never reached the optimal solution and often took more than 20 iterations to find a solution better than the initial solution. With subtree crossover, the solution quality always improved in the first iteration. The final solution always exceeded the reference cost by a factor of more than 1.05. More importantly, the final solution cost was higher than the minimum spanning tree on the terminals in around half the runs. In addition, the crossover of two parents takes around twice as long as subtree crossover. We conclude that separate crossover is not useful for geometric Steiner tree problems. Instead, we recommend subtree crossover.

Nudging point mutation also proved not to be very effective with Steiner trees. We tested both nudging mutation on its own and combined with link-length minimization. The minimization was only run before the nudging mutation and with a probability p_m which was set to the overall mutation rate. We ran both configurations on 50 terminals. With a mutation rate of 0.01, half the runs reached a cost only 0.5% higher than the minimal length with the combined mutation. This quality was never reached without link length minimization. The best median solution quality with just nudging mutation was 0.7% higher than the minimum at a mutation rate of 0.03. Other mutation rates yielded worse results. The minimal length was never reached at any mutation rate in both configurations. Nudging mutation is thus not useful with regular Steiner trees. This is, however, not very surprising. Nudging mutation is a random movement towards a neighbor while link-length minimization moves the Steiner points to their optimal positions. Nudging mutation is thus likely to increase the total length and thus decreasing the fitness.

6.1.4 Performance on Different Instance Sizes

We tested our method on Steiner tree instance sizes of up to 100 terminals with a step of 10 terminals between instance sizes. The configuration was chosen in accordance with the experiments above. We used a mutation rate of 0.01 which

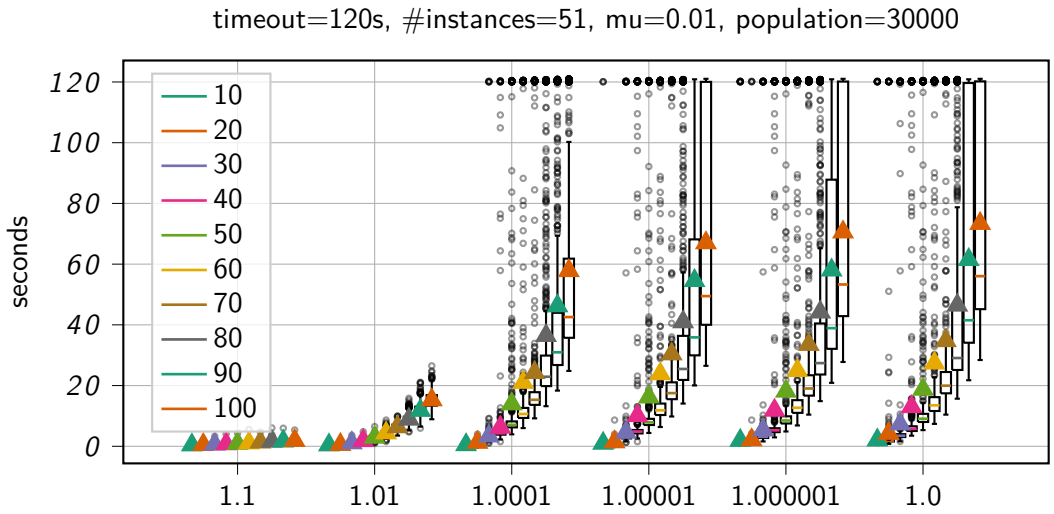


Figure 6.7: Running times on different numbers of terminals with mutation rate $p_m = 0.01$ and population 30000. We ran 51 instances and repeated each 10 times for each number of terminals with a timeout of 120s.

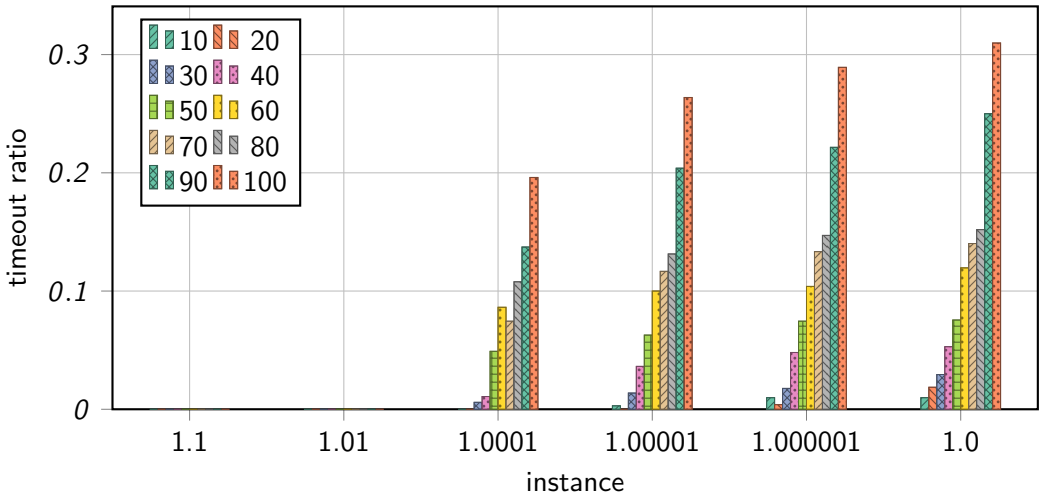


Figure 6.8: Ratio of timeouts before reaching multiples of the minimum length on instance sizes between 10 and 100 terminals. The mutation rate was $p_m = 0.01$ and the population size was 30000 for all instances.

yielded good convergence with both 50 and 100 terminals and a population size of 30000 individuals. We used the first 51 instances of each instance size and repeated them 10 times each. The running times shown in fig. 6.7 are far higher than GeoSteiner's which took on average 0.5s to solve an instance of 100 terminals. This is below the median running time of 0.9 seconds our algorithm took on 10 terminals. As stated above, the goal of this experiment was not to find a better algorithm for Steiner trees but to test whether our method would perform reasonably well.

We set a limit of 2 minutes to find an optimal solution. The median running times for the optimal solution were below that threshold for all instance sizes. Running times grew rapidly from 0.9s median doubling to 1.8s for 20 terminals, reaching 9s with 50 terminals and rising further to 56s with 100 terminals. For all instances with more than 50 terminals, the running time was around 1.4 times the running time of the next smaller instance. A slightly higher factor could be found at the 75% quantile which increased by more than 1.5 times from the previous size for all sizes until reaching the time limit on 90 terminals. The 25% quantile increased by a factor of around 1.4 like then median. This fits an exponential growth of running time, which can be expected in an NP-hard problem.

With 10 terminals, more than 99% all runs reached the optimum. The remaining runs were exclusively caused by single instance which contained an angle very close to 120° in the optimal solution. The optimal solutions found by our algorithm were thus extremely close to the optimal solution, higher by at most a factor of $1 + 10^{-6}$ in all but one runs. For all instance sizes of 50 or less terminals, more than 90% of all runs found the optimal solution. The larger instances failed more frequently, reaching a failure rate of 31% with 100 terminals.

We give three examples of the performance in fig. 6.9. The three instances were chosen randomly among the available instances of 20, 50 and 100 terminals and run once with a timeout of 120 seconds. We show the minimal length in the population after each iteration as well as the best tree found in the end. In the example, the 20-terminal instance was completed optimally after less than 8 seconds with 29 iterations. The best initial solution was already shorter than the minimum spanning tree. The optimal topology was found in the 18th iteration after which only the Steiner point positions changed. With 50 terminals it took 175 generations and 120 seconds to find the optimal topology with all Steiner points. here the initial solutions were longer than the minimum spanning tree. After the first iteration the best solution was shorter than the minimum spanning tree. With a total length of 46077 the solution is very close to the optimum from the 110th generation. This intermediate result is however not optimal and does not have all Steiner points of the optimal solution. The last missing Steiner point was inserted in generation 160 after which the Steiner point positions were still not optimal. The fittest individual only reached the minimal length as computed by GeoSteiner in the very last iteration

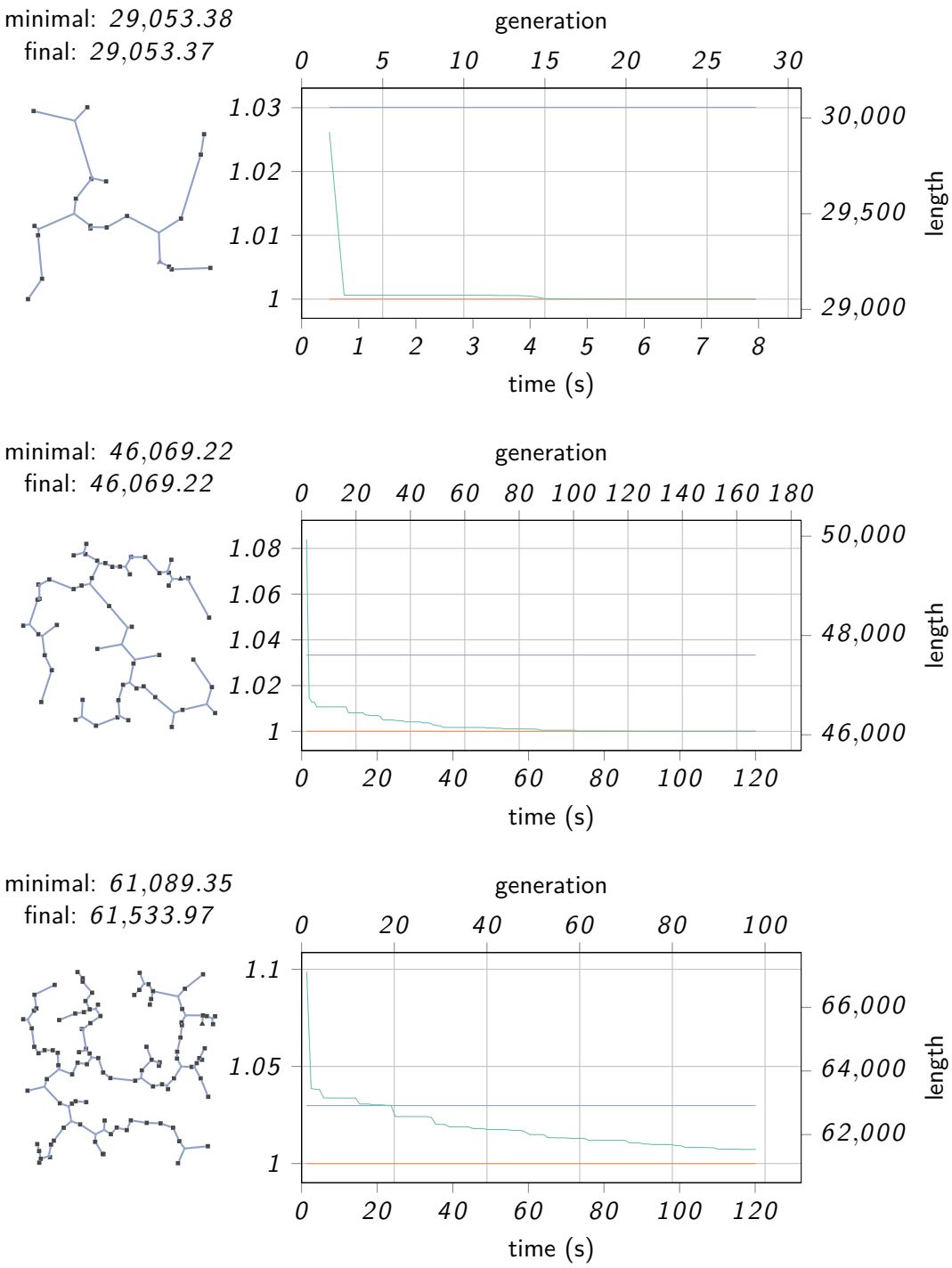


Figure 6.9: Examples of final result and solution quality over time for 20, 50 and 100 terminals. We plot the minimal length by time in green and the SMT length in orange as computed by geosteiner and the minimum spanning tree length in blue.

Table 6.1: Cable types used for planning the Idjwi site [34].

cross section [mm^2]	cost [\$/m]	ampacity [A]	line resistance [$\Omega\,km^{-1}$]
16	2.5	66	1.91
35	5	132	0.87
70	8	205	0.44
95	11.18	245	0.32

before timeout. With 100 terminals the simulation did not find the minimum Steiner tree in the example run. While the best solution surpassed the minimum spanning tree after the first iteration in the two smaller examples, here the first generation to contain a better solution than the minimum spanning tree was the 21st. After 120 seconds the best solution did not reach the minimum.

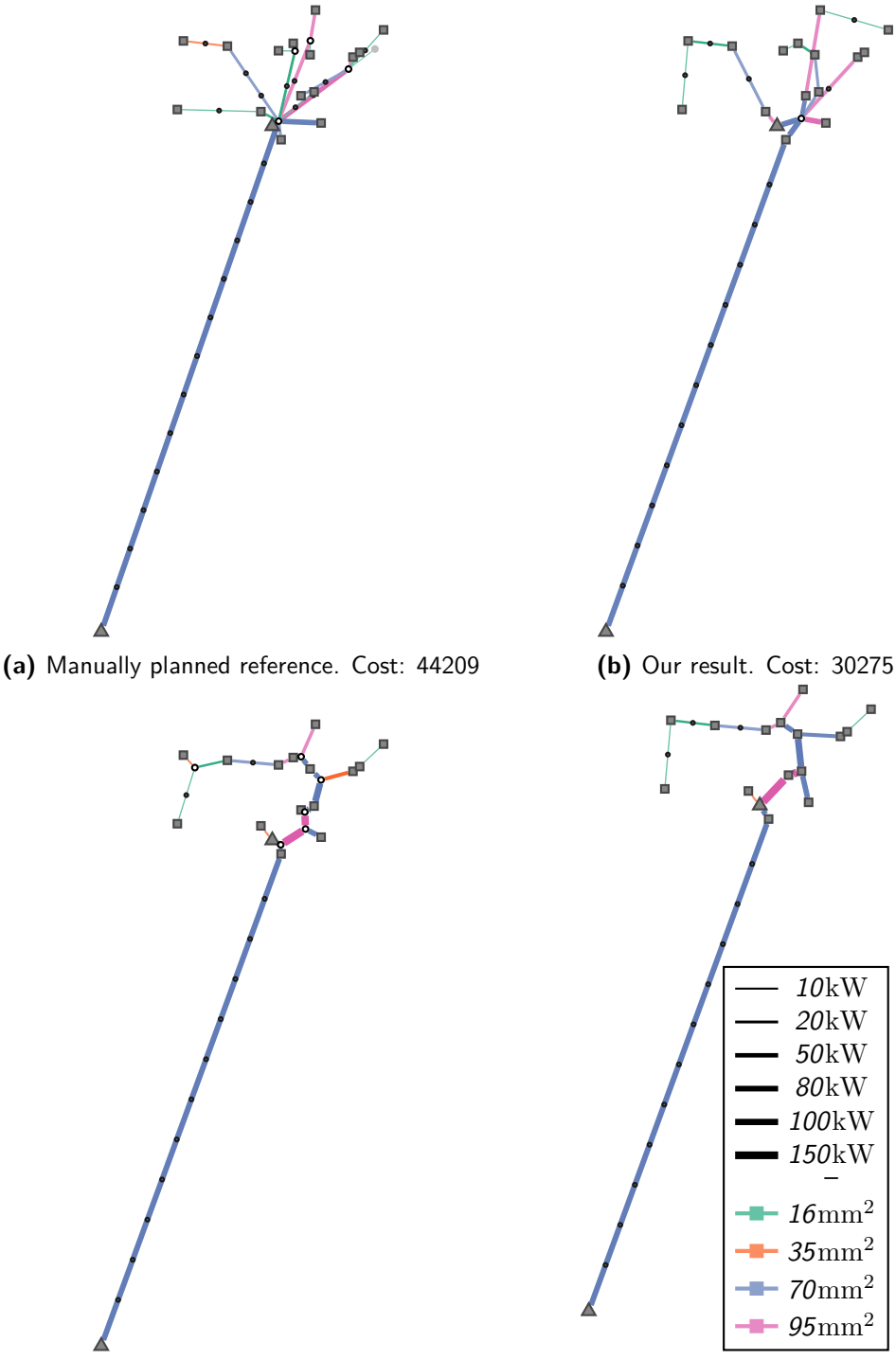
6.2 Idjwi

On the island of Idjwi in lake Kivu in the Democratic Republic of the Congo, Engineers Without Borders Karlsruhe helps to install and design a microgrid powered by renewable energies[3]. A small industrial campus is powered by a hydroelectric plant and an additional solar plant is planned to be installed in the future. We used their real-world positions and demands to compute a cable layout. We deviated from the actual site in that we included the solar plant that as of now has not been built. We further excluded the diesel generator which is planned to be replaced by the upgraded hydro plant and solar panels [34].

The grid is operated at 400 V AC line voltage at a frequency of 50 Hz. Power losses should not exceed 10% and voltage drop is also limited to 10%. We used four cable types listed in table 6.1. The cable types are based on the NFA2X overhead lines. In accordance with Kraft [34], we assume a power factor of $\cos\varphi = 0.8$. The cost of distribution nodes and poles is listed in table 6.2.

We used our algorithm to generate a grid layout for the project site. We used subtree crossover and a sequence of close edge mutation, link-length minimization and nudge mutation. The result at a mutation rate of 0.05 with a population of 50000 can be found in fig. 6.10. Our result has the lowest cost of 30275 followed by

[3] *Hydroélectricité Idjwi – Chancen durch Wasserkraft.* EWB Karlsruhe. 2015. URL: `https://ewb-karlsruhe.de/dr-congo/` (visited on 08/01/2020).



(a) Manually planned reference. Cost: 44209

(b) Our result. Cost: 30275

(c) Minimum Steiner tree. Cost: 54869

(d) Minimum Spanning tree. Cost: 50160

Figure 6.10: Different grid topologies and cable assignments for Idjwi. The generators are marked by triangular nodes. The black square nodes mark the consumers. The line thickness corresponds to the edge ampacity. Colors encode the cable types. The small black nodes along the lines indicate poles.

Table 6.2: Costs and other properties of Distribution nodes and Poles [34]

Property	Value	Unit
distribution node base cost	100	$
distribution branch cost	200	$
distribution cost per kW	30	$/kW
pole installation cost	180	$
span length	50	m

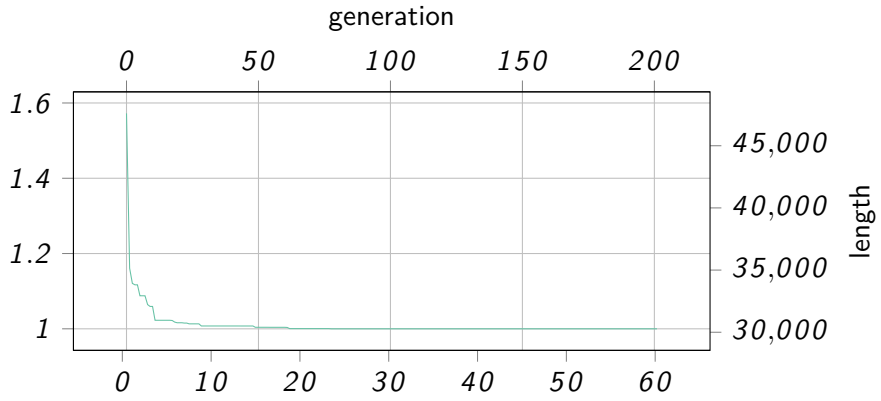


Figure 6.11: Solution quality over time for the Idjwi scenario. Truncated at 60 seconds after which the minimum cost did not decrease further.

the planned network with 44209. Both are similar in that the power line from the hydroelectric plant in the far south is first connected to a central distribution point. Both our result and the reference are similar in that they use a central distribution point connected to the generators from which several branches lead to groups of consumers. Our result uses only a single distribution point whereas the manually planned layout uses several further distribution points.

The final result was found in the 201st generation after 60 seconds. Its topology was found after 19 seconds in the 67th generation after which only the Steiner node position changed. The minimal network cost in the 67th generation was 30291. It first dropped below 30275 after 83 generations. The cost of 30275.117 was reached after 201 generations and did not drop further until reaching timeout after 288 generations.

There are several reasons for the planned network being more expensive in our model. To use a common basis for comparison we computed the cost of both layouts

using our cost model. The planning process involved factors not considered in our simplified model. For example combining the grid connections of close consumers is not possible with our model but used in several instances in the actual installation. In the distribution network not all cables are laid in straight lines but move around or along buildings. Lastly, some cables in the actual grid are routed in parallel using the same poles. Another relevant reason is that the planned layout involves existing cables used with the Diesel generator.

Observe that in our result two cables cross each other in the upper right part. Our model does not explicitly forbid such intersections. They can make sense if otherwise different cables had to be used or a distribution point had to be installed. In this instance the pink high-capacity cable would need to be redirected to either a new distribution point int he middle or through the two terminals on the left. The first option is more expensive because of the distribution node. The second option would require other cables to be used also increasing the cost. Whether crossing cables makes sense in an actual installation needs to be evaluated on a case by case basis.

With the distribution node, high line ampacities can be avoided. The necessary high-capacity cables greatly increase the cost of the minimum spanning tree fig. 6.10d and Steiner tree layouts fig. 6.10c. Both have sections where the combined power of both generators is routed through a single edge. This requires more and thicker cables and thus increases the cost to 50160 for the minimum spanning tree. The Steiner tree also places more Steiner nodes which raises the cost even further to 54896.

6.3 Influence of Voltage Drop and Power Loss

We use the longest generator-consumer path through an edge to calculate the maximum voltage drop. If we lower the tolerated voltage drop, we thus expect to see the distances between consumers and generators to go down and to see more direct paths between consumers and generators. The lower tolerated voltage drop has to be compensated for by installing more high capacity cables with lower line resistance. Lowering the power loss tolerance should lead to a shorter total grid length weighted by power capacity. Because more cables are needed on high-capacity edges, we don't expect a convergence toward the minimum Steiner tree.

Low tolerances require many more cables on each edge to meet the tolerances. Therefore, the cost of the networks is not directly comparable when using different parameters. To isolate the effect of the losses, distribution nodes did not come at a

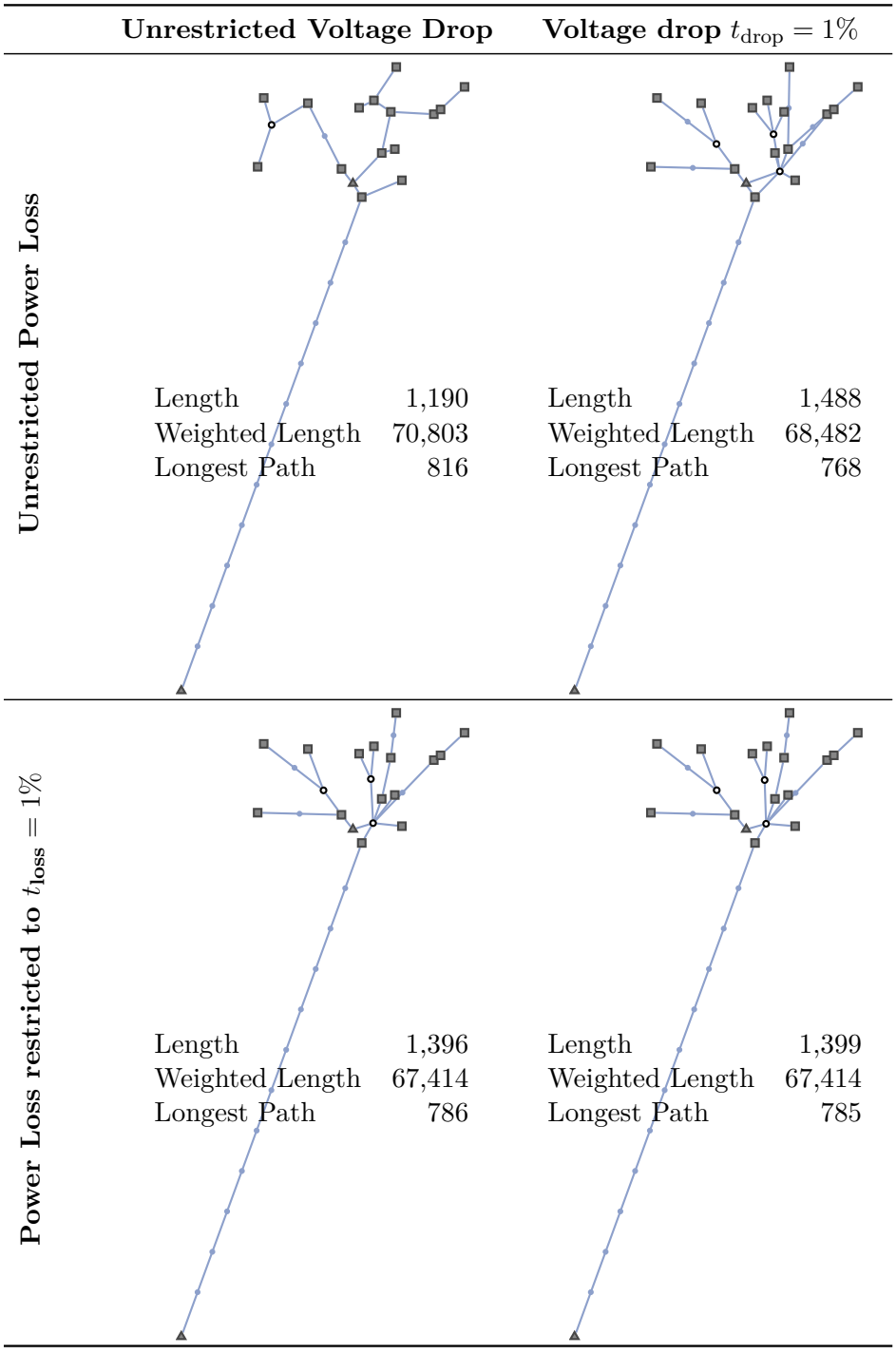
cost during this experiment.

Figure 6.12 shows the effect of restricting power loss and voltage drop to 1% each both on their own and combined. For each configuration we picked the best result out of ten runs with the same setup as above. At 1190m, total unweighted length is lowest if we do not restrict the losses. The total length is longer for restricted power and restricted voltage drop loss at 1396m and 1488m, respectively. Instead, the weighted length becomes more important. As expected, the weighted length is lowest with restricted power loss. The best result we found was almost the same for restricted power loss with both unrestricted and restricted voltage drop. The two layouts differ slightly in the placement of the central distribution node but are otherwise identical. With restricted voltage drop, the longest path is around 1m shorter. This would fit our expectation with voltage drop but, considering that both layouts are almost identical down to the assigned cables, is more likely to be a statistical artifact.

When we only restrict the voltage drop, the network becomes more centralized. Observe that the longest generator-consumer path is almost 20m shorter than with the other layouts. In contrast to the unrestricted network, the terminals are connected via a central hub and most cables are directed towards this hub. Observe that the four terminals in the upper right are not directly connected to the upper generator but to the terminal on the line to the lower generator. This reduces their distance from the furthest generator. In a grid with a single generator we would expect all consumers to be directly connected to the generator if the voltage drop tolerance is set low enough. Our example, however, uses two generators and thus minimizing the voltage drop does not necessarily yield a star topology with a single central hub. We found that if we further tighten the voltage drop tolerance, the solution converges towards the layout in fig. 6.13. In this layout, all terminals are either on the path between the two generators or have an edge to a node on that path.

In these experiments we saw that the importance of length weighted by assigned power increased with tighter restriction of power loss and voltage drop. Restricting power loss had a greater influence that overshadowed the effect of restricted voltage drop. Restricting the losses too far will not result in a realistic result. With power loss restricted to 1% several cables including the long cable to the generator in the south west required 70 cables to be installed in parallel. This is completely infeasible in a real project.

Figure 6.12: Influence of voltage drop and power loss on the network topology.



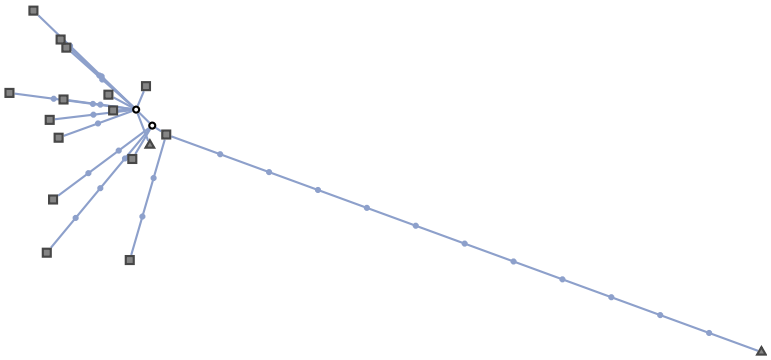


Figure 6.13: Idjwi network optimized for voltage drop (rotated by 90°)

6.4 Avoiding regions

With the region avoidance operator we can solve the geometric Steiner tree problem with convex solid obstacles. Our implementation should be seen as a proof-of-concept showing the extensibility of our genetic algorithm. We did not compare the results to an optimum solution. Our experiments did still show that with the obstacle avoidance operator our algorithm finds promising trees that do not intersect with the obstacles. Figure 6.14 shows an example with two obstacles and 8 terminals.

We ran the simulation until the optimal solution did not increase in 100 iterations. In our first scenario with 2 obstacles and 8 terminals this took 25.3 second and 123 iterations. The second scenario with 15 terminals and 5 obstacles finished in 83 seconds after 183 iterations. The convergence speed is much lower than on 20 terminals without obstacles.

Further evaluation of this technique is necessary including the performance on larger instances. One major improvement would be an extension to overlapping and non-convex obstacles which currently cannot be handled by our algorithm. Current approaches to the Euclidean Steiner Tree Problem with Obstacles handle such cases as well as soft obstacles [24, 48].

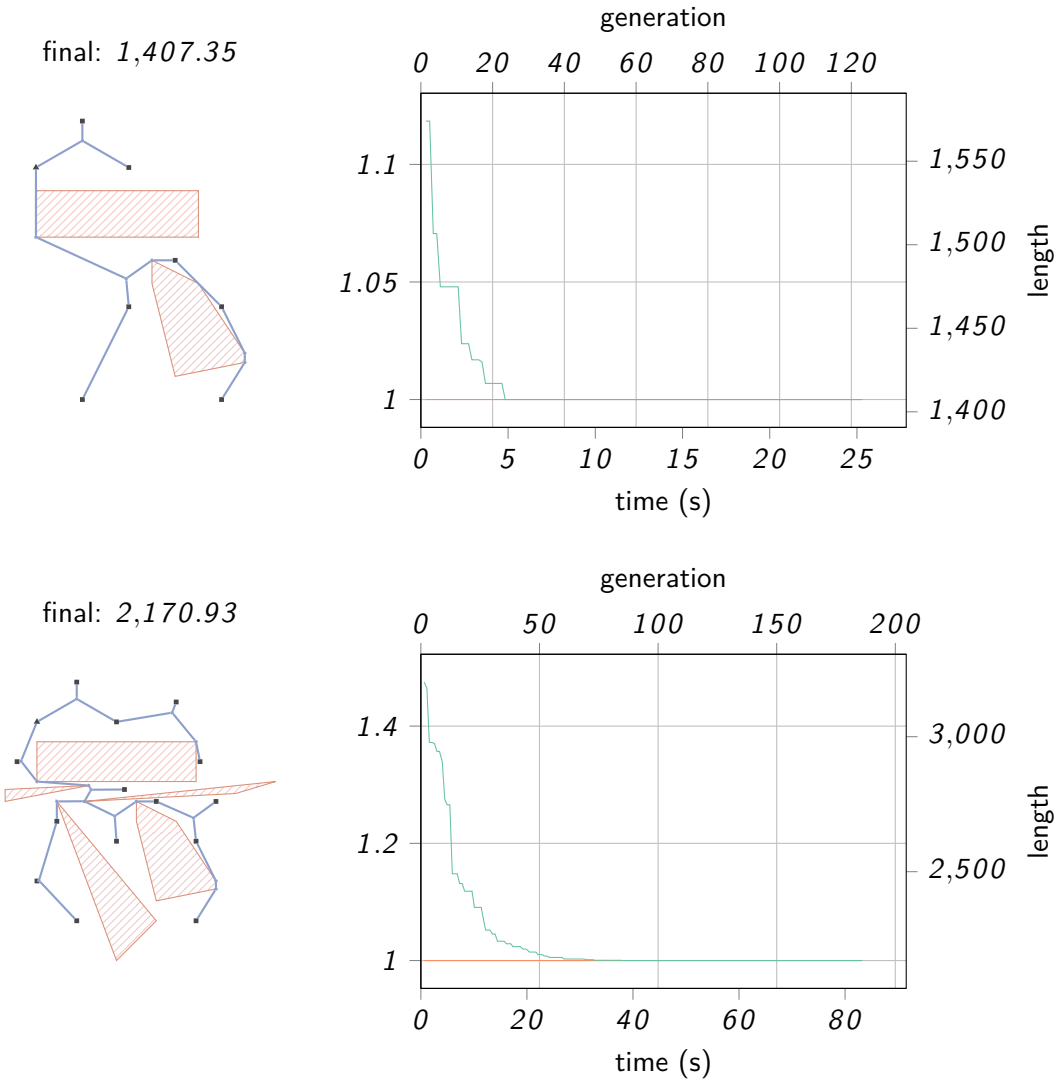


Figure 6.14: Obstacle-avoiding Euclidean Steiner tree

7 Conclusion

We presented a genetic algorithm that can be used for different problems related to Steiner trees and successfully applied it to microgrid planning. Our algorithm can compute good solutions to the Microgrid Cabling Problem in less than a minute. This is way slower than real time but may still be acceptable in the planning of such small scale layouts. Our algorithm can compute minimum Steiner trees on up to 100 terminals in less than 2 minutes.

One great advantage of using a genetic algorithm is its flexibility and extensibility. We already showed that by changing the objective function we can solve both the Steiner tree problem and Microgrid Cabling. Further extensions are possible through the introduction of new genetic mutation operators. This advantage comes at the cost of lower convergence speeds and possibly lower solution qualities than in a dedicated solution.

One hope we had with the genetic approach was to obtain multiple possible topologies. In our experiments, however, the population diversity mainly resulted from minor changes to a common topology. Providing several options can help planners to consider external factors not reflected in the model. Possible approaches could try to maintain a more diverse population to increase the likelihood of different good but non-optimal topologies to survive. Maintaining a diverse population is also advantageous to the convergence speed of the optimum solution and to escape local minima. Possible approaches to be applied to our algorithm in the future involve the selection process [37] or separate populations [42, 45]. Finding alternatives is also a problem of route planning [5]. It might be possible to use methods from alternative route planning to find good but sufficiently distinct topologies for microgrid planning. Future work on alternatives would first have to clarify the vague notion of similarity for network topologies.

Further research can be done on the cable assignment problem. Its complexity when only a single cable type is allowed remains an open question which we could not solve. Our heuristic solution is not optimal and can be replaced by a better one, possibly even an efficient optimal solution. Also a more accurate model of power loss and voltage drop could be used. This could be achieved by incorporating existing tools such as PyPSA [10].

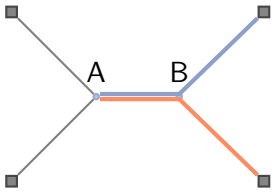


Figure 7.1: Cables laid in parallel between A and B. The blue and orange cables pass through B which is not a distribution node. Distribution equipment only needs to be installed in A.

To reduce transmission losses long distance cables are usually operated at higher voltages. Building low voltage transmission lines longer than a few kilometers results in high losses that render their operation practically infeasible. An incorporation of a transmission voltage level with transformer stations connecting to the low voltage network would be useful for planning larger scale grids.

Other limitations concern the cable layout. Our model does not differentiate cables that terminate in a node and such cables that simply pass through. Figure 7.1 shows a network with a single distribution node A with four connected cables. Two of the cables are laid in parallel to a common node B from where they follow separate routes to their destinations. This layout has the advantage that the two cables can share the poles along the way reducing the cost. Our model cannot handle parallel cables and would consider both A and B distribution nodes with expensive equipment. Parallel cables are also interesting for applications in the wind farm cabling problem. Our method is also only capable of producing a tree layout. An extension to mesh layouts might be interesting, but requires a different cable assignment strategy. Planning different topologies can also help to increase the grid resilience against cable and equipment failure and in case of maintenance works. We did not cover resilience in this thesis and in fact the network planned in section 6.2 has a single point of failure disconnecting most of the consumers from electric supply. This vulnerability is characteristic of radial networks. Relaxing the requirements on grid topology can be particularly useful in extending existing grids.

The algorithm performed better on smaller instance sizes. Splitting large instances and using a divide and conquer approach might improve the solution quality and performance. This could be done by separating the instances along a spanning tree, running the genetic algorithm separately on the subinstances and rejoining them. If different spanning trees are used this approach could help explore the possible solution more efficiently than our current algorithm.

In a real world application, one might want to mark exclusion zones where no cables

or poles may be placed. We provide a proof-of-concept that our method can be used with such zones but further research is needed here. Our method does not handle non-convex obstacles. The cost of installing the cable can also differ locally. Placing a pole close to a road is easier and thus cheaper than in hard to reach terrain. This can be accounted for by using soft obstacles that can be passed at a certain cost instead of impassable hard obstacles.

Bibliography

[1] *11th DIMACS implemenation challenge*. Center for Discrete Mathematics and Computer Science. URL: http://dimacs11.zib.de/downloads.html (visited on 07/12/2020).

[2] Aboriginal Affairs and Northern Development Canada (AANDC) and Natural Resources Canada (NRCan). *Status of Remote/Off-Grid Communities in Canada*. Report. 2011. URL: https://www.nrcan.gc.ca/sites/www.nrcan.gc.ca/files/canmetenergy/files/pubs/2013-118_en.pdf (visited on 08/21/2021).

[3] Alex M Andrew. "Another efficient algorithm for convex hulls in two dimensions." In: *Information Processing Letters* 9.5 (1979), pp. 216–219.

[4] Thomas Bäck and Frank Hoffmeister. "Extended selection mechanisms in genetic algorithms." In: (1991).

[5] Roland Bader et al. "Alternative route graphs in road networks." In: *International Conference on Theory and Practice of Algorithms in (Computer) Systems*. Springer. 2011, pp. 21–32.

[6] Jorge Barreiros. "An hierarchic genetic algorithm for computing (near) optimal Euclidean Steiner trees." In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*. Citeseer. 2003, pp. 56–65.

[7] Norbert Beckmann et al. "The R*-tree: An efficient and robust access method for points and rectangles." In: *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*. 1990, pp. 322–331.

[8] Jon Louis Bentley and Thomas A Ottmann. "Algorithms for reporting and counting geometric intersections." In: *IEEE Transactions on computers* 28.09 (1979), pp. 643–647.

[9] Marcus Brazil et al. "On the history of the Euclidean Steiner tree problem." In: *Archive for history of exact sciences* 68.3 (2014), pp. 327–354.

[10] T. Brown, J. Hörsch, and D. Schlachtberger. "PyPSA: Python for Power System Analysis." In: *Journal of Open Research Software* 6.4 (1 2018). DOI: 10.5334/jors.188. eprint: 1707.09913. URL: https://doi.org/10.5334/jors.188.

[11] Siu-Wing Cheng et al. *Delaunay mesh generation*. CRC Press Boca Raton, FL, 2013.

[12] Wilson Wolf Costa et al. "Application of Enhanced Particle Swarm Optimization in Euclidean Steiner Tree Problem Solving in RN." In: *Computational Intelligence, Optimization and Inverse Problems with Applications in Engineering*. Ed. by Gustavo Mendes Platt, Xin-She Yang, and Antônio José Silva Neto. Cham: Springer International Publishing, 2019, pp. 63–85. ISBN: 978-3-319-96433-1. DOI: 10.1007/978-3-319-96433-1_4. URL: https://doi.org/10.1007/978-3-319-96433-1_4.

[13] Jesús De Loera, Jörg Rambau, and Francisco Santos. *Triangulations: structures for algorithms and applications*. Vol. 25. Springer Science & Business Media, 2010.

[14] Reinhard Diestel. "The Basics." In: *Graph Theory*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 1–34. ISBN: 978-3-662-53622-3. DOI: 10.1007/978-3-662-53622-3_1. URL: https://doi.org/10.1007/978-3-662-53622-3_1.

[15] *Merkmale der Spannung in öffentlichen Elektrizitätsversorgungsnetzen; Deutsche Fassung EN 50160:2010 + Cor.:2010 + A1:2015 + A2:2019 + A3:2019*. Nov. 2020.

[16] *Starkstromkabel. Isolierte Freileitungsseile für oberirdische Verteilungsnetze mit Nennspannungen 0,6/1 (1,2) kV; Deutsche Fassung HD 626 S1 Teile 1, 2 und 4F-1:1996*. Jan. 1997.

[17] *Verwendung von Kabeln und isolierten Leitungen für Starkstromanlagen. Teil 4: Empfohlene Werte für die Strombelastbarkeit von Kabeln und Leitungen für feste Verlegung in und an Gebäuden und von flexiblen Leitungen*. June 2013.

[18] Sudipta Dutta and Thomas Overbye. "A graph-theoretic approach for addressing trenching constraints in wind farm collector system design." In: *2013 IEEE Power and Energy Conference at Illinois (PECI)*. IEEE. 2013, pp. 48–52.

[19] Patrik Fagerfjäll. "Optimizing wind farm layout: more bang for the buck using mixed integer linear programming." In: *Chalmers University of Technology and Gothenburg University* (2010), p. 111.

[20] Martina Fischetti and David Pisinger. "Optimizing wind farm cable routing considering power losses." In: *European Journal of Operational Research* 270.3 (2018), pp. 917–930.

[21] Steven Fortune. "A sweepline algorithm for Voronoi diagrams." In: *Algorithmica* 2.1 (1987), pp. 153–174.

[22] Ian Frommer and Bruce Golden. "A genetic algorithm for solving the Euclidean Non-Uniform Steiner Tree problem." In: *Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies*. Springer, 2007, pp. 31–48.

[23] Michael R Garey, Ronald L Graham, and David S Johnson. "The complexity of computing Steiner minimal trees." In: *SIAM journal on applied mathematics* 32.4 (1977), pp. 835–859.

[24] Luis Garrote et al. "Weighted Euclidean Steiner trees for disaster-aware network design." In: *2019 15th International Conference on the Design of Reliable Communication Networks (DRCN)*. IEEE. 2019, pp. 138–145.

[25] Ronald L. Graham. "An efficient algorithm for determining the convex hull of a finite planar set." In: *Info. Pro. Lett.* 1 (1972), pp. 132–133.

[26] Sascha Gritzbach, Dorothea Wagner, and Matthias Wolf. "Negative Cycle Canceling with Neighborhood Heuristics for the Wind Farm Cabling Problem." In: *Proceedings of the Eleventh ACM International Conference on Future Energy Systems*. 2020, pp. 299–307.

[27] Marinus OW Grond et al. "Practice-oriented optimization of distribution network planning using metaheuristic algorithms." In: *2014 Power Systems Computation Conference*. IEEE. 2014, pp. 1–8.

[28] *Hydroélectricité Idjwi – Chancen durch Wasserkraft*. EWB Karlsruhe. 2015. URL: `https://ewb-karlsruhe.de/dr-congo/` (visited on 08/01/2020).

[29] IEA, IRENA, UNSD, World Bank, and WHO. *Tracking SDG7: The Energy Progress Report*. World Bank. Washington DC., 2021. URL: `https://trackingsdg7.esmap.org/data/files/download-documents/2021_tracking_sdg7_report.pdf` (visited on 08/21/2021).

[30] Mário Jesus, Sérgio Jesus, and Alberto Márquez. *Steiner trees optimization using genetic algorithms*. Tech. rep. Technical report, Centro de Simulação e Cálculo, 2004.

[31] Daniel Juhl et al. "The GeoSteiner software package for computing Steiner trees in the plane: an updated computational study." In: *Mathematical Programming Computation* 10.4 (2018), pp. 487–532.

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

[33] Mohammad E Khodayar. "Rural electrification and expansion planning of off-grid microgrids." In: *The Electricity Journal* 30.4 (2017), pp. 68–74.

[34] Johann Kraft. "Untersuchung der Kombination von Solar und Wasserkraft zur Off-Grid-Stromversorgung." Fakultät für Elektrotechnik und Informationstechnik. Bachelorarbeit. Karlsruher Intsitut für Technologie, 2020.

[35] Joseph B Kruskal. "On the shortest spanning subtree of a graph and the traveling salesman problem." In: *Proceedings of the American Mathematical society* 7.1 (1956), pp. 48–50.

[36] Harold W Kuhn and Robert E Kuenne. "An efficient algorithm for the numerical solution of the generalized Weber problem in spatial economics." In: *Journal of Regional Science* 4.2 (1962), pp. 21–33. DOI: https://doi.org/10.1111/j.1467-9787.1962.tb00902.x.

[37] Ting Kuo and Shu-Yuen Hwang. "Using disruptive selection to maintain diversity in genetic algorithms." In: *Applied Intelligence* 7.3 (1997), pp. 257–267.

[38] Taewon Lee and Sukho Lee. "OMT: Overlap Minimizing Top-down Bulk Loading Algorithm for R-tree." In: *CAISE Short paper proceedings*. Vol. 74. 2003, pp. 69–72.

[39] Hoang N Luong et al. "Medium-voltage distribution network expansion planning with gene-pool optimal mixing evolutionary algorithms." In: *International Conference on Artificial Evolution (Evolution Artificielle)*. Springer. 2013, pp. 93–105.

[40] William Miehle. "Link-length minimization in networks." In: *Operations research* 6.2 (1958), pp. 232–243.

[41] Riham Moharam and Ehab Morsy. "Genetic algorithms to balanced tree structures in graphs." In: *Swarm and Evolutionary Computation* 32 (2017), pp. 132–139.

[42] Jason Morrison and Franz Oppacher. "Maintaining genetic diversity in genetic algorithms through co-evolution." In: *Conference of the Canadian Society for Computational Studies of Intelligence*. Springer. 1998, pp. 128–138.

[43] Christos H Papadimitriou. "The complexity of the capacitated tree problem." In: *Networks* 8.3 (1978), pp. 217–230.

[44] AC Pillai et al. "Offshore wind farm electrical cable layout optimization." In: *Engineering Optimization* 47.12 (2015), pp. 1689–1708.

[45] David Power, Conor Ryan, and R Muhammad Atif Azad. "Promoting diversity using migration strategies in distributed genetic algorithms." In: *2005 IEEE Congress on Evolutionary Computation*. Vol. 2. IEEE. 2005, pp. 1831–1838.

[46] Robert Clay Prim. "Shortest connection networks and some generalizations." In: *The Bell System Technical Journal* 36.6 (1957), pp. 1389–1401.

[47] Jan Remy and Angelika Steger. "Approximation schemes for node-weighted geometric Steiner tree problems." In: *Algorithmica* 55.1 (2009), pp. 240–267.

[48] Manou Rosenberg et al. "A genetic algorithm approach for the Euclidean Steiner tree problem with soft obstacles." In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2021, pp. 618–626.

[49] Adolf J Schwab. *Elektroenergiesysteme: Erzeugung, Übertragung und Verteilung elektrischer Energie*. Springer-Verlag, 2017.

[50] SN Sivanandam and SN Deepa. "Genetic algorithms." In: *Introduction to genetic algorithms*. Springer, 2008, pp. 15–37.

[51] Robert E Tarjan and Jan Van Leeuwen. "Worst-case analysis of set union algorithms." In: *Journal of the ACM (JACM)* 31.2 (1984), pp. 245–281.

[52] Endre Weiszfeld. "Sur le point pour lequel la somme des distances de n points donnés est minimum." In: *Tohoku Mathematical Journal, First Series* 43 (1937), pp. 355–386.

[53] Martin Zachariasen and Pawel Winter. "Obstacle-avoiding Euclidean Steiner trees in the plane: an exact algorithm." In: *Workshop on Algorithm Engineering and Experimentation*. Springer. 1999, pp. 286–299.