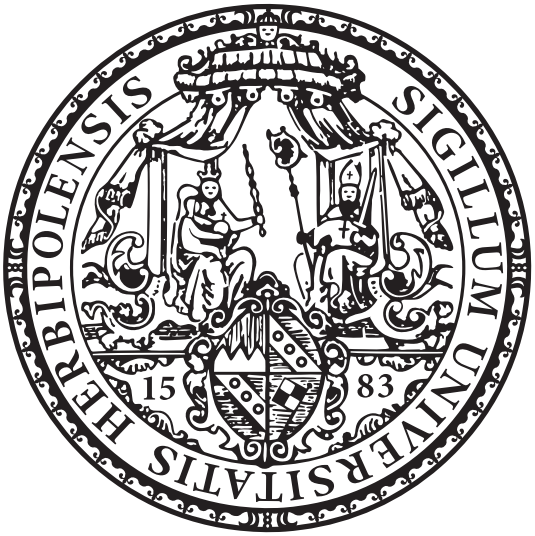
Bachelor Thesis

Infrastructure Planning via Algorithms for the Weighted Region Problem

Samuel Wolf

Date of Submission: July 21, 2021
Advisors: Prof. Dr. Alexander Wolff
Dr. Alex Ravsky (National Academy of Sciences of Ukraine)
Salvador Bayarri (Gilytics)



Julius-Maximilians-Universität Würzburg
Lehrstuhl für Informatik I
Algorithmen und Komplexität

Abstract

Infrastructure planning aims to minimize the impact of large structures such as pipes, power lines or cables on society, environment, and investors. In the process of designing these structures, finding a minimum-cost path via graph theory is a key aspect. We challenge the ubiquitous usage of rasterized data in infrastructure planning for finding such paths and propose a new approach to identify minimum-cost paths. For this purpose, we accelerate the performance of an existing algorithm that approximates shortest paths in weighted polygonal input and combine it with a method for minimizing turn angles. Moreover, for the application on transmission line routing, we offer an algorithm to place pylons along a given path, with respect to given constraints and costs. Finally, we conduct an experimental analysis of our novel approach and compare it to previous work operating on raster-based input. Our approximation matches the performance in terms of cost for rasters at high resolution and is significantly faster.

Zusammenfassung

In der Infrastrukturplanung geht es darum, die Kosten großer Netzwerke wie Straßen-, Wasser- oder Stromnetze für Auftraggeber und Umwelt zu minimieren. Dabei ist das Finden eines kostengünstigsten Pfades mithilfe der Graphentheorie ein maßgeblicher Bestandteil der Gestaltung solcher Strukturen. Wir stellen die allgegenwärtige Verwendung von gerasterten Geodaten zum Finden dieser Wege in Frage und schlagen einen alternativen vektorbasierten Ansatz vor. Dazu beschleunigen wir einen vorhandenen Algorithmus, der kürzeste Wege in einem gewichteten Polygonnetz approximiert und kombinieren ihn mit einer Methode zur Minimierung von Winkeln an Kurven. Darüber hinaus entwickeln wir einen Algorithmus, der unter Berücksichtigung von Kosten und Beschränkungen auf einem gegebenen Pfad Masten für Stromtrassen platziert. Ferner führen wir eine experimentelle Analyse unseres neuen Ansatzes durch und vergleichen diesen mit raster-basierten Lösungen. Die Auswertung zeigt, dass unsere Methode ähnliche Ergebnisse hinsichtlich der Kosten erzielt, aber signifikant weniger Zeit benötigt.

Contents

1 Introduction

Infrastructure planning is as old as civilization itself. Already in ancient Rome, intricate networks of aqueducts and roads were necessary. Constructing this infrastructure required careful planning in which many environmental factors such as ground conditions, slopes, rivers, and mountains had to be taken into account. Nowadays, the list of aspects that need to be considered has grown even longer with ecological, economic, and social constraints with cities and infrastructure in continuous flux. Climate change set the construction of renewable power plants such as wind, hydro or solar power in motion, making power infrastructure planning even more relevant than ever, as new power plants have to be connected in a multi-million dollar endeavor including various investors and stakeholders.

For example, in 2016 the German government adopted the Climate Action Plan 2050 that outlines measures to vastly cut down greenhouse gas emissions. In order to achieve this goal, the restructuring of the energy infrastructure from coal-fired and nuclear power plants to wind power is necessary. As a consequence, electricity produced by wind turbines in windy regions of Northern Germany have to be distributed to the South via large transmission lines. A project that tackles the construction of such new power lines is "SuedLink" by TenneT TSO and TransnetBW. In this project, two transmission lines from Wilster to Bergrheinfeld/West and from Brunsbüttel to Großgartach are planned. These long transmission lines cross highly populated parts of Germany, making it important to reduce cost and environmental impact.

Finding minimum-cost paths in geographical information is one of the core facets of infrastructure planning and is crucial in the design of power infrastructure. On top of identifying such paths, specific challenges arise in this field, including the minimization of angles between pylons for transmission lines.

Previous work and related work. As there is a high demand for software to aid the infrastructure design process, extensive work has been done. In practice, almost all path-finding algorithms in this context operate on a raster-based discretization of geographical data, similar to an image, where each cell, i.e., pixel, has as certain cost value assigned to it. This grid is used for a graph-based approach, in which each cell is converted to a vertex and an edge between two vertices constitutes a way to go from one cell to another. This approach is applied by Seegmiller et al. [SST21] to find least-cost corridors – paths with a constant width, useful for modeling roads. More specific to power infrastructure design is the work of Hanssen et al. [HTMB12], which provides a toolbox for identifying optimal routing corridors of high voltage power lines under various criteria. Bachmann et al. [BBK$^+$18] back the difficult decision process with a tool-chain for a client-server based Pareto-optimization system for power line routing that utilizes

a generalized form of Dijkstra's algorithm to generate a set of Pareto-optimal routes. Piveteau et al. [PSRW18] also take technical data into account, i.e., what *type* of pylon is needed depending on the soil and other factors. Similarly, Santos et al. [SdLP19] propose a multi-step approach to find shortest paths in raster-based input and an optimal pylon placement that finds an ideal distribution of pylons along the topographical profile of the shortest path.

Raster-based input data is considered the state of the art in this field and is being employed by all works mentioned. This approach, however, suffers from some problems inherent to rasters:

(1) A high resolution of the raster is indispensable for finding satisfying solutions, making the search space extremely large such that some real world instances are not feasible to compute. Moreover, a homogeneous cell size is fundamental to this approach, which leads to uninteresting areas having the same resolution as important areas, inflating the input size unnecessarily.

(2) A grid-based approach restricts the freedom of movement since a cell is only connected to a small number of neighboring cells, each having a predetermined position, which might be sub-optimal, leading to undesirable zig-zags, hindering the minimization of turn angles or other factors. Also, the predetermined position of the raster cells restricts the freedom of possible pylon placements since often pylons are only placed in the center of the cells.

One might be tempted to evade the problems of disadvantage (2) by a high resolution and dense connection between vertices. However, zig-zags can inherently not be avoided in rasters. Nonetheless, Tomlin [Tom10] tries to mitigate this problem by introducing virtual "threads" that keep track of each change of direction where each change is a new source of propagation.

These problems led Gonçalves et al. [GBMS21] to the idea of using a Monte Carlo algorithm called random rapid trees that does not discretize the input plane but picks points randomly on the plane in a certain proximity of already sampled points and connecting the closest points to a tree with the starting point as root. This tree is then iteratively improved by changing edges and choosing new points. Their algorithm also takes the maximum curvature of the path into account but it cannot guarantee any upper bound on the quality of the output.

A further possible realization of the input map is a set of polygons, each representing a different cost region. Many algorithms exist for the special case of robot path planning, where polygons merely serve as obstacles. Here we observe the strong connection between infrastructure planning and robotics: when constructing a route, certain regions must be avoided, too. The key idea is that a shortest Euclidean path consists of straight line segments and only bends on vertices of obstacle polygons. One way of using this insight is to use a *visibility graph* whose vertices are the corners of polygons including start and end point. Two vertices are connected by an edge if and only if a straight line between these vertices is not intersected by any polygon. The weight of an edge is the

Euclidean distance between those vertices. On this visibility graph a traditional shortest path algorithm such as Dijkstra's can be applied. In the worst case the construction of such a graph takes $\mathcal{O}(n^2)$ time, where n is the number of vertices of all polygons. Following a separate idea, Mitchell and Papadimitriou [MMP87] proposed a method, called *continuous Dijkstra*, that avoids building such graphs. Initially, this strategy had a worse runtime, namely $\mathcal{O}(n^2 \log n)$, but this was later improved by Hershberger and Suri [HS97], who gave an $\mathcal{O}(n \log n)$ time and space algorithm, proving this method to be more fruitful. The core concept is analogous to tossing a stone into a pond, creating waves that collide with, for instance, water lilies or rocks, which reflect these waves thereby becoming the source of the reflected waves. The moment a wave hits the end point the shortest route can be deduced by tracing back the points of reflection. The interested reader is referred to the work of Hershberger and Suri [HS97] to see how this idea can be efficiently simulated.

However, infrastructure planning is more complex than avoiding certain areas, since many cost factors have to be taken into consideration. A natural generalization is to assign each polygon a weight that embodies certain costs when traversing it. This leads to the *Weigthed Region Problem*, which asks for a shortest path in a planar polygonal subdivision. For convenience, polygons are usually assumed to be triangulated. Solving this problem exactly appears to be very hard. In fact, according to Carufel et al. [DGM$^+$14], it cannot be solved in a finite sequence of operations such as $+$, $-$, $\cdot$, $/$, $\sqrt[k]{\ }$ on rational numbers. As of now, no exact algorithm for solving the Weighted Region Problem has been devised, and it is unknown whether this problem is NP-hard or not. As a consequence, all published algorithms are approximations. By modifying the continuous Dijkstra method, Mitchell and Papadimitriou [MP91] proposed a $(1+\varepsilon)$-approximation algorithm for any $\varepsilon > 0$. Unfortunately, the running time is $\mathcal{O}(n^8 \log(nNW/\varepsilon))$, where W is the maximum weight and N is the largest coordinate of a polygon vertex, therefore rendering the algorithm unpractical.

Other approximation algorithms are based on discretizing the problem similar to visibility graphs by placing so-called *Steiner points* on polygon edges. Two Steiner points are then connected with an edge if a straight-line segment between them is completely contained inside the face of a triangle. A normal shortest path algorithm is subsequently applied on the resulting weighted graph. Algorithms that employ this technique mostly differ in the way they place these Steiner points. Newer versions also reduce the runtime by avoiding to build the whole graph explicitly. Cheng et al. [CNVW08] give a $(1+\varepsilon)$-approximation algorithm with $\mathcal{O}(((W \log W)/\varepsilon)n^3 \log(Wn/\varepsilon))$ runtime. The core concept is to place Steiner points in an elliptic region whose foci are the start and endpoint. Aleksandrov et al. [AMS05] achieve a $(1+\varepsilon)$-approximation with a running time of $\mathcal{O}((n/\sqrt{\varepsilon}) \log(n/\varepsilon) \log(1/\varepsilon))$. Unfortunately, there is a hidden constant, $\mathcal{O}(\Gamma \log(W/\theta_{\min}))$, where Γ is the average of the reciprocals of the sines of the angles of the triangulation and $\theta_{\min}$ is the minimum angle of the triangulation. Noteworthy is that their strategy involves the placement of Steiner points *inside* triangles, while the other approaches mentioned above only place Steiner points on the edges of the triangulated input. An alternative geometry dependent approximation is due to Cheng et al. [CJV15] with a runtime of $\mathcal{O}((kn + k^4 \log(k/\varepsilon)/\varepsilon) \log^2(Wn/\varepsilon))$, where $k \in \Theta(n)$ is the smallest

integer such that the sum of the k smallest angles in the triangular faces is at least π.

Contribution. We contest the state of the art that is using raster-based input in infrastructure design and lay the groundwork for a promising alternative to raster discretizations. Our contribution is as follows:

- We describe a fast and simple implementation of a variant of the Steiner point approach by Lanthier et al. [LMS01] for approximating shortest paths with respect to the Weighted Region Problem.
- We device an algorithm for optimal *pylon spotting* for transmission line routing along a given, fixed path.
- We extend the algorithm of Lanthier et al. [LMS01] to a) boost performance on large instances by employing the A* algorithm as well as the creation of the search graph only in areas A* explores and b) minimize angles along the path in addition to the polygon costs.
- We do an experimental analysis of the proposed algorithm on randomly generated data and compare our algorithm with LION, a raster based approach by Wiedemann and Adjiashvili [WA21] on comparable data. We can show that our method yields paths at least as cost-efficient as LION, but our method runs up to five times faster.

Organization. We start by formalizing some terms and by introducing new definitions and background that we will need later on in Chapter 2. Since the Steiner point method by Lanthier et al. [LMS01] is an important basis of our work, we will describe it in more detail; see Chapter 3. After that, we propose an algorithm to place pylons on a path given by the Steiner point method in Chapter 4 and describe implementation specifics in Chapter 5. Finally, we do an extensive experimental analysis of both methods in Chapter 6 and conclude our work with next steps one could take in Chapter 7.

2 Preliminaries

We start with a definition of the Weighted Region Problem which we have already mentioned before.

Definition 1 (Weighted Region Problem [MP91])**.** *Let S be a subdivision of the plane $\mathbb{R}^2$ into a finite number of polygonal regions, where each region has a positive weight $w \in \mathbb{R}_0^+$. Let $s, t \in \mathbb{R}^2$ be points in the plane. Find a weighted shortest path from s to t, where the distance of a path is defined to be the weighted sum of all sub-paths within each region crossed by the path. If a sub-path coincides with an edge of two polygons, the minimum weight of the weights of the two adjacent regions is chosen.*

For convenience, we assume without loss of generality that all our polygons are triangles and that the points s and t are vertices of triangles. We call this input a *triangulation*. In the following we represent a triangulation T of the plane as an undirected, planar graph $G = (V, E)$ whose vertices (edges) are vertices (edges) of T. We write $\{u, v\} \in E$ for an undirected edge and (u, v) for a directed edge. If it is clear whether an edge is directed or undirected, we shorten the notation to simply uv to denote an edge with end points u and v. Furthermore, we will use the terms minimum-cost path and shortest path interchangeably since we always mean paths in weighted regions. For two points s and t in the plane, we denote the Euclidean distance from s to t by $d(s, t)$ and write $|L| = \max_{\{u,v\} \in E} d(u, v)$ for the length of the longest edge in our triangulation. If not specified otherwise, s is the start or source vertex and t is the end or target vertex for which we try to find the shortest s–t path. We denote the total cost of an optimal s–t path by $c(P^\star(s, t))$ and the cost of an approximated s–t path by $c(P(s, t))$. Since we only speak of s–t paths, we will also shorten this notation to $c(P^\star)$ and $c(P)$, respectively, unless we talk about paths with other start or end points.

We now define the concept of a *line graph*.

Definition 2 (Line Graph)**.** *Given an undirected graph $G = (V, E)$, the* line graph *$L(G) = (V', E')$ of G is a graph such that*

1. *$V' = E$, that is, each edge in G is a vertex in $L(G)$*
2. *two vertices $e, f \in V'$ are adjacent in $L(G)$ if and only if the corresponding edges in G are incident to the same vertex*

Figure 2.1 shows an example of the line graph of K_4, the complete graph with four vertices.

We also introduce a generalization of Dijkstra's algorithm, called A*, devised by Hart et al. [HNR68]. This algorithm can achieve a more directed search towards the target t by

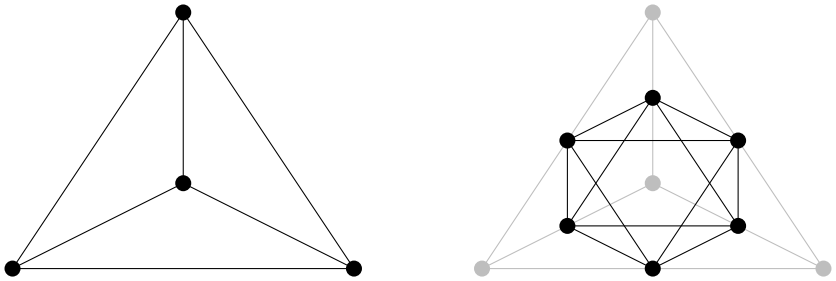


Fig. 2.1: Line graph (right) of the K_4 (left). The original graph has six edges, so the line graph has six vertices. Each vertex of $L(K_4)$ is adjacent to four other vertices since each edge of K_4 is incident to four other edges.

modifying the priority of a discovered vertex v with information of the problem domain. This more goal-oriented search is done by adding a heuristic $h(v)$ to the priority of vertex v, i.e., the priority $f(v)$ is computed by $f(v) = g(v) + h(v)$ where $g(v)$ are the current costs from s to v. Note, that by setting $h(v) = 0$ we obtain Dijkstra's algorithm in which we always pick the vertex with the currently lowest cost $g(v)$ regardless of the "direction" we are heading. However, if the heuristic $h(v)$ is a good estimation of the costs from v to t, then vertices that have a low cost estimation $h(v)$ are preferred while exploring the graph leading to a bias "towards" t. Hart et al. prove the following about the heuristic $h(v)$:

Lemma 3 ([HNR68])**.** *If the estimation $h(v)$ is a lower bound for the actual costs from v to the target t, $A^\star$ yields a shortest path from s to t.*

We also point out that A* usually terminates, once the target t has been chosen as vertex with lowest priority – in contrast to Dijkstra's algorithm, where we often aim to find a shortest path tree to all other vertices from source s.

3 Steiner Point Method

In this chapter, we discuss an approximation algorithm with respect to the Weighted Region Problem proposed by Lanthier et al. [LMS01] called the Steiner point method. The name stems from computational geometry, where Steiner points are points that are added during the solution to obtain better results.

A simple approximation. Before analyzing the Steiner point method, we consider a straightforward estimation algorithm for finding a shortest path in a weighted region represented by a graph $G = (V, E)$ without altering the input. We can approximate a shortest path by only walking along the edges of the input triangles, instead of trying to traverse the interior. For each triangle edge $\{u, v\}$ we compute the weighted distance in $\mathcal{O}(|E|)$ total time. Since we demand that our source s and our target t are triangle vertices, we can run a shortest path algorithm such as Dijkstra's with no further modifications in $\mathcal{O}(|V| \log(|V|) + |E|)$ time given an appropriate data structure. In consideration of our planar input graph we can omit the summand $\mathcal{O}(|E|)$ by bounding it via Euler's polyhedron formula $|E| \leq 3|V| - 6$. Furthermore, Lanthier et al. [LMS01] show that the algorithm described above provides an approximation of a minimum-cost s–t path with an approximation factor of at most $2/\sin\theta_{\min}$, where $\theta_{\min}$ is the minimum interior angle of an input triangle.

3.1 Approximating Snell's Law

While this first strategy is fast, the approximation bound is not practical since real world inputs can have an angle $\theta_{\min}$ very close to zero, rendering the bound very large. Therefore, the need for a better approximation arises. The algorithm of Lanthier et al. [LMS01] is based on an important property of shortest paths in weighted regions observed by Mitchell and Papadimitriou [MP91]: An optimal path only bends at edges of regions, where the two adjacent regions are of unequal weight, analogous to light only bending on borders between unequal material. The latter is described by *Snell's Law of Refraction* in physics. As a consequence, this property of optimal paths is referred to as Snell's Law in the literature. Based on this, we characterize shortest paths in weighted regions to be polylines with possible bends only at edges of the subdivision.

We can therefore approximate a minimum-cost path by estimating possible refraction points at a triangle t_i. We place *Steiner points* along each edge and create, for each triangle t_i of the given triangulation, a complete graph $G_i = (V_i, E_i)$ that we call *face graph*. The vertices of G_i are all Steiner points on the edges of t_i and all vertices of t_i. An edge $e = uv \in E_i$ has a weight $w(e) = w_i \cdot d(u, v)$ equal to the weighted distance

between its endpoints, where w_i is the weight of the triangle t_i. See Figure 3.1 for an example of two adjacent triangles that contain two face graphs. To find a shortest path estimation, we form the union of all face graphs G_i to a graph $G = (V, E)$, where $V = \bigcup_i V_i$, $E = \bigcup_i E_i$ and then run a shortest path algorithm on G.

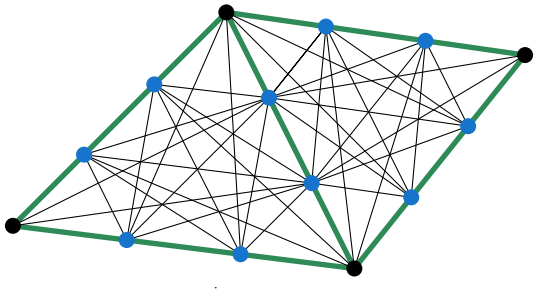


Fig. 3.1: Two Steiner points (blue) are evenly distributed along each triangle edge (green). Both triangles contain a complete graph with edges in the interior. Note that since face graphs are complete graphs, edges between all Steiner points and triangle vertices exist. For the sake of the clarity of this illustration, no edges that coincide with a triangle edge have been drawn.

We consider two of several strategies to place Steiner points; a uniform *fixed scheme* and its refinement, the *interval scheme*. In the former, a fixed number of Steiner points, m, is distributed evenly along each triangle edge. The latter takes an interval length $l \in \mathbb{R}$, which is then used to place Steiner points on each edge such that there is at most a distance of l between two consecutive points on an edge of the triangulation. Figure 3.1 shows an example for the fixed scheme for $m = 2$.

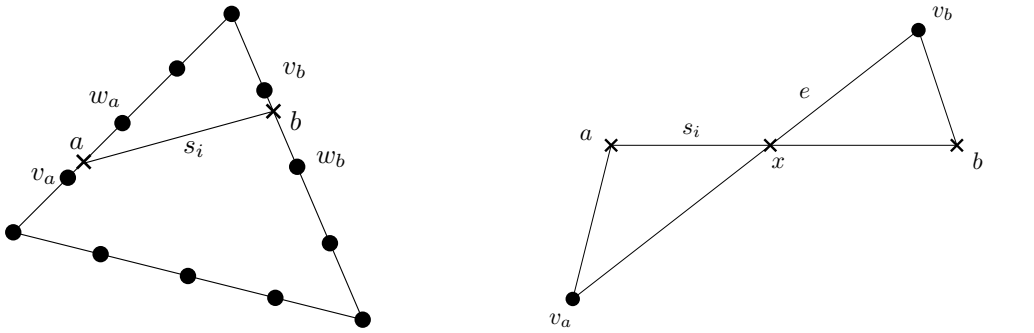
3.2 Approximation Bounds

We first prove the approximation bounds for the fixed scheme. Then we will argue that the same arguments hold for the interval scheme, showing that the interval scheme estimates a shortest path at least as good as the fixed scheme if l and m are selected appropriately. The proof is a more detailed version of the proof offered by Lanthier et al. [LMS01] and utilizes an intermediate result regarding the maximum increase of cost in one face graph, explained in the following:

Lemma 4 ([LMS01])**.** *Given a segment s_i of the optimal shortest s–t path that crosses triangle t_i then there exists an edge $e \in E_i$ of the face graph $G_i = (V_i, E_i)$ built by the fixed or the interval scheme such that $|e| \leq |s_i| + |L|/(m+1)$, where L is the longest triangle edge.*

Proof. Each edge of a triangle t_i is subdivided by its Steiner points into $m+1$ intervals. The distance between two Steiner points on the same triangle edge can thus be at most $|L|/(m+1)$. Let a and b be the end points of s_i lying on edges e_a and e_b of triangle t_i. Also, assume that $e_a \neq e_b$. Let $\overline{v_a w_a}$ be the segment spanned by the Steiner points

v_a and w_a on which a is located and let v_b and w_b be the Steiner points that enclose b. Without loss of generality, let v_a (v_b) be the Steiner point closer to a (b). See Figure 3.2a for an example of all definitions. Since G_i is a complete graph, G_i contains the edge $e = \{v_a, v_b\}$. *Case I*: $|s_i \cap e| = 1$. The segments s_i, $\overline{v_a a}$, $\overline{v_b b}$, and e form two triangles as illustrated by Figure 3.2b. Let x be the intersection point of e and s_i. We decompose e and s_i into segments $\overline{v_a x}$, $\overline{x v_b}$ and $\overline{ax}$, $\overline{xb}$, respectively.



(a) Segment s_i is the optimal line segment crossing triangle t_i.

(b) Two triangles formed by the intersection of s_i and e.

Fig. 3.2: Edge e of the face graph of t_i approximates the segment s_i of an optimal path.

By the triangle inequality, we have

$$|\overline{v_i x}| \leq |\overline{v_i a}| + |\overline{ax}| \tag{3.1}$$

$$|\overline{x v_k}| \leq |\overline{v_k b}| + |\overline{xb}| \tag{3.2}$$

by adding Equations (3.1) and (3.2), we get:

$$|\overline{v_a x}| + |\overline{x v_a}| \leq |\overline{v_a a}| + |\overline{ax}| + |\overline{v_b b}| + |\overline{xb}|$$

$$|e| \leq |s_i| + |\overline{v_a a}| + |\overline{v_b b}| \tag{3.3}$$

Case II: $s_i \cap e = \emptyset$. Equation (3.3) immediately follows by the triangle inequality.

Since we have chosen v_a and v_b to be the Steiner points closer to a and b, respectively, we know that $|\overline{v_i a}| \leq |L|/2(m+1)$ and $|\overline{v_k b}| \leq |L|/2(m+1)$. In the case of the interval scheme, we can do this proof in a similar way using a maximum length l between adjacent Steiner points. We obtain that $|\overline{v_a a}| \leq l/2$ and $|\overline{v_b b}| \leq l/2$. By setting l to $|L|/(m+1)$, the claim follows for both schemes. □

With this lemma we can prove an approximation guarantee for the paths found by the Steiner point method.

Theorem 5 ([LMS01])**.** *Let $G = (V, E)$ be the representation of a triangulation. By utilizing the fixed scheme, the Steiner point method yields a shortest s–t path approximation for the Weighted Region Problem that is not longer than $c(P^\star) + w_{\max}|L|$, where $w_{\max}$ is the maximum weight of all triangles of the triangulation and L is the longest edge of the triangulation.*

Proof. The idea of this proof is to join Lemma 4 with a property of shortest paths that solve the Weighted Region Problem. This property, observed by Mitchell and Papadimitriou [MP91], tells us that such a path consists of $\mathcal{O}(|V|^2)$ segments.

Let the optimal path consist of $\langle s_1, \ldots, s_k \rangle$ segments. Lemma 4 grants us a bounded approximation for each segment s_j of the optimal path by an edge of the corresponding face graph G_i for a triangle t_i, in which s_j is situated. We choose the edges closest to the end points of s_j such that each edge e_j approximating s_j shares a Steiner point with its predecessor. We obtain a path of edges $\langle e_1, \ldots e_k \rangle$.

We apply Lemma 4 multiplied by w_{f_i}, where f_i is the triangle, in which s_j and e are contained, to each segment of our approximated path:

$$\sum_{i=1}^{k} w_{f_i}|e_i| \leq \sum_{i=1}^{k} w_{f_i} \left(|s_i| + \frac{|L|}{m+1} \right)$$

$$\Leftrightarrow c(P) \leq c(P^\star) + \frac{|L|}{m+1} \cdot \sum_{i=1}^{k} w_{f_i}$$

In the worst case, we only traverse triangles of maximum weight. We therefore estimate the cost of our path by

$$c(P) \leq c(P^\star) + k \cdot \frac{|L|}{m+1} \cdot w_{\max}$$

With the property of Mitchell and Papadimitriou [MP91], we can bound k by $\mathcal{O}(|V|^2)$. Using $m \in \Theta(|V|^2)$, the claim follows and we indeed obtain an upper bound of $c(P^\star) + |L| \cdot w_{\max}$ for the cost of the path approximation found by the Steiner point method. □

3.3 Time Complexity

Lanthier et al. [LMS01] provide a succinct runtime analysis from the perspective of Theorem 5 by assuming that $m \in \Theta(|V|^2)$. They obtain a runtime of $\mathcal{O}(|V|^5)$. Here we provide a more fine grained analysis applicable for any m which will help us understand the runtime behavior studied later in Chapter 6. From here on we set $n = |V|$ for a graph $G = (V, E)$.

Theorem 6. *Let a triangulation be represented by a graph $G = (V, E)$ and let $m \in \mathbb{N}$ be the number of Steiner point per triangle edge. The Steiner point method runs in $\mathcal{O}(nm \log(nm) + nm^2)$ time using the fixed scheme.*

Proof. Since we assume our input to be a planar triangulation, Euler's polyhedron formula implies that we do not have more than $2n - 4$ triangles. Therefore, we can bound the number of face graphs by $\mathcal{O}(n)$. Each of the face graphs has at most $\mathcal{O}(m)$ vertices and so at most $\mathcal{O}(m^2)$ edges. Our search graph of the triangulation is the union of all face graphs, so it has at most $\mathcal{O}(nm)$ vertices and $\mathcal{O}(nm^2)$ edges, which means that the construction of this graph takes $\mathcal{O}(nm^2)$ time. We find a minimum cost s–t path in this graph with Dijkstra's algorithm using a Fibonacci heap in $\mathcal{O}(nm \log(nm) + nm^2)$ time. □

3.4 Angle Minimization

In many infrastructures the turn angle of path bends is of importance. Not all algorithms for infrastructure planning support the multi-objective of finding an approximate minimum-cost path with few of turn angles. We aim to fill this gap and expand the Steiner point method with such a feature.

In the literature the examination of edge pairs for shortest paths has been coined as the *Quadratic Shortest Path Problem* (QSPP) by Rostami et al. [RMFB15]. Unfortunately, in the same publication, in which Rostami et al. proposed the QSPP, they also showed strong NP-hardness for this problem. For our purposes, we do not have to consider arbitrary edge combinations but only adjacent edge pairs. This problem formulation is called the *Adjacent Quadratic Shortest Path Problem* and has been found to be NP-hard as well by Rostami et al. [RCH+18]. Hu and Sotirov [HS18] showed that in the special case of directed acyclic graphs (DAG) the AQSPP is solvable in polynomial time. Nevertheless, since the search graph of the Steiner point method is not acyclic, we cannot employ the work of Hu and Sotirov and have to resort to a relaxed version proposed by Wiedemann and Adjiashvili [WA21], in which we only ask for a least-angle minimum-cost *walk* instead of a path. This means that we can avoid sharp turns by cycles. This loosening allows for an efficient computation of minimized turn angles with the help of a line graph (see Definition 2) that does not have to be built explicitly. Although the approach by Wiedemann and Adjiashvili [WA21] has been implemented using the Bellman–Ford algorithm, we can easily adapt their method to other shortest path algorithms such as A*. In the following, we describe how to implicitly use a line graph to obtain such a minimal-angle walk.

We define an edge angle cost function $\alpha\colon E \to \mathbb{R}$ to be a function that maps two incident edges e_1 and e_2 to a cost value subject to the angle $\measuredangle(e_1, e_2) \in [0, 180]$. We assign to each edge e of G an accumulated cost $D[e]$, similar to the distance from the source to a vertex v in a ordinary shortest path algorithm. Initially, we set $D[e] = \infty$ for all edges of G except for outgoing edges of the source s. Each such edge e_s gets $D[e_s] = w(e_s)$. At each step, the algorithm updates $D[e]$ for an edge $e = (v, w)$ as follows:

$$D[e] = w(e) + \min_{e_{\text{in}}} \left\{ D[e'] + \alpha(e, e') \right\}, \tag{3.4}$$

see Figure 3.3.

In the update step for a vertex v, we have to update all accumulated costs for all its outgoing edges. A straightforward way of doing this for an out-going edge e_{out} from v is to iterate over all in-going edges e_{in} to v and to search the minimum by Equation (3.4). This results in a time complexity of $\mathcal{O}(\text{outdeg}(v) \cdot \text{indeg}(v))$ for each update step. While the implicit line graph has a space advantage, the asymptotic runtime is the same with this update approach. A remarkable improvement can be achieved, however, with Wiedemann's and Adjiashvili's [WA21] accelerated update algorithm for convex edge angle cost functions that reduces the required runtime per update to $\mathcal{O}((\text{outdeg}(v) + \text{indeg}(v)) \cdot \log(\text{outdeg}(v) \cdot \text{indeg}(v)))$.

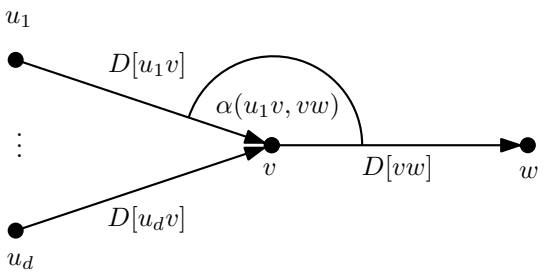


Fig. 3.3: The accumulated cost for an edge vw is updated via accumulated costs of all incoming edges $u_1v, \ldots, u_dv$ of v.

4 Pylon Spotting

A path with optional minimized total cost of bends does no suffice while designing infrastructure. While the Steiner point method is generic for infrastructure planning that requires shortest paths, in specific applications different challenges arise. In this work, we evaluate the approach on projects of power infrastructure planning as a use case. In transmission line planning not only the route must be optimized, but also the location of the pylons along the route which is referred to as pylon spotting. Power lines consist of two main components we can optimize: the cost of placing pylons and the cable costs between two pylons. These costs vary depending on which area pylons are placed or which regions cables cross. There are several choices that can be made regarding the setting of the cost optimization. Distinguishing cable and pylon costs, for example, is crucial in places that can easily be traversed with a cable but are not suitable for building a pylon, e.g. a river. In other places such as housing, both pylons and cables have to abide to the forbidden area. Typically, an additional constraint of a minimum and a maximum distance between pylons is posed as there are technical limits to pylons and cables. Such distance restrictions can force the cables to leave the original path of the Steiner point method. So in the case of forbidden areas that must not be crossed by cables or a consideration of cable costs, an evaluation of the cables or "shortcuts" are required.

Based on the previously obtained Steiner point path, we subsequently model the pylon spotting problem in a graph. We construct a connected *pylon graph* $G_p = (V_p, E_p)$, in which a vertex $v \in V_p$ represents a place of a possible pylon with a given cost $c(v)$ and an edge $\{u, v\} \in E_p$ corresponds to a cable between pylons placed at u and v with costs $w(\{u, v\})$. If cable costs are not considered, these weights are ignored. Since we have a direction from source to target and going back can never be cheaper, we can assume a directed graph. The desired power line configuration will be the shortest path in the graph G_p, each path vertex being a pylon of the transmission line and each edge (straight line) between two vertices being a cable.

The vital part is the construction of the graph G_p. From there on, we employ a suitable shortest path algorithm such as Dijkstra's or a DAG shortest path algorithm with a minimum-angle modification. To consider pylon costs with a shortest path algorithm, we also have to make a small modification when updating the cost of vertex v. In addition to changing the current cost of a shortest path from s to v with the total costs of the predecessor on the shortest s–v path and the costs of the edge between them, we also add the pylon cost $c(v)$ to this sum.

So, let $P_S = \langle s, v_2, \ldots, v_{k-1}, t \rangle$ be the provided Steiner path. The graph G_p will be generated in the following three steps:

(1) clean the path,

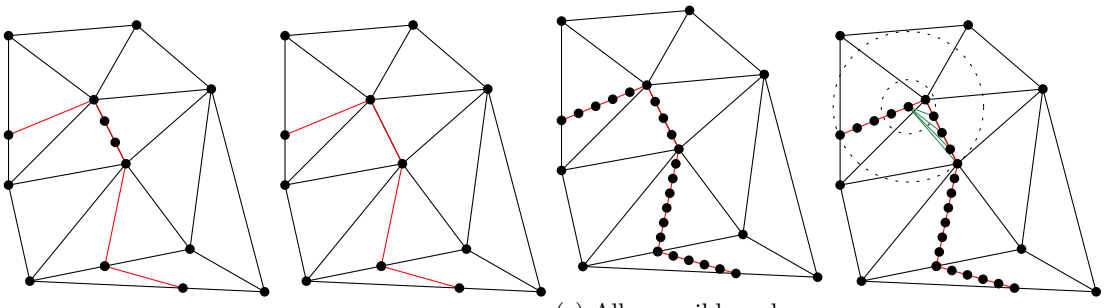
(2) discretize the search space by utilizing a technique similar to the interval scheme,

(3) create all viable cable shortcuts.

Path Cleaning. In order to station eventual pylons along the path, we first need to delete path vertices of P_S that might interfere with the even arrangement of pylons later on; that is, end points of line segments which are collinear to adjacent lines of the polyline P_S and do not have two triangle vertices as end points. Compare Figures 4.1a and 4.1b for an example. It is worth mentioning that we can obtain a cleaned path automatically by the Steiner point method, if we add an additional cost to each distance of a vertex – a "vertex cost" – in the path finding process similar to the consideration of pylon costs. As a consequence, the path-finding algorithm chooses the path with as few vertices as possible among possible paths with the same overall cost.

Placement of possible pylons. Given a new interval length l_p we uniformly put pylon places v_p along the cleaned path, analogous to the interval scheme. Note that we also position pylon vertices inside triangles in contrast to the original interval scheme, in which Steiner points are only placed on edges, as Figure 4.1c illustrates. Since we have vastly reduced the search space of possible pylon positions by restricting ourselves to only considering locations on the path P_S, we can have small interval lengths l_p.

Creation of feasible shortcuts. After we have created potential pylon positions, the edge generation remains. For this purpose, we do a linear search through the augmented path and find for each v_p all u_p in the ring $R(v_p) = \{u_p \mid d_{\min} \leq d(u_p, v_p) \leq d_{\max}\}$ spanned by the minimum and maximum distance. Out of these $u_p \in R(v_p)$ only those that are successors of v_p are of interest for a directed graph approach. If we only consider pylon costs we can safely connect each v_p and u_p with an edge. Otherwise, we have to verify that the direct connection between v_p and u_p does not cross a forbidden area and if necessary, evaluate the total cost of this shortcut. Since this edge might traverse several triangles as it is the case in Figure 4.1d (green edges), a more careful approach of assessing the cost and feasibility of this possible cable is necessary. This can be done by exploiting topological information of our triangulation. Since a cable is always a straight line we start from v_p and walk from intersection to intersection between triangle edges and the possible cable. By measuring the distance between two intersection points in one triangle, we can accumulate the total cost of our potential cable. For details, see Section 5.3.



(a) Original path containing unnecessary points.

(b) Cleaned path. No intermediary points exist.

(c) All possible places for pylons are distributed along the path.

(d) Possible short cuts for an exemplary possible pylon.

Fig. 4.1: All steps of the pylon placement process: We clean the path and place points along the path, that represent possible pylon positions. Each point v_p is connected with other points u_p on the path within the distance of $d_{\min} \leq d(v_p, u_p) \leq d_{\max}$.

5 Implementation

This chapter describes implementation details and modifications of the Steiner point method we made and the intricacies of the pylon spotting process. As a general design choice of the implementation, we made heavy use of the strategy pattern and inheritance, to grant a maximum amount flexibility. Everything has been written in C++17 and was compiled with the MSVC compiler with level 2 optimization.

5.1 Input Format

We store the triangulation with the corresponding weights in a simple text file and focus on simple rules how to read the input, to assure that input files are compressed since real world triangulations can be large. Along with the triangle vertices, edges and weights, we also have to store some topological information about the triangulation, namely the neighboring triangles for each triangle vertex. This data is important for the cost evaluation of the shortcuts in the pylon spotting process, described in Chapter 4 and our modification of the Steiner point method. We also require that every edge knows to which of at most two triangles it belongs.

We encode this information the following way. Given a triangulation with N vertices, M edges and T triangles, we assign each triangle, vertex, and edge an index. The first line of our file consists of N, M, T, the index of the source s and the index of the target t. The subsequent N lines list the coordinates of each vertex. After that, the next M consecutive lines contain the edge information. Each edge is represented by the index of the two vertices the edge connects. T lines follow, each holding a triangle that is composed of the indices of the triangle edges and a weight. The weight can also have a flag, indicating that this triangle is forbidden to traverse by the Steiner point method. We use the sign of the weight as a flag, where values smaller than zero imply a forbidden triangle. The remaining topological data is expressed by N lines. The first number in each line declares the number of triangles a vertex is part of, followed by the indices of these triangles.

5.2 Details and Augmentations to the Steiner Point Method

We briefly describe our graph representation and expand on how the graph building and path finding step of the Steiner point method can be merged.

The input format already suggests the storage of the triangulation in arrays. Since Steiner points inherently belong to triangle edges, each edge will claim ownership of the Steiner points that are placed on it and will store them in a list. The search graph

will consist of Steiner points and triangle vertices, so each vertex has a list containing pointers to its neighbors in the search graph. This representation can also be used to later find the cheapest pylon placement configuration.

Our implementation will use the interval scheme, as it has important advantages over the inflexible fixed scheme. It should be noted that the interval l is an upper bound for the interval between two adjacent Steiner points. This is due to our requirement to place the Steiner points evenly along the edge. For an edge $\{u, v\}$, we calculate the location of its Steiner points as follows: First we calculate the number of Steiner points, p, we have to place by dividing the length of the edge $|\overline{uv}|$ by the interval length l. If s is smaller than 1, we return and place no points along the edge. However, if s is bigger or equal than 1, we compute the length of the distance between each vertex by $|\overline{uv}|/\lceil p \rceil \leq l$ which may result in a smaller distance between two adjacent Steiner points as we round p. The placement is straightforward.

The original Steiner point method is a two step approach that first builds the search graph by creating a face graph for each triangle and then constructs the union of all face graphs. Since each triangle edge is unique, i.e., one edge is shared by up to two triangles and each edge has ownership of its Steiner points, the union is implicitly formed by the adjacency lists of all vertices. In the second step, we apply a suitable shortest path algorithm on it. To improve the performance of the Steiner point method, we merge these two steps together to avoid creating face graphs which are not necessary for finding a shortest path with the A* algorithm. This can result in a significant runtime decrease if source s and target t are close together relative to the input domain. Algorithm 1 outlines the merged Steiner point method:

Algorithm 1: Modified Steiner point method

```
1  FibonacciHeap heap
2  s.d = 0
3  heap.insert(s)
4
5  while heap ≠ ∅ do
6      u = heap.extractMin()
7      if u == t then
8          return
9      buildCompleteFaceGraph(u)
10     foreach v ∈ u.Adj do
11         relax(v)
```

The function `buildCompleteFaceGraph(`u`)` uses the topological information of the input to build the graph in the neighborhood of vertex u. Concretely, if u is a Steiner point, we build the face graph of each adjacent triangle of the corresponding edge, provided the face graph of the triangle has not been build yet. Otherwise, u is a triangle vertex. In this case, we build the face graphs of each triangle the vertex u is part of.

The subroutine `relax(`v`)` is the standard update function of the A* algorithm. We use the heuristic $h(v) = w_{\min} \cdot d(v, t)$. Since this heuristic cannot overestimate the costs and is at best the actual cost, the A* algorithm yields the correct solution according to Lemma 3.

5.3 The Pylon Spotting Process

We omit the details of cleaning a path since the Steiner point method is able to provide a cleaned path as described in Chapter 4. We focus on how the cable costs can be computed efficiently by "walking" along the cable with the help of topological information of the neighborhood of triangles.

So far, we had two possible types of vertices: triangles vertices and Steiner points, both being aware of their neighborhood of triangles. With the placement of possible pylons inside triangles, we obtain a new type of vertex with respect to the triangulation that we call *interior vertex*. In order to efficiently walk along a cable that connects an interior vertex with another vertex, we need to know in which triangle it is contained. We compute this information during the possible pylon placement phase while placing vertices between two consecutive path vertices u_i and u_{i+1}. If vertices u_i and u_{i+1} are not both triangle vertices, interior vertices are placed as described in Chapter 4. With the information of the neighboring triangles of vertices u_i and u_{i+1} we can compute the index, in which the interior vertices are contained, by finding the common neighboring triangle.

The topological information of all vertex types and triangle edges is sufficient to walk along a potential cable as a means to calculate the total costs. We first locate the triangle that is intersected by the cable in at least two points. Since the edges that are cut by the cable know their neighboring triangles, we can reach the next triangle that is intersected by the possible cable. Note, that a cable might cross a triangle parallel to one of its edges. In that case, the next triangle has to be determined by iterating over all neighboring triangles of the end point of that edge. We have computed the total cost if one intersection point is either the end point of the possible cable or we cannot find the next triangle since the end point is an interior vertex.

After we have calculated the costs of all possible cables, we employ Dijkstra, modified to also minimize the angles between two edges, since the total runtime of this algorithm is negligible compared to the runtime of the Steiner point method.

6 Experiments

In this chapter, we experimentally analyze several aspects of our proposed two-step algorithm for finding transmission lines. We examine the performance of the Steiner point method in both runtime and in cost convergence behavior in Section 6.1. Then we explore the trade-off between angle penalization and total resistance cost in Section 6.2. Finally, we compare our algorithm with a novel algorithm by Wiedemann et al. [WA21] called LION, that is based on rasterized input data; see Section 6.3. All tests have been conducted on a machine with an Intel Core i5-7600K 3.8 GHz 4 core CPU and 16GB DDR4 RAM.

6.1 Analysis of the Steiner Point Method

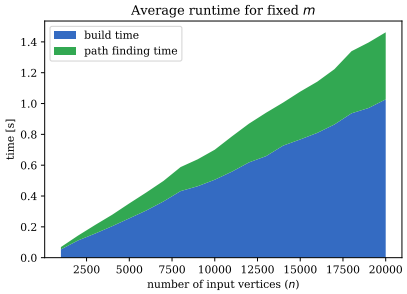
In the following, we use the interval scheme in our analysis. According to the time complexity analysis in Section 3.3 and the approximation bound in Section 3.2, the number of input points, n, and the number of Steiner points per edge, m, impact the performance.

Test data. Since the interval scheme places Steiner points such that their number is proportional to the length of each edge, we have to ensure that the average length of the triangle edges in each instance for testing the runtime is the same. Furthermore, in order to provide comparable instances, the ratio of triangles that have been explored by the A* algorithm to the total number of triangles needs to be constant.

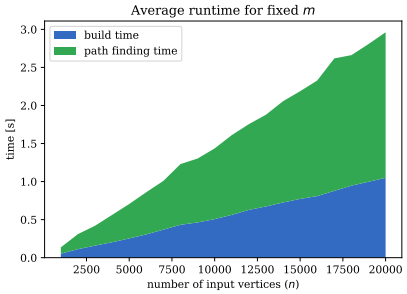
For this purpose we randomly generate instances with the same average triangle edge length by keeping the *global point density* $\rho(I) = n/A$ constant, where the variable A denotes the area of the domain from which we sample the random points. We choose a disk as sampling domain to avoid long, unrealistic edges at the border. To generate a sequence of test cases with increasing input size, while keeping the global point density the same in each, we start with an initial instance I_0 with input size n_0 and radius r_0 of the disk. For a test case I_{i+1} with n_{i+1} points, we have to set the radius r_{i+1} to $r_0 \cdot \sqrt{n_{i+1}/n_0}$ to keep the point densities $\rho(I_0)$ and $\rho(I_{i+1})$ the same. By picking a suitable interval of weights per triangle, we manipulate the search space of the A* algorithm because this interval influences the heuristic we use which was described in Section 5.2. If not specified otherwise, we assign each triangle a uniform random weight in the interval $[100, 500]$ and choose source and target to be vertices at points $(-r_i, 0)$ and $(r_i, 0)$, respectively.

In order to test the impact of m on the performance, we use randomly generated instances of fixed size and gradually decrease the interval length to obtain an increasing average number of Steiner points per edge.

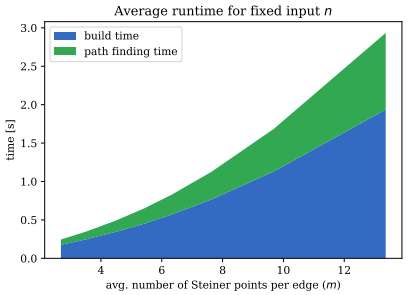
Runtime. A runtime measurement depending on input size by fixed average number of Steiner points per edge is conducted in Section 6.3 in conjunction with the comparison of LION and in the runtime breakdown into different factors. The effect on the performance by varying interval length is also included in the detailed runtime analysis, therefore we will omit a high-level analysis of the runtime in that regard and immediately study the impact of the main components of the augmented Steiner point method, namely the graph building and the path-finding step. For the tests with a fixed average number of Steiner points per edge, we ran test cases with input sizes $n = 1000$ to $n = 20000$ and for tests of varying m and fixed input size we picked a test instance of input size $n = 10000$ and altered the interval length such that the average number of Steiner points per edge ranges from 2.5 to 13. Figure 6.1 reveals all results in stack plots, where the time of finding a path is placed on top of the time we need to build the graph, so the total execution time can be read from the top curve.



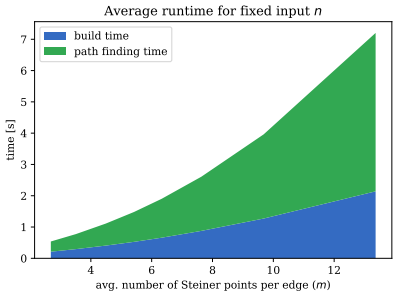
(a) Almost linear growth of the runtime without angle minimization for fixed m. The graph building step dominates the overall runtime.



(b) Runtime with angle minimization for fixed m. Growth is still almost linear but now minimizing angles is the more expensive factor.



(c) Quadratic runtime increase without angle minimization for fixed n and varying m. Most of the time is spent on building the graph.



(d) Runtime with angle minimization for fixed n and increasing m. Now, finding minimum-angle paths prevails the runtime.

Fig. 6.1: A runtime breakdown by the main factors graph building and path finding. The total time spent on path finding (green) is on top of the time we need for building the graph (blue).

Figure 6.1a confirms the time complexity analysis in Section 3.3 as we see an almost linear growth with increasing input size. In Figure 6.1b we observe that the angle minimization of our method contributes a factor of about 2 for fixed m. A quadratic

increase in runtime by declining interval length for the original Steiner point method can be identified, while the runtime curve of the angle minimization modification with the naive update function is characterized by a sharper growth, as by increasing m the number of in-coming and out-going edges increases by a factor of $\mathcal{O}(m)$, culminating into an additional quadratic change in runtime. We can make two distinct conclusions about the main factor that shapes the runtime depending whether we use the original Steiner point method or also search for an angle minimal least cost path. Figures 6.1a and 6.1c pinpoint the building time of the face graphs as main factors for the runtime regarding the non-augmented Steiner point method, whereas Figures 6.1b and 6.1d determine the angle minimization to be the more expensive element. The latter strongly suggest the usage of a convex and monotonically increasing angle cost function, so the accelerated angle update function of Wiedemann and Adjiashvili [WA21] can be used instead of the naive approach, as mentioned in Section 3.4. This significant influence on the performance of the algorithm is concerning from the perspective of large input data as common in infrastructure planning and, more importantly, a large number of Steiner points.

These considerations lead us to the study of cost convergence. Such a study will allow us to reduce the number of Steiner points. We want to show that even a small value of m yields paths of near optimal cost.

Cost convergence. We empirically inspect the cost convergence behavior in different instances. Since no algorithm is known to solve the Weighted Region Problem exactly, we cannot do a definitive comparison. On all instances we vary the average number of Steiner points per edge by changing the interval length. We include the result for the simplest approximation, where the input graph is used as search graph and no Steiner points are used ($m = 0$), as outlined in Chapter 3. Furthermore, we ran an extreme case, in which we set the interval length l to 1 meter in all test instances, which corresponded to 65 Steiner points on average on each edge.

Figure 6.2 shows the resulting cost convergence in six different randomly generated instances. These inputs have been generated in a similar manner to those in Section 6.1. The dashed line is the cost obtained by setting the interval length extremely low. We can confirm the observation of Lanthier et al. [LMS01], that a small number of Steiner points per edge seem to suffice to achieve good approximations, as all our instances seem to converge rapidly to a certain cost. Figures 6.2b and 6.2d show particularly benevolent input configurations, in which only two Steiner points per edge seem to be enough to be very close to the optimal path cost.

It should be pointed out that we indeed do not get a strictly decreasing path cost for increasing m as Figures 6.2a and 6.2f illustrate. This is due to the fact that the even distribution of Steiner points on each edge can lead to a worse approximation of the optimal bending point of the path by varying m. To see this, consider an edge that is crossed by the optimal path at point $s^\star$. Let $m^\star$ be the number of Steiner points, such that the even distribution of them along this edge exactly coincides with $s^\star$ and for the sake of simplicity, we say that the length of this edge is divisible by $m^\star$. Now, if we choose a new $m' > m^\star$, that does not divide the corresponding edge length evenly, the

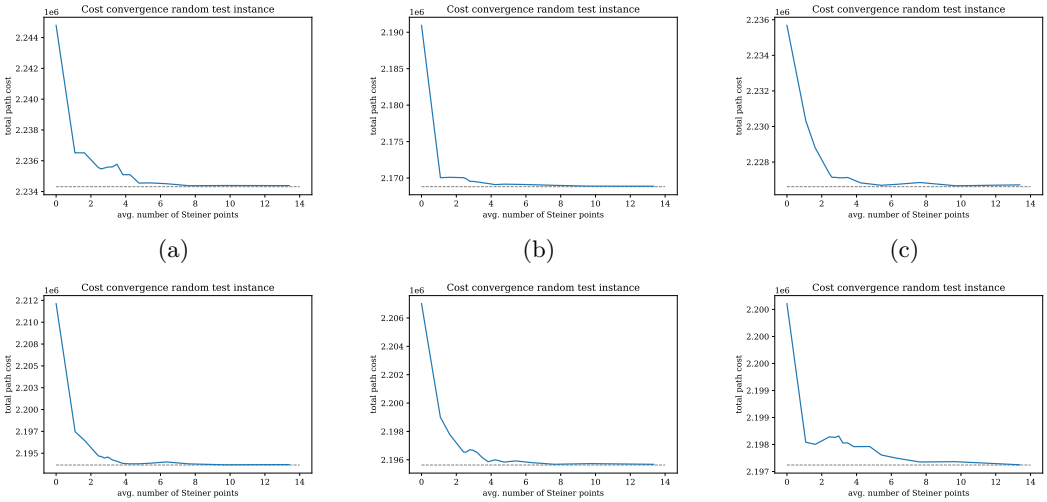


Fig. 6.2: The cost convergence behavior of the Steiner point method on different test instances. Each instance has $n = 10000$ triangle vertices and an average edge length of 65. The dashed line represents the costs returned by setting the interval length to $l = 1$ resulting in an $m = 65$. This data indicates that a small number of Steiner points yields a good approximation. Figures 6.2a to 6.2f show that a small value of m suffices to produce near optimal paths.

even distribution will be slightly shifted compared to the positions of Steiner points in the case of $m^\star$. This results in a worse approximation of $s^\star$ with a higher number of Steiner points. See Figure 6.3 for an illustration.

Fig. 6.3: Three Steiner points evenly distributed along the triangle edges achieve the optimal path segment. By increasing m to 4, all Steiner points shift on all edges and the optimal segment cannot be found anymore. This results in worse approximation with a bigger value for m.

6.2 Angle Cost Trade-Off

Finding a good power line poses a multi-objective optimization problem, as several factors, such as financial, geometric (bending angles), environmental and other aspects have to be considered. In this section, we consider the trade-off between the financial costs of placing pylons and the geometric form of transmission lines by minimization of angles discussed in Section 3.4. It is expected that a stronger focus on the minimization of angles results in higher financial costs, because straighter paths are more favored at the expense of sub-optimal pylon positions.

We analyze a positive linear edge angle cost function $\alpha(e_1, e_2) = c \cdot (180° - \measuredangle(e_1, e_2))$, where $\measuredangle(e_1, e_2)$ is the smaller angle between e_1 and e_2 with varying $c > 0$ in a Pareto chart, shown in Figure 6.4. We measure the sum of the angles to characterize the the straightness of the path.

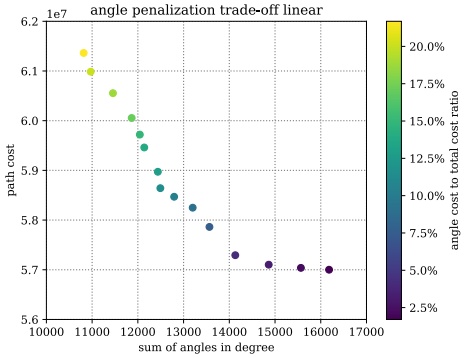


Fig. 6.4: Pareto chart showing the trade-off between minimizing the turn angles and the total path cost for a linear function $\alpha(e_1, e_2) = c \cdot \measuredangle(e_1, e_2) \geq 0$. With a higher value for c, the angle between two incident edges is weighted more, resulting in a straighter path (smaller sum of angles) but in higher costs.

Each point in Figure 6.4 corresponds to one path computed with a specific factor c, the higher the factor, the brighter the color of that point. Each path is scattered with respect to the achieved angle sum and path costs, shaping the trade-off surface. Note, that the points are not even distributed regularly even though the factor c changes by regular steps. This happens because the minimum cost paths constitute a discrete family. So, if much more intermediate points were plotted, they would cluster at certain angle sums, since the factor c needs to exceed a threshold in order to result in a new path with a smaller costs.

We observe a clear trade-off between the minimization of angles and the corresponding path costs, resulting in no factor c that provides low path costs and low turn angles. For a distribution of angles for three different values of c, see Figure 6.5. We notice the expected shift towards smaller angles, up to the point where the mean angle is below ten degrees.

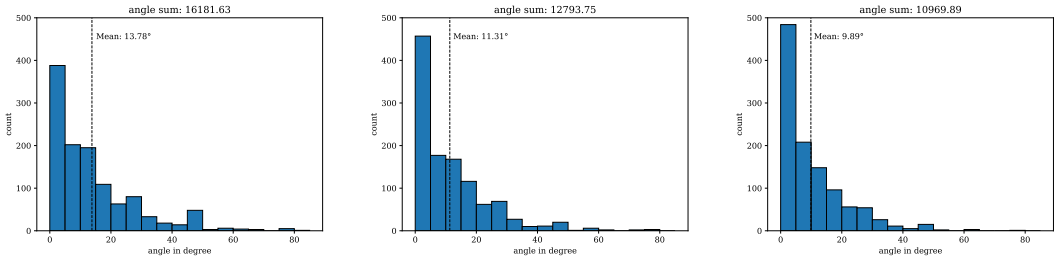


Fig. 6.5: Three angle distributions for different values of c. Each bar represents the number of angles steps of five degrees. We see a significant increase in smaller angles with increasing c.

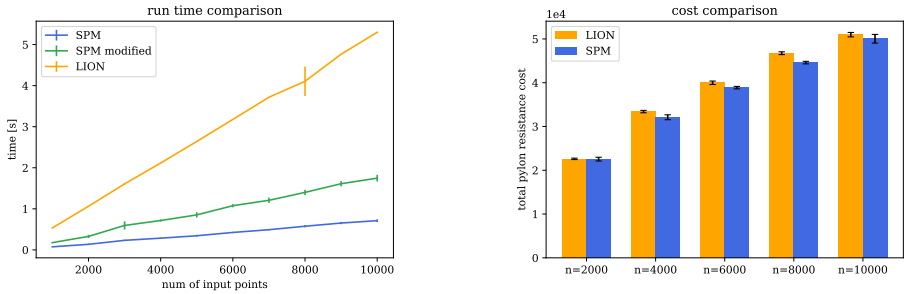
6.3 Comparison with LION

LInear Optimization Networks (LION) is an open-source package containing the algorithms of Wiedemann [WA21], which aims to find optimal transmission lines by taking rasterized geographic data as input. It applies the same angle minimization approach we use but other than our algorithm, it minimizes the total cost of pylons for a power transmission line globally. Instead of using Dijkstra's or A*, LION relies on the Bellman–Ford algorithm which runs in $\mathcal{O}(|V| \cdot |E|)$ compared to Dijkstra's algorithm which runs in $\mathcal{O}(|V| \log |V| + |E|)$ if a Fibonacci heap is used as priority queue, where $|V|$ ($|E|$) is the number of vertices (edges) of the search graph. Furthermore, an iterative procedure to decrease time and space by scaling the raster is implemented. This procedure first down samples the input to a more coarse raster, in which it computes a path. It then builds a corridor of a certain width around the path and iteratively improves the resolution and the width of the corridor until the initial resolution is reached. While a considerable improvement in time and space can be achieved with this scaling approach, only a locally optimal path can be guaranteed. For comparable results we set the memory limit and the input to manageable magnitudes, such that the scaling approach will not be applied. Additionally, LION handles angle weights differently, so for a fair comparison, we do not consider the angle minimization and set all angle weights to zero.

Test data. We focus on a runtime and cost comparison on randomly generated data and employ the same input types of Section 6.1 for the triangulated data with an average edge length of 120m. To obtain the corresponding rasterized input, we lay a square grid of 10m by 10m for each cell on top of the triangulation and assign each cell the associated weight of the triangle, in which the center of the cell is contained. For a comparable results we set the interval length to $l = 10$m.

Results. Figure 6.6 shows the runtime and cost comparison. For the experiments, buckets of input sizes $n = 1000$ to $n = 10000$ were created; each bucket having five test instances. Figure 6.6a shows the average runtime per input size, together with the

standard deviation and Figure 6.6b reveals the average cost per input size. Note, that raster size grows by a factor of $\mathcal{O}(\sqrt{n})$ with increasing n, since the radius of the sample domain grows by a factor of $\mathcal{O}(\sqrt{n})$, as discussed in Section 6.1.



(a) (b)

Fig. 6.6: Runtime and cost comparison of our method with LION on randomly generated data. For more comparable runtime results, the Steiner point method has been ran with and without the augmentations to minimize angles.

In this uniform setting, we can also infer that our algorithm runs approximately five times faster. However, this conclusion cannot be easily generalized to other input data, as the number of input points and thus the number of triangles do not have to strictly correlate with the raster size but mainly depend on the triangle layout.

Figure 6.6b juxtaposes the total costs of pylons according to each algorithm. We notice slightly better results by using our method. Small differences in triangulated and rasterized inputs have to be taken into account, while judging which algorithm achieves lower costs, as source and target costs might differ, due to inaccuracies with the cost assignment of the grid-based input. Since start and end point are triangle vertices, our method chooses the minimum weight of all adjacent triangle for the cost of the start and end point, respectively; whereas LION cannot make that distinction and has to use the cost of the cell of the starting point, whose center might be contained in a triangle with higher costs. Nonetheless, test cases displayed in Figure 6.6b with input size $n = 4000, 6000, 8000$ suggest that due to the monolithic discretization intrinsic to the grid-based approach, sub-optimal paths with pylons centered in the raster cells are found. Our results demonstrate that the Steiner point method can achieve superior performance through the dynamic placement of points along triangle edges that is not restricted to a homogeneous distribution of cells. We can expect that raster-based approaches can mitigate this disadvantage by increasing the resolution of the grid. However, as shown in Figure 6.6a and in more detail in Wiedemann [WA21], this results in a quadratic increase in runtime.

In consequence, we conjecture that our method can yield paths at least as cost efficient as paths produced by LION in a significantly shorter time, showing the superiority of polygon based data compared to rasterized data. This is surprising, considering the fact that we do not claim to optimize the pylon spotting problem globally, as LION, but do only make an estimation by an approximation of the Weighted Region Problem.

7 Conclusion and Future Work

We have introduced a new approach to infrastructure planning based on polygonal data and used the Weighted Region Problem to formulate the problem of finding a minimal cost path in this setting. Unfortunately, there is no exact algorithm known to find a shortest path in weighted regions. Therefore, we made use of the Steiner point method that finds a theoretically provable approximation of a shortest path. Since infrastructure planning often requires angle minimized shortest paths, we combined the Steiner point method with an angle minimization procedure by Wiedemann and Adjiashvili [WA21]. We tested that generic approach with an algorithm for finding transmission lines. For that, we used an angle minimized shortest path obtained by the augmented Steiner point method as a baseline for placing pylons and cables as cheap as possible.

Our experiments show promising evidence that our polygonal based approximation can achieve results on-par with globally optimal raster based methods or even better results due to the constraints of the raster layout, dependent on the raster resolution. Raster based algorithms have a clear trade-off between resolution and runtime and we were able to show that the runtime of our approach scales much better in project size and can be up to 5 times faster while returning similar results. However, more tests on real world data have to be conducted to make definitive statements about the accuracy in comparison with both approaches.

Nonetheless, our method can be improved. It should be pointed out that a performance increase in runtime can be expected with an improved memory allocation technique in the graph building step of the Steiner point method and employment of adequate strategies for parallelizing runtime intensive subroutines. If an angle minimization is not required, it is also possible to not store the edges of the graph and calculate the costs when needed which can result in a considerable improvement in storage. If, however, an angle minimized path is of importance, we strongly recommend to implement the accelerated update function by Wiedemann and Adjiashvili [WA21] as the execution time dramatically increases with the naive update approach that we have employed. For extremely large data or to obtain more accurate results a similar corridor approach common in raster data can also be easily implemented within the Steiner point method. Further improvements in regards of finding transmission lines can be made by considering slopes in the input data with anisotropic regions (see [CNVW08] for details).

Our new approach can support experts in infrastructure design. It makes planning faster and potentially more accurate. This helps stakeholders, planning agencies, and residents by eventually providing better infrastructures.

Bibliography

[AMS05] Lyudmil G. Aleksandrov, Anil K. Maheshwari, and Jörg-Rüdiger Sack: Determining approximate shortest paths on weighted polyhedral surfaces. *Journal of the ACM*, 52(1):25–53, 2005, 10.1145/1044731.1044733.

[BBK+18] Daniel Bachmann, Fritz Bökler, Jakob Kopec, Kira Popp, Björn Schwarze, and Frank Weichert: Multi-objective optimisation based planning of power-line grid expansions. *ISPRS International Journal of Geo-Information*, 7(7), 2018, 10.3390/ijgi7070258.

[CJV15] Siu Wing Cheng, Jiongxin Jin, and Antoine Vigneron: Triangulation refinement and approximate shortest paths in weighted regions. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, 2015:1626–1640, 2015, 10.1137/1.9781611973730.108.

[CNVW08] Siu Wing Cheng, Hyeon Suk Na, Antoine Vigneron, and Yajun Wang: Approximate shortest paths in anisotropic regions. *SIAM Journal on Computing*, 38(3):802–824, 2008, 10.1137/06067777X.

[DGM+14] Jean Lou De Carufel, Carsten Grimm, Anil Maheshwari, Megan Owen, and Michiel Smid: A note on the unsolvability of the weighted region shortest path problem. *Computational Geometry*, 47(7):724–727, 2014, 10.1016/j.comgeo.2014.02.004.

[GBMS21] Vinicius Mariano Gonçalves, Eduardo Mauro Baptista Bolonhez, Guilherme Esteves Mendes Campos, and Lara Hoffmann Sathler: Transmission line routing optimization using rapid random trees. *Electric Power Systems Research*, 194:107096, 2021, 10.1016/j.epsr.2021.107096.

[HNR68] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael: A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968, 10.1109/TSSC.1968.300136.

[HS97] John Hershberger and Subhash Suri: An optimal algorithm for euclidean shortest paths in the plane. *SIAM Journal on Computing*, 28:2215–2256, December 1997, 10.1137/S0097539795289604.

[HS18] Hao Hu and Renata Sotirov: Special cases of the quadratic shortest path problem. *Journal of Combinatorial Optimization*, 35(3):754–777, 2018, 10.1007/s10878-017-0219-9.

[HTMB12] Frank Hanssen, Jørn Thomassen, Roel May, and Kjetil Bevanger: A least cost path toolbox for optimal routing of high voltage power lines for a sustainable future. In *32nd Annual Conference of the International Association for Impact Assessment. Energy Future – The Role of Impact Assessment*, 2012, 10.13140/2.1.2908.1601.

[LMS01] Mark Lanthier, Anil Maheshwari, and Jörg-Rüdiger Sack: Approximating shortest paths on weighted polyhedral surfaces. *Algorithmica*, 30(4):527–562, 2001, 10.1007/s00453-001-0027-5.

[MMP87] Joseph S. B. Mitchell, David M. Mount, and Christos H. Papadimitriou: The discrete geodesic problem. *SIAM Journal on Computing*, 16:4, 1987, 10.1137/0216045.

[MP91] Joseph S. B. Mitchell and Christos H. Papadimitriou: The weighted region problem: Finding shortest paths through a weighted planar subdivision. *Journal of the ACM*, 38(1):18–73, 1991, 10.1145/102782.102784.

[PSRW18] Nadine Piveteau, Joram Schito, Martin Raubal, and Robert Weibel: A novel approach to the routing problem of overhead transmission lines. In *38. Wissenschaftlich-Technische Jahrestagung der DGPF und PFGK*, pages 798–801. Deutsche Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation e.V., 2018.

[RCH+18] Borzou Rostami, André Chassein, Michael Hopf, Davide Frey, Christoph Buchheim, Federico Malucelli, and Marc Goerigk: The quadratic shortest path problem: complexity, approximability, and solution methods. *European Journal of Operational Research*, 268(2):473–485, 2018, 10.1016/j.ejor.2018.01.054.

[RMFB15] Borzou Rostami, Federico Malucelli, Davide Frey, and Christoph Buchheim: On the quadratic shortest path problem. In Evripidis Bampis (editor): *International Symposium on Experimental Algorithms (SEA)*, volume 9125 of *Lecture Notes in Computer Science*, pages 379–390. Springer, 2015, 10.1007/978-3-319-20086-6_29.

[SdLP19] Afonso Henriques Moreira Santos, Rodolfo Mendes de Lima, and Camilo Raimundo Silva Pereira: Optimizing routing and tower spotting of electricity transmission lines: An integration of geographical data and engineering aspects into decision-making. *Electric Power Systems Research*, 176:105953, 2019.

[SST21] Lindsi Seegmiller, Takeshi Shirabe, and C. Dana Tomlin: A method for finding least-cost corridors with reduced distortion in raster space. *International Journal of Geographical Information Science*, 35(8):1570–1591, 2021, 10.1080/13658816.2020.1850734.

[Tom10] Dana Tomlin: Propagating radial waves of travel cost in a grid. *International Journal of Geographical Information Science*, 24(9):1391–1413, 2010, 10.1080/13658811003779152.

[WA21] Nina Wiedemann and David Adjiashvili: An optimization framework for power infrastructure planning. arXiv, 2021. `https://arxiv.org/abs/2101.03388`.

Erklärung

Hiermit versichere ich die vorliegende Abschlussarbeit selbstständig verfasst zu haben, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben, und die Arbeit bisher oder gleichzeitig keiner anderen Prüfungsbehörde unter Erlangung eines akademischen Grades vorgelegt zu haben.

Würzburg, den 21.07.2021

. . . . . .
Samuel Wolf