Edge Sparsification for Geometric Tour Problems

Sándor P. Fekete
Department of Computer Science, TU Braunschweig, Germany

Phillip Keldenich
Department of Computer Science, TU Braunschweig, Germany

Dominik Krupke
Department of Computer Science, TU Braunschweig, Germany

Eike Niehs
Institute for High Voltage Technology and Power Systems, TU Braunschweig, Germany

Abstract

We study a variety of sparsification approaches for a spectrum of geometric optimization problems related to tour problems, such as the Angular TSP, the Minimum Perimeter Problem, and the Minimum/Maximum Area Polygon Problem. To this end, we conduct a thorough study that compares the solution quality and runtime achieved by integer programming solvers on geometrically reduced edge sets to the exact solution on the full edge set; considered sparsification techniques include a variety of triangulations (Delaunay, Greedy, Minimum Weight), Theta and Yao graphs, Well-Separated Pair Decomposition, and Onion graphs.

We demonstrate that edge sparsification often leads to significantly reduced runtimes. For several of the considered problems, we can compute within a few seconds solutions that are very close to being optimal for instances that could not be solved to provable optimality within an hour; for other problems, we encounter a significant loss in solution quality. However, for almost all problems we considered, we find good solutions much earlier in the search process than for the complete edge set; thus, our methods can at least be used to provide initial bounds for the exact solution, demonstrating their usefulness even if optimality cannot be established.

Keywords and phrases edge sparsification, algorithm engineering, integer programming, tour problems

Digital Object Identifier 10.57717/cgt.v3i1.20

Supplementary Material `https://github.com/tubs-alg/edge_sparsification`

Funding This work was supported by DFG project "Computational Geometry: Solving Hard Optimization Problems" (CG:SHOP), FE407/21-1.

1 Introduction

Computing optimal or near-optimal solutions for hard optimization problems is a fundamental challenge of applied computation. For an optimization problem, typical approaches include (1) exact algorithms (e.g., based on integer programming) that compute provably optimal solutions for interesting benchmark instances; (2) fast lower (or upper) bounds for minimization (or maximization, resp.) problems, e.g., based on linear programming relaxations, that provide useful tools for judging the quality of solutions; (3) approximation algorithms (e.g., based on combinatorial constructions) that achieve feasible solutions in worst-case polynomial time with provable worst-case guarantees on the objective value; and (4) mere heuristics (e.g., based on local search) that provide *some* feasible solution, but without any performance bounds. Each of these approaches suffers from some deficiencies: (1) Exact solutions may not be achievable beyond limited instance sizes; (2) lower bounds may be relatively loose, and do not necessarily provide useful feasible solutions; (3) approximation algorithms may be too pessimistic and still leave large gaps; (4) heuristics usually do not provide any quality measure.

A critical issue when trying to compute good or even optimal solutions for hard optimization problems is the complexity of the relevant parameters for increasing instance sizes. In the case of geometric optimization problems, instances are typically characterized by n, the number of vertices or points. However, when aiming for optimal geometric structures (e.g., good polygonizations), the parameters involved in a problem model may have complexity $\Theta(n^2)$ (the number of edges), $\Theta(n^3)$ (the number of turns) or even $\Theta(n^4)$ (the number of intersections between edges). As a consequence, exact methods are faced with rapid growth of model complexity, sometimes even before the difficulty of computing optimal solutions kicks in.

To mitigate these complexities, *sparsification* techniques can be employed. These techniques simplify the model by reducing its complexity to a linear or quadratic number of variables and constraints. However, note that the mere size is not necessarily a good indicator for the difficulty of a model. A good sparsification technique not only makes the instance easier to solve but also ensures the retention of solution quality. Such sparsification is particularly beneficial for enhancing the scalability of mathematical models or heuristic approaches, often without necessitating significant alterations to the existing code. Furthermore, sparsification is invaluable in column generation-based methods, providing an effective initial model.

In this paper, we focus on achieving good (though not necessarily optimal) feasible solutions for a spectrum of NP-hard geometric optimization problems related to the classical Traveling Salesman Problem (TSP): for a given set of n points in the Euclidean plane, find a closed round trip that optimizes some geometric measure. The key objective is to combine exact methods with geometrically motivated *sparsification* of the considered edge set to a linear-sized and sometimes also non-crossing subset, thereby limiting the model size to $O(n)$ or $O(n^2)$ instead of $\Theta(n^4)$. This allows us to significantly extend the range of computable solutions. Our main emphasis is on the *practical* performance of such methods; to this end, we conduct a study of four natural geometric polygonization problems, for which we compare the solution qualities of exact solutions for various sparsified models and the exact solutions on the original edge set.

1.1 Our results

We conduct a thorough study that compares the solution quality and runtime achieved by integer programming solvers on geometrically reduced edge sets to the exact solution on the full edge set; considered sparsification techniques include a variety of triangulations (Delaunay, greedy, Minimum Weight Triangulation (MWT)), Theta and Yao graphs, Well-Separated Pair Decomposition (WSPD) and Onion graphs. For a variety of tour problems, we get a spectrum of results, as follows.

- For the *Angular TSP*, which aims at minimizing the total turn cost for a closed tour of n points in the plane, we found that our sparsification methods allow us to find reasonably good solutions much faster than for the complete instance. While the final optimality gap of around 5 % is not negligible, it takes significantly longer to reach this quality on the complete instances.
- For the *Angular-Distance TSP*, which aims at minimizing a linear combination of turn cost and distance cost for a closed tour of n points in the plane, we find that our sparsification methods seem to perform even better than for the Angular TSP. In particular, we find that even some of our sparsest sparsifications can find solutions that are within not more

than 4 % from the optimal solution within a few seconds for instances that cannot be solved to provable optimality within an hour. Denser sparsifications need slightly longer time, but then achieve a solution quality that is even within 1 % of the optimal solution, and are still achieved significantly faster than for the complete instances.

- For the *Minimum Perimeter Polygon Problem* (MP3), which aims at finding a polygon (possibly with holes) of minimum total perimeter on a set of n points, Fekete et al. [23] presented experimental results for instances of up to 600 points. They showed experimental results that indicate gaps between optimal solutions and solutions restricted to the edges of the Delaunay triangulation that on average are about 0.5 %. However, further insights, e.g., on runtimes, were not provided; we provide such insights. We further extend their research by applying a larger selection of sparsification methods on a broader range of instances. We find that some of our sparsification methods perform even better than the Delaunay triangulation.
- We also consider the *Optimal Area Polygon Problem*, which aims at finding a simple polygon of minimum or maximum enclosed area for a set of n points. For both problem variants, our practical results show that it is often much easier to find good solutions than to establish their quality (or even optimality) by establishing good (or even tight) bounds. For area minimization, we find that our sparsification methods do result in improved runtimes; however, the solutions tend to be of little value because the optimality gap is large. For area maximization, our methods appear to be more successful; nevertheless, the gaps are still growing with n and could become prohibitively large for even larger instances.

1.2 Related work

In the following, we first discuss related work on the considered optimization problems, and then on sparsification techniques.

1.2.1 Optimization problems

Sparsification for the classic Euclidean TSP (in which vertices correspond to points in the plane, and distances are measured according to the L_2 norm)) was studied by Letchford and Pearson [40], who considered the Delaunay and greedy triangulations for constructing a planar subset of edges, for which then a shortest tour is computed. On this sparsified set, they computed optimal solutions for all 58 instances of the TSPLIB [5] with at most 1500 points, and demonstrated that the resulting tours are on average within 0.1 % of optimality, showing the potential of sparsification. However, this does not address the suitability for a whole range of related problems that are not just based on minimizing the total perimeter of a simple polygon, but also on polygons with holes, as well as cost based on turns and area. We consider a larger spectrum of sparsifying structures, such as Minimum-Weight Triangulation (which is hard to compute in theory, but easy in practice), see Section 3.1; Theta and Yao graphs, see Section 3.2; Well-Separated Pair Decompositions, see Section 3.3; and Onion graphs, see Section 3.4. In addition, we also conduct experiments on a considerably richer set of instances than the relatively sparse set of TSPLIB instances; see Section 4.

The *Angular TSP* on Euclidean point sets was introduced and shown to be NP-hard by Aggarwal et al. [1]; further related work includes Fekete and Woeginger [29], who gave a spectrum of results for angle-restricted tours, Arkin et al. [3], who considered both general covering problems with turn cost and the restricted problem on grid graphs, and Fekete et al. [27], who studied edge covering problems arising from rotating satellites for communication.

The *Angular-Distance TSP* was introduced by Savla et al. [50] and further considered by Medeiros and Urrutia [43]. On the practical side, Staněk et al. [52] were able to compute exact solutions based on integer programming for instances with up to 75 points for Angular TSP, and with up to 100 points for Angular-Distance TSP.

Fekete et al. [22] considered the *Minimum Perimeter Polygon Problem* (MP3) and presented a generic integer programming formulation for several polygonization problems, which managed to solve instances with up to 50 points. Fekete et al. [23] have proved the NP-hardness of the MP3. Furthermore, they presented a more problem-specific integer program with additional cutting planes, enabling them to solve instances with up to 1000 points. They already studied sparsification techniques by restricting the initial edge set to edges of the Delaunay triangulation; it turned out that solutions on this restricted edge set achieve objective function values that lie on average within 0.5 % of the optimum.

The *Minimum Area Polygon* and *Maximum Area Polygon* problems were proven to be NP-hard by Fekete et al. [20, 21, 28], who also gave a 0.5-approximation for Maximum Area Polygonization. There are practical approaches with a focus on heuristics [47, 48, 53]. Exact methods were considered by Fekete et al. [22], who have solved instances with up to 16 (for minimum area) and 19 points (for maximum area). The recent 2019 CG Challenge [16], based on a wide range of benchmark instances, brought some progress; see Demaine et al. [16] for an overview, and [15, 18, 33, 39, 49] for a variety of heuristic approaches. Moreover, see Fekete et al. [25] for progress on exact methods. Note that even the latter ceases to achieve optimal solutions for instances larger than 25–30 points.

1.2.2 Sparsification

The worst-case ratio t between the shortest distance in a substructure S and the full structure F is called the *stretch factor*; if t is bounded, then S is a t-spanner of F. Most of the existing body of work focuses on theoretical worst-case bounds; for a broad overview, see the book by Narasimhan and Smid [46] on geometric spanners. To the best of our knowledge, practical studies have been limited: Farshi and Gudmondson [19] considered geometric spanners for randomly generated instances with uniform or normal distribution and 100 to 10 000 points.

There is a substantial body of work on worst-case stretch factors for planar (and thus, of cardinality $\Theta(n)$) subsets of edges between n points in the plane; e.g., see the survey by Bose and Smid [8]. A lower bound of $\sqrt{2.005367532} \approx 1.41611$ was shown by Mulzer [44] and recently improved to 1.4308 by Dumitrescu and Ghosh [17]. They also showed a lower bound of 2.0268 for the greedy triangulation. Chew [13] showed that the L_1- and L_∞-metric Delaunay graph is a $\sqrt{10}$-spanner; this was improved by Bonichon et al. [6] to $\sqrt{4+2\sqrt{2}}$, which is tight in the worst case. The best known upper bound for the Delaunay triangulation is 1.998, as shown by Xia [55]. If a constrained plane geometric graph has the α-visible diamond property for each of its edges and the d-good polygon property, it is a $\frac{8d(\pi-\alpha)^2}{\alpha^2 \sin^2(\alpha/4)}$-spanner of the visibility graph of P (point set) and L (constrained edges that block visibility); see [7]. Every triangulation has the 1-good polygon property and the MWT has the diamond property for $\alpha = \pi/4.6$, implying that the MWT is a 3591.33-spanner.

Because of the close connection to practical exploitation and implementation, more specific related work on particular sparsification techniques is provided in the respective parts of Section 3.

2 Integer Programming models

Now we describe the integer programming models that we use to solve sparsified and complete instances of our problems. The Angular TSP and Angular-Distance TSP are two special cases of the *Quadratic TSP* (QTSP), which was originally introduced by Jäger and Molitor [37], based on an application in biology. Asymmetric QTSP instances that arise from this application have been solved with up to 100 vertices [30]. In the QTSP, instead of the edge costs c_{ij} that we have to pay for each edge ij that we use in the regular TSP, we incur costs for all pairs of incident edges in the tour. In other words, we have to pay some cost c_{ijk} if we use the edges ij and jk, i.e., if our tour visits the vertices i, j, k consecutively in that order. Thus, the Angular TSP is a special case of the QTSP in which the costs c_{ijk} correspond to the cost of the turn i, j, k. Similarly, the Angular-Distance TSP can be cast as a special case of the QTSP by setting c_{ijk} to the cost of the turn i, j, k plus half the cost of each of the edges ij and jk. We use the integer programming model by Fischer et al. [30], which has one binary variable x_{ijk} for each turn i, j, k, to solve both the Angular and the Angular-Distance TSP.

In the Minimum Perimeter Polygon Problem (MP3), in contrast to the TSP, holes within the polygon are allowed, so not every occurring subtour is necessarily invalid. Therefore, Dantzig's classical IP formulation (based on the well-known Subtour Elimination constraints) cannot be applied in a straightforward manner; instead, the hierarchical geometric structure of the MP3 requires the use of more advanced cutting planes, as provided by Fekete et al. [23]. We make use of these to model and solve the MP3.

For the Optimal Area Polygon Problem, we use the formulations proposed by Fekete et al. [24]. Their experimental evaluation has shown that the minimization and the maximization variants require different models to achieve a reasonable performance. For area maximization, they used an edge-based integer program, which has one binary variable for each directed edge. The area is deduced by creating a single triangle per edge to a reference point and adding or subtracting its area depending on the direction of the edge. For area minimization, they used a triangle-based formulation that has one binary variable for each empty triangle of the given point set.

Because the details of these formulations are not crucial for the following analysis, we refer to the respective papers for more detailed descriptions of these formulations.

3 Sparsification methods

The complete graph on a given set P of n points contains all $\Theta(n^2)$ possible edges on the points from P. Furthermore, the number of constraints when aiming for a non-crossing set of edges is even larger: As shown by Lovász et al. [42], any set of n points in the plane contains $\Theta(n^4)$ convex quadrilaterals, and thus, $\Theta(n^4)$ crossing pairs of edges. As a consequence, employing straightforward IP formulations quickly becomes impractical. Our aim is to replace the complete graph by a suitable subgraph with a strongly reduced edge set (but the same points) such that we can, in practice, solve larger instances of our problems without drastically degrading our solution quality. Generally, our goal is to reduce the number of possible edges to $O(n)$. In this section, we describe the sparsification methods we considered as candidates for the generation of such subgraphs; we experimentally evaluate their suitability for the considered problems in Section 4. Examples for the sparsified graphs can be found in Figure 3.

3.1 Triangulations

Triangulations are a particularly well-studied type of sparse subset of the set of all edges on a point set P. Furthermore, triangulations have been applied to find an initial (or reduced) edge set for the exact (or heuristic) solution of several tour problems [23], including the seminal work of Applegate et al. [2] on the TSP. We consider some of the most famous and well-studied types of triangulation, namely the *Delaunay Triangulation* (DT(P)), *Minimum-Weight Triangulation* (MWT(P)), and *Greedy Triangulation* (GT(P)). The Delaunay triangulation maximizes the minimum angle of all triangles and can be computed in $O(n \log n)$ time. We use the CGAL [54] implementation to compute Delaunay triangulations.

The Minimum-Weight Triangulation (MWT) minimizes the total weight of all edges. In theory, the MWT(P) is NP-hard to compute, as shown by Mulzer and Rote [45]. While this establishes hardness from a theoretical perspective, Haas [35] has demonstrated that by combining clever local heuristics (previously considered by Beirouti and Snoeyink [4]) based on geometric insights, along with an array of modifications, parallelizations, and improved geometric encodings and data structures, it becomes feasible to compute provably optimal solutions for instances containing up to 30 000 000 points in less than 4 minutes. As a consequence, computing and using an optimal MWT is a practically feasible approach. We used an extended [26] implementation of Haas's procedure [35] to compute the MWT for our experiments.

The greedy triangulation attempts to approximate the MWT by iterating over all edges in non-decreasing order of their length, inserting the edges that do not cross any previously inserted edges. While this naive method runs in time $O(n^3)$, the greedy triangulation can also be computed in subquadratic time [41]. For our experiments, we use the algorithm of Goldman [32] that runs in $O(n^2 \log n)$ time and utilizes a constrained Delaunay triangulation for its construction.

3.2 Theta and Yao graphs

The *Theta*$_k$ and *Yao*$_k$ graphs are two similar types of graphs, based on the idea of subdividing the plane around each point $p \in P$ into k cones of equal angle $\Theta = \frac{2\pi}{k}$ and connecting p to the closest point in each cone. The graphs differ in how they define the closest point in each cone. Both are sparse spanners for constant $k > 1$, have at most $k \cdot n$ edges, and are typically not planar. Note that these graphs do not immediately have vertices of bounded degree. Theta graphs were introduced by Clarkson [14] to approximate a motion planning problem with obstacles, and independently by Keil [38]. Yao graphs were introduced by Flinchbaugh and Jones [31], and Yao [56]. Both graphs can be computed in $O(n \log n)$ time [46, 11]. We use the CGAL library [54] to compute these graphs.

The closest point in each cone is defined as follows. In Theta graphs, each cone originating from a point p has an associated ray ℓ, which is the bisector of that cone, and the closest point in a cone is the one whose orthogonal projection to ℓ is closest to p. In Yao graphs, the closest point in each cone is the one closest to p w. r. t. the Euclidean distance.

3.3 Well-Separated Pair Decomposition

The *Well-Separated Pair Decomposition* (WSPD) was introduced by Callahan and Kosaraju [9, 10] for efficiently computing the k-nearest neighbors. There are many other applications, such as approximating the Euclidean Minimum Spanning Tree or solving closest pair problems, see [51] for a comprehensive overview.

A WSPD graph of a point set P with some given separation factor s induces a t-spanner with $t = \frac{(s+4)}{(s-4)}$ for $s > 4$ by selecting an arbitrary edge $pq, p \in A_i, q \in B_i$ for each well-separated pair A_i, B_i in the WSPD [9]. A WSPD of a point set P with respect to separation factor $s > 0$ can be computed in $O(n \log n)$ time. Arbitrarily selecting edges between well-separated pairs always results in a t-spanner, and the resulting number of edges for a WSPD is always the same. However, the suitability of the resulting edge set may depend on how the endpoints are selected. The used implementation is based on the description of Smid [51]. We use three different variants to create a sparsified graph from the WSPD, which differ in how they select the edge for connecting the well-separated pairs:

Variant 1: For every well-separated pair A, B, choose $p \in A$ and $q \in B$ with minimum distance $d(p, q)$.

Variant 2: Choose the edges based on an auxiliary integer program that minimizes the maximum degree of all vertices. As the integer program is not guaranteed to yield a solution within a reasonable time for all instances, we are not able to use this variant for all instances.

Variant 3: Choose the edges using a greedy algorithm that tries to minimize the maximum degree by selecting some $p \in A$ and $q \in B$ with $deg(p)$ minimal in A and $deg(q)$ minimal in B for every well-separated pair A, B.

3.4 Onion graphs

Chazelle [12] provided a space- and time-optimal algorithm to compute the convex layers of a point set. This algorithm runs in $O(n \log n)$ time and requires $O(n)$ space. The convex layers of a point set P can be obtained by repeatedly computing the convex hull of P and removing these points until no more points are left. We use this principle to compute the convex layers, and finish up by merging successive layers with one of the following strategies. We call these graphs *Onion graphs*.

onion-0 Connect each vertex to all vertices of the next layer.

onion-1-k Connect each vertex to the k closest vertices of the next layer.

onion-2-k Given a vertex v on layer L. Let u, respectively w, be the predecessor, respectively successor of v on L. We connect v to the k vertices of the next layer with the lowest turning angle when moving from u to v. Analogously, we insert k edges with lowest turning angle when moving from w to v.

3.5 Overview

We adopt the following notation to identify sparsification methods and their parameters.

- *mwT*, *greedyT* and *delaunayT*, are the Minimum-Weight, greedy and Delaunay triangulations,
- *onion-c-k* is the Onion graph, with c encoding which of the three inter-layer connection methods described in Section 3.4 we use, and k denoting the connection parameter,
- *theta-k-o* and *yao-k-o* are the Theta and Yao graphs, with k denoting the number of cones, and o encoding whether we take only even ($o = 0$), only odd ($o = 1$) or all cones ($o = 2$), and
- *wspd-t-c* is the WSPD, with $t \in \{2, \dots, 8\}$ encoding the guaranteed stretch factor, and $c \in \{1, 2, 3\}$ denoting how the well-separated pairs were connected, as described in Section 3.3.

If a sparsification method depends on parameters, for our experimental evaluation, we consider the parameters outlined above.

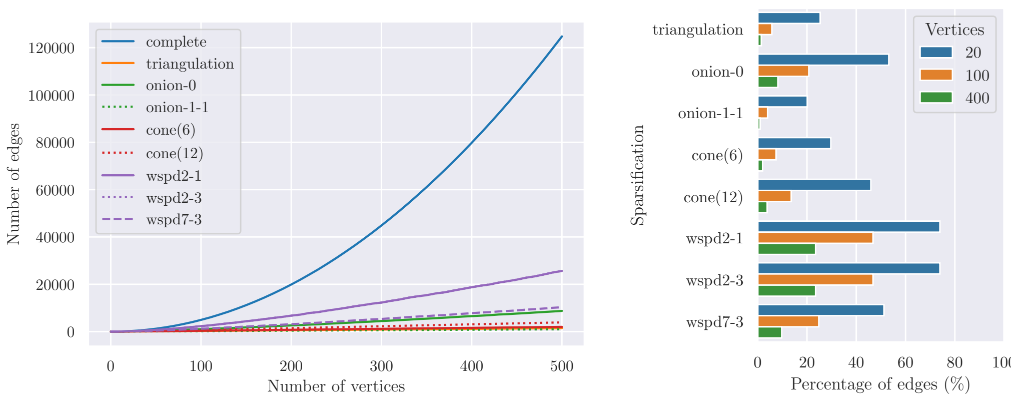


Figure 1 Average number of edges for a selection of sparsification methods. **(Left)** The line plot shows how the absolute number of edges increases with the number of vertices. **(Right)** The bar plot shows the relative number of remaining edges compared to the full graph for fixed numbers of vertices ($\{20, 100, 400\}$). All triangulations and all full k-cone graphs for fixed k contain the same number of edges; therefore, they are combined in the figure. We see in the bar plot that especially the Well-Separated Pair Decompositions still have around 50 % of the edges in a graph with 100 vertices. Among all depicted methods, the triangulations have the smallest numbers of edges. As all considered sparsification methods grow linearly in size (as shown in the line plot), all methods show a significantly reduced edge set for larger instances. The instances were taken from the benchmark set described in Section 4.1.

The different sparsification methods result in vastly different total edge counts, though all of them become small compared to the full edge set for larger n. Figure 1 shows the number of edges over a large set of sample instances. Generally speaking, of all methods under consideration, triangulations have the smallest numbers of edges, followed by cone graphs and Onion graphs. The WSPD-based sparsifications tend to have the largest number of edges; their edge counts grow when reducing the spanning ratio t. (Note that the numbers of edges do not yet indicate the actual performance for a sparsified set; this is evaluated in the following section.)

3.6 Infeasible instances

When we restrict our edge set, we cannot always guarantee that the problem we are interested in still has a solution on this reduced set. In particular, for the Angular and Angular-Distance TSP, the resulting graph needs to be Hamiltonian. Additionally, for Optimal Area Polygonization, the Hamiltonian cycle needs to be non-crossing. For the MP3, the graph needs to contain a polygon (with holes) on all vertices. As some type of Hamiltonicity plays a role for all polygonizations, we depict the fraction of non-Hamiltonian instances resulting from our sparsification methods in Figure 2. We see that half-cone graphs are often non-Hamiltonian; we thus disregarded them for all ensuing evaluations.

4 Experimental evaluation

Now we evaluate the practical impact of the edge sparsification methods presented in Section 3 on the solution of our problems using the integer programming models presented in Section 2.

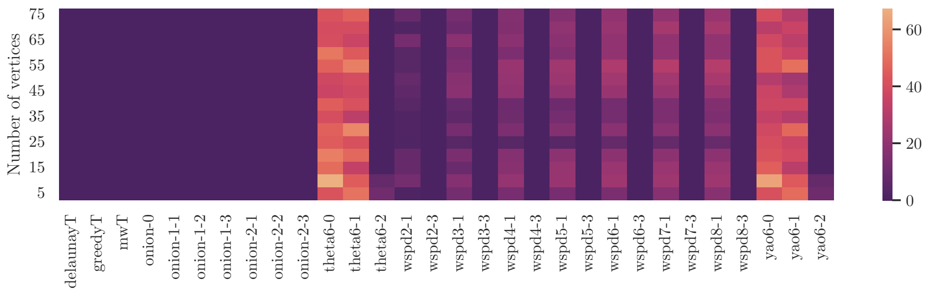


Figure 2 Number of non-Hamiltonian sparsifications. The color of each field indicates the percentage of non-Hamiltonian graphs for a sparsification method and instance size. Brighter columns correspond to sparsification techniques that yield many non-Hamiltonian graphs.

We evaluate three different elements in our experiments, highlighting three different aspects: (potential) solution quality, runtime, and solution quality over time.

Solution quality: How good are the solutions on the sparsified graphs compared to the complete graph? To assess the quality of solutions on the sparsified graphs in comparison to the complete graph, we face the challenge of computing optimal solutions for both scenarios. However, this may not always be feasible, prompting us to solve the integer programs with generous time limits and utilize the best solution achieved within that time frame. Instead of comparing to a possibly non-optimal solution on the complete graph, we opted to benchmark against the best lower bound provided by Gurobi or CPLEX for the complete graph. By dividing the objective value by this lower bound, we obtain a reliable indicator of solution quality, with 1.0 representing optimality.

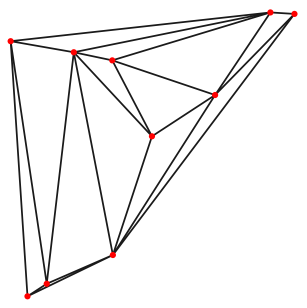
We could also use lower bounds for the sparsified graphs, to get a true lower bound on the achievable solution quality, but this would give an undesirable advantage to sparsified graphs in which the optimal solution is difficult to find, contradicting the primary objective of simplifying the involved computation.

Our analysis involves using a heatmap to identify the most intriguing sparsification methods, as well as a line plot for a detailed examination of these.

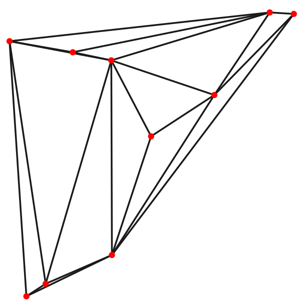
Runtime: How much faster can we solve the problem on the sparsified graphs compared to the complete graph? Determining the extent of speed improvement when solving the problem on sparsified graphs as compared to the complete graph poses a challenging question. Numerous methods exist to solve these problems beyond our integer programs, making a definitive answer elusive. Nonetheless, employing state-of-the-art integer programming solvers and carefully chosen formulations allows us to assume that the runtime differences between the complete and sparsified graphs serve as a valuable indicator for the speed-up of other algorithms.

To gain insights, we employ a heatmap to visualize the percentage of instances solved within a short time limit, enabling us to identify the most intriguing sparsification methods. The short time limits for each problem are carefully fine-tuned to reveal differences between sparsification techniques, while ensuring a reasonable number of solved instances. Subsequently, we use a line plot to delve into the concrete runtime of these selected methods.

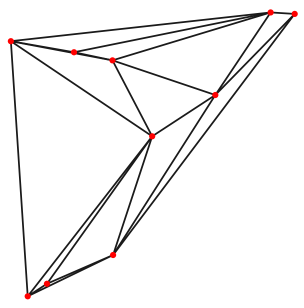
Solution quality over time: How efficiently can we discover good solutions on the sparsified graphs in comparison to the complete graph? This question provides deeper insights into how sparsification impacts the search process, extending the previous question, which



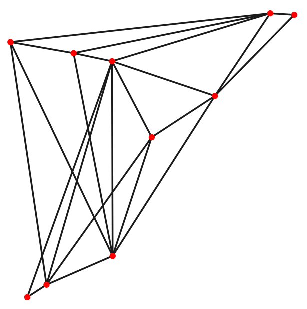
(a) Delaunay triangulation: $n = 10$, $m = 22$.



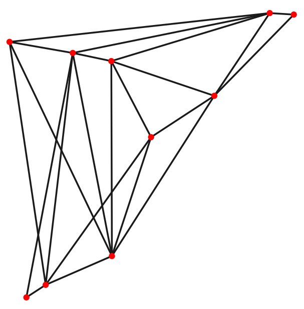
(b) Greedy triangulation: $n = 10$, $m = 22$.



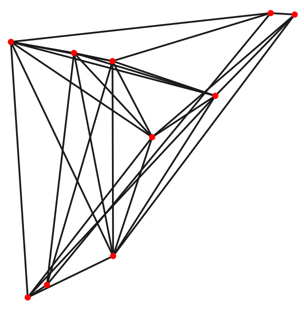
(c) Min-Weight Triangulation: $n = 10$, $m = 22$.



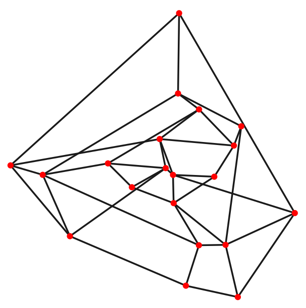
(d) Theta (*theta-6-2*): $n = 10$, $m = 22$.



(e) Yao (*yao-6-2*): $n = 10$, $m = 22$.



(f) WSPD (*wspd-8-3*): $n = 10$, $m = 29$.



(g) Onion graph (*onion-1-1*): $n = 20$, $m = 40$.

Figure 3 Examples of the various sparsification types, with n points and m edges. The example for the Onion graph is a different instance because the smaller instance would not show any characteristics.

only considered the runtime until termination. In some instances, the integer program on the complete graph may quickly find a good solution but may still require a long time to prove optimality. In such cases, the integer programming solver gains no advantage from the sparsified graph. However, if the integer program exhibits significant speed-up on the sparsified graph, it suggests that the sparsification could also be beneficial for computing optimal solutions via column generation or other approaches.

To assess the quality over time, we monitor the objective values in Gurobi and compare them to the best proven lower bound on the complete graph. This metric is identical to the previously used one, but measured continuously over time. Due to the use of a different implementation based on CPLEX for the two area optimization problems, we lack this data, and exclude them from this specific analysis.

4.1 Instance types

Our research questions were addressed by conducting computational experiments on a variety of meticulously crafted benchmark instances with varying types and sizes. To this end, we generated instances from five specific categories. We obtained *uniform instances* by picking points uniformly at random within a square with some side length a; the coordinates can be restricted to integers (`uniform_int`) or not (`uniform_real`). Instances of type `normal` arise by picking point coordinates independently according to a normal distribution with mean 0 and some standard deviation σ.

We also generated instances based on images by deriving a two-dimensional density function on some rectangular subregion of $\mathbb{R}^2$ from some feature of the image's pixels. In particular, we produced `image_brightness` instances by mapping the brightness distribution of an image to a density function from which points were sampled. `image_edge` instances were produced by first applying an edge-finding algorithm to an image, resulting in a preprocessed image in which pixels that are likely to contain an edge of the original image are black and all other pixels are white. We then sampled points from regions corresponding to black pixels. As images, we used satellite imagery (illumination by night), images of space telescopes, famous art, portraits of politicians, and some random photos, such as a picture of our university building.

For each of the previous instance types, we generated 10 instances for every size from 5 to 600 with step size 5. In addition, instances from the `TSPLIB` [5] were used; note that their quantity is low compared to the other instance types. As their total number is also low, they only have a minor impact on the overall results for the total set of 7007 instances.

4.2 Experimental environment

We ran our experiments on the following computing platforms, each with a common software environment based on Ubuntu 20.04 LTS:

algpc Intel Core i7-6700K CPU with 4 cores and 8 threads clocked at 4.00 GHz base-frequency with 64 GB of RAM.

algry AMD Ryzen 7 5800X CPU with 8 cores and 16 threads clocked at 4.00 GHz base-frequency with 128 GB of RAM.

We use the MIP solvers IBM ILOG CPLEX 12.9 [36] for optimal area polygonization and Gurobi Optimizer 9.1.1 [34] for all other problems, both with their default parameter settings.

4.3 Angular TSP

Recall that we use the integer programming formulation by Fischer et al. [30] to solve the Angular TSP: To deal with rounding issues, this involves scaling the actual angles in radians by 1000 and then rounding them to 12 decimal places to obtain the actual costs c_{ijk}. All Angular TSP experiments were run on **algpc**. We evaluated our approach on instances of size 5 to 75, resulting in 758 instances in total.

We disregarded all sparsification techniques for which less than 95 % of the edge sets were Hamiltonian. The results of our experiments for the remaining methods are summarized in Figures 4–6.

Our analysis reveals a consistent advantage in runtime by the sparsified instances. Among the sparsifications evaluated, those based on the Well-Separated Pair Decomposition (WSPD) stand out for delivering stable and reasonable solution quality, maintaining an optimality gap of less than 4 % (measured at $n = 70$). In contrast, the other approaches exhibit rapidly increasing optimality gaps, quickly exceeding 20 % and even exceeding 50 %.

For example, in cases with 60 vertices, the computation time for the model using the complete graph averaged 893 s. Meanwhile, the *wspd-2-1* model required significantly less time, averaging 156 s, and the *onion-1-1* model terminated even faster, in just 0.1 s. This disparity raises an intriguing question about the potential utility of the faster, yet less accurate methods, such as the *onion-1-1* sparsification, in generating initial solutions.

Observing the optimality gap over time, it becomes apparent that WSPD-based methods almost match the onion-based methods in speed for arriving at high-quality solutions. They quickly achieve a high solution quality, but then require a prolonged period to confirm optimality. The Onion and Yao graph methods initially provide a slightly better solution quality, but only for the first few seconds. Therefore, if the primary goal is not solely rapid solution generation, the *WSPD-2-1* method emerges as the preferable option for the Angular Traveling Salesman Problem.

4.4 Angular-Distance TSP

When the cost of a tour does not only depend on its turning angles, but also on the edge length of the involved edges, the Angular TSP turns into the Angular-Distance TSP, for which travel costs arises as a linear combination of the turning angle and the mean edge length. Following the literature [30, 52], we set the edge costs to

$$c_{ijk} = 40\alpha_{ijk} + \frac{d_{ij} + d_{jk}}{2},$$

where α_{ijk} is the turning angle for the turn i, j, k in radians. According to previous studies, this value is a suitable choice—neither too close to the TSP nor too close to the Angular TSP—for instances with integer coordinates uniformly sampled from a square of side length 500. Just like [30, 52], we scale all instances to coordinates in the range $[0, 500]$; the costs are then scaled by 100 and rounded to 12 decimal places. (Because of the linear combination of distance and angle, this differs from the pure Anguler TSP, where a factor of 1000 was used.)

We evaluated instances ranging from 5 to 90 points, resulting in a total of over 900 instances. All Angular-Distance TSP experiments were run on **algry** using the same integer programming implementation as for the Angular TSP, only with a different cost function. The results of our experiments are summarized in Figures 7–9.

Again we find that solving the problem on the sparsified graphs has a significant advantage w. r. t. runtimes. The loss of solution quality on the sparsified graphs for the Angular-Distance TSP is also significantly less than for the Angular TSP. However, for most sparsification

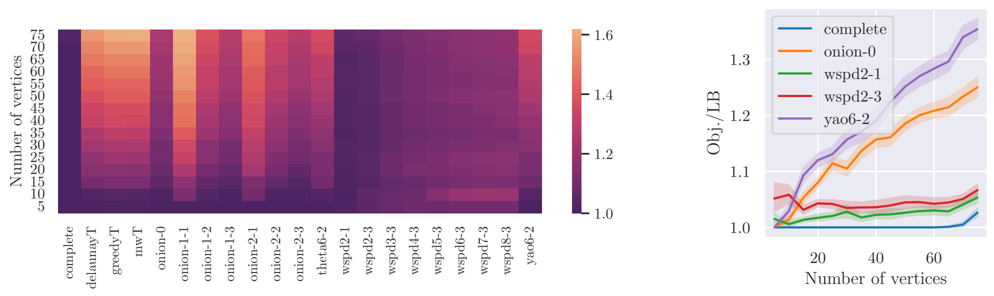


Figure 4 Solution quality for Angular TSP, measured in objective value divided by best known lower bound for the corresponding instance (1.0 is optimal). **(Left)** The heatmap provides an overview of all sparsification techniques. Brighter columns show a higher deviation from the optimum. If the top of the column gets brighter, the solution quality decreases with the instance size. The triangulations show particularly bad solutions. **(Right)** The line plot shows the mean solution quality over instance size for a selected number of promising candidates. *wspd-2-1* shows a stable optimality gap of around 4 %. The optimality gaps of *onion-0* and *yao-6-2* start low but are linearly increasing, far above the previous two.

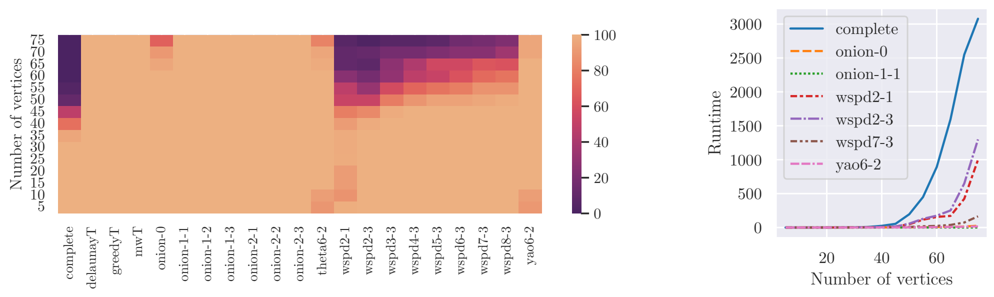


Figure 5 Runtime for Angular TSP. **(Left)** The heatmap measures the runtime in percentage of instances that could be solved within 30 s to proved optimality (on the limited edge set). Brighter columns indicate a better mean runtime. While the WSPD yields a high solution quality, the runtime is also relatively high but better than on the complete graph. The other sparsification techniques are significantly faster. **(Right)** The line plot shows the mean runtime over instance size for promising candidates. *wspd-7-3* is much faster than *wspd-2-1* and *wspd-2-3*, which sometimes need almost half the complete graph's solution time. *onion-0* and *yao-6-2* are almost immediately solved.

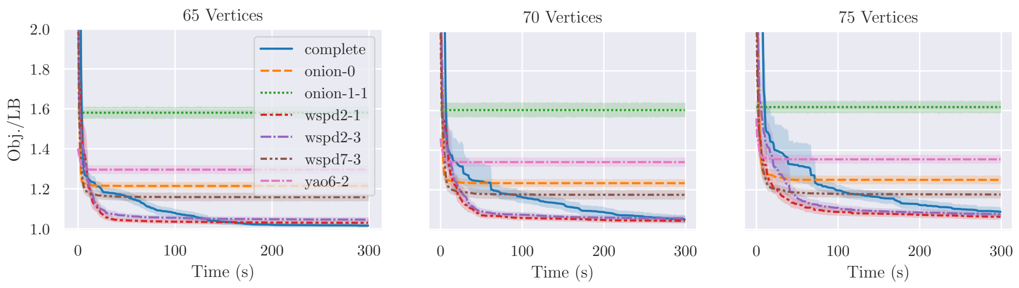


Figure 6 Average optimality gap during the optimization over time for Angular TSP in Gurobi (without model building) for instances with 65, 70, resp. 75 vertices. The complete graph reaches good objectives reasonably fast; however, in the first seconds, the sparsified graphs perform considerably better. The WSPD-graphs can even maintain this lead over several minutes. While solving them to proved optimality still takes a lot of time, they reach near optimality very quickly and significantly faster than the complete graph. Further, despite the later termination, *wspd-2-1* reaches the solution quality of the quickly terminating onion approaches similarly fast, it just continues to improve afterward. The runtimes in Figure 5 can thus be misleading on their own.

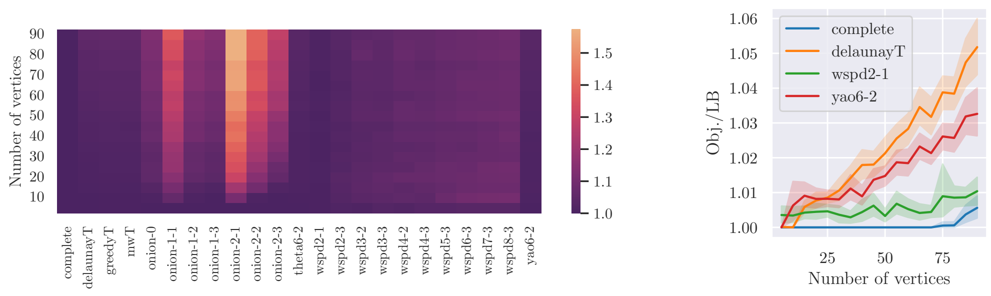


Figure 7 Solution quality for Angular-Distance TSP, measured in objective value divided by best known lower bound for the corresponding instance (1.0 is optimal). **(Left)** The heatmap provides an overview of all sparsification techniques. Brighter columns show a high deviation from the optimum. If the top of the column gets brighter, the solution quality decreases with the instance size. The solution quality is generally much better than for Angular TSP; all sparsification methods except for the Onion graphs achieve relative gaps below 10 %. **(Right)** The line plot shows the mean solution quality over instance size for a selected number of promising candidates. *wspd-2-1* shows a stable optimality gap of less than 1 %. The optimality gaps of the other sparsification techniques are worsening with the instance size.

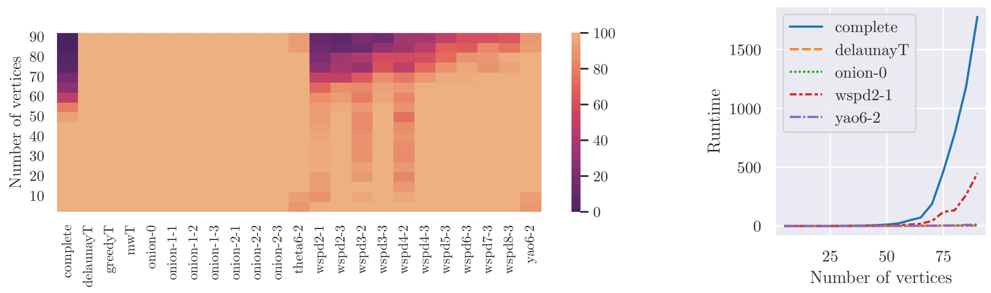


Figure 8 Runtime for Angular-Distance TSP. **(Left)** The heatmap measures the runtime in percentage of instances that could be solved within 15 s to proved optimality (on the limited edge set). Brighter columns indicate a better mean runtime. The runtime is similar to Angular-TSP; the WSPD-graphs are slower than the others, but faster than the complete graph. **(Right)** The line plot shows the mean runtime over instance size for promising candidates. *wspd-2-1* takes nearly half the time of the complete graphs, while the other techniques have a negligible runtime.

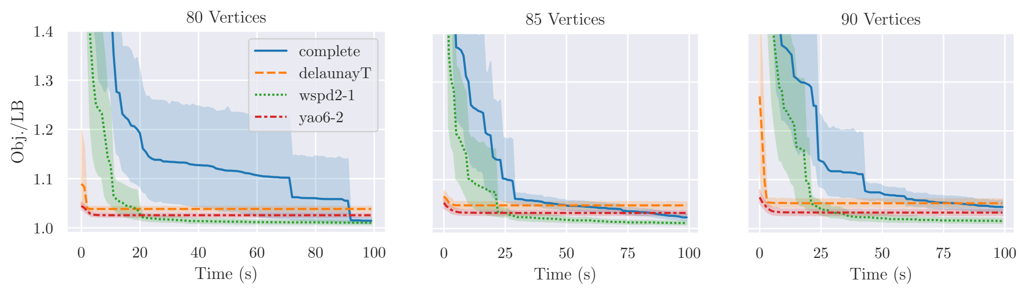


Figure 9 Average optimality gap during the optimization over time for Angular-Distance TSP in Gurobi (without model building) for instances with 80, 85, 90 vertices, resp. Similar to Angular TSP, Gurobi finds a near-optimal solution within the first two minutes, even on the complete graph. The full-cone Yao graph with 6 cones finds solution with an optimality gap of less than 4 % within the first seconds, but barely improves afterward. The WSPD graphs need around 30 s, but then yield solutions with an even better gap of only around 1 %. The complete graph often needs several minutes to find a competitive solution.

methods, the relative gap to the optimal solution still seems to grow with n. The WSPD may again provide a significant improvement in runtime with a low and stable gap. If we consider the task of finding good upper bounds during the solution process, the sparsification methods again produce good solutions much earlier than the solution process on the complete graph and may be used to provide useful initial solutions for computing optimal solutions. In contrast to the Angular TSP, the lower quality sparsifications actually have visibly better solutions than the WSPD-based ones for multiple seconds. For instances with 90 vertices, *wspd-2-1* needs around 30 s to reach a lower optimality gap than *yao-6-2*. For the Yao graphs, we find that it can take the solver on the complete graph more than 90 s to find a solution that is as good as the one found on the sparsified graph within 5 s. Furthermore, for several of the fast sparsification methods, we were able to compute within a few seconds a solution of at most 5 % above the optimum value for some instances that could not be solved to optimality within one hour. This implies that it coule be beneficial for an approach based on column generation to start with *yao-6-2*, and then add the edges of *wspd-2-1* in the next iteration.

4.5 Minimum perimeter polygons

Our experimental results on the Minimum Perimeter Polygon Problem (MP3) are based on instances ranging from 5 to 600 points, resulting in a total of over 6000 instances. All MP3 experiments were run on **algry** utilizing the integer programming formulation by Fekete et al. [23]. We set a time limit of 600 s.

Our experimental findings, detailed in Figures 10–12, highlight the significant benefit of sparsification in solving MP3 problems. Remarkably, in cases involving up to 300 points, sparsification methods like MWT or greedy triangulation achieved solutions with an optimality gap under 0.35 % in mere seconds. This performance surpasses that of the Delaunay triangulation method, initially proposed by Fekete et al. [23].

Apart from onion hulls and WSPDs, which showed subpar results, other sparsification techniques consistently yielded an optimality gap near 0.5 %. However, as the problem size increases, the optimality gap for larger instances rises to approximately 1.2 % for MWT or greedy triangulation and 1.5 % for Delaunay triangulation, *theta-6-2*, and Yao graphs. For these larger instance sizes, the solver struggles to find a feasible solution on the complete graph within 600 s, which potentially also weakens the lower bounds used for evaluations.

A plausible explanation for the difficulty of obtaining feasible solutions on the complete graph is that the majority of found integral solutions are infeasible, i.e., they have forbidden holes that have to be prevented by lazy constraints. While the MP3 is less constrained than the TSP, separating infeasible solutions needs more precision, and is thus more difficult. Here the sparsified models offer an additional advantage besides their significantly fewer variables: The sparsified graphs allow fewer perturbations that can be used to circumvent the newly added constraints. This allows us to rapidly obtain near-optimal solutions on instance sizes for which the IP solver struggles to even find feasible solutions on the complete graph.

It should be noted that our model's implementation appears less efficient compared to the one by Fekete et al. [23]. This observation can be attributed to a couple of key factors. Firstly, unlike the approach by Fekete et al., we do not employ warm starts and column generation. Secondly, our study utilizes a more diverse set of instances, which may contribute to the observed differences in efficiency.

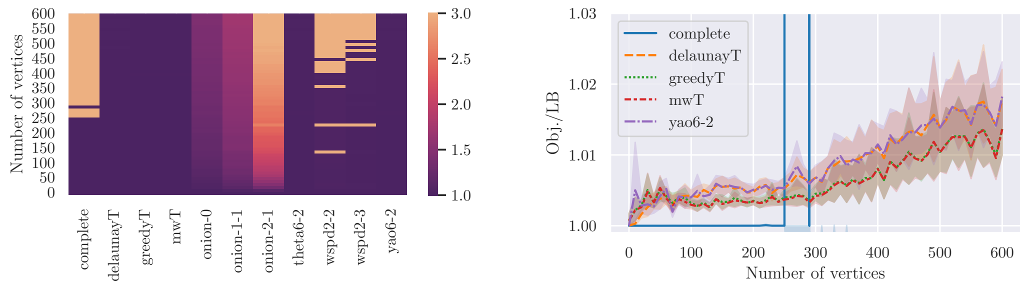


Figure 10 Solution quality for MP3, measured in objective value divided by best known lower bound for the corresponding instance (1.0 is optimal). **(Left)** The heatmap provides an overview of all sparsification techniques for which at most 5 % of the edge sets were non-hamiltonian. Brighter columns show a high deviation (clipped at 3.0) from the optimum, thus, a worse performance. If the top of a column gets brighter, the solution quality decreases with the instance size. The solution quality is excellent for all considered sparsification techniques, except for the onion hulls and the WSPD. **(Right)** The line plot shows the mean solution quality over instance size for a selected number of promising candidates. For the complete graph, it can become difficult to find feasible solutions for larger instances, resulting in some erratic behavior around 250 points. Up to this size, Minimum Weight and greedy triangulations have an optimality gap of less than 0.35 %. Delaunay triangulation, theta-6-2, and Yao graphs are slightly above 0.5 %.

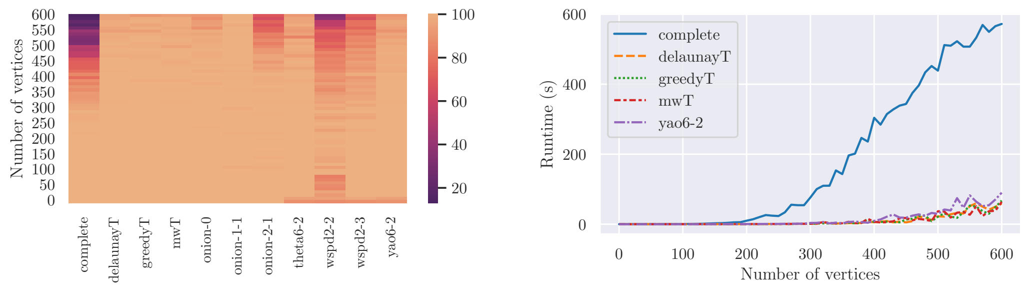


Figure 11 (Left) Percentage of MP3 instances that could be solved to optimality within 600 s. All infeasible instances are counted as unsolved. Brighter columns are better. We see that triangulations and cone-based graphs are all fast, finishing within mere seconds, whereas *wspd-2-2*, *wspd-2-3* and *onion-0* have a visibly higher runtime. **(Right)** Mean runtime over instance size for selected sparsification techniques.

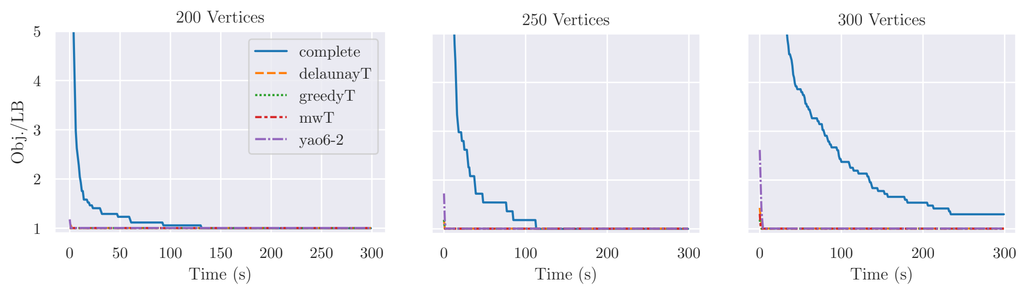


Figure 12 Average optimality gap during the optimization over time for MP3 in Gurobi (without model building) for instances with 200, 250, 300 vertices, resp. All considered sparsification techniques reach a near-optimal solution equally fast, with the minimum weight and greedy triangulation reaching slightly better solutions over time. The sparsification techniques are significantly faster than working on the complete graph.

4.6 Area-optimal polygons

Among all problems we consider, the optimal area polygonization problems appear to be the hardest to solve in practice. The best approaches that we are aware of are only sufficient for solving instances of up to 25 points. All experiments were run on **algpc**, utilizing the two integer programming formulations by Fekete et al. [24], which we adapted to sparse graphs. We used instances ranging from 5 to 23 points, resulting in more than 950 instances in total.

As discussed in Section 2, we used different formulations for area minimization, where we used a *triangle-based formulation*, and area maximization, where we employed an *edge-based approach*. This affects the feasibility of instances, as the triangle-based formulation only works if the edge set contains an n-vertex (simple) polygon and a triangulation, whereas the edge-based approach only requires an n-vertex polygon. Compared to the situation for Angular TSP, this does not only exclude half-cone graphs, but also all WSPD-based sparsifications, in particular those with larger stretch factors. For area minimization, all WSPD-based spanners resulted in more than 10 % infeasible instances. For this reason, we disregarded the WSPD-based sparsifications for area minimization, but included them for area maximization.

4.6.1 Area minimization

Figure 13 summarizes our experimental results with respect to runtimes and optimality gaps for area minimization. The different sparsifications achieve fast, almost indistinguishable runtimes, *theta-12-2* and *yao-12-2* being the slowest among them. In terms of runtime, the triangle-based formulation gives triangulations an advantage in the solution process because there are no overlapping triangles.

Triangulations and 6-cone-graphs result in solutions with average gaps above 100 % on larger instances with ongoing continuous growth. 12-cone graphs achieve lower gaps of (on average) less than 100 %, but are also getting larger with the number of points. Despite being very fast to solve, *onion-1-3* initially performs surprisingly well. However, unlike the 12-cone graphs, *onion-1-3* sparsified instances show a clear and steep increase in the objective value gap. *onion-0* starts slightly worse that *onion-1-3*, but its gap increases slightly slower. At 20 points, both have a gap of around 60 %. Overall, none of the sparse graphs resulted in practically interesting objective value gaps. We conclude that none of our approaches seem to capture the relevant edges for area minimization well; in particular, optimal solutions tend to include long edges, which none of our sparsification approaches focus on. Furthermore, we note that for area minimization, the difficulty of finding provably optimal solutions largely stems from finding good lower bounds. Thus, we are rather pessimistic about the speed-up that can be achieved by providing the optimization process with a good initial solution early on.

All this reflects the known difficulties of area-minimal polygonization, which is extremely hard to solve in practice. A possible explanation is the combination of a large number of potential neighbors (because a polygon with small area may contain very long edges, so there are many potential neighbors for each point), with significant gaps between similar solutions (because even a small modification of a polygon with small objective value may result in significantly increased area). This greatly differs from problems such as the TSP, for which solutions are both better localized and gaps between similar solutions are relatively small.

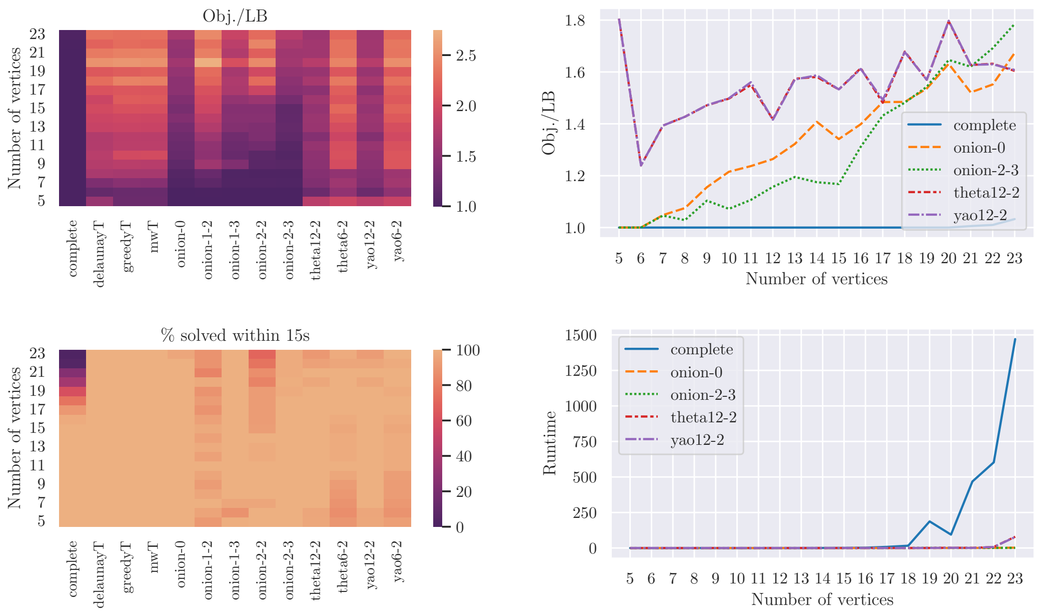


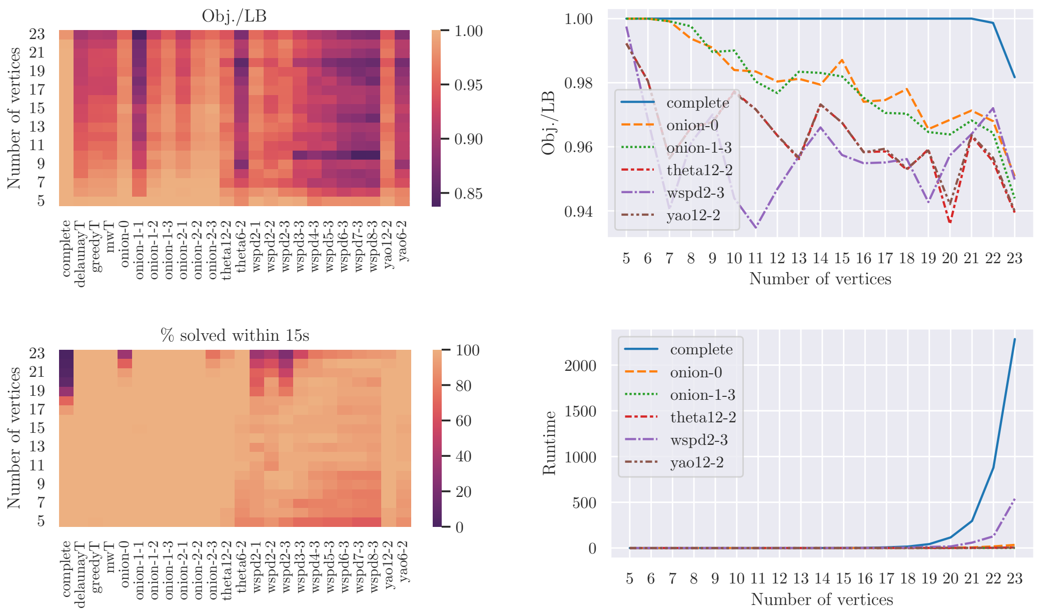
Figure 13 Solution quality (top) and runtime (bottom) for various sparsification methods for area *minimization*. **(Top left)** The upper heatmap shows that all methods have a visible optimality gap, some above twice the optimal value (yellowish color). *onion-0*, *onion-1-3*, *onion-2-3*, *theta-12-2*, and *yao-12-2* (nearly identical to *theta-12-2*) yield the best results, as indicated by the darker colors. **(Top right)** The line plot provides a more detailed view of the optimality gap over instance size for these three methods. *onion-0* and *onion-1-3* produce good solutions for small instances, but their optimality gaps increase until it has the same as *theta-12-2* and *yao-12-2* (around 60 %). **(Bottom left)** The lower heatmap shows how many instances could be solved within 15 s. While optimizing on the complete graph quickly already needs more than 15 s for just 20 points (dark color), nearly all sparsified instances can be solved within this time, as indicated by the uniformly bright color. **(Bottom right)** The line plot shows the mean runtime over instance size for the three methods with the best solution quality. The *yao-12-2* has a slight increase at the end, but all methods are significantly faster than considering the complete graph.

4.6.2 Area maximization

Area maximization turns out to be more tractable. Figure 14 summarizes the results of our experiments: Sparsification leads to a significant improvement in runtime. We find that our methods perform better than for area minimization; however, not just the absolute, but also the relative gap between the optimal solution on the complete graph and on sparsified graphs still grows with n, albeit more slowly. Similarly to the situation for area minimization, we find that *onion-0* performs well, even compared to sparsifications with much greater solution times; however, its objective value gap also quickly increases with instance size. The fast solution time of the sparsified instances may still make them useful for finding initial solutions for area maximization.

5 Conclusions

We have studied a considerable number of sparsification techniques for a collection of geometric tour problems, and provided a spectrum of practical outcomes. By and large, the practical performance of the corresponding practical techniques corresponds to the practical



■ **Figure 14** Solution quality (top) and runtime (bottom) for various sparsification methods for area *maximization*. **(Top left)** The upper heatmap shows by a bright color a good solution quality for *onion*-variants, *theta-12-2*, *wspd-2-3*, and *yao-12-2*. Note that here the columns with darker colors indicate a worse solution quality, as the objective value is maximized. **(Top right)** The line plot provides a more detailed view of the optimality gap over instance size for these four methods. *onion-0* has the best solution quality of less than 3 % below the optimum, but its optimality gap increases with the instance size. **(Bottom left)** The lower heatmap shows how many instances could be solved within 15 s. *onion-0* and *wspd-2-3* are hitting the time limit for some instances with more than 20 points (dark color), but visibly less than the complete graph. **(Bottom right)** The line plot shows the mean runtime over instance size for the four methods with the best solution quality. Here we see that all methods are indeed significantly faster than the complete graph, only *wspd-2-3* shows a visible increase in runtime.

difficulty of solving benchmark instances of the considered optimization problems to provable optimality: For the problems that were known to be somewhat accessible to exact methods (such as the Minimum Perimeter Polygon Problem), we were able to tighten previously known methods; for problems of notorious difficulty (such as Minimum Area Polygonization), the resulting gaps were larger, but still useful due to the lack of other good bounds. An overview is given in Table 1.

Overall, this shows that sparsification techniques are definitely useful tools for extending the practical spectrum of solution time versus quality tradeoff for hard geometric optimization problems.

Acknowledgment

We thank reviewers and editor for a thorough evaluation of a preliminary version of this paper; this resulted in a considerable number of constructive suggestions that helped to improve the presentation. This type of professional community service is greatly appreciated!

Table 1 Summary of the most promising sparsification techniques for each problem. For all problems, the best sparsification technique is significantly faster than the complete instance. The most notable effect is the drastic speedup for the Minimum Perimeter Problem. For the Minimum Area Polygonization, the sparsified graphs are significantly faster, but the solution quality is low.

Problem	Best sparsification	Optimality gap	Speed-up
Angular TSP	*wspd-2-1*	4 % (stable)	significantly faster
Angular-Distance TSP	*wspd-2-1*	0.5 % to 1.0 % (stable)	significantly faster
Minimum Perimeter	*greedyT*, *MWT*	0.3 % to 1.2 % (incr.)	drastically faster
Min Area Polygonization	*onion-0*	60 % (incr.)	significantly faster
Max Area Polygonization	*onion-0*	2 % to 3 % (incr.)	significantly faster

References

1 Alok Aggarwal, Don Coppersmith, Sanjeev Khanna, Rajeev Motwani, and Baruch Schieber. The angular-metric Traveling Salesman Problem. *SIAM Journal on Computing*, 29(3):697–711, 2000. doi:10.1137/S0097539796312721.

2 D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton Series in Applied Mathematics. Princeton University Press, 2011.

3 Esther M Arkin, Michael A Bender, Erik D Demaine, Sándor P Fekete, Joseph SB Mitchell, and Saurabh Sethia. Optimal covering tours with turn costs. *SIAM Journal on Computing*, 35(3):531–566, 2005.

4 Ronald Beirouti and Jack Snoeyink. Implementations of the lmt heuristic for minimum weight triangulation. In *Symposium on Computational Geometry (SoCG)*, pages 96–105, 1998.

5 B. Bixby and G. Reinelt. TSPLIB, a traveling salesman problem library. *ORSA Journal on Computing*, 3:376–384, 1991. doi:10.1287/ijoc.3.4.376.

6 Nicolas Bonichon, Cyril Gavoille, Nicolas Hanusse, and Ljubomir Perković. The stretch factor of L_1- and L_∞-Delaunay triangulations. In *European Symposium on Algorithms (ESA)*, pages 205–216, 2012.

7 Prosenjit Bose, Aaron Lee, and Michiel Smid. On generalized diamond spanners. In *Workshop on Algorithms and Data Structures (WADS)*, pages 325–336, 2007. doi:10.1007/978-3-540-73951-7_29.

8 Prosenjit Bose and Michiel Smid. On plane geometric spanners: A survey and open problems. *Computational Geometry*, 46(7):818–830, 2013. doi:10.1016/j.comgeo.2013.04.002.

9 Paul B. Callahan. *Dealing with higher dimensions: the well-separated pair decomposition and its applications*. Ph.D. thesis, Johns Hopkins University, 1995.

10 Paul B. Callahan and S. Rao Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM (JACM)*, 42(1):67–90, 1995. doi:10.1145/200836.200853.

11 Maw Shang Chang, Nen-Fu Huang, and Chuan-Yi Tang. An optimal algorithm for constructing oriented voronoi diagrams and geographic neighborhood graphs. *Information Processing Letters*, 35(5):255–260, 1990. doi:10.1016/0020-0190(90)90054-2.

12 Bernard Chazelle. On the convex layers of a planar set. *IEEE Transactions on Information Theory*, 31(4):509–517, 1985. doi:10.1109/TIT.1985.1057060.

13 L. Paul Chew. There are planar graphs almost as good as the complete graph. *Journal of Computer and System Sciences*, 39(2):205–219, 1989. doi:10.1145/10515.10534.

14 Ken Clarkson. Approximation algorithms for shortest path motion planning. In *Symposium on Theory of Computing (STOC)*, pages 56–65, 1987. doi:10.1145/28395.28402.

15 Loïc Crombez, Guilherme D. da Fonseca, and Yan Gerard. Greedy and local search heuristics to build area-optimal polygons. *Journal of Experimental Algorithmics*, 27:2.2.1–2.2.11, 2022.

16 Erik D. Demaine, Sándor P. Fekete, Dominik Krupke, Phillip Keldenich, and Joseph S.B. Mitchell. Area-optimal simple polygonalizations: The CG Challenge 2019. *Journal of Experimental Algorithmics*, 27:2.4:1–2.4:12, 2022.

17 Adrian Dumitrescu and Anirban Ghosh. Lower bounds on the dilation of plane spanners. *International Journal of Computational Geometry & Applications*, 26(02):89–110, 2016. `doi:10.1142/S0218195916500059`.

18 Günther Eder, Martin Held, Steinthor Jasonarson, Philipp Mayer, and Peter Palfrader. 2-opt moves and flips for area-optimal polygonalizations. *Journal of Experimental Algorithmics*, 27:2.7:1–2.7:12, 2022.

19 Mohammad Farshi and Joachim Gudmundsson. Experimental study of geometric t-spanners. *Journal of Experimental Algorithmics*, 14:3:1.3–3:1.39, 2010. URL: `http://doi.acm.org/10.1145/1498698.1564499`, `doi:10.1145/1498698.1564499`.

20 Sándor P. Fekete. *Geometry and the Travelling Salesman Problem.* Ph.D. thesis, Department of Combinatorics and Optimization, University of Waterloo, Waterloo, ON, 1992.

21 Sándor P. Fekete. On simple polygonalizations with optimal area. *Discrete & Computational Geometry*, 23(1):73–110, 2000. `doi:10.1007/PL00009492`.

22 Sándor P. Fekete, Stephan Friedrichs, Michael Hemmer, Melanie Papenberg, Arne Schmidt, and Julian Troegel. Area- and Boundary-Optimal Polygonalization of Planar Point Sets. In *European Workshop on Computational Geometry (EuroCG)*, 2015.

23 Sándor P. Fekete, Andreas Haas, Michael Hemmer, Michael Hoffmann, Irina Kostitsyna, Dominik Krupke, Florian Maurer, Joseph S.B. Mitchell, Arne Schmidt, and Christiane Schmidt. Computing nonsimple polygons of minimum perimeter. In *International Symposium on Experimental Algorithms (SEA)*, pages 134–149, 2016. `doi:10.20382/jocg.v8i1a13`.

24 Sándor P. Fekete, Andreas Haas, Phillip Keldenich, Michael Perk, and Arne Schmidt. Computing Area-Optimal Simple Polygonalizations. In *European Workshop on Computational Geometry (EuroCG)*, 2020.

25 Sándor P. Fekete, Andreas Haas, Phillip Keldenich, Michael Perk, and Arne Schmidt. Computing area-optimal simple polygonalization. *Journal of Experimental Algorithmics*, 27:2.6:1–2.6:23, 2022.

26 Sándor P. Fekete, Andreas Haas, Yannic Lieder, Eike Niehs, Michael Perk, Victoria Sack, and Christian Scheffer. On Hard Instances of the Minimum-Weight Triangulation Problem. In *36th European Workshop on Computational Geometry (EuroCG 2020)*, 2020.

27 Sándor P. Fekete, Linda Kleist, and Dominik Krupke. Minimum scan cover with angular transition costs. *SIAM Journal on Discrete Mathematics*, 35(2):1337–1355, 2021. `doi:10.1137/20M1368161`.

28 Sándor P. Fekete and William R. Pulleyblank. Area optimization of simple polygons. In *Symposium on Computational Geometry (SoCG)*, pages 173–182, 1993. `doi:10.1145/160985.161016`.

29 Sándor P. Fekete and Gerhard J. Woeginger. Angle-restricted tours in the plane. *Computational Geometry*, 8(4):195–218, 1997. `doi:10.1016/S0925-7721(96)00012-0`.

30 Anja Fischer, Frank Fischer, Gerold Jäger, Jens Keilwagen, Paul Molitor, and Ivo Grosse. Exact algorithms and heuristics for the quadratic traveling salesman problem with an application in bioinformatics. *Discrete Applied Mathematics*, 166:97–114, 2014. `doi:10.1016/j.dam.2013.09.011`.

31 B.E. Flinchbaugh and L.K. Jones. Strong connectivity in directional nearest-neighbor graphs. *SIAM Journal on Algebraic Discrete Methods*, 2(4):461–463, 1981. `doi:10.1137/0602049`.

32 Sally A. Goldman. A space efficient greedy triangulation algorithm. *Information Processing Letters*, 31(4):191–196, 1989. `doi:10.1016/0020-0190(89)90122-1`.

33 Nir Goren, Efi Fogel, and Dan Halperin. Area optimal polygonization using simulated annealing. *Journal of Experimental Algorithmics*, 27:2.3:1–2.3:17, 2022.

34 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2021. Accessed: 2021-10-15. URL: `https://www.gurobi.com`.

35 Andreas Haas. Solving large-scale minimum-weight triangulation instances to provable optimality. In *Symposium on Computational Geometry (SoCG)*, pages 44:1–44:14, 2018. doi:10.4230/LIPIcs.SoCG.2018.44.

36 IBM Corporation. IBM ILOG CPLEX 12.9 User Manual, 2021. Accessed: 2021-10-15. URL: https://www.ibm.com/docs/en/icos/12.9.0?topic=cplex-users-manual.

37 Gerold Jäger and Paul Molitor. Algorithms and experimental study for the traveling salesman problem of second order. In *International Conference on Combinatorial Optimization and Applications (ICCOA)*, pages 211–224, 2008. doi:10.1007/978-3-540-85097-7_20.

38 J. Mark Keil. Approximating the complete euclidean graph. In *Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 208–213, 1988. doi:10.1007/3-540-19487-8_23.

39 Julien Lepagnot, Laurent Moalic, and Dominique Schmitt. Optimal area polygonization by triangulation and visibility search. *Journal of Experimental Algorithmics*, 27:2.5:1–2.5:23, 2023.

40 Adam N. Letchford and Nicholas A. Pearson. Good triangulations yield good tours. *Computers & Operations Research*, 35(2):638–647, 2008.

41 Christos Levcopoulos and Drago Krznaric. The greedy triangulation can be computed from the Delaunay triangulation in linear time. *Computational Geometry*, 14(4):197–220, 1999. doi:10.1016/S0925-7721(99)00037-1.

42 László Lovász, Katalin Vesztergombi, Uli Wagner, and Emo Welzl. Convex quadrilaterals and k-sets. *Contemporary Mathematics*, 342:139–148, 2004.

43 André César Medeiros and Sebastián Urrutia. Discrete optimization methods to determine trajectories for Dubins' vehicles. *Electronic Notes in Discrete Mathematics*, 36:17–24, 2010.

44 Wolfgang Mulzer. Minimum dilation triangulations for the regular n-gon. Master's thesis, Freie Universität Berlin, October 2004.

45 Wolfgang Mulzer and Günter Rote. Minimum-weight triangulation is NP-hard. *Journal of the ACM*, 55(2):1–29, 2008. doi:10.1145/1346330.1346336.

46 Giri Narasimhan and Michiel Smid. *Geometric Spanner Networks*. Cambridge University Press, Cambridge, 2007. doi:10.1017/CBO9780511546884.

47 Jiju Peethambaran, Amal Dev Parakkat, and Ramanathan Muthuganapathy. A randomized approach to volume constrained polyhedronization problem. *Journal of Computing and Information Science in Engineering*, 15(1):011009, 2015. doi:10.1115/1.4029559.

48 Jiju Peethambaran, Amal Dev Parakkat, and Ramanathan Muthuganapathy. An empirical study on randomized optimal area polygonization of planar point sets. *Journal of Experimental Algorithmics (JEA)*, 21:1–24, 2016. doi:10.1145/2896849.

49 Natanael Ramos, Rai Caetan de Jesus, Pedro de Rezende, Cid de Souza, and Fabio Luiz Usberti. Triangle-based heuristics for area optimal polygonizations. *Journal of Experimental Algorithmics*, 27:2.1:1–2.1:25, 2022.

50 Ketan Savla, Emilio Frazzoli, and Francesco Bullo. Traveling salesperson problems for the Dubins vehicle. *IEEE Transactions on Automatic Control*, 53(6):1378–1391, 2008.

51 Michiel Smid. The well-separated pair decomposition and its applications. In *Handbook of Approximation Algorithms and Metaheuristics*, pages 71–84. Chapman and Hall/CRC, 2018.

52 Rostislav Staněk, Peter Greistorfer, Klaus Ladner, and Ulrich Pferschy. Geometric and LP-based heuristics for angular travelling salesman problems in the plane. *Computers & Operations Research*, 108:97–111, 2019. doi:10.1016/j.cor.2019.01.016.

53 María Teresa Taranilla, Edilma Olinda Gagliardi, and Gregorio Hernández Peñalver. Approaching minimum area polygonization. In *Congreso Argentino de Ciencias de la Computación*, pages 161–170. Facultad de Informática (UPM), 2011. URL: http://hdl.handle.net/10915/18574.

54 The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.3 edition, 2021. URL: https://doc.cgal.org/5.3/Manual/packages.html.

55 Ge Xia. Improved upper bound on the stretch factor of Delaunay triangulations. In *Symposium on Computational Geometry (SoCG)*, pages 264–273, 2011. doi:10.1145/1998196.1998235.

56 Andrew Chi-Chih Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, 1982. `doi:10.1137/0211059`.