COMPUTING MAXMIN EDGE LENGTH TRIANGULATIONS[*]

Sándor P. Fekete,[†] *Winfried Hellmann,*[†] *Michael Hemmer,*[†] *Arne Schmidt,*[†]
and Julian Troegel [†]

Abstract. In 1991, Edelsbrunner and Tan gave an $O(n^2)$ algorithm for finding the MinMax Edge Length Triangulation of a set of points in the plane, but stated the complexity of finding a MaxMin Edge Length Triangulation (MELT) as a natural open problem. We resolve this long-standing problem by showing that computing a MELT is NP-complete. Moreover, we prove that (unless P=NP), there is no polynomial-time approximation algorithm that can approximate MELT within any polynomial factor.

While this may be taken as conclusive evidence from a *theoretical* point of view that the problem is hopelessly intractable, it still makes sense to consider powerful optimization methods, such as integer programming (IP), in order to obtain provably optimal solutions for intances of non-trivial size. A straightforward IP based on pairwise disjointness of the $\Theta(n^2)$ segments between the n points has $\Theta(n^4)$ constraints, making this IP hopelessly intractable from a *practical* point of view, even for relatively small n. The main algorithm engineering twist of this paper is to demonstrate how the combination of geometric insights with refined methods of combinatorial optimization can still help to put together an exact method capable of computing optimal MELT solutions for planar point sets up to $n = 200$. Our key idea is to exploit specific geometric properties in combination with more compact IP formulations, such that we are able to drastically reduce the number of constraints. On the practical side, we combine two of the most powerful software packages for the individual components: CGAL for carrying out the geometric computations, and CPLEX for solving the IPs. In addition, we discuss specific analytic aspects of the speedup for random point sets.

1 Introduction

Triangulating a set of points is one of the basic problems of Computational Geometry: given a set P of n points in the plane, connect them by a maximal set of non-crossing line segments. This implies that all bounded faces of the resulting planar arrangement are triangles, while the exterior face is the complement of the convex hull of P.

Triangulations are computed and used in a large variety of contexts, e.g., in mesh generation, but also as a stepping stone for other tasks. While it is not hard to compute some triangulation, most of these tasks require triangulations with special properties that

[*]A preliminary extended abstract appears in the Proccedings of ALENEX 2015 [16].

[†]Department of Computer Science, TU Braunschweig, 38106 Braunschweig, Germany. {s.fekete, w.hellmann, m.hemmer, arne.schmidt, j.troegel}@tu-bs.de.

should be optimized. Examples include maximizing the minimum angle, and minimizing the total edge weight or the longest edge length.

In this paper we consider the task of computing a triangulation whose shortest edge is as long as possible. We show that this problem is NP-complete, resolving an open problem stated by Edelsbrunner and Tan in 1991 [13]. As a corollary, it follows that (unless P=NP) there cannot be any polynomial-time approximation that gets with a polynomial factor of the optimum value. On the positive side, we develop, refine, and apply a framework that is able to compute provably optimal solutions for point sets of size up to $n = 200$ and beyond.

Related Work. For a broad survey of triangulations in a variety of settings, see the book [11] by De Loeara, Rambau, and Santos. Maximizing the minimum *angle* in a triangulation is achieved by the Delaunay triangulation [12]; making use of Fortune's sweepline algorithm [19], it can be computed in $O(n \log n)$ time. Minimizing the maximum *edge* can be computed in quadratic time, as shown by Edelsbrunner and Tan [13, 14]. One of the most notorious problems regarding triangulations was finally resolved by Mulzer and Rote [23], who proved that finding a triangulation of minimum total edge length (a *minimum-weight triangulation*) is an NP-hard problem. As shown by Remy and Steger [27], there is a QPTAS for this problem. The maximum-weight triangulation problem has also been considered by Qian and Wang [25], who gave a linear-time approximation scheme for the case of a point set in convex position, and by Chin, Qian, and Wang [6], who gave a 4.238-approximation algorithm. Computing a MaxMin Edge Length Triangulation has been considered by Hu [20], who gave a linear-time algorithm for a convex polygon (and thus for a sorted set of points in convex position), and proved that the *graph* version of the problem is NP-hard. Schmidt [29] showed that finding a *geometric* MaxMin Edge Length Triangulation is NP-complete in the presence of obstacles, such as inside of a polygon with holes; she also showed that computing a MaxMin Edge Length Triangulation for a simple polygon can be solved in polynomial time by making use of dynamic programming.

There are a number of other geometric optimization problems for which practical solution methods based on integer programming have been developed. These include a variety of different approaches for computing optimal solutions for different versions of Art Gallery Problems [3–5, 8, 9, 15, 30], an exact approach for solving Chromatic Art Gallery Problems [32], an exact method for solving NP-hard stabbing problems [24], where the objective is to determine a matching, a spanning tree or a triangulation with a small stabbing or crossing number (see [17, 18]), and an approach for computing optimal solutions for wireless localization methods [10], which aims at computing location and angle for a small number of wireless transmitters in a polygonal environment, such that the resulting intersection of communication regions allows distinguishing inside and outside of a building. All papers combine geometric modelling with methods from combinatorial optimization that allow non-trivial reductions to integer programs, which are then further refined and tuned.

Our Results. This paper presents the following results.

- We establish that MELT is NP-complete, even to approximate, resolving the long-standing open problem by Edelsbrunner and Tan.

- We show that a naive IP formulation based on finding a maximum independent set of line segments has $\Theta(n^2)$ variables and $\Theta(n^4)$ constraints, for any set of n points in the plane.
- We develop an alternative IP formulation based on identifying an independent hitting set that yields IPs that are much smaller.
- We give a further reduction on IP size, based on using appropriate linear combinations of valid constraints.
- We discuss a number of approaches for solving the resulting IPs, combining different heuristic bounds and enumeration techniques.
- We show and discuss experimental results. These include optimal solutions for benchmark instances up to $n = 200$ based on random points in a unit square, as well as all real-world geometric instances up to $n = 200$ in the well-known TSPLIB. Also included is an experimental evaluation of different heuristics for upper and lower bounds, as well as various implementation details.
- We discuss further extensions, implications, and analytic questions.

2 Preliminaries

We are given a set P of n points in the plane. We denote by E the set of all $\Theta(n^2)$ line segments (or *edges*) e connecting two points in P; furthermore, X is the set of all pairs $e_i, e_j \in E$ that *cross*, i.e., share a point that is relatively interior to at least one of them. For an edge e, $\ell(e)$ is the Euclidean length of e.

Throughout this paper (in particular in Section 4) we assume without loss of generality that edges are presorted by length, i.e., for $e_i, e_j \in E$ with $i \leq j$, we have $\ell(e_i) \leq \ell(e_j)$. Of particular interest for our problems are sets of edges that are shorter or longer than a given threshold L; with presorted edges, this is described as follows. For an index threshold b, the set $E_{<b}$ is the set of all edges e_i with $i < b$, so $\ell(e_i) \leq \ell(e_b)$ for all $i < b$. Similarly, $E_{\geq b}$ is the set of all edges e_i with $i \geq b$. Finally, $E_{[a,c[}$ is the set of all edges $e_i \in E$ with $i \in [a, c[$.

We are also interested in specific sets of edges that cross. More specifically, an edge e_i *separates* another edge e_j, if e_i and e_j share a point that is interior for both of them. We denote by $E^*_{\geq b}$ the set of all edges $e_i \in E_{\geq b}$ for which there is an edge $e_j \in E_{<b}$ that is separated by e_i. As the presence of e_i in a triangulation Δ implies that e_j cannot be part of Δ, we say that e_i is a *separator* of e_j. For a threshold index b and an edge e with index smaller than b, we denote the set of all separators e_i with $i \geq b$ by $E^*_{\geq b}(e)$. Furthermore, the set of all crossing pairs of edges $e_i, e_j \in E^*_{\geq b}$ is denoted by $X^*_{\geq b}$ and $X^*_{\geq b}(e)$ contains all edges e' with $\{e, e'\} \in X^*_{\geq b}$.

3 NP-Completeness

3.1 An Auxiliary Problem

We start by showing that the following auxiliary problem is NP-complete.

Problem 3.1 Covering by Disjoint Segments *(CDS)*

Given: *A specified set S of line segments ("stabbers") in the Euclidean plane, and a subset T of their intersection points ("targets").*

Wanted: *A non-intersecting subset of the stabbers that covers all targets.*

This problem is somewhat related to one considered by Megiddo and Tamir [22], who showed that it is NP-hard to compute the minimum number of straight lines that are necessary to cover a given set of points in the plane. Note, however, that CDS considers line segments of finite length and that the solution is required to be a disjoint set of segments covering the targets.

Lemma 3.2 *The problem CDS is NP-complete.*

Proof. We give a reduction from Planar 3SAT, which is the subclass of 3-satisfiability problems for which the variable-clause incidence graph is planar—see Figure 1 (Left). To this end, we start with an arbitrary planar 3SAT instance I in conjunctive normal form, and use it to construct a CDS instance $(S, T)_I$ that is solvable iff I can be satisfied; see Figure 1 (Right).

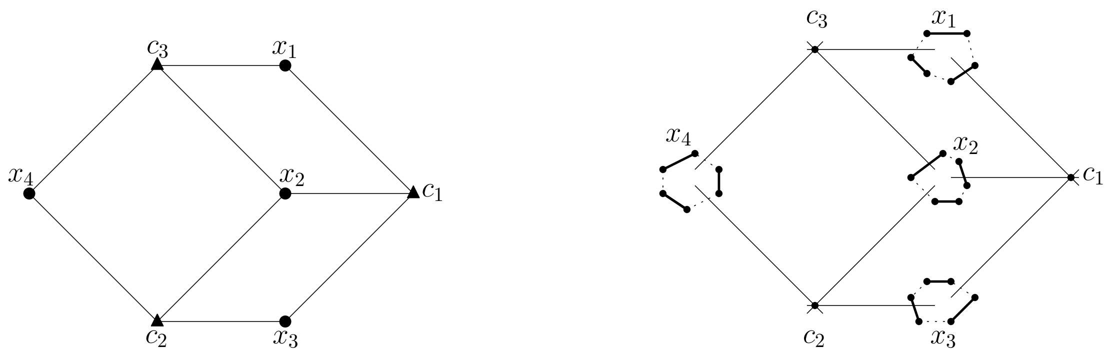


Figure 1: (Left) A planar straight-line drawing of the variable-clause incidence graph G_I of the Planar 3SAT instance $I = (x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee x_3 \vee \overline{x_4}) \wedge (\overline{x_1} \vee x_2 \vee x_4)$. (Right) A CDS instance for the planar 3SAT instance I. Even variable segments are shown in bold, odd variable segments are dotted; clause segments are shown in thin solid. Dots indicate the target points. Labels indicate the original locations of the variable and clause vertices in the (left) drawing of G_I.

As a first step, the variable-clause incidence graph G_I is embedded into the plane, such that all vertices have integer coordinates bounded by $O(n)$ and the resulting edges are represented by line segments; this can be achieved in polynomial time by a variety of graph-drawing algorithms [31]. Let V_{var} be the set of vertices that represent variables, let V_{cla} be the set of vertices that represent clauses, and let E_{cla} be the set of line segments that represent edges in G_I. Scaling the resulting graph layout by a factor of $\Theta(n^2)$ results in an arrangement in which each pair of vertices are a distance of $\Omega(n^2)$ apart.

Next we replace each variable vertex $v_x \in V_{\text{var}}$ by an even cycle of $O(n)$ intersecting "variable" line segments surrounding the original vertex at distance $\Theta(n)$. Each cycle consists of $\texttt{deg}(v)$ "even" and $\texttt{deg}(v)$ "odd" variable segments, where $\texttt{deg}(v)$ is the degree of v; their precise location and parity is chosen such that a segment $\overline{v_x v_c}$ in E_{cla} intersects an odd segment iff variable x occurs in clause c in an unnegated fashion, and a segment $\overline{v_x v_c}$ in E_{cla} intersects an even segment iff variable x occurs in clause c in a negated fashion. Let S_{var} be the resulting set of variable segments. Moreover, let S_{cla} be the set of line segments obtained by shifting all segments in E_{cla} by a distance of $\Theta(n)$ towards its clause endpoint, such that intersection with its corresponding variable segment is maintained. Let $S = S_{\text{var}} \cup S_{\text{cla}}$.

Now let T_{var} be the set of intersection points of variable segments, let T_{cla} be the set of intersection points of clause segments, and $T = T_{\text{var}} \cup T_{\text{cla}}$. We claim: There is a subset $C \subset S$ that covers all points in T iff there is a satisfying truth assignment for I.

For the "if" part, consider the truth assignment of a variable x. If x is set to be true, choose all the odd variable segments for x; if x is set to be false, choose all the even variable segments for x. In either case, all intersection points of the variable segments for x are covered. Because every clause c must have a satisfying literal, picking the segment that connects the clause vertex with the corresponding variable does not intersect one of the selected variable segments.

For the converse "only if" part, consider a set $C \subset S$ of non-crossing segments that covers all points in T. First it is easy to see by induction that if for some x, C contains any even segments, it must contain all even segments; otherwise, it must contain all odd segments. This induces a truth assignment for all variables. Now it is easy to see that a clause vertex can only be covered by a clause segment that does not cross a variable segment, which implies a satisfying truth assignment. □

3.2 Hardness of MaxMin Edge Length Triangulations

Before exploiting the construction of Lemma 3.2 for the main result, we note a helpful lemma; the proof is elementary.

Lemma 3.3 *Let P be a set of points in the plane, and let $p_i, p_j \in P$. A triangulation Δ contains the edge $\overline{p_i p_j}$ iff there is no edge in Δ that separates p_i and p_j.*

Now we proceed to the main theorem.

Theorem 3.1 *Given some positive number L, it is NP-hard to decide whether a set P of n points in the plane has a triangulation with shortest edge of length at least L.*

Proof. We reduce CDS to MELT. Consider the arrangement constructed for the proof of Lemma 3.2. Let Q be the set of all end points of segments in S, and T be the set of target points. Note that the total number of points is in $\Theta(n)$, because the original planar graph has $\Theta(n)$ edges and vertices. We perturb all points in $Q \cup T$ by appropriate powers of $1/2$, thereby modifying their coordinates by amounts between 2^{-cn} and $2^{-(2c)n}$ for a suitable constant $c > 0$, such that the only triples of collinear points correspond to segments in S with their covered points. Note that the resulting point coordinates still have encoding size polynomial in n. For simplicity, we continue to refer to the resulting sets as T and S. As a result, we get a set of points and line segments, such that any triangle formed by three points or a segment and a point not on the segment has smallest height at least some $\delta := 2^{-(2c)n} > 0$.

Furthermore, the segments covering a target point $t \in T$ subdivide its neighborhood into four (for t on a variable cycle) or six (for t at a clause vertex) sectors. Replace each point $t \in T$ by a pair of points t_1, t_2 at a small distance $\varepsilon := 2^{-(3c)n} \ll 2^{-(2c)n} = \delta$ in opposite sectors. See Figure 2. (Note that for the purposes of this proof, simply guaranteeing $\varepsilon \ll \delta$ is sufficient; for the following Corollary 3.4, the choice of ε guarantees an appropriate lower bound on δ/ε.)

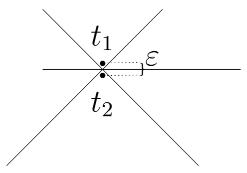


Figure 2: Replacing the target points of the CDS instance by point pairs at distance ε.

We claim: there is a triangulation with shortest edge of length L greater than ε iff the corresponding CDS instance can be solved.

For the "if" part, note that by construction, any segment that covers a target point t must separate the corresponding edge $\overline{t_1 t_2}$. As all points in T are covered, all short edges are separated, and the claim follows from Lemma 3.3.

Conversely, any edge $\overline{t_1 t_2}$ at distance ε corresponding to a target point t must be separated. By construction (and the perturbation argument), an edge e that connects two points p_1 and p_2 can only get within distance δ of t if e covers t in the CDS construction. Thus, by picking all edges that intersect an edge of length ε we get a solution for CDS covering all targets. □

We note an important implication of our construction.

Corollary 3.4 *Let $p(x)$ be some polynomial. Then the existence of a polynomial-time algorithm that yields a $p(n)$-approximation for MELT implies P=NP.*

Proof. In the proof of Theorem 3.1, choose ε as described, implying that $\delta/\varepsilon = 2^{cn} > p(n)$. Assume that there is a solution to CDS covering all targets but the approximation yields a smallest edge in MELT with length ε. An optimal solution requires that the smallest edge has length at least δ. Then the ratio of optimum and approximation is at least δ/ε, which is strictly greater than $p(n)$. Therefore, a $p(n)$-approximation requires finding a triangulation in which all ε-edges are separated. □

4 IP Formulations

4.1 Using Independent Sets

In a valid triangulation, edges must not cross, i.e., they may only meet on endpoints. For a set P of n points in the plane, n_c of which are on the convex hull of P, it is a straightforward consequence of Euler's formula that the number of edges in a triangulation must be precisely $3n-3-n_c$. This leads to the following an IP-formulation for MELT, where b is any threshold index; we only check for feasibility, as the objective value L is captured as $\ell(e_b)$.

Integer Program 4.1

$$\begin{array}{llcll}
 & \max \quad 0 & & & (1)\\
s.t. & x_{e_i} + x_{e_j} & \leq & 1 \qquad \forall \{e_i, e_j\} \in X_{\geq b} & (2)\\
 & \sum_{e_i \in E_{\geq b}} x_{e_i} & = & 3n-3-n_c & (3)\\
 & x_e & \in & \{0,1\} \qquad \forall e \in E_{\geq b} & (4)
\end{array}$$

Now it is straightforward to establish the following.

Theorem 4.2 *We can solve MELT by finding a maximum index b for which IP 4.1 is feasible.*

The number of variables in this IP corresponds to the number of (long) segments, so it is in $\Theta(n^2)$. The number of constraints corresponds to the number of segment pairs that cross, which is the number of convex 4-tuples in P. It is straightforward to see that this number is *sometimes* $\Omega(n^4)$ and never worse than $O(n^4)$; it follows from a deep result of Lovász et al. [21], and earlier by Scheinerman and Wilf [28] that it is *never* smaller than $\Omega(n^4)$, implying $\Theta(n^4)$ constraints. Also note that the *number* of IPs to be considered depends on additional geometric aspects, which are further explored and exploited in the following.

4.2 Using Independent Hitting Sets

A crucial observation for greatly reducing the size of each IP formulation is based on Lemma 3.3, which was already used for establishing the NP-completeness of MELT: A triangulation Δ contains the edge e_j, iff there is no edge e_i in Δ that separates e_j. Therefore, instead of focusing on the long edges and trying to find a non-crossing set of maximum cardinality, we can focus on the short edges and try to find an independent, non-crossing set of long segments that separates all of them. Once such a separating set has been established, completing it into any triangulation (a straightforward postprocessing task) will always yield an optimal MELT solution.

This yields the following type of IP, for any threshold index b, and thus threshold length $L := \ell(e_b)$. (For an illustrative example for the rest of this subsection, the reader is referred to Figures 3 and 4.)

Integer Program 4.3

$$\begin{aligned}
\max \quad & 0 && && \qquad (5)\\
s.t. \quad & x_{e_i} + x_{e_j} && \leq 1 && \forall \{e_i, e_j\} \in X^*_{\geq b} \qquad (6)\\
& \sum_{e_i \in E^*_{\geq b}(e_j)} x_{e_i} && \geq 1 && \forall e_j \in E_{<b} \qquad (7)\\
& x_{e_i} && \in \{0,1\} && \forall e_i \in E^*_{\geq b} \qquad (8)
\end{aligned}$$

From the above argument we can conclude the following.

Theorem 4.4 *We can solve MELT by finding a maximum value b for which IP 4.3 is feasible.*

Proof. Let b^* be the index of the smallest edge in an optimal solution of MELT. Then for each edge e_i with $i < b^*$, there is an edge e_j with $j \geq b^*$ separating e_i. Choosing $x_{e_j} = 1$ gives a valid setting fulfilling all constraints in IP 4.3: None of the chosen edges cross each other (constraint (6)), and for any edge e_i we have at least one edge e_j with $j \geq b^*$ and $x_{e_j} = 1$ (constraint (7)).

For the converse, consider a solution for IP 4.3 for some b. Computing a constrained Delaunay triangulation with all chosen edges yields a triangulation in which none of the edges e_i with $i < b$ is contained. This can be done for any b for which IP 4.3 has a solution. □

In practice, IP 4.3 promises to be considerably smaller than IP 4.1: Typically, the relevant (i.e., small) choices of b make $b - 1 = |E_{<b}|$ relatively small, and the number $\overline{m}_{\geq b} := |E^*_{\geq b}|$ of long edges intersecting these small edges is also small compared to m, i.e., the total number of edges. However, we still have $O(\overline{m}^2_{\geq b})$ constraints of type (6). Using an IP reformulation and $k_e(b) := |X^*_{\geq b}(e)|$ for any $e \in E^*_{\geq b}$, we can condense families of constraints into one, leading to a further reduction to $O(\overline{m}_{\geq b})$ constraints.

Integer Program 4.5

$$
\begin{array}{llcll}
\max & 0 & & & (9)\\
s.t. & k_e(b)\cdot x_e + \sum\limits_{e_i\in X^*_{\geq b}(e)} x_{e_i} & \leq & k_e(b) & \forall e\in E^*_{\geq b} \quad (10)\\
 & \sum\limits_{e_i\in E^*_{\geq b}(e)} x_{e_i} & \geq & 1 & \forall e\in E_{<b} \quad (11)\\
 & x_{e_i} & \in & \{0,1\} & \forall e_i\in E^*_{\geq b} \quad (12)
\end{array}
$$

Theorem 4.6 *We can solve MELT by finding a maximum index b for which IP 4.5 is feasible.*

Proof. We show that IP 4.5 is equivalent to IP 4.3. This only requires proving that any solution to IP 4.5 satisfies all constraints (6), and any solution to IP 4.3 satisfies all constraints (10).

For the first part, assume that some feasible solution of IP 4.5 violates a constraint of type (6), so there are two intersecting edges $e, e' \in X^*_{\geq b}$ with $x_e = x_{e'} = 1$; however, this implies that the left-hand side of constraint (10) for e becomes at least $k_e(b) + 1$, as $e' \in X^*_{\geq b}(e)$, a contradiction.

For the converse, consider a feasible solution of IP 4.3 and observe that a constraint (10) for some edge $e \in E^*_{\geq b}$ can only be violated if $x_e = 1$ as well as $x_{e'} = 1$ for some $e' \in X^*_{\geq b}(e)$; this implies that $\{e, e'\} \in X^*_{\geq b}$, and the corresponding constraint (6) of IP 4.3 is violated, a contradiction.

All previous IPs test whether it is possible to find a triangulation for one specific threshold index. If we have a whole interval $[a, c[$ of indices that contains the critical one (so there is a triangulation for threshold value a, but not for threshold value c), then we can compute this largest feasible index as follows.

Integer Program 4.7

$$
\begin{array}{llcll}
\max & b & & & (13)\\
s.t. & x_{e_i}\cdot i + (1-x_{e_i})\cdot c & \geq & b & \forall i\in [a,c[\quad (14)\\
 & k_e(b)\cdot x_e + \sum\limits_{e_i\in X^*_{\geq a}(e)} x_{e_i} & \leq & k_e(b) & \forall e\in E^*_{\geq a} \quad (15)\\
 & x_e + \sum\limits_{e_i\in E^*_{\geq a}(e)} x_{e_i} & \geq & 1 & \forall e\in E_{[a,c[} \quad (16)\\
 & \sum\limits_{e_i\in E^*_{\geq a}(e)} x_{e_i} & \geq & 1 & \forall e\in E_{<a} \quad (17)\\
 & x_{e_i} & \in & \{0,1\} & \forall e_i\in E^*_{\geq a} \quad (18)
\end{array}
$$

Thus, b is the largest possible index for which there is a feasible triangulation: it marks the first index for which $x_{e_i} = 1$, i.e., for which using e_i cannot be avoided. Constraints (14) enforce that this index is indeed met, as it sets a non-critical bound of c for all unused edges. The additional Constraint (16) ensures that an edge e_i with $i \in [a, c[$ is either picked for the triangulation, or separated by another edge. From this and Theorem 4.6, we conclude the following.

Theorem 4.8 *If we know that the critical index $b \in [a, c[$, we can solve MELT by solving IP 4.7.*

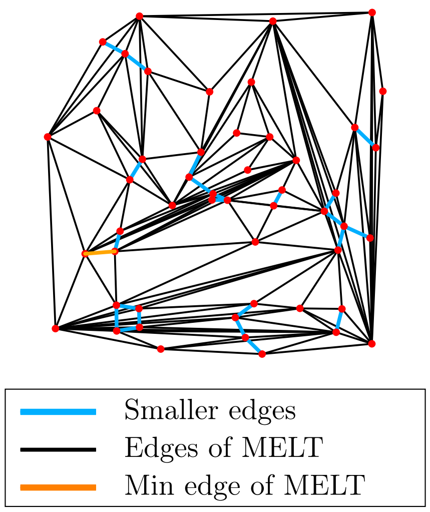


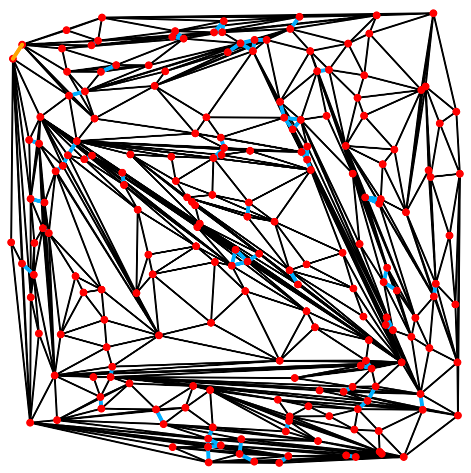
Figure 3: A solution of MELT for an instance with 50 random points in the plane. The smallest edge of the triangulation has index 24. This smallest edge (orange) has possible separating edges, but all these edges cross other separators of even smaller edges.

4.3 Finding the Critical Index

Exploiting the above IPs for computing the optimal MELT value hinges on what set of indices is examined, either because it determines the number of IPs that need to be tested for feasibility (e.g, when using IP 4.3 or IP 4.5), or because it determines the size of the IP that needs to be optimized (i.e., when using IP 4.7). To this end, we give a natural upper bound and an improved one that both decrease the set of indices to be searched. Furthermore, we describe different alternatives for evaluating the candidate indices.

4.3.1 Heuristic Upper Bounds

All edges on the convex hull of P must be part of any triangulation of P, so the length of a shortest edge on the convex hull is an upper bound for the optimal value b^*. However, this *hull bound* b_{CH} tends to be far from the optimal bound; see Figure 17. By Lemma 3.3, *any*



🟦	Smaller edges
⬛	Edges of MELT
🟧	Min edge of MELT

Figure 4: A solution of MELT for an instance with 200 random points. The smallest edge of the triangulation has index 82 and lies on the convex hull, so it matches the hull bound.

non-separated edge must be contained in any triangulation of P, so the refined *non-separated edge bound* $\hat{b}$ is the index of the smallest such edge.

4.3.2 Search Schemes

We have developed and evaluated a number of different search schemes, described in the following. We make use of the non-separated edge bound $\hat{b}$.

Pedestrian Approach. Starting with $b = 1$, we perform feasibility tests, increasing b by 1 while the test is feasible. (The hope being that the first tests are small and fast to carry out.)

Normal Approach. We simply use the optimization IP 4.7 with the index interval $[a, c[=: [1, \hat{b}]$. Then b^* is computed in one step, but on a relatively large IP.

Binary Approach. We perform a binary search on the index interval, starting with $[1, \hat{b}]$. In each binary step, a feasibility test based on IP 4.5 is used.

Doubling Approach. Starting with index $b = 1$, a sequence of feasibility tests with IP 4.5 is carried out. As long as this turns out to be feasible, the index is doubled, but not beyond $\hat{b}$; as soon as a test comes back as infeasible, we have an interval containing b^*. For this interval, b^* is computed with one optimization IP of type 4.7. If we end up with $\hat{b}$ being feasible, $b^* = \hat{b}$.

Combined Approach. We start as in the doubling approach, until an interval with b^* is identified. This is evaluated by a binary search, also based on feasibility tests.

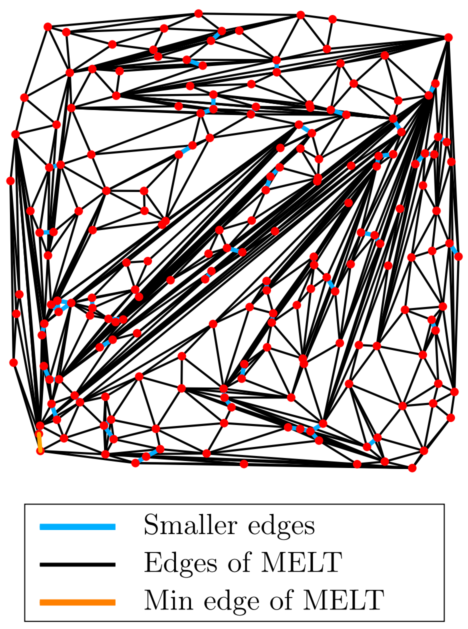


Figure 5: A solution of MELT for an instance with 200 random points. The smallest edge of the triangulation has index 66 and lies close to the convex hull, without a crossing edge, so it matches the non-separated bound.

5 Implementation Framework

In the following, we describe the technical details of our experiments. These can be subdivided into (1) the computation of all geometric objects and their relations and (2) the solution of the integer programs. Component (1) consists of geometric procedures that are all of polynomial complexity: either as pre-processing steps (such as generating the segments of the point set, their intersections) or as post-processing (for generating the final triangulation); on the other hand, component (2) solves the NP-hard class of IP instances. It is essential to use appropriate tools for both parts.

5.1 Geometric Pre- and Postprocessing

All geometric computations were carried out with CGAL, the *Computational Geometry Algorithms Library* [1], a powerful open-source `C++` library for computational geometry. One of its strengths is the ability to generate robust and exact output, even in the presence of sensitive geometric operations, with the option of providing fast qualitative results if no precise coordinates are needed.

5.1.1 Kernel

Once the set of input points is given, there are no actual geometric constructions, because triangles and edges are simply combinatorial objects defined on top of the initial points.

Therefore, it is sufficient to use the `Exact_predicates_inexact_constructions_kernel`. It uses inexact double arithmetic to represent geometric objects (such as points), but guarantees correct decisions for all predicates, e.g., whether two segments intersect or not, which is precisely what we need for our purpose.

5.1.2 Convex Hull

The naive bound is based on computing the shortest edge on the convex hull. Making use of `convex_hull_2`, the time for this step is basically negligible.

5.1.3 Intersections

The most critical part of geometric pre-processing is to compute the set of relevant intersecting edge-pairs. We made use of `do_intersect`, which decides whether two segments intersect. As we only need this Boolean information, using this subroutine is superior to actually computing the precise intersection point. In addition, we also made use of a more sophisticated approach based on Axis-Aligned Bounding Box trees (AABB-trees) [2], which is also available in CGAL.

Definition 5.1 *An* axis-aligned bounding box *is a bounding area. This means, it is a geometric box $B = \{p_{min}, p_{max}\}$, with sides parallel to the axes, which contains a set of geometric objects S. The box B is defined by two points p_{min} and p_{max}, which are the lower left corner and the upper right corner of the box, such that B contains S completely and is minimal in the area.*

This allows us to define bounding boxes of edges and sets of edges.

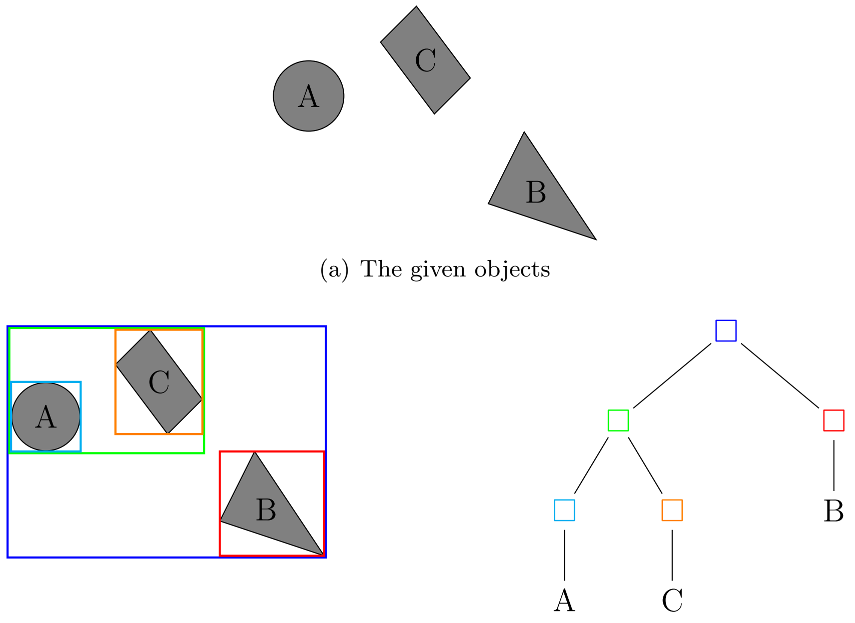
Definition 5.2 *An* axis-aligned bounding box B *over a set of edges E with vertex set V is* $B := \{p_{min}, p_{max}\}$, *with* $p_{min} := \begin{pmatrix} \min_{v \in V} x(p_v) \\ \min_{v \in V} y(p_v) \end{pmatrix}$ *and* $p_{max} := \begin{pmatrix} \max_{v \in V} x(p_v) \\ \max_{v \in V} y(p_v) \end{pmatrix}$.

We arrange these *axis-aligned bounding boxes* to create a data structure that speeds up the task of determining intersecting edges.

Definition 5.3 *An* AABB-tree *is a tree structure that stores a set of geometric objects; these are stored in the leaves. Each inner node is associated with an axis-aligned bounding box that contains all objects in the corresponding subtree. The* AABB-tree of edges *is a hierarchical organization of axis-aligned bounding boxes in which all leaves are edges.*

Figure 6 shows an example of an AABB-tree. To compute all intersecting edges of a given edge q, we start a query comparing q with the data structure. We start at the root of the tree (an axis-aligned bounding box containing all edges of the graph) and test if q intersects with any of the children. We recurse on each such child until we reach the leaves. A final intersection test with all reached leaves gives us a list of edges that intersect with q.

This implies that axis-aligned bounding boxes that do not intersect q are filtered out; moreover, the whole subtree of the node containing this axis-aligned bounding box is not tested for intersections. This reduces the number of intersection tests and thus the time needed to compute all intersections (see Figure 16).



(a) The given objects

(b) The AABBs of the objects

(c) The AABB-Tree

Figure 6: An example for an AABB-tree with 3 objects. Template taken from: [7]

5.1.4 Triangulation

After we solve an instance of MELT, we have the indices of a set of (non-crossing) edges, which separate the smallest segments in the point set. These are completed into a triangulation with the help of `Constrained_Delaunay_triangulation_2`.

5.1.5 Point Generation

For the random instances, points are generated uniformly at random within a unit square with the help of the CGAL operation `Random_points_in_square_2`. The real-world benchmarks are based on the geometric instances of the TSPLIB benchmark library [26].

5.2 Solving the IPs

The second component was implemented based on CPLEX, which is a powerful commercial software tool for solving Linear and Integer Programs. We made use of the Concert-API of CPLEX, which contains all libraries, methods and calls for creating and solving problems in `C++`.

6 Experimental Results

6.1 Setup

Our experiments were carried out on a machine with 64-Bit `Ubuntu 16.04 LTS`, using 32 GB RAM, and a 3.4 GHz `Intel Core i7-4770 CPU`, a processor with four physical and four virtual cores. More than one core was only used for large and complex aspects of solving an IP.

The program itself was compiled with the GNU `GCC 5.4.0` compiler for `C++`. It uses the following options: `-O3 -frounding-math`, used libraries are CGAL 4.8.1 and CPLEX 12.6.2.0. The option `-O3` optimized the whole program when possible and the option `-frounding-math` is needed for CGAL. `Simple-SVG` is available in version 1.0.0, but is not important for finding a MELT-solution.

For each size $n \in \{10, 20, \ldots, 200\}$, we generated 100 instances uniformly at random within a unit square. Every instance was run for at most 30 minutes; in case of a timeout, this was recorded. As a consequence, average times for inferior methods (with a larger number of timeouts) are lower bounds for the actual runtimes, so the superior methods (with virtually no timeouts) fare even better in comparison.

In addition, we considered non-random, real-world point sets from the TSPLIB [26]. We solved all geometric instances up to $n = 200$, along with many larger ones. See Figures 8 and 9 for solutions of two instances. As the two examples show, some of the TSPLIB instances are more uniformly distributed than random point sets (e.g., Figure 8), while others are more clustered (e.g., Figure 9). In addition, these instances may have some degeneracies, such as many edges of the same length. Nevertheless, the results are of similar quality; see Figure 7 for a graphical comparison to the random instances.

6.2 Detailed Comparisons

6.2.1 Comparing IP 4.3 and IP 4.5/4.7

We start by comparing the consequences of using IP 4.3 versus IP 4.5, based on *normal* index search. (For this purpose, we turn IP 4.3 into an optimization version analogous to 4.7.) Figure 10 gives an overview of the time spent by CPLEX, with aborted instances marked by a 30-minute timestamp. Figure 11 shows the overall average time per instance for solving the IPs, while Figure 12 compares the number of instances (out of 100) with timeout. Clearly, IP 4.7 is greatly superior. The fact that the time used by IP 4.5/4.7 is sometimes worse or almost equal to the time of IP 4.3 is due to the number of aborted instances. With IP 4.5/4.7 we can solve a larger number of hard instances, which has a negative effect on the runtime compared to IP 4.3, for which the set of solved instances contains a larger share of easier ones. As a consequence, only IP 4.5/4.7 is used in further experiments. (Note that IP 4.1 is quite terrible, so we are not showing its numerical results.)

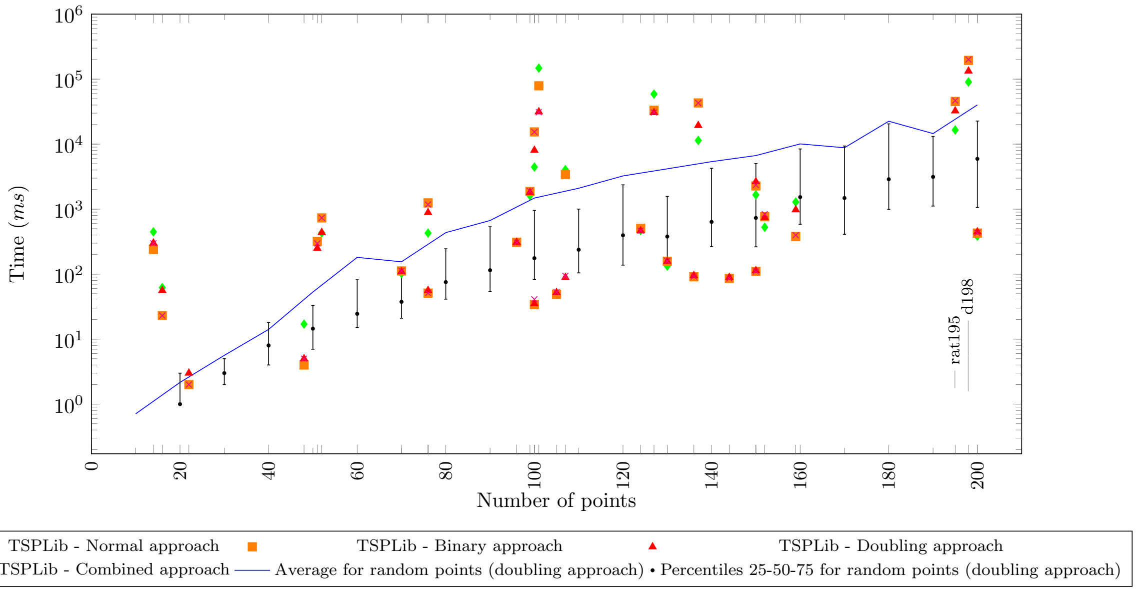


Figure 7: Comparison of TSPLIB instances with average times (including time for parsing, finding intersections, solving IPs, etc.) for random instances using the doubling approach. (Error bars indicate quartiles.)

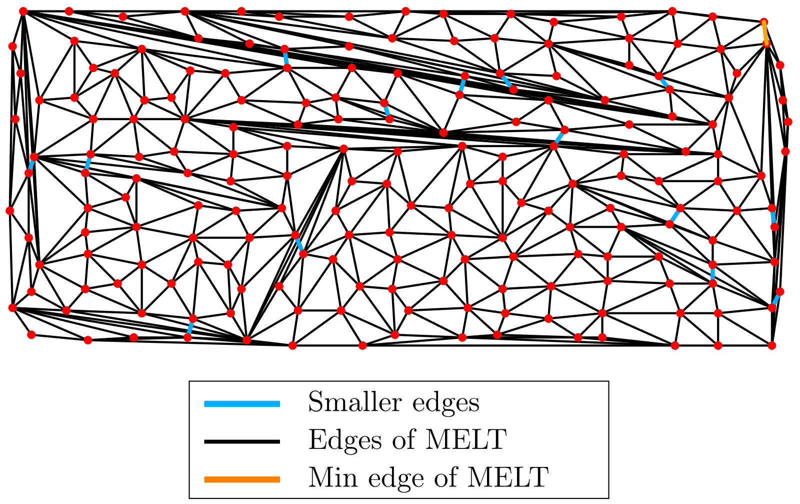


Figure 8: A solution of MELT for the rat195 instance with 195 points of the TSPLIB. The smallest edge of the triangulation has index 15. Also the points of this instance are more evenly distributed.

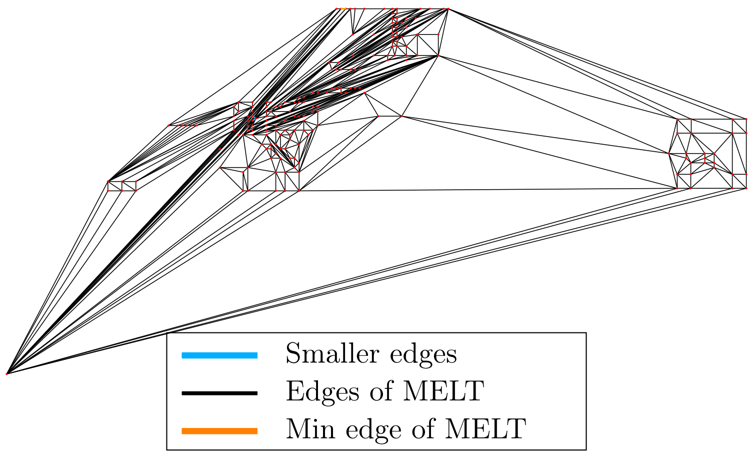


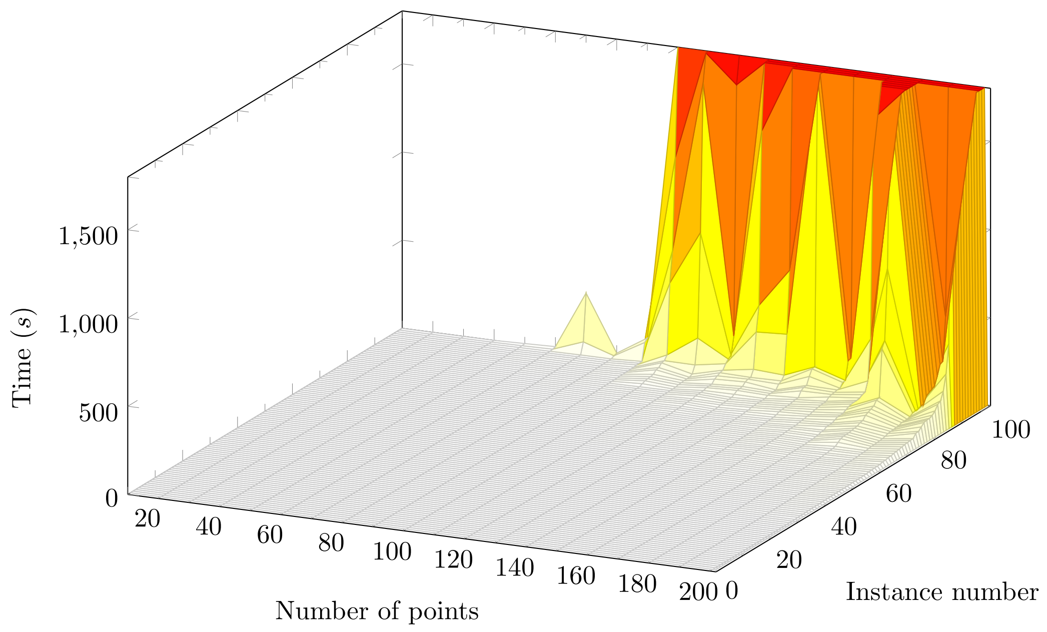
Figure 9: A solution of MELT for the d198 instance with 198 points of the TSPLIB. The smallest edge of the triangulation has index 3 and lies on the convex hull, so it matches the hull bound. Also this instance has many edges of the same length.

6.2.2 Comparing Index Searches

While the better performance of IP 4.7 was to be expected a priori, the best choice for determining the optimal index is not clear at all, as it depends on both the typical critical index, as well as the time spent on an IP feasibility check and on solving an optimization IP. Clearly, a small index range implies both a small number of possible choices (and thus, a small set of IPs that need to be checked), as well as relatively small individual IPs. Somewhat surprisingly, it seems that the doubling approach presents the best compromise between carrying out many relatively cheap IP feasibility tests and few expensive optimization IPs. As can be seen from Figure 13, this advantage is significant, but much less pronounced than between the different IPs. (The simple pedestrian approach was significantly worse than the others, so it is not shown.)

6.2.3 Comparing Program Components

As discussed in the description of our program components, we used different twists for improving both the geometric and the optimization components. How much improvement could be obtained from further tuning, and which components should be targeted? Figure 14 shows how much time on average was spent on each component, "other" summarizes the time for all other geometric pre- and postprocessing operations (such as reading the input, creating the graph, sorting the edges by size, computing the hull bound, triangulating the points, recording the output), and other overhead; in four of the cases shown in Figure 15, this included non-trivial swap times due to RAM limitations. It can be seen that the time spent on optimization is not more than 3-4 times bigger than the time spent on mostly geometric computations. This indicates that for the full range of instance sizes, aiming for a moderate further speedup should target at the IPs, while significant speedup can only be achieved by tuning *both* geometry and optimization.



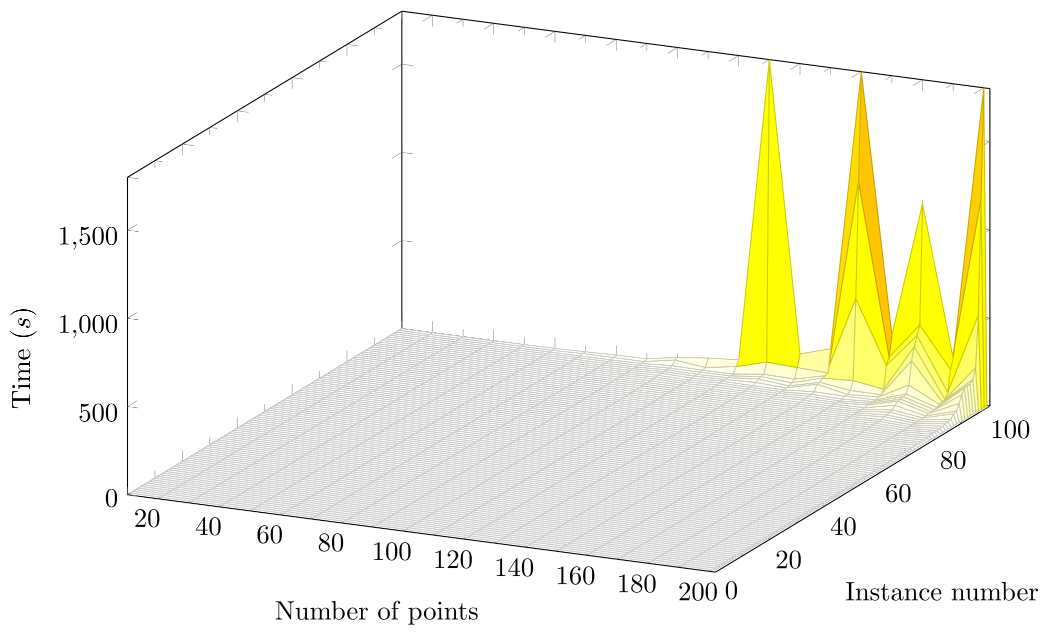


Figure 10: Time of running CPLEX for various instance sizes, using the normal approach for index search. For each size (corresponding to the number of points), 100 instances were evaluated; these are sorted from 1 to 100 by increasing runtime, with a cutoff at 30 minutes. The top figure shows the results for IP 4.3, while the bottom visualizes the results for the improved IP 4.5/4.7; clearly, the latter is significantly better.

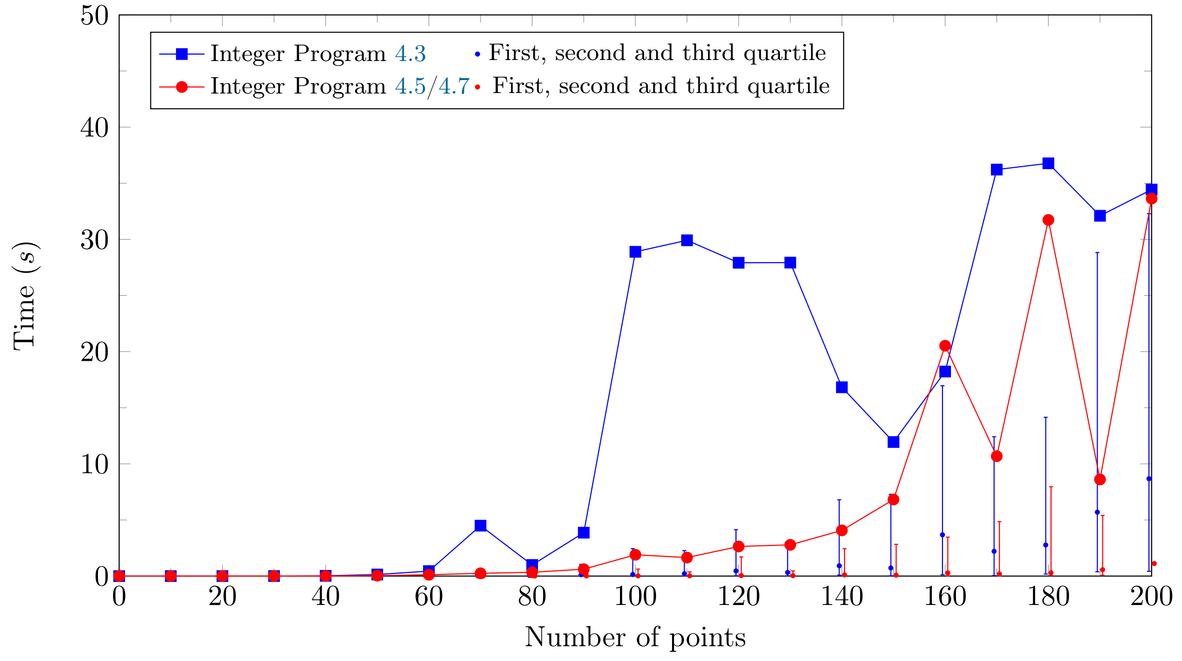


Figure 11: Average time for solved instances spent on running CPLEX with normal approach.

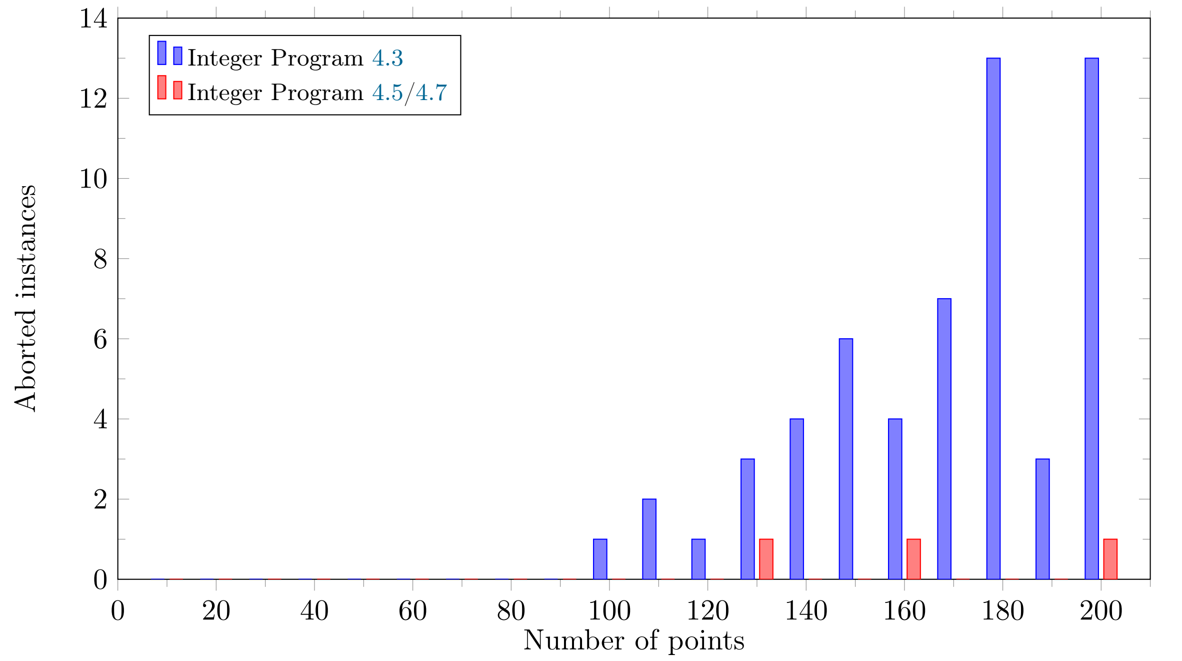


Figure 12: Number of aborted instances for both IPs, out of 100 instances.

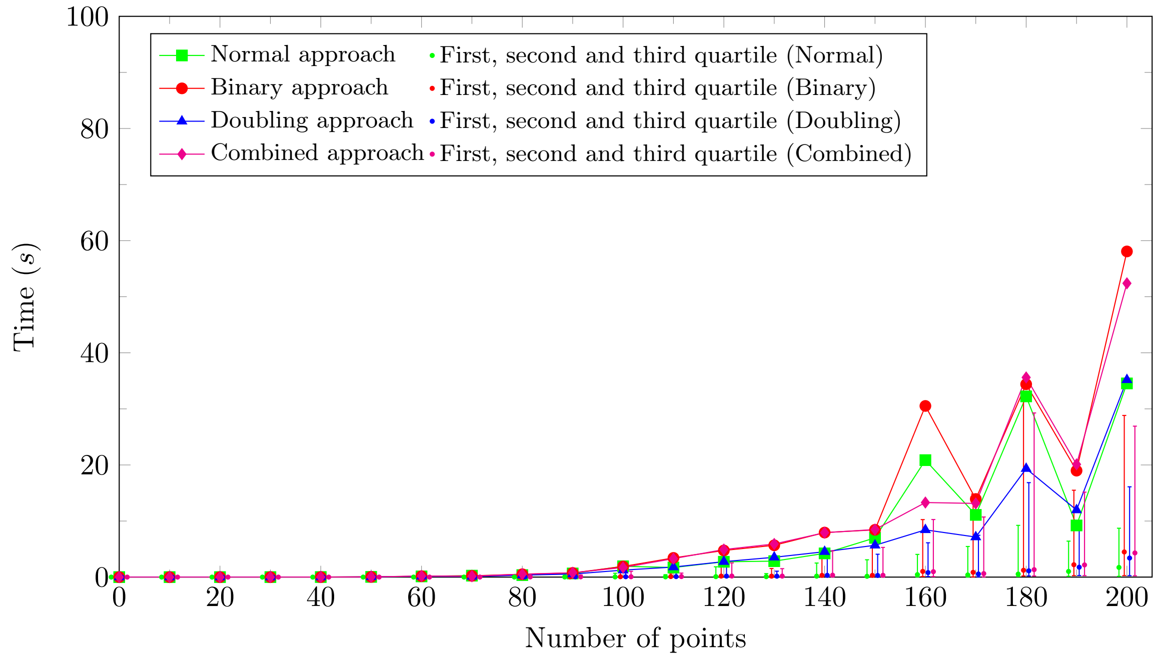


Figure 13: Times spent by CPLEX for the different index searches on random instances.

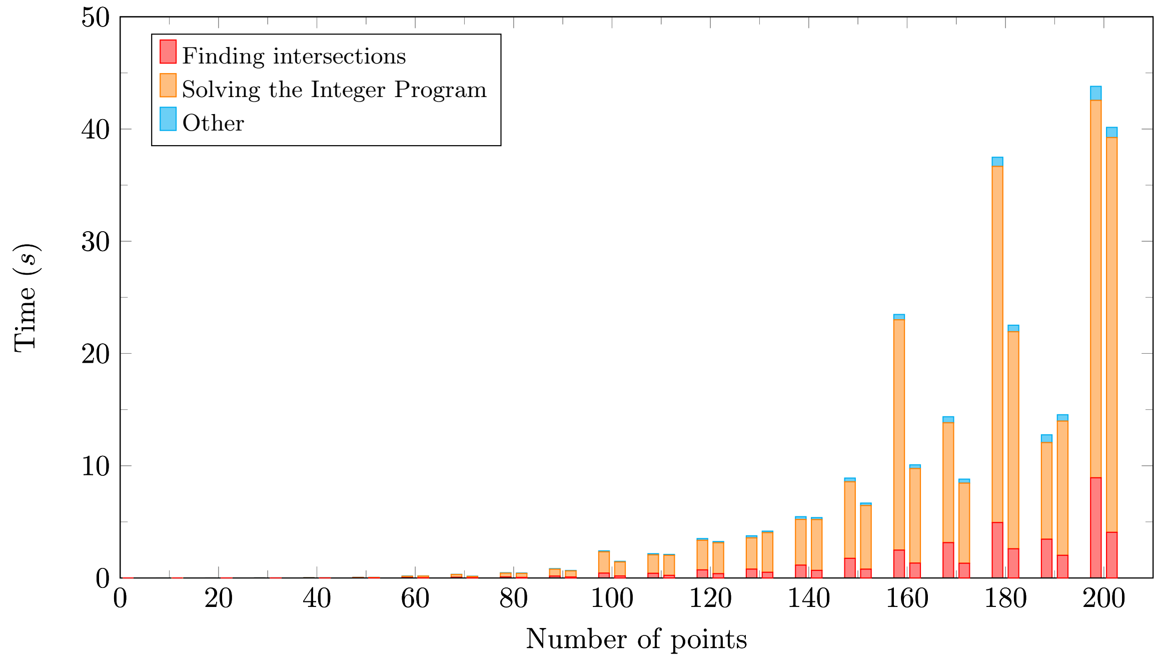


Figure 14: Average time (for random 100 random instances for each size) of the program components.

Left bars: Normal approach, based on IP 4.7

Right bars: Doubling approach, with AABB-Tree and IP 4.7

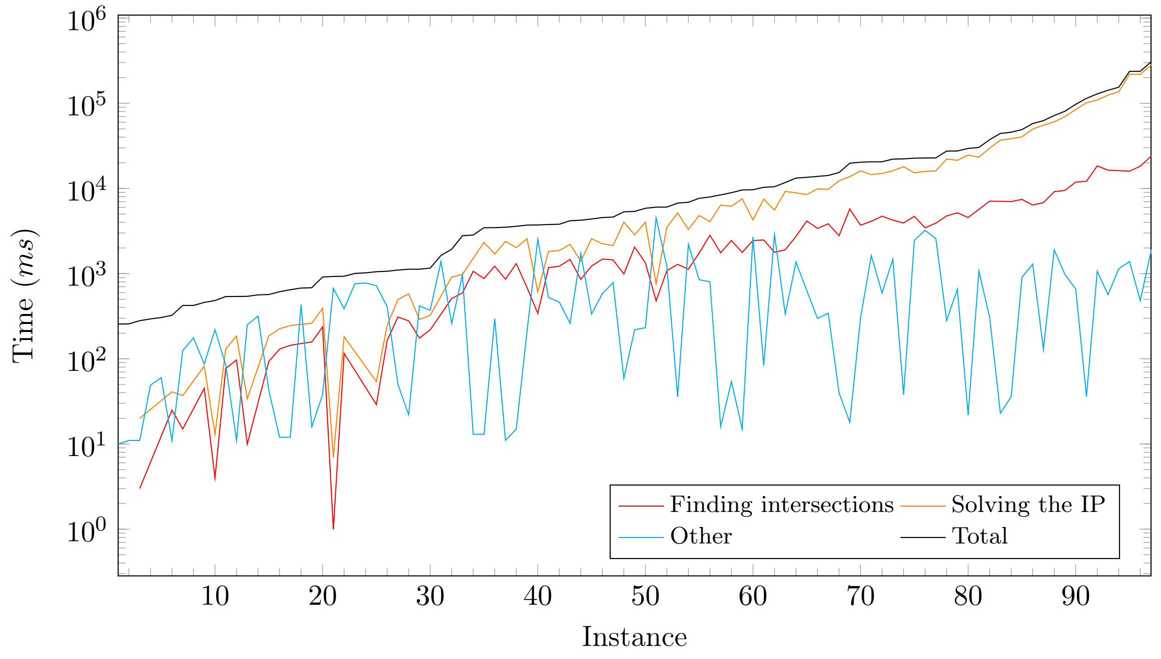


Figure 15: Graph for the time for all 100 solved random instances with 200 points. For a better overview, the y-axis is scaled logarithmically.

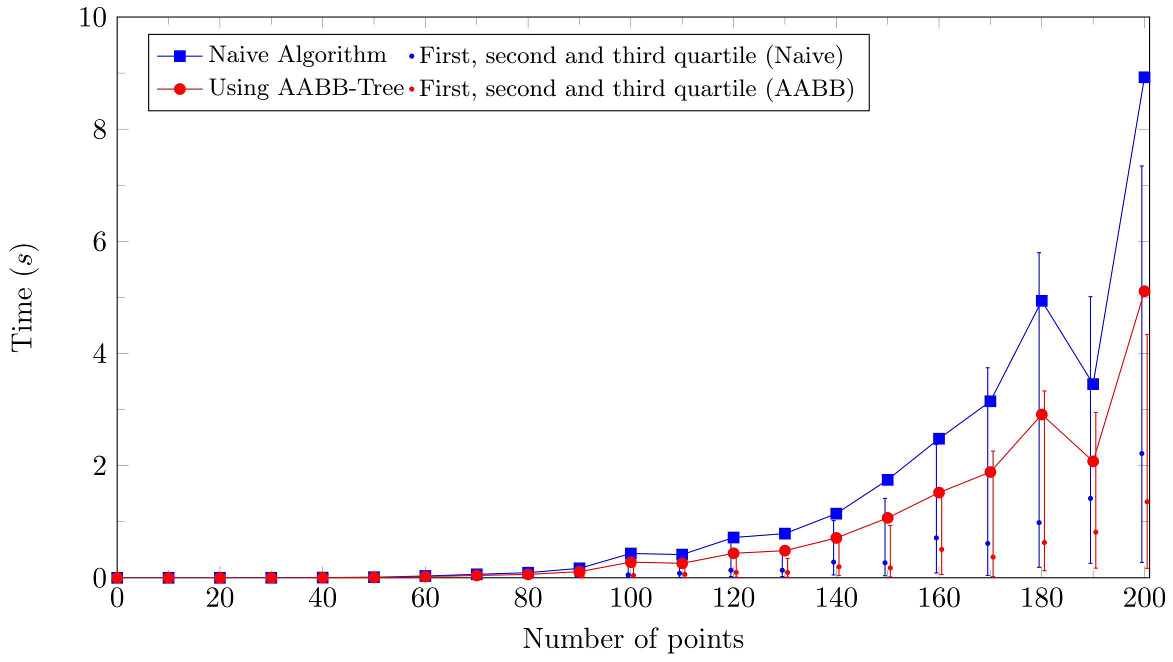


Figure 16: Average time for finding intersections in random instances. Bars indicate the first, second and third quartile.

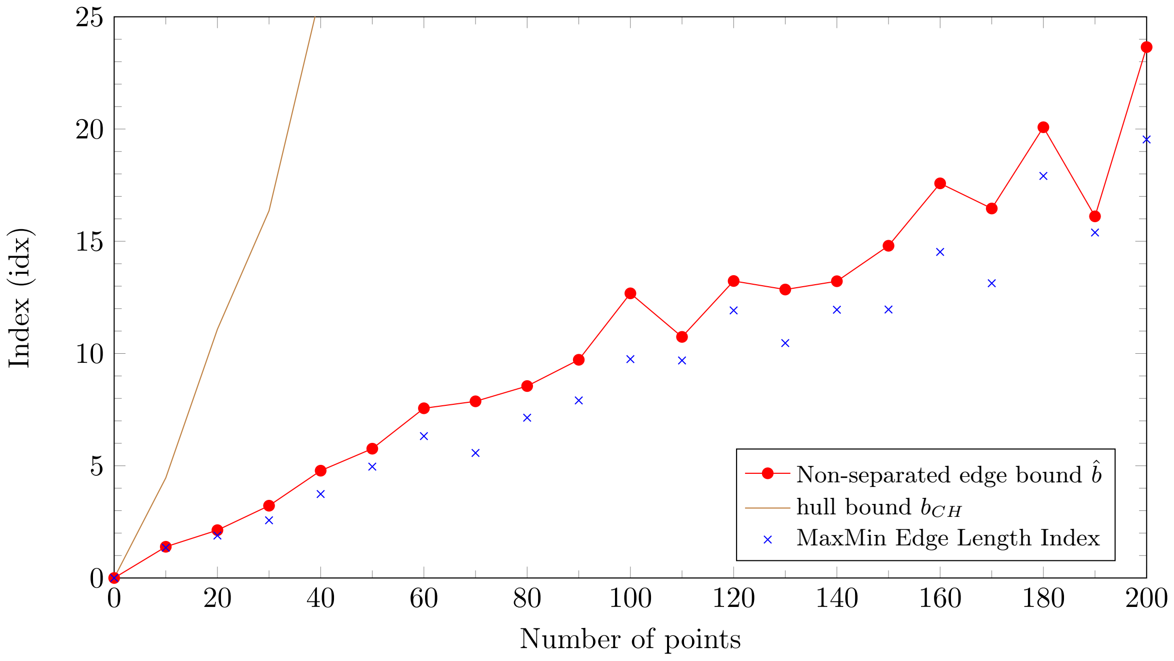


Figure 17: Average values for hull bound, non-separated edge bound, and optimal index for random instances.

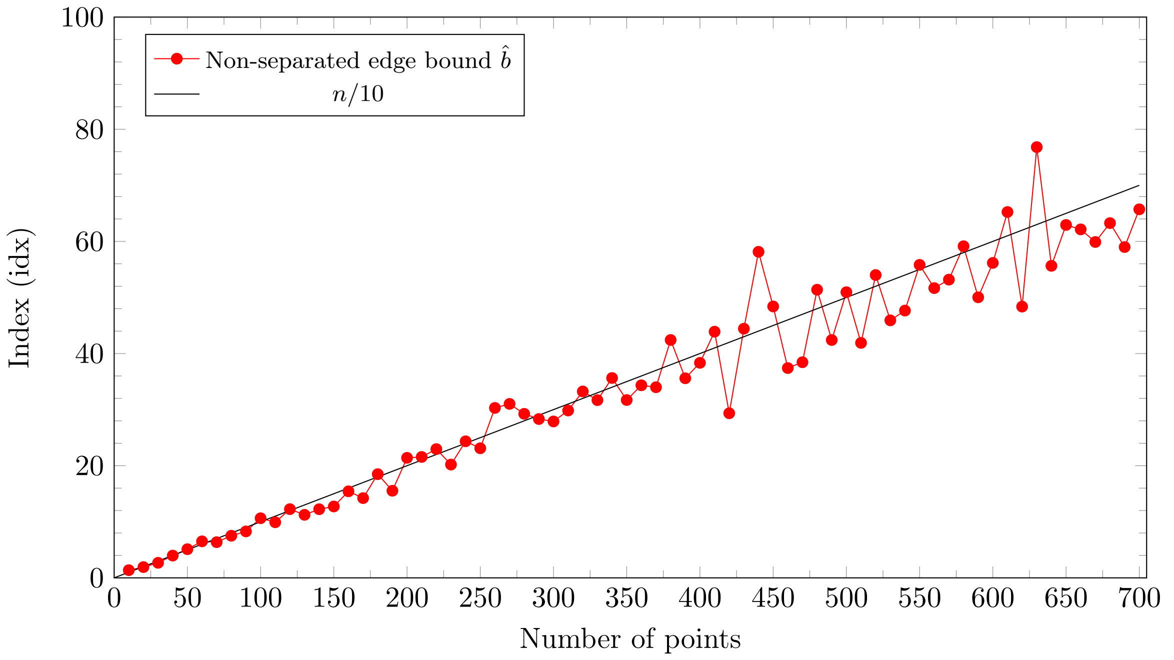


Figure 18: Average indices (for 100 random instances for each size) of the non-separated edge bound $\hat{b}$ for random sets with up to 700 points.

7 Discussion and Conclusions

We have demonstrated how combining geometric insights with a number of optimization and implementation ideas can be used for solving non-trivial instances of a natural geometric optimization problem that is NP-hard, even to approximate. A possible improvement could arise by using more sophisticated methods for solving the involved IPs, e.g., by identifying and employing appropriate cutting planes of the LP relaxation. As mentioned in the comparison of time spent on running different program components, a larger speedup would require also improving the geometric components; a possible line of attack could be a combination of column generation using only subsets of geometric constraints. This requires a more sophisticated interleaving of program components, so it is not clear how much this would help.

Clearly, one of the critical aspects concerns the typical size of the longest non-separated edge: if this tends to be relatively small, the overall runtime will also be not too large, as only a relatively small number of short edges need to be considered. Moreover, if the gap between this upper bound $\hat{b}$ and the optimal index b^* is small, an index search method such as the doubling approach may usually end up with a relatively small remaining interval for the final IP 4.7, and thus with a small overall runtime. See Figure 17 for a comparison of bounds and optimal indices. It appears that the expected values both for $\hat{b}$ and b^* grow relatively slowly with n: a linear regression of 70,000 random instances with up to 700 points yields a growth of close to $n/10$, out of $\Theta(n^2)$ total edges; see Figure 18.

Proving analytical results for these expected values is an interesting challenge. Note that it is an easy exercise (based on standard collision arguments from hashing) to prove that the expected length for the shortest edge between n points chosen uniformly at random from a unit square is $\Theta(1/n)$. Therefore, establishing the same bound for the shortest *non-separated* edge would imply that the expected value of the optimal value b^* is also $\Theta(1/n)$. We conjecture that this is the case, based on the empirical evidence shown in Figure 18.

Finally, we remark that our techniques for MELT can be used as a first step for computing other types of triangulation problems in which no edges below a certain bound must be used. To some extent, this is already present in our completion routine for turning a set of non-crossing separators into a triangulation, as we obtain a constrained Delaunay triangulation.

Acknowledgment

We thank Christiane Schmidt and Joe Mitchell for helpful conversations. We thank two anonymous reviewers for various constructive comments that helped to improve the presentation of this paper.

References

[1] Cgal, Computational Geometry Algorithms Library. http://www.cgal.org.

[2] P. K. Agarwal, M. de Berg, J. Gudmundsson, M. Hammar, and H. J. Haverkort. Box-trees and r-trees with near-optimal query time. *Discrete & Computational Geometry*, pages 291–312, 2002.

[3] T. Baumgartner, S. P. Fekete, A. Kröller, and C. Schmidt. Exact solutions and bounds for general art gallery problems. In *Proceedings of the Twelfth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 11–22. SIAM, 2010.

[4] T. Baumgartner, S. P. Fekete, A. Kröller, and C. Schmidt. Exact solutions and bounds for general art gallery problems. *Journal of Experimental Algorithmics*, 17(2.3), 2012.

[5] D. Borrmann, P. de Rezende, C. de Souza, S. P. Fekete, A. Kröller, A. Nüchter, and C. Schmidt. Point guards and point clouds: Solving general art gallery problems. In *Proceedings of the 29th Annual ACM Symposium on Computational Geometry (SoCG)*, pages 347–348, 2013.

[6] F. Y. L. Chin, J. Qian, and C. A. Wang. Progress on maximum weight triangulation. In *Proceedings of the 10th International Computing and Combinatorics Conference (COCOON)*, pages 53–61, 2004.

[7] M.-L. Chou. Game physics: Broadphase – dynamic aabb tree. http://allenchou.net/2014/02/game-physics-broadphase-dynamic-aabb-tree/.

[8] M. C. Couto, P. J. de Rezende, and C. C. de Souza. An IP solution to the art gallery problem. In *Proceedings of the 25th Symposium on Computational Geometry (SoCG)*, pages 88–89, 2009.

[9] M. C. Couto, C. C. de Souza, and P. J. de Rezende. Experimental evaluation of an exact algorithm for the orthogonal art gallery problem. In *Proceedings of the 7th International Workshop Experimental Algorithms (WEA)*, pages 101–113, 2008.

[10] B. Crepaldi, P. J. de Rezende, and C. C. de Souza. An efficient exact algorithm for the natural wireless localization problem. In *Proceedings of the 25th Canadian Conference on Computational Geometry (CCCG)*, 2013.

[11] J. A. De Loera, J. Rambau, and F. Santos. *Triangulations*. Springer, 2010.

[12] B. Delaunay. Sur la sphère vide. *Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk*, 7:793–800, 1934.

[13] H. Edelsbrunner and T. S. Tan. A quadratic time algorithm for the minmax length triangulation (extended abstract). In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 414–423, 1991.

[14] H. Edelsbrunner and T. S. Tan. A quadratic time algorithm for the minmax length triangulation. *SIAM Journal on Computing*, 22:527–551, 1993.

[15] S. P. Fekete, S. Friedrichs, A. Kröller, and C. Schmidt. Facets for art gallery problems. In *Proceedings of the 19th International Computing and Combinatorics Conference (COCOON)*, pages 208–220, 2013.

[16] S. P. Fekete, W. Hellmann, M. Hemmer, A. Schmidt, and J. Troegel. Computing maxmin edge length triangulations. In *Proceedings of the 17th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 55–69, 2015.

[17] S. P. Fekete, M. E. Lübbecke, and H. Meijer. Minimizing the stabbing number of matchings, trees, and triangulations. In *Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 437–446, 2004.

[18] S. P. Fekete, M. E. Lübbecke, and H. Meijer. Minimizing the stabbing number of matchings, trees, and triangulations. *Discrete & Computational Geometry*, 40(4):595–621, 2008.

[19] S. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987.

[20] S. Hu. A linear time algorithm for max-min length triangulation of a convex polygon. *Information Processing Letters*, 101(5):203–208, Mar. 2007.

[21] L. Lovász, K. Vesztergombi, U. Wagner, and E. Welzl. Convex quadrilaterals and k-sets. In *Contemporary Mathematics Series, 342, AMS 2004*, pages 139–148, 2004.

[22] N. Megiddo and A. Tamir. On the complexity of locating linear facilities in the plane. *Operations Research Letters*, 1:194–197, 1983.

[23] W. Mulzer and G. Rote. Minimum-weight triangulation is NP-hard. *Journal of the ACM*, 55(2):A11, 2008.

[24] B. Piva, C. C. de Souza, Y. Frota, and L. Simonetti. Integer programming approaches for minimum stabbing problems. *RAIRO - Operations Research*, 48(2):211–233, 2014.

[25] J. Qian and C. A. Wang. A linear-time approximation scheme for maximum weight triangulation of convex polygons. *Algorithmica*, 40(3):161–172, 2004.

[26] G. Reinelt. TSPlib — A Traveling Salesman Problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

[27] J. Remy and A. Steger. A quasi-polynomial time approximation scheme for minimum weight triangulation. *Journal of the ACM*, 56(3):A15, 2009.

[28] E. R. Scheinerman and H. S. Wilf. The rectilinear crossing number of a complete graph and sylvester's "four point problem" of geometric probability. *The American Mathematical Monthly*, 101(10):939–943, 1994.

[29] C. Schmidt. Maxmin length triangulation in polygons. In *Proceedings of the 28th European Workshop on Computational Geometry (EuroCG)*, pages 121–124, 2012.

[30] D. C. Tozoni, P. J. de Rezende, and C. C. de Souza. The quest for optimal solutions for the art gallery problem: A practical iterative algorithm. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA)*, pages 320–336, 2013.

[31] L. Vismara. Planar straight-line drawing algorithms. In R. Tamassia, editor, *Handbook of Graph Drawing and Visualization*, chapter 6, pages 193–222. CRC Press, 2013. Availabe at https://cs. brown.edu/˜rt/gdhandbook/chapters/straightline.pdf.

[32] M. J. O. Zambon, P. J. de Rezende, and C. C. de Souza. An exact algorithm for the discrete chromatic art gallery problem. In *Proceedings of the 13th International Symposium on Experimental Algorithms (SEA)*, pages 59–73, 2014.