Computing MaxMin Edge Length Triangulations

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

Abstract

In 1991, Edelsbrunner and Tan gave an $O(n^2)$ algorithm for finding the MinMax 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 IPs. 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 should be optimized. Examples include maximizing the minimum angle, 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 [12]. The proof implies 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 [10] by De Loeara, Rambau, and Santos. Maximizing the minimum *angle* in a triangulation is achieved by the Delaunay triangulation [11]; making use of Fortune's sweepline algorithm [17], it can be computed in $O(n \log n)$. Minimizing the maximum *edge* can be computed in quadratic time, as shown by Edelsbrunner and Tan [12, 13]. One of the most notorious problems regarding triangulations was finally resolved by Mulzer and Rote [21], who proved that finding a triangulations of minimum total edge length (a *minimum-weight triangulation*) is an NP-hard problem. As shown by Remy and Steger [25], there is a PTAS for this problem. The maximum-weight triangulation problem has also been considered by Qian, and Wang [23], 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 triangulation has been considered by Hu [18], 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-

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

hard. C. Schmidt [26] showed that finding a *geometric* MaxMin triangulation is NP-complete in the presence of obstacles, such as inside of a polygon with holes; she also showed that computing a MaxMin 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 [3–5,7,8,14,27], an exact approach for solving Chromatic Art Gallery Problems [29], an exact method for solving NP-hard stabbing problems [22], where the objective is to determine a matching, a spanning tree or a triangulation with a small stabbing or crossing number (see [15,16]), and an approach for computing optimal solutions for wireless localization methods [9], 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 allow 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 promises to be much smaller.
- We give a further reduction on IP size, based on using appropriate linear combinations of valid constraints, resulting in a further reduction of size.
- We discuss a number 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 all real-world geometric instances up to $n = 200$ in the well-known TSPLIB, and 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 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 $j < b$, 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 $X^*_{\geq b}$.

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 [20], 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 that are required to be disjoint.

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

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

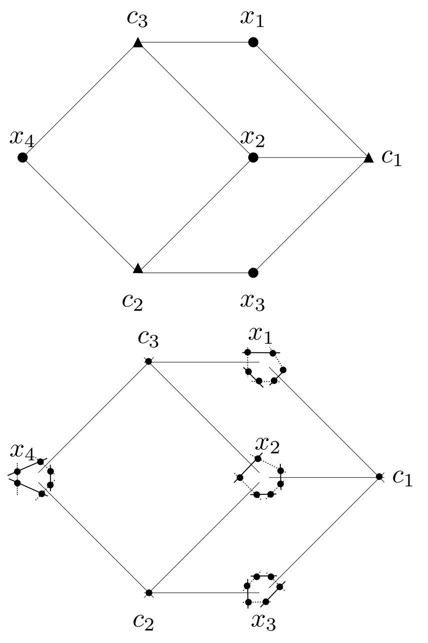

Figure 1: (Top) A planar straight-line drawing of the variable-clause incidence graph G_I of the PLANAR 3SAT instance $I = (x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee x_3 \vee \overline{x_4}) \wedge (\overline{x_1} \vee x_2 \vee x_4)$. (Bottom) A CDS instance for the 3SAT instance 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 (top) 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 [28]. 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 $O(n^2)$ results in an arrangement in which each pair of vertices are at least 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 $\delta(v)$ "even" and $\delta(v)$ "odd" variable segments; their precise location and parity is chosen such that an edge $\overline{v_x v_c}$ in G_I intersects an odd segment iff variable x occurs in clause c in an unnegated fashion, and an edge $\overline{v_x v_c}$ in G_I 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 an appropriate 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, if and only if 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 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 $\overline{p_i p_j}$.*

Now we proceed to the main theorem.

THEOREM 3.1. *It is NP-hard to decide whether a set P of n points in the plane has a triangulation with smallest edge of length at least L, for some positive number L.*

Proof. 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. We perturb all points in $Q \cup T$ by appropriate powers of $1/n$, such that the only triples of collinear points correspond to segments in S with their covered points. 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 $\delta > 0$.

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 an appropriately small distance $\varepsilon << \delta$ in opposite sectors. See Figure 2.

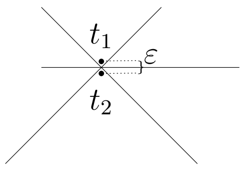

Figure 2: Replacing the target points of the CDS instance by point pairs at distance ε.

We claim: there is a triangulation with shortest edge of length 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 $\overline{t_1t_2}$. As all points in T are covered, all close edges are separated, and the claim follows from Lemma 3.3.

Conversely, any edge $\overline{t_1t_2}$ 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. This induces a solution to the CDS instance. □

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 small enough that $\delta/\varepsilon > p(n)$. Then 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 in 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.

$$\text{(4.1)} \quad \max \quad 0$$

$$\text{(4.2)} \quad s.t. \quad x_{e_i} + x_{e_j} \leq 1 \qquad \forall \{e_i, e_j\} \in X_{\geq b}$$

$$\text{(4.3)} \quad \sum_{e_i \in E} x_{e_i} = 3n - 3 - n_c \qquad \forall e_i \in E_{\geq b}$$

$$\text{(4.4)} \quad x_e \in \{0,1\} \qquad \forall e \in E_{\geq b}$$

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 edges corresponds to the number of segment crossings, 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. [19] 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 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 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 illustrating example for the rest of the rest of this subsection, the reader is referred to Figures 3 and 4.)

Integer Program 4.3.

$$\max \quad 0 \tag{4.5}$$

$$\text{s.t.} \quad x_{e_i} + x_{e_j} \leq 1 \qquad \forall \{e_i, e_j\} \in X^*_{\geq b} \tag{4.6}$$

$$\sum_{e_i \in E^*_{\geq b}(e_j)} x_{e_i} \geq 1 \qquad \forall e_j \in E_{<b} \tag{4.7}$$

$$x_{e_i} \in \{0,1\} \qquad \forall e_i \in E^*_{\geq b} \tag{4.8}$$

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.*

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 greatly reduced from m. However, we still have $O(\overline{m}^2_{\geq b})$ constraints of type (4.6). Using an IP reformulation and $k_e := |E^*_{\geq L}(e)|$ for any $e \in E_{<b}$, we can condense families of constraints into one, leading to a further reduction to $O(\overline{m}_{\geq b})$ constraints.

Integer Program 4.5.

$$\max \quad 0 \tag{4.9}$$

$$\text{s.t.} \quad k_e \cdot x_e + \sum_{e_i \in E^*_{\geq b}(e)} x_{e_i} \leq k_e \qquad \forall e \in E^*_{\geq b} \tag{4.10}$$

$$\sum_{e_i \in E^*_{\geq b}(e)} x_{e_i} \geq 1 \qquad \forall e \in E_{<a} \tag{4.11}$$

$$x_{e_i} \in \{0,1\} \quad \forall e_i \in E^*_{\geq b} \tag{4.12}$$

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 (4.6), and any solution to IP 4.3 satisfies all constraints (4.10).

For the first part, assume that some feasible solution of IP 4.5 violates a constraint of type (4.6), so there are two intersecting edges $e, e' \in X^*_{\geq L}$ with $x_e = x_{e'} = 1$; however, this implies that the left-hand side of constraint (4.10) for e becomes at least $k_e + 1$, as $e' \in E^*_{\geq L}(e)$, a contradiction.

For the converse, consider a feasible solution of IP 4.3 and observe that a constraint (4.10) for some edge $e \in E^*_{\geq L}$ can only be violated if $x_e = 1$ as well as $x_{e'} = 1$ for some $e' \in E^*_{\geq L}(e)$; this implies that $\{e, e'\} \in X^*_{\geq L}$, and the corresponding constraint (4.6) 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.

$$\max \quad b \tag{4.13}$$

$$\text{s.t.} \quad x_{e_i} \cdot i + (1 - x_{e_i}) \cdot c \geq b \qquad \forall i \in [a, c[\tag{4.14}$$

$$k_e \cdot x_e + \sum_{e_i \in E^*_{\geq a}(e)} x_{e_i} \leq k_e \qquad \forall e \in E^*_{\geq a} \tag{4.15}$$

$$x_e + \sum_{e_i \in E^*_{\geq a}(e)} x_{e_i} \geq 1 \qquad \forall e \in E_{[a,c[} \tag{4.16}$$

$$\sum_{e_i \in E^*_{\geq L}(e)} x_{e_i} \geq 1 \qquad \forall e \in E_{<a} \tag{4.17}$$

$$x_{e_i} \in \{0,1\} \quad \forall e_i \in E^*_{\geq a} \tag{4.18}$$

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 (4.14) enforce that this index is indeed met, as it sets a non-critical bound of c for all unused edges. The additional Constraint (4.16) ensures that an edges 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.*

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

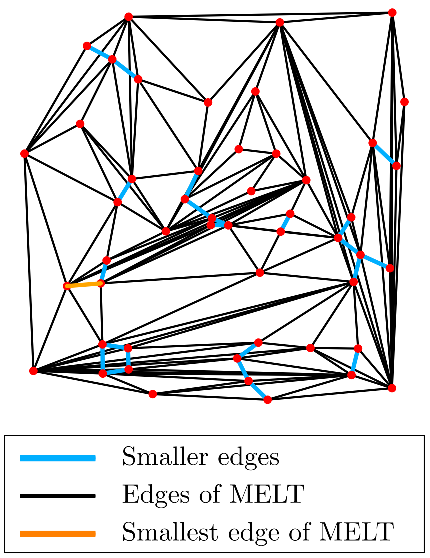

Figure 3: A solution of MELT for an instance with 50 random points in the plane. The smallest edge of the triangulation has index 24. This smallest edge (orange) has possible separating edges, but all these edges cross other separators of even smaller edges.

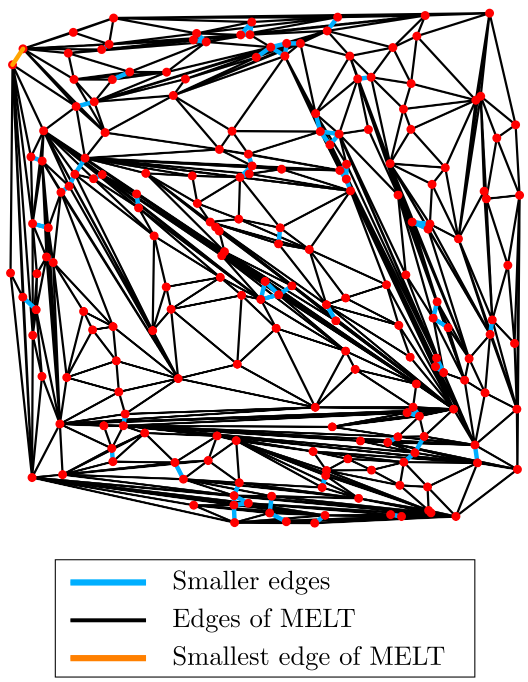

Figure 4: A solution of MELT for an instance with 200 random points. The smallest edge of the triangulation has index 82 and lies on the convex hull, so it matches the hull bound.

be optimized (i.e., when using IP 4.7). To this end, we give a natural upper bound and an improved one that 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 12. By Lemma 3.3, *any* 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.

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

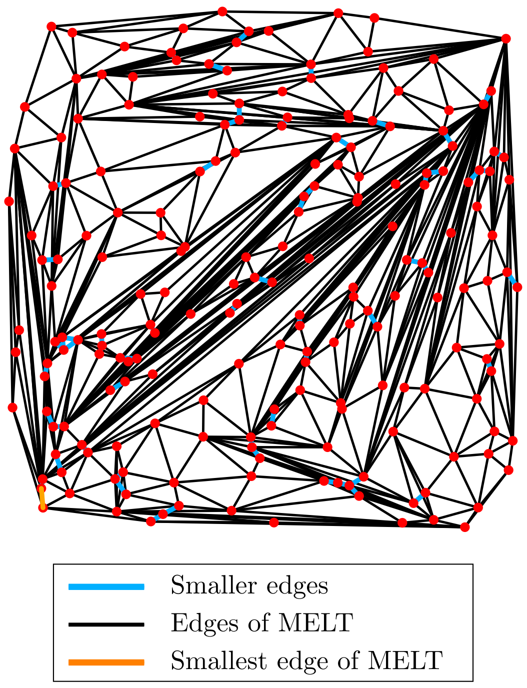

Figure 5: A solution of MELT for an instance with 200 random points. The smallest edge of the triangulation has index 66 and lies close to the convex hull, without a crossing edge, so it matches the non-separated bound.

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.

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`. This has the additional side effect that the resulting triangles are relatively well shaped.

5.1.5 Point Generation. For the random instances, points are generated uniformly at random from 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 [24].

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 14.04 LTS`, using 8 GB RAM, and a 1.8 GHz `Intel Core i3-3217U CPU`, a processor with two physical and two 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 4.8.2` compiler for `C++`. It uses the following options: `-O3 -frounding-math`, used libraries are CGAL 4.4 and CPLEX 12.6. The option `-O3` optimized the whole program if 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.

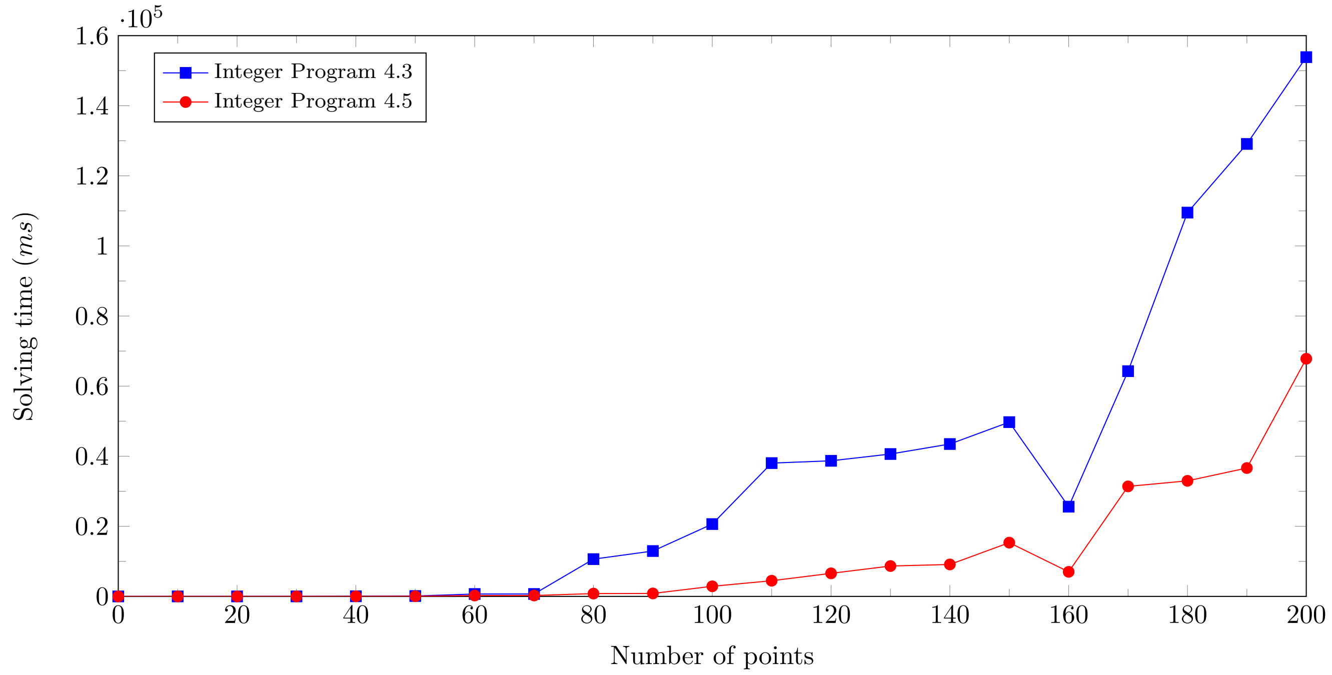

Figure 6: Average solution time of instances: solving time of the IPs.

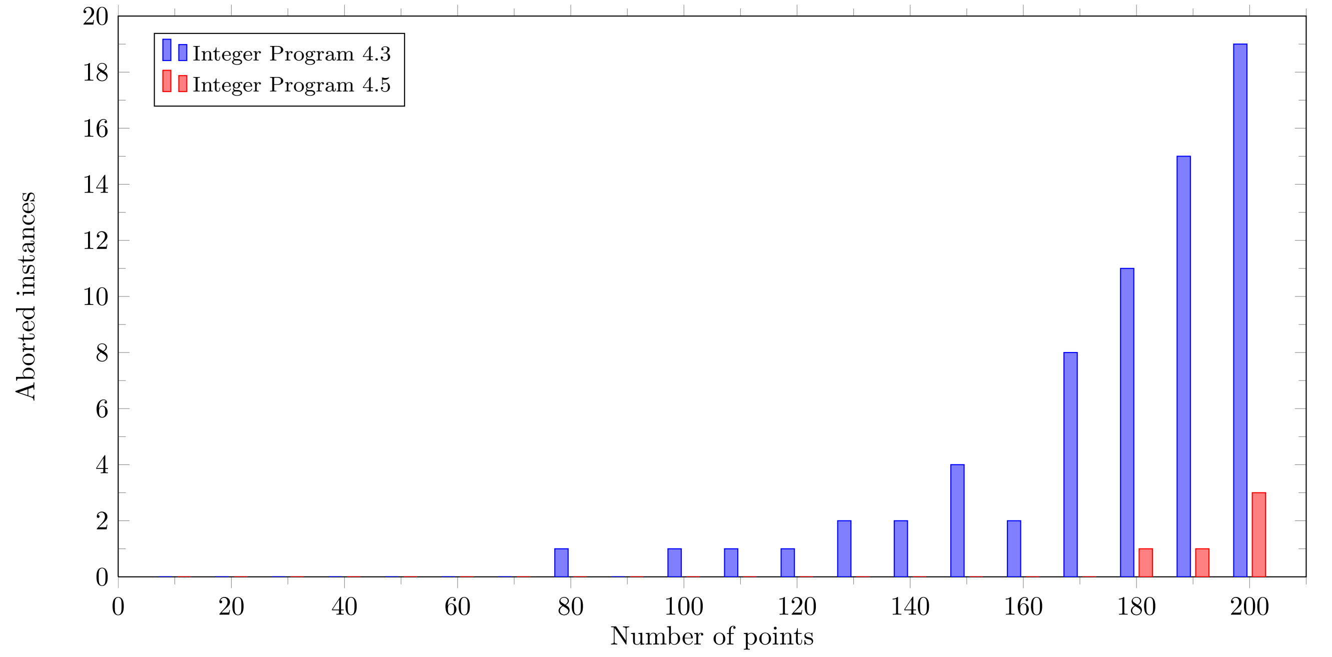

Figure 7: Aborted instances of the IPs, out of 100 instances.

For each size $n \in \{10, 20, \ldots, 200\}$, we generated 100 instances uniformly at random from a unit square. For different solution approaches, always the same instances were used, to ensure a fair comparison. 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 [24]. We solved all geoometric instances up to $n = 200$, along with many bigger ones. See Figures 16 and 15 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 16), while others are more clustered (e.g., Figure 15). 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 14 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 6 shows the overall average time per instance for solving the IPs in multiples of 100 seconds, while Figure 7 compares the number of instances (out of 100) with timeout. Clearly, IP 4.7 is greatly superior. As a consequence, only this IP is used in further experiments. (Note that IP 4.1 is quite terrible, so we are not showing numerical results.)

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 solution times of an IP feasibility check versus 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 8, 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 9 shows how much time on average was spent on each component, with "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, triangulation the points, recording the output), and other overhead; in four of the cases shown in Figure 10, this included nontrivial 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 aiming for a moderate further speedup should aim at the IPs, while significant speedup can only be achieved by tuning *both* geometry and optimization.

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 with 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 big, 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 12 for a comparison of bounds and optimal indices. It appears that the expected values both for $\hat{b}$ and b^* grow relatively slowly

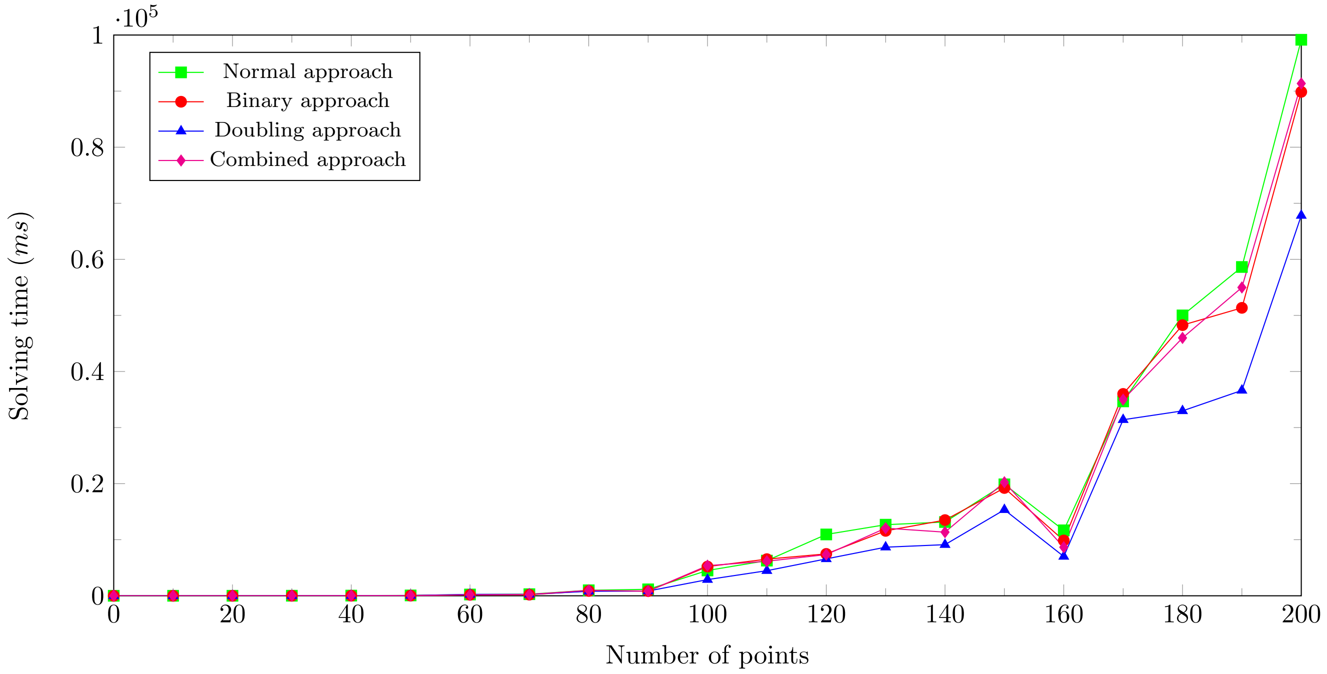

Figure 8: Solution times for the different index searches, for random instances.

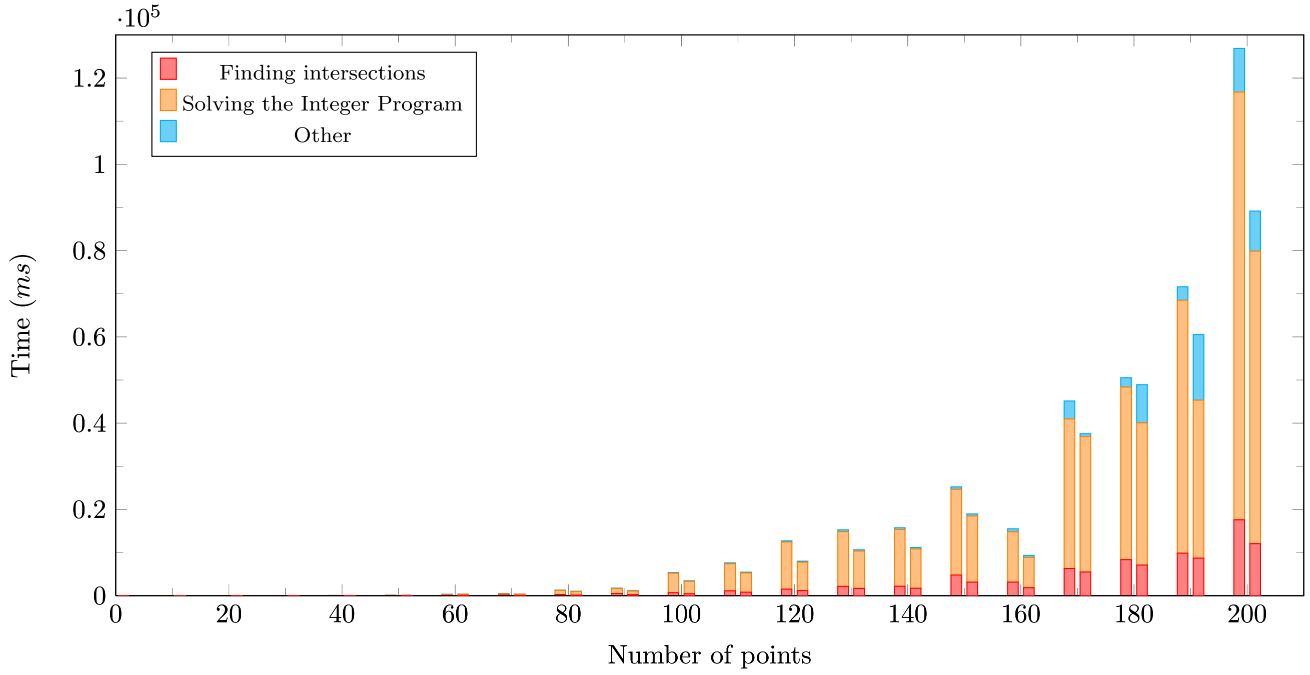

Figure 9: Average time (for random 100 random instances for each size) of the program components.
Left bars: Normal approach, based on IP 4.7
Right bars: Doubling approach, with AABB-Tree and IP 4.7

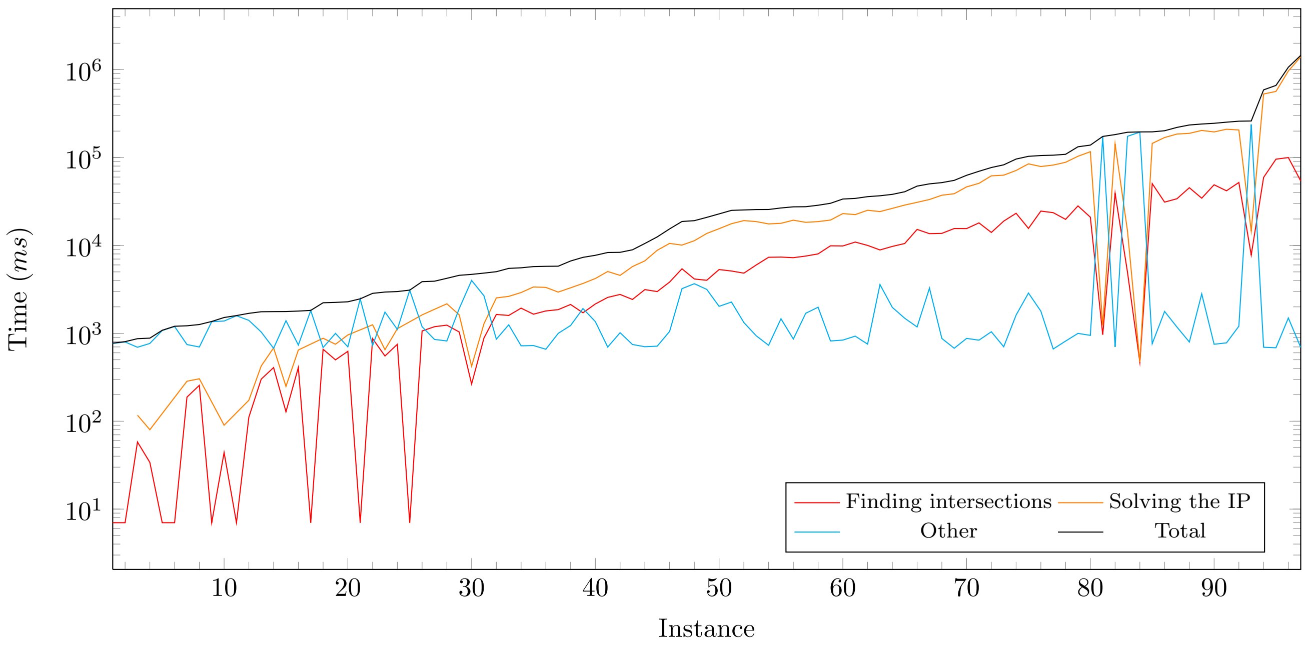

Figure 10: Diagram of the time for all 97 (out of 100) solved random instances of 200 points. For a better overview, the y-axis is scaled logarithmically.

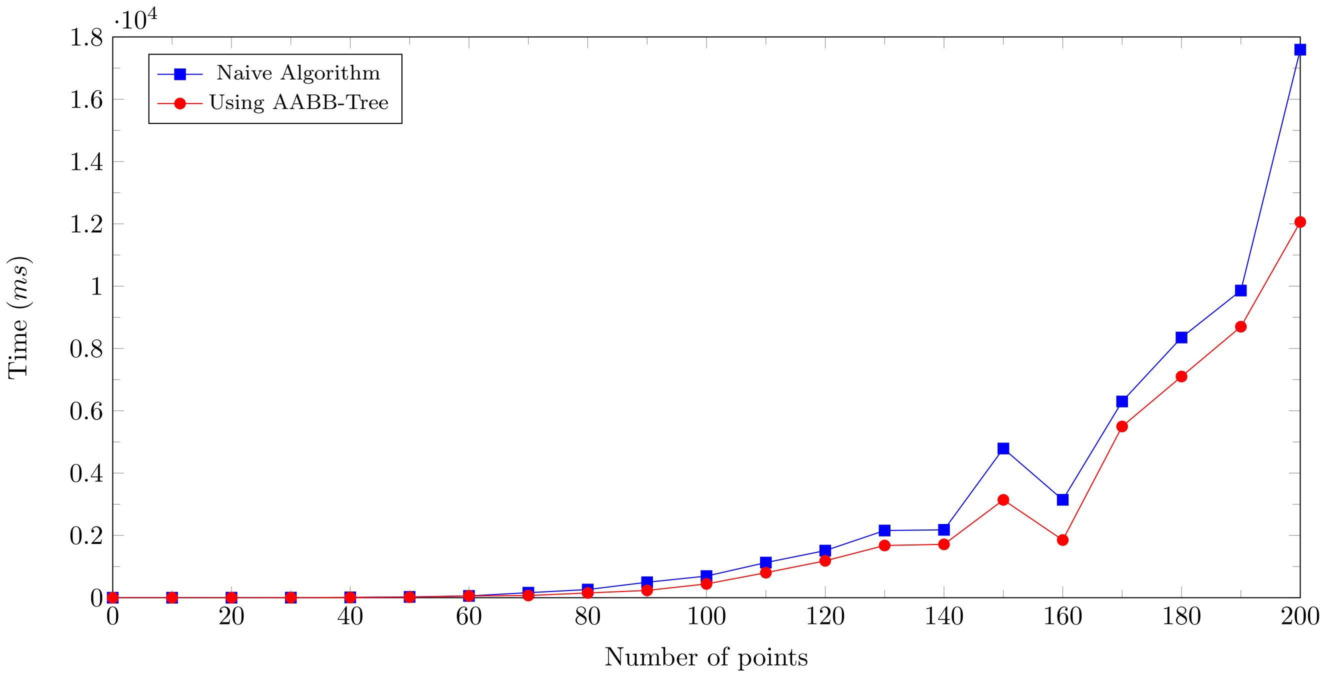

Figure 11: Average time for finding intersections in random instances.

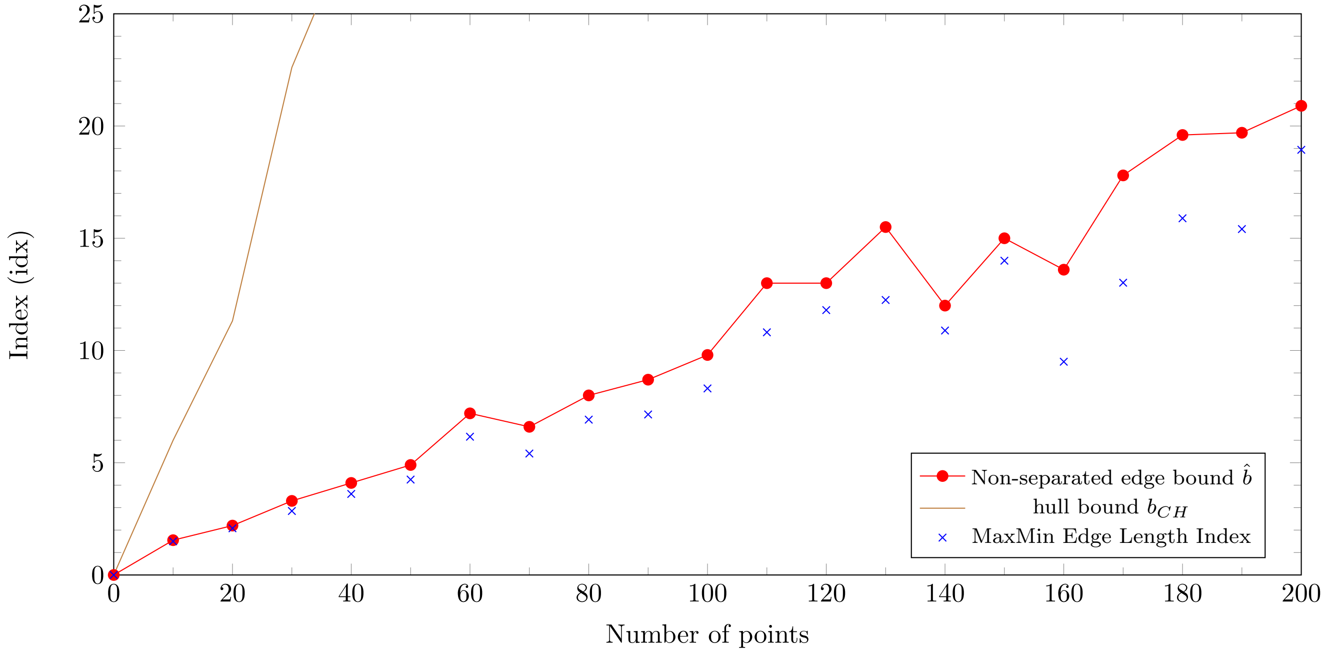

Figure 12: Average values for hull bound, non-separated edge bound, and optimal index for random instances.

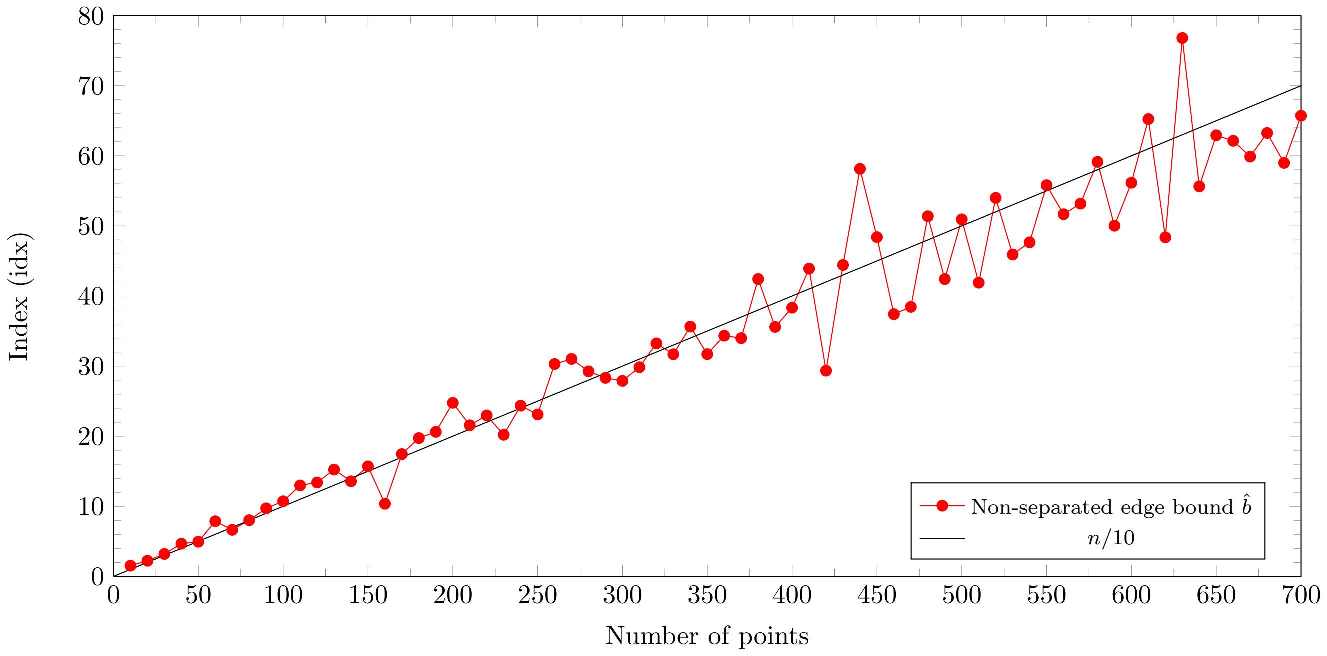

Figure 13: Average indices (for 100 random instances for each size) of the non-separated edge bound $\hat{b}$ for random sets with up to 700 points. (Error bars indicate one standard deviation.)

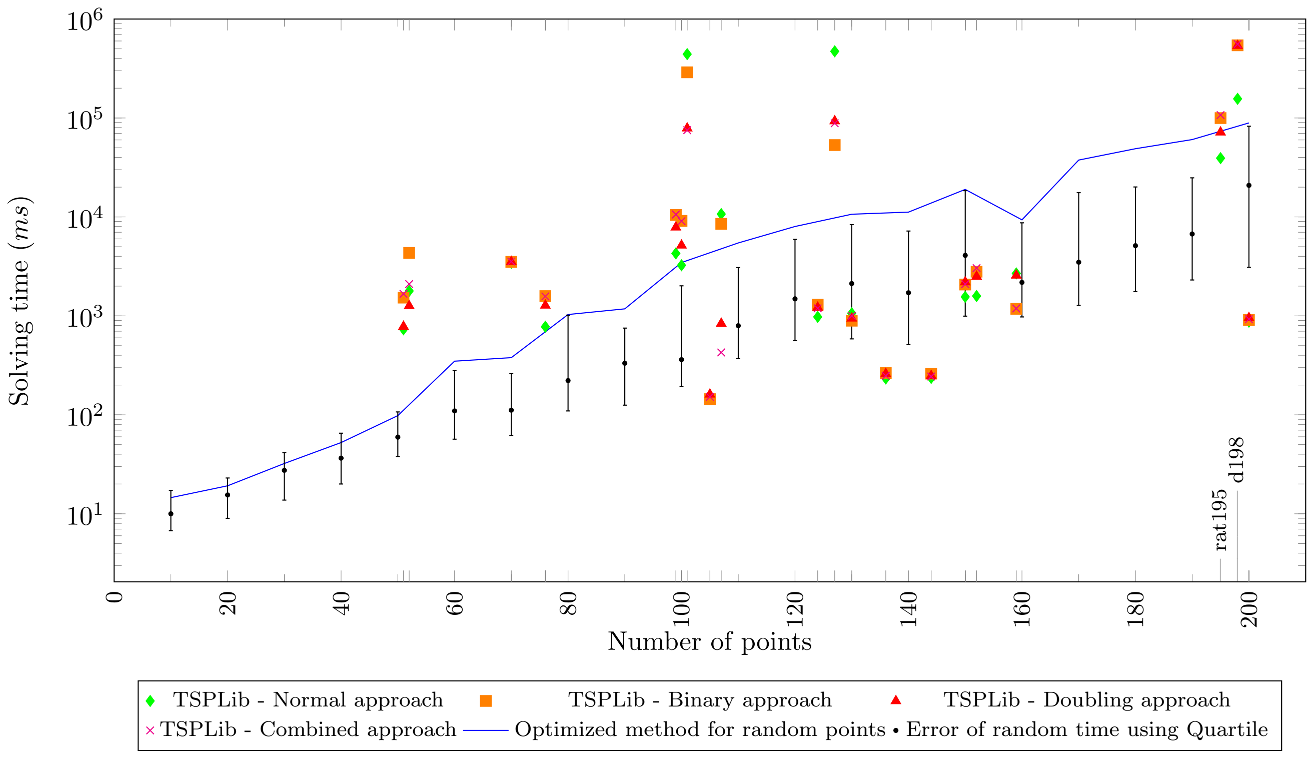

Figure 14: Comparison of TSPLIB instances with average times for random instances, using the doubling approach. (Error bars indicate percentiles 25, 50, 75.)

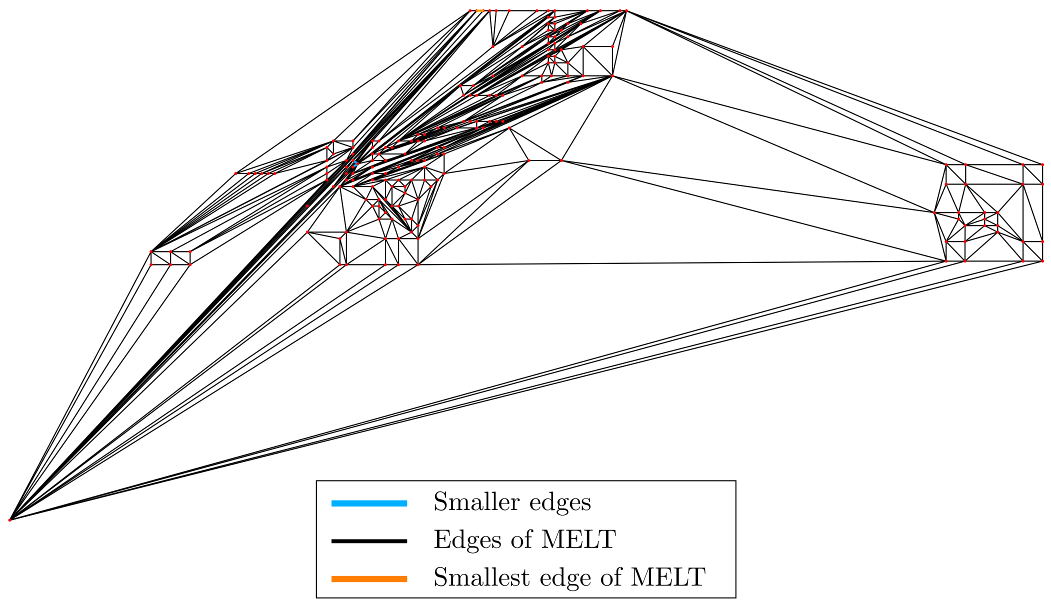

Figure 15: A solution of MELT for the d198 instance with 198 points of the TSPLIB. The smallest edge of the triangulation has index 3 and lies on the convex hull, so it matches the hull bound. Also this instance has many edges of the same length.

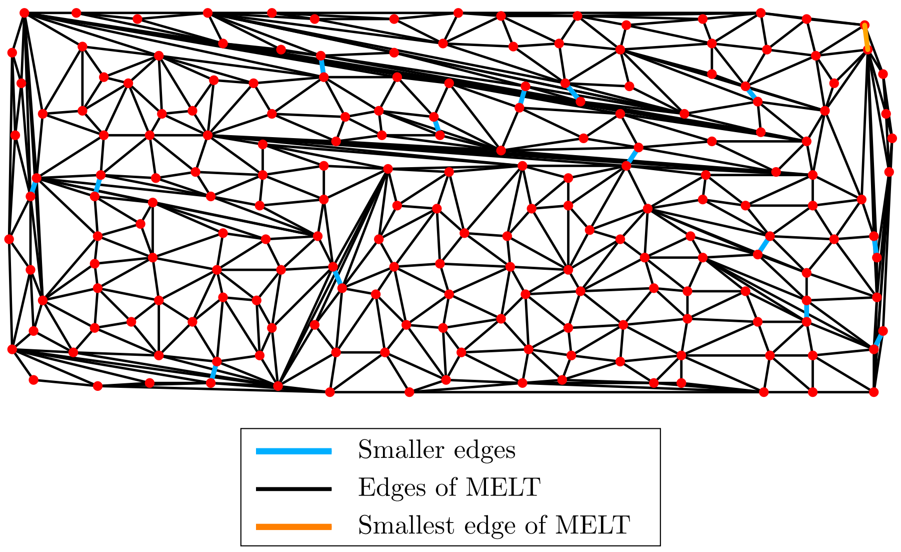

Figure 16: A solution of MELT for the rat195 instance with 195 points of the TSPLIB. The smallest edge of the triangulation has index 15. Also the points of this instance are evenly distributed.

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 13.

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 13.

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.

References

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

[2] Pankaj K. Agarwal, Mark de Berg, Joachim Gudmundsson, Mikael Hammar, and Herman J. Haverkort, *Box-trees and r-trees with near-optimal query time*, Discrete & Computational Geometry (2002), 291–312.

[3] Tobias Baumgartner, Sándor P. Fekete, Alexander Kröller, and Christiane Schmidt, *Exact solutions and bounds for general art gallery problems*, Proceedings of the Twelfth Workshop on Algorithm Engineering and Experiments (ALENEX 2010), SIAM, 2010, pp. 11–22.

[4] ———, *Exact solutions and bounds for general art gallery problems*, Journal of Experimental Algorithmics **17** (2012), no. 2.3.

[5] Dorit Borrmann, Pedro de Rezende, Clécio de Souza, Sándor P. Fekete, Alexander Kröller, Andreas Nüchter, and Christiane Schmidt, *Point guards and point clouds: Solving general art gallery problems*, Proceeding of the 29th Annual ACM Symposium on Computational Geometry (SoCG 2013), 2013, pp. 347–348.

[6] Francis Y. L. Chin, Jianbo Qian, and Cao An Wang, *Progress on maximum weight triangulation*, COCOON, 2004, pp. 53–61.

[7] Marcelo C. Couto, Pedro Jussieu de Rezende, and Cid C. de Souza, *An IP solution to the art gallery problem*, Proceedings of the 25th ACM Symposium on Computational Geometry, Aarhus, Denmark, June 8-10, 2009, 2009, pp. 88–89.

[8] Marcelo C. Couto, Cid C. de Souza, and Pedro Jussieu de Rezende, *Experimental evaluation of an exact algorithm for the orthogonal art gallery problem*, Experimental Algorithms, 7th International Workshop, WEA 2008, 2008, pp. 101–113.

[9] Bruno Crepaldi, Pedro J. de Rezende, and Cid C. de Souza, *An efficient exact algorithm for the natural wireless localization problem*, Proceedings of the 25th Canadian Conference on Computational Geometry, CCCG, 2013.

[10] Jesus A. De Loera, Joerg Rambau, and Francisco Santos, *Triangulations*, Springer, 2010.

[11] Boris Delaunay, *Sur la sphère vide*, Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk **7** (1934), 793–800.

[12] Herbert Edelsbrunner and Tiow Seng Tan, *A quadratic time algorithm for the minmax length triangulation (extended abstract)*, Proceedings of FOCS, 1991, pp. 414–423.

[13] ———, *A quadratic time algorithm for the minmax length triangulation*, SIAM J. Comput **22** (1993), 527–551.

[14] Sándor P. Fekete, Stephan Friedrichs, Alexander Kröller, and Christiane Schmidt, *Facets for art gallery problems*, Computing and Combinatorics (COCOON 2013), Lecture Notes in Computer Science, vol. 7936, 2013, pp. 208–220.

[15] Sándor P. Fekete, Marco E. Lübbecke, and Henk Meijer, *Minimizing the stabbing number of matchings, trees, and triangulations*, Proc. 15th ACM-SIAM Sympos. Discrete Algorithms, SIAM, 2004, pp. 437–446.

[16] ———, *Minimizing the stabbing number of matchings, trees, and triangulations*, Discrete Comput. Geom. **40** (2008), no. 4, 595–621.

[17] Steven Fortune, *A sweepline algorithm for voronoi diagrams*, Algorithmica **2** (1987), 153–174.

[18] Shiyan Hu, *A linear time algorithm for max-min length triangulation of a convex polygon*, Inf. Process. Lett. **101** (2007), no. 5, 203–208.

[19] László Lovász, Katalin Vesztergombi, Uli Wagner, and Emo Welzl, *Convex quadrilaterals and k-sets*, Contemporary Mathematics Series, 342, AMS 2004, 2004, pp. 139–148.

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

[21] Wolfgang Mulzer and Günter Rote, *Minimum-weight triangulation is NP-hard*, J. ACM **55** (2008), no. 2, A11.

[22] Breno Piva, Cid C. de Souza, Yuri Frota, and Luidi Simonetti, *Integer programming approaches for minimum stabbing problems*, RAIRO - Operations Research **48** (2014), no. 2, 211–233.

[23] Jianbo Qian and Cao An Wang, *A linear-time approximation scheme for maximum weight triangulation of convex polygons*, Algorithmica **40** (2004), no. 3, 161–172.

[24] Gerhard Reinelt, *TSPlib - A Traveling Salesman Problem library*, ORSA Journal on Computing **3** (1991), no. 4, 376–384.

[25] Jan Remy and Angelika Steger, *A quasi-polynomial time approximation scheme for minimum weight triangulation*, J. ACM **56** (2009), no. 3, A15.

[26] Christiane Schmidt, *Maxmin length triangulation in polygons*, Proceedings of the 28th European Workshop on Computational Geometry, 2012, pp. 121–124.

[27] Davi C. Tozoni, Pedro J. de Rezende, and Cid C. de Souza, *The quest for optimal solutions for the art gallery problem: A practical iterative algorithm*, Experimental Algorithms, 12th International Symposium, SEA 2013, Rome, Italy, June 5-7, 2013. Proceedings, 2013, pp. 320–336.

[28] Luca Vismara, *Planar straight-line drawing algorithms*, Handbook of Graph Drawing and Visualization (Roberto Tamassia, ed.), CRC Press, 2013, To appear, availabe at https://cs.brown.edu/~rt/gdhandbook/chapters/straightline.pdf, pp. 193–222.

[29] Maurício J. O. Zambon, Pedro J. de Rezende, and Cid C. de Souza, *An exact algorithm for the discrete chromatic art gallery problem*, Experimental Algorithms - 13th International Symposium, SEA 2014, 2014, pp. 59–73.