Solving Large-Scale Minimum-Weight Triangulation Instances to Provable Optimality*

Andreas Haas[1]

1 **Department of Computer Science, TU Braunschweig, 38106 Braunschweig, Germany.** haas@ibr.cs.tu-bs.de

Abstract

We consider practical methods for the problem of finding a minimum-weight triangulation (MWT) of a planar point set, a classic problem of computational geometry with many applications. While Mulzer and Rote proved in 2006 that computing an MWT is NP-hard, Beirouti and Snoeyink showed in 1998 that computing provably optimal solutions for MWT instances of up to 80,000 *uniformly distributed* points is possible, making use of clever heuristics that are based on geometric insights. We show that these techniques can be refined and extended to instances of much bigger size and different type, based on an array of modifications and parallelizations in combination with more efficient geometric encodings and data structures. As a result, we are able to solve MWT instances with up to 30,000,000 uniformly distributed points in less than 4 minutes to provable optimality. Moreover, we can compute optimal solutions for a vast array of other benchmark instances that are *not* uniformly distributed, including normally distributed instances (up to 30,000,000 points), all point sets in the TSPLIB (up to 85,900 points), and VLSI instances with up to 744,710 points. This demonstrates that from a practical point of view, MWT instances can be handled quite well, despite their theoretical difficulty.

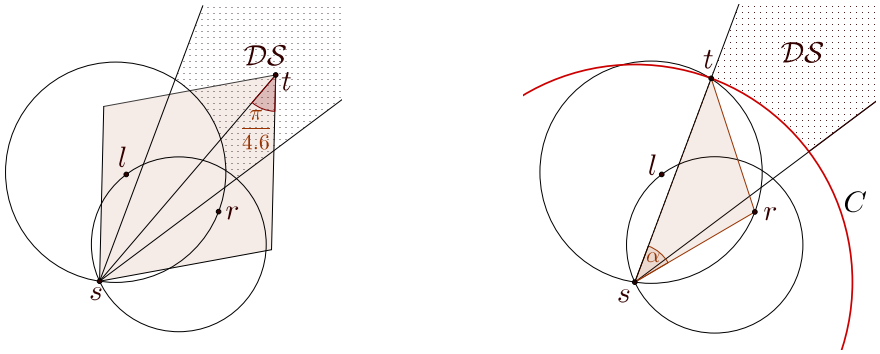
1 Introduction

Triangulating a set of points in the plane is a classic problem in computational geometry: given a planar point set S, find a maximal set of non-crossing line segments connecting the points in S. Triangulations have many real-world applications, for example in terrain modeling, finite element mesh generation and visualization. In general, a point set has exponentially many possible triangulations and a natural question is to ask for a triangulation that is optimal with respect to some optimality criterion. A natural criterion is to minimize the total weight of the resulting triangulation. As Mulzer and Rote [11] showed, it is NP-hard to compute a minimum-weight triangulation (MWT).

Practical approaches for computing an MWT are based on heuristics for including or excluding edges with certain properties from any minimum-weight triangulation. Das and Joseph [4] showed that every edge in an MWT has the *diamond property*. An edge e cannot be in MWT(S) if both of the two isosceles triangles with base e and base angle $\pi/8$ contain other points of S. Drysdale et al. [7] improved the angle to $\pi/4.6$. This can greatly reduce the edge set and works exceedingly well on uniformly distributed point sets, for which only $O(n)$ edges remain in expectation. Dickerson et al. [5,6] proposed the *LMT-skeleton heuristic*, based on a local criterion fulfilled by every edge in MWT(S). The LMT-skeleton algorithm often yields a connected graph, and the remaining polygonal faces can be triangulated with dynamic programming to obtain an MWT. Combining the diamond property and the LMT-skeleton makes it possible to compute the MWT for large, well-behaved point sets. Beirouti and Snoeyink [2] showed an efficient implementation of these two heuristics and they reported that their implementation could compute the exact MWT of 40,000 uniformly

* A full version of the paper is available at [9].

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.



(a) Points l and r induce a region $\mathcal{DS}$ such that all edges $e = st$ with $t \in \mathcal{DS}$ fail the diamond test. $\mathcal{DS}$ is called a dead sector (dotted area).

(b) Simplified dead sector $\mathcal{DS}$ is bounded by two rays and circle C. C is induced by the longer of the two edges sl resp. sr and angle α.

Figure 1 Dead sectors.

distributed points in less than 5 minutes and even up to 80,000 points with the improved diamond property.

We revisit diamond test and LMT-skeleton based on Beirouti's and Snoeyink's [2] ideas and describe several improvements. Our bucketing scheme for the diamond test does not rely on a uniform point distribution and filters more edges. For the LMT-skeleton we provide a number of algorithm engineering modifications. These contain a data partitioning scheme for parallelized implementation and other changes for efficiency. We also use an improvement suggested by Aichholzer et al. [1]. Furthermore, we implemented, streamlined and evaluated our implementation on various point sets. For the uniform case, we computed the MWT of 30,000,000 points in less than 4 minutes on commodity hardware; the limiting factor arose from the memory of a standard machine, not from the runtime. We achieved the same performance for normally distributed point sets. The third class of point sets were benchmark instances from the TSPLIB [12] (based on a wide range of real-world and clustered instances) and the VLSI library. These reached a size up to 744,710 points. This shows that from a practical point of view, a wide range of huge MWT instances can be solved to provable optimality with the right combination of theoretical insight and algorithm engineering.

2 Our Improvements and Optimizations

2.1 Diamond Property

For a uniformly distributed point set S with n points, the expected number of edges to pass the diamond test is only $O(n)$. More precisely, Beirouti and Snoeyink [2] state that the number is less than $3\pi n/\sin(\alpha)$, where α is the base angle for the diamond property. We were able to tighten this value.

▶ **Theorem 2.1.** *For a uniformly distributed point set, the expected number of edges that pass the diamond test is less than* $3\pi n/\tan(\alpha)$.

For $\alpha = \pi/4.6$ less than $11.5847n$ edges are expected to pass the test, which is very close to the values observed and achieved by our implementation; see Table 1 in Section 3. In contrast, the value achieved by the implementation of Beirouti and Snoeyink is $\approx 14.3n$ [2].

2.2 Dead Sectors and Bucketing

Our bucketing scheme is based on the same idea of *dead sectors* (see Figure 1a) as described by Beirouti and Snoeyink [2]. We simplify the shape of dead sectors: Instead of bounding

a sector $\mathcal{DS}$ by two circles (as shown in Figure 1a), we only use a single big circle C with center s at the expense of losing a small part of $\mathcal{DS}$. This allows representing dead sectors by just three numbers: an interval of two polar angles, and a squared radius δ; see Figure 1b.

The main ingredient for our bucketing scheme is a spatial search tree with support for incremental nearest neighbor queries, such as a quadtree. Incremental nearest neighbor search queries allow to traverse all nearest neighbors of a point in order of increasing distance. Such queries can be implemented with a priority queue that stores all tree nodes encountered during tree traversal together with the distances to their resp. bounding box (see Hjaltason and Samet [10]). Pruning tree nodes whose bounding box lie in dead sectors is rather simple as follows: consider a nearest neighbor query for point s: when we are about to push a new node n into the priority queue, we compute the smallest polar angle interval I that encloses the bounding box of n and discard n if I is contained in the dead sectors computed so far.

Because nearest neighbors and tree nodes are processed in order of increasing distance, we can store sectors in two stages. On creation, they are inserted into a FIFO-queue; later only the interval component is inserted in a search filter used by the tree. The queue can be seen as a set of pending dead sectors with attached activation distance δ. As soon as we process a point t with $\mathrm{d}(s,t) > \delta$ we can insert the corresponding interval into our filter.

This leaves deciding which points are used to construct dead sectors. We store all points encountered during an incremental search query in an ordered set N, sorted by their polar angle with respect to s. Each time we find a new point t, we insert it into N; dead sectors are computed with the predecessor and the successor of t in N. Computing δ for new sectors only requires multiplying the current squared distance to t with a precomputed constant. The diamond property of edge st is tested against a subset of N.

If we apply the above procedure to every single point, we generate each edge twice, once on each of the two endpoints. Therefore, we output only those edges $e = st$ such that $s < t$, i.e., s is lexicographically smaller than t. As a consequence, we can exclude a part of the left half-space right from the beginning by inserting an initial dead sector $\mathcal{DS}_0 = (1/2\pi + \alpha, 3/2\pi - \alpha)$ at distance 0. Points in the two wedges $(1/2\pi, 1/2\pi + \alpha]$ and $[3/2\pi - \alpha, 3/2\pi]$ are specially treated because they are still useful to generate dead sectors for the right half-space.

2.3 LMT-Skeleton

For "nicely" distributed point sets, a limiting factor of the heuristic is the space required to store the half-edge data structure in memory. We reduce storage overhead by storing all edges in a single array sorted by source vertex (also known as a *compressed sparse row graph*). All outgoing edges of a single vertex are still radially sorted. In addition to the statuses *possible*, *certain*, *impossible*, we store whether an edge lies on the convex hull.

In essence our implementation is still the same as the one given by Beirouti and Snoeyink [2], however, with some optimizations applied. We refer to the central `while loop` in their implementation as the `LMT-Loop`. First, the convex hull edges are implicitly given during initialization of their half-edge structure and can be marked as such without any additional cost. Determining the convex hull edges beforehand allows to remove the case distinction inside the `LMT-Loop`, i.e., it removes all intersection tests that are applied to impossible edges. Secondly, sorting the stack by edge length destroys spatial ordering and the loss of locality of reference outweighs all gains on modern hardware. Without sorting, it is actually not necessary to push all edges onto the stack upfront. Lastly, with proper partitioning of the edges, the `LMT-Loop` can be executed in parallel – described in more detail in Section 2.4.

Additionally, we incorporated an improvement to the LMT-skeleton suggested by Aichholzer et al. [1]. Because the improved LMT-skeleton is computationally much more expensive, we apply it only to edges surviving an initial round of the normal LMT-heuristic.

n	Edges	Number of visited neighbors per point: Mean	SD	Min	Max	$\mathcal{DS} = 2\pi$
10^1	36.16 ±2.63	9 ±0	0 ±0	9 ±0	9 ±0	0 ±0
10^2	882.8 ±27.69	55.6 ±3.1	16.6 ±2.04	23.72 ±4.82	98.56 ±1.27	30.4 ±4.61
10^3	10,731.7 ±159.9	72.52 ±1.56	23.16 ±1.3	22.68 ±4.55	173 ±14.61	737.84 ±10.91
10^4	$1.1316 \cdot 10^5$ ±471.24	77.64 ±0.69	26.64 ±0.73	19.08 ±2.3	363.72 ±20	9,126.08 ±18.74
10^5	$1.15 \cdot 10^6$ ±1,538.64	72.84 ±0.29	23.76 ±0.47	15.96 ±1.61	846.24 ±24.4	97,200.9 ±40.29
10^6	$1.1562 \cdot 10^7$ ±4,737.67	74 ±0.51	25.76 ±0.39	13.28 ±1.31	2,884.96 ±38.53	$9.9117 \cdot 10^5$ ±61.86
10^7	$1.1579 \cdot 10^8$ ±19,254	77 ±0.6	27.24 ±0.79	11.88 ±0.99	9,567.52 ±78.84	$9.9721 \cdot 10^6$ ±100.61
10^8	$1.1585 \cdot 10^9$ ±56,063.1	72 ±0.94	24.08 ±0.69	10.6 ±0.49	25,017.8 ±107.4	$9.9911 \cdot 10^7$ ±239.64

Table 1 Diamond test on uniformly distributed points. The table shows statistics for 25 different instances. The extreme values are assumed by points at the point set boundary.

2.4 Parallelization

Because the LMT-heuristic performs only local changes, most edges can be processed in parallel without synchronization. Problems occur only if adjacent edges are processed concurrently (for the improved LMT-skeleton this is unfortunately not true, because marking an edge *impossible* affects a larger neighborhood of edges). To parallelize the normal LMT-heuristic, we implemented a solution based on data partitioning without explicit locking.

We recursively cut the vertices V into two disjoint sets $V = V_1 \cup V_2$ and process only those edges with both endpoints in V_1 (resp. V_2) in parallel. Define X as the cut set $\{\{s,t\} \in E \mid s \in V_1, t \in V_2\}$, i.e., all edges with one endpoint in V_1 and the other in V_2. While edges in $E(V_1)$ resp. $E(V_2)$ are processed in parallel by two threads, edges in X are accessed read-only by both threads and are handled after both threads join. This way we never process two edges with a common endpoint in parallel. To avoid a serial scan at the top, we push the actual work of computing X down to the leaves in the recursion tree. Scanning of the half-edge array starts at the leave nodes: processing of half-edges that belong to some cut set is postponed, instead they are passed back to the parent node. The parent in turn scans the edges it got from its two children, processes all edges it can and passes up the remaining ones. In other words, the final cut set X bubbles up in the tree, while all intermediate cuts are never explicitly computed. This way, partitioning on each level of the recursion tree only takes constant time, while the actual work is fully parallelized at the leaf level. After the LMT-heuristic completes, we are left with many polygonal faces that still need to be triangulated. Our implementation traverses the graph formed by the edges with one producer thread in order to collect all faces and multiple consumer threads to triangulate them with dynamic programming.

3 Computational Results

Computations were performed on a machine with an Intel i7-6700K quad-core and 64GB memory. The code was written in C++ and compiled with gcc 5.4.0.

3.1 Uniformly and Normally Distributed Point Sets

Table 1 shows results of our diamond test implementation on uniformly distributed point sets with sizes ranging from 10 to 10^8 points. The table shows the mean values and the standard deviation of 25 different instances. Each instance was generated by choosing n points uniformly from a square centered at the origin. The diamond test performs one incremental nearest neighbor query for each point in order to generate the edges that pass the test. The last column shows the number of queries where all nodes in the spatial tree were discarded because dead sectors covered the whole search space. The numbers show that this is the regular case; the exceptional cases occur at points near the point set "boundary".

n	Possible edges after Diamond		Possible edges after LMT		Possible edges after LMT+		Certain edges after LMT		Certain edges after LMT+		Simple Polygons	
$1 \cdot 10^1$	36.76	±2.78	3.8	±3.84	3.72	±3.62	19.32	±2.22	19.32	±2.22	0.68	±0.61
$1 \cdot 10^2$	871.92	±46.37	84.04	±20.14	74.56	±18.1	251.48	±7.12	252.28	±7.12	10.52	±2.55
$1 \cdot 10^3$	10,687.4	±146.68	1,150.32	±98.05	1,031.96	±86.46	2,540	±32.33	2,548.04	±31.41	128	±9.2
$1 \cdot 10^4$	$1.1322 \cdot 10^5$	±661.16	12,637	±281.25	11,271.76	±251.6	25,193.44	±73.29	25,287.56	±76.43	1,367.08	±24.65
$1 \cdot 10^5$	$1.1503 \cdot 10^6$	±1,696.31	$1.2941 \cdot 10^5$	±1,198.41	$1.1523 \cdot 10^5$	±973.14	$2.5129 \cdot 10^5$	±322.29	$2.5227 \cdot 10^5$	±306.72	13,819.44	±67.93
$1 \cdot 10^6$	$1.1563 \cdot 10^7$	±5,459.02	$1.3044 \cdot 10^6$	±2,708.78	$1.1617 \cdot 10^6$	±2,486.36	$2.5098 \cdot 10^6$	±847.61	$2.5194 \cdot 10^6$	±860.53	$1.3904 \cdot 10^5$	±232.43
$1 \cdot 10^7$	$1.1579 \cdot 10^8$	±17,587.01	$1.3074 \cdot 10^7$	±11,021.75	$1.1645 \cdot 10^7$	±8,825.57	$2.5088 \cdot 10^7$	±2,774.11	$2.5184 \cdot 10^7$	±2,727.23	$1.3931 \cdot 10^6$	±607.95
$3 \cdot 10^7$	$3.4747 \cdot 10^8$	±28,678.6	$3.9239 \cdot 10^7$	±18,919.14	$3.4949 \cdot 10^7$	±15,068.66	$7.5258 \cdot 10^7$	±4,637.8	$7.5547 \cdot 10^7$	±4,563.03	$4.1797 \cdot 10^6$	±969.6

Table 2 LMT-skeleton statistics on uniformly distributed point sets.

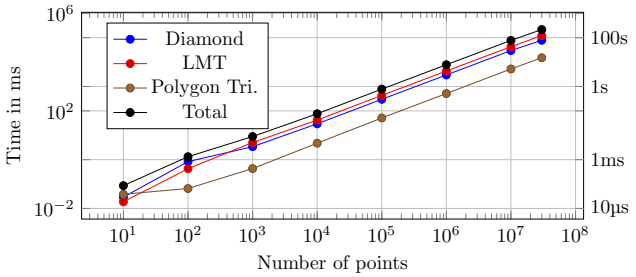


Figure 2 LMT-skeleton runtime on uniformly distributed point sets.

Table 2 shows statistics for the LMT-heuristic on uniformly distributed point sets. The instance sizes range from 10 points up to 30,000,000 points. For each size 25 different instances were generated. For the largest instances, the array storing the half-edges consumes nearly 39 GB of memory on its own. The serial initialization of the half-edge data structure, which basically amounts to radially sorting edges, takes longer than the parallel `LMT-Loop` on uniformly and normally distributed points. The improved LMT-skeleton by Aichholzer et al. is denoted LMT+ in the tables. The resulting skeleton was almost always connected in the computations and the number of remaining simple polygons that needed to be triangulated is shown in the last column. Only one instance of size $3 \cdot 10^7$ contained one small disconnected polygon. As we can see, the LMT-skeleton eliminates most of the possible edges with only $\approx 11\%$ remaining. The certain edges amount to $\approx 83\%$ of the complete triangulation. The improved LMT-skeleton reduces the amount of possible edges by another 10%, but it provides hardly any additional certain edges.

The results on normally distributed point sets are basically identical. Point coordinates were generated by two normally distributed random variables $X, Y \sim \mathcal{N}(\mu, \sigma^2)$, with mean $\mu = 0$ and standard deviation $\sigma \in \{1, 100, 100000\}$. The tables are given in the full version.

3.2 TSPLIB + VLSI

In addition to uniformly and normally distributed instances, we ran our implementation on instances found in the well-known TSPLIB [12], which contains a wide variety of instances with different distributions. The instances are drawn from industrial applications and from geographic problems. All 94 instances have a connected LMT-skeleton and can be fully triangulated with dynamic programming to obtain the minimum-weight triangulation. The total time it took to solve all instances of the TSPLIB was approximately 8.5 seconds.

Additional point sets can be downloaded at `http://www.math.uwaterloo.ca/tsp/vlsi/`. This collection of 102 TSP instances was provided by Andre Rohe, based on VLSI data sets studied at the Universität Bonn. The LMT-heuristic is sufficient to solve all instances, except `lra498378`, which contains two disconnected polygonal faces. Our implementation of the improved LMT-skeleton performs exceedingly bad on some of these instances; see Table 3. These instances contain empty regions with many points on the "boundary". Such regions are the worst-case for the heuristics because most edges inside them have the diamond property, which in turn leads to vertices with very high degree.

Table 3 VLSI instances with long runtime.

	Time in ms					
Instance	Total	DT	LMT-Init	LMT-Loop	LMT+	Dyn. Prog.
ara238025	15,325	4,954	446	496	9,279	148
lra498378	382,932	44,267	1,238	7,532	329,292	599
lrb744710	484,430	7,952	1,377	2,661	471,564	872
sra104815	1,937	559	191	198	922	65

4 Conclusion

We have shown that despite of the theoretical hardness of the MWT problem, a wide range of large-scale instances can be solved to optimality.

Difficulties for other instances arise from two sources. (1) Instances with almost regular k-gons with one or more points near the center can lead to highly disconnected LMT-skeletons (see Belleville et al. [3]) and require exponential time algorithms to complete the MWT. Preliminary experiments suggest that such configurations are best solved with integer programming. The instance by Belleville et al. can be solved with CPLEX in less than a minute, while the dynamic programming implementation of Grantson et al. [8] cannot solve it within several hours. (2) Instances containing empty regions with many points on their "boundary", such as empty k-gons and circles may be solvable in polynomial time, but trigger the worst-case behavior of the heuristics. Dealing with both is left for future work.

Acknowledgments. I want to thank Sándor Fekete and Victor Alvarez for useful discussions and suggestions that helped to improve the presentation of this paper.

References

1 O. Aichholzer, F. Aurenhammer, and R. Hainz. New Results on MWT Subgraphs. *Inf. Process. Lett.*, 69(5):215–219, 1999.
2 R. Beirouti and J. Snoeyink. Implementations of the LMT Heuristic for Minimum Weight Triangulation. In *Proc. Symp. Comp. Geom. (SoCG)*, pages 96–105, 1998.
3 P. Belleville, J. M. Keil, M. McAllister, and J. Snoeyink. On Computing Edges That Are In All Minimum-Weight Triangulations. In *Proc. Symp. Comp. Geom.*, pages 507–508, 1996.
4 G. Das and D. Joseph. Which triangulations approximate the complete graph? In *Optimal Algorithms: Int. Symp. Varna, Bulgaria, May 29–June 2, 1989 Proc.*, pages 168–192. 1989.
5 M. Dickerson, J. M. Keil, and M. H. Montague. A Large Subgraph of the Minimum Weight Triangulation. *Discrete & Computational Geometry*, 18(3):289–304, 1997.
6 M. Dickerson and M. H. Montague. A (Usually?) Connected Subgraph of the Minimum Weight Triangulation. In *Proc. Symp. Comp. Geom. (SoCG)*, pages 204–213, 1996.
7 R. L. S. Drysdale, S. A. McElfresh, and J. Snoeyink. On exclusion regions for optimal triangulations. *Discrete Applied Mathematics*, 109(1-2):49–65, 2001.
8 M. Grantson, C. Borgelt, and C. Levcopoulos. Fixed Parameter Algorithms for the Minimum Weight Triangulation Problem. *Int. J. Comp. Geom. Appl.*, 18(3):185–220, 2008.
9 A. Haas. Solving Large-Scale Minimum-Weight Triangulation Instances to Provable Optimality. 2018. http://arxiv.org/abs/1802.06415.
10 G. R. Hjaltason and H. Samet. Ranking in Spatial Databases. In *Proc. 4th Int. Symp. Adv. Spatial Datab. (SSD)*, pages 83–95, London, UK, 1995.
11 W. Mulzer and G. Rote. Minimum-weight Triangulation is NP-hard. *J. ACM*, 55(2):11:1–11:29, May 2008.
12 G. Reinelt. TSPLIB- a Traveling Salesman Problem Library. *ORSA Journal of Computing*, 3(4):376–384, 1991.