Minimum-Weight Triangulation is NP-hard

WOLFGANG MULZER
Princeton University
and
GÜNTER ROTE
Freie Universität Berlin

A triangulation of a planar point set S is a maximal plane straight-line graph with vertex set S. In the *minimum-weight triangulation* (MWT) problem, we are looking for a triangulation of a given point set that minimizes the sum of the edge lengths. We prove that the decision version of this problem is NP-hard, using a reduction from PLANAR 1-IN-3-SAT. The correct working of the gadgets is established with computer assistance, using dynamic programming on polygonal faces, as well as the β-skeleton heuristic to certify that certain edges belong to the minimum-weight triangulation.

Categories and Subject Descriptors: F.2.2 [**Nonnumerical Algorithms and Problems**]: Geometrical problems and computations; G.2.2 [**Graph Theory**]: Graph algorithms

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Optimal triangulations, PLANAR 1-IN-3-SAT

1. INTRODUCTION

Given a set S of points in the Euclidean plane, a *triangulation* T of S is a maximal plane straight-line graph with vertex set S. The *weight* of T is defined as the total Euclidean length of all edges in T. A triangulation that achieves the minimum weight is called a *minimum-weight triangulation* (MWT) of S.

The problem of computing a triangulation for a given planar point set arises naturally in many applications such as stock cutting, finite element analysis, terrain modeling, and numerical approximation. The *minimum-weight* triangulation has attracted the attention of many researchers, mainly due to its natural definition of optimality, and not so much because it would be important for the mentioned applications. We show that computing a minimum-weight triangulation is NP-hard. Note that it is not known whether the MWT problem is in NP, because it is an open problem whether sums of radicals (namely, Euclidean distances) can be compared in polynomial time [Blömer 1991]. This problem is common to many geometric optimization problems, like, perhaps most famously, the Euclidean Traveling Salesperson Problem. To get a variant of the problem that is in NP, one can take the weight of an edge e as the rounded value $\lceil \|e\|_2 \rceil$. Our proof also shows that this variant is NP-complete.

Our proof uses a polynomial time reduction from POSITIVE PLANAR 1-IN-3-SAT, a variant of the well-known PLANAR 3-SAT problem, which is a standard tool for showing NP-hardness of geometric problems.

1.1 Optimal Triangulations

Usually, a planar point set has (exponentially) many different triangulations, and many applications call for triangulations with certain good properties (see Figure 1). Optimal triangulations under various optimality criteria were extensively surveyed

Technical Report B05-23 (revised), arXiv:cs/0601002v3, February 2008, 45 pages.

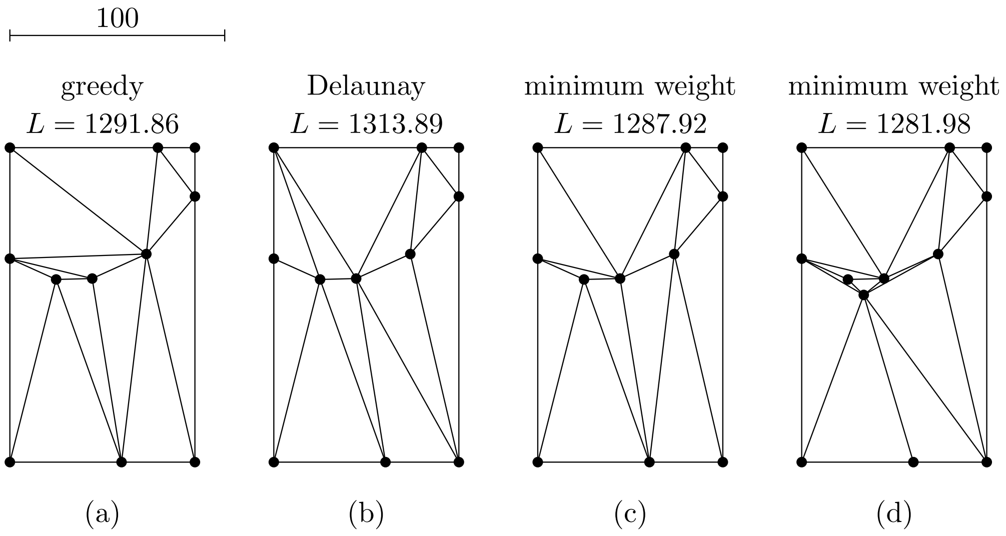


Fig. 1. A planar point set and different ways to triangulate it. The greedy triangulation (a) is constructed incrementally, always adding the shortest possible edge. In this example, it is shorter than the Delaunay triangulation (b), which avoids skinny triangles. A minimum weight triangulation for the point set is shown in (c). The length L of the MWT can decrease when additional Steiner points are allowed (d).

by Bern and Eppstein [1992].

The *Delaunay triangulation* is perhaps the best known triangulation. It arises as the dual of the Voronoi diagram and can be computed optimally in $O(n \log n)$ time, using classical techniques [de Berg et al. 2000]. Since it simultaneously optimizes many objective functions (it maximizes the smallest angle, it minimizes the maximum circumcircle, as well as the maximum smallest enclosing circle of all triangles), it is often the triangulation of choice.

Certain other criteria, such as the minimum maximum angle, the maximum minimum height, or the minimum maximum distance of a triangle from its circumcenter, can be optimized in polynomial time using the *edge insertion* technique [Bern et al. 1993; Edelsbrunner et al. 1992], a sophisticated application of dynamic programming.

It makes a difference whether it is allowed to add new points, so-called *Steiner points*, to the planar point set. As Figure 1 shows, this can sometimes help to improve the objective function. Bern and Eppstein [1992] give many similar examples. Here we only mention one result by Eppstein [1994], who showed that the minimum weight *Steiner* triangulation can be approximated up to a constant factor in $O(n \log n)$ time, using quadtrees.

1.2 History of the MWT problem

The minimum-weight triangulation problem dates back to the 1970s and has been called "perhaps the most longstanding open problem in computational geometry" [Bern and Eppstein 1992], At the end of the classical book of Garey and Johnson [1979] on NP-completeness, there is a list of 12 major problems whose complexity status was open at the time of writing. With the present paper, ten problems from this list have been resolved by proving NP-hardness or by exhibiting a polynomial-time algorithm (see [Johnson 2005] for a recent status update on the list). As of now, only two problems from the original list remain open, namely Precedence-Constrained 3-Processor Scheduling and the notorious Graph Isomorphism problem.

Early attempts. It seems that the MWT problem was first considered by Düppe and Gottschalk [1970] who proposed a greedy algorithm which always adds the shortest possible edge to the triangulation. Later, Shamos and Hoey [1975] suggested using the Delaunay triangulation as a minimum-weight triangulation. Lloyd [1977] provided examples which show that both proposed algorithms usually do not compute the MWT (Figure 1). He also shows that it is NP-complete to decide whether the edge set of a given planar straight-line graph (with crossing edges) contains a triangulation. After this, countless researchers attacked the MWT problem from many different angles. In one line of attack, researchers used classical optimization techniques such as dynamic programming or branch and bound, but this soon became infeasible. In other lines of research, people looked at relaxed variants of the problem: Maybe there exist reasonable restrictions of the problem for which efficient algorithms can be found, and if the problem cannot be solved exactly, maybe good approximations can be computed efficiently. Finally, there was an attempt to gain a better understanding of the geometric properties of the MWT in order to find footholds for effective heuristics. We now describe these approaches in more detail.

Dynamic Programming. Gilbert [1979] and Klincsek [1980] independently showed how to compute a minimum-weight triangulation of a simple polygon in $O(n^3)$ time by dynamic programming. In fact, this problem has become one of the standard textbook examples (or exercises) for illustrating the dynamic programming paradigm. There have also been attempts to attack the general problem with dynamic programming techniques. For example, Cheng et al. [1995] used dynamic programming in order to compute a minimum-weight triangulation of a given point set S in $O(n^{k+2})$ time if a subgraph of a MWT of S with k connected components is known. Using branch and cut, Kyoda et al. [1997] managed to compute MWTs of 100 points, but for large point sets mere dynamic programming becomes absolutely infeasible.

Restricted Instances. For restricted classes of point sets, it is possible to compute the MWT in polynomial time. For example, Anagnostou and Corneil [1993] gave an algorithm to compute the MWT of the vertex set of k nested convex polygons in $O(n^{3k+1})$ time. More recently, Hoffmann and Okamoto [2006] showed how to obtain the MWT of a point set with k inner points in $O(6^k n^5 \log n)$ time.

Approximations. In another line of attack, researchers were looking for triangulations that approximate the MWT. The Delaunay triangulation is not a good candidate, since it may be longer by a factor of $\Omega(n)$ [Kirkpatrick 1980; Manacher and Zobrist 1979]. The greedy triangulation approximates the MWT by a factor of $\Theta(\sqrt{n})$ [Manacher and Zobrist 1979; Levcopoulos 1987; Levcopoulos and Krznaric 1996]. Plaisted and Hong [1987] showed how to approximate the MWT up to a factor of $O(\log n)$ in $O(n^2 \log n)$ time. Levcopoulos and Krznaric [1996] introduced quasi-greedy triangulations, which approximate the MWT within a constant factor. Remy and Steger [2006] discovered an approximation scheme for MWT that runs in quasi-polynomial time: for every fixed ε, it finds a $(1+\varepsilon)$-approximation in $n^{O(\log^8 n)}$ time.

Subgraphs and supergraphs. A different line of research tried to identify criteria to include or exclude certain edges of the MWT. Gilbert [1979] showed that the MWT always contains the shortest edge. Yang et al. [1994] extended this result by proving that edges which join mutual nearest neighbors are in the MWT. A larger subgraph of the MWT, the *β-skeleton*, was discovered by Keil [1994]. We describe it in Section 4.1. In practice, the β-skeleton has many connected components, and thus does not help much in computing a MWT. Often, the *LMT-skeleton heuristic* described by Dickerson et al. [1997] yields much better results. It uses the simple fact that a MWT is locally optimal in the sense that it cannot be improved by flipping the diagonals of a convex empty quadrilateral in the point set. The LMT-skeleton made it feasible to compute the MWT for larger, well-behaved point sets, and it has been the subject of numerous further investigations [Beirouti and Snoeyink 1998; Cheng et al. 1996; Aichholzer et al. 1999; Belleville et al. 1996; Bose et al. 2002].

Approaching the problem from the other direction, Das and Joseph [1989] defined the diamond test, which yields a supergraph of the MWT: An edge e can only be contained in the MWT if at least one of the two isosceles triangles with base e and base angles $\pi/8$ is empty. This constant was improved to $\pi/4.6$ [Drysdale et al. 2001] (see Figure 9 below). The diamond test gives an easy criterion to exclude impossible edges from the MWT. Usually, this eliminates all edges except a set of $O(n)$ remaining candidate edges. (This statement is true for random point sets, with high probability. With bucketing techniques, such a set of $O(n)$ edges can be found in linear expected time [Drysdale et al. 1995].)

1.3 Our Methods and Results

1.3.1 *Historical Perspective.* The crucial necessary condition for a geometric optimization problem to be NP-hard is that the solution depends non-locally on the data.

With the LMT-skeleton heuristic, it became feasible to compute minimum-weight triangulations fast enough that one could carry out experiments and play with various point sets. An instance of a non-local effect was hence discovered by Jack Snoeyink [Beirouti and Snoeyink 1998]: the so-called *wire* is a symmetric polygon that does not have a symmetric minimum-weight triangulation, and hence it has (at least) two different minimum-weight triangulations, see Figure 2.

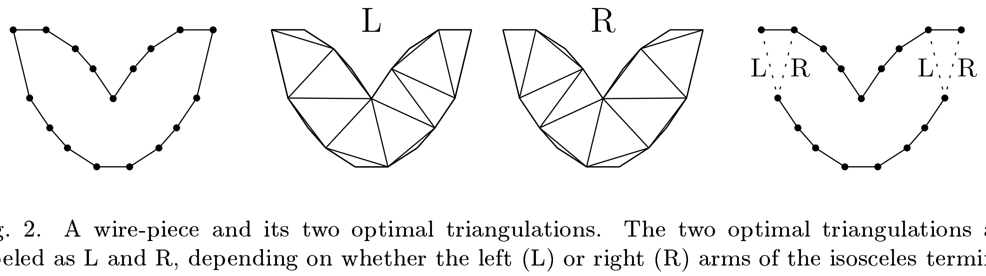


Fig. 2. A wire-piece and its two optimal triangulations. The two optimal triangulations are labeled as L and R, depending on whether the left (L) or right (R) arms of the isosceles terminal triangles, as shown dotted in the right part, belong to the triangulation. This convention is used throughout the paper.

L

R

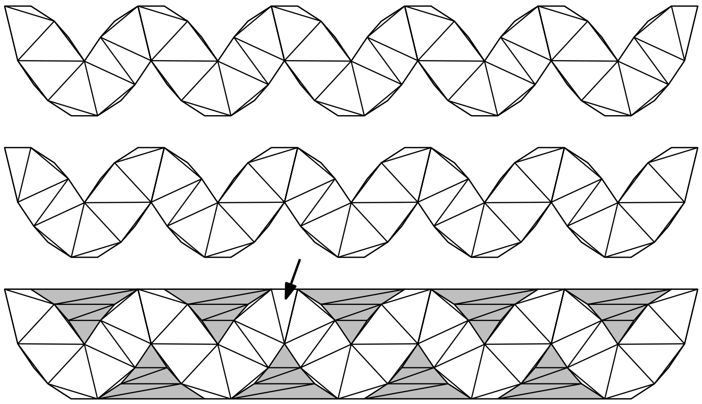


Fig. 3. A longer wire with several optimal triangulations. Besides the two triangulations of type L and R, there are mixed forms where the transition between L and R can occur in intermediate positions, as marked by an arrow in the lower example. If the wire is regarded as a point set, the shaded pockets outside the wire also have to be triangulated. For each pocket, there are four optimal triangulations. However, the pockets are independent of each other, and do not affect the properties of the wire.

Wires can be extended to any length, see Figure 3. *A small change in the point set at one end of the wire will cause the optimum triangulation to topple globally throughout the whole wire.* Wires can thus be used to "transmit information" from one area of the plane to a remote area (hence the name "wire"). We use various forms of wires as building blocks for constructing gadgets in our NP-hardness reduction.

Such a non-local effect is usually a strong indication that the problem is NP-hard. Still, it took almost a decade until another crucial building block was designed: a gadget that allows several wires to meet and that carries out "logic" with the information that is transmitted by them (the so-called C-connection, see Figure 14). The design of this gadget heavily depended on computer assistance. We used an implementation of the LMT-skeleton heuristic using Otfried Cheong's program IPE [Schwarzkopf 1995] as a graphical user-interface.

1.3.2 *Computer Assistance.* It seems inevitable that a NP-hardness proof for the MWT problem requires *some* amount of computer calculation, since computing the weight of a triangulation involves Euclidean distances, and thus square roots. Therefore, when comparing two triangulations that differ in more than a pair of edges, it is hard to compare their weight by just looking at them.

However, since the conference version of this paper [Mulzer and Rote 2006], we have made an effort to reduce the part of the proof that depends on computer assistance for verification. We have also made other simplifications of the proof and made it more accessible. (a) We have tried to simplify and specialize the problem as much as possible already at the logical (discrete) level. In particular, the PLANAR 1-IN-3-SAT problem is more specialized than in [Mulzer and Rote 2006] and does not need negations. (b) We designed the gadgets in such a way that the β-skeleton edges already form large connected components. This part of the construction is trivial to check by computer, and is open even to visual inspection.

(c) In the proof, the gadgets split into modules ("pieces") that can be analyzed separately. (d) The analysis of each module boils down to checking a small number of possibilities, each amounting to computing the minimum-weight triangulation of a simple polygon. This can be programmed from scratch in less than an hour's work by an experienced programmer. (e) To make the computer-assisted part even safer, we have rerun the final calculations with interval arithmetic, using the open-source high-level programming language PYTHON, which automatically supports integer arithmetic of unbounded length.

The final computer runs to verify our gadgets took about 10 hours on a relatively slow computer. The code is given in Appendix B and amounts to approximately 1000 lines.

2. OVERVIEW OF THE REDUCTION

We begin with a high-level overview of the geometric construction. We build our point set from a small number of tube-like *pieces* that fit together at the openings, where they share *terminal triangles*, see Figure 4. The boundary edges of the pieces must belong to every optimal triangulation, regardless of how we put the pieces together (Proposition 6.4 below). If we draw these edges we get a system of tubes, separated by polygonal holes. The holes can be triangulated in polynomial time by dynamic programming; The tubes will form the essential part of the gadgets that model the logical structure of a POSITIVE PLANAR 1-IN-3-SAT formula, whose satisfiability is NP-complete to decide, see Section 3.

For the analysis, we will show that at each terminal triangle, we have a choice of two edges to insert (Lemma 5.1). When one of the edges is inserted for every terminal triangle, the edges enclose a simple polygon, whose optimum solution can be computed by dynamic programming. Every piece can therefore be analyzed in isolation, by considering a small number of possibilities.

3. POSITIVE PLANAR 1-IN-3-SAT

In this section, we describe the POSITIVE PLANAR 1-IN-3-SAT problem, which we will use for our reduction.

Definition 3.1. Let Φ be a Boolean formula in 3-CNF. The *associated graph* of Φ, $G(\Phi)$, has one vertex v_x for each variable x in Φ and one vertex v_C for each clause C in Φ. There is an edge between a variable-vertex v_x and a clause-vertex v_C if and only if x or $\neg x$ appears in C.

The Boolean formula Φ is called *planar* if its associated graph $G(\Phi)$ is planar.

Lichtenstein [1982] showed that 3-SAT remains NP-complete if the input is restricted to a planar formula (the PLANAR 3-SAT problem). As Knuth and Raghunathan [1992] observed, Lichtenstein's proof implies that it suffices to consider formulae whose associated graph can be embedded such that the variables are arranged on a straight line, with three-legged clauses above and below them. The edges between the variables and the clauses are embedded in a rectilinear fashion (see Figure 5).

In our reduction we will use a variant of PLANAR 3-SAT in which there are only positive variables and we ask for an assignment to the variables such that in each clause exactly one variable is set to true.

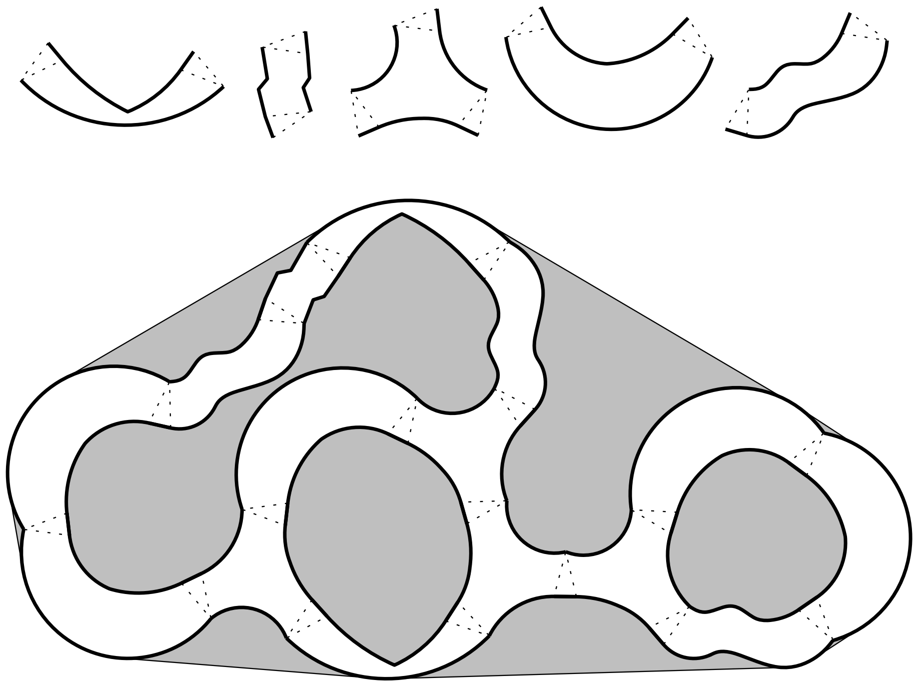


Fig. 4. A schematic view of the construction: From the five types of building blocks shown at the top, one can build a network of tubes like in the lower part. The area within this network is the interesting part where the simulation of the logic takes place. The *holes* inside the structure and the *pockets* between the structure and the convex hull (drawn shaded) form simple polygons that can be optimally triangulated in polynomial time.

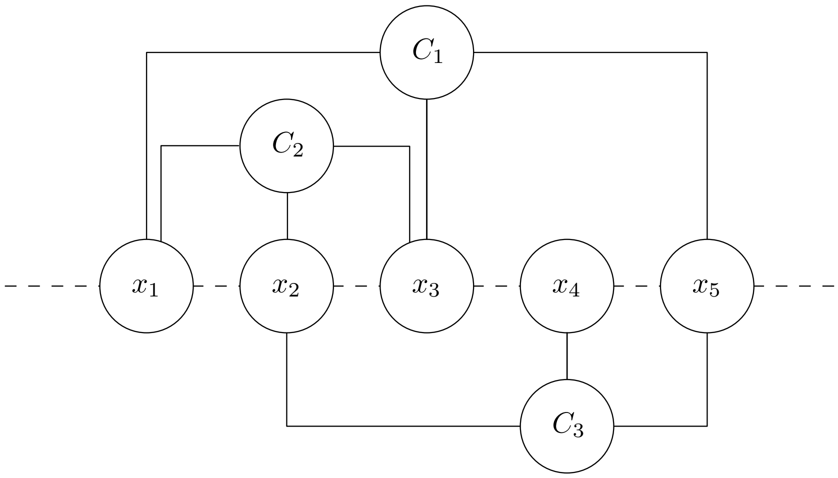


Fig. 5. A rectilinear embedding of graph that is associated with the Boolean formula $(x_1 \vee \neg x_3 \vee x_5) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_4 \vee \neg x_5)$.

Definition 3.2. In the *POSITIVE PLANAR 1-IN-3-SAT* problem, we are given a collection Φ of clauses containing exactly three variables together with a planar embedding of the associated graph $G(\Phi)$ as described above.

The problem is to decide whether there exists an assignment of truth values to the variables of Φ such that exactly one variable in each clause is true.

The POSITIVE 1-IN-3-SAT problem without the planarity restriction was shown to be NP-complete by Schaefer [1978]. This problem can also be interpreted as a SET PARTITIONING problem, where every element of the ground set has precisely three sets by which it can be covered. For completeness, we include a proof that the planar version of the problem is NP-complete.

PROPOSITION 3.3. *POSITIVE PLANAR 1-IN-3-SAT is NP-complete.*

PROOF. It is easy to verify in polynomial time that a given embedding of a formula is rectilinear (the validity of an instance) and that a given assignment has the 1-IN-3 property (the validity of a certificate). Hence, the problem is in NP. To show completeness, we describe a reduction from PLANAR 3-SAT. Let I be an instance of PLANAR 3-SAT, i.e. a 3-CNF formula Φ and a planar, rectilinear embedding of the associated graph. We describe how to transform I into an instance of POSITIVE PLANAR 1-IN-3-SAT while maintaining the rectilinear embedding. We consider the clauses of Φ one by one.

If clause C contains only one literal, it can easily be eliminated. If clause C contains two literals, say x and y, we can replace it by two three-variable clauses $(x \vee y \vee z)$ and $(x \vee y \vee \neg z)$, where z is a new variable.

Before we consider clauses with three variables, we first discuss two useful gadgets which enforce equality and inequality between variables in terms of 1-IN-3 clauses.

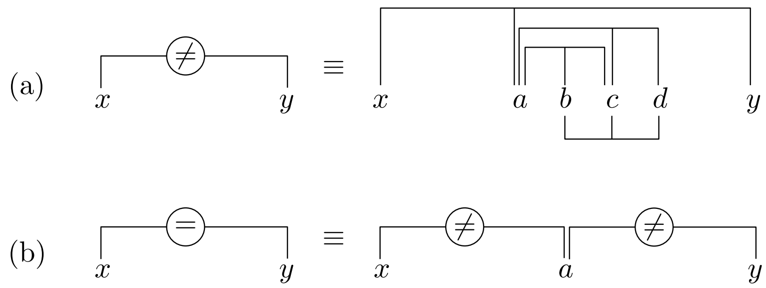


Fig. 6. The gadgets to enforce inequality (a) and equality (b) between two variables.

The inequality gadget. The inequality gadget serves to enforce the constraint $x \Leftrightarrow \neg y$ for two variables x and y. We abbreviate it as $(x \neq y)$. It is implemented as

$$x \neq y \equiv (x, a, y) \wedge (a, b, c) \wedge (a, c, d) \wedge (b, c, d), \tag{1}$$

where a, b, c, d are new variables used only inside the gadget (see Figure 6a). Here, as always, we denote a 1-IN-3 clause by a triple (v_1, v_2, v_3). The clause (v_1, v_2, v_3) is *satisfied* if exactly one of the variables v_1, v_2, v_3 is true.

LEMMA 3.4. *Given x and y, the expression* (1) *is satisfiable iff exactly one of x and y is true.*

PROOF. The last three clauses enforce that $a = 0$, because otherwise we get $b = c = d = 0$, and the last clause is not satisfied. The first clause now ensures that exactly one of x and y is true. □

The equality gadget. Using two copies of the inequality gadget and an extra variable a, we can build a gadget to enforce the constraint $x \Leftrightarrow y$ (see Figure 6b). We abbreviate it as $(x = y)$.

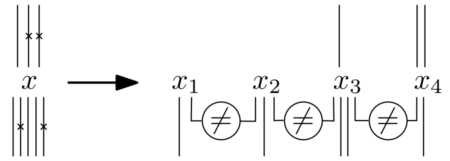


Fig. 7. Eliminating negations: The variable x in this example has eight connections, some of which are negated, as indicated by crosses. We replace x by a chain of variables $x_1, x_2, \ldots$ with alternating truth values and place the connections to the clauses containing x correspondingly.

Elimination of negated variables. To eliminate the negated variables in the original 3-SAT clauses, we replace a variable x by a chain of variables $x_1, x_2, \ldots$ and use the negation gadget to enforce the appropriate relations between them (see Figure 7).

Transformation of disjunctions. It remains to handle a disjunctive clause $C = (x \vee y \vee z)$ with three literals x, y, and z. C can be replaced by

$$(x, u, a) \wedge (y, u, b) \wedge (a, b, q) \wedge (u = c) \wedge (d \neq z) \wedge (c, d, r)\ . \tag{2}$$

See Figure 8. Note that there is space to accommodate the middle legs of the equality and inequality gadgets.

LEMMA 3.5. *Given x, y, and z, the expression* (2) *is satisfiable iff $x \vee y \vee z$ holds.*

PROOF. The clause (c, d, r) is equivalent to $\neg c \vee \neg d$, since the variable r appears nowhere else. Hence $c \to \neg d \iff u \to z$ by the third and fourth constraint. Thus, satisfiability of the last three constraints is equivalent to $u \to z$.

Now, if $u = 1$, the first three clauses can be satisfied only by $x = y = 0$, and we must have $z = 1$. The third clause is satisfied by setting $a = b = 0$ and $q = 1$. For $u = 0$, the first two clauses reduce to $x \neq a$ and $y \neq b$. Since q appears nowhere else, the clause (a, b, q) is equivalent to $\neg a \vee \neg b \iff x \vee y$. The value of z can be arbitrary in this case. □

To make sure that x and z remain reachable from other clauses, we also add two additional variables x' and z' and equality constraints. Clauses that were above the variables and nested between x and y can now connect to x' instead of x, and those nested between y and z can connect to z' instead of z.

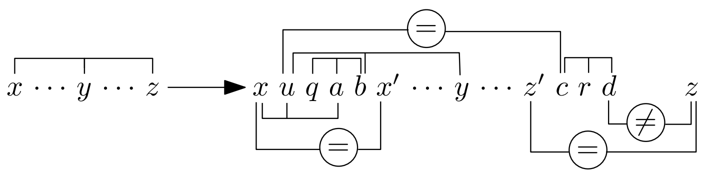


Fig. 8. The modification for a clause $x \vee y \vee z$ with three literals.

This transformation can be carried out in polynomial time, and by construction, the transformed positive formula has a rectilinear embedding, and it has a 1-IN-3 assignment if and only if the original formula is satisfiable.

4. GEOMETRIC VALIDATION TOOLS

We summarize the geometric and computational tools we need for establishing properties of the gadgets and their minimum-weight triangulations.

4.1 The β-Skeleton

The *β-skeleton* of a point set S is defined as the set of all edges pq between two points of S such that the two circles of diameter $\beta \cdot |pq|$ passing through p and q are empty, see Figure 9. It provides a sufficient condition for including edges in the minimum-weight triangulation:

THEOREM 4.1. *The β-skeleton is a subgraph of the MWT for*

$$\beta = \sqrt{1 + \sqrt{4/27}} \leq 1.17682.$$

□

Theorem 4.1 was first proven by Keil [1994] for $\beta \leq \sqrt{2}$ and was later improved by Cheng and Xu [1996]. The value for β is nearly optimal, since there is a lower bound of $\sqrt{5/4 + \sqrt{1/108}} \approx 1.16027$ [Wang and Yang 2001]. In Proposition 6.4, we will use the β-skeleton to identify the *boundary* edges of our gadgets, which must belong to every optimal triangulation. Since the β-skeleton is defined by a local condition, it is very easy to verify that a certain edge is a boundary edge. Our gadgets have many points, hence we used a computer to check the β-skeleton property of our boundary edges. The source-code of the straightforward program can be found in Appendix B.3.

4.2 Triangulating a Polygon by Dynamic Programming

As we explained above, the analysis of our reduction essentially boils down to computing the minimum-weight triangulation of a small number of simple polygons. Gilbert [1979] and Klincsek [1980] independently showed how to do this:

PROPOSITION 4.2. *The minimum-weight triangulation of a simple polygon can be computed in $O(n^3)$ time.* □

Proposition 4.2 is proved by a straight-forward application of dynamic programming, and we implemented a variant of the algorithm in PYTHON. (The source code of our implementation is given in Appendix B.2.) The dynamic programming

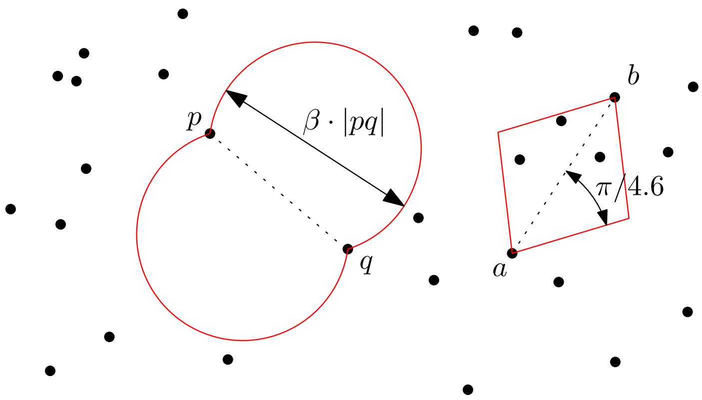


Fig. 9. An edge pq of the β-skeleton. The edge ab does not belong the MWT since the diamond test is violated.

algorithm assumes an idealized model in which arithmetic with edge lengths can be done exactly. This means that sums of radicals (i. e. Euclidean distances) need to be compared in constant time. However, it is not known how to do this even in polynomial time [Blömer 1991]. Therefore, our program uses interval arithmetic to compare sums of distances, and for all our gadgets it computes an unambiguous answer.

Our program identifies the edges in a minimum-weight triangulation T inside a given simple polygon P. We call the total weight of these edges the *internal cost* of P and denote it by $c(T)$. The internal costs computed by our program will be discussed extensively in Section 6.1. It is easy to modify the program so that it computes the optimum under the restriction that certain edges are forbidden. This is needed for the proof of Lemma 5.1.

5. TERMINAL TRIANGLES

The point set is constructed from small *elementary pieces* that fit together at *terminal triangles*. These terminal triangles are isosceles triangles xyz with base yz, that lie symmetrically with respect to a coordinate axis, and they come in two sizes (small and large). The coordinates of vertical terminal triangles are given in Table I. See Figure 10 or 16–17 for an illustration. These triangles can be rotated

	x	y	z	δ
small	(0.00, 0.00)	(−2.7 , 11.2)	(2.7 , 11.2)	$\delta_{\text{small}} =$ 5.655172
large	(0.00, 0.00)	(−11.61, 48.16)	(11.61, 48.16)	$\delta_{\text{large}} =$ 24.06

Table I. Terminal triangles. The "difference terms" δ will described below in Section 5.1.

by multiples of 90°, and translated by multiples of 0.01 in each coordinate. The large triangle is a scaled copy of the small triangle, by a factor of 4.3.

The basic pieces that we construct are point sets with two or three terminal triangles. The β-skeleton edges will form a simple polygon through these points, with a missing edge at each terminal triangle. We will show that in each terminal

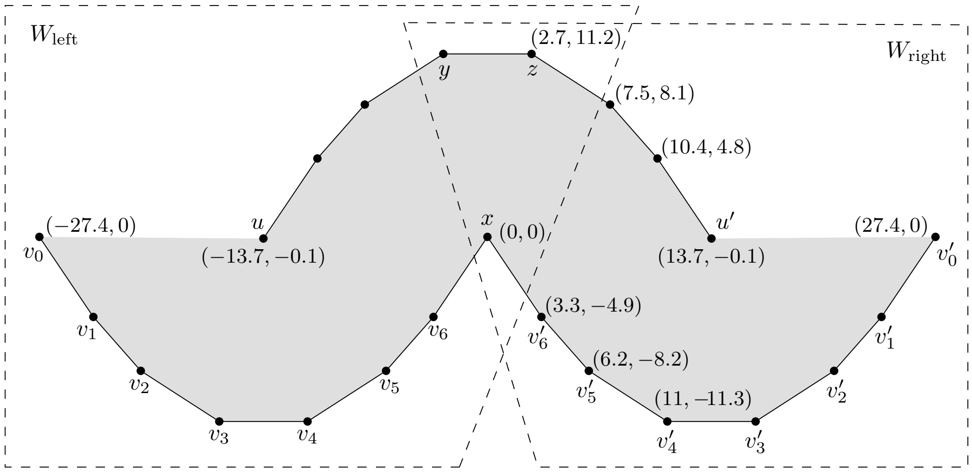


Fig. 10. The point set W in Lemma 5.1

triangle, the edge xy or the edge xz must be present in every minimum-weight triangulation (Lemma 5.1). Thus, in order to determine how a piece with k terminals can be triangulated in a minimum-weight triangulation, it suffices to consider 2^k possibilities for the positions of each terminal edge. For each possibility, we find the optimum solution by dynamic programming.

Let W be the point set given in Figure 10, which forms part of the wire mentioned in the introduction (see Section 1.3.1). It is symmetric about the y-axis. The points v_0, x, and v'_0 lie on the x-axis, and the points u and u' lie 0.1 units below this line. The left and right halves of the lower boundary are equal, and they are equal to the upper boundary (turned by 180°). We denote by W_{left} and W_{right} the left half and the right half of the set W, up to and including the points x, y, and z.

Lemma 5.1. *Let P be a point set that contains a (translated, scaled, rotated) copy of the point set W, but no other points in the shaded polygon of Figure 10. Any minimum-weight triangulation T of P that contains all edges shown in Figure 10 must contain at least one of the edges xy or xz.*

Proof. The proof distinguishes 21 cases and deals with each case by a small calculation.

Since in any triangulation, the point u must be incident to some edge that emanates into the lower half-plane, T must contain at least one of the edges uv_1, uv_2, $\dots$, uv_6. Similarly, T must contain at least one of the edges $u'v'_1$, $u'v'_2$, $\dots$, $u'v'_6$. Now, for each $i, j = 1, \dots, 6$, the edges uv_i and $u'v'_j$ enclose a simple polygon. For each polygon, we calculated, by dynamic programming, the optimum triangulation as well as the optimum triangulation that uses none of the edges xy and xz. By symmetry, we have to consider only $1 \le i \le j \le 6$. The results are shown in Table II. □

We remark that the lemma is somewhat robust against perturbations and remains true even if all points of W are moved by a small distance. Figure 11 shows that small perturbations do not make triangulations feasible or infeasible. The triangulations in Table II contain at most 18 internal edges. If each point is moved

v_1, v_1' 4.00304	[202.72577, 202.72594]	[198.72255, 198.72273]	[198.72255, 198.72273]			
v_2, v_2' 4.67187	[181.98535, 181.98550]	[177.31332, 177.31348]				
v_3, v_3' 4.00308	[159.65909, 159.65922]	[155.65587, 155.65601]	[155.65587, 155.65601]			
v_4, v_4' 4.67191	[137.62231, 137.62242]	[132.95028, 132.95040]				
v_5, v_5' 3.83677	[116.25971, 116.25980]	[112.42284, 112.42294]	[112.42284, 112.42294]			
v_6, v_6' 4.67195	[94.68883, 94.68890]	[90.01680, 90.01688]				
v_1, v_2' 4.50551	[192.35556, 192.35572]	[187.84988, 187.85005]				
v_1, v_3' 4.00306	[181.19243, 181.19258]	[177.18921, 177.18937]	[177.18921, 177.18937]			
v_1, v_4' 4.50553	[170.17404, 170.17418]	[165.66836, 165.66851]				
v_1, v_5' 4.00308	[159.49274, 159.49287]	[155.48952, 155.48966]		v_3, v_4' 4.50555	[148.64070, 148.64082]	[144.13502, 144.13515]
v_1, v_6' 4.50555	[148.70730, 148.70742]	[144.20162, 144.20175]		v_3, v_5' 4.00310	[137.95940, 137.95951]	[133.95618, 133.95630]
v_2, v_3' 4.50553	[170.82222, 170.82236]	[166.31654, 166.31669]		v_3, v_6' 4.50557	[127.17396, 127.17406]	[122.66828, 122.66839]
v_2, v_4' 4.67189	[159.80383, 159.80396]	[155.13180, 155.13194]		v_4, v_5' 4.50557	[126.94101, 126.94111]	[122.43533, 122.43544]
v_2, v_5' 4.50555	[149.12253, 149.12265]	[144.61685, 144.61698]		v_4, v_6' 4.67193	[116.15557, 116.15566]	[111.48354, 111.48364]
v_2, v_6' 4.67191	[138.33709, 138.33720]	[133.66506, 133.66518]		v_5, v_6' 4.50559	[105.47427, 105.47435]	[100.96859, 100.96868]

Table II. The optimum triangulations with and without the edges xy and xz, for each polygon bounded by edges uv_i and $u'v_j'$, as indicated in the left column. The second column shows an optimal triangulation without the edges xy and xz, together with the interval for its internal cost, calculated to 5 digits after the decimal point. (There are always at least two optimal triangulations, since the upper diagonal in the middle can be flipped. The program has picked one solution whose lower interval bound is smallest.) The remaining columns list all candidates for optimal solutions. One can check that all these solutions contain the edges xy and xz. (In fact, one can see that, if there are two optimal solutions in a row, they must have exactly the same weight since they contain parts which can be replaced by symmetric parts.) The lower bound on the difference between the optimum in the third column and the optimum in the second column is given in the first column. The smallest difference (3.83677) occurs for the case v_5, v_5'. The numbers in this table were computed with the Python program in Appendix B.4.

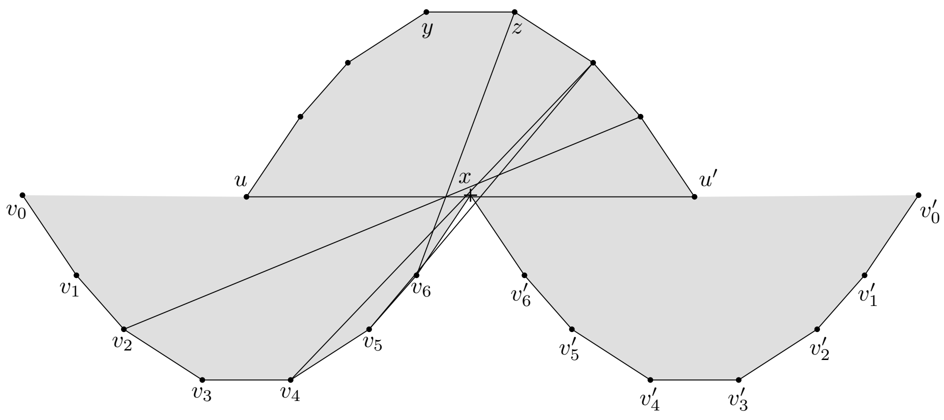


Fig. 11. The point set W in Lemma 5.1 is in sufficiently general position.

by at most $\sigma := 0.04$, the length of each edge changes by at most 2σ, and the weight of any triangulation changes by at most $36\sigma = 1.44$.

Since the difference between the optimal triangulation and the best triangulation without the edges xy and xz is bigger than $3.83677 > 2 \cdot 1.44$, the claim of the lemma is not invalidated if each point is moved by at most 0.04 in any direction. We will use this observation in Section 6, where we describe a version of the wire-piece in which the offset between the terminal triangles is slightly larger than in the standard wire-piece. This makes it possible to build wires of arbitrary length (see Lemma 6.1).

5.1 Putting pieces together

When putting pieces together, we make sure that Lemma 5.1 can be applied at the junction: at every terminal triangle, the piece contains a copy of half of the point set W (possibly rotated, and possibly scaled by 4.3). This allows us to split the input set into components at the terminal triangles. For each terminal triangle xyz, we need to consider just the two choices xy and yz for the *terminal edges.* We denote these choices by L and R, depending on whether we use the left or right arms, as viewed from the tip of the terminal triangle.

When determining the behavior of a piece with k terminals in a minimum-weight triangulation, we have to compare the optimal triangulations for the 2^k choices for the positions of the k terminal edges. However, these triangulations cover different areas and cannot be compared directly. Depending on whether the terminal edge xy or the terminal edge xz is chosen at some given boundary, the triangulated area excludes or includes the area of the terminal triangle itself, and the optimal triangulation can thus be expected to be cheaper or more expensive. To offset this effect, we define the *reduced weight* of a triangulation T, $\bar{c}(T)$, by adding a penalty or subtracting a bonus term from the actual weight. This will allow us to see directly which configurations of terminal edges are better than others, and which configurations cannot possibly be part of an optimal triangulation.

More precisely, we define a difference term δ that depends on the size of the triangle (large or small), see Table I. When comparing the costs of different triangulations of a piece or of different pieces, we only look at the *internal cost*, the

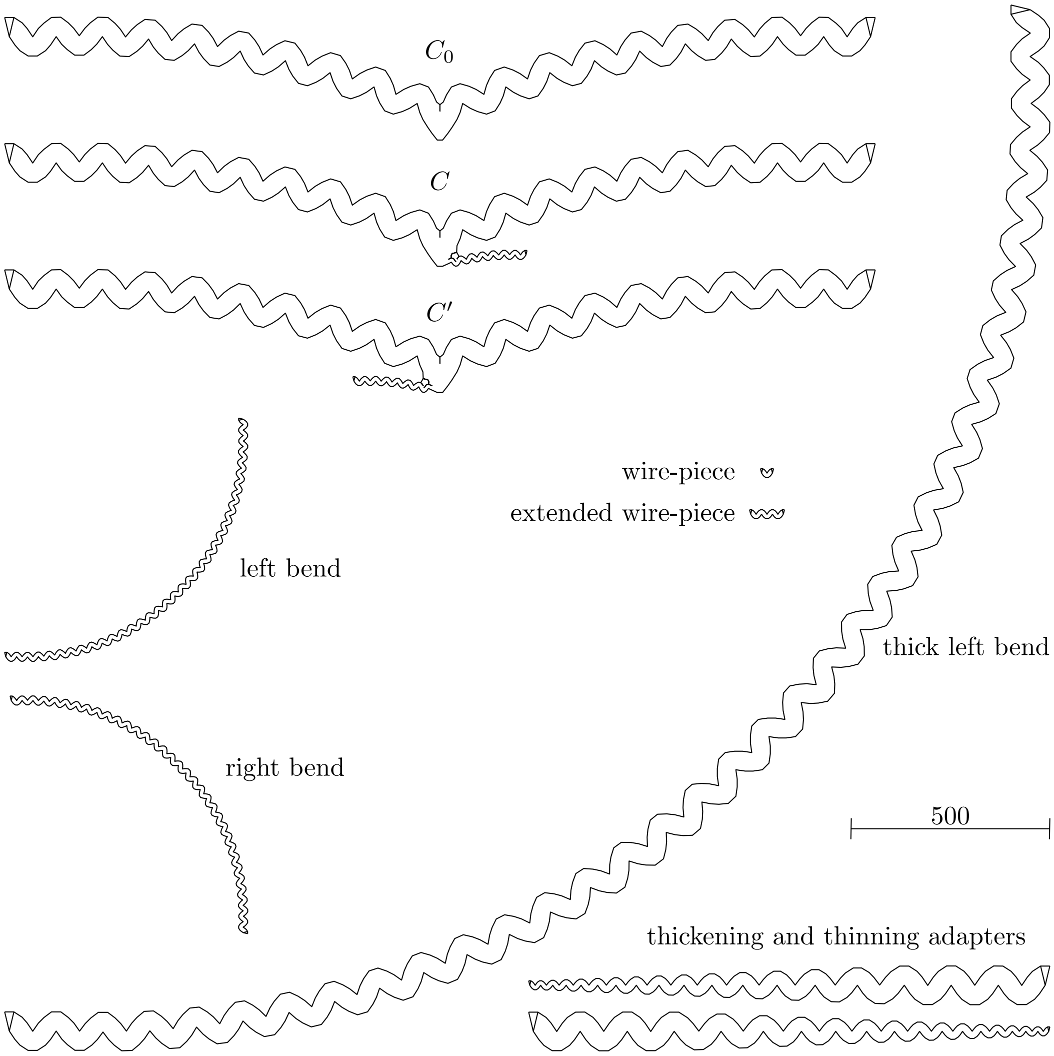


Fig. 12. All elementary pieces

cost of the internal edges, since the boundary edges are fixed. When the triangulated area of a piece includes the terminal triangle, we subtract δ from the actual weight; otherwise we add δ. The precise values of δ_{small} and δ_{large} are not important and have been chosen to make the proofs convenient. The following lemma holds irrespective of the values of δ_{small} and δ_{large}.

LEMMA 5.2. *Consider a layout of pieces S where every terminal triangle is shared by two pieces. For each terminal triangle, fix one of its two long sides. Consider the minimum-weight triangulation T of S that is restricted to contain those fixed edges as well as all piece boundaries. For each piece, calculate the reduced internal weight of the minimum-weight triangulation of the piece bounded by the given fixed edges.*

Then the overall reduced internal weight of all pieces differs from the total weight of the triangulation T by a constant that is independent of the choice of the fixed

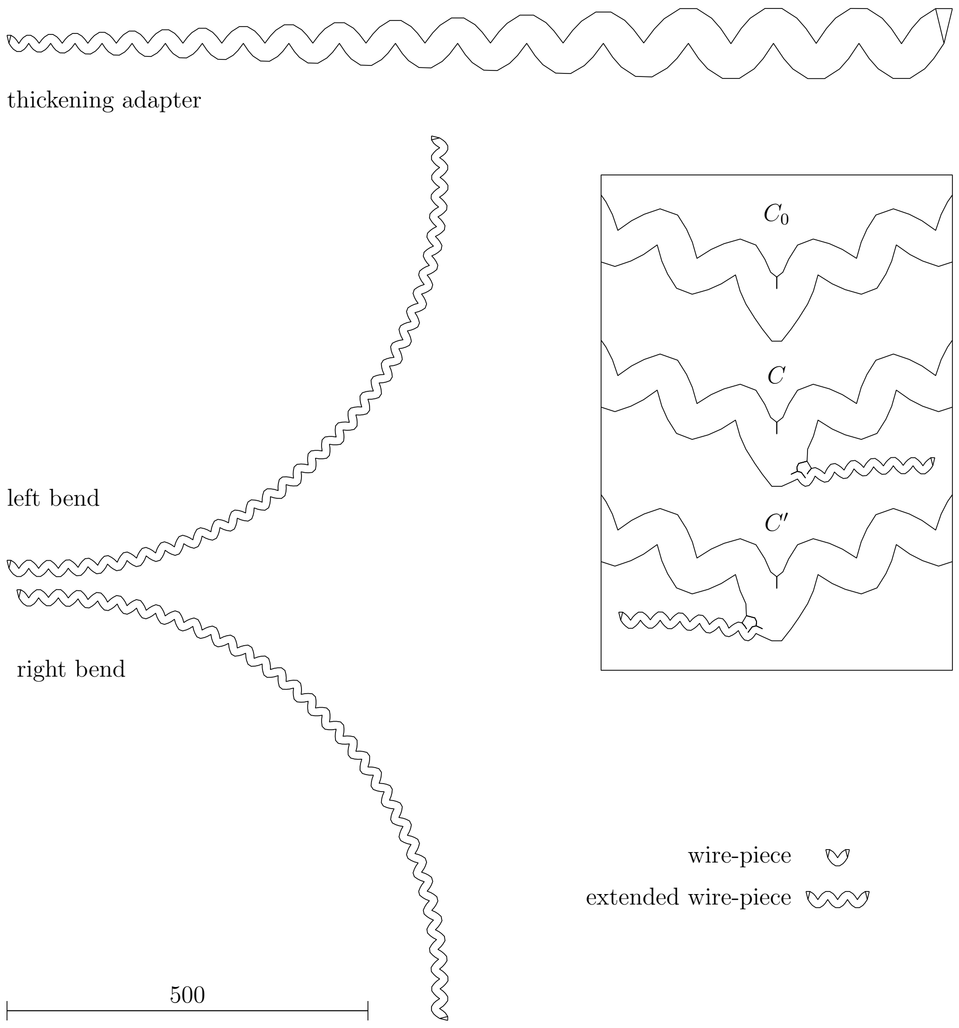


Fig. 13. Enlarged view of the small elementary pieces

edges.

Proof. In total, the effect of adding and subtracting δ cancels on the two sides of each terminal triangle. The cost of the terminal edge (xy or xz) itself is not accounted for in the sum of internal costs, but since xy and xz have the same length, this amounts to a constant difference. The boundaries are also fixed, and so is the cost of triangulating the pockets and the holes. □

We will see below (Proposition 6.4) that all boundary edges are part of the MWT, since they belong to the β-skeleton. We know from Lemma 5.1 that, by choosing a terminal edge from each terminal triangle in all possible ways, we are guaranteed to find the optimal triangulation. The consequence of Lemma 5.2 is that we need only look at the reduced (internal) cost when comparing these choices.

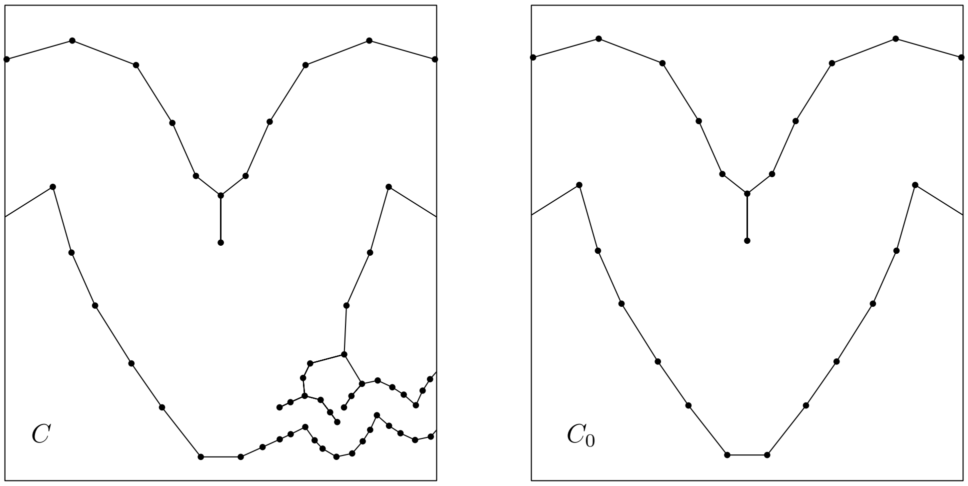


Fig. 14. Enlarged view of the central part of the C and C_0 connections.

6. ELEMENTARY PIECES

We have 10 elementary pieces, shown in Figures 12–14: The (i) *wire-piece* (Figure 16) is the main tool achieving a non-locality for conveying information over long distance. As mentioned in the introduction, such wire-pieces were originally conceived by Jack Snoeyink [Beirouti and Snoeyink 1998]. In order to construct wires of arbitrary length, we also have (ii) an *extended wire-piece.* (Figure 16). The horizontal offset between the two terminal triangles of a wire-piece is 27.4, i. e., with these wire-pieces alone, one can bridge distances which are multiples of 27.4. The offset for the extended wire-piece is $82.21 = 3 \cdot 27.4 + 0.01$. Thus, with the right combination of wire-pieces and extended wire-pieces, one can form a straight connection of arbitrary length, provided it is long enough:

Lemma 6.1. *Consider two small vertical terminal triangles at the same height with a horizontal distance $d > 230\,000$ that is a multiple of 0.01. Then the two triangles can be connected by a sequence of wire-pieces and extended wire-pieces.*

An analogous statement holds for vertical connections.

Proof. If the distance between two terminal triangles is $d = 0.01 \cdot z$ for some integer $z \geq 3 \cdot 2740 \cdot 2739 = 22\,514\,580$, they can be connected by concatenating $y := z \bmod 2740$ extended wire-pieces with $\lfloor z/2740 \rfloor - 3y$ wire-pieces. $\square$

The work-horse of our gadgets is (iii) the *C-connection* and its mirror image, (iv) the *C'-connection.* These are the only pieces with three terminal triangles, two large ones and a small one. They serve two purposes: they allow us to introduce branches into the network of wires, and they effect "negation" of the information that is transmitted through the wires. We also have a (v) *C_0-connection* with only two terminals. It serves as a placeholder for the C or C' connection when the third terminal is not needed.

The remaining pieces are variations of the wire-piece for building more flexible wire shapes: The (vi) *left bend* and the (vii) *right bend* allow us to introduce arbitrary turns into wires. The vertical part of the left bend corresponds to a wire-

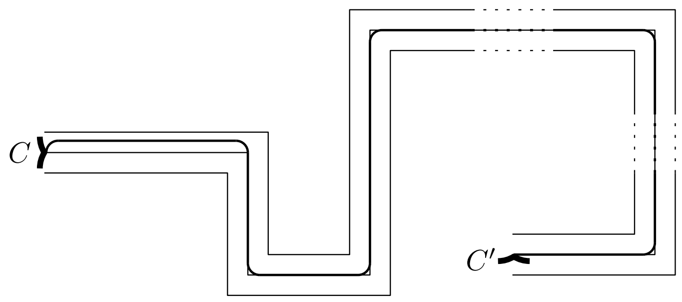


Fig. 15. Tracing a wire through a rectangular corridor, forming a so-called C-C'-link (see Section 7.2.) The dotted parts are long enough to allow connections of any length (Lemma 6.1).

piece that is rotated 90° counter-clockwise, whereas the vertical part of a right bend corresponds to a wire-piece that is rotated 90° clockwise. We need both a left and a right bend, because each piece fits to a given terminal triangle of another piece only at one end.

With these pieces, we can arrange wires to follow any rectangular layout, provided we blow the layout up sufficiently, see Figure 15:

Lemma 6.2. *Given a C-connection piece and a C'-connection piece and a rectangular path between them, the small triangles of the two pieces can be connected by a sequence of wire-pieces, extended wire pieces, left and right bends, within the corridor of width* 2000 *around the path, provided that the path contains a straight portion of length at least* 250 000*, both in the horizontal and in the vertical direction, and the corridor does not interfere with other pieces or intersect itself.*

Proof. The left and right bend both fit into a box of size 700×700, and by Lemma 6.1, we can bridge any distance larger than 230 000 in the horizontal and vertical direction. The wires can be positioned sufficiently far away from the boundary of the rectangular strip to make sure that the boundaries of different connections do not interfere with each other. □

We also have (viii) a *thick left bend*, which is just a scaled copy of the left bend.

Finally, we have two *resizing wire-pieces*: the (ix) *thickening adapter* and its mirror image, the (x) *thinning adapter*. They are used to interpolate between small and large terminal triangles in the clause gadgets.

Figures 16 and 17 show the coordinates of the wire-piece and the extended wire-piece. The complete data for the pieces are available on the Internet.[1]

Proposition 6.3. *The elementary pieces have the following properties:*

All coordinates are multiples of 0.0001*. Each piece contains two or three copies of a terminal triangle, together with a copy of W_{left} or W_{right} (Figure 10), possibly rotated by* 90°*, and possibly scaled by* 4.3*. The coordinates of the terminal triangles are multiples of* 0.01*.* □

Each piece has two terminal triangles, with the exception of the connection pieces, which have three terminal triangles, as shown in the figures. The edges shown in

[1] `http://www.inf.fu-berlin.de/inst/ag-ti/people/rote/Software/MWT/`all-coordinates.txt

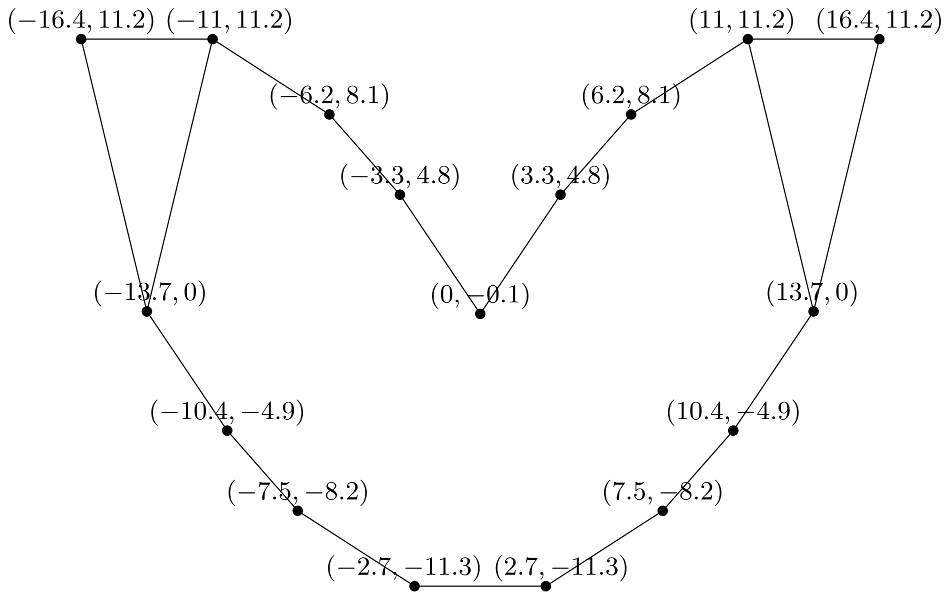


Fig. 16. The coordinates of a wire-piece.

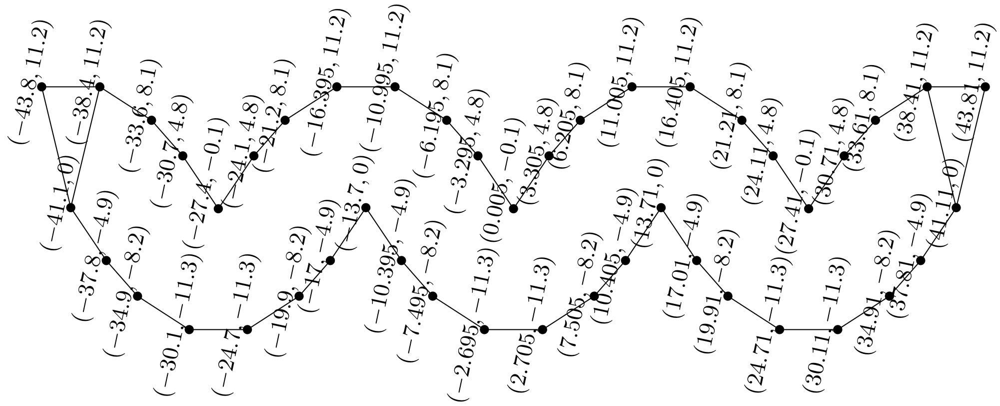


Fig. 17. The coordinates of the extended wire-piece.

the figure, with the exception of the two equal sides of each terminal triangle, form the *boundary* of the pieces.

Proposition 6.4. *The boundary edges of all elementary pieces are contained in the β-skeleton, for $\beta = 1.1806$. This remains true when the pieces are connected using their terminal triangles, as long as the boundary part of different pieces are sufficiently far apart. Hence, all boundary edges belong to the MWT.*

Proof. The lemma can be verified manually by inspecting the relevant regions for the boundary edges. Since the number of boundary edges is very large, we have implemented a computer program which computes the β-skeleton. The source code can be found in Appendix B.3. $\square$

The boundary decomposes into two or three connected *boundary* pieces. As can be seen in Figure 14, some boundary pieces are not just paths, but they form trees.

6.1 Analysis of the Pieces

In this section we discuss the exact properties of our gadgets as they were computed by our computer program. We present tables which show the internal cost, the reduced internal cost $\bar{c}$ (see Section 5.1) and the *relative reduced internal cost* $\tilde{c}$ (i.e., the difference between the reduced cost of a certain configuration and the minimum reduced cost incurred by any configuration of a gadget).

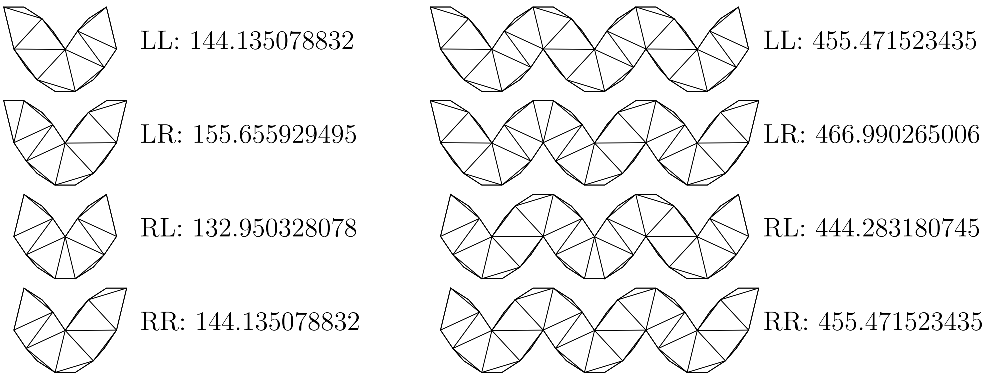


Fig. 18. Optimal solutions for all cases for the wire-piece and the extended wire-piece

For example, let us look at the extended wire-piece. It has two terminal triangles. For each of the four combinations of states, LL, LR, RL, and RR, the right part of Figure 18 shows an optimal solution, together with the internal cost. Table III summarizes this information. The second column lists the number of optimal solutions (or potential optimal solutions, i. e., all solutions which the program could not exclude from being an optimal solution, with the given precision of the calculation). Indeed, looking back at Figure 18, one can see that the solution for case LR has a symmetric copy with the same cost. The astute reader may wonder why the multiplicities for cases LR and RL are not larger. For example, it looks as if the isosceles "xyz" triangle for case LR could be placed also at the very left end or at the very right end. However, this is not the case, since the three repetitions of the wire-piece that constitute the extended wire-piece are not exact repetitions: the ends have been moved apart, stretching the middle piece and thus causing a deviation from the regular periodical pattern. The third column of Table III lists the internal cost, repeating the information from Figure 18.

pattern	multiplicity	internal cost c	reduced internal cost $\bar{c}$	relative reduced cost $\tilde{c}$
LL	1	455.471 523 435	455.471 523 435	0.000 000 000
LR	2	466.990 265 006	455.679 921 006	0.208 397 570
RL	1	444.283 180 745	455.593 524 745	0.122 001 310
RR	1	455.471 523 435	455.471 523 435	0.000 000 000

Table III. Analysis of the extended wire-piece

The fourth column is the reduced internal cost $\hat{c}$, obtained by adding or subtracting δ_{small}, as appropriate. For example, for case LR, we have to subtract $2\delta_{\text{small}} = 11.310\,344$, since this triangulation covers the larger area on both the left and the right end. For cases LL and RR, addition and subtraction of δ_{small} cancel, and the reduced cost remains unchanged.

The rightmost column reduces all numbers by subtracting the column minimum, to make the differences more visible. It is now plain to see that LL and RR have the same cost, which is also obvious by symmetry. We denote these *relative reduced costs* by $\tilde{c}$, and we think of them as *penalties* for deviating from the "ground state" where the smallest reduced cost is achieved. They are non-negative by definition.

The corresponding table for all pieces, together with pictures of the optimal triangulations, is given in Table VII of Appendix A. Table IV condenses the information for all pieces with two terminals.

Since the pieces are symmetric, LL and RR always have the same cost. The only exceptions are the resizing pieces. Let $\varepsilon_1 \approx 0.000\,051$ be the relative reduced cost of the thickening adapter in state LL.

	wire-piece	extended wire-piece	thickening adapter	thinning adapter
LL	0.000 000 000	0.000 000 000	$0.000\,051\,402\ldots = \varepsilon_1$	0.000 000 000
LR	0.210 506 663	0.208 397 570	0.018 887 246	0.018 887 246
RL	0.125 593 246	0.122 001 310	0.014 627 250	0.014 627 250
RR	0.000 000 000	0.000 000 000	0.000 000 000	$0.000\,051\,402\ldots = \varepsilon_1$

	C_0 connection	left bend	right bend	thick left bend
LL	0.000 000 000	0.000 000 000	0.000 000 000	0.000 000 000
LR	0.044 001 701	0.086 460 895	0.210 506 663	0.891 261 046
RL	0.020 571 757	0.125 593 246	0.125 593 246	0.020 571 757
RR	0.000 000 000	0.000 000 000	0.000 000 000	0.000 000 000

Table IV. The relative reduced costs $\tilde{c}$ for the pieces with two terminals

The C-connection (and its mirror image, C') is the only piece with three terminals. The results for this piece is shown in Table V, and in more visual form in Figure 19. We encode the states by three letters ABc, where the capital letters A and B refer to the left and right large terminal triangles (L or R), as usual, and c refers to the small terminal triangle (l or r).

C	internal cost c	reduced internal cost $\hat{c}$	relative reduced cost $\tilde{c}$	C'
LLl	14 027.752 986 494	14 033.408 158 494	$0.003\,861\,076\ldots = \delta_1$	RRr
LLr	14 039.059 470 993	14 033.404 298 993	$0.000\,001\,575\ldots = \varepsilon_2$	RRl
LRl	14 075.894 120 134	14 033.434 292 134	0.029 994 716	LRr
LRr	14 087.200 604 634	14 033.430 432 634	0.026 135 215	LRl
RLl	13 979.654 697 176	14 033.424 869 176	0.020 571 757	RLr
RLr	13 990.965 042 750	14 033.424 870 750	0.020 573 332	RLl
RRl	14 027.749 125 419	14 033.404 297 419	$0.000\,000\,000 = 0$	LLr
RRr	14 039.064 100 296	14 033.408 928 296	$0.004\,630\,878\ldots = \delta_2$	LLl

Table V. Analysis of the pieces C and C'. Since C' is the mirror image of C, it is represented in the same table

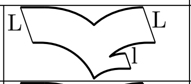
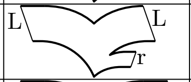
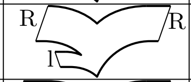
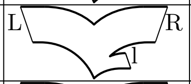
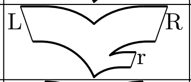
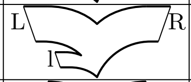
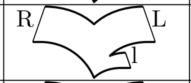
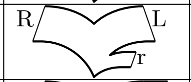
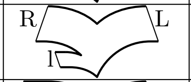
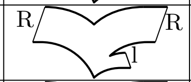
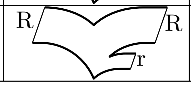
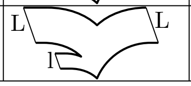
	C	relative reduced cost	C'	
LLl		$0.003\,861\,076\ldots = \delta_1$		RRr
LLr		$0.000\,001\,575\ldots = \varepsilon_2$		RRl
LRl		$0.029\,994\,716$		LRr
LRr		$0.026\,135\,215$		LRl
RLl		$0.020\,571\,757$		RLr
RLr		$0.020\,573\,332$		RLl
RRl		$0.000\,000\,000 = 0$		LLr
RRr		$0.004\,630\,878\ldots = \delta_2$		LLl

Fig. 19. All cases for the analysis of the pieces C and C'.

All tables were computed using fixed-precision interval arithmetic with 15 decimal digits after the decimal point (i.e., multiples of 10^{-15}). The resulting intervals were rounded to 9 digits for displaying. It turned out that the intervals were small enough so that they could be rounded to unique 9-digit numbers. Thus, for example, $0.000\,051\,402$ denotes a number that is guaranteed to lie strictly between $0.000\,051\,401\,5$ and $0.000\,051\,402\,5$.

By inspecting Tables IV and V, one can see that there is a penalty for having two triangles of different states. If both terminal triangles of a piece with two terminal triangles, or the two large terminal triangles of C or C', are in a different state, we call this a *breach*. A breach is so expensive that it will never occur in an optimal solution. We summarize this information for later use.

PROPOSITION 6.5. *In every piece, a breach has relative reduced cost at least* $\delta_3 := 0.01$. *If there is no breach, the relative reduced cost is zero, except for* C, *and* C', *and the resizing wire-pieces. For the resizing wire-pieces, the relative reduced cost is at most* $\varepsilon_1 < 0.000\,051$. □

It follows that long wires have a preference to be triangulated uniformly:

LEMMA 6.6 (THE WIRE LEMMA). *Let* T_1, T_2 *be two small terminal triangles connected by an arbitrary sequence of wire-pieces, extended wire-pieces, and left and right bends. If* T_1 *and* T_2 *are in the same state, all the connecting pieces are in that state and the relative reduced cost is 0. If* T_1 *and* T_2 *are in different states, the relative reduced cost is at least* $\delta_3 = 0.01$. □

The states of C and C' without a breach are LLl, LLr, RRl, and RRr. From Table V, one sees that the preferred states are LLr, and RRl, where the small triangle has the opposite state from the large triangles. We call these two states the *consistent* states, and the states LLl and RRr *inconsistent*. We define

$$\begin{aligned}\varepsilon_2 &:= \tilde{c}(C, \mathrm{LLr}) = \bar{c}(C, \mathrm{LLr}) - \bar{c}(C, \mathrm{RRl}) = c(C, \mathrm{LLr}) - c(C, \mathrm{RRl}) - 2\delta_{\mathrm{small}} \\ \delta_1 &:= \tilde{c}(C, \mathrm{LLl}) = \bar{c}(C, \mathrm{LLl}) - \bar{c}(C, \mathrm{RRl}) = c(C, \mathrm{LLl}) - c(C, \mathrm{RRl}) \\ \delta_2 &:= \tilde{c}(C, \mathrm{RRr}) = \bar{c}(C, \mathrm{RRr}) - \bar{c}(C, \mathrm{RRl}) = c(C, \mathrm{RRr}) - c(C, \mathrm{RRl}) - 2\delta_{\mathrm{small}},\end{aligned}$$

where $c(C, \mathrm{LLr})$, $\bar{c}(C, \mathrm{LLr})$, etc. denote the cost of the C-connection in the respective configuration, see Table V.

PROPOSITION 6.7. *In C or C', every consistent state has relative reduced cost at most $\varepsilon_2 < 0.000\,002$. The two inconsistent states have cost $\delta_1 \approx 0.003\,861$ and $\delta_2 \approx 0.004\,631$. Thus, if there is no breach in C or C', the relative reduced cost is at most δ_2.* □

In general, we denote by ε_1, ε_2, "small" quantities that we would like to neglect, and which can be made arbitrarily small by refining the coordinates of the pieces or the digits of δ_{small} and δ_{large}. On the other hand, δ_1, δ_2, …, denote "large" quantities whose difference will be made productive for the proof.

7. LARGER GADGETS

We now describe how to assemble the elementary pieces from the last section to obtain larger gadgets which model the logical structure of a PLANAR 1-IN-3-SAT formula. We need *variables*, *clauses*, and the connections between them. The main building block for the variables and clauses is the *bit loop* (Section 7.1). Each bit loop stores a logical state (L or R). To ensure consistency between different bit loops, we use *C-C'-links* (Section 7.2). By connecting several bit loops, we build clauses (Section 7.3) and variables (Section 7.4). The optimal triangulation of a clause is obtained if and only if exactly one of its inputs is in state L (the 1-IN-3 property). A variable is modeled as a chain of bit loops connected by C-C'-links so that all loops are in the same state. Variables can be connected to clauses using the wire-pieces from the last section.

7.1 The Bit Loop

The *bit loop* is used to represent a logical state (L or R). It if formed by connecting any selection of four connection pieces (C, C' or C_0) into a loop, using four thick bends, as shown schematically in Figure 20.

LEMMA 7.1. *In an optimal triangulation, the 8 large terminal triangles of a bit loop are in the same state (L or R).*

PROOF. The only interaction with other pieces is through the at most four small "exit" triangles. Let us keep the states of these triangles fixed. If we set all 8 large terminal triangles to L or to R, the maximum relative reduced cost is at most $4 \cdot \delta_2 < 0.02 = 2\delta_3$, see Proposition 6.7.

On the other hand, if the 8 large terminal triangles are not all equal, there are at least two breaches. Hence, by Proposition 6.5, the relative reduced cost is at least $2\delta_3$, and such a triangulation cannot be optimal. □

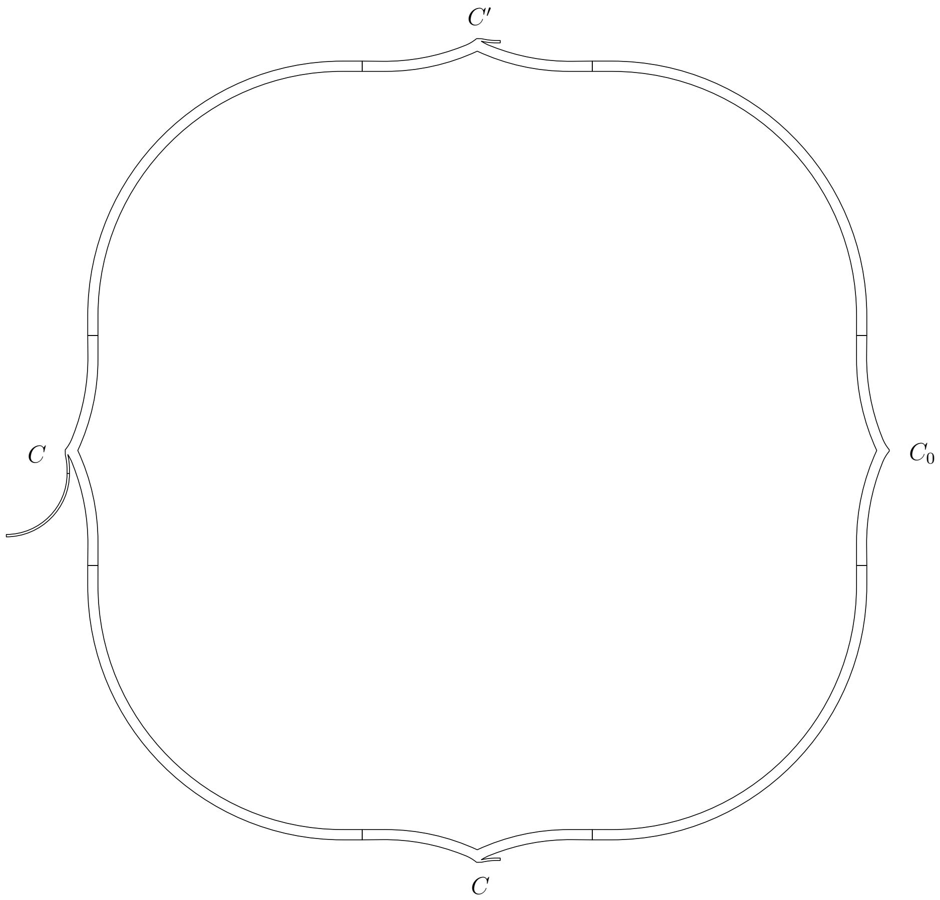


Fig. 20. An instance of a bit loop with two C-connections and one C'-connection. At the left C-connection, a right bend is attached, as the beginning of a wire joining it to another C'-connection. The drawing is to scale, but the shapes are simplified,

We can therefore simply refer to the *state of the bit loop* as the common state of its large terminal triangles. Generally, when it is clear that there is no breach, the state of a C, C' or C_0-connection refers to the state of its large terminal triangles.

7.2 The C-C'-Link

A C-connection can be joined to a C'-connection by an arbitrary sequence of wire-pieces, extended wire-pieces, and left and right bends (Lemma 6.2), see Figure 15. We call the two connections at the end together with their joining wire a *C-C'-link*.

C-C'-links are used in two contexts: The most common case is when the C-connection of the C-C'-link belongs to a bit loop V_1 and the C'-connection belongs to another bit loop V_2. In the clause gadget (Section 7.3), we also have C-C'-links where the C-connection is connected to a thickening adapter and a thinning adapter, while the C' connection is part of a bit loop. By Lemma 7.1 we know that in each of V_1 and V_2, the large triangles are in a uniform state (L or R), and in Section 7.3 we will see that an analogous statement holds for the large terminal triangles of the thickening and the thinning adapter. We thus have to consider four cases for these states, as shown in Table VI. For each combination, we can work out the optimum state of the small terminal triangles by considering Table V. (The first two and last two rows of Table V are sufficient.) The optimal choice for the thin wire is shown in the third column.

V_1	V_2	wire	state of C	state of C'	optimum relative reduced cost
L	L	r	LLr	LLr	$\tilde{c}(C, \mathrm{LLr}) + \tilde{c}(C', \mathrm{LLr}) = \varepsilon_2$
L	R	l (or r)	LLl	RRl	$\tilde{c}(C, \mathrm{LLr}) + \tilde{c}(C', \mathrm{RRr}) = \varepsilon_2 + \delta_1$
R	L	l (or r)	RRl	LLl	$\tilde{c}(C, \mathrm{RRr}) + \tilde{c}(C', \mathrm{LLl}) = \delta_2$
R	R	l	RRl	RRl	$\tilde{c}(C, \mathrm{RRl}) + \tilde{c}(C', \mathrm{RRl}) = \varepsilon_2$

Table VI. The best state of the wire for a C-C'-link. For each given combination of states (first two columns), the last column gives the resulting cost.

LEMMA 7.2. *Each C-C'-link incurs a relative reduced cost of at least ε_2. The cost reaches this minimum if and only if the large terminal triangles in the two pieces have the same state. Otherwise, the cost increases to $\delta_1 + \varepsilon_2$ or δ_2, respectively, see Table VI.*

PROOF. We need only exclude the possibility that the two small terminal triangles are in different states. By the Wire Lemma (Lemma 6.6), this would cause a relative reduced cost of at least $\delta_3 = 0.01$, which is larger than any of the costs in Table VI. □

We call the C-C'-link *inconsistent* if the large triangles in the two connection pieces have different states. An inconsistent C-C'-link incurs a relative reduced cost bigger than δ_1.

7.3 The Clause Gadget

The *clause* gadget is formed by 3 pairs of bit loops, α and $\hat{\alpha}$, β and $\hat{\beta}$, γ and $\hat{\gamma}$, see Figure 21. The two loops of each pair are connected by a C-C'-link in which a thickening adapter, a C-connection piece labeled DOWN, and a thinning adapter is interspersed, and another similar C-C'-link with a C_0-connection piece in the middle. The α-$\hat{\alpha}$ group has three *external* connections: DOWN, UP, and ENTRY.

The DOWN C-connection piece of the pair α-$\hat{\alpha}$ is connected to the UP C' connection of $\hat{\beta}$, and so on in a circular way. Finally, each of α, β, and γ, has a C-connection piece, labeled ENTRY, which will be connected to one of the "input" variables of the clause by a C-C'-link.

LEMMA 7.3. *In an optimal triangulation, α and $\hat{\alpha}$ are in the same state (L or R), and the two large terminal triangles in the DOWN C-connection are in the opposite state.*

Analogous statements hold for $\beta, \hat{\beta}$ and $\gamma, \hat{\gamma}$.

PROOF. By symmetry, it suffices to consider α and $\hat{\alpha}$. The α-$\hat{\alpha}$ group has three external triangles UP, DOWN, and ENTRY which connect it to the outside world.

There are two "ground states" with no internal breaches or inconsistencies. In these states, α and $\hat{\alpha}$ (including UP and ENTRY) have the same state and DOWN and the upper C_0-connection piece in the middle are in the opposite state.

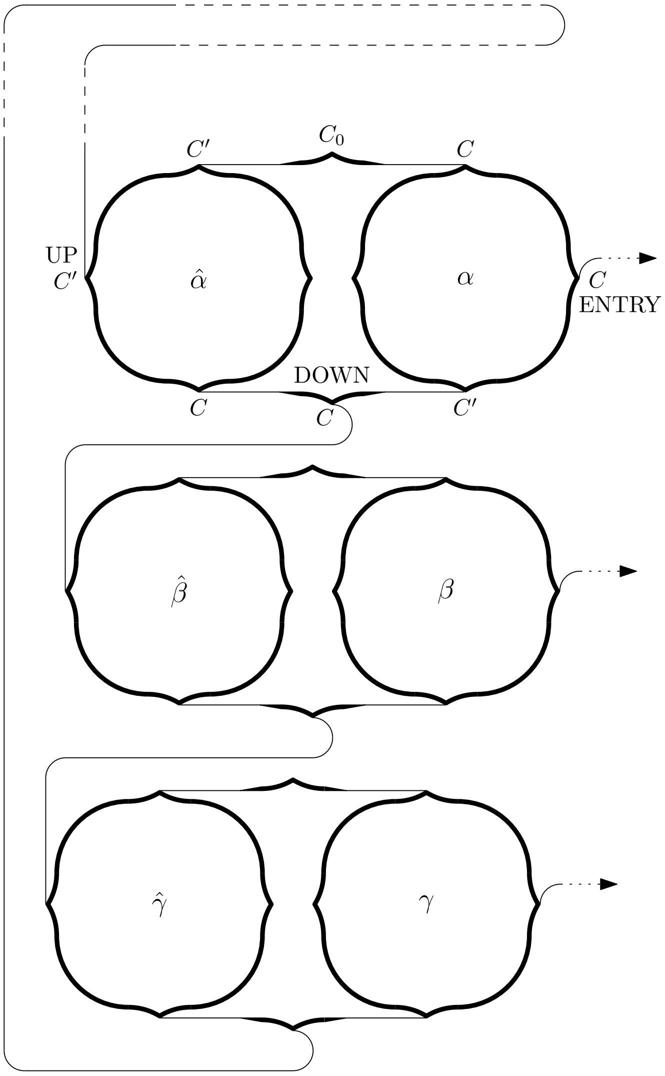


Fig. 21. Schematic view of the clause gadget. The dashed parts of the wires are long enough for sufficiently many copies of the wire-piece and the extended wire-piece to ensure that the wire can reach from the lowest C connection to the left C' connection of $\hat{\alpha}$ (Lemma 6.1).

If we ignore the three external connections, the relative reduced cost of these two states is $2\varepsilon_1 + 2\varepsilon_2$: The contribution from the two C-C'-links is $2\varepsilon_2$. The contribution from the four resizing wire-pieces is $2\varepsilon_1$ since we have two thickening and two thinning adapters: the "penalty state" for the thickening adapter (LL) is the optimal state for the thinning adapter, and vice versa.

Now, fix a setting for the three small external triangles. If we select the ground state which is consistent with the majority of the three exit triangles, we have at most one inconsistent exit connection, causing an additional penalty of at most $\delta_2+2\varepsilon_2$ (two consistent connections with at most ε_2, and one inconsistent connection with at most δ_2).

Thus, there is always a ground state solution with cost at most $\delta_2 + 2\varepsilon_1 + 4\varepsilon_2 <$

0.005. If α and $\hat{\alpha}$ are not in the same state, or if DOWN is in the same state as α and $\hat{\alpha}$, there must be at least two inconsistencies (or even breaches) in the connections between α, $\hat{\alpha}$, and DOWN, causing a cost of at least $2\delta_1 > 0.007$, which is bigger than for the ground state solution. $\square$

Since DOWN is always in a different state from UP, ideally, to successive loops in the cyclic sequence $\alpha\beta\gamma$ should be in a different state. However, since there are three loops, there has to be at least one inconsistency, which provides us the asymmetry necessary for a 1-IN-3-clause.

Thus, we obtain:

LEMMA 7.4. *The reduced cost of the clause gadget (excluding the three ENTRY C-connections) achieves its minimum if and only if exactly one of α, β, and γ is in state* L. *Any other triangulation incurs a cost that is at least $\delta_4 := 0.0007$ larger.*

PROOF. By Lemma 7.3, it suffices to analyze the 8 possible configurations of the pairs α-$\hat{\alpha}$, β-$\hat{\beta}$, and γ-$\hat{\gamma}$ (ignoring the contribution of the ENTRY connections to the clause).

The "internal" contribution or each pair of $2\varepsilon_2 + 2\varepsilon_1$ from its resizing pieces and its two internal C-C'-links is constant, and thus we can ignore this amount when comparing the various possibilities.

Let us look at the C-C'-link between two successive pairs, say α-$\hat{\alpha}$ and β-$\hat{\beta}$. If they are equal, they cause an inconsistency, and the relative reduced cost is δ_2 if $\alpha = \beta =$ L and $\delta_1 + \varepsilon_2$ if $\alpha = \beta =$ R, according to Table VI. If α and β are in different states, the link is consistent, and the cost is ε_2.

Since the situation is unchanged under cyclic shifts of the sequence $\alpha\beta\gamma$, it is enough to consider four cases:

—$\alpha\beta\gamma =$ LLL or $\alpha\beta\gamma =$ RRR: In these cases, we have three inconsistencies, and the relative reduced cost is bigger than $3\delta_1 > 0.01$.

—$\alpha\beta\gamma =$ LLR: the relative reduced cost is $\delta_2 + 2\varepsilon_2 > 0.0046$.

—$\alpha\beta\gamma =$ RRL: the relative reduced cost is $\delta_1 + 3\varepsilon_2 < 0.0039$.

$\square$

In the previous lemma, we have ignored the relative reduced cost of the ENTRY C-connections. They will be accounted for as part of the C-C'-links that they form with the variables, to be described next.

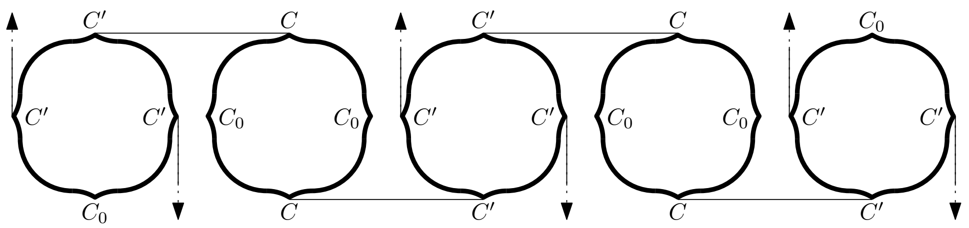


Fig. 22. Schematic view of a variable gadget with three upper and three lower exits. The pattern can be repeated arbitrarily.

7.4 The Variable Gadget

The gadget that represents a variable of the 1-IN-3 instance is formed by chaining together sufficiently many bit loops and connecting them in a row using C-C'-links, as shown in Figure 22. The dotted arrows denote potential exit wires that establish C-C'-links with the corresponding clause gadgets of the previous section. Unused exits can be replaced by C_0 connections.

LEMMA 7.5. *Consider a variable chain V together with the C-C'-links that connect it to the clauses, including the ENTRY C-connections in the clauses. The reduced cost of this point set achieves its minimum if and only if all the bit loops and the C-connections in the clauses are in the same state. Any other triangulation incurs a cost that is at least δ_1 larger.*

PROOF. This follows directly from Lemmas 7.1 and 7.2. □

8. THE REDUCTION

THEOREM 8.1. *Minimum-weight triangulation is strongly NP-hard.*

PROOF. A rectilinear embedding of the given PLANAR 1-IN-3-SAT formula can be constructed on a grid of size $O(n) \times O(n)$. The reduction procedure then simply replaces the edges, variables and clauses of the PLANAR 1-IN-3-SAT formula by the appropriate gadgets, taking care to leave enough space between the individual wires.

This procedure yields a point set S. By construction, the boundaries of all the gadgets are part of the β-skeleton of S (Proposition 6.4), and therefore they belong to the minimum-weight triangulation. The faces outside the wires are simple polygons and can be optimally triangulated using dynamic programming. For each gadget, we know the desired "ideal triangulation" (Lemmas 6.6 and 7.1–7.5) and can calculate its weight. Adding up these weights and the weights of the faces outside the wires yields a target weight w. By construction, the input instance is 1-IN-3 satisfiable if and only if the minimum weight of a triangulation of S is w. Otherwise, the weight of the shortest triangulation is at least $w + 0.0007$ (Lemmas 7.4 and 7.5).

The set S is a subset of an $O(n) \times O(n)$ grid, but it does not fill the whole area: it follows the linear structure of the edges of the rectilinear embedding. Therefore, S has $O(n)$ points. Hence the triangulation has $O(n)$ edges of length $O(n)$. By calculating all edge lengths with an absolute error of $O(1/n^2)$, the reduction algorithm can thus calculate, in polynomial time, a threshold $\hat{w}$ such that the input formula is satisfiable iff there is a triangulation of length at most $\hat{w}$. □

9. CONCLUSION

9.1 Running Times for Computer Verification of the Proof

In designing our gadgets and our proof, we have tried to achieve a balance between the number of different pieces, which affects the complexity of the human-readable part of the proof and the number of case distinctions, and the size of the pieces, which affects the complexity (running time) for the mechanical part of the proof that has to be checked by computer (or accepted by faith).

The running time is dominated by the $O(n^3)$ dynamic programming algorithm for triangulating simple polygons. The largest point sets that we handle have 493 points (the left bends). To total time to run all verifications (with exact integer interval arithmetic) was about 10 hours on a four-year-old moderate PC.

9.2 Dimensions.

Our gadgets have constant size, but they consist of several thousands of points and are quite enormous. For example, the clause gadget (see Figure 21) has dimensions on the order of $250\,000 \times 250\,000$, with coordinates that are specified as multiples of 10^{-4}. On the other hand, the difference between a satisfiable and an unsatisfiable SAT instance is reflected in a minute difference of 0.0007 in the MWT cost.

With some work, it would be possible to reduce this to some more "reasonable" figures or to amplify the difference between satisfiable and unsatisfiable instances, but we did not find it worth the effort to do so. First of all, the current gadgets are already a result of tedious experiments, pushing points into various directions and trying to understand what happens. Some parts of the design, in particular the C-connections, are very delicate. Secondly, the dimensions would still be very large. One can certainly reduce the constant 230 000 of Lemma 6.1 by providing a greater variety of extended wire-pieces, but the bit loop, for example, (Figure 20) already has size approximately 7000×7000, and it does not seem easy to push the size very much below these limits, unless one comes up with a completely different design.

9.3 Open Problems

Several interesting problems remain open. First of all, it is not known whether the MWT problem is in NP, since it is not known how to compare sums of Euclidean lengths in polynomial time [Blömer 1991], but this difficulty is more of an algebraic nature. To define a variant of MWT which is in NP, one can take the weight of an edge e as the rounded value $\lceil \|e\|_2 \rceil$. With appropriate scaling, our proof also establishes NP-completeness for this variant.

Our reduction shows that it is NP-hard to approximate the MWT with a relative approximation error which is better than $O(1/n^2)$: The difference between a satisfiable instance of PLANAR 1-IN-3-SAT and an unsatisfiable instance is reflected in a constant increase of the MWT cost, and, as mentioned in the proof of Theorem 8.1, the total cost of the MWT is $O(n^2)$.

One can probably reduce this bound to $O(n \log n)$, and thus establish that it is NP-hard to achieve a relative approximation error better than $O(1/(n \log n))$, by using the fact that the interior of a convex k-gon of perimeter p can be triangulated with weight $O(p \log k)$. First, the wires and all gadgets form linear structures of total length $O(n)$; thus, the length of the gadget boundaries, and the MWT inside the gadgets is only $O(n)$. This leaves the holes to be triangulated. It should be quite straightforward to extend our construction in such a way that, apart from $O(n)$ constant-size holes, only $O(n)$ convex holes with a total of $O(n)$ vertices remain: one would insert paths of additional points whose β-skeleton separates the big holes from the jagged wire boundaries and cuts the holes into convex pieces, apart from "linear" structures that cover only $O(n)$ area.

These non-approximability results do not rule out the existence of a polynomial-time approximation scheme. For a long time, attempts to extend techniques from geometric approximation algorithms to the MWT problem have only led to constant factor approximations (see [Bern and Eppstein 1996] for a survey). Remy and Steger [2006] showed that it is possible to compute a $(1+\varepsilon)$-approximation of the MWT in time $n^{O(\log^8 n)}$, providing strong evidence that a PTAS might exist.

In practice, the LMT-skeleton heuristic is extremely fast in computing LMTs. Combined with with bucketing techniques and fast preprocessing techniques [Drysdale et al. 1995], one empirically achieves almost linear running times. Thus, in this respect the MWT problem seems to be similar to the Knapsack Problem, which is also NP-hard but easy to solve in practice [Kellerer et al. 2007]. It would be interesting to analyze the LMT-skeleton heuristic for random point sets. The good practical performance indicates that the expected running time for random inputs might be polynomial, or even close to linear. On the other hand, as point sets get huge, they will contain, with non-negligible probability, some larger and larger point configurations that are hard to triangulate.

ACKNOWLEDGMENTS

The authors would like to thank Eric Demaine for pointing out that the negations in our original paper are not necessary.

The gadgets in this paper were designed using Otfried Cheong's extensible drawing editor IPE [Schwarzkopf 1995] as a graphical user interface. We also used the xml-format of IPE to generate many of the illustrations of this paper directly from computer output.

We also thank an anonymous reviewer for extensive remarks that helped to improve the presentation of the paper.

REFERENCES

Aichholzer, O., Aurenhammer, F., and Hainz, R. 1999. New results on MWT subgraphs. *Inf. Process. Lett. 69,* 5, 215–219.

Anagnostou, E. and Corneil, D. 1993. Polynomial-time instances of the minimum weight triangulation problem. *Comput. Geom. Theory Appl. 3,* 5, 247–259.

Beirouti, R. and Snoeyink, J. 1998. Implementations of the LMT heuristic for minimum weight triangulation. In *SCG'98: Proceedings of the Fourteenth Annual Symposium on Computational Geometry.* ACM Press, New York, 96–105.

Belleville, P., Keil, M., McAllister, M., and Snoeyink, J. 1996. On computing edges that are in all minimum-weight triangulations. In *SCG'96: Proceedings of the Twelfth Annual Symposium on Computational Geometry.* ACM Press, New York, 507–508.

Bern, M., Edelsbrunner, H., Eppstein, D., Mitchell, S., and Tan, T. S. 1993. Edge insertion for optimal triangulations. *Discrete Comput. Geom. 10,* 1, 47–65.

Bern, M. W. and Eppstein, D. 1992. Mesh generation and optimal triangulation. In *Computing in Euclidean Geometry*, D.-Z. Du and F. K.-M. Hwang, Eds. Number 1 in Lecture Notes Series on Computing. World Scientific, River Edge, NJ, 23–90.

Bern, M. W. and Eppstein, D. 1996. Approximation algorithms for geometric problems. In *Approximation Algorithms for NP-hard Problems*, D. Hochbaum, Ed. PWS Publishing, Boston, MA, Chapter 8, 296–345.

Blömer, J. 1991. Computing sums of radicals in polynomial time. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science.* IEEE Computer Society Press, Los Alamitos, CA, 670–677.

Bose, P., Devroye, L., and Evans, W. S. 2002. Diamonds are not a minimum weight triangulation's best friend. *Int. J. Comput. Geometry Appl. 12,* 6, 445–454.

CHENG, S.-W., GOLIN, M., AND TSANG, J. 1995. Expected case analysis of β-skeletons with applications to the construction of minimum-weight triangulations. In *Proceedings of the 7th Canadian Conference on Computational Geometry*. Quebec City, 279–284.

CHENG, S.-W., KATOH, N., AND SUGAI, M. 1996. A study of the LMT-skeleton. In *ISAAC '96: Proceedings of the 7th International Symposium on Algorithms and Computation*. Lecture Notes in Computer Science, vol. 1178. Springer-Verlag, London, 256–265.

CHENG, S.-W. AND XU, Y.-F. 1996. Approaching the largest β-skeleton within a minimum weight triangulation. In *SCG'96: Proceedings of the Twelfth Annual Symposium on Computational geometry*. ACM Press, New York, 196–203.

DAS, G. AND JOSEPH, D. 1989. Which triangulations approximate the complete graph? In *Proceedings of the International Symposium on Optimal Algorithms*. Lecture Notes in Computer Science, vol. 401. Springer-Verlag, New York, 168–192.

DE BERG, M., VAN KREVELD, M., OVERMARS, M., AND SCHWARZKOPF, O. 2000. *Computational Geometry: Algorithms and Applications*, 2 ed. Springer Verlag, Berlin Heidelberg New York.

DICKERSON, M. T., KEIL, J. M., AND MONTAGUE, M. H. 1997. A large subgraph of the minimum weight triangulation. *Discrete Comput. Geom. 18,* 3, 289–304.

DRYSDALE, R., ROTE, G., AND AICHHOLZER, O. 1995. A simple linear time greedy triangulation algorithm for uniformly distributed points. Tech. Rep. IIG-408, Institute for Information Processing, Technische Universität Graz.

DRYSDALE, R. L., MCELFRESH, S., AND SNOEYINK, J. S. 2001. On exclusion regions for optimal triangulations. *Discrete Appl. Math. 109,* 1-2, 49–65.

DÜPPE, R.-D. AND GOTTSCHALK, H.-J. 1970. Automatische Interpolation von Isolinien bei willkürlich verteilten Stützpunkten. *Allgemeine Vermessungs-Nachrichten 77,* 10, 423–426.

EDELSBRUNNER, H., TAN, T. S., AND WAUPOTITSCH, R. 1992. An $O(n^2 \log n)$ time algorithm for the minmax angle triangulation. *SIAM J. Sci. Statist. Comput. 13,* 4, 994–1008.

EPPSTEIN, D. 1994. Approximating the minimum weight Steiner triangulation. *Discrete & Computational Geometry 11,* 2, 163–191.

GAREY, M. R. AND JOHNSON, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York.

GILBERT, P. D. 1979. New results in planar triangulations. Tech. Rep. R–850, Univ. Illinois Coordinated Science Lab.

HOFFMANN, M. AND OKAMOTO, Y. 2006. The minimum weight triangulation problem with few inner points. *Comput. Geom. Theory Appl. 34,* 3, 149–158.

JOHNSON, D. S. 2005. The NP-completeness column. *ACM Trans. Algorithms 1,* 1, 160–176.

KEIL, J. M. 1994. Computing a subgraph of the minimum weight triangulation. *Comput. Geom. Theory Appl. 4,* 1, 13–26.

KELLERER, H., PFERSCHY, U., AND PISINGER, D. 2007. *Knapsack Problems*. Springer-Verlag, Berlin.

KIRKPATRICK, D. G. 1980. A note on Delaunay and optimal triangulations. *Inf. Process. Lett. 10,* 3, 127–128.

KLINCSEK, G. T. 1980. Minimal triangulations of polygonal domains. In Combinatorics 79 (Proc. Colloq., Univ. Montréal, 1979), Part II, M. Deza and I. G. Rosenberg, Eds. *Ann. Discrete Math. 9*, 121–123.

KNUTH, D. E. AND RAGHUNATHAN, A. 1992. The problem of compatible representatives. *SIAM J. Discrete Math. 5,* 3, 422–427.

KYODA, Y., IMAI, K., TAKEUCHI, F., AND TAJIMA, A. 1997. A branch-and-cut approach for minimum weight triangulation. In *ISAAC '97: Proceedings of the 8th International Symposium on Algorithms and Computation*. Lecture Notes in Computer Science, vol. 1350. Springer-Verlag, London, 384–393.

LEVCOPOULOS, C. 1987. An $\Omega(\sqrt{n})$ lower bound for the nonoptimality of the greedy triangulation. *Inf. Process. Lett. 25,* 4, 247–251.

LEVCOPOULOS, C. AND KRZNARIC, D. 1996. Quasi-greedy triangulations approximating the minimum weight triangulation. In *SODA '96: Proceedings of the Seventh Annual ACM-SIAM*

Symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics, Philadelphia, 392–401.

Lichtenstein, D. 1982. Planar formulae and their uses. *SIAM J. Comput. 11,* 2, 329–343.

Lloyd, E. L. 1977. On triangulations of a set of points in the plane. In *18th Annual Symposium on Foundations of Computer Science (Providence, R.I., 1977)*. IEEE Computer Society, Long Beach, Calif., 228–240.

Manacher, G. K. and Zobrist, A. L. 1979. Neither the greedy nor the Delaunay triangulation of a planar point set approximates the optimal triangulation. *Inf. Process. Lett. 9,* 1, 31–34.

Mulzer, W. and Rote, G. 2006. Minimum weight triangulation is NP-hard. In *SCG '06: Proceedings of the Twenty-Second Annual Symposium on Computational Geometry*. ACM Press, New York, 1–10.

Plaisted, D. A. and Hong, J. 1987. A heuristic triangulation algorithm. *J. Algorithms 8,* 5, 405–437.

Remy, J. and Steger, A. 2006. A quasi-polynomial time approximation scheme for minimum weight triangulation. In *STOC '06: Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*. ACM Press, New York, 316–325.

Schaefer, T. J. 1978. The complexity of satisfiability problems. In *STOC '78: Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*. ACM Press, New York, 216–226.

Schwarzkopf, O. 1995. The extensible drawing editor Ipe. In *SCG '95: Proceedings of the Eleventh Annual Symposium on Computational Geometry*. ACM Press, New York, 410–411.

Shamos, M. I. and Hoey, D. 1975. Closest-point problems. In *16th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, Long Beach, Calif., 151–162.

Wang, C. A. and Yang, B. 2001. A lower bound for β-skeleton belonging to minimum weight triangulations. *Comput. Geom. Theory Appl. 19,* 1, 35–46.

Yang, B.-T., Xu, Y.-F., and You, Z. 1994. A chain decomposition algorithm for the proof of a property on minimum weight triangulations. In *ISAAC '94: Proceedings of the 5th International Symposium on Algorithms and Computation*. Lecture Notes in Computer Science, vol. 834. Springer-Verlag, London, 423–427.

A. THE DATA FOR ALL PIECES

Table VII shows, for all pieces, the internal cost, the reduced cost, and the relative reduced cost, for all configurations of boundary triangles. Figures 23–28, as well as Figure 18, show for each configuration an optimal triangulation. The two resizing wire-pieces are symmetric, and thus only one of them, the thickening adapter, is given. When applying the results to the thinning adapter, the labels must be flipped: LL becomes RR and RR becomes LL. The raw costs for the thick left bend are the same as for the left bend, scaled by 4.3. But since the large terminal triangles use a different value of δ, the reduced costs bear no direct relation to those of the left bend. The last column is what is summarized in Table IV.

We extended our Python programs to output their results in the form of tables (in TeX format) and figures. As the output format for the figures, we used the xml-format of the Ipe program[2] [Schwarzkopf 1995], which can be directly converted to eps or pdf format or further edited. Many illustrations of this paper were obtained in this way. In particular, the table and all figures of this section, as well as Figures 18 and Table II, were generated directly from our programs without manual intervention.

[2] `http://tclab.kaist.ac.kr/ipe/`

pattern	multiplicity	internal cost c	reduced internal cost $\bar{c}$	relative reduced cost $\tilde{c}$
wire-piece				
LL	1	144.135078832	144.135078832	0.000000000
LR	2	155.655929495	144.345585495	0.210506663
RL	1	132.950328078	144.260672078	0.125593246
RR	1	144.135078832	144.135078832	0.000000000
extended wire-piece				
LL	1	455.471523435	455.471523435	0.000000000
LR	2	466.990265006	455.679921006	0.208397570
RL	1	444.283180745	455.593524745	0.122001310
RR	1	455.471523435	455.471523435	0.000000000
thickening adapter				
LL	1	7188.980062460	7207.382390460	0.000051402
LR	1	7237.113898304	7207.401226304	0.018887246
RL	1	7177.684294307	7207.396966307	0.014627250
RR	1	7225.784667058	7207.382339058	0.000000000
C connection				
LLl	1	14027.752986494	14033.408158494	0.003861076
LLr	1	14039.059470993	14033.404298993	0.000001575
LRl	1	14075.894120134	14033.434292134	0.029994716
LRr	1	14087.200604634	14033.430432634	0.026135215
RLl	1	13979.654697176	14033.424869176	0.020571757
RLr	1	13990.965042750	14033.424870750	0.020573332
RRl	1	14027.749125419	14033.404297419	0.000000000
RRr	1	14039.064100296	14033.408928296	0.004630878
C_0 connection				
LL	1	12733.864882577	12733.864882577	0.000000000
LR	2	12782.023884279	12733.908884279	0.044001701
RL	2	12685.770454334	12733.885454334	0.020571757
RR	1	12733.864882577	12733.864882577	0.000000000
left bend				
LL	1	5425.386907832	5425.386907832	0.000000000
LR	2	5436.783712726	5425.473368726	0.086460895
RL	2	5414.202157077	5425.512501077	0.125593246
RR	1	5425.386907832	5425.386907832	0.000000000
right bend				
LL	1	5271.525633165	5271.525633165	0.000000000
LR	2	5283.046483828	5271.736139828	0.210506663
RL	2	5260.340882410	5271.651226410	0.125593246
RR	1	5271.525633165	5271.525633165	0.000000000
thick left bend				
LL	1	23329.163703675	23329.163703675	0.000000000
LR	2	23378.169964722	23330.054964722	0.891261046
RL	2	23281.069275432	23329.184275432	0.020571757
RR	1	23329.163703675	23329.163703675	0.000000000

Table VII. Analysis of the pieces

B. COMPUTER PROGRAMS

We have written programs that check all of the claimed properties. (Propositions 6.3–6.4). If any condition fails, the programs raise an exception. To give

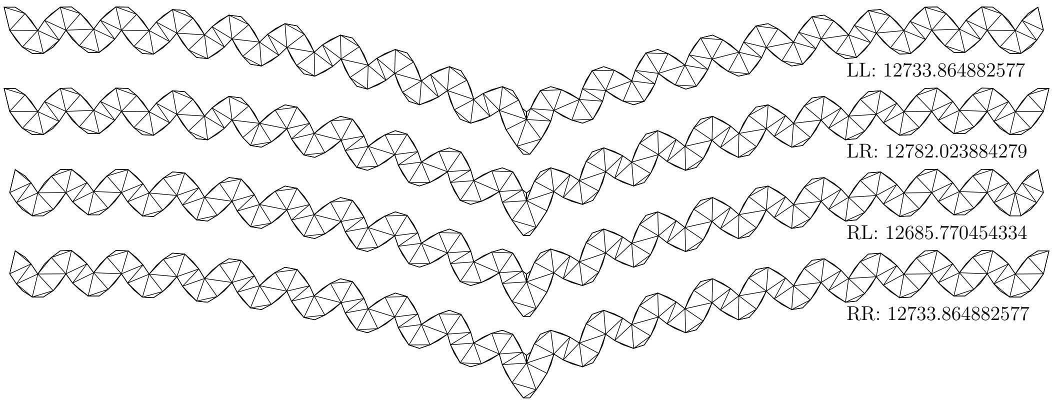


Fig. 23. Optimal solutions for all cases for the C_0 connection

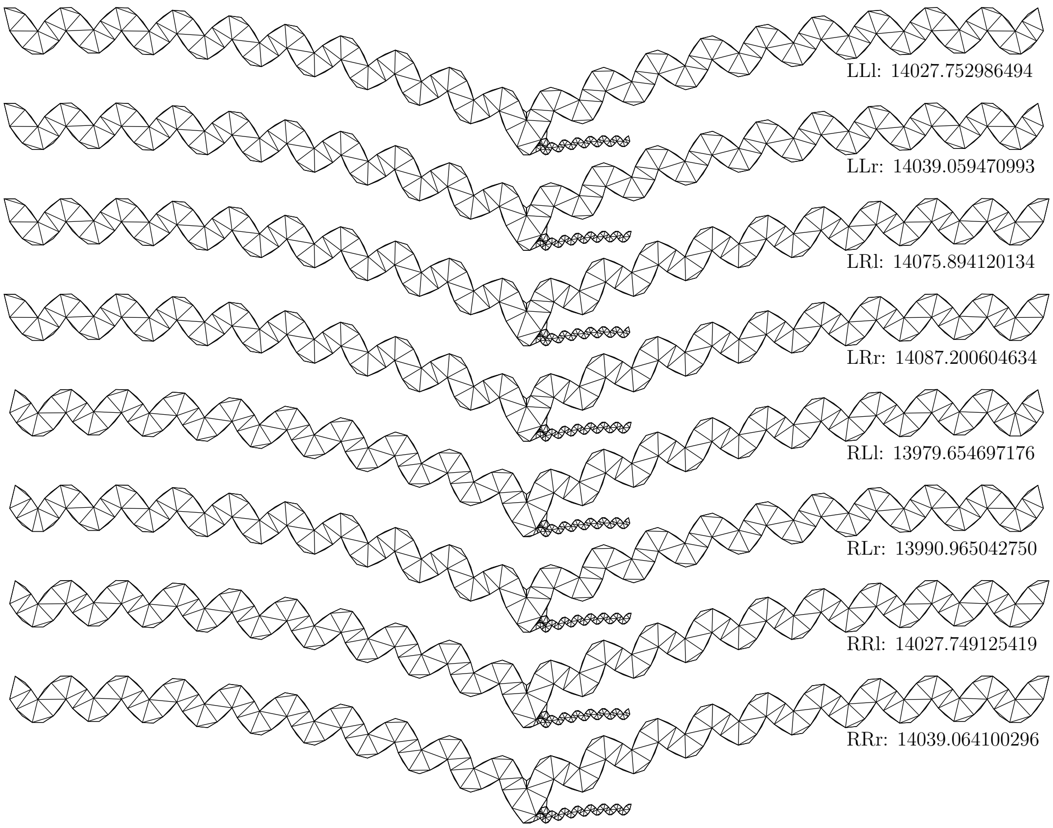


Fig. 24. Optimal solutions for all cases for the C connection

an idea of what was automatically checked, we show an excerpt of the log-file, concerning the extended wire-piece (Figures 17 and 18).

```
======================= extended wire-piece ====================
All coordinates are multiples of 0.0001
terminal triangle basepoint: (-41.100,0)
     edge vector in state L: (-2.700,11.2)
```

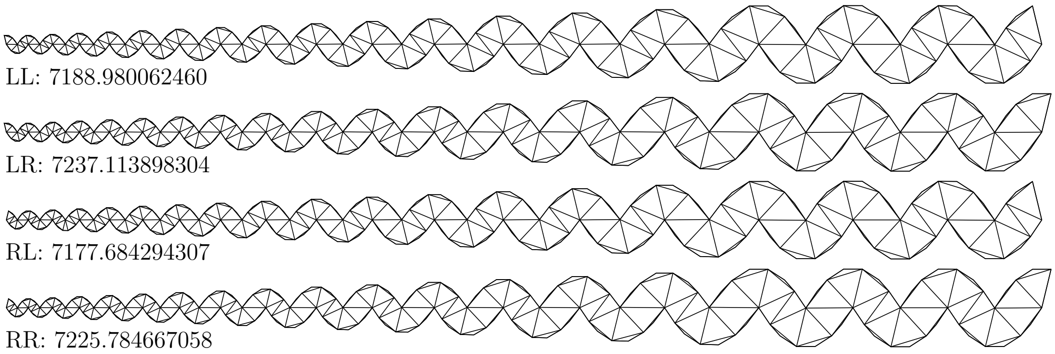


Fig. 25. Optimal solutions for all cases for the thickening adapter

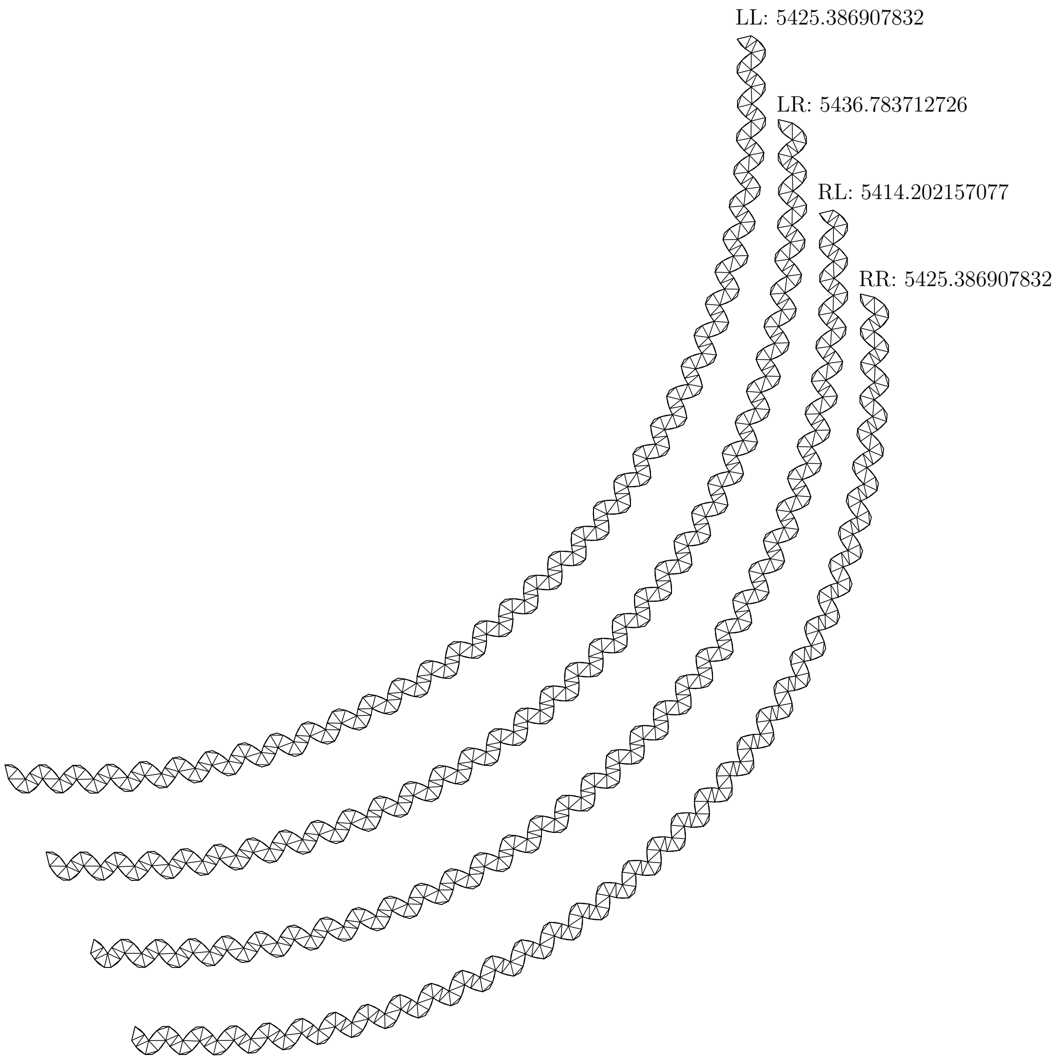


Fig. 26. Optimal solutions for all cases for the left bend

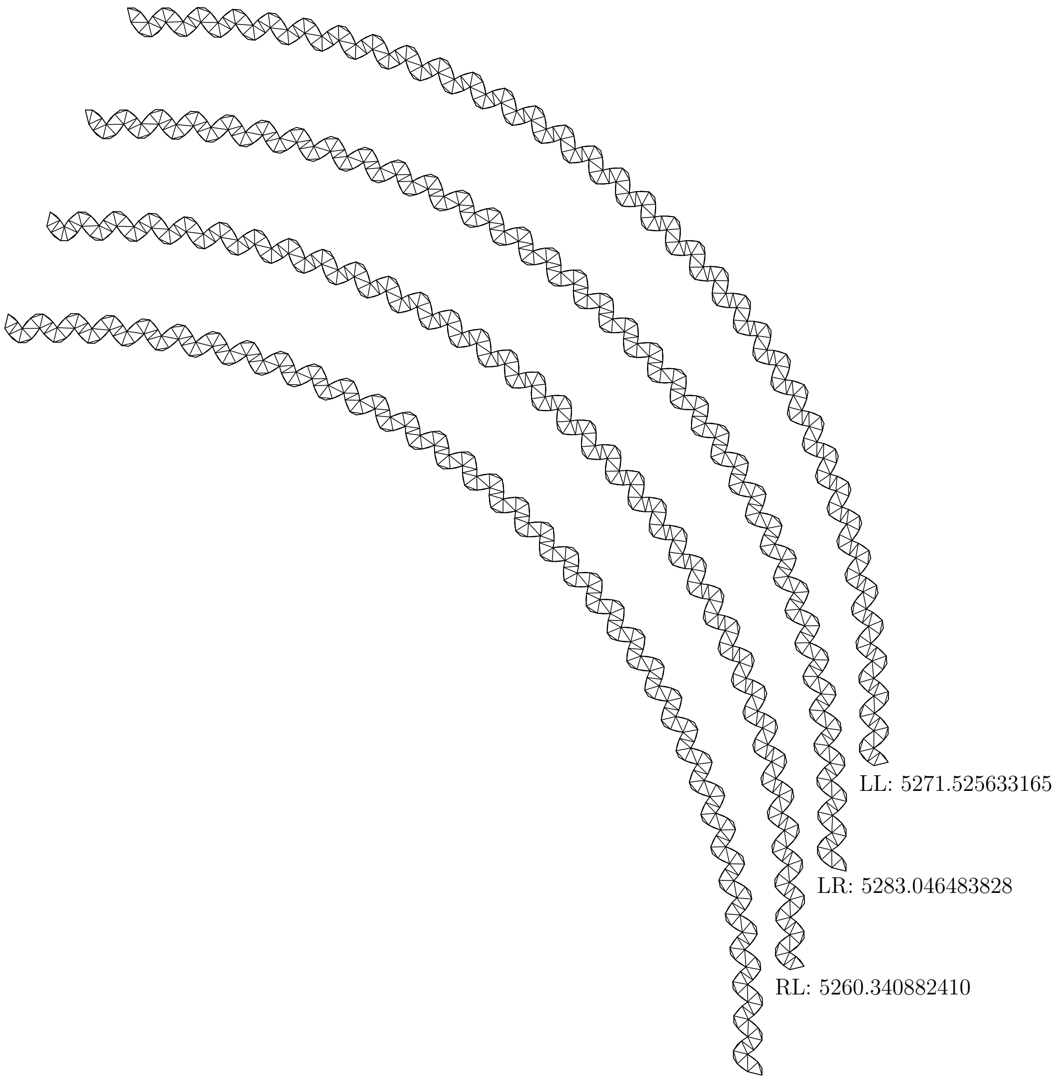


Fig. 27. Optimal solutions for all cases for the right bend

```
     edge vector in state R: (2.700,11.2)
terminal triangle basepoint: (41.110,0)
     edge vector in state L: (-2.700,11.2)
     edge vector in state R: (2.700,11.2)
All terminal coordinates are multiples of 0.01
0 duplicate point(s).
The point set is symmetric with respect to the vertical axis x=0.005.
cos(alpha)^2 = 3120343/10000000 = 0.312034; beta = 1.205637.
case LL: 44 points. 455.471523435 455.471523435
case LR: 45 points. 466.990265006 455.679921006
case RL: 43 points. 444.283180745 455.593524745
case RR: 44 points. 455.471523435 455.471523435
```

The complete PYTHON source code, as well as the data for the pieces, are avail-

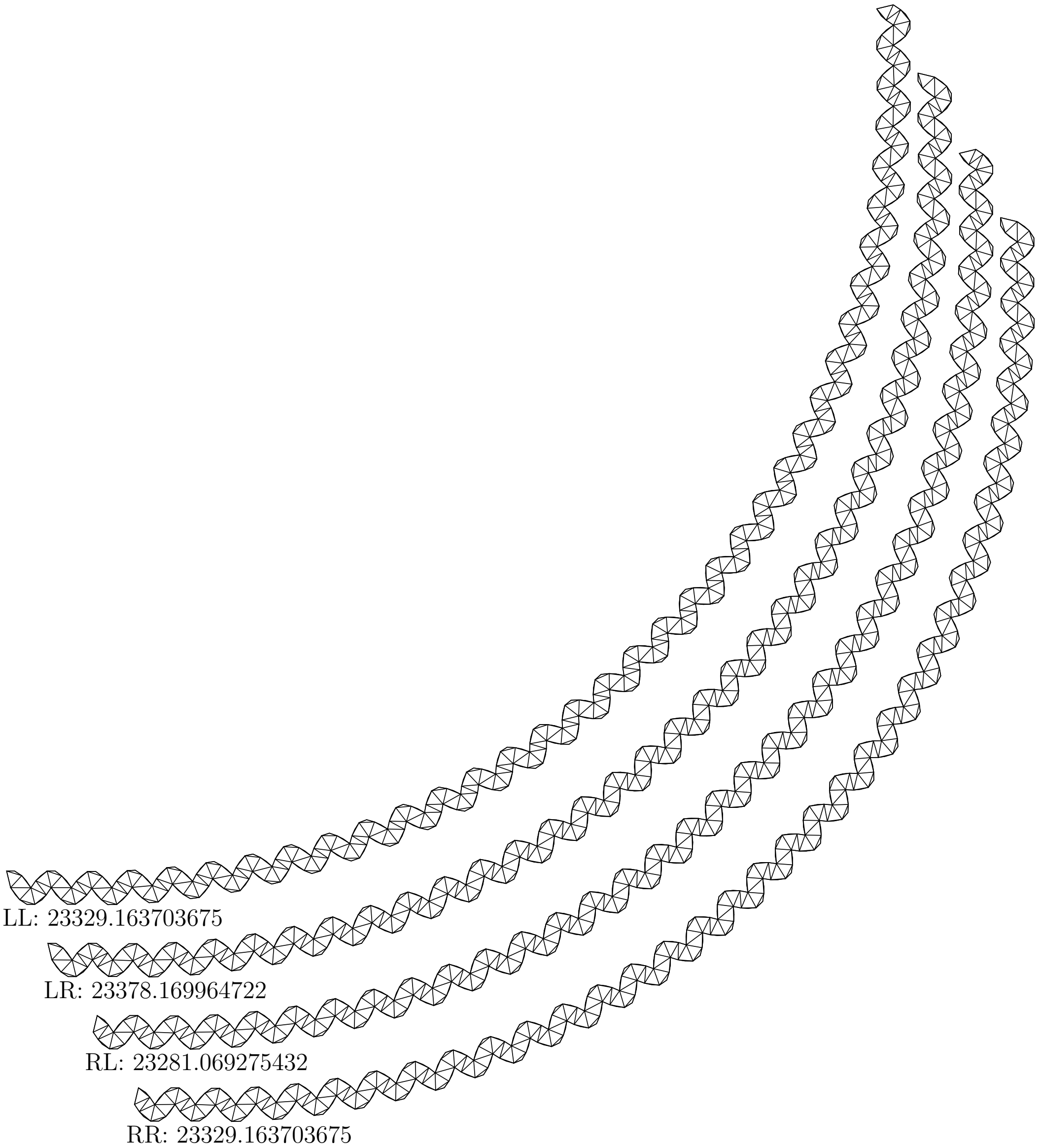


Fig. 28. Optimal solutions for all cases for the thick left bend

able on the Internet.[3] In total, there are about 1000 lines of code. The programs are mostly rather straightforward, so we only show four modules: The module **`mwt_polygon.py`** contains the dynamic programming algorithm for computing the MWT of a simple polygon (Section B.2). It relies on the module **`arithmetic.py`** for the representation of fixed-precision decimal quantities as (long) integers, for (rudimentary) interval arithmetic with integers, and for calculating the Euclidean length as an interval (Section B.1). The module **`check_beta.py`** determines the smallest value β for which the given boundary edges belong to the β-skeleton (Section B.3). The module **`check_W.py`** checks all cases for the point set W for proving

[3] **`http://www.inf.fu-berlin.de/inst/ag-ti/people/rote/Software/MWT/python/`**

Lemma 5.1 (Section B.4).

B.1 Arithmetic

Since the dynamic programming algorithm involves only additions and comparisons, interval arithmetic is not a big deal. We use fixed-point arithmetic, which is simulated by integer arithmetic. Hence all calculations are exact, and we need not care about directed rounding when doing arithmetic with intervals.

The only non-trivial part is the calculation of Euclidean distances by the `length` function, which involves a square root. We use the `decimal` package of PYTHON, which provides an arbitrary-precision square root operation, to obtain a starting approximation of the square root. The approximation is then checked and refined directly, in a straightforward way.

```
from decimal import *

cdigits = 5
scale = 100000 # initial values

def setprecision(computedigits):
  global cdigits,scale
  cdigits = computedigits
  scale = 10**computedigits

def length (p1,p2):
    """Euclidean distance between two points
    p1 and p2 are points with integer coordinates.
    The results is an integer interval."""
    l_squared = (p2[0]-p1[0])**2 + (p2[1]-p1[1])**2

    # obtain an initial approximation using the
    # square root operation of the decimal package
    oldprec = getcontext().prec
    getcontext().prec = len(str(l_squared))/2+2
    a = int(Decimal(l_squared).sqrt().quantize(Decimal(1),
                                             rounding=ROUND_FLOOR))
    getcontext().prec = oldprec # restore the old precision

    # correct and verify the result.
    # The final result will be correct regardless of how good
    # the initial approximation was (although it may take longer).
    while a**2 > l_squared:      a -= 1
    while (a+1)**2 <= l_squared: a += 1
    if a*2 < l_squared: return Interval_integer(a,a+1)
    else:               return Interval_integer(a,a)

def decimal_to_scaled_int(s):
    return int(s*scale)

def print_scaled_int(n,ndigits=None):
    """print integer with decimal point inserted
```

```
      "ndigits" digits from the right end"""
    if n<0:
        sign = "-"
        n = -n
    else:
        sign = ''
    if ndigits==None: ndigits=cdigits # default
    return sign + ("%d.%0"+str(ndigits)+"d") % divmod(n, 10**ndigits)

class Interval_integer(object):
  """ represents an interval [a,b] with a<=b """
  def __init__(self,a,b=None):
    self.a = a
    if b==None:
        self.b = a
    else:
        self.b = b

  def __add__(self,y):
    if isinstance(y, (int,long)):
      return Interval_integer(self.a + y, self.b + y)
    else: # assume y is an Interval_integer:
      return Interval_integer(self.a + y.a, self.b + y.b)

  def quantize(self,ndigits):
    """try to find a unique value which is guaranteed
    to be the rounded value of the interval"""
    scalehalf = 10**ndigits/2
    rounded_value = (self.a/scalehalf + 1)/2
    val = rounded_value*2*scalehalf
    if val-scalehalf < self.a and val+scalehalf > self.b:
        return rounded_value
    print self, "cannot be rounded uniquely to",
    print ndigits, "fewer digits."
    raise ValueError

  def quantize_and_print(self,ndigits,initialdigits=None):
    """print the unique value which is the rounded value
    of the interval with "ndigits" remaining digits, if it exists"""
    if initialdigits==None: initialdigits=cdigits # default
    return print_scaled_int(
                  self.quantize(initialdigits-ndigits), ndigits)

  def __str__(self):
    return "["+str(self.a)+","+str(self.b)+"]"
  def __repr__(self):
    return "Interval("+repr(self.a)+","+repr(self.b)+")"

def keep_best(l):
  """keeps a list that is guaranteed to contain the smallest
     value from a list of intervals"""
```

```
  threshold = min(x.b for x in l)
  return [ x for x in l if x.a <= threshold ]
```

B.2 Dynamic Programming for Triangulating a Polygon

The procedure `mwt` of the module `mwt_polygon.py` implements the classical $O(n^3)$ dynamic programming algorithm for optimally triangulating a simple polygon p [Gilbert 1979; Klincsek 1980]. It has an optional argument `excluded` for specifying edges that cannot be used in the triangulation. The input polygon p is assumed to have integer coordinates. Euclidean edge lengths are calculated as integer intervals using `arithmetic.length()`.

In contrast to Klincsek [1980], our procedure does not explicitly test the edges for crossings; it only tests whether all triangles that are used in the triangulation are oriented counterclockwise. If the input is a simple polygon, this is sufficient to ensure that the resulting triangulation is non-crossing, see Lemma B.1 below. (Hence, the program will also triangulate certain non-simple polygons, but it will not triangulate polygons which are oriented clockwise!)

To show that the our constructions do not depend on assumptions about handling point sets which are not in general position, triangles with three points on a line are (temporarily) considered as valid triangles of a triangulation, but these triangles, as well as any (partial) triangulations that contain such triangles, are flagged as degenerate. If such a triangulation would "survive" as a candidate for an optimal solution, the checking routines would report an error (see for example the procedure `check_W_cases()` in Section B.4). Triangles with coinciding vertices are not considered.

LEMMA B.1. *Let T be a set of triplets (i, j, k) with $1 \le i < j < k \le n$ such that for a convex polygon $P = p_1 \dots p_n$, the triangles $p_i p_j p_k$ for $(i, j, k) \in T$ form a triangulation of P.*

Let P be an arbitrary simple polygon. If all triangles $p_i p_j p_k$ for $(i, j, k) \in T$ are oriented counter-clockwise, they form a triangulation of P.

PROOF. This can be seen by counting the number of triangles in which a given point x of the plane is contained. This number can only change when x crosses an edge of a triangle. However, all triangle edges have another triangle on the opposite side, with the exception of the triangle edges that are edges of P. Thus, the number of triangles in which a point of the plane is contained is constant except at the boundary of P, where it changes by ± 1. Since this number is 0 when the point x is far away, every point x in the interior of P is covered by exactly one triangle, and no triangle sticks out of P. $\square$

```
import arithmetic

def mwt(p,excluded=[]):
  """compute minimum-weight triangulation of a simple polygon p
     by dynamic programming, using integer interval arithmetic.
     The coordinates of p are integers.

     Some edges may be excluded from consideration.
```

```
   The polygon p must be oriented counter-clockwise!
   Otherwise, no solution will be found.
   The result is a list of triplets
        (weight, solution, degenerate_flag)
   where weight is given as an interval, sorted by the
   lower bound weight.a of the solution quality
   To show that the result does not depend on assumptions
   about handling point sets which are not in general position,
   degenerate solutions (triangles with three points on a line)
   are considered, but they are flagged as degenerate."""
opt = {} ## array of optimal (partial) solutions.
n = len(p)
for i in range(n-1): # initialization
  o = arithmetic.Interval_integer(0)
          ## calculate the weight only of interior edges
  o.info = None # backtracking information and flag is stored
               # within the Interval_integer objects
  o.degenerate_flag = False
  opt[i,i+1] = [o]
  # opt[i,j] stores a LIST of all potential candidate solutions.

for l in range(2,n): # l="length" of the edges
  for i in range(n-l):
    j = i+l
    # check for triangulations containing edge (i,j):
    if (i,j) in excluded or (j,i) in excluded:
      ## we are interested in MWT that does not contain those edges
      continue
    if p[i]==p[j]: # this point appears twice on the boundary
      continue
    if (i,j) == (0,n-1):
      edgelength = 0 # final edge is not counted
    else:
      edgelength = arithmetic.length(p[i],p[j])
    for k in range(i+1,j):
      if opt.has_key((i,k)) and opt.has_key((k,j)):
        ori = orientation(p[i],p[k],p[j])
        if ori >= 0:
          in1 = opt[i,k]
          in2 = opt[k,j]
          allcand = [ computesol(in1,in2,n1,n2,k,ori==0,edgelength)
                         for n1 in range(len(in1))
                         for n2 in range(len(in2)) ]
          opt[i,j] = arithmetic.keep_best(opt.get((i,j),[])+allcand)
# sort final result by lower bounds of intervals:
allopt = [(x.a,x) for x in opt[0,n-1]]
allopt.sort()
opt[0,n-1] = [x[1] for x in allopt]
# retrace optimal solution:
return [ (arithmetic.Interval_integer(x.a,x.b),
                     retrace(opt,0,n-1,ind), x.degenerate_flag)
```

```
          for x,ind in zip( opt[0,n-1], range(len(opt[0,n-1])) ) ]

def orientation(p1,p2,p3):
  x2 = p2[0]-p1[0]
  y2 = p2[1]-p1[1]
  x3 = p3[0]-p1[0]
  y3 = p3[1]-p1[1]
  return x2*y3-x3*y2

def computesol(l1,l2,i1,i2,k,flag,edgelength):
    r = l1[i1] + l2[i2] + edgelength
    r.info = (k,i1,i2) # k=splitting point.
                       # i1,i2: pointers into sublists
    r.degenerate_flag = (flag or l1[i1].degenerate_flag
                              or l2[i2].degenerate_flag)
    return r

def retrace(opt,i,j,ind):
  """recursively recover the solution, based on the "info"
     fields stored with the solution values in the array "opt".
     Recover the solution for position "ind" in the list "opt".
     The result is a list of triplets (i,k,j), each representing
     the two edges p[i]p[k] and p[k]p[j]."""
  if opt[i,j][ind].info:
    k,i1,i2 = opt[i,j][ind].info
    return [(i,k,j)]+retrace(opt,i,k,i1)+retrace(opt,k,j,i2)
  else:
    return []
```

B.3 Checking the β-skeleton

This procedure is straightforward: for every edge pq of the boundary, it runs through all remaining points r and checks whether they violate the β-skeleton condition, by calculating the (squared) cosine of the angle $\alpha = prq$ with the cosine law. The running time is $O(n)$ per edge of the boundary, thus at most $O(n^2)$ in total. The relation between the angle α and the ratio β between the diameter of the circumcircle of pqr and the distance $|pq|$ is given by

$$\sin\alpha = 1/\beta, \text{ or } \cos^2\alpha = 1 - 1/\beta^2$$

To understand this program, one has to know how the pieces are given. Each `piece` is represented as a dictionary. The field `piece["partlist"]` is a sequence of (names of) boundary parts, usually just `("lower", "upper")`; only the C-connection has three parts. Then `piece["lower"]` and `piece["upper"]` (or whatever the names in the `"partlist"` are) are the actual boundary parts, as lists of points. Each point is a pair of precise `Decimal` numbers as provided by the `decimal` package.

```
from math import sqrt
from checking_routines import smallest_unit
from basic_routines import map_coordinates
```

```
beta_threshold = 1.18  # ">=1.17683 !"
betaprecision = 10**7

def coordinate_to_int(c):
    "scale coordinate to integer"
    return int(c/smallest_unit) # smallest_unit=0.0001

def length2(p):
    """squared length of vector"""
    return p[0]**2 + p[1]**2

def checkbeta(piece):
  min_cos_squared = 100*betaprecision # infinity
  for part in piece["partlist"]:
    # scaling all coordinates does not affect beta-skeleton:
    poly = map_coordinates(coordinate_to_int, piece[part])
    for i in range(len(poly)-1): # run through all edges:
      p1 = poly[i]
      p2 = poly[i+1]
      # check edge (p1,p2):
      # run through all points p:
      for part_2 in piece["partlist"]:
        for p in map_coordinates(coordinate_to_int,piece[part_2]):
          if p<>p1 and p<>p2:
            # compute cos_squared, representing the
            # squared cosine of the angle alpha=p1,p,p2
            x1 = (p1[0]-p[0], p1[1]-p[1])
            x2 = (p2[0]-p[0], p2[1]-p[1])
            innerproduct = x1[0]*x2[0] + x1[1]*x2[1]
            # so far, all calculations were exact.
            if innerproduct>0:
               cos_squared = ( innerproduct**2 * betaprecision /
                                       (length2(x1)*length2(x2)) )
               # scaled by betaprecision and rounded down;
               # yields a conservative (lower) estimate of beta
            min_cos_squared = min(min_cos_squared,cos_squared)
  min_cos_squared_unscaled = min_cos_squared/float(betaprecision)
  beta = 1/sqrt(1-min_cos_squared_unscaled)
  # This final calculation is just in double-precision floating-point.
  print "cos(alpha)^2 = %d/%d = %8.6f; beta = %8.6f." % (
    min_cos_squared, betaprecision, min_cos_squared_unscaled, beta)
  if beta < beta_threshold:
    print "beta too small: ", beta, ">=1.17683 !", beta_threshold
    raise RuntimeError
  return beta
```

B.4 Proving Lemma 5.1

This program runs through all 21 subpolygons of the polygon W in the proof of Lemma 5.1. For illustration, we give here the output produced for the first case:

```
Case v1, v1': difference =
```

```
4.00304
          Best solution value without terminal edges:
              [202.72577,202.72594]
          There is/are 2 best solution(s) without restriction:
              [198.72255,198.72273]
              [198.72255,198.72273]
```

There is an extended version (not shown here) which also generates a drawing of the triangulations, and which has been used to generate Table II. The module `solve_all_patterns.py`, by which the data of Appendix A were generated, is similar.

```
# checks presence of one ot two "terminal" edges in thin wire

from basic_routines import map_coordinates
import arithmetic
from arithmetic import print_scaled_int, decimal_to_scaled_int
import mwt_polygon

from pieces import W_upper, W_lower, W_point_x, W_point_y, W_point_z
# W_upper and W_lower form the set W.

def check_W_cases():
   mindiff = 999 * arithmetic.scale # infinity

   for i in range(1,7):
     for j in range(i,7): #  for 1<=i<=j<=6:

       p = W_lower[i:len(W_lower)-j] + W_upper
       i_x = p.index(W_point_x)
       i_y = p.index(W_point_y)
       i_z = p.index(W_point_z)
       excl = [(i_x,i_y),(i_x,i_z)] # critical edges

       pcalc = map_coordinates(decimal_to_scaled_int, p)

       # First, compute optimum solution without restriction
       elist = mwt_polygon.mwt(pcalc)

       for (result,sol,degenerate) in elist:
          # convert list of triplets to list
          # of pairs representing edges:
          sole = [(x,y) for [x,y,z] in sol] + \
                 [(y,z) for [x,y,z] in sol]
          if not((i_x,i_y) in sole or (i_x,i_z) in sole or
                 (i_y,i_x) in sole or (i_z,i_x) in sole):
              print "incorrect:",i,j,result,sol,degenerate
              print ("optimal solution does not contain" ,
                        i_x,i_y, "or", i_x,i_z)
              raise StandardError
          if degenerate:
              print "DEGENERATE SOLUTION:",i,j,result,sol,degenerate
```

```
            raise StandardError

        # Now, compute optimum solution with the two edges excluded:
        elist2 = mwt_polygon.mwt(pcalc, excluded=excl)

        # consider just the first solution from the list:
        diff = elist2[0][0].a - max(result.b for (result,s,f) in elist)
        mindiff = min(mindiff,diff)

        print "Case v%d, v%d': difference =" % (i,j)
        print print_scaled_int(diff)
        print " "*10+"Best solution value without terminal edges:"
        print_scaled_interval(elist2[0][0])
        print "          There is/are",len(elist),
        print    "best solution(s) without restriction:"
        for result,sol,f in elist: print_scaled_interval(result)
        print
    print "Minimum overall difference =", print_scaled_int(mindiff)

def print_scaled_interval(x):
 print " "*14+"[%s,%s]"%(print_scaled_int(x.a), print_scaled_int(x.b))

print "Program check_W.py. Check terminal edges xy and xz in W"
arithmetic.setprecision(5) # 5 digits are sufficient.
check_W_cases()
```