Computational Geometry ••• (••••) •••–•••

Contents lists available at ScienceDirect

Computational Geometry: Theory and Applications

www.elsevier.com/locate/comgeo

Optimization for first order Delaunay triangulations ☆

Marc van Kreveld, Maarten Löffler *, Rodrigo I. Silveira

Department of Information and Computing Sciences, Utrecht University, 3508TB Utrecht, The Netherlands

ARTICLE INFO

Article history:
Received 26 September 2007
Received in revised form 2 October 2008
Accepted 27 January 2009
Available online xxxx

Keywords:
Delaunay triangulation
Higher order Delaunay triangulation
Optimization

ABSTRACT

This paper discusses optimization of quality measures over first order Delaunay triangulations. Unlike most previous work, our measures relate to edge-adjacent or vertex-adjacent triangles instead of only to single triangles. We give efficient algorithms to optimize certain measures, including measures related to the area ratio of adjacent triangles, angle between outward normals of adjacent triangles (for polyhedral terrains), and number of convex vertices. Some other measures are shown to be NP-hard. These include maximum vertex degree, number of convex edges, and number of mixed vertices. For the latter two measures we provide, for any constant $\varepsilon > 0$, factor $(1-\varepsilon)$ approximation algorithms that run in $2^{O(1/\varepsilon)} \cdot n$ and $2^{O(1/\varepsilon^2)} \cdot n$ time (when the Delaunay triangulation is given). For minimizing the maximum vertex degree, the NP-hardness proof provides an inapproximability result. Our results are presented for the class of first order Delaunay triangulations, but also apply to triangulations where for every triangle at least two edges are fixed. The approximation result on maximizing the number of convex edges is also extended to k-th order Delaunay triangulations.

1. Introduction

Triangulation is a well-studied topic in computational geometry. The input is a point set or planar straight line graph in the plane, and the objective is to generate a subdivision where all faces are triangles, except for the outer face. In some cases extra points are allowed, in which case we speak of a Steiner triangulation. Since a point set (or planar straight line graph) allows many different triangulations, one can try to compute one that optimizes a criterion. For example, one could maximize the minimum angle used in any triangle, or minimize the total edge length (minimum weight triangulation). The former optimization is solved with the Delaunay triangulation in $O(n \log n)$ time for n points. The latter optimization is NP-hard [19].

Several other optimization measures exist. In finite element methods, triangular meshes with various quality constraints are used, and Steiner points may be used to achieve this. See Bern and Plassmann [4] for a survey. Other optimization measures arise if the triangulation represents a terrain (in which case it is called a polyhedral terrain in computational geometry): all vertices have a specified height, and the height of points on edges and on triangles is obtained by linear interpolation. Such a terrain representation is common in GIS and is called a TIN [6,26].

Bern et al. [3] show that measures such as minimum triangle height, maximum slope, and maximum eccentricity of any triangle can be optimized with a technique called *edge insertion*. The technique yields $O(n^3)$ or $O(n^2 \log n)$ time algorithms. Other measures such as maximum angle [9] and maximum edge length can also be minimized in polynomial time [8].

☆ This research has been partially funded by the Netherlands Organisation for Scientific Research (NWO) under FOCUS/BRICKS grant number 642.065.503 (GADGET) and under the project GOGO.

* Corresponding author.
E-mail addresses: marc@cs.uu.nl (M. van Kreveld), loffler@cs.uu.nl (M. Löffler), rodrigo@cs.uu.nl (R.I. Silveira).

0925-7721/$ – see front matter
doi:10.1016/j.comgeo.2009.01.010

Please cite this article in press as: M. van Kreveld et al., Optimization for first order Delaunay triangulations, Computational Geometry (2009), doi:10.1016/j.comgeo.2009.01.010

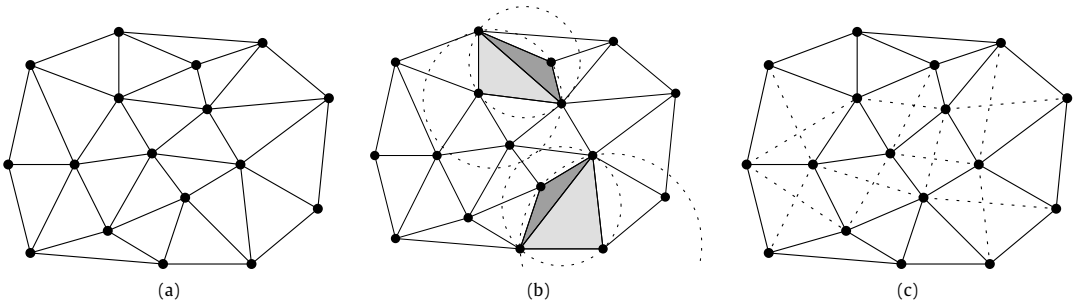


Fig. 1. (a) Delaunay triangulation (zero-th order). (b) Second order Delaunay triangulation (light grey triangles are first order, medium grey triangles are second order). (c) Structure of first order Delaunay triangulations: one of every pair of intersecting edges must be chosen.

Interestingly, the Delaunay triangulation optimizes several measures simultaneously: it maximizes the minimum angle, and it minimizes the maximum circumscribed circle, maximum smallest enclosing circle, and the integral of the gradient squared (e.g. [3]).

For terrain modeling in GIS [6,26], Steiner points cannot be used because their elevation would not be known. Terrain modeling leads to a number of optimization criteria, both to yield good rendering of the terrain for visualization, and to make it suitable for modeling processes like water runoff and erosion [13,18,25]. The terrain characteristics *slope* and *aspect* are especially important. Furthermore, local minima and artificial dams, which may be artifacts due to the creation of the triangulation, should be avoided [7,14,17,24] (a vertex is a local minimum if all neighboring vertices are higher).

The *Delaunay triangulation* of a set P of points is defined as the triangulation where all vertices are points of P and the circumcircle of the three vertices of any triangle does not contain any other point of P. If no four points of P are cocircular, then the Delaunay triangulation is uniquely defined. Gudmundsson et al. [10] define *higher order Delaunay triangulations*, a class of triangulations where a few points are allowed inside the circumcircles of triangles. A triangulation is *k-th order Delaunay* if the circumcircle of the three vertices of any triangle contains at most k other points (see Fig. 1). First order Delaunay triangulations have a special structure. If we take all edges that are certain to be in the first order Delaunay triangulation, then the resulting subdivision only has triangles and convex quadrilaterals (and an unbounded face). In the convex quadrilaterals, both diagonals are possible to obtain a first order Delaunay triangulation. We call these diagonals *flippable*, and similarly we call the quadrilateral *flippable*. Due to their special structure, measures like the number of local minima or extrema can be minimized in $O(n \log n)$ time. The same holds for minimizing the maximum area triangle, minimizing the total edge length, and various other measures. On the other hand, minimizing the maximum vertex degree was only approximated by a factor of roughly 1.5 [10].

For higher order Delaunay triangulations, fewer results are known. Minimizing local minima in a terrain becomes NP-hard for orders higher than n^{ε}, where $\varepsilon > 0$ is any constant [7]. Experiments showed that low order Delaunay triangulations can reduce the number of local minima significantly. For first order Delaunay triangulations, the reduction is already 15–20% with respect to the Delaunay triangulation on natural terrains, and for fourth order a heuristic achieved reductions of roughly 50% [7].

Most of the measures mentioned above are measures for single triangles. Exceptions are total edge length, number of local minima or extrema, and maximum vertex degree. In this paper, we consider measures that depend on pairs of triangles that are edge-adjacent, and measures that depend on groups of triangles that are vertex-adjacent. Notice that a single flip in a first order Delaunay triangulation influences five pairs of edge-adjacent triangles and four vertex-adjacent groups. The edge insertion paradigm [3] cannot be used for such problems, because it relies on incremental improvement of the worst situation that occurs in a single triangle.

We consider objectives of the maxmin or minmax type, but also objectives where the number of undesirable situations must be minimized. An example of a minmax problem for edge-adjacent triangles is minimizing the maximum ratio of edge-adjacent triangle areas:

$$\min_{T} \max_{e \in T} \left(\frac{\text{maxarea}(t, t')}{\text{minarea}(t, t')} \right), \quad \text{where } e \in \partial t \text{ and } e \in \partial t' \text{ and } T \text{ is first order Delaunay.}$$

This measure may be relevant for mesh generation for numerical methods. For polyhedral terrains, an example in the same class is minimizing the maximum spatial angle of the normals of edge-adjacent triangles. This measure is important for good slope characteristics, needed for flow modeling. Geomorphologists classify parts of mountains or hills as footslopes, hillslopes, valley heads, etc. [13]. Certain types of classes are characterized by terrain convexity or concavity. If we know that a part of a terrain is a valley head, we should maximize the number of convex edges or convex vertices in that part. An edge of a polyhedral terrain is *convex* if there is a plane containing the edge such that both triangles adjacent to the edge are on or below the plane, with one of the triangles with a vertex strictly below it. Analogously, a vertex is *convex* if there is a plane through that vertex such that all of its neighbors are on or below that plane, and at least one strictly below. The definition of *concave* edges and vertices is analogous. A vertex is *mixed* if every plane containing it has neighbors strictly

Please cite this article in press as: M. van Kreveld et al., Optimization for first order Delaunay triangulations, Computational Geometry (2009), doi:10.1016/j.comgeo.2009.01.010

Table 1
Optimization problems and complexity results for first order Delaunay triangulations. d is the maximum vertex degree in the Delaunay triangulation.

Triangles incident to	Opt. worst local measure (minmax)	Result	Opt. # occurrences	Result
edge	area ratio	$O(n \log n)$	max # convex edges	NP-hard
	angle of outward normals	$O(n \log n)$		
vertex	area ratio	$O(nd \log n)$	max # convex vertices	$O(n \log n)$
	angle of outward normals	$O(nd \log n)$	min # local minima	$O(n \log n)$ [10]
	vertex degree	NP-hard	min # mixed vertices	NP-hard

above and below the plane. We study maximization of the number of convex edges, maximization of the number of convex vertices, and minimization of the number of mixed vertices.

Given a planar point set P with or without elevation, we study the complexity of optimizing measures over all first order Delaunay triangulations. Measures we consider are shown classified in Table 1, which also shows our results. The optimization of other worst local measures for edge-adjacent triangles can also be solved in $O(n \log n)$ time with the same technique, like minimizing the largest minimum enclosing circle of any two edge-adjacent triangles.

Our proof of NP-hardness of minimizing the maximum vertex degree justifies the factor 1.5 approximation algorithm given before in [10]. While it was already known that triangulating a biconnected planar graph while minimizing the maximum degree is NP-hard [15], our proof also proves that no polynomial-time approximation algorithm for this problem with approximation guarantee below a certain constant greater than 1 exists, unless P = NP. We further note that minimizing the number of local minima and extrema in polyhedral terrains can be solved in $O(n \log n)$ time [10].

The NP-hard problems of maximizing the number of convex edges and maximizing the number of non-mixed vertices can be approximated — for any constant $\varepsilon > 0$ — within a factor of $1 - \varepsilon$ in $2^{O(1/\varepsilon)} \cdot n$ and $2^{O(1/\varepsilon^2)} \cdot n$ time, if the Delaunay triangulation is given. We show this using bounded width tree decompositions. The approximation algorithm to maximize convex edges can be generalized to k-th order Delaunay triangulations. The running time becomes $2^{2^{O^*(k)}} 2^{O^*(\frac{1}{\varepsilon^2})} n$. This constitutes an interesting theoretical result, given that very little is known about optimization of higher order Delaunay triangulations for arbitrary orders.

The NP-hardness results show that, despite the simple structure of first order Delaunay triangulations, optimization of various measures is hard. They are the first NP-hardness results for first order Delaunay triangulations. The status of these problems over the class of all triangulations of a point set is open. All of our results also apply to what we could call *singly-flippable triangulations*: triangulations in which only the edges from a designated subset may be flipped, and no triangle is incident to more than one flippable edge. This implies that our techniques are not restricted to any Delaunay-related criterion.

The remainder of this paper is organized as follows. Section 2 gives the $O(n \log n)$ and $O(nd \log n)$ time algorithms of Table 1, Section 3 gives the NP-hardness proofs, and Section 4 gives the $(1 - \varepsilon)$-approximation algorithms. In this paper we assume non-degeneracy of the input set P of points: no four points are cocircular.

2. Exact algorithms

We start this section with a problem that turns out to be surprisingly easy to solve, namely, maximizing the number of convex vertices over all possible first order Delaunay triangulations. Let P be a set of n points in the plane, where each point has a height value. As observed before, if we take the Delaunay triangulation T of P, it has a number of edges that are in any first order Delaunay triangulation, and a number of flippable edges, and no two flippable edges bound the same Delaunay triangle [10]. The Delaunay triangulation and its flippable edges can be determined in $O(n \log n)$ time.

For any flippable quadrilateral, one diagonal is reflex and the other diagonal is convex in 3-dimensional space, unless the four vertices of the quadrilateral are co-planar. Consider a convex vertex v in T. If it is incident to a flippable quadrilateral where the convex diagonal is present, then v will remain convex if we use the reflex diagonal instead (regardless of which diagonal is incident to v). In other words: using only reflex edges in flippable quadrilaterals does not cause any vertex to become non-convex. At the same time, it may turn non-convex vertices into convex ones. It follows that the maximization problem can be solved in linear time once the flippable quadrilaterals have been identified, by simply selecting the reflex flippable edge in every flippable quadrilateral. Note that the flippable quadrilaterals can be easily identified in linear time, given the Delaunay triangulation.

Theorem 1. *A first order Delaunay triangulation that maximizes the number of convex vertices can be computed in $O(n \log n)$ time.*

2.1. Measures on edge-adjacent triangles

In this section we show how to optimize a measure function M defined for a triangulation T, over all first order Delaunay triangulations of P. The function M should be of the shape $M(T) = \max_{q \in T} \mu(q)$, where q is a (not necessarily flippable) quadrilateral, and μ is a function that assigns a value to each quadrilateral. The goal is to minimize $M(T)$ over all first order

Please cite this article in press as: M. van Kreveld et al., Optimization for first order Delaunay triangulations, Computational Geometry (2009), doi:10.1016/j.comgeo.2009.01.010

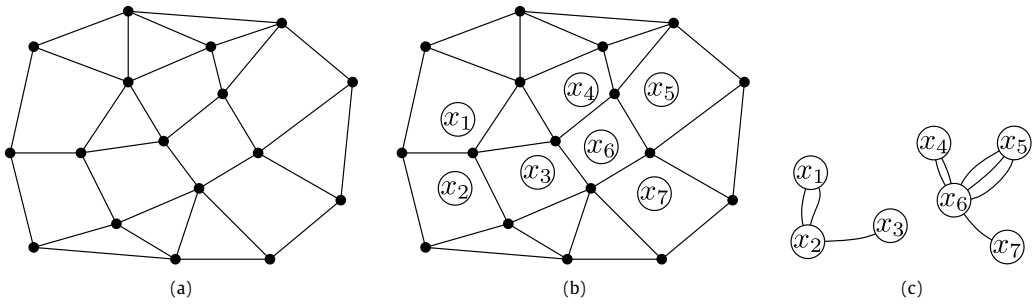


Fig. 2. (a) The fixed first order Delaunay edges and flippable quadrilaterals. (b) Variables for the quadrilaterals. (c) Clauses that make a 2-SAT instance.

Delaunay triangulations T of the given point set. We also use $\mu(e)$ for any edge e in a triangulation to denote $\mu(q)$, where e is the diagonal of q.

We begin by observing that a first order Delaunay triangulation has four types of edges: between two fixed triangles, between a fixed triangle and a flippable quadrilateral, between two flippable quadrilaterals, and flippable edges. As a consequence, there are only $O(n)$ possible values for $M(T)$, and we can determine and sort them in $O(n \log n)$ time.

In order to solve the $\min M(T)$ problem, we transform it into a series of 2-SAT instances. We will use 2-SAT to answer the following question: Is there a first order Delaunay triangulation T such that $M(T) \leqslant \mu_0$? Since there are $O(n)$ interesting values for μ_0, we can apply binary search to find the smallest one. This can be achieved efficiently because the function we are optimizing is local to quadrilaterals, that is, $M(T) = \mu(q)$, for some quadrilateral q. Since the number of quadrilaterals (either flippable or not) is linear in n, the set of all the possible values for $M(T)$ can be computed in linear time once the flippable quadrilaterals have been identified. After some elimination of choices of diagonals, we will model the flippable quadrilaterals by variables, and the two diagonals will be its truth assignments. We describe the elimination first.

Let S be the subdivision that is the Delaunay triangulation of P with all flippable edges removed, see Fig. 2(a), and let μ_0 be given. For every edge e of S between a triangle and a quadrilateral, we decide which of the two diagonals of the quadrilateral induces $\mu(e) > \mu_0$. If both do, then we can immediately answer the question with "no". If only one diagonal has $\mu(e) > \mu_0$, then we fix the other diagonal in S. Otherwise, we continue with the next edge between a triangle and a quadrilateral. This step may have made flippable quadrilaterals into two fixed triangles in S. Next we test the possible diagonals of each quadrilateral q of S. If both diagonals give $\mu(q) > \mu_0$, then we answer with "no" again. If only one diagonal gives $\mu(q) > \mu_0$, then we fix the other diagonal to make two new triangles in S. Next we test all edges of S between adjacent fixed triangles. If any such edge gives $\mu(e) > \mu_0$, then we answer the question with "no" again.

It remains to solve the problem for edges between quadrilaterals of S. For every quadrilateral q, we introduce a Boolean variable x_q as shown in Fig. 2(b), and let one diagonal choice represent TRUE and the other FALSE. Let e be an edge of S between two quadrilaterals q and r. For each choice of diagonals in q and r that gives $\mu(e) > \mu_0$, for example the one with TRUE in q and FALSE in r, we make a clause $(\neg x_q \vee x_r)$, see Fig. 2(c). We get at most four clauses for any edge between two quadrilaterals, hence there are $O(n)$ clauses overall. The conjunction of all clauses is a 2-SAT instance.

The satisfiability of the constructed 2-CNF expression (together with a satisfying truth assignment, if it exists) can be found in linear time with the algorithm of Aspvall et al. [1], hence the original question can also be answered within this time bound.

The optimization algorithm proceeds by doing binary search on the possible values for $M(T)$ until it finds the optimal one, which requires at most $O(\log n)$ steps. Each step involves constructing and solving an instance of 2-SAT, which can be done in linear time; hence, the overall running time of the algorithm is $O(n \log n)$.

Once the optimal value of $M(T)$ is found, we take the corresponding 2-SAT solution and compute the assignment of diagonals for each flippable quadrilateral accordingly, resulting in an optimal first order Delaunay triangulation.

Theorem 2. *A first order Delaunay triangulation that minimizes the maximum area ratio of edge adjacent triangles can be computed in $O(n \log n)$ time. If the triangulation represents a polyhedral terrain, the same result holds for minimizing the maximum angle of outward normals.*

2.2. Measures on vertex-adjacent triangles

The algorithm described in the previous section can easily be extended to minimize measure functions of the form $M(T) = \max_{t,t' \in T} \mu(t, t')$ for t and t' triangles in T with a common vertex. Let $d(v)$ denote the degree of v in the Delaunay triangulation. The maximum degree that v can have, in any first order Delaunay triangulation, is $2d(v)$. Hence the set of possible values of $M(T)$ induced by pairs of triangles incident to a vertex v is $\binom{2d(v)}{2}$. Since the sum of the degrees of all vertices is $O(n)$, the total number of possible values of $M(T)$ is upper-bounded by $\sum_{v \in T} 2d(v)^2 = 2d \cdot \sum_{v \in T} d(v) = O(dn)$, where d is the maximum degree of any vertex in the triangulation. It follows that this type of optimization problem can be solved in $O(nd \log n)$ time.

Please cite this article in press as: M. van Kreveld et al., Optimization for first order Delaunay triangulations, Computational Geometry (2009), doi:10.1016/j.comgeo.2009.01.010

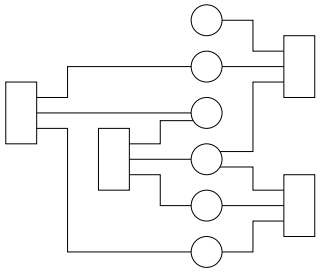


Fig. 3. An instance of planar 3-SAT is an embedded graph that consists of a set of variables (the circles), a set of clauses (the boxes), and an edge between a variable and a clause if the variable occurs in that clause.

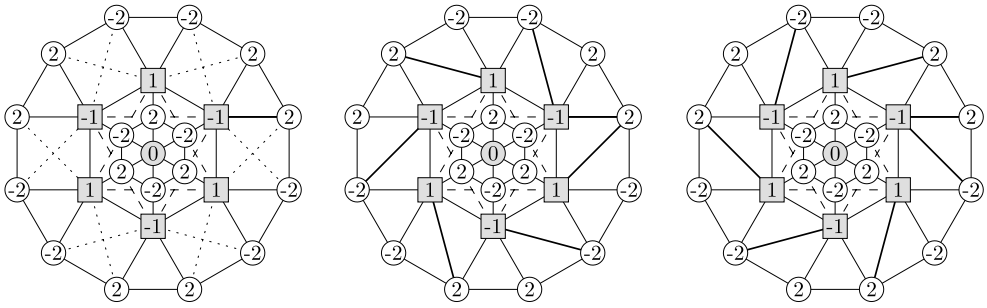


Fig. 4. A fan gadget. (a) The structure of the fan. The solid edges are fixed; the dotted and dashed edges are first order Delaunay edges. Only the dotted edges play a role in the construction. (b) One solution is a left-turning fan. (c) The other solution is a right-turning fan.

Theorem 3. *A first order Delaunay triangulation that minimizes the maximum area ratio of vertex adjacent triangles can be computed in* $O(nd \log n)$ *time, where* d *is the maximum vertex degree in the Delaunay triangulation. If the triangulation represents a polyhedral terrain, the same result holds for minimizing the maximum angle of outward normals.*

3. NP-hardness results

In this section we show NP-hardness for three different optimization problems on first order Delaunay triangulations.

3.1. Mixed vertices

Suppose we have a triangulated terrain, that is, a triangulation where every vertex has an elevation attribute. In such a terrain, we call a vertex *mixed* if there exists no plane through this vertex such that all neighboring vertices are on one side of the plane. In real terrains, such mixed vertices are uncommon, so we want to minimize their number.

Problem 1. Given a set of points with elevation information, construct a first order Delaunay triangulation of this point set such that the number of mixed vertices is minimum.

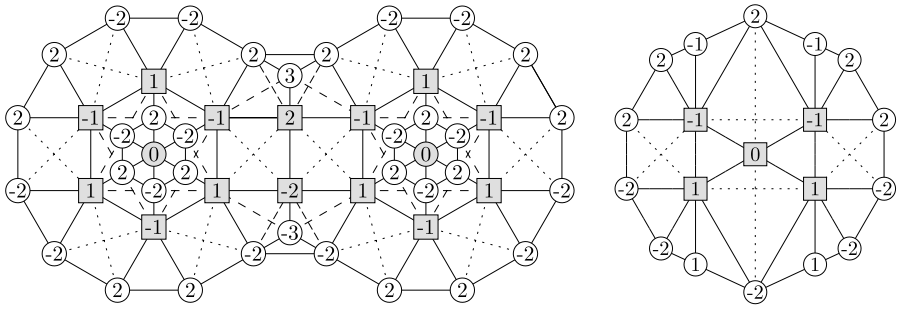
This problem is NP-hard. We prove this by reduction from planar 3-SAT [16]. Fig. 3 depicts a typical planar 3-SAT instance.

We represent the variables occurring in a 3-SAT instance by *fan*-gadgets, as in Fig. 4(a). A fan gadget consists of 25 points with a certain elevation. In the figure, all possible first order Delaunay edges are shown. Solid edges are in every first order Delaunay triangulation; dashed and dotted edges are flippable. The square nodes and the dotted edges are the most important part. We make the following observation:

Observation 1. A square vertex is mixed if and only if both incident dotted edges are in the triangulation.

Furthermore, the gadget is constructed in such a way that the state of the round vertices does not depend on any of the dotted edges. The white round vertices are always non-mixed, even if all possible incident edges would be in the triangulation; the gray round vertex is always mixed, already if only the fixed edges are in the triangulation. This implies that the number of mixed vertices is only affected by square vertices, and can only be minimal if there are never two dotted edges incident to the same square vertex. A fan-gadget therefore has two possible states, see Figs. 4(b) and 4(c).

Please cite this article in press as: M. van Kreveld et al., Optimization for first order Delaunay triangulations, Computational Geometry (2009), doi:10.1016/j.comgeo.2009.01.010



(a) (b)

Fig. 5. (a) Connecting variables. (b) An inverter gadget.

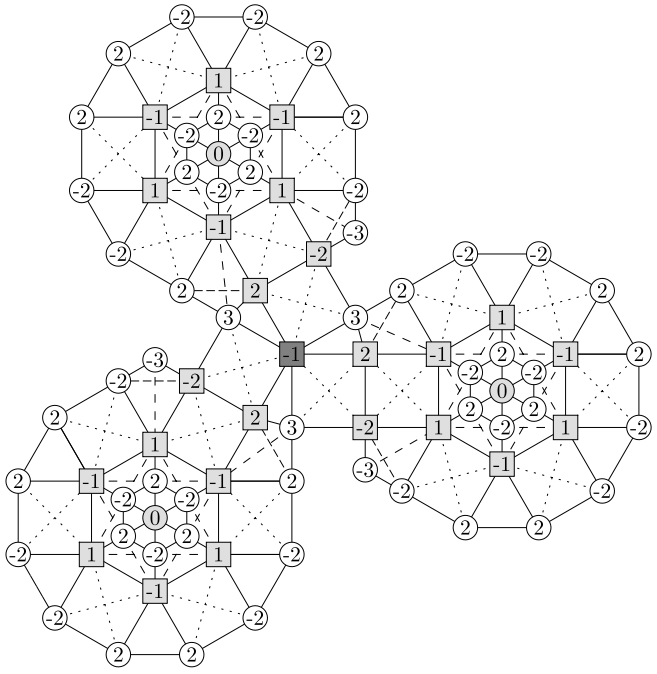


Fig. 6. Three variables come together in a clause.

We can connect fans together to form larger chains that are all in the same state, see Fig. 5(a). We turned two more vertices into square vertices, and if the left fan is left-turning, the right fan must also be left-turning and the other way around. We can connect up to three fans to an existing fan, so chains can also branch.

We also need to make negators in chains; for this we use the inverter gadget shown in Fig. 5(b). Here, if the leftmost flippable quadrilateral has its positive sloping diagonal in the triangulation, the rightmost flippable quadrilateral must have its negative sloping diagonal and the other way around. We can incorporate an inverter gadget in a chain to change its value.

We represent the clauses occurring in the 3-SAT instance by a special clause vertex, see Fig. 6. Here three fan chains come together at one square vertex shown in a darker shade of gray. This vertex has a slightly different property than the other square vertices.

Observation 2. A clause vertex is mixed if and only if all three incident dotted edges are in the triangulation.

Therefore the clause can be satisfied if at least one of the three fans is *not* right-turning, and by including inverters at the appropriate places this can represent any Boolean clause.

With these gadgets we can build the whole planar 3-SAT instance, by following the planar graph as in Fig. 3. Note that this planar graph may contain cycles, and our gadgets may not fit exactly when laying them out as described. This problem can be solved by slightly deforming the construction. The gadgets have some flexibility in them: a variable ring, for instance,

Please cite this article in press as: M. van Kreveld et al., Optimization for first order Delaunay triangulations, Computational Geometry (2009), doi:10.1016/j.comgeo.2009.01.010

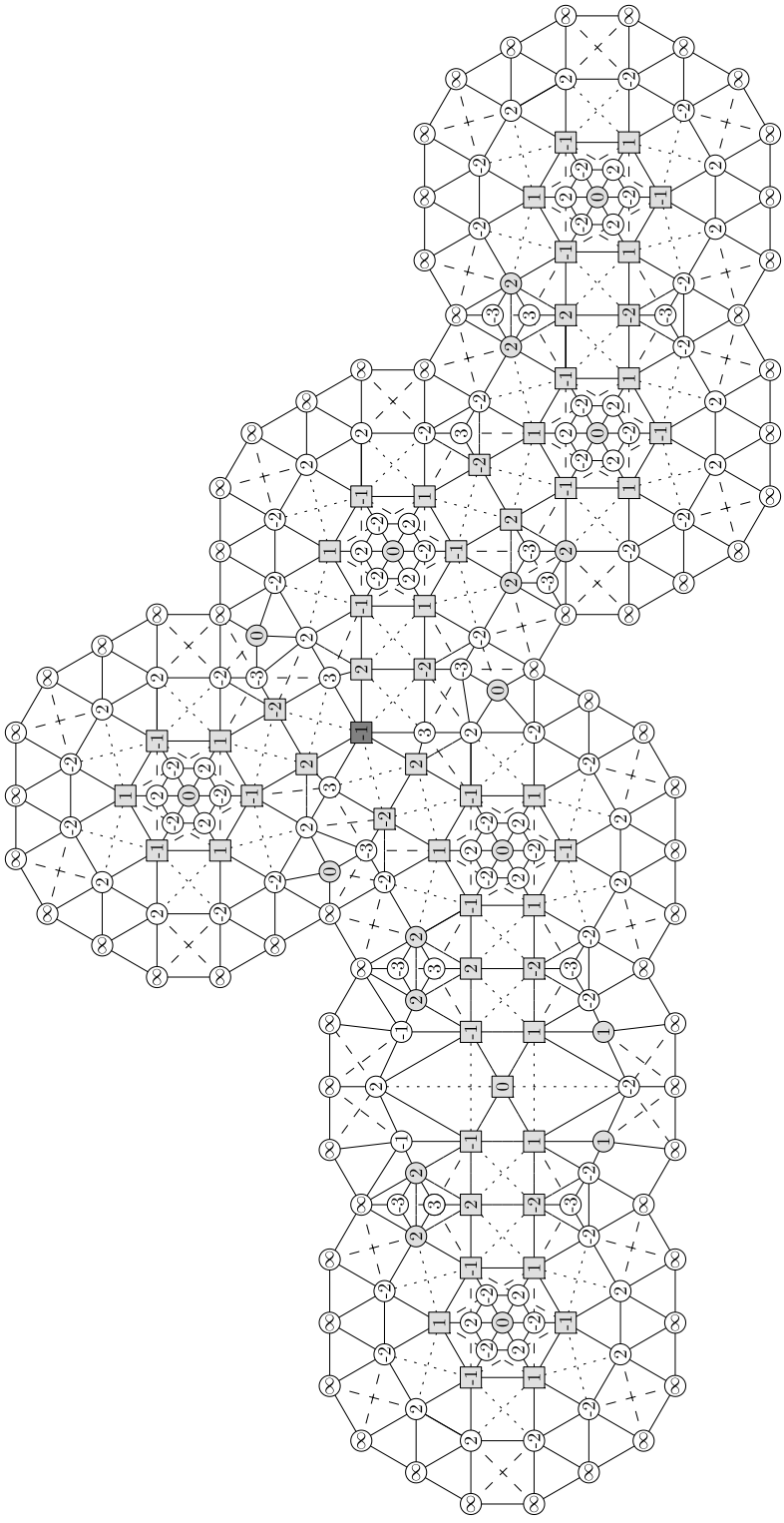


Fig. 7. A coating to shield the construction from interference from outside.

can be stretched up to a small constant without damaging the required properties. By making the chains long enough (the length depends only on this constant, not on n), we can shift the end of a chain around enough to make it fit exactly on the clause it needs to be attached to.

Please cite this article in press as: M. van Kreveld et al., Optimization for first order Delaunay triangulations, Computational Geometry (2009), doi:10.1016/j.comgeo.2009.01.010

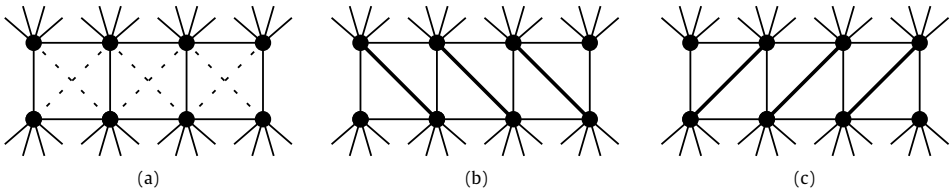


Fig. 8. (a) A variable chain. (b) One state. (c) The other state.

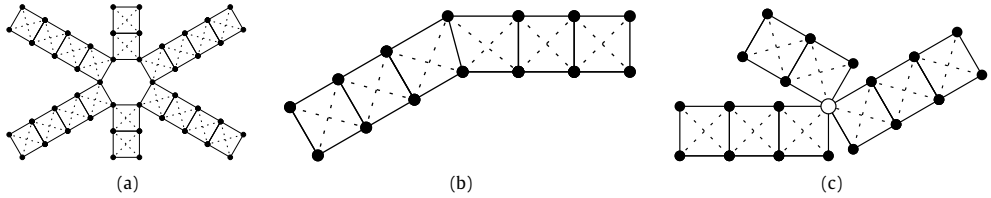


Fig. 9. (a) Variable chains can be branched. (b) Chains can turn over angles of 30°. (c) A clause gadget.

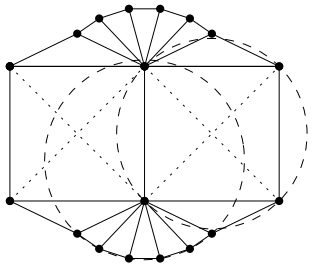


Fig. 10. We add fixed triangles to increase the degree.

Finally, we need to triangulate the remaining gaps; hence, we need to ensure that the vertices on the boundary really have a fixed status. To do this, we add an extra layer of vertices at a very high elevation, see Fig. 7. The vertices need not really be infinitely high, just high enough. Now these vertices will all be non-mixed, and for the vertices that are not on the boundary their properties can just be checked locally.

Theorem 4. *Minimizing the number of mixed vertices over all first order Delaunay triangulations is NP-hard.*

3.2. Triangles incident to a vertex

Problem 2. Given a set of points, construct a first order Delaunay triangulation of the point set such that the maximum vertex degree is minimum.

This problem is NP-hard. We prove this by reduction from planar 3-SAT [16]. Given a 3-SAT instance, we will construct a point set such that a triangulation with maximum vertex degree d exists if and only if the instance is satisfiable, for some suitably chosen d.

We represent variables by chains of flippable quadrilaterals, see Fig. 8(a). The idea is that the vertices of the quadrilaterals have $d-1$ fixed edges, so only one of the two diagonals incident to each vertex is allowed to be in the triangulation. This implies that there are only two possible triangulations of the whole chain.

We can also branch these chains, by using a ring of quadrilaterals, see Fig. 9(a). In the ring, all flippable diagonals must either be situated in clockwise or in counterclockwise direction. To turn the chains over small angles, we just deform some quadrilaterals slightly as in Fig. 9(b). Finally we represent clauses by a special clause vertex that has $d-2$ fixed outgoing edges, see Fig. 9(c). This means that only two of the flippable diagonals can be in the triangulation.

These gadgets are sufficient to build the whole 3-SAT instance. However, we need to include some more points to ensure that the degree of each vertex is exactly d. For normal vertices there are always two flippable edges, so we need to include exactly $d-1$ fixed edges. By including these, we need to ensure that the flippable edges remain flippable and the fixed edges are really fixed. In Fig. 10 we see how we can achieve this. If we have a horizontal chain of quadrilaterals, we add one point beneath the middle of each quadrilateral, and one point above it, such that they are just outside the circle through the four corners of the quadrilateral. This ensures that the four possible triangles inside the quadrilateral are still first order Delaunay triangles. Then, for each vertex, we add $d-6$ more points on a circle that goes through the two newly added

Please cite this article in press as: M. van Kreveld et al., Optimization for first order Delaunay triangulations, Computational Geometry (2009), doi:10.1016/j.comgeo.2009.01.010

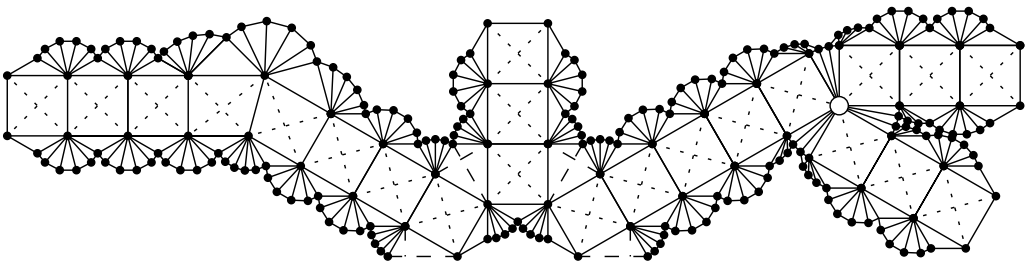


Fig. 11. A shiny coating of fixed triangles.

neighbors of this vertex and that contains this vertex and its opposing vertex in its interior. If we would flip any of the edges of the resulting triangulation, there would be a face that has its three corners on this circle, and thus would not be first order Delaunay.

Similar constructions can be used on the other gadgets, so the new triangulation will look like the one shown in Fig. 11. The only problem occurs in the branch gadget, in the middle of the figure. Here some new flippable quadrilaterals are introduced, marked by the dashed edges. However, if we just make sure that the inner vertices of the ring have fixed degree $d-1$, then these all have to be flipped away, and will then count as fixed edges for the two vertices that then become connected.

Now we only need to fill the remaining holes with triangles, such that the whole construction becomes a triangulation. The outer triangles are all fixed, so anything that happens outside the construction does not influence anything inside. So we need to make sure that there will be no vertices of high degree outside. To this end, we will add some Steiner points such that the Delaunay triangulation of the resulting set has bounded vertex degree.

We can do this, for example, by using the algorithm of Ruppert [22]. It takes as input a planar straight line graph and generates a triangulation respecting the edges and vertices of the graph, where no angle is smaller than a parameter $\alpha < 20°$. The algorithm produces an output of size related to the *local feature size* of the input (that in turn is related to the vertex spacing, so in our case is some value depending on d). The running time of the algorithm is quadratic in the output size, hence it is also polynomial.

If all angles of the resulting triangulation are larger than, say, 18°, then no vertex has a degree higher than 20 in the Delaunay triangulation, so a value of $d = 20$ is suitable for the construction.

Theorem 5. *Minimizing the maximum vertex degree over all first order Delaunay triangulations is NP-hard.*

3.3. Convex edges

In a triangulated terrain, every edge between two triangles is either convex or reflex. If we know that the (part of the) real terrain we are modeling is globally convex, we may want to maximize the number of convex edges in our model.

Problem 3. Given a set of points with elevation information, construct a first order Delaunay triangulation of this point set such that the number of convex edges is maximum.

This problem is NP-hard, and we prove this by a reduction from planar MAX-2-SAT [11].

We build the SAT instance on a regular grid of unit quadrilaterals. On such a grid, all diagonals are flippable, while the edges between neighboring quadrilaterals are fixed. A flippable quadrilateral always has one convex and one reflex diagonal. We assign the heights in such a way that an edge between two flippable quadrilaterals is convex if at least one neighbor has a reflex diagonal, and reflex otherwise. The edges at the border of the grid should always be reflex.

We build variables by making large areas of adjacent flippable quadrilaterals, where the width of such an area is at least two everywhere, see Fig. 12(a). For such an area the maximal value is achieved by alternating convex and reflex edges in a chessboard pattern, and there are two solutions of equal value, as shown in Figs. 12(b) and 12(c). We get a credit for every convex edge, so in these situations we get one credit for every edge between two adjacent quadrilaterals, and one for each convex quadrilateral, which is half the total number of quadrilaterals.

To connect variables to each other, we make one large *pool* of quadrilaterals for each variable, and send *tentacles* to meet the other variables, see Fig. 13. A meeting point consists of a single shared edge. This edge gives a credit when at least one of the incident quadrilaterals is in the reflex state.

This way we can build the whole MAX-2-SAT instance, and we get a fixed number of credits for all variables, plus the number of satisfiable clauses. The variable pools need to be large enough to make breaking the value of a variable too expensive. If a variable has k outgoing tentacles, we achieve this by making a square pool of size at least $2k \times 2k$ for the variable. If we would break the value of the variable by cutting off j tentacles, this breakline must then cut through at least $2j$ cells of the pool. A breakline costs as many credits as it has edges, divided by two. It can gain as many points as the lowest number of tentacles that were cut off. Therefore, breaking the value costs more than it earns.

Please cite this article in press as: M. van Kreveld et al., Optimization for first order Delaunay triangulations, Computational Geometry (2009), doi:10.1016/j.comgeo.2009.01.010

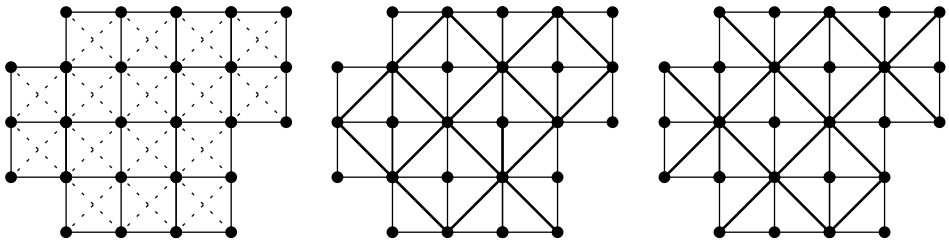


Fig. 12. (a) A group of quadrilaterals. (b) One optimal state. (c) The other optimal state.

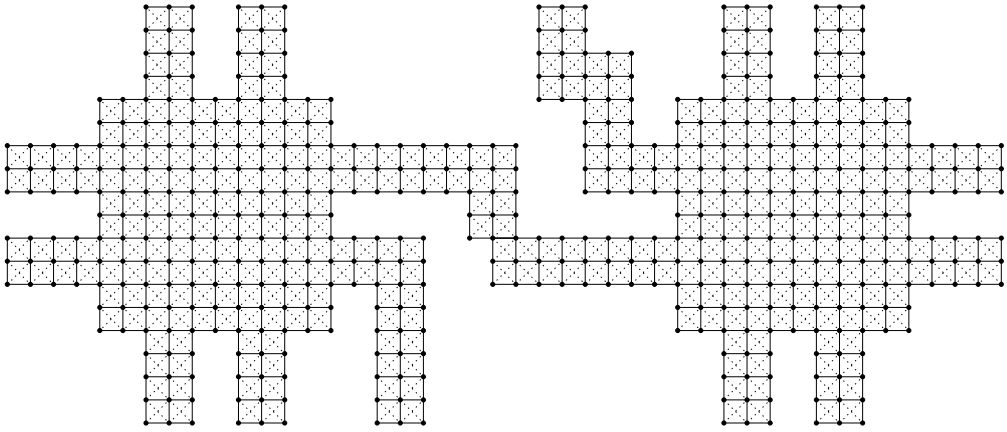


Fig. 13. Connecting variables.

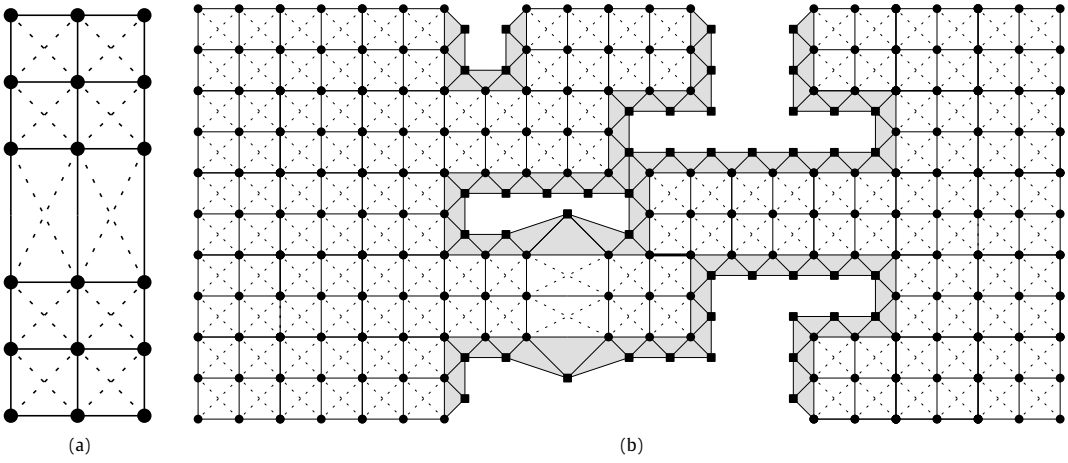


Fig. 14. (a) Skipping a step. (b) A coating of high peaks. The square vertices are at ∞, the round ones at $xy - \frac{x^2+y^2}{10}$.

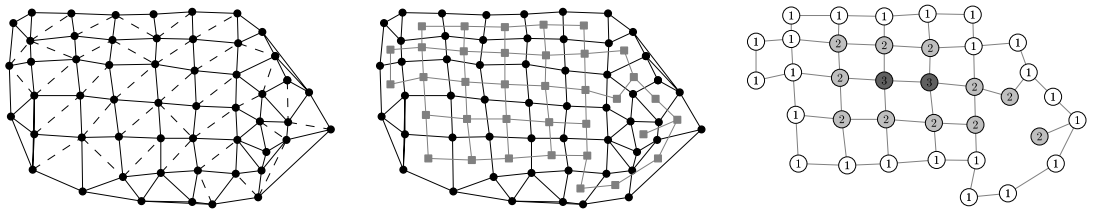
To make sure that we can connect every pair of variables in every desired way, we need to be able to skip a column or row sometimes. A chain with a skip in it is shown in Fig. 14(a).

Finally, we need to assign heights to the vertices to realize the required properties. We can do this by placing the vertices on a surface, for example of the function f:

$$f(x, y) = xy - \frac{1}{10}\left(x^2 + y^2\right).$$

This assignment realizes the properties within the construction. We also required all edges on the border to be always reflex; for that we add a coating of points at elevation ∞, see Fig. 14(b). What happens outside this coating is irrelevant: any first order Delaunay triangulation of the remaining faces can be used.

Please cite this article in press as: M. van Kreveld et al., Optimization for first order Delaunay triangulations, Computational Geometry (2009), doi:10.1016/j.comgeo.2009.01.010



(a) (b) (c)

Fig. 15. (a) Initial triangulation (solid edges are fixed). (b) Graph (in gray) where each vertex represents a flippable quadrilateral. (c) The same graph showing the outerplanarity layers.

Theorem 6. *Maximizing the number of convex edges over all first order Delaunay triangulations is NP-hard.*

4. Approximation algorithms

The problems of optimizing the number of convex edges or mixed vertices and minimizing the maximum vertex degree were shown to be NP-hard; hence, it is of interest to develop approximation algorithms for them. For the last problem there is already a 1.5-approximation [10], and our NP-hardness proof shows that no polynomial time approximation scheme exists unless P = NP. For the other two problems we present polynomial time approximation schemes. We also show how the algorithm for maximizing convex edges can be extended to k-th order Delaunay triangulations.

The general idea is as follows. First we transform the problem into a graph problem on some planar graph that can be obtained from the Delaunay triangulation after removing all flippable edges. The resulting graph is partitioned into layers of outerplanarity at most λ. For each choice of i, where $0 \leqslant i < \lambda$, we delete every $(j\lambda + i)$-th layer of vertices, where $j = 0, 1, 2, \ldots$. The resulting "thick" layers are independent. For each thick layer, we compute a tree decomposition of width at most $3\lambda - 1$ and solve the problem optimally on this decomposition in $2^{O(\lambda)}n$ time, using dynamic programming. Finally, the union of the solutions of all the thick layers for a given i yields a solution to the original problem. We simply choose i such that the size of the solution is maximal, and return the corresponding triangulation as the output.

Such an approach gives a $(1 - \varepsilon)$-approximation if λ is chosen suitably, depending on ε and the problem [2,12]. This can be seen as follows. Let S_i, $0 \leqslant i < \lambda$, be the union of the solutions of the thick layers for a given i. Let S^*, an optimal solution with value $M(S^*)$, be partitioned according to its outerplanarity layers modulo λ. Define C_i to be the group of layers that are l-outerplanar (in the original graph) for those l with l **mod** $\lambda = i$. Since $S^* = \bigcup_i C_i$, there must be an index r such that $|C_r| < \frac{|S^*|}{\lambda}$. Now $S^* \setminus C_r$ is a (probably suboptimal) solution to the problem, where the $(j\lambda + r)$-th layers have been removed. Then $|S_r| \geqslant |S^*| - |C_r| \geqslant (1 - \frac{1}{\lambda})|S^*|$. Therefore the size of the solution given by the algorithm will be at least $(1 - \varepsilon)M(S^*)$.

In the remainder of this section we present the algorithms in more detail.

4.1. Maximizing the number of convex edges

We build a graph G that has a vertex (called q-vertex) for each flippable quadrilateral, and an edge between two q-vertices if and only if their corresponding quadrilaterals share an edge. The rest of the input (all the fixed triangles) are not explicitly represented, see Fig. 15(b). Each q-vertex has two possible states, *convex* or *reflex*, depending on the choice of the diagonal. It also has a value that depends on its state and represents the number of convex edges among the flippable edge and any edges that the quadrilateral shares with fixed triangles when the q-vertex is in that state (from 0 to 5). Furthermore, every edge in G has a value that depends on the states of both incident q-vertices. The goal of the algorithm is to find a state for each q-vertex such that the sum of the values (total number of convex edges) is maximized.

To create the independent thick layers from the graph we will remove the edges that connect two consecutive layers $j\lambda + i$ and $j\lambda + i + 1$ in G, where $j = 0, 1, 2, \ldots$, for all choices of $0 \leqslant i < \lambda$. The layers created after removing one set of layers of edges are independent, so if we optimize them separately and then join them by adding the removed edges, the number of convex edges after the join cannot decrease. Some edges are not considered for every i, but only in $\lambda - 1$ out of λ solutions. We get a $(1 - \varepsilon)$-approximation algorithm by taking $\lambda = \lceil \frac{1}{\varepsilon} \rceil$, due to the pigeonhole principle [2,12].

Once we have the thick layers, each layer is solved optimally by using a tree decomposition approach. Since each layer is a λ-outerplanar graph, a tree decomposition with treewidth at most $3\lambda - 1$ can be computed in time linear in the number of nodes of the graph [5]. Once we have this decomposition we can apply one of the standard techniques to deal with problems on graphs of small treewidth. The technique consists of building tables of partial solutions in the nodes of the tree decomposition [5,20].

Definition 1. (From [20], originally in [21].) Let $G = (V, E)$ be a graph. A *tree decomposition* of G is a pair $\langle \{X_i \mid i \in I\}, T \rangle$ where each X_i is a subset of V, called a *bag*, and T is a tree with the elements of I as nodes. The following three properties must hold:

Please cite this article in press as: M. van Kreveld et al., Optimization for first order Delaunay triangulations, Computational Geometry (2009), doi:10.1016/j.comgeo.2009.01.010

Table 2
Each table A_i contains all the possible assignments for the quadrilaterals in the bag. Each flippable quadrilateral x_i can be assigned a convex (c) or reflex (r) state.

x_1	x_2	…	x_{n_i-1}	x_{n_i}	$m_i()$
c	c	…	c	c	
c	c	…	c	r	
c	c	…	r	c	
c	c	…	r	r	
		…			
r	r	…	r	r	

1. $\bigcup_{i\in I} X_i = V$.
2. For every edge $\{u, v\} \in E$, there is an $i \in I$ such that $\{u, v\} \subseteq X_i$.
3. For all $i, j, k \in I$, if j lies on the path between i and k in T then $X_i \cap X_k \subseteq X_j$.

The *width* of $\langle\{X_i \mid i \in I\}, T\rangle$ equals $\max\{|X_i| \mid i \in I\} - 1$. The *treewidth* of G is the minimum ω such that G has a tree decomposition of width ω.

Let $\langle\{X_i \mid i \in I\}, T\rangle$ a tree decomposition of our graph G. We will make T rooted by choosing any node to be the root. For each bag X_i, we will store a table A_i ($i \in I$), see Table 2. Tables will be created in a bottom up fashion as follows. For each bag X_i, the table A_i has 2^{n_i} rows and $n_i + 1$ columns, where $n_i = |X_i|$. Each row represents an assignment of a state (reflex/convex) to each q-vertex (flippable quadrilateral) in X_i. All the different possible assignments for the bag are represented in the table. Furthermore, for each assignment Γ_j an extra value $m_i(\Gamma_j)$ is stored, containing the number of convex edges in an optimal triangulation of the point set induced by the subtree rooted at X_i that includes the current assignment as a subset. To compute the tables, we simply have to merge the tables of child-nodes and take, for each entry, the largest value among those in the child-tables that has the same assignments to the common quadrilaterals. The details on how to compute these values are presented below.

Step 1: Table initialization. For every table A_i and each assignment Γ_j, we set $m_i(\Gamma_j)$ to be the number of convex edges for that assignment: The sum of the values of all q-vertices (that will vary according to its state), plus 1 for each edge with both incident q-vertices in X_i if their states define a convex edge between the corresponding quadrilaterals (with diagonals chosen).

Step 2: Table update. Next the tree is traversed, starting from the leaves and finishing at the root. For each node, the column m_i of A_i is updated based on its children. Let i be the parent of node j. Bags X_i and X_j have some q-vertices in common. We sort both tables first by the columns of the shared q-vertices, and second by m_i. Then we scan A_i row by row, and for each assignment Γ_l we update $m_i(\Gamma_l)$ based on the highest value that $m_j()$ has for that combination of the shared q-vertices. For later reconstruction of the triangulation we also store a pointer to the corresponding row in A_j. When a node X_i has several children, we update A_i against each child, one at a time, in the same way. Once the root node is updated, the number of convex edges in an optimal triangulation will be in the last column of one of the rows of its table. The final triangulation can be computed by following the pointers in the tables.

The correctness of the method follows from the definition and properties of tree decompositions, and the arguments are identical to the ones that hold for other well-known problems where the same technique has been used, such as vertex cover and dominating set (see [20]).

The running time is dominated by the computation and merging of the tables. The sorting of each table can be done in time $O(2^\omega\omega)$ (because all but one column have only two states). The time for updating a table based on another one is linear in the size of the largest one, which is $O(2^\omega)$. The number of tables is linear in the number of nodes $|I|$ of tree T, hence the total running time is $O(2^\omega\omega \cdot |I|)$. Since the graph is λ-outerplanar, we can compute a tree decomposition of width $\omega \leqslant 3\lambda - 1$ and $|I| = O(n)$ nodes [5,20]. We apply this algorithm to the λ different values of i to get an approximation scheme, so the worst-case running time is $O(\lambda 2^\omega\omega \cdot |I|) = O(\lambda^2 8^\lambda \cdot n) = O(\frac{1}{\varepsilon^2} 8^{\frac{1}{\varepsilon}} \cdot n) = 2^{O(1/\varepsilon)} \cdot n$.

Theorem 7. *For any $\varepsilon > 0$, a $(1-\varepsilon)$-approximation algorithm for maximizing the number of convex edges over all first order Delaunay triangulations exists that takes $2^{O(1/\varepsilon)} \cdot n$ time (if the Delaunay triangulation is given).*

4.2. Maximizing the number of non-mixed vertices

The approach above requires several adaptations before it can be used to maximize the number of non-mixed vertices, mainly because the state of a vertex (mixed/non-mixed) is determined by all incident quadrilaterals. As before, let S be the subdivision that is the Delaunay triangulation of the set P of points, with all flippable edges removed. The graph G of which we will compute a tree decomposition has one vertex for each quadrilateral of S (called a q-vertex), and one vertex for each vertex of S that has at least one incident quadrilateral (called a p-vertex of G). There is an edge between a q-vertex and a p-vertex if the quadrilateral of the q-vertex is incident to the vertex in S of the p-vertex. Note that G is planar and

Please cite this article in press as: M. van Kreveld et al., Optimization for first order Delaunay triangulations, Computational Geometry (2009), doi:10.1016/j.comgeo.2009.01.010

bipartite. Each q-vertex has two possible states, flipped or non-flipped (Delaunay), and each p-vertex has a value, mixed or non-mixed, that depends on the states of the incident q-vertices. The goal is to assign a state to each q-vertex that maximizes the number of non-mixed p-vertices.

Graph G still needs some preprocessing. Firstly, it may contain p-vertices that can never be non-mixed, because their fixed incident edges already make them mixed. All such p-vertices will be removed. Secondly, there can be q-vertices that are connected to only one p-vertex. So we can always choose the diagonal of the quadrilateral non-incident to the p-vertex, helping it to become non-mixed. Hence, all q-vertices of degree 1 can also be removed. In graph G, each q-vertex has degree between 2 and 4, and each p-vertex can be turned non-mixed (for example by choosing all diagonals in incident quadrilaterals non-incident to that vertex).

We will also remove from G all p-vertices with degree larger than some value d. This implies that we will not use these vertices in the maximization of non-mixed vertices, and hence we will lose optimality in this step. Since G is a planar graph, we remove at most $\lfloor 6n/d \rfloor$ vertices, if G had n vertices. We will show that sufficiently many p-vertices remain that can be turned non-mixed to yield an approximation scheme, if d is chosen suitably.

After these preprocessing steps we get a number of connected components with p-vertices of degree at most d. We proceed to solve each component in a similar way as the previous problem: we take thick layers and solve each layer optimally by using dynamic programming on a tree decomposition. For each component we create a series of thick layers that are λ-outerplanar, for λ even, by deleting every $(j\lambda + i)$-th layer of vertices where $j = 0, 1, 2, \ldots$, and i takes values $i = 0, 2, 4, \ldots, \lambda$. Notice that we only need to remove layers of p-vertices, hence i must be even.

We can analyze the two steps where we approximate separately. Let the first one have a ratio ε_1 and the second one ε_2. The solution obtained by our algorithm will be at least $(1-\varepsilon_1)(1-\varepsilon_2)OPT$. We will choose $\varepsilon_1 = \varepsilon_2 = \frac{1}{2}\varepsilon$.

The approximation by removing layers is almost as before, but now each p-vertex is considered in $\lambda/2 - 1$ out of $\lambda/2$ solutions. Therefore we get a $(1-\varepsilon_2)$-approximation by taking $\lambda = \lceil 2/\varepsilon_2 \rceil$ (to simplify the presentation, we are assuming $\lceil 2/\varepsilon_2 \rceil$ is even, otherwise we take $\lambda = \lceil 2/\varepsilon_2 \rceil + 1$). It remains to show that the removal of higher degree vertices gives a $(1-\varepsilon_1)$-approximation. The graph G has a number of connected components where all the q-vertices have degree at most 4, and each p-vertex can be turned non-mixed, possibly at the expense of others. In any component, we can always make a linear number of the p-vertices non-mixed by choosing a p-vertex and choosing the diagonal in all incident quadrilaterals non-incident to that p-vertex. Every time we do this, the "neighboring" p-vertices (connected through exactly one q-vertex) may be prevented from being non-mixed. Since the component is planar, there must be a p-vertex of degree at most 5. We can make that vertex non-mixed, preventing at most 10 (two per quadrilateral) other p-vertices from being non-mixed, and continue in this way. Hence, at least $1/11$ of the p-vertices can be made non-mixed.

We want to ensure that what we threw away because of too high degree is not more than an ε_1 fraction of all possible non-mixed vertices. At least a fraction of $\frac{1}{11}(1-\frac{6}{d})$ of the vertices will be non-mixed, since we removed at most a factor of $\frac{6}{d}$ of the vertices and of the remaining vertices at least 1 out of 11 can be made non-mixed. In the worst case, all $\frac{6}{d}$ vertices we threw away would be non-mixed in the optimal solution. Therefore, we must take $d = \lceil \frac{66}{\varepsilon_1} \rceil + 6$ to guarantee a $(1-\varepsilon_1)$-approximation.

On every thick layer we compute a tree decomposition $\langle \{X_i \mid i \in I\}, T \rangle$ of width at most $3\lambda - 1$. We need to modify this decomposition, because to be able to count a p-vertex as non-mixed, we need to ensure that some bag X_i of the tree decomposition contains that p-vertex with all neighboring q-vertices. The modification simply involves adding these neighboring q-vertices. Any bag X_i will become larger by a factor d, the maximum degree of a p-vertex, because each p-vertex already had at least one adjacent q-vertex in the same bag. One can verify that the new tree decomposition still satisfies the properties of Definition 1, and its width has become at most $(3\lambda - 1)(1 + d)$.

Now we can apply the dynamic programming approach to solve the problem optimally, in the same way as before. Some further details are given next.

Following the notation from the previous problem, we have a tree decomposition of $G = (V, E)\langle \{X_i \mid i \in I\}, T \rangle$, where T is a tree with the elements of I as bags, and $\omega = (3\lambda - 1)(1 + d)$ is the width of the decomposition. Each bag X_i now contains two types of vertices: q-vertices $(q_1, \ldots, q_{n_i})$ and p-vertices $(p_{n_i+1}, \ldots, p_{m_i})$, $|X_i| = m_i$.

For each bag $X_i = \{q_1, \ldots, q_{n_i}, p_{n_i+1}, \ldots, p_{m_i}\}$, compute a table A_i with the same shape as before: 2^{n_i} rows and $n_i + 1$ columns. Only q-vertices have a column in the table. Each row represents an assignment of a value (flipped/non-flipped) to each of the flippable quadrilaterals in X_i. The extra value $m_i(\Gamma_j)$ contains the number of non-mixed p-vertices in an optimal triangulation of the point set induced by the subtree rooted at X_i that includes the current assignment as a subset. Note that due to the extra vertices added to the tree decomposition, we can compute the value (mixed/non-mixed) of all the p-vertices in X_i.

The table initialization is similar to the initialization done before. For each assignment to the q-vertices the number of non-mixed p-vertices is stored in $m_i()$.

The update of the tables is also done in the same way as before. In this case, when processing row by row of A_i, $m_i()$ will be updated by considering the best (highest) entry from A_j for the same assignment for the vertices shared, plus the number of non-mixed vertices added by the assignment of non-shared q-vertices in A_i. It is worth mentioning that it is always possible to compute a value for each p-vertex in X_i because all its neighbors were added to its bag after getting the original tree decomposition. Without this addition, situations could arise in which the value of a p-vertex depends on a q-vertex that is not in the current bag, and hence cannot be computed.

Please cite this article in press as: M. van Kreveld et al., Optimization for first order Delaunay triangulations, Computational Geometry (2009), doi:10.1016/j.comgeo.2009.01.010

The way the final triangulation is obtained, the correctness of the dynamic programming algorithm and its running time are the same as in the previous problem. In particular, the running time is $O(2^{\omega}\omega|I|)$.

As to the running time of the remainder of the algorithm, the removal of higher-degree vertices can be done in linear time. The dynamic programming step on the tree decomposition takes $O(2^{\omega}\omega \cdot |I|)$ time. We take $\lambda = \lceil \frac{2}{\varepsilon_2} \rceil = \lceil \frac{4}{\varepsilon} \rceil$ to ensure that the removal of layers gives a $(1-\varepsilon_2)$-approximation. The treewidth ω of the modified tree decomposition is $(3\lambda - 1)(d+1)$, which is $O(1/\varepsilon^2)$. Therefore the running time of the algorithm is $2^{O(1/\varepsilon^2)} \cdot n$.

Theorem 8. *For any $\varepsilon > 0$, a $(1-\varepsilon)$-approximation algorithm for maximizing the number of non-mixed vertices over all first order Delaunay triangulations exists that takes $2^{O(1/\varepsilon^2)} \cdot n$ time (if the Delaunay triangulation is given).*

4.3. Maximizing the number of convex edges when $k > 1$

Up to now we have always assumed that $k = 1$. However, some of the approximation techniques can also be used to obtain approximation schemes for the same problems when $k > 1$, at the cost of k appearing (rather unpleasantly) in the time bound. To illustrate this, we will now describe how to adapt the algorithm for maximizing the number of convex edges.

The general idea is very similar to the case when $k = 1$. First we compute the Delaunay triangulation of the point set. The resulting triangulation is partitioned into thick layers of outerplanarity at most λ.

First we need to introduce some definitions and results on higher order Delaunay triangulations.

Definition 2. (From [10].) A triangle $\triangle uvw$ in a point set P is *k-th order Delaunay* if its circumcircle $C(u, v, w)$ contains at most k points of P. A triangulation of a set P of points is a *k-th order Delaunay triangulation* if every triangle of the triangulation is k-th order Delaunay.

Definition 3. (From [10].) For a set of points P, an edge $\overline{pq}$ between two points $p, q \in P$ is *k-th order Delaunay* if there is a circle that passes through p and q and contains at most k points of P inside. The *useful order* of an edge is the lowest order of a triangulation that includes that edge.

For brevity, we will sometimes write *k-OD* instead of *k-th order Delaunay*. From now on we assume that $k \geqslant 2$ is a given integer.

Lemma 1. *(From [10].) Let $\overline{uv}$ be a k-OD edge, let s_1 be the point to the left of $\overrightarrow{vu}$, such that the circle $C(u, s_1, v)$ contains no points to the left of $\overrightarrow{vu}$. Let s_2 be defined similarly but to the right of $\overrightarrow{vu}$. Edge $\overline{uv}$ is a useful k-OD edge if and only if $\triangle uvs_1$ and $\triangle uvs_2$ are k-OD triangles.*

Lemma 2. *(From [10].) The Delaunay edges intersecting one useful k-OD edge $\overline{uv}$ are connected to at most k vertices on each side of the k-OD edge.*

Note that the previous lemma implies that a useful k-OD edge can cross at most $2k-1$ Delaunay edges and $2k$ Delaunay triangles.

Lemma 3. *(From [10].) Let $\overline{uv}$ be any Delaunay edge. The number of useful k-OD edges in a triangulation T that intersect $\overline{uv}$ is $O(k)$.*

After computing the Delaunay triangulation of the point set, the algorithm considers thick layers of width (outerplanarity) λ. After every thick layer, we will skip the next k layers. This separation distance of k outerplanarity layers guarantees that no useful k-OD edge will go from one thick layer to the next one, creating independent thick layers (this is because crossing k layers involves crossing $2k$ Delaunay triangles). For each thick layer, we compute an initial tree decomposition of width at most $3\lambda - 1$. In order to solve the problem optimally for that layer, the initial decomposition will be augmented in several ways, without increasing the treewidth too much. This will be detailed in the next subsection.

As before, we will consider each of the possible shifts of the thick layers, solving in this case $\lambda + k$ problems. The union of the solutions of all the thick layers for a given shift yields a solution to the original problem. We simply choose the shift such that the size of the solution is the maximum, and return the corresponding triangulation as the output.

With this approach we obtain a $(1-\varepsilon)$-approximation algorithm by taking $\lambda = \lceil \frac{2k}{\varepsilon} \rceil$. Each useful k-OD edge is considered in $\lambda - k$ out of $\lambda + k$ problems that are solved optimally. The pigeon-hole principle then shows that at least one of the $\lambda + k$ problems has a value that is a $(1-\varepsilon)$-approximation of the optimum.

4.3.1. Exact algorithm for a single layer

For a given thick layer we get a subgraph of the Delaunay triangulation of bounded outerplanarity. This subgraph is planar, but not necessarily a triangulation, since it can have multiple components, a non-convex outer face, and holes. The

Please cite this article in press as: M. van Kreveld et al., Optimization for first order Delaunay triangulations, Computational Geometry (2009), doi:10.1016/j.comgeo.2009.01.010

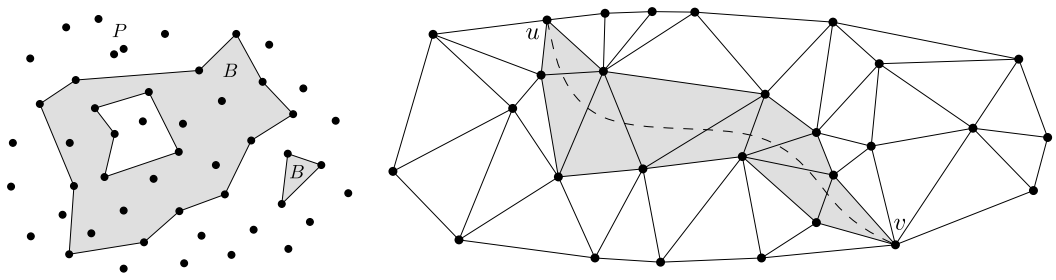


Fig. 16. (a) A point set P, and a polygon B that needs to be retriangulated. (b) A path from u to v. The distance between u and v is 9.

part that is triangulated defines a polygon. We want to retriangulate this polygon, in such a way that we optimize the number of convex edges, while making sure that all triangles are k-OD *with respect to the original point set*. More precisely, we are given a set of n points P and a polygon B that has only points of P as vertices, has no self-intersections, but may have holes and multiple components, see Fig. 16(a). We want to compute a triangulation of the inside of B and the part of P inside B such that the circumcircle of each triangle does not contain more than k points of P.

Definition 4. Given a pair of vertices (u, v) we define their *distance* as the smallest number of triangles that need to be crossed to walk from u to v in the Delaunay triangulation. See Fig. 16(b).

Lemma 4. *Let $\langle\{X_i \mid i \in I\}, T\rangle$ be a tree decomposition of the Delaunay triangulation of a set of points, with width ω. Then for all pairs of vertices (u, v) whose distance is at most d, we can add both vertices to all the bags in the shortest path between u and v in the tree decomposition, without increasing the treewidth to more than $O(2^d\omega)$.*

Proof. For any pair (u, v) at distance at most d, we add vertices u and v to every bag on the shortest path in the tree decomposition. This means that every bag of the tree decomposition that is on this shortest path must include at least one edge of the Delaunay triangulation that is crossed by the path from u to v. This is because of a property of tree decompositions: for any bag b, all parts of the graph outside the bag are disconnected, so if u and v are outside b in different components, any path from u to v must pass through b.

In every bag of the tree decomposition where u and v are added, there is a Delaunay edge that is crossed by the path from u to v in the Delaunay triangulation. Every bag of the initial tree decomposition contains at most ω vertices, so at most $3\omega - 6$ Delaunay edges, and every edge is crossed by at most $O(2^d)$ paths, because the paths have length at most d, and paths walk over triangles, giving two possible directions to continue walking at each step. Therefore, for every bag at most $O(\omega 2^d)$ vertices are added. □

Observation 3. After adding the vertices as in the previous lemma, with $d \geqslant 2k$, every useful k-OD edge appears in at least one bag, and hence also every k-OD triangle.

The first part follows from the fact that if $\overline{uv}$ is a useful k-OD edge, then the distance between u and v cannot exceed $2k$. Since every useful k-OD edge is in some bag, property 3 of tree decompositions implies that every triangle made of three of these edges is also in some bag.

Lemma 5. *(From [23].) Let $\overline{uv}$ be a useful k-OD edge. There are at most $O(k)$ k-OD triangles that have $\overline{uv}$ as one of its edges.*

With this result, we can prove the following lemma, which will ensure that each k-OD edge with two neighboring triangles appears at least once in some bag, so if this edge is convex we encounter this situation in the algorithm.

Lemma 6. *A tree decomposition of width ω containing all useful k-OD edges can be augmented to include every pair of k-OD triangles that share a useful k-OD edge in at least one bag, thereby increasing the treewidth to at most $O(k\omega^2)$.*

Proof. For each useful k-OD edge $\overline{uv}$ we will add all the points that create a k-OD triangle together with $\overline{uv}$. Since every useful k-OD edge is present in some bag, there exists a bag containing all pairs of k-OD triangles sharing a particular k-OD edge. Each bag contains at most ω^2 useful edges; hence, from Lemma 5 we know that each bag increases its size to at most $O(k\omega^2)$. □

Lemma 7. *Given a tree decomposition T of width ω of some graph G, we can construct a tree decomposition T' of G with the following properties*:

Please cite this article in press as: M. van Kreveld et al., Optimization for first order Delaunay triangulations, Computational Geometry (2009), doi:10.1016/j.comgeo.2009.01.010

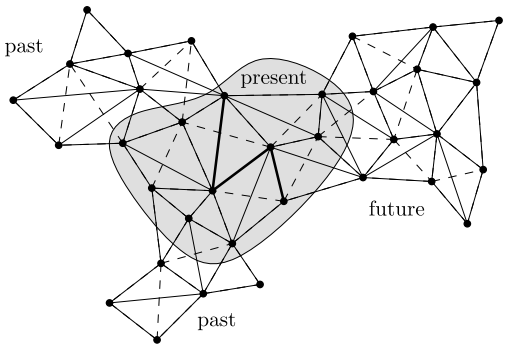


Fig. 17. A bag of the tree decomposition partitions the vertices into three groups: the past, the present, and the future. Inside the bag, a fence separates the past from the future. The black edges form a second order Delaunay triangulation; the dashed edges are Delaunay edges.

- *Every pair of adjacent bags* (X_i, X_j) *in* T' *differs by exactly one vertex, that is,* $X_i = X_j \cup \{x\}$ *or* $X_j = X_i \cup \{x\}$ *for some vertex* x *of the graph* G.
- *The size increases by at most a factor* 2ω*, that is,* $|T'| \leqslant 2\omega|T|$.
- *The width increases by at most a factor* 2*, that is,* $\omega' \leqslant 2\omega$.

Proof. Let X_i and X_j be adjacent bags in T. Let $X'_i = X_i - X_j$ and $X'_j = X_j - X_i$ be the unique parts of the bags. Now we will create $|X'_i| + |X'_j| - 1$ new bags between X_i and X_j. First we add the elements of X'_j to X_i, one by one, until we create a new bag which is $X_i \cup X_j$. Then we remove the elements of X'_i from this bag, one by one, until we are left with just X_j. In this way, we create fewer than $2\omega(T)$ new bags of width at most $2\omega(T)$. Note that the inclusion of the new bags does not violate the tree decomposition property: any vertex of G is still present in a connected subset of the bags of the tree. If we do this for every pair of adjacent bags in T, we obtain the required tree decomposition T'. □

We now augment our initial tree decomposition of width $3\lambda + 1$ to a tree decomposition of width at most $O(k2^{8k+1}(3\lambda + 1)^2) = 2^{O(k)}\lambda^2$, applying Lemma 4 (with $d = 4k$), and Lemmas 6 and 7.

Let T be the resulting tree decomposition. Now, for any bag in T, we define three subsets of the vertices of T. The vertices inside the current bag we call the *present*. The vertices in bags lower in the hierarchy (which we encountered earlier in a bottom-up algorithm), but not in the current bag, we call the *past*. Finally, the vertices in bags higher up in the hierarchy, but not in the current bag, we call the *future*. Fig. 17 shows an example. Note that, by property 3 of tree decompositions, a vertex cannot be in the past and in the future at the same time so the partitioning is well-defined. In fact, since every k-OD edge is in some bag of T, we can make the following observation.

Observation 4. If $\overline{uv}$ is a useful k-OD edge, it cannot be that u is in the past and v is in the future.

Before stating the algorithm, we need one more concept. Intuitively, we may assume that, when processing a certain bag, the vertices in the past are already triangulated and the vertices in the future are not. So, in the present, there has to be some transition. Recall that B is the polygon that we are triangulating.

Definition 5. Inside a bag, a *fence* is a collection of edges between vertices in the bag, such that those edges together with the boundary of B separate the past from the future. See Fig. 17.

In the algorithm, we will identify a fence that separates the triangulated part from the untriangulated part for each configuration inside the current bag. Because of the following lemma, we will not miss any solutions by doing this.

Lemma 8. *Given a polygon with some points inside, and a tree decomposition* T *following the previous lemmata: if* G *is a* k*-OD triangulation of that polygon, any bag of* T *contains a set of edges of* G *that form a fence for* T.

Proof. By Observation 4, there are never any edges going from the past to the future, with respect to the current bag. Therefore, there are no triangles that block one boundary from the opposite boundary and there is always a path over the edges from boundary to boundary. □

Dynamic programming. Let X_i be a bag of the tree. X_i has a collection V_i of vertices. For every pair of vertices $v, w \in V_i$ we have a Boolean variable if $\overline{vw}$ is a useful k-OD edge. A variable will be true if the edge is present, and false if it is not. So, every assignment of the variables in X_i determines a graph.

Please cite this article in press as: M. van Kreveld et al., Optimization for first order Delaunay triangulations, Computational Geometry (2009), doi:10.1016/j.comgeo.2009.01.010

For each assignment, we will store a value. This value describes the best possible score of any triangulation of the vertices in the present and the past that adheres to this assignment. Furthermore, we maintain the edges of this triangulation that are not in the current bag, that is, the edges between two past vertices or between a past and a present vertices. Of course, this collection of edges is different for each assignment.

Some assignments of the variables lead to invalid graphs. For example, two edges that cross are not allowed to be in the same triangulation, or triangles that are not k-OD are not allowed. In such cases, we discard this assignment. Also, there must be a fence in the bag (using edges that are set to true), such that all vertices on the past side of the fence are properly k-OD triangulated.

For an assignment, we define the value as the number of convex edges in the graph, where an edge is convex when it is the diagonal of a quadrilateral, and the angle between the two triangles thus formed is convex.

We follow the tree using dynamic programming. In every step, we assume that the values for all variable assignments in the child bag are given, and we need to compute the values for all variable assignments in the parent bag. As in Section 4.1, if there are multiple child bags, we merge them with the parent bag one by one. We distinguish two different cases.

- The parent bag has a vertex v that is not in the child bag. The bags are otherwise identical.
  There are several new useful k-OD edges that connect v to existing vertices in the bag. We get a new variable for all those edges. Note that there cannot be useful k-OD edges that connect v to a vertex in the past, by Observation 4.
  For every possible state of the new set of variables, if there is no value stored in the child bag because the state has crossings or no fence with a proper triangulation on the past side, then we store no value in the parent bag either. If there was a value in the child bag, then we compute the new value by adding the number of newly formed convex edges to the stored value.
- The child bag has a vertex v that is not in the parent bag. The bags are otherwise identical.
  Now there are several edges that were in the child bag, but are no longer in the parent bag. We lose a variable for each of these edges.
  For every state in the parent bag, we choose among the states in the child bag the one that has the highest score. We also test whether this state still contains a valid fence, and if not, we discard this state.

4.3.2. Time analysis

The running time of the dynamic programming algorithm depends on the treewidth and size of the decomposition (number of bags). The input graph on which to compute the tree decomposition is λ-outerplanar; hence, we can obtain a decomposition of treewidth $3\lambda-1$. The decomposition is augmented first according to Lemma 4, then according to Lemma 6, and finally according to Lemma 7. The treewidth goes up to $\omega = 2^{O(k)}\lambda^2$. The last augmentation step also increases the number of bags from $O(n)$ to $m = 2^{O(k)}n$.

The dynamic programming algorithm can be easily shown to run in time $O(m2^{\omega}\omega)$ (it is ω because we keep a triangulation of the ω vertices in the bag, and ω points can be triangulated in at most $O(2^{\omega})$ different ways).

Filling in ω and m we obtain a running time of $2^{2^{O(k)}\lambda^2\log\lambda}n$.

Theorem 9. *The subproblem of triangulating the polygon can be solved in* $2^{2^{O(k)}\lambda^2\log\lambda}n$ *time.*

The dynamic programming algorithm of the previous sections is applied to $\lambda+k$ different problems. Each problem is comprised of a number of different thick layers. Their number depends on the outerplanarity of the initial triangulation, and is at most $n/(\lambda+k)$. The dynamic programming algorithm is applied to each of these thick layers. Since the union of them is never larger than the complete triangulation, the sum of the running times for all the thick layers can be upper-bounded by the running time of applying the dynamic algorithm to the complete triangulation. As already mentioned, to obtain a factor $(1-\varepsilon)$ approximation, λ is chosen to be $\frac{2k}{\varepsilon}$. Therefore the total running time is $(\lambda+k)\cdot 2^{2^{O(k)}\lambda^2\log\lambda}n = 2^{2^{O^*(k)}}2^{O^*(\frac{1}{\varepsilon^2})}n$ (assuming the initial Delaunay triangulation is given).

Theorem 10. *For any* $\varepsilon > 0$, *a* $(1-\varepsilon)$*-approximation algorithm for maximizing the number of convex edges over all* k*-th order Delaunay triangulations exists that takes* $2^{2^{O^*(k)}}\cdot 2^{O^*(\frac{1}{\varepsilon^2})}\cdot n$ *time* (*if the Delaunay triangulation is given*).

5. Discussion

We analyzed the algorithmic complexity of optimizing various measures that apply to triangulations, and terrains represented by triangulations. The class of triangulations over which optimization is done is the class of first order Delaunay triangulations. We gave efficient algorithms for four measures, NP-hardness proofs for three other measures, and polynomial time approximation schemes for two measures that were shown NP-hard. One approximation algorithm could be extended to k-th order Delaunay triangulations.

Other measures related to terrain modeling in GIS may be of interest to optimize. Also, certain measures that have efficient, optimal algorithms for first order Delaunay triangulations may become harder for second and higher order Delaunay

Please cite this article in press as: M. van Kreveld et al., Optimization for first order Delaunay triangulations, Computational Geometry (2009), doi:10.1016/j.comgeo.2009.01.010

triangulations. These are interesting topics for further research. It is also unknown how to generalize the approximation algorithm for maximizing non-mixed vertices to higher order Delaunay triangulations. Finally, improving on the doubly-exponential dependency on the order k in the approximation algorithm for maximizing convex edges is worthwhile.

Acknowledgements

We thank Hans Bodlaender and René van Oostrum for helpful discussions, and Norbert Zeh and anonymous referees for their detailed comments.

References

[1] B. Aspvall, M. Plass, R. Tarjan, A linear-time algorithm for testing the truth of certain quantified boolean formulas, Inform. Process. Lett. 8 (1979) 121–123.
[2] B.S. Baker, Approximation algorithms for NP-complete problems on planar graphs, J. ACM 41 (1994) 153–180.
[3] M. Bern, H. Edelsbrunner, D. Eppstein, S. Mitchell, T.S. Tan, Edge insertion for optimal triangulations, Discrete Comput. Geom. 10 (1) (1993) 47–65.
[4] M. Bern, P. Plassmann, Mesh generation, in: J. Sack, J. Urrutia (Eds.), Handbook of Computational Geometry, Elsevier, Amsterdam, 1997, pp. 291–332.
[5] H.L. Bodlaender, A partial k-arboretum of graphs with bounded treewidth, Theoret. Comput. Sci. 209 (1998) 1–45.
[6] L. de Floriani, P. Magillo, E. Puppo, Applications of computational geometry in Geographic Information Systems, in: J. Sack, J. Urrutia (Eds.), Handbook of Computational Geometry, Elsevier, Amsterdam, 1997, pp. 333–388.
[7] T. de Kok, M. van Kreveld, M. Löffler, Generating realistic terrains with higher-order Delaunay triangulations, Comput. Geom. Theory Appl. 36 (2007) 52–65.
[8] H. Edelsbrunner, T.S. Tan, A quadratic time algorithm for the minmax length triangulation, SIAM J. Comput. 22 (1993) 527–551.
[9] H. Edelsbrunner, T.S. Tan, R. Waupotitsch, An $O(n^2 \log n)$ time algorithm for the minmax angle triangulation, SIAM J. Sci. Statist. Comput. 13 (4) (July 1992) 994–1008.
[10] J. Gudmundsson, M. Hammar, M. van Kreveld, Higher order Delaunay triangulations, Comput. Geom. Theory Appl. 23 (2002) 85–98.
[11] L.J. Guibas, J.E. Hershberger, J.S.B. Mitchell, J.S. Snoeyink, Approximating polygons subdivisions with minimum link paths, Internat. J. Comput. Geom. Appl. 3 (4) (Dec. 1993) 383–415.
[12] D.S. Hochbaum, W. Maass, Approximation schemes for covering and packing problems in image processing and VLSI, J. ACM 32 (1985) 130–136.
[13] R.J. Huggett, Fundamentals of Geomorphology, Routledge, 2003.
[14] S.K. Jenson, C.M. Trautwein, Methods and applications in surface depression analysis, in: Proc. Auto-Carto 8, 1987, pp. 137–144.
[15] G. Kant, H.L. Bodlaender, Triangulating planar graphs while minimizing the maximum degree, Inform. Comput. 135 (1997) 1–14.
[16] D. Lichtenstein, Planar formulae and their uses, SIAM J. Comput. 11 (2) (1982) 329–343.
[17] Y. Liu, J. Snoeyink, Flooding triangulated terrain, in: P. Fisher (Ed.), Developments in Spatial Data Handling, Proceedings of the 11th Int. Sympos., Springer, Berlin, 2004, pp. 137–148.
[18] D.R. Maidment, GIS and hydrologic modeling, in: M. Goodchild, B. Parks, L. Steyaert (Eds.), Environmental Modeling with GIS, Oxford University Press, New York, 1993, pp. 147–167.
[19] W. Mulzer, G. Rote, Minimum weight triangulation is NP-hard, in: Proc. 22nd Annu. ACM Sympos. Comput. Geom., 2006, pp. 1–10.
[20] R. Niedermeier, Invitation to Fixed-Parameter Algorithms, Oxford University Press, New York, 2006.
[21] N. Robertson, P.D. Seymour, Graph minors II. Algorithmic aspects of tree width, J. Algorithms 7 (1986) 309–322.
[22] J. Ruppert, A Delaunay refinement algorithm for quality 2-dimensional mesh generation, J. Algorithms 18 (1995) 548–585.
[23] R.I. Silveira, M. van Kreveld, Optimal higher order Delaunay triangulations of polygons, in: Proc. Latin American Theoretical Informatics (LATIN), in: Lecture Notes in Computer Science, vol. 4957, Springer, Heidelberg, 2008, pp. 133–145.
[24] D.M. Theobald, M.F. Goodchild, Artifacts of TIN-based surface flow modelling, in: Proc. GIS/LIS, 1990, pp. 955–964.
[25] G.E. Tucker, S.T. Lancaster, N.M. Gasparini, R.L. Bras, S. Rybarczyk, An object-oriented framework for distributed hydrologic geomorphic modeling using triangulated irregular networks, Comput. Geosci. 27 (2001) 959–973.
[26] M. van Kreveld, Geographic information systems, in: J.E. Goodmann, J. O'Rourke (Eds.), Handbook of Discrete and Computational Geometry, Chapman & Hall/CRC, Boca Raton, FL, 2004, pp. 1293–1314 (Chapter 58).

Please cite this article in press as: M. van Kreveld et al., Optimization for first order Delaunay triangulations, Computational Geometry (2009), doi:10.1016/j.comgeo.2009.01.010