Counting Crossing Free Structures

Victor Alvarez* Karl Bringmann† Radu Curticapean‡ Saurabh Ray§

Abstract

Let P be a set of n points in the plane. A crossing-free structure on P is a straight-edge planar graph with vertex set in P. Examples of crossing-free structures include triangulations of P, and spanning cycles of P, also known as polygonalizations of P, among others. There has been a large amount of research trying to bound the number of such structures. In particular, bounding the number of triangulations spanned by P has received considerable attention. It is currently known that *every* set of n points has at most $O(30^n)$ and at least $\Omega(2.43^n)$ triangulations. However, much less is known about the algorithmic problem of counting crossing-free structures of a given set P. For example, no algorithm for counting triangulations is known that, on all instances, performs faster than enumerating all triangulations. In this paper we develop a general technique for computing the number of crossing-free structures of an input set P. We apply the technique to obtain algorithms for computing the number of triangulations and spanning cycles of P. The running time of our algorithms is upper bounded by $n^{O(k)}$, where k is the number of *onion layers* of P. In particular, we show that our algorithm for counting triangulations is not slower than $O(3.1414^n)$. Given that there are several well-studied configurations of points with at least $\Omega(3.464^n)$ triangulations, and some even with $\Omega(8^n)$ triangulations, our algorithm is the first to asymptotically outperform any enumeration algorithm for such instances. In fact, it is widely believed that any set of n points must have at least $\Omega(3.464^n)$ triangulations. If this is true, then our algorithm is strictly sub-linear in the number of triangulations counted. We also show that our techniques are general enough to solve the *restricted triangulation counting problem*, which we prove to be $W[2]$-hard in the parameter k. This implies a "no free lunch" result: In order to be fixed-parameter tractable, our general algorithm must rely on additional properties that are specific to the considered class of structures.

1 Introduction

Let P be a set of n points on the plane. A crossing-free structure defined by P, or on P, is a straight-edge planar graph whose vertex set is P. Examples of such crossing-free structures are triangulations, spanning cycles, matchings, spanning trees, etc. One can naturally ask: what are upper and lower bounds on the number of such structures over all possible sets of n points on the plane. Or given P, how can the number of such geometric objects be computed. The search for bounds has spawned a large amount of research over almost 30 years, starting with an upper bound of 10^{13n} on the number of crossing-free graphs on every set of n points, see [2]. This bound implies that the size of *each* class of crossing-free structures can be upper-bounded by c^n, with $c \in \mathbb{R}$ depending on the particular class.

*Fachrichtung Informatik, Universität des Saarlandes, alvarez@cs.uni-saarland.de. Partially Supported by CONACYT-DAAD of México.

†Max-Planck-Institut für Informatik, kbringma@mpi-inf.mpg.de.

‡Fachrichtung Informatik, Universität des Saarlandes, curticapean@cs.uni-saarland.de

§Max-Planck-Institut für Informatik, saurabh@mpi-inf.mpg.de.

Since then, research has focused on tightening the upper and lower bounds on c. For example, in the case of spanning cycles, it is currently known that $4.6 \leq c \leq 54.55$, see [3] for the upper bound and [4] for the lower bound. Thus, every set of n points has at least $\Omega(4.6^n)$ and at most $O(54.55^n)$ spanning cycles. For triangulations, [5] provides the bound $c \leq 30$, and [6] provides $c \geq 2.4$. The interested reader can visit [7, 8] for an up-to-date list of bounds on other classes of crossing-free structures. The references therein gather the modern history of all listed bounds.

The second question on crossing-free structures, mentioned above, is of algorithmic flavor since we consider the problem of *computing* the number of crossing-free structures of a particular class for a *given* input set P. This problem is closely related to that of sampling crossing-free structures of the class uniformly at random, that is, if P spans, say t spanning cycles, we want to sample every spanning cycle with probability $1/t$. A first approach to the counting problem would be to produce *all* elements of the class, using methods for enumeration, and then simply count the number of elements. This has the obvious disadvantage that the total time spent will be, at best, linear in the number of elements counted. By the first part, this number is always exponential in the input size. Thus an important question is whether we can count crossing-free structures of a given class in time sub-linear in the number of elements counted.

In this paper we focus on counting the elements of two particular classes of crossing-free structures defined over a given set P. The first class, $\mathcal{F}_T$, is the class of triangulations of P. The second class, $\mathcal{F}_C$, is the class of spanning cycles of P, also known as polygonalizations of P.

To state the main results of our paper, we need to define the notion of an *onion layer* of P. The formal definition will be given in the next section, but intuitively, k onion layers mean k nested convex sets. Thus every point set has at least one onion layer, so $k \geq 1$.

Theorem 1. *Let P be a set of n points with k onion layers. The number of triangulations of P can be computed in time $O(k^2 g(n/k)^n) = n^{O(k)}$, where $g(x) = ((x^3 + 3x^2 + 2x + 2)/2)^{\frac{1}{x}}$. Since $k \leq \lceil \frac{n}{3} \rceil$, this bound never exceeds $O(3.1414^n)$.*

Theorem 2. *Let P be as before. The number of spanning cycles of P can be computed in $n^{O(k)}$ time.*

For $k = O(1)$, our results yield algorithms that have polynomial running time in n, regardless of the cardinalities $|\mathcal{F}_C|$ and $|\mathcal{F}_T|$. Thm. 2 gives in particular a partial answer to Problem 16 of [9].

As stated before, for every set of n points, $|\mathcal{F}_T|$ can be lower-bounded by $\Omega(2.4^n)$, but it is widely believed that this bound can be improved to $\Omega(\sqrt{12}^n) \approx \Omega(3.464^n)$. If this stronger bound is true, the algorithm of Thm. 1 would count triangulations in time $O(3.1414^n) = o(|\mathcal{F}_T|)$, thus answering one of the aforementioned open questions in the positive. The general layout of our algorithms is similar to the one found in [16] where these ideas have been used for optimization problems.

The running times of both algorithms can be stated as $n^{f(k)}$, for some function f that does not depend on n. With regard to parameterized complexity, it is natural to ask if these runtimes can be improved to $f(k) \cdot n^{O(1)}$, thus proving that our problems belong to FPT, the class of fixed-parameter tractable problems. However, our techniques are general enough to solve harder problems, such as the *restricted triangulation counting problem*: Given P and a set of admissible edges E as input, count the triangulations of P that use only edges from E. We prove the following hardness result:

Theorem 3. *The restricted triangulation counting problem is $W[2]$-hard if the parameter is considered to be k, the number of onion layers of P. This result even holds for the problem of deciding the* existence *of a restricted triangulation.*

The book [10] is a standard reference for parameterized complexity theory and defines the classes FPT and $W[2]$. For now, it suffices to say that the separation $FPT \neq W[2]$ is widely believed

and indicates that we may have to exploit the particular structure of the problems in order to obtain fixed-parameter tractable algorithms for counting crossing-free structures.

The rest of the paper is organized as follows: In Section 2, we introduce our general framework. In Section 3 we will formally introduce the terminology and definitions needed to prove Thm. 1. In Sections 4 and 5, we prove Thm. 2 and 3. We finish in Section 6 with some conclusions.

2 General framework

In this section, we describe the central ideas of our counting algorithms. This section aims at an intuitive description and will thus abstain from technical details, which will be given in the next two sections, where the ideas are demonstrated with two concrete applications.

Let P be a set of n points and suppose that we want to count triangulations of P. A set S of non-crossing edges is called a *separator* if the union of edges in S splits $\text{conv}(P)$ into at least two regions. Suppose that there exists a set of separators $\mathcal{S}$ with the following properties: (1) every triangulation T of P contains a unique separator $S \in \mathcal{S}$, and (2) we can enumerate the members of $\mathcal{S}$. With a set of separators $\mathcal{S}$, the triangulations of P can be counted as follows: For each $S \in \mathcal{S}$, let R_1^S, R_2^S,..., R_t^S be the regions of $\text{conv}(P) \setminus S$. Recursively compute the number of triangulations N_i^S of each R_i^S. The number of triangulations containing S is then $N^S = \prod_{i=1}^t N_i^S$, and the total number of triangulations of P is simply $\sum_{S \in \mathcal{S}} N^S$. Of course, in the recursion, a set of separators is required in each R_i^S, and the efficiency of the algorithm depends heavily on the choice of $\mathcal{S}$. One well-known family of separators $\mathcal{S}$ for triangulations is the set of T-paths of a set of points, see [14]. In Section 3, we introduce another set of separators for counting triangulations.

For now, let us move on to a slightly more complicated counting problem. Suppose we want to count crossing-free matchings spanned by the point set. Since any matching can be completed to a triangulation by adding edges, we could try the technique used for counting triangulations. Take any set $\mathcal{S}$ of separators. For each $S \in \mathcal{S}$, count the matchings in triangulations containing S, and finally add this up over all $S \in \mathcal{S}$. In any matching M that can be completed to a triangulation containing S, each vertex in S is either unmatched, or it is matched to a vertex within some R_i^S, or it is matched to another vertex in S. We can *annotate* each separator S with this information. When counting, for each $S \in \mathcal{S}$, we iterate over all annotations of S, and take care to be consistent with the current annotation when recursing into the subproblems.

This simple algorithm fails because some matchings M could contain several, say $s_M > 1$, separators and would thus fool our algorithm to count M exactly s_M times. This is no problem if $s_M = s$ were constant over all matchings, but we are not aware of any set of separators $\mathcal{S}$ with this property.

However, there is a different way to ensure that each matching is counted exactly once: we embed each matching M into a unique triangulation $T \supseteq M$. Given a family $\mathcal{S}$ of separators for the triangulations of P, we associate a unique $S \in \mathcal{S}$ to each matching. For concreteness, let us associate to each M the constrained delaunay triangulation (CDT) Δ^M *constrained* to contain M. Under standard non-degeneracy assumptions, there is a unique CDT for any given set of mandatory edges. We revise our algorithm as follows: Whenever we recurse, in each subproblem we only count matchings M with $S \subseteq \Delta^M$. If this last condition can be satisfied locally in each subproblem, i.e., choices in one subproblem do not depend on choices in others, we are done. While not every $\mathcal{S}$ admits such a locality condition, some do as we will see later.

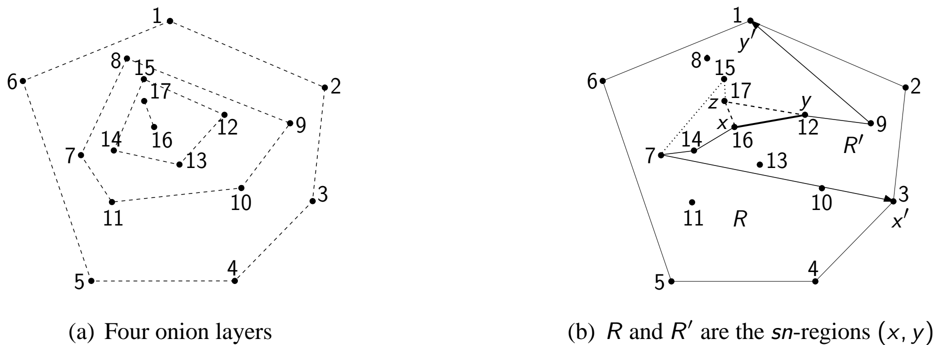
The annotations required for counting *matchings* are not very complicated, but for many other counting problems, they are. An example of more involved annotations is given in Section 4, where we consider the problem of counting spanning cycles.

The techniques we described are fairly general and can be applied to several counting problems.

The choice of separator structures and annotations depends on the specific problem and affects the efficiency of the algorithm. We start now by demonstrating the technique for counting triangulations.

3 Counting triangulations

Let P be a set of n points on the plane. Let conv(P) denote the convex hull of P and let ∂conv(P) denote its boundary. We define the onion layers of P as follows: the first onion layer $P^{(1)}$ of P is $P \cap \partial\text{conv}(P)$. For $i > 1$, the ith onion layer $P^{(i)}$ of P is defined inductively as $P \cap \partial\text{conv}(P \setminus \bigcup_{j=1}^{i-1} P^{(j)})$. By "number of onion layers of P", we mean the number of non-empty onion layers of P. For any $p \in P$, let $\ell(p)$ denote the index of the onion layer to which p belongs. Let us label the points $p \in P$ with distinct labels in $\{1, \dots, n\}$ such that if $\ell(p) < \ell(q)$ then p also receives a label smaller than q. This is clearly possible. Figure 1(a) shows the onion layers of a set of 17 points and the labels assigned to them. From now on we will refer to the points of P by their labels *i.e.*, we will think of P as the set $\{1, \dots, n\}$ and when we say "$p \in P$", we will mean the point with label p.



(a) Four onion layers

(b) R and R' are the sn-regions (x, y)

Figure 1

Let T be any triangulation of P. For $p \in P \setminus P^{(1)}$, let $sn_T(p)$ be the smallest neighbor of p in T. Observe that any such point p has at least one neighbor q such that $\ell(q) < \ell(p)$ and therefore $sn_T(p) < p$. If $p \in P^{(1)}$, we set $sn_T(p) = p$. When T is clear from context, we will just write $sn(p)$ instead of $sn_T(p)$. We denote by sn-path$_T(p)$ the unique path $p = a_0, a_1, \dots, a_m$ in T such that for each $0 \le i < m$, we have that $a_{i+1} = sn(a_i)$ and $sn(a_m) = a_m$. We will also direct this path from a_0 towards a_m and call this the direction of *descent* since $\ell(\cdot)$ decreases along the path. Note that any sn-path consists of at most one point from each onion layer and precisely one point from the first onion layer.

Let (p, q) be some edge in T and suppose that sn-path(p) ends at $p' \in P^{(1)}$ and sn-path(q) ends in $q' \in P^{(1)}$. There are two paths in T from p' to q' along ∂conv(P), one in the clockwise direction and the other in the counter-clockwise direction. Each of these paths along with the edge (p, q) and the two sn-paths starting at p and q respectively, defines a region within conv(P). We call these two regions the sn-regions of (p, q). See Figure 1(b). Given any sn-region R, we refer to the number of triangles in any triangulation of R as the *size* of R. This is well defined since the number of triangles is the same regardless of the triangulation chosen.

Let ab be an edge on ∂conv(P). Observe that in any triangulation, conv(P) is one of the sn-regions of (a, b), the other region being empty. In any triangulation T of P, there is precisely one triangle apb that the edge ab belongs to. Let R_{ap} be the sn-region of (a, p) that does not contain apb and similarly let R_{pb} be the sn-region of (p, b) that does not contain apb.

3.1 The algorithm

The main idea of our algorithm is as follows. We can easily enumerate all the points p such that the triangle apb appears in some triangulation. This is just the set Q of points p such that the triangle apb is free of other points of P. For every element p of Q, suppose that we can enumerate the sn-paths ρ of p over all triangulations of P. For ever pair (p, ρ), let $\mathcal{T}_{p,\rho}$ be the set of triangulations that contain the triangle apb and in which ρ is the sn-path of p. If, for each such pair that we can obtain, we can compute $|\mathcal{T}_{p,\rho}|$, then we are done, since each triangulation of P must contain precisely one pair (p, ρ), adding the numbers over all pairs gives us the total number of triangulations.

Let us fix a pair (p, ρ) for which we would like to compute $|\mathcal{T}_{p,\rho}|$. The pair already defines the regions R_{ap} and R_{pb} for all triangulations in $\mathcal{T}_{p,\rho}$. Observe that any triangulation in $\mathcal{T}_{p,\rho}$ contains a triangulation T_{ap} of R_{ap} and a triangulation T_{pb} of R_{pb}, each of which satisfy the following sn-**constraint**: for each edge (q, r) in ρ there is no edge (q, s) in the triangulation (either T_{ap} or T_{pb}) such that $s < r$. Furthermore, putting together any pair of triangulations T_{ap} and T_{pb}, each satisfying the constraint, and the triangle apb gives a triangulation in $\mathcal{T}_{p,\rho}$. This observation follows from the fact that ρ is an sn-path of p in any triangulation of $\mathcal{T}_{p,\rho}$ and allows us to separately compute the number of (sn-constraint satisfying) triangulations N_{ap} of R_{ap} and N_{pb} of R_{pb} whose product gives $|\mathcal{T}_{p,\rho}|$.

The numbers N_{ap} and N_{pb} are computed recursively. We will maintain that at any point in the recursion we are dealing with an sn-region of some edge. This is certainly true in the beginning since we start with an sn-region of the edge ab and also in the next step since we recurse on sn-regions defined by the edges (a, p) and (p, b) respectively. At any point, let us say that we are dealing with an sn-region R defined by an edge (x, y) and let ρ_x and ρ_y be the sn-paths starting at x and y respectively.

We recurse almost exactly as we did before: we enumerate the set of points z such that the triangle xzy lies within R and is free of other points of P contained in R, see figure 1(b). Furthermore, we ensure that if z happens to be a point in either ρ_x or ρ_y, and (z, t) is an edge in that sn-path, then both a and b are smaller than t. This way, we do not violate the sn-constraint. For each such z, we enumerate the portions of sn-paths starting at z that lie within R. See Figure 1(b). Each such path splits the region R into regions R_{xz} and R_{zy} which are sn-regions defined by (x, z) and (z, y) respectively. Each of the regions R_{xz} and R_{zy} have sizes smaller than R. Recall that the size of a region is the number of triangles required to triangulate it. The recursion bottoms out when the size is ≤ 1 - in which case we know that there is exactly one triangulation. Note that even though we enumerate only the portions of the sn-paths of z that lie within R, these portions implicitly define the entire sn-path of z. This is because such a portion either ends at a point on the first onion layer in which case it is the entire sn-path, or at a point w on either ρ_x or ρ_y. The direction of descent along that sn-path, starting at w, is then the remaining portion of the sn-path of z.

One detail is still missing. How do we enumerate the portions of the sn-paths of z that belong to at least one triangulation of R? Well, we will not do it, instead, we enumerate a superset of paths which are *descending* in the sense that they start at z and each successive point is in a strictly smaller layer (layer with a smaller index). Again, we only enumerate the portion of such paths that lie inside R since the rest is implicitly defined. For any descending path that does not really belong to any triangulation of R, at least one of the regions R_{xz} or R_{zy} has no triangulations satisfying the sn-constraint. This will be detected somewhere down the recursion where we will not be able find any z satisfying the sn-constraint. At that point, we return 0 as the number of triangulations. Thus the algorithm still works. There is one other ingredient that we add for efficiency: memoization. Whenever we compute the number of triangulations of a certain sn-region that satisfy the sn-constraint dictated by the sn-paths defining the region, we store it in a hash table (or any other data structure). Consider a *call graph* in which each node represents an sn-region and there is a directed edge from a region R to a region R'

if from R we make a recursive call to R'. The number of egdes in this graph is an upper bound on the running time of the algorithm since, because of memoization, no edge is *traversed* more than once.

We will now prove an upper bound on the number of edges in the call graph. Each call from a region R to a region R' can be charged to a triple of descending paths - two defining R and a third that, along with a triangle, splits R into two regions, one of which is R'. The triples (ρ_1, ρ_2, ρ_3) that are produced in the algorithm have the property that once two paths merge in the direction of descent, they never split again. This is ensured by the fact that we only enumerate the portions of the third descending path within the region R and the rest is implicitly defined, as noted before. Let ρ_2' be the portion of ρ_2 that does not have any point in common with ρ_1, and let ρ_3' be the portion of ρ_3 that does not have any point in common with either ρ_1 or ρ_2. The descending paths ρ_1, ρ_2' and ρ_3' are vertex disjoint, and along with some additional information they completely describe ρ_1, ρ_2 and ρ_3. The additional information that is required is whether, and where, ρ_2 merges with ρ_1, and whether, and where, ρ_3 merges with one of the other paths. If P has k onion layers, then each descending path has length at most k and therefore there are at most k ways that ρ_2' may merge with ρ_1, and at most $2k$ ways ρ_3' may merge with one of ρ_1 or ρ_2. Therefore, if U is an upper bound on the number of triples of vertex disjoint descending paths, then $2k^2U$ is an upper bound on the number of triples (ρ_1, ρ_2, ρ_3) as described above, and hence also an upper bound on the running time of the algorithm.

3.2 Number of vertex-disjoint triples of descending paths

Each descending path uses at most one vertex from every onion layer. Let $n_i = |P^{(i)}|$ be the size of the ith onion layer. Let us count how many ways there are for any triple of paths to use at most one vertex each from this layer. There is one way for each of the paths to skip this onion layer. There are n_i ways of choosing one point among the n_i which may then be used by any of the paths. This gives $3n_i$ ways for the three paths. There are $\binom{n_i}{2}$ ways to choose two points, and any two of the paths may use them. This gives $6\binom{n_i}{2}$ ways for the three paths. Finally there are $\binom{n_i}{3}$ ways of choosing three points, and there are three (not six) ways for the three paths to use one of these vertices. This is because these paths are non-crossing planar curves, and therefore the clockwise order of these paths along any $\partial \mathrm{conv} P^{(i)}$ that intersects all three of them is the same for each i. The overall number of ways in which at most three points can be used from the ith layer is therefore $f(n_i)$, where $f(x) = 1 + 3x + 6\frac{x(x-1)}{2} + 3\frac{x(x-1)(x-2)}{6}$.

The number of triples of vertex disjoint descending paths is therefore at most $U = \prod_{i=1}^{k} f(n_i)$. Since each n_i is a positive integer, and the function $f(\cdot)$ is log-concave, as can be checked, for $x \geq 1$, the above product is maximized when each n_i is equal to n/k. This give an upper bound of $f(n/k)^k = g(n/k)^n$, where $g(x) = f(x)^{1/x}$. Now, $g(x)$ is maximized for some value of x between 0 and 1 and is a decreasing function for $x \geq 1$. Since each onion layer except the kth one must have at least three points, we have $U = O(g(3)^n)$. The fact that the kth onion layer may have less than three points makes only a difference of a constant factor. Therefore the running time of our algorithm is $O(k^2 g(3)^n) = O(3.1414^n)$. This concludes the proof of Thm. 1.

Often the number of onion layers can be much smaller than the maximum possible $\lceil n/3 \rceil$. For example, Dalal [11] has shown that if n points are chosen uniformly at random from a disk, then the expected number of onion layers of the resulting point set is $\Theta(n^{2/3})$.

4 Counting spanning cycles

In this section we give an algorithm for counting crossing-free spanning cycles of a point set P. We start by defining a suitable family of separator.

4.1 Triangular Path

We assume again that P has k onion layers. For every point $p \in P$ (on layer $P^{(i)}$ which is not the first layer) we fix in advance a ray ρ_p which emanates from p and does not intersect the interior of conv$(P^{(i)})$.

For any triangulation T of P there is a unique triangle $\Delta_p = p, q_1, q_2$ adjacent to p and intersecting ρ_p. Let q_p be the smaller of q_1 and q_2. Clearly q_p lies in a layer lower than the one containing p. Let $p_0, p_1, \dots, p_r$ be the sequence so that $p_0 = p$, $p_{i+1} = q_{p_i}$, $\forall\ 0 \leq i < k$, and p_r lies on the first layer. We call $P_p(T) := \bigcup_i \Delta_{p_i}$ the *triangular path* of p w.r.t. T and we call p_r the *last point* of $P_p(T)$. See figure 2(a).

$P_p(T)$ is uniquely defined for any triangulation. Moreover, for distinct triangulations T_1 and T_2, $P_p(T_1), P_p(T_2)$ are either identical or they intersect properly: let i be the first position where $\Delta_{p_i}(T_1) \neq \Delta_{p_i}(T_2)$, then those two triangles intersect, as they both are adjacent to p, intersect ρ_p and have interiors free of points in P.

Before we continue, we describe constrained Delaunay triangulations, which will be needed in our algorithm.

4.2 Constrained Delaunay Triangulation

A constrained Delaunay triangulation (CDT) Δ^S of S is a Delaunay triangulation constrained to include a given set of *mandatory edges* S. More formally, it is the triangulation T containing S such that no edge e in $T \setminus S$ is flippable in the following sense: let Δ_1, Δ_2 be triangles sharing e. The edge e is flippable if and only if $\Delta = \Delta_1 \cup \Delta_2$ is convex, and replacing e with the other diagonal of Δ increases the smallest angle of the triangulation of Δ.

The CDT is unique if the points are non-cocircular, which is a reasonable assumption for counting crossing-free structures, as they do not change when perturbing the point set slightly.

We are now ready to describe our algorithm.

4.3 The Algorithm

Instead of counting spanning cycles, we count *rooted and oriented* spanning cycles. Given any cycle, we make it rooted by designating a *starting vertex*, and we make it oriented by assigning an *orientation*- clockwise or counter-clockwise. We then number the vertices in the cycle from 1 to n, beginning at the starting vertex, and continuing along the assigned direction. We also direct the edges along this direction. This way, each spanning cycle is counted exactly $2n$ times. At the end we divide by $2n$ to get the required number. In the remainder, we use the term HamCycle for rooted an oriented spanning cycles.

Given a HamCycle H let Δ^H be the CDT of H. We annotate Δ^H as follows:

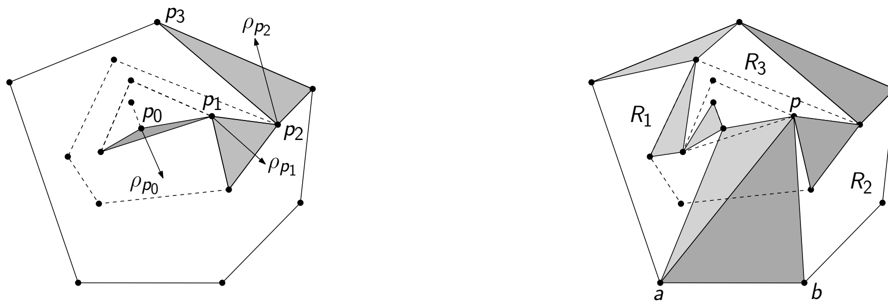
- each vertex v of Δ^H is annotated with $(pos_v, prev_v, next_v)$, where pos_v is the number assigned to v in H, $prev_v$ is the vertex lying immediately before v in H, and $next_v$ is the vertex lying immediately after v in H.
- each edge e in Δ^H is annotated with a bit b_e that indicates whether e belongs to H or not.

The annotated Δ^H will be denoted by $\bar{\Delta}^H$. Let S be a separator contained in Δ^H that splits conv(P) into regions $R_1, \dots R_t$. Separator S inherits the following information from $\bar{\Delta}^H$: each vertex $v \in S$ inherits pos_v from $\bar{\Delta}^H$. If $prev_v$ and $next_v$ are present in S, then this information is also inherited. If $prev_v$ is absent in S then v is annotated with the index i, $1 \leq i \leq t$, of the region R_i that

$prev_v$ falls in. The same holds for $next_v$. Each edge e of S carries the annotation it has in $\bar{\Delta}^H$. The separator S of Δ^H thus annotated will be denoted by $\bar{\Delta}^H_S$.

We say that an annotated constrained Delaunay triangulation is *legal* if and only if it is identical to $\bar{\Delta}^H$, for some HamCycle H. Since there is a one-to-one correspondence between HamCycles and legal constrained Delaunay triangulations, our goal is to count the latter.

Our algorithm is essentially the same as for counting triangulations: instead of sn-paths we use annotated triangular paths. We start with an edge ab on ∂conv(P), and enumerate the set of points p such that the triangle apb is free of other points of P. For each such p, the triangle apb along with the triangular path starting at p forms a separator, see figure 2(b). We enumerate such separators and all possible annotations for each one of them. Each such annotated separator splits conv(P) into smaller regions in which we recurse. In each recursive sub-problem we count (legal) annotated constrained Delaunay triangulations consistent with the annotated separator.



(a) Triangular path P_p starting in onion layer $P^{(4)}$. Onion layers are shown in dashed. P_p can be extended to a triangulation T, in such a case P_p will be unique for T.

(b) In the first call of the algorithm, the triangular path shown in dark gray is created. It divides the problem into regions $R_1 \cup R_3$ and R_2. A call for the latter creates the triangular path shown in light gray. Annotations are not shown for simplicity.

Figure 2

The reason for which we use triangular paths instead of simple sn-paths is the following: no edge in a separator, formed by a triangular path, lies on the boundary of more than one sub-problem. This allows us to verify flippability of edges separately in each sub-problem. If an edge belonged to more than one sub-problem, then the flippability of this edge would depend on the choices made in each sub-problem, thus introducing dependency between these sub-problems.

As in the case for counting triangulations, we use *memoization*. The running time as before is dominated by the number of triples of annotated triangular paths. The size of each triangular path is $O(k)$, thus there are at most $n^{O(k)}$ triangular paths. There are at most $n^{O(k)}$ annotations per triangular path, as can be easily checked. Hence there are $n^{O(k)}$ annotated triangular paths, and $n^{O(k)}$ triples of annotated triangular paths. The overall running time is thus $n^{O(k)}$, which concludes the proof of Thm. 2.

5 Deciding existence of restricted triangulations is $W[2]$-hard

Let P be a set of n points with k onion layers and let E be some set of "admissible" edges spanned by P. We say that a triangulation T of P is *restricted* w.r.t E if $T \subseteq E$. In this section, we consider the following *restricted triangulation existence problem*: On input (P, E), decide whether there exists a triangulation of P that is restricted w.r.t. E. This also defines a counting problem in the natural way, and the existence problem is trivially reducible to the counting problem.

The restricted triangulation existence problem was proven to be NP-complete in [13, 12]. We observed that both reductions are actually parsimonious, directly implying $\#P$-completeness of the restricted triangulation *counting* problem. So far all reductions involving restricted triangulations rely heavily on the ability to specify the set E. If E is instead fixed to the set of all edges spanned by P, we obtain the problem of counting *all* triangulations of P, which we believe to be $\#P$-complete as well. Note that the corresponding existence problem is trivial in this case.

In this section, we parameterize the restricted triangulation counting/existence problems by k, the number of onion layers of P. Observe that the counting algorithm presented in Section 3 can be easily adapted to solve the counting problem, and thus the existence problem, in time $n^{O(k)}$. This, along with Thm. 3, means that our algorithmic framework is general enough to solve a $W[2]$-hard problem and therefore does not admit a general improvement to runtimes of the form $f(k) \cdot n^{O(1)}$ under the assumption $FPT \neq W[2]$. Still, it might be possible to count crossing-free structures of particular classes (other than restricted triangulations) in time $f(k) \cdot n^{O(1)}$ by exploiting the structure of the class.

In the remainder of this section, we turn our attention to the parameterized existence problem to prove Thm. 3. Our proof is by reduction from the *parameterized hitting set problem*, which is proven to be $W[2]$-hard in [10]. An instance A of this problem is formed by numbers $n, m, k \in \mathbb{N}$, along with sets $S_1, \dots, S_m \subseteq [n]$, where k is the parameter, and $[n] := \{0, \dots, n-1\}$. The output to A is "yes" iff there is a set $H \subseteq [n]$ of size at most k, such that $H \cap S_i \neq \emptyset$ for every $1 \leq i \leq m$.

In our reduction, several gadgets are used to transform an instance A of the hitting set problem to an instance $G_A = (P, E)$ of the restricted triangulation existence problem. The reduction is an *fpt-reduction* in the sense of [10], that is, it maps every instance A with parameter k to an instance G_A with $O(k)$ onion layers. Each gadget is given by a set of points with $O(1)$ onion layers, along with a set of admissible edges. The basic gadget is the *pipe*, shown in Fig. 3:

Definition 1. *A* ***pipe*** Q *with* n *states and length* $l > 8(n-1)$ *consists of points* $p_1 \dots p_l$, $q_1 \dots q_l$ *with* $p_t = (t, 0)$, $q_t = (t, 1)$ *and a set* $S \cup F \cup L_0 \cup \dots \cup L_{n-1}$ *of admissible edges. The individual sets are defined as follows:*

$$\begin{aligned}
& a_{0,t} = \{p_t, q_{t+1}\},\ b_{0,t} = \{q_{t+1}, p_{t+1}\} \\
\textit{for } i \geq 1: \ & a_{i,t} = \{p_t, q_{t+4i}\},\ b_{i,t} = \{q_{t+4i}, p_{t+1}\} \\
\textit{for } i \in [n]: \ & L_i = \{a_{i,1}, \dots, a_{i,l-w}, b_{i,1}, \dots, b_{i,l-w-1}\} \textit{ with } w = 1 \textit{ for } i = 0 \textit{ and } w = 4i \textit{ else} \\
& S = \{\{p_1, q_{1+t}\} \mid 0 \leq t \leq 4(n-1)\} \cup \{\{p_{l-t}, q_l\} \mid 0 \leq t \leq 4(n-1)\} \\
& F = \{\{p_i, p_{i+1}\} \mid i < l\} \cup \{\{q_i, q_{i+1}\} \mid i < l\}
\end{aligned}$$

Intuitively speaking, each set L_i, for $i \in [n]$, forms a *zig-zag line*, where edges of the form $a_{i,t}$ are the "zigs", and edges of the form $b_{i,t}$ are the "zags". It is clear that in any triangulation T of a pipe Q, exactly one zig-zag L_i, for $i \in [n]$, is contained, since different zig-zags lines cross. If $L_i \subseteq T$, we say that Q "carries" the value i in T. The edges of the set S allow to complete a triangulation of a pipe once a zig-zag is chosen.

A pipe will always be "horizontal", i.e., it will not turn in any other direction. Thus we will also have "wires", which run between pipes. Wires are pipes with two states. As such, they can be stretched by arbitrary factors, and bent by arbitrary angles, while increasing their length only by $O(1)$. This is shown in Fig. 3. For wires, we relabel the values 0 and 1 by false and true respectively.

The remaining gadgets are defined as follows, their correctness can be found in the appendix:

- An **or-gadget** is connected to input wires W_1, W_2, and an output wire W_3, as shown in Fig. 4. If one of W_1 or W_2 carries true, then W_3 may carry true. If W_3 carries true, then at least one of W_1 or W_2 must carry true.

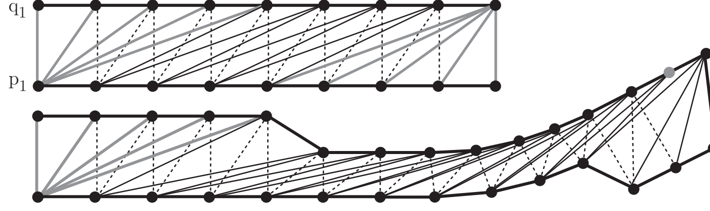


Figure 3: **(Top)** A pipe with 2 states and $l = 9$. Thick black edges constitute F, thick grey edges constitute S, thin edges are zig-zag L_1, and dashed edges are zig-zag L_0. This pipe is also a wire. **(Bottom)** A strechted and bent wire with a terminal.

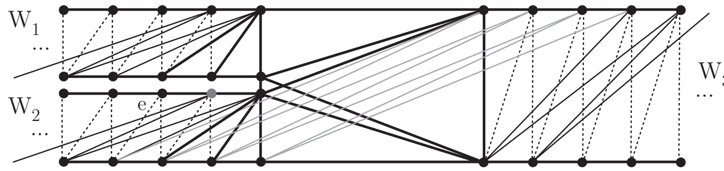


Figure 4: The or-gadget. The grey edges from W_2 to W_3 are "transfer edges". An analogous set of edges is also present from W_1 to W_3, but suppressed in this figure to improve legibility.

- A **terminal-gadget** can be attachted to a wire W, replacing its "end part" as exemplified in the bottom part of Fig. 3. It admits a valid triangulation iff W carries $true$
- A **tester-gadget** for i at t is connected to a pipe Q, between $a_{i,t}$ and $b_{i,t}$, and has an output wire W, see Fig. 5a. If Q carries i, W may carry $true$. If W carries $true$, Q must carry i.
- A **crossing-gadget** allows an input wire V to intersect a pipe Q, leaving it as an output wire W. The value carried by Q is not influenced by V. If V carries $true$, then W may carry $true$. If W carries $true$, then V must also carry $true$.

  As shown in Fig. 5b, V enters the crossing-gadget from the top. Between points q_t and q_{t+1}, a new point r is added. For all p_u adjacent to q_{t+1} via some a-edge $a_{i,u}$, the edge $a_{i,u}$ is replaced by $a'_{i,u} = \{p_u, r\}$ and $\{p_{u+1}, r\}$. Latter edge is shown dashed in Fig. 5b.

  In the area between $a'_{i,u}$ and $b_{i,u}$, the wire V is prolonged to W_i, as shown in Fig. 5c for $i = 0$. All wires W_i are stretched by a factor α. This α is chosen small enough to ensure that, for all i, the grey edges shown in Fig. 5c do not intersect either $a'_{i,u}$ or $b_{i,u}$.

  Finally, the wires $W_0, \dots, W_{n-1}$ are connected to a chain of or-gadgets, as shown in Fig. 5b.

To describe how the gadgets fit together, recall that an instance A of the parameterized hitting set problem is formed by numbers $n, m, k \in \mathbb{N}$, along with sets $S_1, \dots, S_m \subseteq [n]$, where k is the parameter. We create, in polynomial time, an instance $G_A = (P, E)$ of size $poly(n, m)$ that has $O(k)$ onion layers and admits a triangulation w.r.t. E iff A admits a hitting set of size $\leq k$. The mapping $A \mapsto G_A$ will clearly be polynomial-time computable, and thus an fpt-reduction.

In the construction, we start with parallel pipes $Q_1, \dots, Q_k$ of n states each, and of length polynomial in m and n, as shown in Fig. 6. Pipe Q_i lies above pipe Q_{i+1}. Let Q_i be a pipe, $1 \leq i \leq k$, and let $S_j = \{s_{j,1}, \dots s_{j,t}\} \subseteq [n]$ be a set of instance A. We define the *stripe* $B_{i,j}$ as a set of t testers attached to Q_i that check if Q_i carries any of the values of set S_j. The stripe $B_{i+1,j}$ will lie in the same

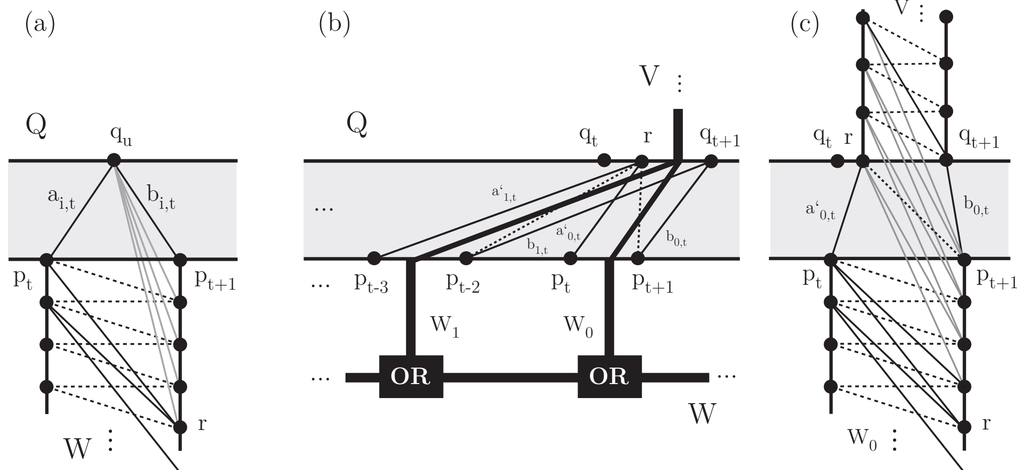


Figure 5: **(a)** The tester-gadget for i at t. Q is modified by shifting, for $k > 0$, all p_{t+k} and q_{u+k} to the right until the triangle (r, p_{t+1}, q_u) is oriented counterclockwise. **(b)** A crossing between pipe Q and wire W. **(c)** Details at W_0.

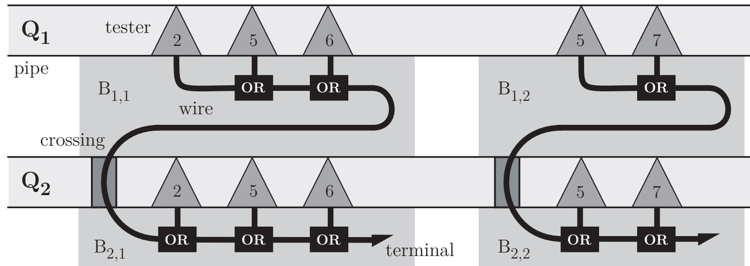


Figure 6: Instance G_A produced from instance A of the parameterized hitting set problem with $n = 8$, $k = 2$ and $S_1 = \{2, 5, 6\}$, $S_2 = \{5, 7\}$.

vertical slab as $B_{i,j}$. The testers of $B_{i,j}$ are connected to a chain of or-gadgets that lies between pipes Q_i, Q_{i+1}. For $i < k$, the output of the last or-gadget in $B_{i,j}$ is carried to $B_{i+1,j}$ by a crossing-gadget, see Fig. 6. For $i = k$, the last or-gadget in $B_{i,j}$ is connected to a terminal-gadget instead.

We now define the *block* $\mathcal{B}_j$ as the union of the stripes $B_{1,j}, \ldots, B_{k,j}$. The blocks $\mathcal{B}_1, \ldots, \mathcal{B}_m$ are arranged horizontally in such a way that the points in stripes $B_{i,1}, \ldots, B_{i,m}$, with $1 \leq i \leq k$, are aligned by their y-coordinates.

Finally, P is defined to be the set of points of all the gadgets involved. To define the set E of edges, we first include the admissible edges from all gadgets involved. Then, the empty spaces between gadgets are then triangulated arbitrarily, adding these edges to E. Having set $G_A = (P, E)$, we can now complete the proof of Thm. 3 by using the following lemma, whose proof can be found in the appendix:

Lemma 1. *G_A has $O(k)$ onion layers and admits a triangulation iff A admits a hitting set of size $\leq k$.*

6 Conclusions

Our more important open questions at this moment are: (1) is it true that every set of n points contains at least $\Omega(3.464^n)$ triangulations? (2) Is it possible to compute the number of triangulations, and of

spanning cycles, of a given point set P in polynomial time? These questions look at the moment very challenging.

It would be very interesting to practically compare our algorithm for counting triangulations with the ones presented in [1, 14, 15]. While we suspect that our algorithm will be fast in practice, it remains to be verified experimentally.

References

[1] V. Alvarez, K. Bringmann, S. Ray, *A Simple Sweep Line Algorithm for Counting Triangulations*, Preprint, 2011.

[2] M. Ajtai, V. Chvátal, M.M. Newborn, E. Szemerédi, *Crossing-free subgraphs*, Annals Discrete Math. **12**:9–12, 1982.

[3] M. Sharir, A. Sheffer, E. Welzl, *Counting Plane Graphs: Perfect Matchings, Spanning Cycles, and Kasteleyn's Technique*, Manuscript, 2011.

[4] A. García, M. Noy, J. Tejel, *Lower bounds on the number of crossing-free subgraphs of* K_n, Computational Geometry: Theory and Applications **16**:211-221, 2000.

[5] M. Sharir, A. Sheffer, *Counting Triangulations of Planar Point Sets*, Electr. J. Comb. **18:1**:P70, 2011.

[6] M. Sharir, A. Sheffer, E. Welzl, *On Degrees in Random Triangulations*, Proc. 26th ACM Symp. on Computational Geometry, 297–306, 2010.

[7] A. Sheffer, *Number of plane graphs: http://www.cs.tau.ac.il/˜sheffera/counting/PlaneGraphs.html*

[8] E. Demaine, *Simple Polygonizations: http://erikdemaine.org/polygonization/*

[9] *http://maven.smith.edu/ orourke/TOPP/P16.html#Problem.16*

[10] J. Flum, M. Grohe, *Parameterized Complexity Theory*, Springer, ISBN 978-3-540-29952-3, 2006.

[11] K. Dalal, *Counting the onion*, Random Struct. Algorithms, **24**:2:155–165, 2004.

[12] A. Schulz, *The Existence of a Pseudo-triangulation in a given Geometric Graph*, Proc. 22nd European Workshop on Computational Geometry, 17–20, 2006.

[13] E. L. Lloyd, *On triangulations of a set of points in the plane*, Proc. 18th Annual Symp. on Foundations of Computer Science, 228–240, 1977.

[14] Oswin Aichholzer, *The path of a triangulation*, Proceedings of the fifteenth annual symposium on Computational Geometry - SoCG '99. 14–23, 1999.

[15] S. Ray, R. Seidel, *A simple and less slow method for counting triangulations and for related problems*, Proceedings of the 20th European Workshop on Computational Geometry - EuroCG '04. 2004.

[16] E. Anagnostou, D. Corneil, *Polynomial-time instances of the minimum weight triangulation problem*, Computational Geometry: Theory and Applications, **3:5**:247–259, 1993.

A Omitted proofs in Section 5

We prove that each of the gadgets fulfills its specification.

Or-gadget: First claim: Assume wlog that W_2 carries *true* in some triangulation T. We construct a triangulation as follows: Add the transfer edges at W_2 to T, as shown in Fig. 4. Then triangulate W_3 by L_{true}. Triangulate the middle part of the or-gadget by thick edges. If W_1 carries *false*, this already suffices, else additionally include the thick edges in W_1.

Second claim: If W_3 carries *true*, the transfer edges from either W_1 or W_2, say W_2, must be present in the triangulation. If W_2 carried *false*, it can include L_{false} only up to edge e, since all following edges intersect transfer edges. But then, the grey point fails to be part of a triangle.

Terminal: If W carries *true*, the terminal is already triangulated. If W carries *false*, the grey point fails to be part of a triangle.

Tester: First claim: Assume Q carries i in some triangulation T: Since no grey edges in the tester intersect L_i, they can be added to T, and W can thus be triangulated by L_{true}.

Second claim: Given some triangulation T, in which W carries *true*, all grey edges must be present in T. But for all $j \neq i$, there is an edge $c \in L_j$ in Q that intersects both $a_{i,t}$ and $b_{i,t}$ and all grey edges, and therefore $c \notin T$. Then $L_j \nsubseteq T$ for all $j \neq i$, forcing $L_i \subseteq T$.

Crossing: First claim: Let T be a triangulation in which Q carries i and V carries *true*. Include the grey edges at W_i, triangulate W_i to carry *true* and leave the or-gadget chain with *true* at W.

Second claim: Assume W carries *true* in a triangulation T. Then, by correctness of the or-gadget, some wire W_i must carry true. But then, the grey edges shown in Fig. 5c must be present at W_i. Thus, V must carry true.

A triangulation is also possible if V and W carry *false*: Assume Q carries i in T and would thus contain $a_{i,u} = \{p_u, q_{t+1}\}$, for some u, if the crossing were not present. Instead of $a_{i,u}$, we include $a'_{i,u} = \{p_u, r\}$ and $\{p_{u+1}, r\}$ into T. This ensures that the parallelogram $(p_u, p_{u+1}, q_{t+1}, r)$ has the diagonal $\{p_{u+1}, r\}$, shown dashed in Fig. 5b and Fig. 5c.

Proof. [of Lemma 1] Consider the number of different y-coordinates of P. This is an upper bound for the number of onion layers of P. The pipes contribute $2k$ different y-coordinates. Every other gadget features $O(1)$ different y-coordinates. Each wire can be stretched and bent with $O(1)$ extra length, and gives $O(1)$ different y-coordinates. Since the points in stripes $B_{i,1}, \dots, B_{i,m}$ are aligned by y-coordinates, each set $\mathcal{C}_i := B_{i,1} \cup \dots \cup B_{i,m}$ has $O(1)$ different y-coordinates. This totals to $2k + O(k) = O(k)$ different y-coordinates for all points in P.

Given a hitting set $H = \{x_1, \dots, x_k\}$, we construct a triangulation that uses only edges from E: For every $i \leq k$, make Q_i carry x_i. For every $j \leq m$, pick some $h(j) \in \{1, \dots, k\}$ with $x_{h(j)} \in S_j$. In stripe $B_{h(j),j}$, triangulate the output wire of the tester for $x_{h(j)}$ to carry *true*. Triangulate all following or-gadgets to output *true*, possibly passing crossing gadgets, until the terminal of $\mathcal{B}_t$ is reached, which can then be triangulated.

On the other hand, the values $H = \{x_1, \dots, x_k\}$ carried by $P_1, \dots, P_k$ in any valid triangulation of G_A form a hitting set: Since every terminal is triangulated, the wire of every block $\mathcal{B}_j$ must carry *true* at some place. Thus, the output of some or-gadget in $\mathcal{B}_j$ must carry *true*. Consider the first or-gadget that fulfills this, and say it lies in stripe $B_{h(j),j}$. It must be connected to a tester that outputs *true*. This implies $x_{h(j)} \in S_j$ and $H \cap S_j \neq \emptyset$. ■