# Counting Triangulations and other Crossing-free Structures via Onion Layers

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

November 12, 2012

## Abstract

Let $P$ be a set of $n$ points in the plane. A crossing-free structure on $P$ is a plane graph with vertex set $P$. Examples of crossing-free structures include triangulations of $P$, 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 (straight-edge) 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, matchings, 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, for $k = O(1)$ our algorithms run in polynomial time. In addition, we show that our algorithm for counting triangulations is *never* slower than $O^*(3.1414^n)$, even when $k = \Theta(n)$. 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 asymptotically outperforms any enumeration algorithm for such instances, and it has better worst-case behavior than the recent algorithm shown in [1], which also beats enumeration in those 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.

[*]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`.

# 1  Introduction

Let $P \subset \mathbb{R}^2$ be a set of $n$ points. A crossing-free structure on $P$ is a plane graph whose vertex set is precisely $P$. Examples of such crossing-free structures, whose proper definition will be given later on, are triangulations, spanning cycles, matchings, spanning trees, etc. Thus one can naturally ask: (1) What are upper and lower bounds on the number of such structures over all possible sets of $n$ points on the plane? or (2) Given $P$, how can the number of such geometric objects be computed? The search for bounds, the first question, 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. 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 $c \leq 54.55$, see [3], and a configuration where $c \geq 4.65$ is known, see [4]. Thus, every set of $n$ points has 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 give a good account of all listed bounds.

The second question on crossing-free structures, which was 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 (see e.g. [9]), 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 in general 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. Only for the super class of *all* plane graphs of $P$ this is known to be *always* possible, see [10], while for the class of all triangulations it is known to be *sometimes* possible using a recent algorithm shown in [1]. There are nevertheless other algorithms to count triangulations that are reported to be faster than enumeration [11, 12], but that have no theoretical runtime guarantees.

# 2  Our contribution

Although the algorithm to count triangulations presented in [1] could potentially *always* do the counting in sub-linear time in the number of elements counted, and thus beat brute force enumeration, its running time might still be pretty large. This is because its running time depends linearly on the number of T-paths that the algorithm encounters

during its execution, and there are configurations having at least $\Omega(4^n)$ T-paths, see [13]. We suggest the paper [12] by O. Aichholzer, where T-paths were first introduced, to find out more about them.

In this paper we present yet another new algorithm to count triangulations. Along with this new algorithm we also present algorithms to count the elements of the classes of spanning cycles and matchings of $P$ respectively. It is important to keep in mind that, so far, no algorithm is known that *always* beats enumeration on those classes.

In order to state the results we present in this paper we need the following definitions:

**Definition 1.** *Let $S$ be a crossing-free structure on $P$. Thus:*

- *$S$ is called a triangulation of $P$ if and only if the boundary of the unbounded face of $S$ coincides with the convex hull, $\mathrm{CH}(P)$, of $P$, and every bounded face of $S$ is an empty triangle.*

- *$S$ is called a matching of $P$ if and only if every vertex of $S$ has degree at most one.*

- *$S$ is called a spanning cycle of $P$ if and only if $S$ is single simple polygon with $n$ sides whose vertex set is $P$.*

We will denote by $\mathcal{F}_T(P), \mathcal{F}_M(P)$ and $\mathcal{F}_C(P)$ the class of *all* triangulations, matchings, and spanning cycles of $P$ respectively.

**Definition 2** (Onion layers)**.** *Let $P$ be a set of $n$ points on the plane and let $\mathrm{CH}(P)$ denote its convex hull. We define the* onion layers *of $P$ as follows: The first onion layer $P^{(1)}$ of $P$ is $\mathrm{CH}(P)$. For $i > 1$, the $i$-th onion layer $P^{(i)}$ of $P$ is defined inductively as $\mathrm{CH}\left(P \setminus \bigcup_{j=1}^{i-1} P^{(j)}\right)$. By "number of onion layers of $P$" we mean the number of non-empty onion layers of $P$.*

Observe that the number of onion layers of *any* non-degenerate set of $n$ points is *always* at most $\left\lceil \frac{n}{3} \right\rceil$. We are now able to state our results.

## 2.1 The result on counting triangulations

**Theorem 1.** *Let $P$ be as before and let $k$ be its number of onion layers. Then the exact value of $|\mathcal{F}_T(P)|$ can be computed in time $O\left(k^2 \cdot g\left(\frac{n}{k}\right)^n\right)$, where $g(x) = \left(\frac{x^3+3x^2+2x+2}{2}\right)^{\frac{1}{x}}$. Since $k \leq \left\lceil \frac{n}{3} \right\rceil$, this bound never exceeds $O^*(3.1414^n)$. This running time can alternatively be bounded by $n^{O(k)}$, which is polynomial for constant $k$.*

The algorithm of the previous theorem has better worst-case behavior than the one presented in [1], which is $O^*(9^n)$. Moreover, it has other nice properties:

- It is the *first* algorithm to be known that can compute the exact value of $|\mathcal{F}_T(P)|$ in *polynomial time* in at least some non-trivial cases.

- As stated before, for *every* set of $n$ points, the size of $\mathcal{F}_T(P)$ can be lower-bounded by $\Omega(2.4^n)$, but it is widely believed that this bound can be improved to $\Omega\left(\sqrt{12}^n\right) \approx \Omega(3.464^n)$. If this stronger bound is true, then the algorithm of Theorem 1 would always count triangulations in time $O^*(3.1414^n) = o(|\mathcal{F}_T(P)|)$. Thus setting in the positive the answer of whether or not one can *always* count triangulations of set of points faster than enumerating them.

## 2.2 The results on counting other crossing-free structures

Moving away from triangulations, the other result that will be proven is the following:

**Theorem 2.** *Let $P$ be as before and let $k$ be its number of onion layers. Then the exact values of $|\mathcal{F}_M(P)|$ and $|\mathcal{F}_C(P)|$ can be computed in $n^{O(k)}$ time.*

Thus again, as long as $k = O(1)$, the algorithms of Theorem 2 compute the said numbers in polynomial time, which then gives a partial answer to Problem 16 of The Open Problems Project, which asks whether $|\mathcal{F}_C(P)|$ can *always* be computed in polynomial time, see [14]. This time, however, we are not able to prove a running time of the sort $c^n$ for large $k$, like in Theorem 1.

The general layout of the algorithms of Theorems 1 and 2 is similar to the one found in [15], where these ideas have been used for optimization problems.

## 2.3 A hardness result

Observe that the running times of the algorithms of Theorems 1 and 2 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 running times can be improved to something of the sort $g(k) \cdot n^{O(1)}$, for some function $g$ independent of $n$, thus proving that our problems belong to the FPT complexity class. Which is the class of fixed-parameter tractable problems. However, the techniques involved in the algorithms of Theorems 1 and 2 are general enough to solve harder problems, such as the following:

RESTRICTED-TRIANGULATION-COUNTING-PROBLEM: Given a set of points $P$ and a subset of edges $E$ over $P$, count the triangulations of $P$ that use only edges from $E$.

We also present the following hardness result:

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

The book by J. Flum and M. Grohe, see [16], is a standard reference for Parameterized Complexity Theory, where the classes FPT and W[2] are defined. For now, however, it suffices to say that the separation FPT $\neq$ W[2] is widely believed. Thus an algorithm with a running time of the sort $g(k) \cdot n^{O(1)}$ is most likely not attainable for the RESTRICTED-TRIANGULATION-COUNTING-PROBLEM. This might be an indication 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, in the general non-restricted case.

We will divide the rest of the paper as follows: In § 3 we give a rough idea on how our algorithms work. We prove Theorems 1, 2, and 3 in § 4, § 5, and § 6 respectively. In § 7 we show experiments comparing our algorithm for counting triangulations with the algorithm presented in [11], which is supposed to be *very* fast in practice. We close the paper in § 8 with some conclusions.

# 3    A general framework for counting crossing-free structures

The overall idea of *all* our algorithms can be roughly described as follows. Suppose we want to count the elements of some particular class $\mathcal{F}$ of crossing-free structures on $P$. A set $S$ of non-crossing edges on $P$ is called a *separator* if the union of the edges in $S$ splits the interior of $\mathrm{CH}(P)$, possibly along with $\mathrm{CH}(P)$, into at least two regions. In such a case we will say that $S$ splits $\mathrm{CH}(P)$ into those regions. Now assume that there exists a set $\mathcal{S}$ of separators with the following properties: (1) Every element of $\mathcal{F}$ contains a *unique* separator $S \in \mathcal{S}$, and (2) we can "quickly" enumerate the members of $\mathcal{S}$. With a set of separators $\mathcal{S}$, the elements of $\mathcal{F}$ can be counted as follows: For each $S \in \mathcal{S}$, let $R_1^S$, $R_2^S$, ..., $R_t^S$ be the regions $S$ splits $\mathrm{CH}(P)$ into. Recursively compute the number $n_i^S$ of elements of $\mathcal{F}$ of each region $R_i^S$. The number of elements of $\mathcal{F}$ containing $S$ is then $N^S = \prod_{i=1}^{t} n_i^S$. Thus the total number of elements of $\mathcal{F}$ 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 $\mathcal{S}$. For example, one well-known family of separators $\mathcal{S}$ for triangulations is the set of T-paths, see [12, 13, 1]. We will introduce some other families of separators, some of them with additional properties, however, for the time being we believe that this vague description of how the algorithms work conveys the main idea appropriately.

# 4    Counting triangulations using the onion layers

In this section we will present yet another new algorithm for counting triangulations that uses the onion layers of the given set of points $P$.

Now, 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 $\{1, \ldots, n\}$ such that if $\ell(p) < \ell(q)$

then $p$ also receives a label smaller than $q$. This is clearly possible. Figure 1 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 $\{1, \ldots, n\}$ and when we say "$p \in P$", we will mean the point with label $p$.
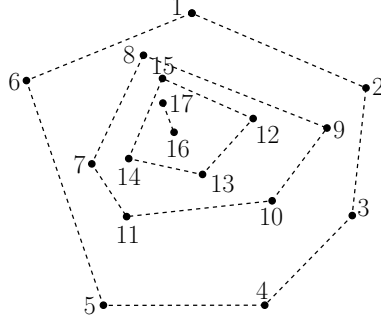


Figure 1: Four onion layers.

Let $T$ be any triangulation of $P$. For $p \in P \setminus P^{(1)}$, let $\text{sn}_T(p)$ 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 $\text{sn}_T(p) < p$. If $p \in P^{(1)}$, we set $\text{sn}_T(p) = p$. When $T$ is clear from context, we will just write $\text{sn}(p)$ instead of $\text{sn}_T(p)$. We denote by $\text{sn-path}_T(p)$ the unique path $p = a_0, a_1, \ldots, a_m$ in $T$ such that for each $0 \leq i < m$, we have that $a_{i+1} = \text{sn}(a_i)$ and $\text{sn}(a_m) = a_m$. 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 $\text{sn-path}(p)$ ends at $p' \in P^{(1)}$ and $\text{sn-path}(q)$ ends in $q' \in P^{(1)}$. There are two paths in $T$ from $p'$ to $q'$ along $\text{CH}(P)$, 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 $\text{CH}(P)$. We call these two regions the sn-regions of $(p, q)$. See Figure 2. 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 $\text{CH}(P)$. Observe that in any triangulation, $\text{CH}(P)$ 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 $apb$, see Figure 3.

## 4.1 The algorithm

Let $ab$ be again an edge on $\text{CH}(P)$. The core idea of our algorithm is as follows: We can easily enumerate all the points $p$ such that the triangle $apb$ appears in some triangulation.
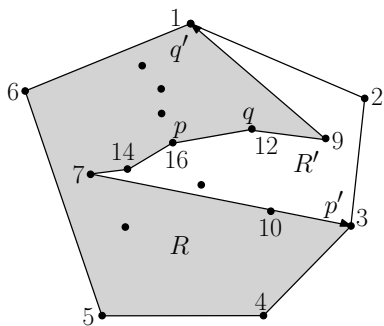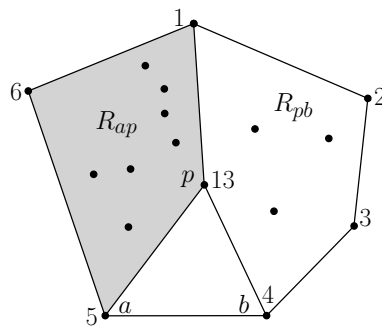
Figure 2: $R$ and $R'$ are the sn-regions of $(p,q)$.

Figure 3: $R_{ap}$ and $R_{pb}$ are the sn-regions of $(a,p)$ and $(p,b)$, respectively, that do not contain triangle $apb$.

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 $\rho$ of $p$ over all triangulations of $P$. For ever pair $(p,\rho)$, let $\mathcal{T}_{(p,\rho)} = \mathcal{T}_{(p,\rho)}(P)$ be the set of triangulations of $P$ that contain the triangle $apb$ and in which $\rho$ 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 $(p,\rho)$, adding the numbers over all pairs gives us the total number of triangulations.

Let us fix a pair $(p,\rho)$ 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 $\rho$ 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 $\rho$ 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 the invariant 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 $\rho_x$ and $\rho_y$ be the sn-paths starting at $x$ and $y$ respectively.

Now, 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 4. Furthermore, we ensure that if $z$ happens to be a point in either $\rho_x$ or $\rho_y$, and $(z,t)$ is an edge in that sn-path, then both $x$ and $y$ are bigger 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 4. 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 $R$, *i.e.*, fewer triangles in any triangulation. The recursion bottoms out when the size is $\leq 1$, 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 $\rho_x$ or $\rho_y$. The direction of descent along that sn-path, starting at $w$, is then the remaining portion of the sn-path of $z$.
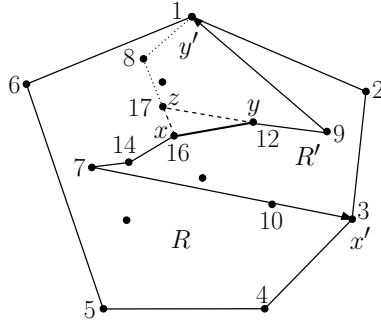


Figure 4: $R$ and $R'$ are the sn-regions of $(x, y)$.

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 $R$? and the answer is: 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 upper layer (a layer containing points with smaller indices). 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 in these cases.

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). Now 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[1] on the running time of the algorithm since, because of memoization, no edge is *traversed* more than once.

---

[1] Up to a polynomial overhead arising from the construction and handling of sub-problems.

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 $(\rho_1, \rho_2, \rho_3)$ 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 $\rho_2'$ be the portion of $\rho_2$ that does not have any point in common with $\rho_1$, and let $\rho_3'$ be the portion of $\rho_3$ that does not have any point in common with either $\rho_1$ or $\rho_2$. The descending paths $\rho_1$, $\rho_2'$ and $\rho_3'$ are vertex disjoint, and along with some additional information they completely describe $\rho_1$, $\rho_2$ and $\rho_3$. The additional information that is required is whether, and where, $\rho_2$ merges with $\rho_1$, and whether, and where, $\rho_3$ 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 $\rho_2'$ may merge with $\rho_1$, and at most $2k$ ways $\rho_3'$ may merge with one of $\rho_1$ or $\rho_2$. Therefore, if $U$ is an upper bound on the number of triples of vertex disjoint descending paths, then $2k^2 U$ is an upper bound on the number of triples $(\rho_1, \rho_2, \rho_3)$ as described above, and hence also an upper bound on the running time of the algorithm.

## 4.2 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 $i$-th 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 the triple of 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 among 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 $\mathrm{CH}\left(P^{(i)}\right)$ 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 $i$-th 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 for $x \geq 1$, the above product is maximized when each $n_i$ is equal to $\frac{n}{k}$. This gives an upper bound of $f\left(\frac{n}{k}\right)^k = g\left(\frac{n}{k}\right)^n$, where $g(x) = f(x)^{\frac{1}{x}}$. 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 $k$-th one must have at least three points, we have $U = O\left(g(3)^n\right)$. The fact that the $k$-th onion layer may have fewer than three points makes only a difference of a constant factor. Therefore the running time of the algorithm presented in this section is $O\left(k^2 g(3)^n\right) = O^*(3.1414^n)$. This concludes the proof of Theorem 1.

We want to point out that often the number of onion layers can be much smaller than the maximum possible $\left\lceil \frac{n}{3} \right\rceil$. For example, Dalal [17] 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 $\Theta\left(n^{2/3}\right)$.

From a theoretical point of view, the algorithm presented in this section, sn-path algorithm for short, has a running time polynomial in $n$ whenever the number of onion layers of $P$ is constant. This is the first known algorithm for counting triangulations having this property. Also, its worst-case behavior is better than the one based on T-paths presented in [1], $O^*(3.1414^n)$ of the former against $O^*(9^n)$ of the latter.

The sn-path algorithm is an excellent candidate for having good experimental behavior as well, due to its polynomial-time instances. Towards the end of paper we shall see how the sn-path performs experimentally against the fastest-known (in practice) algorithm for counting triangulations presented in [11].

# 5 Counting other crossing-free structures

In this section we show how the ideas of the sn-path algorithm can be "modified" or "adapted" so we can develop a general framework that helps to count crossing-free structures in general. We use this framework to count perfect matchings and spanning cycles defined by $P$.

## 5.1 Counting matchings and spanning cycles

Assume we want to count crossing-free matchings spanned by $P$. Clearly any matching can be completed to a triangulation by adding edges, and thus we might want to try the technique used for counting triangulations: Take a set $\mathcal{S}$ of separators and 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 sub-problems.

This simple algorithm fails because some matchings $M$ could be contained in a triangulation that could contain several, say $s_M > 1$, separators and would thus fool our algorithm to count $M$ exactly $s_M$ times. If $s_M = s$ were a constant over all matchings we would not have this problem, however, we are not aware of any set of separators $\mathcal{S}$ with this property.

There is however a way in which we can modify the simple algorithm so that we can count each matching exactly once: We embed each matching $M$ into a unique triangulation $T \supset M$. 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) $\triangle^M$ *constrained* to contain $M$, which we briefly describe next.

**Constrained Delaunay Triangulation:** The constrained Delaunay triangulation (CDT) $\triangle^S$ of $P$ was first introduced in [18]. Formally, it is the triangulation $T$ of $P$ containing $S$ such that no edge $e$ in $T \setminus S$ is flippable in the following sense: Let $\triangle_1, \triangle_2$ be triangles of $P$ sharing $e$. The edge $e$ is flippable if and only if $\square = \triangle_1 \cup \triangle_2$ is convex, and replacing $e$ with the other diagonal of $\square$ increases the smallest angle of the triangulation of $\square$. One of the most important properties of constrained Delaunay triangulations is its uniqueness if no four points of $P$ are cocircular. Thus, under standard non-degeneracy assumptions, there is a unique CDT for any given set of mandatory edges. For a good study on constrained Delaunay triangulations we suggest the book [19] by Ø. Hjelle and M. Dæhlen.

For our counting purposes we will assume that no four points of $P$ are cocircular. This can easily be taken care of by perturbing $P$. We can now go back to our simple algorithm for counting matchings and revise it as follows: Whenever we recurse, in each sub-problem we only count matchings $M$ such that $S \subseteq \triangle^M$, where $S \in \mathcal{S}$ is a separator. If this last condition can be satisfied locally in each sub-problem, *i.e.*, choices in one sub-problem 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 next.

## 5.2 Triangular paths

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 $\rho_p$ which emanates from $p$ and does not intersect the interior of CH $\left(P^{(i)}\right)$.

For any triangulation $T$ of $P$ there is a unique triangle $\triangle_p = p, q_1, q_2$ adjacent to $p$ and intersecting $\rho_p$. Let $q_p$ be the smaller of $q_1$ and $q_2$, using the same labeling as before. Clearly $q_p$ lies in a layer lower than the one containing $p$. Let $p_0, p_1, \ldots, p_r$ be the sequence so that $p_0 = p$, $p_{i+1} = q_{p_i}$, $\forall\, 0 \le i < k$, and $p_r$ lies on the first layer. We call $P_p(T) := \bigcup_i \triangle_{p_i}$ the *triangular path* of $p$ w.r.t. $T$, and we call $p_r$ the *last point* of $P_p(T)$. See Figure 5.

The triangular path $P_p(T)$ 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 $\triangle_{p_i}(T_1) \ne \triangle_{p_i}(T_2)$, then those two triangles intersect, as they both are adjacent to $p$, intersect $\rho_p$ and have interiors free of points in $P$. We are now ready to finish the algorithm for counting matchings.

### 5.2.1 Algorithm for counting matchings

Given a matching $M$, let $\triangle^M$ be the CDT of $M$. By our assumption of no four cocircular points, this CDT is unique for $M$. We annotate $\triangle^M$ as follows:
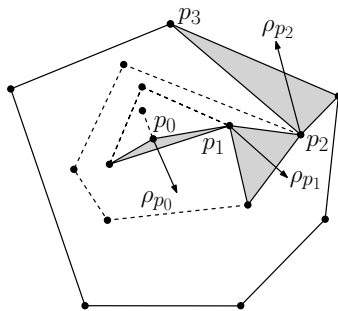
Figure 5: Triangular path $P_p$ starting in onion layer $P^{(4)}$. Onion layers are shown in dashed. $P_p$ can be extended to a triangulation $T$, in such a case $P_p$ will be unique for $T$.

- each vertex $v$ of $\triangle^M$ is annotated with a number $m_v$ that represents the vertex of $M$ that $v$ is matched to. If $m_v$ is 0 say, then we know that $v$ is not matched in $M$.

- each edge $e$ of $\triangle^M$ is annotated with a bit $b_e$ that indicates whether $e$ belongs to $M$ or not.

Let us denote by $\overline{\triangle}^M$ the annotated version of $\triangle^M$. Let $S$ be a separator contained in $\triangle^M$ that splits $\mathrm{CH}(P)$ into regions $R_1, \ldots, R_t$. Separator $S$ inherits all the information from $\overline{\triangle}^M$. We additionally keep track of whether $m_v$ is any of the adjacencies of $v$ in $S$, for each vertex $v \in S$. If not then we set $m_v$ to the index $1 \le i \le t$ of the region $R_i$ the matching vertex of $v$ falls into (it must necessarily be a region having $v$ as a vertex of its boundary). The separator thus annotated will be denoted by $\overline{\triangle}_S^M$.

We say that an annotated constrained Delaunay triangulation is *legal* if and only if it is identical to $\overline{\triangle}^M$, for some matching $M$. Since there is a one-to-one correspondence between matchings 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 $\mathrm{CH}(P)$, 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 6. We enumerate such separators and all possible annotations for each one of them. Each such annotated separator splits $\mathrm{CH}(P)$ into two smaller regions in which we recurse. In each recursive sub-problem we count (legal) annotated constrained Delaunay triangulations consistent with the annotated separator.

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
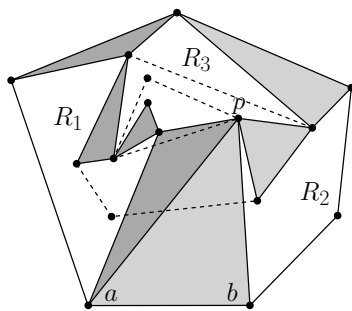
Figure 6: In the first call of the algorithm, the triangular path shown in dark gray is created. It divides the problem into regions $R_1 \cup R_3$ and $R_2$. A call for the latter creates the triangular path shown in light gray. Annotations are not shown for simplicity.

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 clearly at most $n^{O(k)}$ triangular paths. Also, trying all possible annotations per triangular path leads to no more than $n^{O(k)}$ annotations per triangular path, as can be easily checked. Hence there are $n^{O(k)}$ annotated triangular paths, and also $n^{O(k)}$ triples of annotated triangular paths. The overall running time is thus $n^{O(k)}$, which even considers the polynomial overheads arising from checking flippability of edges and inclusion of points into sub-problems. This concludes one part of Theorem 2.

The annotations required for counting matchings are not very complicated, but for many other counting problems this is a highly non-trivial task. An example of more involved annotations is given next, where we consider the problem of counting spanning cycles.

### 5.2.2 Algorithm for counting spanning cycles

Counting spanning cycles is more complicated than counting matchings. What we will actually do is that, instead of counting spanning cycles, we will 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 (which is the root of the cycle), 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 the computed number by $2n$ to get the desired number. In the remainder we use the term HamCycle for rooted and oriented spanning cycles.

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

- each vertex $v$ of $\triangle^H$ is annotated with $(\mathrm{pos}_v, \mathrm{prev}_v, \mathrm{next}_v)$, where $\mathrm{pos}_v$ is the number assigned to $v$ in $H$, $\mathrm{prev}_v$ is the vertex lying immediately before $v$ in $H$, and $\mathrm{next}_v$ is the vertex lying immediately after $v$ in $H$.

- each edge $e$ in $\triangle^H$ is annotated with a bit $b_e$ that indicates whether $e$ belongs to $H$ or not.

As in the case for matchings, the annotated $\triangle^H$ will be denoted by $\overline{\triangle}^H$. Let $S$ be again a separator contained in $\triangle^H$ that splits $\mathrm{CH}(P)$ into regions $R_1, \ldots R_t$. Separator $S$ inherits the following information from $\overline{\triangle}^H$: Each vertex $v \in S$ inherits $\mathrm{pos}_v$ from $\overline{\triangle}^H$. If $\mathrm{prev}_v$ and $\mathrm{next}_v$ are already adjacent to $v$ in $S$ then this information is also inherited. If $\mathrm{prev}_v$ is absent in $S$ then $v$ is annotated with the index $i$, $1 \le i \le t$, of the region $R_i$ that $\mathrm{prev}_v$ falls in. The same holds for $\mathrm{next}_v$. Each edge $e$ of $S$ carries the annotation it has in $\overline{\triangle}^H$. The separator $S$ of $\Delta^H$ thus annotated will be denoted by $\overline{\triangle}_S^H$.

The algorithm, as the reader might be thinking right now, is no other than the algorithm for counting matchings. The only difference are the annotations, they encode a different problem. Thus again, the number of triangular paths is $n^{O(k)}$. The number of annotations per triangular path stays $n^{O(k)}$, and hence the total running time will stay at $n^{O(k)}$, including again the other polynomial overheads. This finishes the proof of Theorem 2.

# 6  The hardness result

In this section we show a hardness result related to our counting algorithms. Observe that those algorithms are parameterized by the number $k$ of onion layers of $P$. They have running times of the sort $n^{O(k)}$. Thus, from the complexity point of view, it is natural to ask whether algorithms with running times of the sort $g(k) \cdot n^{O(1)}$, for some function $g$ independent of $n$, are possible. That would mean that our problems belong to the FPT complexity class. Unfortunately, our techniques are general enough to solve harder problems, such as the RESTRICTED-TRIANGULATION-COUNTING-PROBLEM explained before, on page 4.

Here we prove Theorem 3, which states that the RESTRICTED-TRIANGULATION-COUNTING-PROBLEM, RTCP for short, is W[2]-hard if the parameter is considered to be the number of onion layers of $P$. More, this result even holds for the problem of just deciding the *existence* of a restricted triangulation.

The algorithms of Theorems 1 and 2 require little to no modification to be run on instances of RTCP, that is, those algorithms are quite generic. Since the separation FPT $\neq$ W[2] is widely believed, and we do not really know about the complexity of the counting problems studied in this paper, we can still hope that, by exploiting structural properties, we could obtain fixed-parameter tractable algorithms for them. The book by

J. Flum and M. Grohe, see [16], is an excellent reference for Parameterized Complexity Theory.

## 6.1 Preliminaries

Let $P$ be a set of $n$ points with $k$ onion layers, and let $E$ be some set of pre-specified edges spanned by $P$. We say that a triangulation $T$ of $P$ is *restricted* w.r.t. $E$ if $T \subseteq E$. Here 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 defines the RESTRICTED-TRIANGULATION-COUNTING-PROBLEM in the natural way, and the existence problem is trivially reducible to the counting problem.

The RESTRICTED-TRIANGULATION-EXISTENCE-PROBLEM, RTEP for short, was proven to be NP-complete in [20, 21]. Something very important can be observed here, namely, both reductions are actually parsimonious[2], implying #P-completeness of its natural *counting* problem, RTCP.

So far all reductions involving restricted triangulations rely heavily on the ability to specify a particular set $E$ as part of the input. 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 strongly believe to be #P-complete.

In this section we parameterize RTCP and RTEP by $k$, the number of onion layers of $P$. As we mentioned before, the counting algorithm, for triangulations, presented in Section 4 can easily be adapted to solve RTCP, and thus also RTEP, in time $n^{O(k)}$. Our proof is by reduction from the PARAMETERIZED-HITTING-SET-PROBLEM, PHSP for short, which is proven to be W[2]-hard in [16]. An instance $A$ of this problem is formed by numbers $n, m, k \in \mathbb{N}$, along with sets $S_1, \ldots, S_m \subseteq [n]$, where $k$ is considered a parameter, and $[n] := \{0, \ldots, n-1\}$. 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 [16], 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 pre-specified edges. The gadgets that will be used are called: *pipes*, *wires*, *ORs*, *terminals*, *testers*, and *crossings*, their specifications will be given later on, for now we would like to explain how the gadgets fit together as well as the intuition behind it.

## 6.2 Construction and intuition

Given an instance $A$ of PHSP, as explained above, we will create in polynomial time an instance $G_A = (P, E)$ of RTEP of size $\text{poly}(n, m)$ that has $O(k)$ onion layers and admits

---

[2]This means that there is a one-to-one correspondence between the solution sets.

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. Figure 7 is a reference for the construction that follows.

In the construction, we start with parallel pipes $Q_1, \ldots, Q_k$ of $n$ states each, and of length polynomial in $m$ and $n$. Pipe $Q_i$ lies above pipe $Q_{i+1}$. Let $Q_i$ be a pipe, $1 \leq i \leq k$, and let $S_j = \{s_{j,1}, \ldots s_{j,t}\} \subseteq [n]$ 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 $S_j$, see Figure 7. The stripe $B_{i+1,j}$ will lie in the same 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 Figure 7. For $i = k$, the last or-gadget in $B_{i,j}$ is connected to a terminal-gadget.
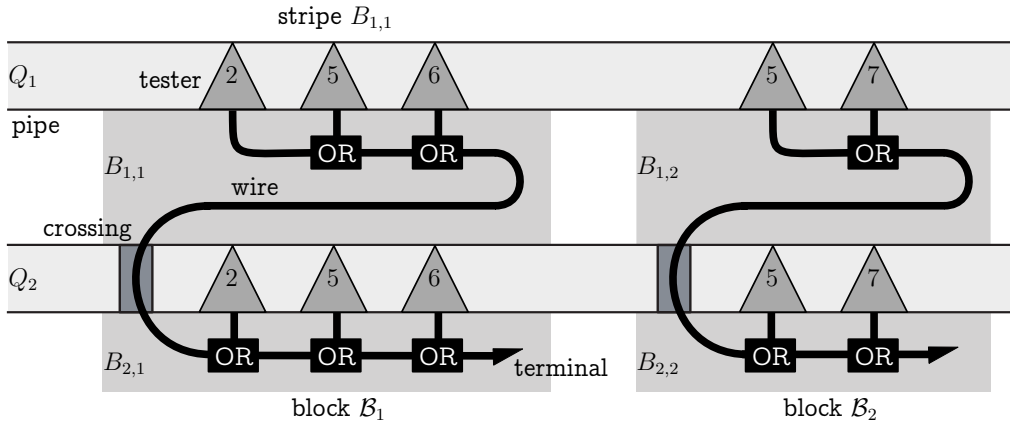


Figure 7: Instance $G_A$ produced from instance $A$ of the PARAMETERIZED-HITTING-SET-PROBLEM with $n = 8, m = 2, k = 2$ and $S_1 = \{2, 5, 6\}$, $S_2 = \{5, 7\}$.

The *block* $\mathcal{B}_j$ is 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 horizontally collinear, that is, they are aligned by their $y$-coordinate.

Finally, $P$ is defined to be the set of points of all the gadgets involved. To define the set $E$ of pre-specified edges, we first include the edges of all gadgets involved. Then, the empty spaces between gadgets are triangulated arbitrarily, and these edges are added to $E$. We now set $G_A = (P, E)$.

The intuition behind the construction is the following: Horizontally, pipe $Q_i$ transmits a single value between 1 and $n$. The testers in stripe $B_{i,j}$ verify if the value transmitted by $Q_i$ hits one of the elements of the set $S_j$ of $A$. If so, this information is transmitted vertically along block $\mathcal{B}_j$, in such a case the transmitted value is true. For this transmission we need ORs, wires and crossing gadgets. At the end of block $B_j$ the terminal gadget can be triangulated iff the value transmitted to it is true. If $S_j$ is not hit by the value transmitted in $Q_i$, then the testers will transmit false and this value will be transmitted vertically along $\mathcal{B}_j$ until it is possibly flipped by another pipe $Q_r$, with

$i < r \leq k$, thus $S_j$ is not hit by $Q_i$ but it is hit by $Q_r$. If the value transmitted to the terminal gadget in block $\mathcal{B}_j$ is false, this means that the terminal cannot be triangulated, thus no restricted triangulation of $G_A$ exists. This in turns implies that $S_j$ was not hit by any value transmitted by the pipes $Q_1, \ldots, Q_k$. If this is always the case then no hitting set of size at most $k$ exists for $A$.

All this will be formally proven later on, for now we believe that this rough intuition is enough. Therefore we will jump now to define the gadgets formally.

## 6.3 Defining the gadgets

The basic gadget is the *pipe*, shown in Figure 8, whose definition is the following:

**Definition 3** (Pipe). *A pipe $Q$ with $n$ states and length $l > 4(n-1)$ consists of points $p_1 \ldots p_l$, $q_1 \ldots q_l$ with $p_t = (t, 0)$, $q_t = (t, 1)$, $1 \leq t \leq l$, and a set $E_Q = S \cup F \cup L_0 \cup \cdots \cup L_{n-1}$ of pre-specified edges. The individual sets that form $E_Q$ are defined as follows:*

*For $1 \leq i \leq n-1$ and $1 \leq t \leq l - 4i$ we define the* zig-edges $a_{i,t} = \{p_t, q_{t+4i}\}$ *and the* zag-edges $b_{i,t} = \{q_{t+4i}, p_{t+1}\}$. *For $i = 0$, we define other zig- and zag-edges by $a_{0,t} = \{p_t, q_{t+1}\}$ and $b_{0,t} = \{q_t, p_t\}$, where this time $1 \leq t \leq l - 1$. For $i \in [n]$, we define the* zig-zag $L_i = \{a_{i,1}, \ldots, a_{i,l-w}, b_{i,1}, \ldots, b_{i,l-w}\}$ *with $w = 1$ for $i = 0$, and $w = 4i$ otherwise.*

*Next, we add the set of* completion edges $S$:

$$S = \{\{p_1, q_t\} \mid 1 \leq t \leq 4(n-1)\} \cup \{\{p_{l-t}, q_l\} \mid 0 \leq t \leq 4(n-1) + 2\}$$

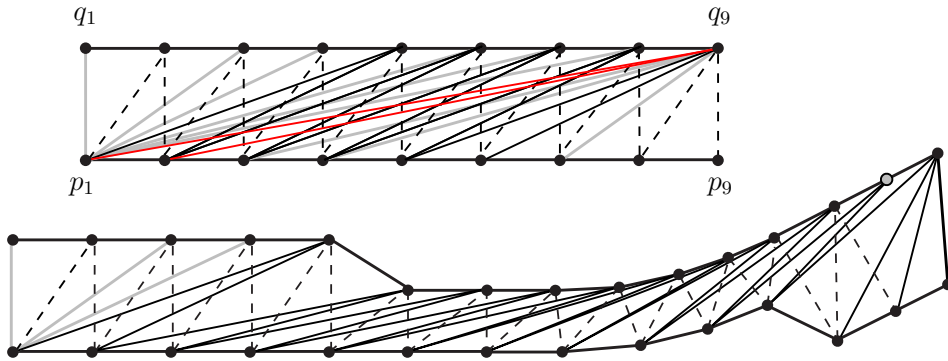*Finally, we add the* frame edges $F = \{\{p_i, p_{i+1}\} \mid i < l\} \cup \{\{q_i, q_{i+1}\} \mid i < l\}$.



Figure 8: (Top) A pipe with 3 states and $l = 9$. Thick black edges constitute $F$, thick gray edges constitute $S$, red edges are $L_2$, solid thin black edges are zig-zag $L_1$, dashed edges are zig-zag $L_0$. (Bottom) A stretched and bent wire with a terminal gadget attached to it.

It is clear that *any* triangulation $T$ of a pipe $Q$ contains exactly one zig-zag $L_i$, for $i \in [n]$, since different zig-zags lines cross. The sets $S, F$ help to complete a triangulation of $Q$ whenever zig-zag $L_i$ is present. If $L_i \subseteq T$, we say that $Q$ "carries" the value $i$ in $T$. Note that $F \subseteq T$ holds for every triangulation $T$ of $Q$. We cannot say the same about $S$ however.

A pipe with $n$ states will always be "horizontal", *i.e.*, it will not turn in any other direction. This is required for the final set of points to feature a bounded number of onion layers.

In our construction we will also require vertical connections between pipes. These are obtained by *wires*, which are pipes with two states. Since they feature only two states, wires can be stretched by arbitrary factors, and bent by arbitrary angles, while increasing their length only by a constant additive term. This is shown in Fig. 8. For wires, we relabel the values 0 and 1 by false and true respectively.

The remaining gadgets for our reduction are specified and defined as follows:

**Or-gadget.** This gadget is connected to two input wires $W_1, W_2$, and to an output wire $W_3$, as shown in Figure 9. We have that: (1) If one of $W_1$ or $W_2$ carries true in some restricted[3] triangulation $T$ of the gadget, then $W_3$ may carry true. (2) If $W_3$ carries true in $T$, then at least one of $W_1$ or $W_2$ must necessarily carry true.
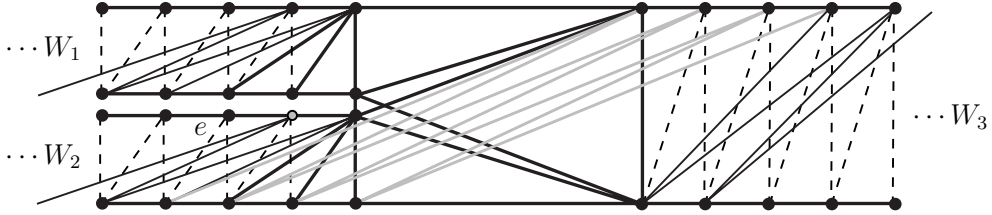


Figure 9: The or-gadget. The gray edges from $W_2$ to $W_3$ are "transfer edges". An analogous set of edges is also present from $W_1$ to $W_3$, but suppressed in this figure to improve legibility.

**A terminal-gadget.** This gadget can be attachted to a wire $W$, replacing its "end part" as exemplified in the bottom part of Figure 8. It admits a triangulation iff $W$ carries true.

**A tester-gadget.** This gadget is connected to a pipe $Q$, for value $i$ at position $t$, between $a_{i,t}$ and $b_{i,t}$, and has an output wire $W$, see to the left in Figure 10. We have that: (1) If $Q$ carries $i$ in some restricted triangulation $T$ of $Q$, then $W$ may carry true. (2) If $W$ carries true, then $Q$ must carry $i$ in $T$.

---

[3]Restricted w.r.t. the shown adjacencies.

**A crossing-gadget.** This is a more intricate gadget which 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 $V$. We have that: (1) If $V$ carries true in some restricted triangulation $T$ of the gadget, then $W$ may carry true. (2) If $W$ carries true, then $V$ must necessarily carry true.

As shown in the middle in Figure 10, $V$ enters the crossing-gadget from the top. If $V$ intersects $Q$ between points $q_t$ and $q_{t+1}$ then a new point $r$ collinear with those two points is added to $Q$. Wire $V$ will now enter $Q$ between $r$ and $q_{t+1}$ instead, as shown in the middle in Figure 10. Let us assume that $Q$ is an $n$-state pipe, and consider the set $S$ formed by the points $p_u$ such that $a_{i,u}$ is a zig-edge, of zig-zag $L_i$, adjacent to $q_{t+1}$, with $0 \le i \le n-1$. By definition of $a_{i,u}$ we have that $u = t - 4i + 1$ for $1 \le i \le n-1$, and $u = t$ for $i = 0$. There will be an output wire $W_i$, for zig-zag $L_i$, which will go out from $Q$ between $p_u \in S$ and $p_{u+1}$.

Since pipes and wires are purely combinatorial objects, we have some freedom to move their points without affecting the adjacencies between the $p$'s and $q$'s, and without losing collinearities. Thus we will move all the $p$ points of $Q$ from $p_{t-4(n-1)+1}$ to $p_t$ to the right, and condense them in such a way that we keep their linear order, thus we also keep the planarity of the zig-zags $L_i$, $0 \le i \le n-1$. The condensing part is also done in such a way that the following empty convex quadrilateral $C_u^i$ for zig-zag $L_i$ at $p_u \in S$ exists: Both diagonals of $C_u^i$ have negative slopes. One diagonal of $C_u^i$ is formed by $r$ and $p_{u+1}$. The other diagonal of $C_u^i$ is form by the point $\alpha_u$ of $V$, which is vertically aligned with $r$ and lies three points behind $r$ on $V$, and the point $\beta_u$ of $W_i$ which is vertically aligned with $p_{u+1}$ and lies three points ahead of $p_{u+1}$ on $W_i$. Points $\alpha_u$ and $\beta_u$, for $u = t - 3$, can be seen in the middle and to the right in Figure 10. Observe that this re-arrangement of elements is always possible.

Now, the zig-edge $a_{i,u}$ of $L_i$ adjacent to $q_{t+1}$ is replaced by the edges $a'_{i,u} = \{p_u, r\}$ and $\{p_{u+1}, r\}$. The latter edge is a diagonal of $C_u^i$ and is shown in red in Figure 10. The rest of the adjacencies of $L_i$ remains the same.

Intuitively speaking, the red edge $\{p_{u+1}, r\}$ will help $V$ to transmit false to $W_i$, as seen to the right in Figure 10 for $i = 1$. Thus we also need to add the edges that will help $V$ to transmit true to $W_i$. Those edges are shown on solid black for $i = 1$ to the right in Figure 10. The adjacencies are equivalent for any other $0 \le i \le n-1$. Observe that all these adjacencies intersect neither $a'_{i,u}$ nor $b_{i,u}$.

Finally, the output wires $W_0, \dots, W_{n-1}$ are connected to a chain of or-gadgets, as shown in the middle in Figure 10, whose output is precisely the output wire $W$.

## 6.4 Formal proofs

**Lemma 1.** *All the previous gadgets fulfill their specifications.*
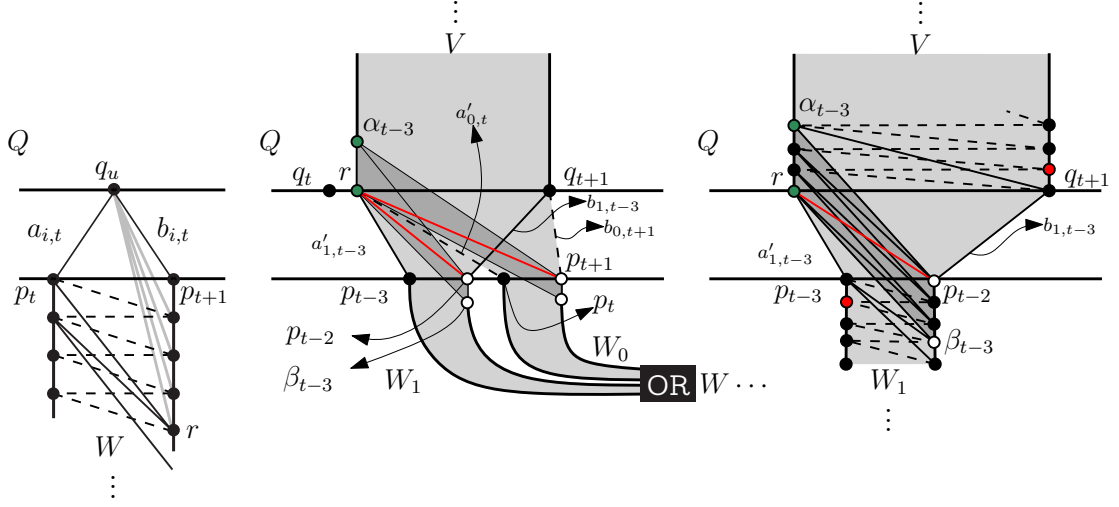
Figure 10: To the left the tester-gadget for $i$ at $t$. $Q$ is modified by shifting, for $k > 0$, all $p_{t+k}$ and $q_{u+k}$ to the right until the triangle $r, p_{t+1}, q_u$ is oriented counter-clockwise. In the middle a crossing between pipe $Q$ and input wire $V$ which becomes output wire $W$. To the right the details of the crossing for $i = 1$ at $p_{t-3}$.

*Proof.* **Or-gadget:** (1) Assume without loss of generality that $W_2$ carries true in some restricted triangulation $T$ of the gadget. Observe that $W_3$ carries true or false in $T$ depending on whether the transfer edges, shown in gray in Figure 9, are chosen. For (2) note that if $W_3$ carries true in $T$, the transfer edges from either $W_1$ or $W_2$, say $W_2$ without loss of generality, must be present. If $W_2$ carried false, it can do it only up to edge $e$ shown in Figure 9, since all following edges intersect transfer edges. But then, the gray point fails to be part of a triangle in *any* restricted triangulation of the gadget, thus $W_2$ must necessarily carry true in $T$.

**Terminal:** If $W$ carries true, the terminal is triangulated as shown at the bottom of Figure 8. However, if $W$ carries false, then the gray point shown in the same figure fails to be in a triangle of the restricted triangulation of the gadget.

**Tester:** For (1) assume $Q$ carries $i$ in some restricted triangulation $T$. Since no gray edges in the tester intersect $L_i$, they can be added to $T$ or not. That would make $W$ carry true or false respectively. For (2) given some restricted triangulation $T$, in which $W$ carries true, all gray edges must be present in $T$. But the gadget is designed such that for every $0 \le j \ne i \le n - 1$, there is an edge $e \in L_j$ of $Q$ that intersects both $a_{i,t}$ and $b_{i,t}$, and thus all gray edges. Therefore $e \notin T$, and hence $L_j \not\subseteq T$, forcing $L_i \subseteq T$.

**Crossing:** (1) Let $T$ be a restricted triangulation of the whole gadget in which $Q$ carries $i$. Observe that if either, $V$ or $W_i$, carries true in $T$, then the black solid edges that

20

cross $Q$ from $V$ to $W_i$, shown to the right in Figure 10 for $i = 1$, must be present. Those edges in turn imply that the other gadget must necessarily carry true in $T$ as well, otherwise the red points shown to the right in Figure 10 will fail to be part of $T$, which would give us a contradiction since $T$ is a triangulation. Thus $V$ carries true iff $W_i$ carries true, as long as $Q$ carries $i$. By using the chain of or-gadgets that the $W_j$'s are connected to we could leave the output wire $W$ carrying true. (2) Assume that $W$ carries true and $Q$ carries $i$ in $T$. Observe that in the chain of or-gadgets that the $W_j$'s are connected to, we can always force to transmit true from $W$ to $W_i$, while we transmit false to every other $W_j$, $0 \le j \ne i \le n - 1$. This in turn will force the edges that cross $Q$ from $W_i$ to $V$ to be included in $T$, the black solid edges that cross $Q$ from $W_i$ to $V$ shown to the right in Figure 10 for $i = 1$. This will make $V$ carry true.

A triangulation is also possible if $V$ and $W$ carry false. If $Q$ carries $i$ in $T$, then the edges $a'_{i,u} = \{p_u, r\}$ and $\{p_{u+1}, r\}$, for some $t - 4(n - 1) + 1 \le u \le t$, are also present in $T$. Thus we transmit false from $W$ to every $W_j$, $0 \le j \le n - 1$. However, as we said before, the red edge $\{p_{u+1}, r\}$ will help to transmit false from $W_i$ to $V$ through $Q$. Thus $V$ would also carry false in $T$.  ∎

Theorem 3 follows from the following lemma:

**Lemma 2.** $G_A$ *has $O(k)$ onion layers and admits a triangulation iff $A$ admits a hitting set of size $\le k$.*

*Proof.* 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 $O(1)$ overhead, thus giving $O(1)$ different $y$-coordinates. Since the points in stripes $B_{i,1}, \ldots, B_{i,m}$ are aligned by their $y$-coordinates, each set $B_{i,1} \cup \cdots \cup B_{i,m}$ has $O(1)$ different $y$-coordinates. This totals to $2k + O(k) = O(k)$ different $y$-coordinates among all points in $P$.

Given a hitting set $H = \{x_1, \ldots, x_k\}$ of $k$ elements, we construct a triangulation that uses only edges from $E$ as follows: For every $i \le k$, make $Q_i$ carry $x_i$. For every $j \le m$ pick some $x = x_i \in H$ such that $x \in S_j$. In stripe $B_{i,j}$ triangulate the output wire of the tester for $x$ to carry true, and transmit this true value along the or-gadgets of $B_{i,j}$. When crossing a pipe $Q_z$, with $z > i$, the true value will get transmitted through the output wire $W_z$ of the corresponding crossing-gadget. The true value will eventually reach the terminal of $\mathcal{B}_j$, which can then be triangulated without problems.

On the other hand, the values $H = \{x_1, \ldots, x_k\}$ carried by the pipes $Q_1, \ldots, Q_k$ in any restricted triangulation of $G_A$ form a hitting set. To see this, observe that every terminal must be triangulated, so 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 top-down, and say it lies in stripe $B_{i,j}$. This or-gadget must be connected to a tester that outputs true. This implies $x_i \in S_j$ and $H \cap S_j \ne \emptyset$.  ∎

# 7 Experimental results on counting triangulations

We have implemented the sn-path algorithm presented in § 4, and in this section we compare it with the algorithm presented in [11]. All experiments were run on a server generously provided by Prof. Bernd Finkbeiner, head of the Reactive Systems group at Saarland University. All implementations are single-threaded, so all algorithms were run a on single core of a dual-core processor AMD Opteron at 2.6 Ghz. Linux was the used operating system, and the amount of RAM available was 122 GB. Finally, all implementations use the GMP library to handle big numbers. All statistics reported here were obtained from the output of the command 'time -v'.

The main idea behind the experiments was to obtain evidence of the practical limits of the algorithms, thus they are really provided without statistical analysis. If we denote the number of points by $n$, the number of onion layers by $k$, and the size of the convex hull by $h$, we were interested in knowing for different values of those parameters what are the largest sets of points we can solve. Also, besides providing the number of triangulations, we also provide: (1) Total number of sub-problems generated by each algorithm, (2) Memory consumption, and (3) Total running time. Since we used memoization in all algorithms, the total number of sub-problems is just the size of the database at the end of execution.

We have four kinds of sets of points, of selected cardinalities, we ran the algorithms on: (1) Sets of points having three onion layers. (2) Sets of points generated in a square (3) Sets of points having the largest possible number of onion layers, w.r.t. the cardinality of the set (4) Grids. Sets (1), (2) and (3) were generated at random. For (1) we generated random points on three concentric circles and we only kept configurations having three onion layers.

Table 1 summarizes the largest sets of points, of each type, that we were able to solve within 140 hours, the complete results can be seen in the tables at the end of the paper. Results for (1) are shown in Tables 2 and 3, for (2) in Tables 4 and 5, for (3) in Tables 6 and 7, and for (4) in Tables 8 and 9. In the tables the algorithm of [11] is called "ray-seidel", and our algorithm simply "sn-paths". We did not run ray-seidel on sets of kind (4) due to degeneracy; this functionality was not implemented. All columns are self-explanatory except for the columns "Base" and "Exp". The former refers to the base $c$, truncated to two decimal digits, of a number expressed as $c^n$. The latter refers to the term $d$, also truncated to two decimal digits, of a number expressed as $n^{d \cdot k}$, this makes sense for sn-paths since we know that for fixed $k$ the running time is $n^{O(k)}$.

Since we are interested in the largest $n$ we can solve, we started the experiments with at least 25 points, below this threshold all algorithms perform very well, where ray-seidel is notably the fastest, giving answers in at most a couple of seconds, and sn-paths the slowest for $k = 8$. All empty entries, except for the last entry of sn-paths in Tables 4 and 5, mean that the corresponding algorithm consumed all available RAM memory *before* finishing the corresponding set of points. In the same sense, one complete empty row means that *no* algorithm managed to finish the corresponding set of points. The

|              | # Points |      |      |      |
|--------------|:--------:|:----:|:----:|:----:|
|              | (1)      | (2)  | (3)  | (4)  |
| ray-seidel   | 43       | 43   | 34   | NA   |
| sn-paths     | 80       | 43   | 28   | 6x17 |

Table 1: Largest sets of points of kind $(i)$, $1 \leq i \leq 4$, solved by each algorithm within 140 hours.

last entry of sn-paths in tables 4 and 5 was explicitly stopped due to its potentially large running time.

To verify the correctness of the algorithms we ran them on configurations available in [22, 23], for which an answer is known via other algorithms. We also run them on sets of points in convex position, there the number of triangulations is a Catalan number. In all cases the two algorithms confirmed the known answers.

The experiments turned out to be what we had expected, namely, generally worse behavior as the number of onion layers increases, since the number of triangulations should certainly increase with the number of onion layers. The experiments show however that *both* algorithms are counting triangulations by generating far fewer sub-problems: Having a glimpse at the number of sub-problems in Tables 2 to 9, each looks as something of the sort $\sqrt{|\mathcal{F}_T(P)|}$, which was already reported in [11] for the ray-seidel algorithm.

The ray-seidel algorithm showed a consistent behavior across all experiments. This algorithm lived up to its expectations, it turned out to be simply the fastest algorithm, but this came with the price of being *very* resource-consuming. We can see in the tables that the resources the algorithm uses increase very fast, at that rate we could say that increasing RAM to a couple of Terabytes will not really allow us to run the algorithm on significantly larger set of points. However, there are other techniques we could use to alleviate this situation, we could for example decide to store only a subset of the produced sub-problems and re-compute a sub-problem whenever needed. Since apparently computing sub-problems is very fast, we could expect that this method does not severely blow up the running time.

Now turning to sn-paths, the algorithm really performed best for three convex layers, see Tables 2 and 3. For those configurations the algorithm allowed us to go up to 80 points in a "reasonable" amount of time without exhausting the RAM, which is almost twice as much as the ray-seidel algorithm allowed. In this regard we believe that increasing computational power and resources, say 512 GB of RAM, could allow us to go somewhere near 160 points. The ray-seidel algorithm would get nowhere close to this number of points *per se*. This "nice" behavior can also be seen in Tables 8 and 9 with grids having three and four onion layers, however, grids are believed to have far less triangulations than sets of points in general position. This is also supported by the experiments. For fewer than three onion layers the algorithm gets better, so they are really not an issue. Also, since the running time of sn-paths can be expressed as $n^{d \cdot k}$, for some positive $d$,

the idea of column Exp in the tables was to see whether that value comes out roughly as a constant for the same values of $k$, however, the data set seems to be small to show this. We believe that the right value should be $3 \leq d \leq 4$. Finally, by increasing the number of onion layers, Tables 4 to 7, we can see how the behavior of sn-paths quickly deteriorates, in all aspects.

The conclusions of the experiments really suggest themselves. In the "low" end, up to 25 points, any algorithm will do but ray-seidel is the fastest. In the "high" end it really seems that sn-paths is a better choice due to its smaller memory footprint, so up to 6–7 onion layers we would stick with it, but beyond that number of onion layers we would consider ray-seidel a better option.

# 8   Conclusions

In this paper we have presented algorithms to count triangulations, crossing-free matchings and crossing-free spanning cycles of a given set of points $P$. All algorithms use the onion layers of $P$ and the divide-and-conquer paradigm.

The algorithm to count triangulations presented in this paper has the best provable worst-case running time as of this writing, $O^*(3.1414^n)$. We consider important to note that configurations of points having from $\Omega\left(3.464^n\right)$, see [24], to $\Omega\left(8.65^n\right)$, see [25], triangulations are known. Thus the algorithm presented in this paper counts triangulations faster than enumeration algorithms in at least those cases. Also, this algorithm has polynomial-time instances whenever the number of onion layers of the given set of points is constant. As we saw in the experiments we showed, for up to 3 onion layers this algorithm outperforms the algorithm of [11], which is reported to be extremely fast in practice. At this point the most interesting open questions w.r.t. counting triangulations are the following: (1) Is it possible to *always* count triangulations in polynomial time? or is this problem #P-complete? (2) Is it true that *every* set of $n$ points, $n$ being large enough, spans at least $\Omega\left(3.464^n\right)$ triangulations? If this is true, then the algorithm we presented in this paper *always* counts triangulations in $o\left(|\mathcal{F}_T(P)|\right)$.

Speaking about counting other kinds of crossing-free structures, we showed again two algorithms. These algorithms could be seen more as a framework for counting essentially "every" kind of crossing-free structures, since it depends on a labeling scheme, which is the hardest part of the algorithm to come up with. This "framework" implies algorithms with running times of the sort $n^{O(k)}$, where $k$ is the number of onion layers of the given set of points, which again, for fixed $k$ implies polynomial time. Algorithms like these were not known before for this kind of problems. This gives a partial answer to Problem 16 of The Open Problems Project, which asks whether $|\mathcal{F}_C(P)|$ can *always* be computed in polynomial time, see [14]. These counting algorithms also allow us to generate crossing-free matchings and spanning cycles uniformly at random. The latter being a problem that has attracted the attention of researchers for almost 20 years, in the form of generating random simple polygons on $P$, which is nothing but a crossing-free spanning cycle of $P$. Since our algorithms are based on the divide-and-conquer

paradigm, we can adapt the method explained in [12] to produce such random structures, once the counting has been done. Other methods to generate random simple polygons without having to count are known, for example, in [26] many heuristics for polygons are presented. There the authors reported that (uniform) random generation can be done in polynomial time when the random polygon is star-shaped, but in the general case the algorithms therein presented are either unpractical or unable to generate uniformly at random.

It is worth noting that although we could have tried to come up with an annotation scheme for pseudo-triangulations, and thus we could have obtained an algorithm to count pseudo-triangulations using the onion layers, the resulting algorithm would have had a running time of the sort $n^{O(k)}$. This running time is in general not better than that of the algorithm presented in [1] for pseudo-triangulations, at least from the theoretical point of view, since for the latter we have $O^*(c^n)$, for some constant $c \in \mathbb{R}$, while for the former $k$ can get linearly large. The most important open problem here is whether the number of matchings and spanning cycles can *always* be computed in polynomial time.

We also showed a hardness result of a very particular instance of the problem of counting triangulations exactly, namely the RESTRICTED-TRIANGULATION-COUNTING-PROBLEM (RTCP). We showed that this problem is W[2]-hard if the parameter is the number of onion layers of the set of points it is defined on. The algorithm for counting triangulations of Theorem 1 needs little to no modifications to be run on instances of RTCP, and the separation FPT $\neq$ W[2] is widely believed, so we can still hope that, by exploiting structural properties of triangulations, and also of other crossing-free structures, we can obtain FPT algorithms for the counting problems studied in § 4 and § 5. Thus, another interesting question at this moment is: Do those counting problems belong to FPT or not?

Finally, we showed experimental results comparing the algorithms for counting triangulations of Theorem 1 and the algorithm of [11]. Those experiments give a rough idea of what to expect when running each one of those algorithms on real configurations of points. It would be very interesting to see a hybrid algorithm that uses the sn-path and the ray-seidel algorithms, if possible. That algorithm could combine the small memory footprint of the sn-path algorithm with the fast execution of the ray-seidel algorithm. This could allow us to solve larger sets of points.

# 9    Acknowledgement

|  |  |  |  | Time in hh:mm:ss.ms | | RAM in Mb | |
| $n$ | $h$ | #Triangulations | Base | ray-seidel | sn-paths | ray-seidel | sn-paths |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 30 | 10 | 1610014656152655441 | $\approx 3.74$ | 20.18 | 55.77 | 303 | 33 |
|  | 10 | 312513373686594183 | $\approx 3.82$ | 1:07.72 | 1:09.17 | 1160 | 38 |
| 33 | 11 | 32155601714553665796 | $\approx 3.90$ | 3:50.96 | 2:19.19 | 2762 | 62 |
|  | 11 | 68598010833407738067 | $\approx 3.99$ | 58.43 | 2:30.94 | 857 | 62 |
| 37 | 12 | 93349476679230323509429 | $\approx 3.92$ | 1:53.60 | 3:43.51 | 1521 | 90 |
|  | 12 | 311130688130120764443512 | $\approx 4.05$ | 3:20.41 | 4:42.24 | 2465 | 97 |
| 40 | 14 | 2642143054680217856074126 | $\approx 4.07$ | 18:12.16 | 8:10.26 | 12557 | 131 |
|  | 15 | 2903778262295075928823011 | $\approx 4.08$ | 7:01.40 | 9:50.98 | 4325 | 149 |
| 43 | 14 | 452371697808162396583055656 | $\approx 4.16$ | 1:34:48 | 21:39.66 | 53263 | 243 |
|  | 14 | 461550214764369881018564051 | $\approx 4.16$ |  | 19:25.53 |  | 242 |
| 47 | 16 | 157759710540671985436621922639 | $\approx 4.18$ |  | 32:46.68 |  | 363 |
|  | 15 | 341037585238678346710372748758 | $\approx 4.24$ |  | 39:33.42 |  | 420 |
| 50 | 16 | 54782168649020627430413001433261 | $\approx 4.31$ |  | 1:06:53 |  | 606 |
|  | 16 | 158997592723683977758501079915910 | $\approx 4.40$ |  | 1:07:19 |  | 553 |
| 60 | 21 | 3830519327225667656835917480230390004428 | $\approx 4.56$ |  | 7:51:53 |  | 2006 |
|  | 20 | 19003078026633792670003377149358696361338 | $\approx 4.69$ |  | 8:59:36 |  | 2063 |
| 70 | 25 | 48142357864275890897765127796753253695164862078 | $\approx 4.64$ |  | 22:59:43 |  | 3937 |
|  | 23 | 224411547729672469823709078962020667530864087588 | $\approx 4.74$ |  | 26:54:49 |  | 4506 |
| 80 | 26 | 39697885166896605395758278879689934038647554224464030090 | $\approx 4.81$ |  | 81:29:23 |  | 8932 |
|  | 26 | 18527998227718271520712658325966253393485474040452858832 | $\approx 4.90$ |  | 90:34:22 |  | 9617 |

Table 2: $n$ random points on a square, having three onion layers and $h$ points on their convex hull.

|   |   | # Sub-problems | | | | |
|---|---|---|---|---|---|---|
| $n$ | $h$ | ray-seidel | Base | sn-paths | Base | Exp |
| 30 | 10 | 2050514 | $\approx 1.62$ | 215732 | $\approx 1.50$ | $\approx 1.20$ |
|    | 10 | 7879754 | $\approx 1.69$ | 246657 | $\approx 1.51$ | $\approx 1.21$ |
| 33 | 11 | 18992928 | $\approx 1.66$ | 405580 | $\approx 1.47$ | $\approx 1.23$ |
|    | 11 | 5812991 | $\approx 1.60$ | 410357 | $\approx 1.47$ | $\approx 1.23$ |
| 37 | 12 | 10027300 | $\approx 1.54$ | 575255 | $\approx 1.43$ | $\approx 1.22$ |
|    | 12 | 16250100 | $\approx 1.56$ | 626274 | $\approx 1.43$ | $\approx 1.23$ |
| 40 | 14 | 82635240 | $\approx 1.57$ | 866278 | $\approx 1.40$ | $\approx 1.23$ |
|    | 15 | 28333612 | $\approx 1.53$ | 982791 | $\approx 1.41$ | $\approx 1.24$ |
| 43 | 14 | 347603518 | $\approx 1.57$ | 1604269 | $\approx 1.39$ | $\approx 1.26$ |
|    | 14 |  |  | 1591423 | $\approx 1.39$ | $\approx 1.26$ |
| 47 | 16 |  |  | 2287764 | $\approx 1.36$ | $\approx 1.26$ |
|    | 15 |  |  | 2720786 | $\approx 1.37$ | $\approx 1.28$ |
| 50 | 16 |  |  | 3631525 | $\approx 1.35$ | $\approx 1.28$ |
|    | 16 |  |  | 3998798 | $\approx 1.35$ | $\approx 1.29$ |
| 60 | 21 |  |  | 12527119 | $\approx 1.31$ | $\approx 1.33$ |
|    | 20 |  |  | 13076694 | $\approx 1.31$ | $\approx 1.33$ |
| 70 | 25 |  |  | 23762305 | $\approx 1.27$ | $\approx 1.33$ |
|    | 23 |  |  | 27937551 | $\approx 1.27$ | $\approx 1.34$ |
| 80 | 26 |  |  | 54047260 | $\approx 1.24$ | $\approx 1.35$ |
|    | 26 |  |  | 58561612 | $\approx 1.25$ | $\approx 1.36$ |

Table 3: Number of sub-problems generated by the configurations (entry-wise) presented in Table 2.

| $n$ | $k$ | $h$ | #Triangulations | Base | Time in hhh:mm:ss.ms | | RAM in Mb | |
|---|---|---|---|---|---|---|---|---|
| | | | | | ray-seidel | sn-paths | ray-seidel | sn-paths |
| 30 | 5 | 9 | 29762284427845618 | $\approx 3.54$ | 7.60 | 3:56.61 | 130 | 141 |
| | 6 | 7 | 54648952555202115 | $\approx 3.61$ | 30.69 | 16:17.12 | 470 | 535 |
| 33 | 5 | 11 | 8830953374442248378 | $\approx 3.75$ | 34.80 | 14:47.26 | 643 | 394 |
| | 6 | 7 | 23407918365649149382 | $\approx 3.86$ | 15.10 | 1:10:34 | 288 | 1292 |
| 37 | 5 | 11 | 83171978925687988832050 | $\approx 3.91$ | 3:35.27 | 1:15:00 | 2796 | 1524 |
| | 5 | 13 | 153476097829879667672248 | $\approx 3.97$ | 15:23.53 | 2:16:32 | 10707 | 1957 |
| 40 | 6 | 12 | 1146138971033715203926926 | $\approx 3.99$ | 25:42.43 | 13:29:43 | 18525 | 8889 |
| | 7 | 10 | 50504932821694624290012536 | $\approx 4.14$ | 1:35:45 | 46:49:41 | 54128 | 25533 |
| 43 | 6 | 10 | 98140331329825983429202925 | $\approx 4.24$ | 3:20:54 | 107:48:48 | 116506 | 37407 |
| | 7 | 8 | | | | | | |

Table 4: $n$ random points on a square, having $k$ onion layers and $h$ points of their convex hull.

| $n$ | $k$ | $h$ | # Sub-problems | | | | |
|---|---|---|---|---|---|---|---|
| | | | ray-seidel | Base | sn-paths | Base | Exp |
| 30 | 5 | 9 | 854579 | $\approx 1.56$ | 947262 | $\approx 1.58$ | $\approx 0.80$ |
| | 6 | 7 | 3150228 | $\approx 1.64$ | 3590878 | $\approx 1.65$ | $\approx 0.73$ |
| 33 | 5 | 11 | 4245399 | $\approx 1.58$ | 2554063 | $\approx 1.56$ | $\approx 0.84$ |
| | 6 | 7 | 1907449 | $\approx 1.54$ | 8731943 | $\approx 1.62$ | $\approx 0.76$ |
| 37 | 5 | 11 | 18477670 | $\approx 1.57$ | 9735430 | $\approx 1.54$ | $\approx 0.89$ |
| | 5 | 13 | 70483691 | $\approx 1.62$ | 12535632 | $\approx 1.55$ | $\approx 0.90$ |
| 40 | 6 | 12 | 121049523 | $\approx 1.59$ | 56587195 | $\approx 1.56$ | $\approx 0.80$ |
| | 7 | 10 | 354717051 | $\approx 1.63$ | 155716531 | $\approx 1.60$ | $\approx 0.73$ |
| 43 | 6 | 10 | 752596823 | $\approx 1.60$ | 239084256 | $\approx 1.56$ | $\approx 0.85$ |
| | 7 | 8 | | | | | |

Table 5: Number of sub-problems generated by the configurations (entry-wise) presented in Table 4.

| k | n | #Triangulations | Base | Time in hh:mm:ss.ms | | RAM in Mb | |
|---|---|---|---|---|---|---|---|
| | | | | ray-seidel | sn-paths | ray-seidel | sn-paths |
| 9 | 25 | 248441701550196 | ≈ 3.76 | 7.82 | 1:36:07 | 154 | 3828 |
| | 27 | 6632755933105064 | ≈ 3.85 | 1:16.84 | 9:56:09 | 1239 | 17406 |
| 10 | 28 | 134806114688321888 | ≈ 4.09 | 1:04.33 | 39:29:17 | 1130 | 56197 |
| | 28 | 259051751512786147 | ≈ 4.18 | 58.55 | 32:31:58 | 903 | 41745 |
| 11 | 32 | 188748482026800154083 | ≈ 4.30 | 1:43:22 | | 70647 | |
| | 33 | 2680138023948109608080 | ≈ 4.46 | 1:38:04 | | 62477 | |
| 12 | 34 | 166051861631664445755560 | ≈ 4.50 | 2:52:16 | | 114676 | |
| | 34 | | | | | | |

Table 6: $n$ random points having $k = \lceil \frac{n}{3} \rceil$ onion layers.

| k | n | # Sub-problems | | | |
|---|---|---|---|---|---|
| | | ray-seidel | Base | sn-paths | Base |
| 9 | 25 | 1078399 | ≈ 1.74 | 24811886 | ≈ 1.97 |
| | 27 | 8738535 | ≈ 1.80 | 110817524 | ≈ 1.98 |
| 10 | 28 | 8015023 | ≈ 1.76 | 347448787 | ≈ 2.01 |
| | 28 | 6303203 | ≈ 1.74 | 266661064 | ≈ 1.99 |
| 11 | 32 | 478692844 | ≈ 1.86 | | |
| | 33 | 423236754 | ≈ 1.82 | | |
| 12 | 34 | 773361622 | ≈ 1.82 | | |
| | 34 | | | | |

Table 7: Number of sub-problems generated by the configurations (entry-wise) presented in Table 6.

| $n$ | $m$ | $k$ | #Triangulations | | Time in hhh:mm:ss.ms | RAM in Mb |
|---|---|---|---|---|---|---|
| | | | | Base | sn-paths | sn-paths |
| 6 | 6 | 3 | 260420548144996 | $\approx 2.51$ | 10.36 | 16 |
| 6 | 7 | 3 | 341816489625522032 | $\approx 2.61$ | 45.79 | 41 |
| 6 | 8 | 3 | 464476385680935656240 | $\approx 2.69$ | 2:31.70 | 107 |
| 6 | 9 | 3 | 645855159466371391947660 | $\approx 2.76$ | 7:28.22 | 213 |
| 6 | 10 | 3 | 913036902513499041820702784 | $\approx 2.81$ | 17:27.63 | 460 |
| 6 | 11 | 3 | 1306520849733616781789190513820 | $\approx 2.85$ | 39:35.88 | 840 |
| 6 | 12 | 3 | 1887591165891651253904039432371172 | $\approx 2.89$ | 1:22:15 | 1555 |
| 6 | 13 | 3 | 2747848427721241461905176361078147168 | $\approx 2.93$ | 2:50:42 | 2479 |
| 6 | 14 | 3 | 4024758386310801427793602374466243714608 | $\approx 2.96$ | 5:14:23 | 4439 |
| 6 | 15 | 3 | 5924744736041718687622958191829471010847132 | $\approx 2.98$ | 8:43:14 | 6315 |
| 6 | 16 | 3 | 8757956199571261116690226598764501142088496860 | $\approx 3.01$ | 14:55:29 | 10344 |
| 6 | 17 | 3 | 12991215957916577635251095613859465176216530106080 | $\approx 3.03$ | 26:08:38 | 15023 |
| 7 | 7 | 4 | 1999206934751133055518 | $\approx 2.72$ | 6:09.75 | 187 |
| 7 | 8 | 4 | 1216940995414198870718052 | $\approx 2.80$ | 36:59.53 | 869 |
| 7 | 9 | 4 | 7608333633329475136555549189994 | $\approx 2.87$ | 2:28:42 | 2344 |
| 7 | 10 | 4 | 4847725121672666884983996329182196 | $\approx 2.93$ | 8:12:59 | 6465 |
| 7 | 11 | 4 | 3131521959869770128138491287826065904 | $\approx 2.97$ | 27:42:55 | 14870 |
| 7 | 12 | 4 | 204437676119275998232172917694684494885488 | $\approx 3.01$ | 67:06:41 | 34752 |
| 8 | 8 | 4 | 3326338408441131037515979995920 | $\approx 2.89$ | 6:00:49 | 6171 |
| 8 | 9 | 4 | 93693635175012088195304299672807080 | $\approx 2.96$ | 32:49:11 | 19071 |
| 8 | 10 | 4 | 2696211097537325182524932578284131372728 | $\approx 3.02$ | 139:58:01 | 75220 |

Table 8: Grid of $n$ by $m$ with $k$ onion layers.

| | | | # Sub-problems | | |
|---|---|---|---|---|---|
| $n$ | $m$ | $k$ | sn-paths | Base | Exp |
| 6 | 6 | 3 | 69908 | $\approx 1.36$ | $\approx 1.03$ |
| 6 | 7 | 3 | 207193 | $\approx 1.33$ | $\approx 1.09$ |
| 6 | 8 | 3 | 465416 | $\approx 1.31$ | $\approx 1.12$ |
| 6 | 9 | 3 | 1002029 | $\approx 1.29$ | $\approx 1.15$ |
| 6 | 10 | 3 | 1883205 | $\approx 1.27$ | $\approx 1.17$ |
| 6 | 11 | 3 | 3409331 | $\approx 1.25$ | $\approx 1.19$ |
| 6 | 12 | 3 | 5705962 | $\approx 1.24$ | $\approx 1.21$ |
| 6 | 13 | 3 | 9417222 | $\approx 1.22$ | $\approx 1.22$ |
| 6 | 14 | 3 | 14471156 | $\approx 1.21$ | $\approx 1.24$ |
| 6 | 15 | 3 | 22201708 | $\approx 1.20$ | $\approx 1.25$ |
| 6 | 16 | 3 | 32491047 | $\approx 1.19$ | $\approx 1.26$ |
| 6 | 17 | 3 | 46979052 | $\approx 1.18$ | $\approx 1.27$ |
| 7 | 7 | 4 | 972496 | $\approx 1.32$ | $\approx 0.88$ |
| 7 | 8 | 4 | 3527752 | $\approx 1.30$ | $\approx 0.93$ |
| 7 | 9 | 4 | 10558836 | $\approx 1.29$ | $\approx 0.97$ |
| 7 | 10 | 4 | 25013282 | $\approx 1.27$ | $\approx 1$ |
| 7 | 11 | 4 | 55453561 | $\approx 1.26$ | $\approx 1.02$ |
| 7 | 12 | 4 | 109901193 | $\approx 1.24$ | $\approx 1.04$ |
| 8 | 8 | 4 | 14569428 | $\approx 1.29$ | $\approx 0.99$ |
| 8 | 9 | 4 | 50333235 | $\approx 1.27$ | $\approx 1.03$ |
| 8 | 10 | 4 | 122283519 | $\approx 1.26$ | $\approx 1.06$ |

Table 9: Number of sub-problems generated by the configurations (entry-wise) presented in Table 8.

# References

[1] V. Alvarez, K. Bringmann, and S. Ray, "A simple sweep line algorithm for counting triangulations and pseudo-triangulations," *Submitted*, 2012. 1, 2, 3, 5, 10, 25

[2] M. Ajtai, V. Chvátal, M. Newborn, and E. Szemerédi, "Crossing-free subgraphs," in *Theory and Practice of Combinatorics A collection of articles honoring Anton Kotzig on the occasion of his sixtieth birthday* (G. S. Peter L. Hammer, Alexander Rosa and J. Turgeon, eds.), vol. 60 of *North-Holland Mathematics Studies*, pp. 9 – 12, North-Holland, 1982. 2

[3] M. Sharir, A. Sheffer, and E. Welzl, "Counting plane graphs: Perfect matchings, spanning cycles, and kasteleyn's technique," *CoRR*, vol. abs/1109.5596, 2011. 2

[4] A. G. Olaverri, M. Noy, and J. Tejel, "Lower bounds on the number of crossing-free subgraphs of $k_n$," *Comput. Geom.*, vol. 16, no. 4, pp. 211–221, 2000. 2

[5] M. Sharir and A. Sheffer, "Counting triangulations of planar point sets," *Electr. J. Comb.*, vol. 18, no. 1, 2011. 2

[6] M. Sharir, A. Sheffer, and E. Welzl, "On degrees in random triangulations of point sets," *J. Comb. Theory, Ser. A*, vol. 118, no. 7, pp. 1979–1999, 2011. 2

[7] A. Sheffer, "Numbers of plane graphs." `http://www.cs.tau.ac.il/~sheffera/counting/PlaneGraphs.html`. 2

[8] E. Demaine, "Simple polygonizations." `http://erikdemaine.org/polygonization/`. 2

[9] N. Katoh and S. ichi Tanigawa, "Fast enumeration algorithms for non-crossing geometric graphs," *Discrete & Computational Geometry*, vol. 42, no. 3, pp. 443–468, 2009. 2

[10] A. Razen and E. Welzl, "Counting plane graphs with exponential speed-up," in *Rainbow of Computer Science* (C. S. Calude, G. Rozenberg, and A. Salomaa, eds.), vol. 6570 of *Lecture Notes in Computer Science*, pp. 36–46, Springer, 2011. 2

[11] S. Ray and R. Seidel, "A simple and less slow method for counting triangulations and for related problems," in *EuroCG*, 2004. 2, 5, 10, 22, 23, 24, 25

[12] O. Aichholzer, "The path of a triangulation," in *Symposium on Computational Geometry*, pp. 14–23, 1999. 2, 3, 5, 25

[13] A. Dumitrescu, B. Gärtner, S. Pedroni, and E. Welzl, "Enumerating triangulation paths," *Comput. Geom.*, vol. 20, no. 1-2, pp. 3–12, 2001. 3, 5

[14] E. Demaine, J. S. B. Mitchell, and J. O'Rourke, "Problem 16: Simple polygonalizations." `http://cs.smith.edu/~orourke/TOPP/P16.html#Problem.16`. 4, 24

[15] E. Anagnostou and D. G. Corneil, "Polynomial-time instances of the minimum weight triangulation problem," *Comput. Geom.*, vol. 3, pp. 247–259, 1993. 4

[16] J. Flum and M. Grohe, *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series, Springer, 2006. 5, 15

[17] K. Dalal, "Counting the onion," *Random Struct. Algorithms*, vol. 24, no. 2, pp. 155–165, 2004. 10

[18] L. P. Chew, "Constrained delaunay triangulations," in *Symposium on Computational Geometry*, pp. 215–222, 1987. 11

[19] Ø. Hjelle and M. Dæhlen, *Triangulations and Applications (Mathematics and Visualization)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. 11

[20] E. L. Lloyd, "On triangulations of a set of points in the plane," in *FOCS*, pp. 228–240, IEEE Computer Society, 1977. 15

[21] A. Schulz, "The existence of a pseudo-triangulation in a given geometric graph," in *EuroCG*, 2006. 15

[22] O. Aichholzer, "Counting triangulations - olympics." `http://www.ist.tugraz.at/staff/aichholzer/research/rp/triangulations/counting/`. 23

[23] V. Kaibel and G. M. Ziegler, "Counting lattice triangulations," in *Surveys in Combinatorics* (C. D. Wensley, ed.), vol. 307 of *London Mathematical Society Lecture Note Series*, pp. 277–307, Cambridge University Press, 2003. 23

[24] F. Santos and R. Seidel, "A better upper bound on the number of triangulations of a planar point set," *J. Comb. Theory, Ser. A*, vol. 102, no. 1, pp. 186–193, 2003. 24

[25] A. Dumitrescu, A. Schulz, A. Sheffer, and C. D. Tóth, "Bounds on the maximum multiplicity of some common geometric graphs," in *STACS* (T. Schwentick and C. Dürr, eds.), vol. 9 of *LIPIcs*, pp. 637–648, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011. 24

[26] T. Auer and M. Held, "Heuristics for the generation of random polygons," in *CCCG* (F. Fiala, E. Kranakis, and J.-R. Sack, eds.), pp. 38–43, Carleton University Press, 1996. 25