Counting Triangulations and other Crossing-free Structures via Onion Layers

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

February 12, 2015

Abstract

Let P be a set of n points in the plane. A crossing-free structure on P is a straight-edge plane graph with vertex set P. Examples of crossing-free structures include triangulations and spanning cycles, also known as polygonalizations. In recent years, there has been a large amount of research trying to bound the number of such structures; in particular, bounding the number of (crossing-free) 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. 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. Additionally, we show that our algorithm for counting triangulations in the worst case over all k takes time $O^*(3.1414^n)$[I]. Given that there are several well-studied configurations of points with at least $\Omega(3.47^n)$ triangulations, and some even with $\Omega(8.65^n)$ triangulations, our algorithm asymptotically outperform any enumeration algorithm for such instances. 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 that in order to be fixed-parameter tractable, our general algorithm must rely on additional properties that are specific to the considered class of crossing-free structures.

1 Introduction

Let $P \subset \mathbb{R}^2$ be a finite set of n points in general position. A crossing-free structure on P is a straight-line plane graph whose vertex set is precisely P. Typical examples of crossing-free structures are (crossing-free) triangulations, (crossing-free) spanning cycles, (crossing-free)

*Fachrichtung Informatik, Universität des Saarlandes, alvarez@cs.uni-saarland.de.
†Institute of Theoretical Computer Science, ETH Zurich, karlb@inf.ethz.ch.
‡Fachrichtung Informatik, Universität des Saarlandes, curticapean@cs.uni-saarland.de
§Max-Planck-Institut für Informatik. saurabh@mpi-inf.mpg.de.

[I] In the notation $\Omega^*(\cdot), O^*(\cdot)$, and $\Theta^*(\cdot)$ we neglect polynomial terms and we just present the dominating exponential term.

matchings, and (crossing-free) spanning trees. Throughout the paper we only consider crossing-free structures, therefore, when referring to triangulations, matchings, spanning cycles, etc., we always assume them to be crossing-free.

Given a class of crossing-free structures $\mathcal{F}$, e.g., triangulations, one can naturally ask the following two questions: (1) What upper and lower bounds on the number of elements of type $\mathcal{F}$ can be given over all possible sets of n points in the plane? (2) Given P, how fast can the (exact) number of elements of type $\mathcal{F}$ on P be computed? With respect to the first question, 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 (all) crossing-free graphs on every set of n points [6]. This 10^{13n} bound implies that the size of *each* class $\mathcal{F}$ of crossing-free structures can be upper-bounded by a number of type c^n, where $c = c(\mathcal{F}) \in \mathbb{R}^+$ depends on the particular class $\mathcal{F}$. Since then, research has focused on tightening upper and lower bounds on c for many different classes of crossing-free structures. Table 1 gives the currently best asymptotic bounds on the number of triangulations, spanning cycles, perfect matchings, and spanning trees, which are among the most popular and hence most studied crossing-free structures. The symbols $\leq, \geq$ should be understood as upper and lower bound, respectively.

	Triangulations	Spanning cycles	Perfect matchings	Spanning trees
$\forall P \leq$	$O(30^n)$ [30]	$O(54.55^n)$ [32]	$O(10.05^n)$ [33]	$O(141.07^n)$ [24]
$\forall P \geq$	$\Omega(2.43^n)$ [31]	1	$\Omega^*(2^n)$ [22]	$\Omega^*(6.75^n)$ [20]
$\exists P \leq$	$O^*(3.47^n)$ [4]	1	$O^*(2^n)$ [22]	$O^*(6.75^n)$ [20]
$\exists P \geq$	$\Omega(8.65^n)$ [19]	$\Omega(4.64^n)$ [22]	$\Omega^*(3^n)$ [22]	$\Omega(12.52^n)$ [25]

Table 1: *Asymptotic bounds for various classes of crossing-free structures on P (a set of n points in the plane). The selected (gray) lower bounds are tight.*

It is interesting to point out that the number of spanning cycles, perfect matchings, and spanning trees has been proven to be minimum when P is in convex position. That is, the selected (gray) lower bounds in Table 1 are tight. The interested reader is referred to the work of Aichholzer et al. [3] for a list of other classes of crossing-free structures on P whose cardinality is minimized when P is in convex position, and to the work of Dumitrescu et al. and Sheffer [19, 34] for up-to-date lists of asymptotic bounds for other crossing-free structures.

The second question on crossing-free structures mentioned above is of algorithmic flavor: We consider the problem of *computing* the number of crossing-free structures of a *particular* class, say triangulations, for a *given* input set of points P. This problem is closely related to the problem of sampling crossing-free structures of a particular class uniformly at random. A first approach to the counting problem would be to produce *all* elements of the class, using well-known methods for enumeration [2, 14, 15, 26], and then simply output the number of enumerated elements. This has the obvious disadvantage that the total time spent will be, at best, linear in the number of elements counted. This number, however, is in general exponential in n (the size of the input P). Thus, the following question arises naturally: Can we count crossing-free structures of a given class in time sub-linear in the number of elements counted? This question has in general been much less studied. Until very recently (year 2012) it was only known that this is always possible for the class of *all* plane graphs [28], while for triangulations it was only known to be *sometimes* possible [9]. Empirically, there were other algorithms to count triangulations that are observed to count faster than enumeration [1, 27], but that, until now, have no theoretical runtime guarantees. For spanning trees, matchings, and spanning cycles no efficient counting algorithm was known. However, for some non-trivial classes of spanning cycles (monotone), it was known that exact counting can be done in polynomial time in n [36].

So far we have discussed the literature on counting crossing-free structures at the time when the preliminary version of this paper appeared [7]. We believe it is important to first list our contributions (§ 2) before we elaborate on the newest developments (§ 3), that happened during a very short period of time, and in particular, while this paper was under review.

2 Our contributions

In this paper we present three counting algorithms: To count (1) triangulations, (2) matchings, and (3) spanning cycles. In order to formally state the results contained in this paper we need the following definition, see also Figure 1.

Definition 1 (Onion layers)**.** *Let P be a set of n points in 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 at most $\lceil \frac{n}{3} \rceil$. Let us now denote by $\mathcal{F}_T(P), \mathcal{F}_M(P)$ and $\mathcal{F}_C(P)$ the classes of all triangulations, matchings, and spanning cycles of P, respectively. Our first contribution is the following:

Theorem 1. *Let P be a set of n points in the plane, and let k be its number of onion layers. Then the exact value of $|\mathcal{F}_T(P)|$ can be computed in time $O^*\left(f(\frac{n}{k})^k\right)$, where $f(x) = \frac{x^3+3x^2+2x+2}{2}$. Since $k \leq \lceil \frac{n}{3} \rceil$, 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.*

We remark that (1) the algorithm of Theorem 1 has a better worst-case guarantee than the previously best algorithm for counting triangulations [9], which runs in time $O^*\left(9^n\right)$. (2) It is the first algorithm that can compute the exact value of $|\mathcal{F}_T(P)|$ in *polynomial time* restricted to a non-trivial subset of all instances (constant number of onion layers). (3) As stated before, for *every* set P of n points in the plane, the cardinality of $\mathcal{F}_T(P)$ is at least $\Omega(2.43^n)$, but it has been conjectured that this bound can be improved to $\Omega\left(\sqrt{12}^n\right) \approx \Omega(3.47^n)$ [4, 5, 29]. If this stronger bound is true, then our algorithm counts triangulations in time $O^*(3.1414^n) = o(|\mathcal{F}_T(P)|)$, i.e., faster than by using enumeration algorithms, which was not known to be possible up to year 2012, see also § 3.

Theorem 2. *Let P be a set of n points in the plane 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.*

Again, the algorithms of Theorem 2 compute the exact number of matchings and spanning cycles in polynomial time if the number of onion layers is $k = O(1)$. This gives a partial answer to Problem 16 of The Open Problems Project, which asks whether $|\mathcal{F}_C(P)|$ can be computed in polynomial time [18]. However, in Theorem 2 we are not able to prove a running time guarantee of the form c^n for large k, as in Theorem 1.

The general layout of the algorithms of Theorems 1 and 2 is similar to the one by Anagnostou et al. [12], where these ideas have been used for optimization problems.

Observe that the running times 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 whether these running times can be improved to $g(k) \cdot n^{O(1)}$, for some function g that does not depend on n, thus proving that our problems belong to the complexity class FPT, 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 more general 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 whose set of edges is a subset of E. We prove the following.

Theorem 3. *The* RESTRICTED-TRIANGULATION-COUNTING-PROBLEM *is W[2]-hard if the parameter is 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 [21] is a standard reference for parameterized complexity theory, where the classes FPT and W[2] are defined. The separation FPT $\neq$ W[2] is widely believed. Thus, an algorithm with a running time of the form $g(k) \cdot n^{O(1)}$ for the RESTRICTED-TRIANGULATION-COUNTING-PROBLEM is unlikely to exist. This indicates that we 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.

The rest of the paper is structured as follows: In § 3 we briefly elaborate on the developments that occurred while this paper was under review. We prove Theorems 1 and 2 in § 4 and § 5, respectively. The proof of Theorem 3 is not contained in this extended abstract but can be found in the ArXiv version of this paper [8], where we also present experiments comparing our algorithm for counting triangulations (Theorem 1) with the empirically fast algorithm of Ray et al. [27]. We conclude our paper in § 6.

3 Subsequent developments on algorithmic counting (2013–2014)

While this paper was under review many important developments occurred regarding the problem of counting crossing-free structures algorithmically. We briefly list these developments in this section.

In 2013 a new, and rather simple, algorithm for counting triangulations was presented [11]. This algorithm has a worst-case running time of $O^*(2^n)$ — setting finally in the positive the question whether enumeration algorithms for triangulations can always be beaten, as *every* set of n points in the plane has at least $\Omega(2.43^n)$ triangulations. However, the new algorithm does not seem to have polynomial time instances, unlike the algorithm for counting triangulations presented in this paper, which runs in polynomial time when the input set P has a fixed number of onion layers. In fact, experiments [11] show that when the number of onion layers of P is small, the algorithm presented in this paper greatly outperforms the algorithm by Alvarez et al. [11].

Regarding other classes of crossing-free structures, many strong algorithms were presented in 2014 [35]. These algorithms build upon the ideas by Alvarez et al. [11]. Among other results, it was shown that that the number of *all* crossing-free structures can be computed in time $O^*(2.839^n)$, improving over previous results [28], and it was shown that perfect matchings can be counted in time $O^*(2^n)$. These algorithms show again that enumeration can, at least in these cases, *always* be beaten. It is, however, still open whether for spanning trees and spanning cycles the same can be proven. Preliminary results in this direction can also be found in [35]. As before, these new algorithms seem not to have polynomial time instance, unlike the algorithms presented in this paper for counting matchings and spanning cycles.

We now proceed to the description of our algorithms and the proofs of Theorems 1 and 2.

4 Counting triangulations using onion layers

In this section we present our algorithm for counting triangulations, which we call **sn-path algorithm**. Its main ingredient are geometric separators derived from the onion layers of the given set of points P.

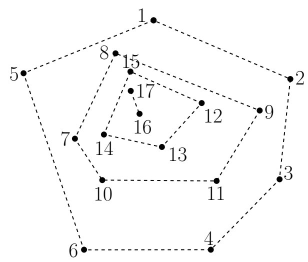

Figure 1: *Four onion layers. The cyclic order of the labels in a layer is not necessary.*

For any point $p \in P$ 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 shows the onion layers of a set of 17 points and the labels assigned to them. From now on we refer to the points of P by their labels, i.e., we think of P as the set $\{1, \dots, n\}$ and when we say "$p \in P$", we mean the point with label p.

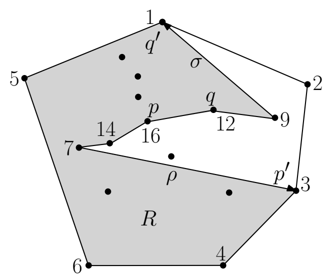

Figure 2: *R is the sn-region of (ρ, σ) with starting points p, q and endpoints p', q'.*

A *descending path* is a sequence of points $\rho = (p_1, \dots, p_k)$ with $\ell(p_{i+1}) < \ell(p_i)$ for all $1 \le i < k$. Consider any crossing-free set of straight-line edges T on P; think of T as a (partial) triangulation. A descending path ρ is *maximal w.r.t.* T if the edges of ρ are contained in T and ρ cannot be extended by edges in T. For any $p \in P$ we construct a unique maximal descending path w.r.t. T starting in p, which we call **sn-path**: For any $q \in P$, if all neighbors q' of q in T have $\ell(q') \ge \ell(q)$, then set $\mathrm{sn}_T(q) = \perp$. Otherwise let $\mathrm{sn}_T(q)$ be the neighbor of q in T with smallest label; in this case $\ell(\mathrm{sn}_T(q)) < \ell(q)$. Then the (unique) descending path $\rho = (p_1, \dots, p_k)$ with (1) $p_1 = p$, (2) $p_{i+1} = \mathrm{sn}_T(p_i)$ for all $1 \le i < k$, and (3) $\mathrm{sn}_T(p_k) = \perp$ is the sn-path of p w.r.t. T. Note that every sn-path consists of at most one point from each onion layer. Also note that for $T' \subseteq T$ if ρ is an sn-path w.r.t. T then it is also an sn-path w.r.t. T'. Any descending path satisfying (1) and (2), but not necessarily (3) is called a *partial sn-path* of p w.r.t. T.

Let ρ, σ be descending paths starting in p, q and ending in p', q'. Let $U = U(\rho, \sigma)$ be the union of the edges of ρ, σ and the edge (p, q). We call (ρ, σ) *legal* if (1) U is crossing-free, (2) ρ and σ are sn-paths w.r.t. U, and (3) ρ and σ end in $P^{(1)}$. In this case ρ, σ induce a region $R = R(\rho, \sigma)$ whose boundary is the union of U and the part of CH(P) from p' to q' in clockwise order, see Figure 2. We call R the *sn-region* of (ρ, σ). A *triangulation of* R is a maximal set of triangles with vertices in P partitioning R such that no triangle contains a point of P in its interior. 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 specific triangulation.

For descending paths ρ, δ, σ we let $\Delta = \Delta(\rho, \delta, \sigma)$ be the triangle formed by the starting points of ρ, δ, σ, and we let $U = U(\rho, \delta, \sigma)$ be the union of the edges of ρ, δ, σ, and Δ. We say that (ρ, δ, σ) is legal if (1) U is crossing-free, (2) ρ, δ, and σ are sn-paths w.r.t. U, (3) ρ, δ, and σ end in $P^{(1)}$ and (4) Δ is free of points from P, apart from its vertices. See Figure 3. Observe that this implies that (ρ, δ), (δ, σ), and (ρ, σ) are legal, since if ρ is an sn-path w.r.t. $U(\rho, \delta, \sigma)$ then it is also an sn-path w.r.t. $U(\rho, \delta) \subseteq U(\rho, \delta, \sigma)$.

4.1 The sn-path algorithm

Our algorithm recursively solves the following problem. Given legal descending paths (ρ, σ), count the number of triangulations T of $R(\rho, \sigma)$ satisfying the following **sn-constraint**: ρ and σ are sn-paths w.r.t. T. We denote the result of instance (ρ, σ) by $\#(\rho, \sigma)$.

Initially, we pick vertices p, q of CH(P) that are consecutive in clockwise order. Set $\rho = (p)$, $\sigma = (q)$. Note that ρ, σ are the sn-paths of p, q w.r.t. any set of edges T, as no point v has $\ell(v)$ smaller than $\ell(p)$ or $\ell(q)$. Thus, the sn-constraint of (ρ, σ) is trivially satisfied. Moreover, the boundary of $R(\rho, \sigma)$ is the whole convex hull of P. Hence, $\#(\rho, \sigma)$ is simply the total number of triangulations of P, as desired.

In order to recursively solve an instance (ρ, σ), we enumerate all descending paths δ such that (ρ, δ, σ) is legal. We return $\sum_\delta \#(\rho, \delta) \cdot \#(\delta, \sigma)$, where the sum ranges over all enumerated δ.

Note that both sn-regions $R(\rho, \delta)$ and $R(\delta, \sigma)$ have size smaller than $R(\rho, \sigma)$, i.e., fewer triangles in any triangulation. The recursion ends when the size is 0, in which case we know that there is exactly one triangulation, or when there are is no δ that makes (ρ, δ, σ) legal, in which case the result is 0.

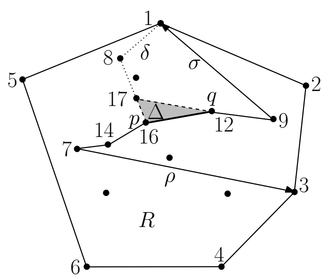

Figure 3: *In the recursive step of the sn-path algorithm, we split the region $R = R(\rho, \sigma)$ along the descending path δ and the triangle $\Delta = \Delta(\rho, \delta, \sigma)$.*

4.2 Correctness

Consider an instance (ρ, σ) with sn-region $R = R(\rho, \sigma)$. We show that (1) every object counted by our algorithm corresponds to a unique triangulation of R satisfying the sn-constraint, and (2) every triangulation of R satisfying the sn-constraint is counted at least once.

For (1), fix some enumerated $\tilde{\delta} = \delta(\rho, \sigma)$ for a given instance (ρ, σ). In both recursive subproblems $(\rho, \tilde{\delta})$ and $(\tilde{\delta}, \sigma)$ fix some enumerated descending paths $\delta(\rho, \tilde{\delta}), \delta(\tilde{\delta}, \sigma)$ as well. Iteratively fix in each recursive subproblem (ρ', σ') some third path $\delta(\rho', \sigma')$. Consider the union S over all recursive subproblems (ρ', σ') (arising from all the sn-paths that we iteratively fixed) of the set $\{\rho', \sigma', \delta(\rho', \sigma'), \Delta(\rho', \delta(\rho', \sigma'), \sigma')\}$. Note that our algorithm counts all objects S. Consider the union T of all edges of all descending paths and all triangles in such a counted set S. Note that T is crossing-free, since at the start of each recursive call (ρ', σ') we have not picked any edges in the interior of the current region $R' \subseteq R$ yet, and every new descending path δ' and triangle $\Delta(\rho', \delta', \sigma')$ that we construct does not cross the boundary of R'. Moreover, T is a triangulation of R, since we repeatedly add triangles and the only base case of the recursion in which we return a non-zero number is when the size of the current region R' is 0, in which case it is already triangulated.

We show that T satisfies the sn-constraint, i.e., ρ and σ are sn-paths w.r.t. T. Assume for contradiction that the sn-path condition of ρ is not satisfied in T for some point a of ρ. Let b be the successor of a on ρ. Consider the first recursive subproblem (ρ', σ'), with third path δ', where we violate the sn-path condition of ρ at a, i.e., δ' or $\Delta(\rho', \delta', \sigma')$ contains an edge (a, c) with $c < b$. Since a appears on the boundary of R and the edge (a, c) is contained in the current region $R' \subseteq R$, the point a also appears on the boundary of R', i.e., a is contained in ρ' or σ', say in ρ'. Its successor on ρ' has label at least b, since (a, c) is the first edge that we add with $c < b$. Thus, the edge (a, c) violates the sn-path condition not only of ρ, but also of ρ', since $c < b$. However, we explicitly check that $(\rho', \delta', \sigma')$ is legal, so we check that ρ' is an sn-path w.r.t. a set $U(\rho', \delta', \sigma')$ that contains the added edge (a, c). This is a contradiction, which implies that the sn-path property is preserved at every point a of ρ. A symmetric statement holds for edges on σ. Hence, each counted object corresponds to a triangulation of R satisfying the sn-constraint.

To see that there is no overcounting, let S_1, S_2 be two counted objects and consider any recursive subproblem (ρ', σ') where they diverge, i.e., where we choose different δ_1 and δ_2 when constructing S_1 and S_2. If $\Delta(\rho', \delta_1, \sigma') \neq \Delta(\rho', \delta_2, \sigma')$, then these triangles are intersecting, so that the triangulations corresponding to S_1 and S_2 are different. Otherwise, δ_1 and δ_2 have the same starting point z. Observe that all further choices produce triangulations in which δ_1 (or δ_2, respectively) is the sn-path of z. Since sn-paths are unique and $\delta \neq \delta'$, the triangulations corresponding to S_1 and S_2 are different.

For (2), consider any triangulation T of R satisfying the sn-constraint. Let p, q be the starting points of ρ, σ. Recall that (p, q) is an edge of the boundary of R. If (p, q) is also an edge of ρ then ρ is (p, q) followed by σ, because two merged sn-paths cannot split again. Thus R has size 0 and we return 1. If (q, p) is an edge of σ we have a symmetric case. Otherwise, in T the points p, q form a triangle with a third point z. Let δ be the sn-path of z w.r.t. T. Observe that (ρ, δ, σ) is legal. Thus, recursively we construct T as a union of sn-paths and triangles, and T is counted in the product $\#(\rho, \delta) \cdot \#(\delta, \sigma)$.

4.3 Running Time

We add one important ingredient for efficiency: Memoization. Whenever we have computed the answer to a recursive subproblem, we store it in a dictionary data structure, such as a hash table. This way, we can bound the total running time of the algorithm by summing the time it takes to enumerate δ over all legal descending paths (ρ, σ). Since all checks take polynomial time, the total running time can be bounded, up to polynomial factors, by the number M of triples $(\rho, \sigma, \hat{\delta})$, where (ρ, σ) are legal descending paths and $\hat{\delta}$ is any intermediate path constructed during the enumeration of all possible δ. Observe that for enumerating δ we can build a descending path step by step, making sure that at all points in time $\hat{\delta}$ is a partial sn-path (w.r.t. $U(\rho, \hat{\delta}, \sigma)$), and that ρ and σ stay sn-paths (w.r.t. $U(\rho, \hat{\delta}, \sigma)$). For any such triple $(\rho, \sigma, \hat{\delta})$ counted by M, let σ' be the portion of σ that does not have any points in common with ρ, and let δ' be the portion of $\hat{\delta}$ that does not have any points in common with ρ or σ. The descending paths ρ, σ', δ' are crossing-free and vertex-disjoint. Moreover, we can reconstruct σ from (ρ, σ') if we know whether σ has a point in common with ρ and what is the first such point. This is because once two sn-paths merge their remaining portions are equal, as subpaths of sn-paths are also sn-paths and thus unique. Thus, we need $O(\log n)$ bits to reconstruct σ from (ρ, σ'). Similarly, we can reconstruct the partial sn-path $\hat{\delta}$ from (ρ, σ, δ') if we know its length and whether and where it merges with ρ or σ, which can be encoded using $O(\log n)$ bits. Hence, we can bound $M \leq 2^{O(\log n)} N = O^*(N)$, where N is the number of crossing-free vertex-disjoint triples of descending paths.

It is left to prove an upper bound for N, which is also an upper bound on the total running time up to polynomial factors. Each descending path uses at most one point 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 point, 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 $f(x) = 1 + 3x + 6\frac{x(x-1)}{2} + 3\frac{x(x-1)(x-2)}{6}$, which can be simplified to $\frac{1}{2}(x^3 + 3x^2 + 2x + 2)$.

The number of triples of non-crossing vertex-disjoint descending paths is therefore $N \leq \prod_{i=1}^{k} f(n_i)$. Since each n_i is a positive integer, and the function $f(\cdot)$ is log-concave[II] 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 $f\left(\frac{n}{k}\right)^k \leq (\frac{n}{k})^{O(k)}$. Alternatively, we can bound the running time by $g\left(\frac{n}{k}\right)^n$, where $g(x) = f(x)^{\frac{1}{x}}$ 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 $N = O\left(g(3)^n\right)$. 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 $O^*(g(3)^n) = O^*(\sqrt[3]{31}^n) = O^*(3.1414^n)$. 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

[II] $f(\cdot)$ is log-concave iff $f(\alpha x + (1-\alpha)y) \geq f(x)^\alpha \cdot f(y)^{1-\alpha}$ for every x, y in the domain of f and $0 \leq \alpha \leq 1$.

$k = \Theta\left(n^{2/3}\right)$. Using Markov's inequality, this implies that with high probability[III] we have $N \leq 2^{n^{2/3+o(1)}}$. Hence, with high probability our algorithm runs in sub-exponential time for points randomly distributed on a disk.

5 Counting other crossing-free structures

In this section we show how the ideas of the sn-path algorithm can be augmented in order to develop a general framework for counting many classes of crossing-free structures. We use this framework to count matchings and spanning cycles.

The overall idea can be roughly described as follows. Suppose we want to count the elements of a 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) CH(P) into at least two regions, say regions $R_1^S, R_2^S, \ldots, R_t^S$. 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 $S \in \mathcal{S}$, (2) choosing an element of $\mathcal{F}$ with separator S can be done *independently* in the regions R_i^S, and (3) we can quickly enumerate the members of $\mathcal{S}$. With such a set of separators $\mathcal{S}$, the elements of $\mathcal{F}$ can be counted as follows: 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. We fill in the details of this approach in the following sections.

5.1 Annotations

Assume we want to count all matchings spanned by P. We have to ensure that each vertex that is contained in the separator S is matched consistently in all of its incident regions. In any matching M that fits to a separator S, each vertex in S is unmatched, or matched to a vertex strictly within some region R_i^S, or 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 ensure consistency with the current annotation in all recursive calls.

In general, the choice of the annotation scheme heavily depends on the class of crossing-free structures. We present annotations for matchings and spanning cycles in this paper, an annotation scheme for spanning trees was designed by Alvarez et al. [10].

5.2 Embedding Crossing-Free Structures into Triangulations

Again assume that we want to count matchings. Property (1) above states that each matching should have a unique separator S. This seems hard to achieve directly, especially since a matching can contain very few edges, leaving much freedom to choose a separator. However, we have seen that unique separators exist for triangulations, specifically sn-paths. Hence, we do not count matchings directly, but we count *matchings embedded in a triangulation*. In order not to over-count matchings, we choose a *unique* triangulation T^M containing the matching M and count all pairs (M, T^M). Given a suitable family $\mathcal{S}$ of separators for the triangulations of P, such as

[III] When we say "with high probability" we mean probability $1 - o(1)$.

sn-paths, we count (M, T^M) and thus M for exactly one $S \in \mathcal{S}$. Specifically, we choose the unique triangulation T^M to be the constrained Delaunay triangulation (CDT) $\triangle^M \supset M$, which we briefly describe next.

Constrained Delaunay Triangulation: The constrained Delaunay triangulation (CDT) $\triangle^S$ of a point set P and a set of (crossing-free straight-line) edges S on P was first introduced by L. P. Chew [16]. 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 by Ø. Hjelle and M. Dæhlen [23].

From now on we will assume that no four points of P are cocircular. We can now go back to our simple algorithm for counting matchings and revise it as follows: After picking a separator S, in each recursive 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 checked locally in each recursive call, i.e., choices in one sub-problem do not depend on choices in others, we are done. Since not every set of separators $\mathcal{S}$ admits such a locality condition, we construct a new family of separators in the next section.

5.3 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 τ_p which emanates from p, avoids other points of 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 that intersects τ_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, see Figure 4. It is easy to see that the triangular path $P_p(T)$ is uniquely defined for any triangulation T. 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 $\triangle_{p_i}(T_1) \neq \triangle_{p_i}(T_2)$, then those two triangles intersect, as they both are adjacent to p, intersect τ_p and have interiors free of points in P. We are now ready to present the algorithm for counting matchings.

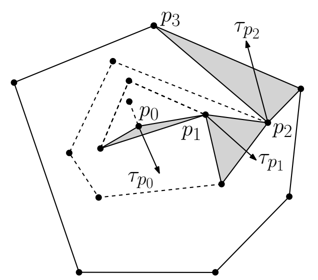

Figure 4: *Triangular path P_p starting in onion layer $P^{(4)}$. Onion layers are drawn dashed. P_p can be extended to a triangulation T, in such a case P_p will be unique for T.*

5.4 Algorithm for counting matchings

Given a matching M, let $\triangle^M$ be the CDT of M (with vertex set P). By our assumption of no four cocircular points, this CDT is unique for M. We annotate $\triangle^M$ as follows:

- each edge e of $\triangle^M$ is annotated with a bit b_e that indicates whether e belongs to M or not.
- each vertex p of $\triangle^M$ is annotated with a number $0 \leq m_p \leq n$ that represents the point in P that p is matched to. If m_p is, say, 0 then we know that p is not matched in M.

We may add the constraint $m_p > 0$ to count only perfect matchings, otherwise we count all (not necessarily perfect) matchings.

Let us denote by $\overline{\triangle}^M$ the annotated version of $\triangle^M$. Let S be a separator contained in $\triangle^M$ that splits CH(P) into regions $R_1^S, \ldots, R_t^S$. Separator S inherits all the information from $\overline{\triangle}^M$. 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 is identical to $\overline{\triangle}^M$, for some matching M. Since there is a one-to-one correspondence between matchings and legal annotated 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. In the first call of the algorithm, we start with an edge ab on 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 a triangular path starting at p forms a separator, see Figure 5. We enumerate such separators and *all possible* annotations for *each one of them*. Each such annotated separator splits CH(P) into two smaller regions, which we solve recursively. In each such recursive sub-problem we count legal annotated constrained Delaunay triangulations consistent with the annotated separator, i.e., for example, if two adjacent vertices of the separator have been annotated, and they agree to be matched to each other and the edge connecting them is annotated to be in the matching, then in future recursive sub-problems other edges adjacent to those two vertices cannot be annotated to be in a matching as well. Clearly, the only sub-problems that will contribute to the final computed number of matchings are the ones for which the algorithm, in its whole run, could complete a full annotated constrained Delaunay triangulation without finding any violation of the annotations inherited by the separators that led to that triangulation.

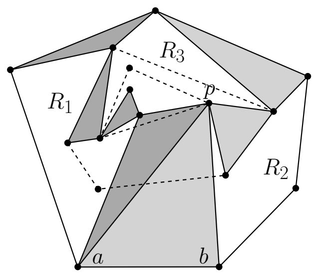

Figure 5: *In the first call of the algorithm, the triangular path shown in light gray is created. It divides the problem into regions $R_1 \cup R_3$ and R_2. A call for the former creates the triangular path shown in dark gray. Annotations are not shown.*

Let us elaborate on why it is necessary to use triangular paths instead of sn-paths. Note that no edge of 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, and independently, in each sub-problem. If an edge belonged to more than one sub-problem (as is the case for sn-paths), 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 algorithm for counting triangulations, we use memoization, so that the running time 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. There are no more than $n^{O(k)}$ possible annotations per triangular path, as can be easily checked. In total, there are $n^{O(k)}$ triples of annotated triangular paths. Thus, the overall running time is $n^{O(k)}$, which even considers the polynomial overhead arising from checking flippability of edges and inclusion of points in triangles. This concludes the first part of Theorem 2.

In contrast to our algorithm for counting triangulations that runs in time $O^*(3.1414^n)$ (see § 4.3), for counting matchings we cannot prove a running time of the form c^n with a reasonably small constant c. The reason is that triangular paths not only consist of descending paths, but also of dangling triangles, whose third point can be on a arbitrary layer. Specifically, if we consider the layers of the third vertices of all dangling triangles, then they are not sorted. This makes it hard to bound the number of triangular paths.

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 in the next section, where we consider the problem of counting spanning cycles.

5.5 Algorithm for counting spanning cycles

Counting spanning cycles is more complicated than counting matchings. We first reduce the problem to counting *rooted and oriented* spanning cycles: Given any spanning 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 spanning 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 corresponds to exactly $2n$ rooted and oriented spanning cycles, so it suffices to count the latter and divide by $2n$. 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 edge e in $\triangle^H$ is annotated with a bit b_e that indicates whether e belongs to H or not.
- each vertex p of $\triangle^H$ is annotated with $(\mathrm{pos}_p, \mathrm{prev}_p, \mathrm{next}_p)$, where pos_p is the number assigned to p in H, prev_p is the point lying immediately before p in H, and next_p is the point lying immediately after p in H.

As in the case of matchings, we denote the annotated $\triangle^H$ by $\overline{\triangle}^H$. A separator S contained in $\triangle^H$ inherits all annotations of the vertices in S. Thus, from the annotated separator $\overline{\triangle}^H_S$ we know for each point p of S its position on the HamCycle as well as its predecessor and successor points. Note that since S contains $O(k)$ vertices, there are again at most $n^{O(k)}$ possible annotations of S.

The algorithm for counting matchings now carries over verbatim, if we appropriately enumerate annotations and check their consistency. The running time is again $n^{O(k)}$. This concludes the proof of Theorem 2.

6 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 a provable worst-case running time of $O^*(3.1414^n)$. Moreover, it runs in polynomial time whenever the number of onion layers of the given set of points is constant. No other algorithm is currently known that runs in polynomial time restricted to any non-trivial set of instances. Finally, recent experiments [8, 11] indicate that our algorithm is highly relevant in practice as well.

Regarding other crossing-free structures, we presented a general framework that allows us to exactly count crossing-free structures that can be unequivocally encoded with an annotation scheme. We showed how to use our framework by giving annotation schemes that encode (perfect) matchings and spanning cycles. Very recently an annotation scheme for spanning trees, which is fully compatible with our framework, was designed by Alvarez at al. [10] (in the context of approximate counting). We obtained algorithms to exactly count these structures in time $n^{O(k)}$, where k is the number of onion layers. This implies polynomial-time algorithms for fixed k. Algorithms with this property were not known before for these problems, and this, in particular, 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 [18].

In presence of very recent developments on counting crossing-free structures [10, 11, 35], the most interesting question at this point is whether exact counting can always be done in sub-exponential time ($2^{o(n)}$), or, even more, whether it can always be done in polynomial time.

Our counting algorithms also allow us to generate crossing-free structures uniformly at random. For example, the problem of generating spanning cycles (uniformly) at random has attracted the attention of researchers for almost 20 years [13], in the form of generating random simple polygons on P. Since our algorithms are based on the divide-and-conquer paradigm, we can easily adapt the method explained by Aichholzer [1] to produce such random structures, after running the counting algorithm.

References

[1] O. Aichholzer. The path of a triangulation. In *Proceedings of the 15th annual Symposium on Computational Geometry*, SoCG '99, pages 14–23. ACM, 1999.

[2] O. Aichholzer, F. Aurenhammer, C. Huemer, and B. Vogtenhuber. Gray code enumeration of plane straight-line graphs. *Graphs and Combinatorics*, 23(5):467–479, 2007.

[3] O. Aichholzer, T. Hackl, C. Huemer, F. Hurtado, H. Krasser, and B. Vogtenhuber. On the number of plane geometric graphs. *Graphs and Combinatorics*, 23(1):67–84, 2007.

[4] O. Aichholzer, F. Hurtado, and M. Noy. A lower bound on the number of triangulations of planar point sets. *Computational Geometry*, 29(2):135 – 145, 2004.

[5] O. Aichholzer and H. Krasser. The point set order type data base: A collection of applications and results. In *Proceedings of the 13th Canadian Conference on Computational Geometry*, CCCG '01, pages 17–20, 2001.

[6] M. Ajtai, V. Chvátal, M. Newborn, and E. Szemerédi. Crossing-free subgraphs. In G. S. Peter L. Hammer, Alexander Rosa and J. Turgeon, editors, *Theory and Practice of Combinatorics A collection of articles honoring*

Anton Kotzig on the occasion of his sixtieth birthday, volume 60 of *North-Holland Mathematics Studies*, pages 9 – 12. North-Holland, 1982.

[7] V. Alvarez, K. Bringmann, R. Curticapean, and S. Ray. Counting crossing-free structures. In *Proceedings of the 28th annual Symposium on Computational Geometry*, SoCG '12, pages 61–68. ACM, 2012.

[8] V. Alvarez, K. Bringmann, R. Curticapean, and S. Ray. Counting triangulations and other crossing-free structures via onion layers. *Computing Research Repository (CoRR)*, 2013. Available at `http://arxiv.org/abs/1312.4628`.

[9] V. Alvarez, K. Bringmann, and S. Ray. A simple sweep line algorithm for counting triangulations and pseudo-triangulations. *Computing Research Repository (CoRR)*, 2013. Available at `http://arxiv.org/abs/1312.3188`.

[10] V. Alvarez, K. Bringmann, S. Ray, and R. Seidel. Counting triangulations and other crossing-free structures approximately. *Computational Geometry*, To appear, 2015. Available at `http://dx.doi.org/10.1016/j.comgeo.2014.12.006`.

[11] V. Alvarez and R. Seidel. A simple aggregative algorithm for counting triangulations of planar point sets and related problems. In *Proceedings of the 29th Annual Symposium on Computational Geometry*, SoCG '13, pages 1–8. ACM, 2013.

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

[13] T. Auer and M. Held. Heuristics for the generation of random polygons. In *Proceedings of the 8th Canadian Conference on Computational Geometry*, CCCG '96, pages 38–43, 1996.

[14] D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1–3):21–46, 1996.

[15] S. Bespamyatnikh. An efficient algorithm for enumeration of triangulations. *Computational Geometry*, 23(3):271–279, 2002.

[16] L. P. Chew. Constrained Delaunay triangulations. *Algorithmica*, 4(1-4):97–108, 1989.

[17] K. Dalal. Counting the onion. *Random Structures & Algorithms*, 24(2):155–165, 2004.

[18] E. Demaine, J. S. B. Mitchell, and J. O'Rourke. Problem 16: Simple polygonalizations, May 2014. `http://cs.smith.edu/~orourke/TOPP/P16.html#Problem.16`. Accessed on 12.05.2014.

[19] A. Dumitrescu, A. Schulz, A. Sheffer, and C. D. Tóth. Bounds on the maximum multiplicity of some common geometric graphs. *SIAM Journal on Discrete Mathematics*, 27(2):802–826, 2013.

[20] P. Flajolet and M. Noy. Analytic combinatorics of non-crossing configurations. *Discrete Mathematics*, 204(1–3):203 – 229, 1999. Selected papers in honor of Henry W. Gould.

[21] J. Flum and M. Grohe. *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series)*. Springer, 2006.

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

[23] Ø. Hjelle and M. Dæhlen. *Triangulations and Applications (Mathematics and Visualization)*. Springer, 2006.

[24] M. Hoffmann, A. Schulz, M. Sharir, A. Sheffer, C. Tóth, and E. Welzl. Counting plane graphs: Flippability and its applications. In J. Pach, editor, *Thirty Essays on Geometric Graph Theory*, pages 303–325. Springer, 2013.

[25] C. Huemer and A. de Mier. Lower bounds on the maximum number of non-crossing acyclic graphs. *arXiv preprint 1310.5882*, 2013. Available at `http://arxiv.org/abs/1310.5882`.

[26] N. Katoh and S. Tanigawa. Fast enumeration algorithms for non-crossing geometric graphs. *Discrete & Computational Geometry*, 42(3):443–468, 2009.

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

[28] A. Razen and E. Welzl. Counting plane graphs with exponential speed-up. In C. Calude, G. Rozenberg, and A. Salomaa, editors, *Rainbow of Computer Science*, volume 6570, pages 36–46. Springer, 2011.

[29] F. Santos and R. Seidel. A better upper bound on the number of triangulations of a planar point set. *Journal of Combinatorial Theory, Series A*, 102(1):186–193, 2003.

[30] M. Sharir and A. Sheffer. Counting triangulations of planar point sets. *Electr. J. Comb*, 18(1):P70, 2011.

[31] M. Sharir, A. Sheffer, and E. Welzl. On degrees in random triangulations of point sets. *Journal of Combinatorial Theory, Series A*, 118(7):1979–1999, 2011.

[32] M. Sharir, A. Sheffer, and E. Welzl. Counting plane graphs: Perfect matchings, spanning cycles, and Kasteleyn's technique. *Journal of Combinatorial Theory, Series A*, 120(4):777–794, 2013.

[33] M. Sharir and E. Welzl. On the number of crossing-free matchings, cycles, and partitions. *SIAM Journal on Computing*, 36(3):695–720, 2006.

[34] A. Sheffer. Numbers of plane graphs, May 2014. `http://www.cs.tau.ac.il/~sheffera/counting/PlaneGraphs.html`. Accessed on 12.05.2014.

[35] M. Wettstein. Counting and enumerating crossing-free geometric graphs. In *Proceedings of the 30th annual Symposium on Computational Geometry*, SoCG '14, pages 1–10. ACM, 2014.

[36] C. Zhu, G. Sundaram, J. Snoeyink, and J. S. Mitchell. Generating random polygons with given vertices. *Computational Geometry*, 6(5):277 – 290, 1996.