# Counting Triangulations and other Crossing-Free Structures Approximately

Victor Alvarez* Karl Bringmann† Saurabh Ray‡ Raimund Seidel§

October 25, 2013

**Abstract**

We consider the problem of counting straight-edge triangulations of a given set $P$ of $n$ points in the plane. Until very recently it was not known whether the *exact* number of triangulations of $P$ can be computed asymptotically faster than by enumerating all triangulations. We now know that the number of triangulations of $P$ can be computed in $O^*(2^n)$ time [9], which is less than the lower bound of $\Omega(2.43^n)$ on the number of triangulations of *any* point set [29]. In this paper we address the question of whether one can approximately count triangulations in sub-exponential time. We present an algorithm with sub-exponential running time and sub-exponential approximation ratio, that is, denoting by $\Lambda$ the output of our algorithm, and by $c^n$ the exact number of triangulations of $P$, for some positive constant $c$, we prove that $c^n \leq \Lambda \leq c^n \cdot 2^{o(n)}$. This is the first algorithm that in sub-exponential time computes a $(1 + o(1))$-approximation of the base of the number of triangulations, more precisely, $c \leq \Lambda^{\frac{1}{n}} \leq (1 + o(1))c$. Our algorithm can be adapted to approximately count other crossing-free structures on $P$, keeping the quality of approximation and running time intact. In this paper we show how to do this for matchings and spanning trees.

## 1 Introduction

Let $P$ be a set of $n$ points on the plane. A crossing-free structure on $P$ is a straight-line plane graph with vertex set $P$. Examples of crossing-free structures include triangulations, trees, matchings, and spanning cycles, also known as the polygonizations of $P$, among others. Let $X$ denote a certain type of crossing-free structures and let $\mathcal{F}_X(P)$

*Information Systems Group, Universität des Saarlandes, Germany, alvarez@cs.uni-saarland.de.

†Max Planck Institute for Informatics, Karl Bringmann is a recipient of the *Google Europe Fellowship in Randomized Algorithms*, and this research is supported in part by this Google Fellowship. kbringma@mpi-inf.mpg.de.

‡Ben Gurion University of the Negev, Israel saurabh@math.bgu.ac.il.

§Fachrichtung Informatik, Universität des Saarlandes, Germany, rseidel@cs.uni-saarland.de.

denote the set of all crossing-free structures on $P$ of type $X$. One of the most intriguing problems in Computational Geometry is the following: given $P$ and a particular type of crossing-free structures $X$, how fast can we compute the cardinality of $\mathcal{F}_X(P)$?

Among all kinds of crossing-free structures on $P$, triangulations are perhaps the most studied ones, so let us first focus on them. How fast can we compute the number of triangulations of $P$? For starters, although triangulations are the most studied, the problem of computing the number of *all* triangulations of $P$ is, in general, neither known to be #P-hard, nor do we know of a polynomial time algorithm to even approximate that number. When the point set is in convex position, an easy recurrence relation can be derived showing that the number of triangulations spanned by $n$ points in convex position is $C_{n-2}$, where $C_k$ is the $k$-th catalan number. F. Hurtado and M. Noy [20] were able to find further formulas to exactly compute the number of triangulations of "almost convex sets". Unfortunately, the approach of finding exact formulas does not seem to take us much further.

For *any* given set of points $P$ it is possible to enumerate all its triangulations in time proportional to its number of triangulations. This is because the *flip graph*[I] of triangulations is connected, see [22, 32]. Thus any graph traversal algorithm like DFS or BFS can be used to enumerate the vertices of the flip graph of triangulations of $P$. One limitation of such traversal algorithms, however, is that the amount of memory used is proportional to the number of vertices in the graph - which is known to *always* be exponential, because the number of triangulations of $P$ lies between $\Omega(2.43^n)$ [29] and $O(30^n)$ [28]. Using a general technique due to D. Avis and K. Fukuda called Reverse Search, see [10], it is possible to enumerate triangulations while keeping the memory usage polynomial in $n$. This technique has been further improved in [11, 21]. However, it is important to observe that, since the number of triangulations is exponential in $n$, counting triangulations by enumeration takes exponential time, so a natural question is whether one can do significantly better.

In [1] O. Aichholzer introduced the notion of the "path of a triangulation", T-path from now on, that allowed a divide-and-conquer approach to speed up counting. From empirical observations, this approach seems to count triangulations in time sub-linear in the number of triangulations, that is, apparently faster than enumeration. Unfortunately, no proof that this algorithm is *always* faster than enumeration - or even a good analysis of its running time - has been found. Subsequently, a simple algorithm based on dynamic programming was presented in [25]. This algorithm empirically appears to be substantially faster than Aichholzer's algorithm. From the limited empirical data, the running time seems to be proportional to the square root of the number of triangulations. However, as with Aichholzer's algorithm, the worst case running time of this algorithms seems difficult to analyze and no bound on the running time has been found. More recently, two algorithms whose running times could actually be analyzed were presented. The first algorithm, presented in [8], combines Aichholzer's idea of T-paths with

[I]The flip graph is a graph whose vertex set is the set of triangulations of $P$ and where there is an edge between vertices of the flip graph if both corresponding triangulations can be transformed into each other by flipping exactly one of their edges.

a sweep-line algorithm and runs in time proportional to the largest number of T-paths found during execution (within a $poly(n)$ factor). In [8] the number of T-paths is shown to be at most $O(9^n)$. It is important to observe that, for very particular and well-studied configurations of points, the number of T-paths can be shown to be significantly smaller than the number of triangulations. Thus, at least for those configurations, the algorithm of [8] counts triangulations faster than by using enumeration techniques. Unfortunately, this algorithm turned out to be very slow in practice, which is most probably due to the fact that the number of T-paths can still be very large, in [15] a configuration having at least $\Omega(4^n)$ T-paths was shown. The second algorithm, presented in [6], uses a divide-and-conquer approach based on the onion layers of the given set of points. This algorithm was shown to have a running time of at most $O^*(3.1414^n)$ and is likely to have a running time sub-linear is the number of triangulations since it is widely believed that the number of triangulations spanned by *any* set of $n$ points is at least $\sqrt{12}^n \approxeq 3.46^n$, see [27, 4, 3]. From the experimental point of view, the second algorithm turned out to be significantly faster, for certain configurations of points, than the one shown in [25]. These experiments can be found in [7]. Finally, in 2013, an algorithm with running time $O^*(2^n)$ was presented [9]. This last algorithm finally shows that enumeration algorithms for triangulations can indeed *always* be beaten, as all point sets with $n$ points have at least $\Omega(2.43^n)$ triangulations. From an experimental point of view it was also shown to be significantly faster than all previous algorithms on a variety of inputs [9].

With respect to crossing-free structures other than triangulations the situation is very similar. In [21] a general framework for enumerating crossing-free structures (including spanning trees and perfect matchings) was presented. However, as for triangulations, the number of enumerated objects is again in general exponential. For example, the number of spanning trees is between $\Theta^*(6.75^n)$ [16] and $O(141.07^n)$ [19], and the number of perfect matching is between $\Theta^*(2^n)$ [17] and $O(10.05^n)$ [30]. The interested reader is referred to [2, 13, 31] for up-to-date lists of bounds for other crossing-free structures. Thus, again the question arises whether we can count these structures faster than by enumeration. In this respect, in [8] an algorithm was shown that counts pseudo-triangulations in time proportional to the largest number of pseudo-triangulation paths [5], that the algorithm encounters. It is however still not known whether there are always considerably less pseudo-triangulation paths than pseudo-triangulations. In [26] an algorithm was shown that counts *all* crossing-free structures of a given set of points $P$ in time sub-linear in the number of counted objects, thus achieving the desired exponential speed-up in this case. Finally, it was until very recently that new fast counting algorithms appeared. In [33] it was shown, for example, that the number of *all* crossing-free structures, perfect matchings, and convex partitions can be computed in time $O^*(2.839^n)$ for the former and $O^*(2^n)$ for the latter two. These algorithms generalize the idea presented in [9] and show that enumeration can, at least in these cases, *always* be beaten. It is, however, still open whether for spanning trees and spanning cycles, two of the most popular classes of crossing-free structures, the same can be said, see [33] for partial results in this direction.

### 1.1 Our contribution

The $O^*(2^n)$ algorithm of [9] for counting triangulations *exactly* seems hard to beat at this point. Instead, we relax the goal to computing an *approximation* of the number of triangulations and pose the question of whether one can reduce the runtime in this setting. The answer presented in this paper is, to the best of our knowledge, the first result in this new line of research.

Note that, since for all sets of $n$ points the number of triangulations is $\Omega(2.43^n)$ [29] and $O(30^n)$ [28], the quantity $\Theta\left(\sqrt{30 \times 2.43}^{\,n}\right)$ approximates the exact number of triangulations within a factor of $O\left(\sqrt{30/2.43}^{\,n}\right)$. Thus, one can trade the exponential time of an exact algorithm for a polynomial time algorithm with exponential approximation ratio. In this paper we bridge the gap between these two solutions by presenting an algorithm with sub-exponential running time and sub-exponential approximation ratio.

Let $\mathcal{F}_X(P)$ denote the set of all crossing-free structures of type $X$ on $P$, where $X$ could mean triangulations, matchings, or spanning trees. The main result of this paper is the following:

**Theorem 1.** *Let $P$ be a set of $n$ points on the plane. Then a number $\Lambda$ can be computed in time $2^{o(n)}$ such that $|\mathcal{F}_X(P)| \leq \Lambda \leq |\mathcal{F}_X(P)|^{1+o(1)} = 2^{o(n)}|\mathcal{F}_X(P)|$.*

The precise $o(n)$ terms mentioned in Theorem 1 are $O\left(\sqrt{n}\log(n)\right)$ for the running time and $O\left(n^{\frac{3}{4}}\sqrt{\log(n)}\right)$ for the approximation factor. At the end of § 4 we mention a trade-off between these two, running time and approximation factor.

While the approximation factor of $\Lambda$ is rather big, the computed value is of the same order of magnitude as the correct value of $|\mathcal{F}_X(P)|$, that is, we compute a $(1 + o(1))$-approximation of the base of the exponentially large value $|\mathcal{F}_X(P)|$. More precisely, if we denote $|\mathcal{F}_X(P)|$ by $c^n$, for some positive constant $c$ that depends on $P$ and $X$, then we have $c \leq \Lambda^{\frac{1}{n}} \leq c^{1+o(1)} \leq (1 + o(1))c$. Also, this approximation can be computed in sub-exponential time, which, at least theoretically, is asymptotically faster than the worst-case running times of the algorithms presented in [8, 6, 9, 33]. This is certainly very appealing.

The rest of the paper is divided as follows: We start in § 2 with basic preliminaries. For simplicity, we first show in § 3 the algorithm for counting triangulations approximately and in § 4 the corresponding proof of Theorem 1. We generalize the algorithm for counting matchings and spanning trees in § 5. We close the paper in § 6 with some remarks and conclusions.

## 2 Preliminaries

Our algorithm uses simple cycle separators as the main ingredient, originally presented in [24] by G. L. Miller, and improved in [14] by H. N. Djidjev and S. M. Venkatesan. The following theorem accounts for both results:

**Theorem 2** (Separator Theorem)**.** *Let $T$ be a triangulation of a set of $n$ points in the plane such that the unbounded face is a triangle. Then there exists a simple cycle $C$ of size at most $\sqrt{4n}$, that separates the set $A$ of vertices of $T$ in its interior from the set $B$ of vertices of $T$ in its exterior, such that the number of elements of each one of $A$ and $B$ is always at most $\frac{2n}{3}$.*

Observe that the Separator Theorem does not imply that *every* triangulation of a set of points contains a *unique* simple cycle separator. One can easily come up with examples in which a triangulation contains more than one simple cycle separator. The important part here is that *every* triangulation contains *at least* one simple cycle separator.

# 3 Counting triangulations approximately

The idea for an approximate counting algorithm is suggested by the Separator Theorem: We enumerate all possible simple cycle separators $C$ of size at most $\sqrt{4n}$ that we can find in the given set $P$. We then recursively compute the number of triangulations of each of the parts $A$ and $B$, specified by the Separator Theorem, that are delimited by $C$[II]. We then multiply the number we obtain for the sub-problem $A \cup C$ by the number we obtain for sub-problem $B \cup C$, and we add these products over all cycle separators $C$. With this algorithm we clearly over-count the triangulations of $P$, and it remains to show that we do not over-count by too much. We will later see that in order to keep over-counting small, we have to solve small recursive sub-problems exactly. Note that problems of size smaller than a threshold $\Delta$ can be solved exactly in time $O^*\left(2^{\Delta}\right) = 2^{O(\Delta)}$, see [9].

However, there are some technicalities that we have to overcome first. For starters, the Separator Theorem holds only if the unbounded face of $T$ is also a triangle. Thus, if we add a dummy vertex $v_\infty$ outside $\mathrm{Conv}(P)$, along with the adjacencies between $v_\infty$ and the vertices of $\mathrm{Conv}(P)$, to make the unbounded face a triangle, we can apply the Separator Theorem. Once a simple cycle with the dividing properties of a separator is found, by the deletion of $v_\infty$ we are left with a separator that is either the original cycle that we found, if $v_\infty$ does not appear as a vertex of the separator, or a path otherwise. Thus, when guessing a separator we have to consider that it might be a path instead of a cycle. This brings us to the second technical issue. As we go deeper in the recursion we might create "holes" in $P$ whose boundaries are the separators that we have considered thus far. That is, the recursive problems are polygonal regions, possibly with holes, containing points of $P$. Therefore, when guessing a separator, cycle or path, we have to arbitrarily triangulate the holes first. This does not modify the size of the sets we guess for a separator in a sub-problem.

We can now prove the first lemma:

**Lemma 1.** *Let $\mathcal{F}_T(P)$ be the set of triangulations of a set $P$ of $n$ points. Then* all *separators, simple cycles or paths, among* all *the elements of $\mathcal{F}_T(P)$ can be enumerated in time $2^{O\left(\sqrt{n}\log(n)\right)}$.*

[II]Thus separator $C$ also forms part of the two sub-problems.

*Proof.* We know by the Separator Theorem and the discussion beneath that *every* element of $\mathcal{F}_T(P)$, a triangulation, contains at least one separator $C$, simple cycle or path. Moreover, the size of $C$ is at most $\sqrt{4n}$. Thus, searching by brute-force will do the job. We can enumerate all the sub-sets of $P$ of size at most $\sqrt{4n}$ along with their permutations. A permutation tells us how to connect the points of the guessed sub-set, after also guessing whether we have a path or cycle. We can then verify if the constructed simple cycle, or path, fulfils the dividing properties of a separator, as specified in the Separator Theorem.

It is not hard to check that the total number of guessed subsets and their permutations is $2^{O\left(\sqrt{n}\log(n)\right)}$. Verifying whether a cycle, or a path, is indeed a separator can be done in polynomial time. Thus, the total time spent remains being $2^{O\left(\sqrt{n}\log(n)\right)}$. ■

We can now proceed with the corresponding proof of Theorem 1.

# 4 Proof of Theorem 1

We first prove that the approximation ratio of the algorithm for counting triangulations is sub-exponential and then we prove that its running time is sub-exponential as well.

## 4.1 Quality of approximation

By the proof of Lemma 1 we also obtain that the number of simple cycle separators cannot be larger than $2^{O\left(\sqrt{n}\log(n)\right)}$. Since at *every* stage of the recursion of the counting algorithm *no* triangulation of $P$ can contain more than the total number of simple cycle separators found at that stage, we can express the *over-counting factor* of the algorithm by the following recurrence:

$$S(P,\Delta) \leq \sum_{C} S(A\cup C,\Delta)\cdot S(B\cup C,\Delta) \leq 2^{O\left(\sqrt{n}\log(n)\right)}\cdot S(A\cup C^*,\Delta)\cdot S(B\cup C^*,\Delta)$$

where the summation is over all separators $C$ available at the level of recursion. $A\cup C$, $B\cup C$ are the sub-problems as explained before, $C^*$ is the cycle that maximizes the term $S(A\cup C,\Delta)\cdot S(B\cup C,\Delta)$ over all $C$, and $\Delta$ is a stopping size. Specifically, whenever the current recursive sub-problem contains $\leq \Delta$ points we stop the recursion and compute the number of triangulations of the sub-problem exactly. Hence, we have $S(P',\Delta)=1$ whenever $|P'|\leq\Delta$. We can now write:

$$S'(P,\Delta) := \log(S(P,\Delta)) \leq O\left(\sqrt{n}\log(n)\right) + S'(A\cup C^*,\Delta) + S'(B\cup C^*,\Delta).$$

Our goal now is to prove the following lemma:

**Lemma 2.** *Let $P$ be a set of $n$ points on the plane and assume $\Delta = n^{\Omega(1)}$, $n > \Delta$, and $\Delta$ is at least a sufficiently large constant. Then we have*

$$S'(P,\Delta) = O\left(\left(\frac{n}{\sqrt{\Delta/3}} - \sqrt{n}\right)\log\Delta\right).$$

*Proof.* We use induction over the size of $P$. Let $P' \subseteq P$ of size $m \leq n$. We have,

$$\begin{aligned} S'(P',\Delta) &\leq O\left(\sqrt{m}\log(m)\right) + S'(A\cup C^*,\Delta) \ + S'(B\cup C^*,\Delta) \\ &\leq O\left(\sqrt{m}\log(m)\right) + c\left(\frac{m_1}{\sqrt{\frac{\Delta}{3}}} - \sqrt{m_1} + \frac{m_2}{\sqrt{\frac{\Delta}{3}}} - \sqrt{m_2}\right)\log\Delta, \end{aligned} \tag{1}$$

where $m_1, m_2$ are the sizes of the sub-problems $A\cup C^*$ and $B\cup C^*$ of $P'$, respectively, and $c$ is some sufficiently large positive constant. By the Separator Theorem, we can express $m_1 \leq \alpha m + \sqrt{4m}$ and $m_2 \leq \beta m + \sqrt{4m}$, such that: (1) $\alpha, \beta$ are constants that depend on the instance, so $\alpha = \alpha\left(A\cup C^*\right)$ and $\beta = \beta\left(B\cup C^*\right)$, (2) $0 < \beta \leq \alpha \leq \frac{2}{3}$, and (3) $\alpha + \beta = 1$.

Now let us for the moment focus on the term $\frac{m_1}{\sqrt{\Delta/3}} - \sqrt{m_1} + \frac{m_2}{\sqrt{\Delta/3}} - \sqrt{m_2}$ of equation (1) above:

$$\begin{aligned} \frac{m_1}{\sqrt{\frac{\Delta}{3}}} - \sqrt{m_1} + \frac{m_2}{\sqrt{\frac{\Delta}{3}}} - \sqrt{m_2} &= \frac{m_1+m_2}{\sqrt{\frac{\Delta}{3}}} - \sqrt{m_1} - \sqrt{m_2} \\ &\leq \frac{\alpha m + \sqrt{4m} + \beta m + \sqrt{4m}}{\sqrt{\frac{\Delta}{3}}} - \sqrt{m_1} - \sqrt{m_2} \\ &\leq \frac{m + 4\sqrt{m}}{\sqrt{\frac{\Delta}{3}}} - \sqrt{\alpha m} - \sqrt{\beta m} \\ &= \frac{m + 4\sqrt{m}}{\sqrt{\frac{\Delta}{3}}} - \sqrt{m}\left(\sqrt{\alpha} + \sqrt{\beta}\right) \\ &\leq \frac{m + 4\sqrt{m}}{\sqrt{\frac{\Delta}{3}}} - \sqrt{m}\,(1+\varepsilon) \end{aligned}$$

The last inequality is obtained by minimizing $\sqrt{\alpha} + \sqrt{\beta}$. Since we mentioned before that $0 \leq \beta \leq \alpha \leq \frac{2}{3}$ and $\alpha + \beta = 1$, the minimum of $\sqrt{\alpha} + \sqrt{\beta}$ is attained at $(\alpha, \beta) = \left(\frac{2}{3}, \frac{1}{3}\right)$, and is strictly larger than one, so we can choose $\varepsilon > 0$. Now, since $\Delta$ is sufficiently large, we have $\frac{4\sqrt{m}}{\sqrt{\Delta/3}} \ll \varepsilon\sqrt{m}$, so $\frac{4\sqrt{m}}{\sqrt{\Delta/3}} - \varepsilon\sqrt{m} \leq -\varepsilon'\sqrt{m}$, for some positive constant $\varepsilon'$. Thus we can continue as follows:

$$\frac{m_1}{\sqrt{\frac{\Delta}{3}}} - \sqrt{m_1} + \frac{m_2}{\sqrt{\frac{\Delta}{3}}} - \sqrt{m_2} \leq \frac{m + 4\sqrt{m}}{\sqrt{\frac{\Delta}{3}}} - \sqrt{m}\,(1+\varepsilon) \leq \frac{m}{\sqrt{\frac{\Delta}{3}}} - (1+\varepsilon')\sqrt{m}. \tag{2}$$

Combining equations (1) and (2) we obtain

$$S'(P',\Delta) \leq O\left(\sqrt{m}\log(m)\right) + c\left(\frac{m}{\sqrt{\frac{\Delta}{3}}} - (1+\varepsilon')\sqrt{m}\right)\log\Delta$$
$$\leq c\left(\frac{m}{\sqrt{\frac{\Delta}{3}}} - \sqrt{m}\right)\log\Delta + O\left(\sqrt{m}\log(m)\right) - c\cdot\varepsilon'\sqrt{m}\log\Delta$$

If we choose $\Delta$ to be sufficiently large, say $\Delta \geq n^{\delta}$, for some constant $\delta > 0$, then we have $\Delta \geq n^{\delta} \geq m^{\delta}$, and the negative term $-c\cdot\varepsilon'\sqrt{m}\log\Delta$ is larger, for appropriately large $c$, than the $O\left(\sqrt{m}\log(m)\right)$ term. Hence, we can conclude that $S'(P',\Delta) \leq O\left(\left(\frac{m}{\sqrt{\Delta/3}} - \sqrt{m}\right)\log\Delta\right)$, which proves the induction step.

It still remains to prove that the inductive claim holds for the boundary condition, so let $Q$ be a recursive sub-problem of size $\leq \Delta$. As $Q$ stems from an application of the Separator Theorem, it is easy to see that $|Q| \geq \frac{\Delta}{3}$. Thus, we have $S'(Q,\Delta) = 0 \leq c\left(\frac{|Q|}{\sqrt{\Delta/3}} - \sqrt{|Q|}\right)\log\Delta$. Lemma 2 follows. ■

Now, let $\Lambda$ be the number computed by our algorithm. Recall that $|\mathcal{F}_T(P)|$ is the exact number of triangulations of $P$. By setting $\Delta = \sqrt{n}\log(n)$ we obtain an over-counting factor of the algorithm of:

$$S(P,\Delta) = 2^{S'(P,\Delta)} = 2^{O\left(\frac{n\log\Delta}{\sqrt{\Delta}}\right)} = 2^{O\left(n^{\frac{3}{4}}\sqrt{\log(n)}\right)}$$

Hence $|\mathcal{F}_T(P)| \leq \Lambda \leq |\mathcal{F}_T(P)|\cdot 2^{O\left(n^{\frac{3}{4}}\sqrt{\log(n)}\right)} = |\mathcal{F}_T(P)|^{1+o(1)}$. This completes the qualitative part of Theorem 1. It remains to discuss the running time of the algorithm.

### 4.2 Running time

The running time of the algorithm can be expressed with the following recurrence:

$$T(n) < 2^{O\left(\sqrt{n}\log(n)\right)}\cdot T\left(\frac{2n}{3} + \sqrt{4n}\right).$$

Taking again $T'(n) = \log(T(n))$ yields $T'(n) := T'\left(\frac{2n}{3} + \sqrt{4n}\right) + O\left(\sqrt{n}\log(n)\right)$, which can then be solved using the well-known Akra-Bazzi Theorem for recurrences, see [23]. This yields $T'(n) = O\left(\sqrt{n}\log(n)\right)$. There is, however, one detail missing, the stopping condition $\Delta$. In order to use the Akra-Bazzi Theorem we need a boundary condition of $T(n) = 1$ for $1 \leq n \leq n_0$ (for some constant $n_0$), but in the algorithm we stop the recursion whenever a sub-problem $Q$ is of size $\leq \Delta$ (which is dependent on the size $n$ of the original point set). At that point we compute the exact number of

triangulations of $Q$, which gives $T(|Q|) = 2^{O(|Q|)} = 2^{O(\Delta)}$. Hence the exponent in the running time of the algorithm is upper-bounded by the solution of $T'(n)$, as given by the Akra-Bazzi Theorem, plus $O(\Delta)$, *i.e.*, $T(n) = 2^{O\left(\sqrt{n}\log(n)+\Delta\right)}$. If as before we assume that $\Delta = \sqrt{n}\log(n)$ then we end up with $T(n) = 2^{O\left(\sqrt{n}\log(n)\right)} = 2^{o(n)}$, which concludes the proof of Theorem 1 for triangulations.

As a final remark observe that we could have used other values for $\Delta$, rather than $\sqrt{n}\log(n)$, without violating any argument in the proofs. This yields a tradeoff with running time $2^{O(\Delta)}$ and approximation ratio $2^{O\left(\frac{n\log\Delta}{\sqrt{\Delta}}\right)}$ for any $\sqrt{n}\log(n) \leq \Delta \leq n$. Although the quality of the approximation improves with larger $\Delta$, the running time increases. Since we see no way of not having over-counting with this algorithm, we regard $\Delta = \sqrt{n}\log(n)$ as the most reasonable setting.

## 5 Extension to other crossing-free structures

In this section we show how to count crossing-free structures other than triangulations. The idea is to use the framework first developed in [6] that allows to exactly count crossing-free structures using the constrained Delaunay triangulation (CDT) $\triangle^S$ constrained to contain the crossing-free structure $S$. For completeness we will briefly describe the constrained Delaunay triangulation as explained in [6, 7].

> **The constrained Delaunay triangulation** (CDT) $\triangle^S$ of $P$ was first introduced in [12]. 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 [18] by Ø. Hjelle and M. Dæhlen.

Without loss of generality we will assume that no four points of $P$ are cocircular. This can always be achieved by perturbing $P$ while keeping the order-type of $P$.

Using the constrained Delaunay triangulation, algorithms to exactly count crossing-free matchings and spanning cycles (polygonizations) of $P$ were shown in [6, 7]. The main idea is the following: Assume we want to count all elements $S \in \mathcal{F}_{\mathcal{C}}(P)$ of a certain class $\mathcal{C}$ of crossing-free structures on $P$. Now, instead of counting all $S$ directly we count the pairs $(\triangle^S, S)$ for all $S \in \mathcal{F}_{\mathcal{C}}(P)$, *i.e.*, we count every element $S \in \mathcal{F}_{\mathcal{C}}(P)$ embedded in its CDT $\triangle^S$. This yields the correct number, since $\triangle^S$ is unique w.r.t. $S$ by our non-degeneracy assumption. As we can see the embedding of $S$ in $\triangle^S$ as annotating

every edge of the triangulation $\triangle^S$ by a bit specifying whether that edge belongs to $S$, we end up counting annotated triangulations. For this we can use similar ideas as in the case of counting triangulations: Having a set of separators $\mathcal{S}$ for triangulations we enumerate all separators $C \in \mathcal{S}$ and use the divide-and-conquer approach to count the crossing-free structures $S$ such that $C \subseteq \triangle^S$. For example, $\mathcal{S}$ could be simple cycle separators, presented in § 2. It is important to observe that when recursing in the smaller sub-problems given by $C$, we have to have a way to locally verify whether $C \subseteq \triangle^S$, *i.e.*, we have to make sure that choices in a sub-problem do not depend on choices in other sub-problems, as otherwise we might get an overcounting by much more than the number of separators. Recall that we want to keep overcounting under control while achieving sub-exponential runtime.

In order to make the previous general idea clear we will adapt the way matchings are counted in [6, 7]. The main difference in the algorithms is the choice of separators. While the separators used in [6, 7] allow to count exactly, the simple cycle separators used in this paper are useful to count approximately, since $\triangle^S$ might have more than one simple cycle separator.

### 5.1 Counting matchings approximately

Let $M$ be a matching of $P$, not necessarily perfect. Let $\triangle^M$ be the CDT of $M$. As stated before, we are working under the assumption that no four points of $P$ are cocircular, thus $\triangle^M$ is unique w.r.t. $M$. Now, annotate $\triangle^M$ as follows:

- Each vertex $v$ of $\triangle^M$ is annotated with a number $m_v$ that indicates the vertex of $M$ that $v$ is matched to. If $v$ is unmatched in $M$ we set $m_v$ to, say, 0.
- 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 CDT $\triangle^M$ annotated according to the previous rules. Let $C \in \mathcal{S}$ be a separator contained in $\triangle^M$ that splits $\mathrm{Conv}(P)$ into regions $R_1, \ldots, R_t$. Separator $C$ inherits all the information, *i.e.*, annotations, from $\overline{\triangle}^M$. The separator thus annotated will be denoted by $C_{\overline{\triangle}^M}$.

#### 5.1.1 The algorithm

An annotated triangulation is said to be *legal* if and only if it is identical to $\overline{\triangle}^M$, for some matching $M$ of $P$. Now observe that there is a one-to-one correspondence between matchings of $P$ and legal annotated triangulations. Thus, instead of counting matchings directly we may count legal annotated triangulations. To do so we proceed essentially as in the algorithm presented in § 3: (1) We enumerate *all* separators in $\mathcal{S}$. (2) We enumerate *all* annotations for each separator $C \in \mathcal{S}$. Each such annotated separator splits $\mathrm{Conv}(P)$ into smaller regions we recur in. In each recursive sub-problem we count

legal annotated triangulations that are 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 sub-problems other edges adjacent to those two vertices cannot be annotated to be in a matching as well. (3) We stop the recursion whenever we find a sub-problem of size at most $\Delta$, just as in the algorithm shown in § 3, and compute the *exact* number of legal annotated triangulations that are consistent with the annotations of the boundary of the sub-problem.

It should be clear by now that 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. There are however two details we have not yet taken care of: (1) How do we verify that *each* edge that is not annotated to be in a matching $M$ is not flippable in $\overline{\triangle}^M$? As it stands right now, for an annotated separator $C$ dividing $\mathrm{Conv}(P)$ into regions $R_1, \ldots, R_t$, we recur into each $R_i$, $1 \leq i \leq t$ independently, although an edge $e$ of $C$ could become flippable depending the two triangles containing $e$ that lie in different regions. (2) How do we *exactly* count legal annotated triangulations that are consistent with the annotations of the boundary of a sub-problem of size at most $\Delta$? We will tackle each difficulty in turn.

### 5.1.2 Triangular cycle separators

To overcome difficulty (1) mentioned before, we make use of what we can call "fat" separators, which are very similar to a construction in [6, 7]. Let $T$ be a triangulation of point set $P$ and let $C \subset T$ be a cycle separator. Every edge of $C$ that belongs to the interior of $\mathrm{Conv}(P)$ is contained in two triangles in $T$. For every such edge choose those two triangles, and for every edge of $C$ on the boundary of $\mathrm{Conv}(P)$ choose the unique triangle that the edge is part of. We make $C$ fat by considering the union of all those chosen triangles (as a set of edges), see Figure 1. We call the resulting fat version of the cycle separator $C$ a *triangular cycle separator* and denote it by $C^\circ$.

Fat separators allow us to overcome the difficulty regarding edge-flippability as follows. Instead of using simple cycle separators, as we did in § 3, we use triangular cycle separators for counting matchings. Since for every edge $e$ contained in a fat separator $C^\circ$ at least one of its incident triangles is fully contained in $C^\circ$, we can check flippability of $e$ in one of the sub-problems, namely the one containing the other triangle incident to $e$. Thus, flippability can be checked independently in each sub-problem.

### 5.1.3 Counting legal annotated triangulations exactly

Assume that at a certain recursion level we have finally reached a sub-problem $Q$ of size at most $\Delta$. The boundary of this sub-problem has been annotated and now we have to

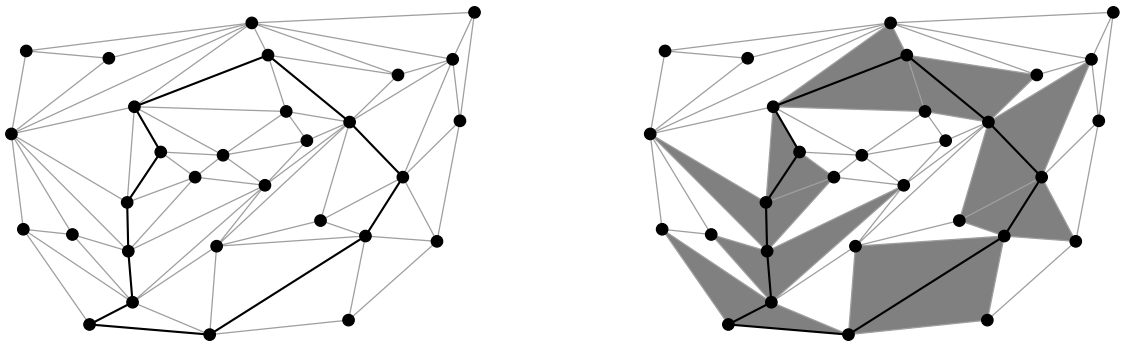

Figure 1: To the left a triangulation and a simple cycle, shown in gray and black respectively. To the right the cycle is made "fat" by attaching the shown triangles to it.

exactly count *all* legal annotated triangulations that are consistent with these annotations. In order to achieve this we make use of enumeration algorithms for triangulations, see [10, 11, 21]. For each enumerated triangulation $T = T(Q)$ we consider *each* sub-set $E' \subseteq E = E(T)$ of its set of edges $E$. For each such $E'$ we check whether $E'$ forms a valid matching, is consistent with the annotations of the boundary of $Q$, and violates no flippability condition. We count all sets $E'$ satisfying these conditions. It should be clear that using this brute-force approach we are exactly counting *all* legal annotated triangulations that are consistent with the annotations of $Q$.

The number of triangulations of $Q$ can be expressed as $c^{\Delta}$, for some positive constant $c$. The number of edges in each triangulation of $Q$ is $O(\Delta)$. Thus, enumerating every sub-set $E'$ of edges takes time $2^{O(\Delta)}$. Verifying whether the chosen sub-set of edges forms a consistent matching and checking for flippability of the edges can easily be done within the same time bound. Hence, overall we can solve sub-problems of size $\Delta$ in time $2^{O(\Delta)}$.

### 5.1.4 Quality of approximation and running time

Having overcome the two difficulties we mentioned previously, we are now left with the analysis of the approximation quality and runtime of the algorithm. Fortunately, essentially all the work has been done in § 4.

From Lemma 1 we obtain that the total number of simple cycle separators is $n^{O(\sqrt{n})}$. Now, we make each such simple cycle separator fat by attaching at most two triangles to each of its edges. It is not hard to see that the total number of fat simple cycle separators, or triangular cycle separators, is $n^{O(\sqrt{n})}$, just observe that for every edge of a simple cycle separator we choose two additional points that will complete the attached triangles to make it fat. Next consider the annotations of the triangular cycle separators. It is easy

to verify that the number of different annotations for the vertices of a separator is at most $n^{O(\sqrt{n})}$ and the number of annotations for its edges is at most $2^{O(\sqrt{n})}$. Therefore, the total number of *annotated* triangular cycle separators is $n^{O(\sqrt{n})}$.

Now, observe that for the analysis in § 4 we essentially only used the following three facts. (1) The number of (annotated) separators of each sub-problems is at most $n^{O(\sqrt{n})}$. (2) A sub-problem of size $\Delta$ can be solved in time $2^{O(\Delta)}$. (3) All further checks can be performed in polynomial time. All three facts still hold in the case of matchings. Hence, the analysis from § 4 goes through and we obtain the same asymptotic bounds for approximation quality and runtime. This finishes the analysis of the algorithm for counting matchings approximately.

### 5.2 Counting trees approximately

Having shown how to count matchings approximately using annotations similar to [6, 7], we now show an annotation scheme that allows to count (crossing-free) spanning trees of $P$ approximately. Fixing an arbitrary vertex $p^* \in P$ we root every spanning tree $F$ at $p^*$ by orienting all of its edges towards $p^*$. This way, every point $p$ in $P \setminus \{p^*\}$ has exactly one outgoing edge, and we call the other end of that edge the *parent* of $p$ in $F$, denoted by $\mathrm{par}(p)$ (for $p^*$ we set $\mathrm{par}(p^*) := \mathrm{nil}$). Moreover, we denote by $d(p)$ the depth of $p$ in $F$, *i.e.*, its distance to $p^*$ in $F$. Note that the resulting oriented spanning trees of $P$ rooted at $p^*$ are in one-to-one correspondence to (ordinary) spanning trees of $P$, so we may count the former. In the following we will write for short *spanning tree* for oriented spanning trees of $P$ rooted at $p^*$. We annotate the CDT $\triangle^F$ of a spanning tree $F$ of $P$ as follows:

- Every vertex $v$ of $F$ is annotated by the pair $<\mathrm{par}(v), d(v)>$.
- Every edge $e$ of $\triangle^F$ is annotated with a bit $b_e$ that again represents whether $e$ is part of $F$ or not.

As for matchings, the annotated CDT $\triangle^F$ will be denoted by $\overline{\triangle}^F$. Again there is a one-to-one correspondence between (oriented, rooted at $p^*$) spanning trees of $P$ and legal annotated triangulations. Therefore, we will again count the latter, just as we did with matchings. In fact, the algorithm is the same as in 5.1.1, the only difference is annotation scheme that encodes a different class of crossing-free structures.

Nevertheless, in the case of spanning trees two things that are less obvious: (1) How to exactly count all legal annotated triangulations of sub-problems of size at most $\Delta$, and (2) the correctness of the annotation scheme: why do we only (over-)count spanning trees? Let us start with the former.

Assume that we arrive at a sub-problem $Q$ of size at most $\Delta$. In $Q$ there are already some annotations present and we have to count *all* annotated triangulations that are consistent with those annotations. We will proceed as before: We enumerate each

triangulation $T = T(Q)$ of $Q$ and every sub-set $E' \subseteq E = E(T)$ of edges of $T$ that have no annotation set yet. All edges of $E'$ are chosen to be in the spanning tree. At this point we can check for flippability conflicts. If we find any conflict we abort and consider the next sub-set of edges, otherwise we continue. Since we are interested in (oriented, rooted at $p^*$) spanning trees, we enumerate *every* possible orientation of the edges in $E'$, observe that there are at most $2^{O(|E'|)} = 2^{O(\Delta)}$ possible orientations we have to go through. This orientation sets the parent of every vertex of the edges of $E'$. Recall that we want to orient towards the root $p^*$. Now we test whether this orientation of the edges of $E'$ is consistent with the annotations already present in $Q$, *e.g.*, a vertex $v$ that was already annotated in $Q$, because it belonged to some separator, must have its parent set. If this parent is in $Q$, then $v$ and its parent must be vertices of some edge of $E'$, otherwise we can abort this run. This in particular tests that the *unique* parent of every vertex of $T$ is set correctly if this parent belongs to $Q$ as well.

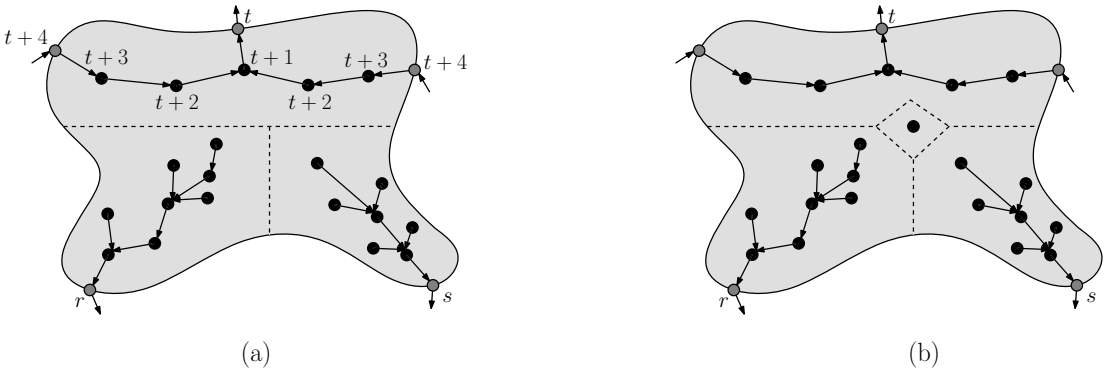


Figure 2: A sub-problem $Q$ of size at most $\Delta$ is shown in gray. The boundary of $Q$ is shown simplified, *i.e.*, no triangular cycle separator is shown. The set of edges $E'$ chosen to be in a tree is shown along with an orientation of its elements. For simplicity, the triangulation of $Q$ that the set $E'$ is chosen from is not shown, so in particular we are assuming that there is no flippability conflict. In (a) there are three connected components $F_1, F_2, F_3$, separated by dashed lines for clarity. The gray vertices on the boundary of $Q$ are fully annotated, so we can start a BFS for each component at any of those vertices. The labels $r, s, t$ represent the annotated depth of those vertices. Starting with the vertex at depth $t$, every vertex of that component gets the shown depth assigned by the BFS. By the time we arrive at the vertices of depth $t + 4$ on the boundary of $Q$, since they are fully annotated, we compare $t + 4$ with the depth therein annotated and verify for consistency. If they are inconsistent then we abort the run with this set of orientations to the edges of $E'$. In (b) there are four connected components but the single vertex in the middle will never be annotated since there is no edge in $E'$ that has it as a vertex, so even if we found consistent annotations for the other components, this run is aborted.

Having chosen $E'$ and having set orientations for its elements, note that we are left with some set of connected components $F_1, \dots F_k$ formed by the (oriented) edges of $T$ that we have chosen to be in the tree, see Figure 2.a. Now, for each connected component $F_i$, $1 \leq i \leq k$, let $v^i$ denote an arbitrary vertex that was found fully annotated when first entering $Q$, any vertex of the boundary of $Q$ could be such a vertex since they are all fully annotated. Perform a Breadth-first search (BFS) of $F_i$ starting at $v^i$ as if the edges of $F_i$ had no orientations. Let $v$ be the current vertex the BFS is about to expand its neighborhood of, so in the beginning $v = v^i$, and let $d_v$ be its depth. Set the depth of every vertex $w$ of $F_i$ that was found while expanding the neighborhood of $v$ to $d_v + 1$ or $d_v - 1$ depending on the orientation of the edge $e = vw$, *i.e.*, if $e$ is oriented towards $w$, then $w$ is the parent of $v$ and thus we set its depth to $d_v - 1$, and if $e$ is oriented towards $v$ we set it to $d_v + 1$. If the depth of $w$ is found to be already set when doing the BFS, then we just verify that the value the algorithm would write there and the value already there coincide. If those two numbers do not coincide, then we abort and discard this set of orientations given to the edges of $E'$, otherwise we continue with the BFS. If the whole connected component $F_i$ was successfully traversed, then we manage to find a consistent set of annotations for it and we can continue with another connected component, if any. If after considering every connected component there is a vertex of $T$ that has not been fully annotated, then we dismiss this run as well, since we were not able to find a consistent set of annotations for $E'$ in $Q$, see Figure 2.b, otherwise we declare this run a success.

With this method we clearly exactly count *all* legal annotated triangulations that are consistent with the annotations we started with for sub-problem $Q$. It remains to show that, overall, *only* spanning trees of $P$ are (over-)counted.

**Lemma 3.** *Let $P$ be a set of $n$ points. Then our annotation scheme encodes the spanning trees of $P$ unequivocally.*

*Proof.* Consider a legal annotated triangulation $\overline{T}$ of $P$. The annotations $b_e$ of its edges induce a set of edges $F$ such that $\overline{T}$ is the CDT $\triangle^F$ of $F$ (by flippability constraints). Moreover, the edges in $F$ are consistent with the annotations $<\mathrm{par}(v), d(v)>$ of the vertices of $\overline{T}$, and the latter give $F$ an orientation. Thus, every vertex except for the root $p^*$ has a unique parent, so it has out-degree 1, and $F$ consists of exactly $n - 1$ edges. It remains to show that $F$ has no undirected cycle, then the annotations induce an (oriented, rooted at $p^*$) spanning tree of $P$. For this, first note that any undirected cycle in $F$ would also be a directed cycle (according to the orientation of $F$ given by the parent annotations), since all vertices have out-degree at most 1. Moreover, consistency of the annotated depths $d(v)$ implies that $d(\mathrm{par}(v)) = d(v) - 1$ for all vertices $v$. Hence, we also cannot have any directed cycle in $F$, as otherwise for at least one edge of the cycle the depths cannot be consistent. This finishes the proof.

■

Finally, it remains to show what approximation factor and running time we obtain with this algorithm. For the former, the explanation given in § 5.1.4 follows verbatim.

For the running time it follows almost verbatim, the only difference is that when choosing a sub-set of edges $E'$ that will be part of a spanning tree, we have to orient them as well. This orientation comes with a $2^{|E'|} = 2^{O(\Delta)}$ overhead. That is, instead of having a runtime of $T(\Delta) = c^{\Delta} \cdot 2^{O(\Delta)} = 2^{O(\Delta)}$, as we had when counting matchings in 5.1.4, we now have a runtime of $2^{O(\Delta)} \cdot \ T(\Delta) = 2^{O(\Delta)}$, which is asymptotically the same. The time required to perform the breath-first searches also have no effect in asymptotic terms. Thus, choosing again $\Delta = \sqrt{n}\log(n)$ gives us an overall running time of $T(n) = 2^{O\left(\sqrt{n}\log(n)\right)}$. Hence, both algorithm have the same asymptotic behavior.

# 6 Conclusions

In this paper we have shown algorithms that, given a set of points, count triangulations, crossing-free matchings and crossing-free spanning trees approximately. Both, the approximation ratio and the running time are sub-exponential. We would like to remark that using the set of annotations for crossing-free spanning cycles shown in [6] we can modify the algorithms presented in this paper to also count crossing-free spanning cycles approximately, again with sub-exponential approximation ratio and running time. After showing the algorithms for matchings and spanning trees, this extension is straightforward. On the other hand, the annotations shown in this paper to count spanning trees were not known before. They are compatible with the algorithm of [6], and combining the two yields an algorithm to count crossing-free spanning trees *exactly* in time $n^{O(k)}$, where $k$ is the number of onion layers of the given set of points. Specifically, as long as $k$ is fixed we can exactly count spanning trees in polynomial time, which is an interesting result in itself.

Finally, we can express the cardinality of any of the classes of crossing-free structures considered in this paper as $c^n$, where $c$ depends on the given set of points but is sandwiched between two positive constants. Although the approximation ratios shown in this paper are rather large, our algorithms compute a $(1 + o(1))$-approximation of the base $c$, and it does so in sub-exponential time. No algorithm with this property was known before. However, it remains an open problem to find an algorithm with sub-exponential approximation ratio and running time $2^{o(\sqrt{n}\log(n))}$, *e.g.*, polynomial.

# References


[1] O. Aichholzer. The path of a triangulation. In *Proceedings of the fifteenth annual symposium on Computational geometry*, SCG '99, pages 14–23, New York, NY, USA, 1999. ACM. One citation on page 2.

[2] O. Aichholzer, T. Hackl, B. Vogtenhuber, C. Huemer, F. Hurtado, and H. Krasser. On the number of plane graphs. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 504–513. ACM, 2006. One citation on page 3.

[3] 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. One citation on page 3.

[4] O. Aichholzer and H. Krasser. The point set order type data base: A collection of applications and results. In *CCCG*, volume 1, pages 17–20, 2001. One citation on page 3.

[5] O. Aichholzer, G. Rote, B. Speckmann, and I. Streinu. The zigzag path of a pseudo-triangulation. In *Algorithms and Data Structures*, pages 377–388. Springer, 2003. One citation on page 3.

[6] V. Alvarez, K. Bringmann, R. Curticapean, and S. Ray. Counting crossing-free structures. In *Proceedings of the twenty-eighth annual symposium on Computational geometry*, SoCG '12, pages 61–68, New York, NY, USA, 2012. ACM. 11 citations on pages 3, 4, 9, 10, 11, 13, and 16.

[7] V. Alvarez, K. Bringmann, and S. Ray. Counting triangulations and other crossing-free structures via onion layers. Submitted, 2012. 7 citations on pages 3, 9, 10, 11, and 13.

[8] V. Alvarez, K. Bringmann, and S. Ray. A simple sweep line algorithm for counting triangulations and pseudo-triangulations. Submitted, 2012. 5 citations on pages 2, 3, and 4.

[9] V. Alvarez and R. Seidel. A simple aggregative algorithm for counting triangulations of planar point sets and related problems. In *Proceedings of the twenty-ninth annual symposium on Computational Geometry*, SoCG '13, pages 1–8, New York, NY, USA, 2013. ACM. 7 citations on pages 1, 3, 4, and 5.

[10] D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1 - 3):21 – 46, 1996. First International Colloquium on Graphs and Optimization. 2 citations on pages 2 and 12.

[11] S. Bespamyatnikh. An efficient algorithm for enumeration of triangulations. *Computational Geometry*, 23(3):271 – 279, 2002. 2 citations on pages 2 and 12.

[12] L. P. Chew. Constrained delaunay triangulations. In *Proceedings of the third annual symposium on Computational geometry*, SCG '87, pages 215–222, New York, NY, USA, 1987. ACM. One citation on page 9.

[13] E. Demaine. Simple polygonizations. `http://erikdemaine.org/polygonization/`. Accessed October 25, 2013. One citation on page 3.

[14] H. Djidjev and S. M. Venkatesan. Reduced constants for simple cycle graph separation. *Acta Inf.*, 34(3):231–243, 1997. One citation on page 4.

[15] A. Dumitrescu, B. Gärtner, S. Pedroni, and E. Welzl. Enumerating triangulation paths. *Computational Geometry*, 20(1):3–12, 2001. One citation on page 3.

[16] P. Flajolet and M. Noy. Analytic combinatorics of non-crossing configurations. *Discrete Mathematics*, 204(1â“3):203 – 229, 1999. ¡ce:title¿Selected papers in honor of Henry W. Gould¡/ce:title¿. One citation on page 3.

[17] A. Garcıa, M. Noy, and J. Tejel. Lower bounds on the number of crossing-free subgraphs of $k_n$. *Computational Geometry*, 16(4):211–221, 2000. One citation on page 3.

[18] O. Hjelle and M. Dæhlen. *Triangulations and Applications (Mathematics and Visualization)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. One citation on page 9.

[19] M. Hoffmann, A. Schulz, M. Sharir, A. Sheffer, C. D. Tóth, and E. Welzl. Counting plane graphs: Flippability and its applications. In *Thirty Essays on Geometric Graph Theory*, pages 303–325. Springer, 2013. One citation on page 3.

[20] F. Hurtado and M. Noy. Counting triangulations of almost-convex polygons. *Ars Combinatoria*, 45:169–180, 1997. One citation on page 2.

[21] N. Katoh and S.-I. Tanigawa. Fast enumeration algorithms for non-crossing geometric graphs. *Discrete & Computational Geometry*, 42(3):443–468, 2009. 3 citations on pages 2, 3, and 12.

[22] C. L. Lawson. Generation of a triangular grid with applications to contour plotting. *Jet Propul. Lab. Techn. Memo*, 299, 1972. One citation on page 2.

[23] T. Leighton. Notes on better master theorems for divide-and-conquer recurrences. Technical report, Massachusetts Institute of Technology, 1996. One citation on page 8.

[24] G. L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *Journal of Computer and System Sciences*, 32(3):265 – 279, 1986. One citation on page 4.

[25] 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*, EuroCG '04, March 2004. 2 citations on pages 2 and 3.

[26] A. Razen and E. Welzl. Counting plane graphs with exponential speed-up. In *Rainbow of computer science*, pages 36–46. Springer, 2011. One citation on page 3.

[27] 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. One citation on page 3.

[28] M. Sharir and A. Sheffer. Counting triangulations of planar point sets. *Electr. J. Comb.*, 18(1), 2011. 2 citations on pages 2 and 4.

[29] M. Sharir, A. Sheffer, and E. Welzl. On degrees in random triangulations of point sets. *J. Comb. Theory Ser. A*, 118(7):1979–1999, Oct. 2011. 3 citations on pages 1, 2, and 4.

[30] M. Sharir, A. Sheffer, and E. Welzl. Counting plane graphs: Perfect matchings, spanning cycles, and kasteleyns technique. *Journal of Combinatorial Theory, Series A*, 120(4):777–794, 2013. One citation on page 3.

[31] A. Sheffer. Numbers of plane graphs. `http://www.cs.tau.ac.il/~sheffera/counting/PlaneGraphs.html`. Accessed October 25, 2013. One citation on page 3.

[32] R. Sibson. Locally equiangular triangulations. *The Computer Journal*, 21(3):243–245, 1978. One citation on page 2.

[33] M. Wettstein. Algorithms for counting crossing-free configurations. Master's thesis, Dept. of Computer Science, ETH Zürich, 2013. 3 citations on pages 3 and 4.