Peeling and Nibbling the Cactus: Subexponential-Time Algorithms for Counting Triangulations and Related Problems*

Dániel Marx and Tillmann Miltzow

Institute for Computer Science and Control,
Hungarian Academy of Sciences (MTA SZTAKI)
dmarx@cs.bme.hu, t.miltzow@gmail.com

Abstract

Given a set of n points S in the plane, a triangulation T of S is a maximal set of non-crossing segments with endpoints in S. We present an algorithm that computes the number of triangulations on a given set of n points in time $n^{(11+o(1))\sqrt{n}}$, significantly improving the previous best running time of $O(2^n n^2)$ by Alvarez and Seidel [SoCG 2013]. Our main tool is identifying separators of size $O(\sqrt{n})$ of a triangulation in a canonical way. The definition of the separators are based on the decomposition of the triangulation into nested layers ("cactus graphs"). Based on the above algorithm, we develop a simple and formal framework to count other non-crossing straight-line graphs in $n^{O(\sqrt{n})}$ time. We demonstrate the usefulness of the framework by applying it to counting non-crossing Hamilton cycles, spanning trees, perfect matchings, 3-colorable triangulations, connected graphs, cycle decompositions, quadrangulations, 3-regular graphs, and more.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases computational geometry, triangulations, exponential-time algorithms

Digital Object Identifier 10.4230/LIPIcs.SoCG.2016.52

1 Introduction

Given a set S of n points in the plane, a triangulation T of S is defined to be a maximal set of non-crossing line segments with both endpoints in S. This set of segments together with the set S defines a plane graph. It is easy to see that every bounded face of a triangulation T is indeed a triangle. We assume that S is in general position: no three points of S are on a line. Triangulations are one of the most studied concepts in discrete and computational geometry, studied both from combinatorial and algorithmic perspectives [8, 9, 11, 22, 23, 27, 28, 43]. It is well known that the number of possible triangulations of n points in convex position is exactly the $(n-2)$-th Catalan number, but counting the number of triangulations of arbitrary point sets seems to be a much harder problem. There is a long line of research devoted to finding better and better exponential-time algorithms for counting triangulations [1–7, 21, 26, 30, 33, 42]. The sequence of improvements culminated in the $O(2^n n^2)$ time algorithm of Alvarez and Seidel [5], winning the best paper award at SoCG 2013. Our main result significantly improves the running time of counting triangulations by making it subexponential:

* Supported by the ERC grant "PARAMTIGHT: Parameterized complexity and the search for tight complexity results", no. 280152. The first author is supported by OTKA grant NK105645.

▶ **Theorem 1** (General Plane Algorithm). *There exists an algorithm that, given a set S of n points in the plane, computes the number of all triangulations of S in $n^{(11+o(1))\sqrt{n}}$ time.*

It is very often the case that restricting an algorithmic problem to planar graphs allows us to solve it with much better worst-case running time than what is possible for the unrestricted problem. One can observe a certain "square root phenomenon": in many cases, the best known running time for a planar problem contains a square root in the exponent. For example, the 3-Coloring problem on an n-vertex graph can be solved in subexponential time $2^{O(\sqrt{n})}$ on planar graphs (e.g., by observing that a planar graph on n vertices has treewidth $O(\sqrt{n})$), but only $2^{O(n)}$ time algorithms are known for general graphs. Moreover, it is known that if we assume the Exponential-Time Hypothesis (ETH), which states that there is no $2^{o(n)}$ time algorithm for n-variable 3SAT, then there is no $2^{o(\sqrt{n})}$ time algorithm for 3-Coloring on planar graphs and no $2^{o(n)}$ time algorithm on general graphs [37]. The situation is similar for the planar restrictions of many other NP-hard problems, thus it seems that the appearance of the square root of the running time is an essential feature of planar problems. A similar phenomenon occurs in the framework of parameterized problems, where running times of the form $2^{O(\sqrt{k})} \cdot n^{O(1)}$ or $n^{O(\sqrt{k})}$ appear for many planar problems and are known to be essentially best possible (assuming ETH) [10, 12–20, 24, 25, 31, 32, 39–41, 44].

A triangulation of n points can be considered as a planar graph on n vertices, hence it is a natural question whether the square root phenomenon holds for the problem of counting triangulations. Indeed, for the related problem of finding a minimum weight triangulation, subexponential algorithms with running time $n^{O(\sqrt{n})}$ are known [34, 35]. These algorithms are based on the use of small balanced separators. Given a plane triangulation on n points in the plane, it is well known that there exists a balanced $O(\sqrt{n})$-sized separator that divides the triangulation into at least two independent graphs [36]. The basic idea is to guess a correct $O(\sqrt{n})$-sized separator of a minimum weight triangulation and recurse on every occurring subproblem. As there are only $n^{O(\sqrt{n})}$ potential graphs on $O(\sqrt{n})$ vertices, one can show that the whole algorithm takes $n^{O(\sqrt{n})}$ time [34, 35].

Unfortunately, this approach has serious problems when we try to apply it to counting triangulations. The fundamental issue with this approach is that a triangulation of course may have more than one $O(\sqrt{n})$-sized balanced separators and hence we may overcount the number of triangulations, as a triangulation would be taken into account in more than one of the guesses. To get around this problem, an obvious simple idea would be to say that we always try to guess a "canonical" separator, for example, the lexicographically first separator. However, it is a complete mystery how to guarantee in subsequent recursion steps that the separator we have chosen is indeed the lexicographic first for all the triangulations we want to count. Perhaps the most important technical idea of the paper is finding a suitable way of making the separators canonical.

This extended abstract is committed to give a comprehensive self-contained explanation of the main concepts of the algorithm. For a version with all details and proofs see [38].

1.1 Preliminaries

We interpret a collection of points V and non-crossing segments E in $\mathbb{R}^2$ as a plane graph, if every segment $e \in E$ shares exactly its endpoints with V. We usually identify points with vertices and edges with segments. We always denote by n the number of vertices. A plane graph induces naturally a partition of the plane into open faces, open segments and a finite collection of points. The unique unbounded face is called the *outer face*. We call a plane graph G a *cactus graph* (or just *cactus*) if all its vertices and edges are incident to

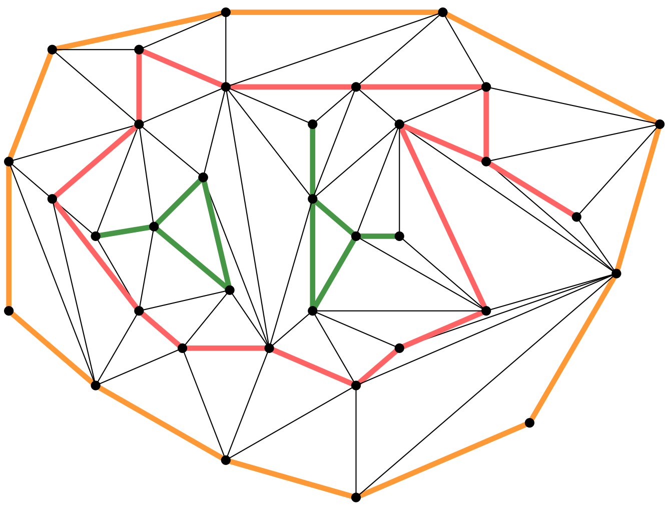


Figure 1 A triangulation with three cactus layers. Layer 1 is colored orange, Layer two is colored red and Layer 3 is colored green. Note that the layers do not need to be connected.

the *outer face*. Such a graph is outerplane, but some outerplane graphs are not cacti. For convenience, we do *not* require cactus graphs to be connected. A *triangulation* of a set of points is a maximal plane graph on those points.

We define a decomposition of a triangulation into nested *(cactus) layers* of cacti. The first (cactus) layer is defined by the set of vertices and edges incident to the outer face. Inductively, the i-th layer is defined by the vertices and edges incident to the outer faces after the first $i-1$ layers are removed and has *index* i. We say layer i is further *outside* then layer j if $i < j$ and in this case layer j is more *inside* than layer i. The *outerplanar index* of a graph is defined by the number of non-empty (cactus) layers. Further, we can give each vertex uniquely the *index* of the layer it is contained in. It is not difficult to see that the index equals the distance to the boundary of the convex hull.

We define $\partial CH(S)$ as the boundary of the convex hull of the point set S. The *onion layers* of a set of points S are defined inductively in a similar fashion. The first layer is $\partial CH(S)$. The i-th layer is the boundary of the convex hull after the first $i-1$ layers are removed.

The definition of cactus layers and onion layers should not be confused: the onion layers are completely defined by the point set only, whereas cactus layers are defined by the point set and the triangulation. In particular, it is easy to construct a point set with an arbitrary large number of onion layers, but having a triangulation with only two cactus layers.

1.2 Results

Given a triangulation T, we define small *canonical* separators by distinguishing two cases. If T has more than $\sqrt{n}$ cactus layers, then one of the first $\sqrt{n}$ layers has size at most $\sqrt{n}$ and we can define the one with smallest index to be the canonical separator. Using such a separator, we *peel off* some cacti to reduce the problem size. In the case when we have only a few cactus layers, we can define short canonical separator paths from any vertex to the outer face of the triangulation. We formalize both ideas into a dynamic programming algorithm. The main difficulty is to define the subproblems appropriately. We define ring subproblems to be suitable for layer separators and nibbled ring subproblems to be suitable for the separator paths.

As a byproduct of this algorithmic scheme, we can efficiently count triangulations with a small number of layers. This is similar to previous work on finding a minimum weight triangulation [6] and counting triangulations [2] for point sets with a small number of onion layers.

▶ **Theorem 2** (Thin Plane Algorithm). *There exists an algorithm that given a set S of n points in the plane computes the number of all triangulations of S with outerplanar index k in $n^{O(k)}$ time.*

One may want to count triangulations subject to certain constraints (e.g., degree bounds, or bounds on the angles of the triangles, etc.) or generalize the problem to counting colored triangulations with colors on the vertices or edges. We introduce an annotated version of the problem to express such generalizations in a clean and formal way. An *annotated* triangle is a 9-tuple consisting of 3 points of S, which form an empty triangle and 6 strings, one for each vertex and edge of the triangle. An *annotation system* is a list L of annotated triangles. An *annotated triangulation* T is a triangulation with a string defined for each vertex and each edge (so these strings define an annotated triangle for each triangle of T). Given an annotation system L, we call an annotated triangulation T *valid* if every annotated triangle Δ of T belongs to L. With little extra effort, we can generalize our algorithms to count also valid annotated triangulations. We denote by $|L|$ the number of annotated triangles and assume that each string can be described with $n^{O(1)}$ bits.

▶ **Theorem 3** (Counting Annotated Triangulations). *Given an annotation system L and a set S of n points in the plane, we can count all valid annotated triangulations in time $n^{(11+o(1))\sqrt{n}} \cdot |L|^{(12+o(1))\sqrt{n}}$.*

As examples of this generalization, we can count triangulations that are 3-colorable or where each point has a specified degree in the triangulation: all we need is to carefully design a suitable annotation system.

▶ **Theorem 4.** *Given a set S of n points in the plane, we can count all 3-colorable triangulations of S in time $n^{O(\sqrt{n})}$.*

▶ **Theorem 5.** *Given a set S of n points in the plane with prescribed degrees on each vertex, we can count all triangulations T satisfying the degree constraints in time $n^{O(\sqrt{n})}$.*

More generally, instead of triangulations, we could be interested in counting other geometric graph classes, such as non-crossing perfect matchings, non-crossing Hamilton cycles, etc. Surprisingly, many such problems can be expressed in a completely formal way in our framework of counting annotated triangulations. The idea here is to extend the geometric graph into a 2-edge-colored triangulation, with one color forming the original geometric graph itself and the other color representing the edges of the triangulation that were not present in the original graph. To make this idea work, we have to ensure that for each member of our graph class, we count only one 2-edge-colored triangulation. This is non-trivial, as a given geometric graph can be extended into a 2-edge-colored triangulation in many different ways. Similarly to previous work [2, 4], we use the notion of constrained Delaunay triangulation (see [29]) to enforce that each graph has a unique extension into a valid 2-edge-colored triangulation. By formalizing this idea and carefully designing annotation systems, it is possible to get $n^{O(\sqrt{n})}$ time algorithms for a large number of graph classes. The following theorem states some important examples to demonstrate the applicability of our approach.

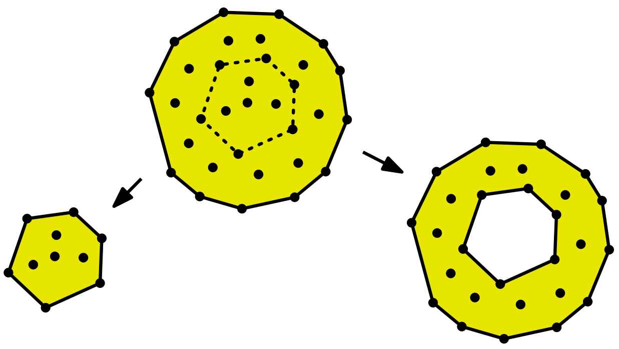


Figure 2 Cactus layes are ideal separators, as they separate in a simple way the inside from the outside and we can easily *peel off* large layers from the outside.

▶ **Theorem 6** (Counting Geometric Structures). *The following non-crossing structures can be counted in time $n^{O(\sqrt{n})}$ on a set of n points in the plane: the set of all graphs, perfect matchings, cycle decompositions, Hamilton cycles, Hamiltonian paths, Euler tours, spanning trees, d-regular graphs, and quadrangulations.*

We would like to emphasize that the proof of Theorem 6 uses the algorithm of Theorem 3 as a black box. Thus these results can be proved in a completely formal way without the need for revisiting the details of the proof of Theorem 3. In addition to the actual algorithms presented in the paper, we consider our second main contribution to be the development of the framework of annotated triangulations and demonstrating its flexibility in modeling other problems.

1.3 The Essence of the Key Ideas of the Algorithm

The complete proof can be found in the full online version. We want to use this extended abstract to present the key ideas of the algorithm in a level of detail that gives the reader a good understanding of the algorithm without indulging into the nuances of the technical details. Before we do this, we want to give the reader the essence in an even more condensed high level form.

Similar to almost all previous algorithms, our algorithm is based on separators. There are two major differences: the separators are defined not in terms of the input (point set), but in terms of the output (triangulation). We use *two* kind of separators. Depending on the type of triangulation we want to count, we choose which kind of separator we use.

Given a triangulation with outerplanar index $k > \sqrt{n}$, there exists a layer of size at most $\sqrt{n}$ by the pigeonhole principle, see Figure 2. This layer is an ideal candidate for a separator. As layers are nested and separate the inner from the outer part completely. In order to make them canonical, we choose the *outermost* small cactus layer and *peel it off*. In subsequent recursions, we have to ensure that we count only triangulations where this was indeed the outermost small cactus layer. This can be done by guessing all possible sizes of all layers that have smaller index.

In case that the outerplanar index k is below $\sqrt{n}$, there exists a path of length at most $k - 1$ from any vertex to the outermost layer, see Figure 3. (The idea is that every vertex either is adjacent to the outer face or has a neighbor with smaller index.) If done correctly, these paths can be used as a separator within a well designed dynamic programming scheme. Anagnostou and Corneil [6] demonstrated this for finding a minimum weight triangulation,

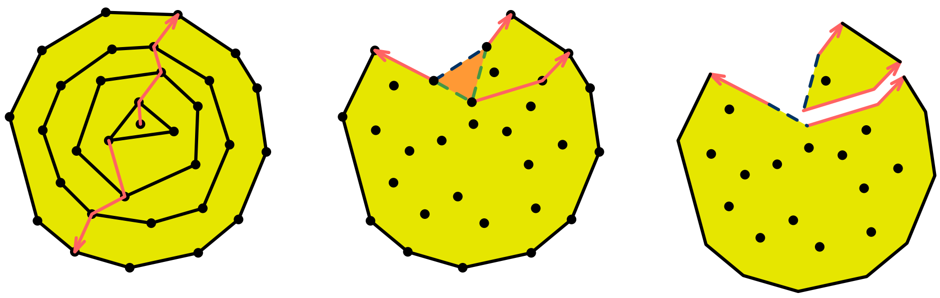


Figure 3 Paths from the interior to the outside are short separators, if the triangulation does not have too many layers. The resulting dynamic programming scheme is a little unintuitive, as it *nibbles off* the algorithmic problem from the outside.

using layers defined in terms of the input instead of layers defined in terms of the output. The algorithmic idea is essentially the same. Alvarez, Bringmann, Curticapean and Ray showed how to make these separators canonical and thus suitable for counting problems [2], by giving each vertex a fixed distinguished rank and always choose the next vertex on the separator path with the smallest available rank.

We use these separator paths in an, at first, unintuitive way. Consider the case, where we have already made a few recursion steps, see Figure 3. (The very first step is degenerate and not suitable for an illustrative example.) In this situation, we have to triangulate a simple polygon, where one edge $e = (u, v)$ is singled out. Some part of the polygon comes from separator paths of previous recursion steps. We guess all triangles $\Delta = \Delta(u, v, w)$ incident to $e = (u, v)$ and all short paths p from w to $\partial CH(S)$. In this way, we attain a large and a small subproblem. The triangle Δ defines special edges for subsequent subproblems. We repeat the procedure for all appearing subproblems till the subproblems are of constant size. We *nibble off* small bites in each recursion step. Only at later stages larger bites are taken.

The technical and conceptual difficulties come from the need to combine the two different separators into *one* dynamic programming scheme. Note that we might first guess a layer then the sizes of layers with smaller index and thereafter use the path separators as described above. Thus, we have to define very carefully subproblems for our dynamic programing routine that are specifically designed to work for both separators.

The runtime bound follows from the size of the separators. Whenever we guess a separator of size $O(\sqrt{n})$, we have at most $\binom{n}{O(\sqrt{n})} = n^{O(\sqrt{n})}$ possibilities.

2 Ring Subproblems

Our algorithm is based on dynamic programming: we define a large number of subproblems that are *more general* than the problem we are trying to solve. We generalize the problem by considering *rings:* we need to triangulate a point set in a region between a polygon and a cactus. Additionally, we may have layer-constraints prescribing that a certain number of vertices should appear on certain layers.

We proceed to give the crucial ingredients of ring subproblems, which serve as the basis of the dynamic programming, see Figure 4 for an illustration. We omit any technical conditions, which are important to show correctness, but are not essential for the main ideas.

▶ **Definition 7** (Ring Subproblems). A *ring subproblem* $\mathcal{S}$ consists of an *outer layer* Q_{out} an *inner layer* Q_{in}, some width information, and an optional *layer-constraint vector* c.

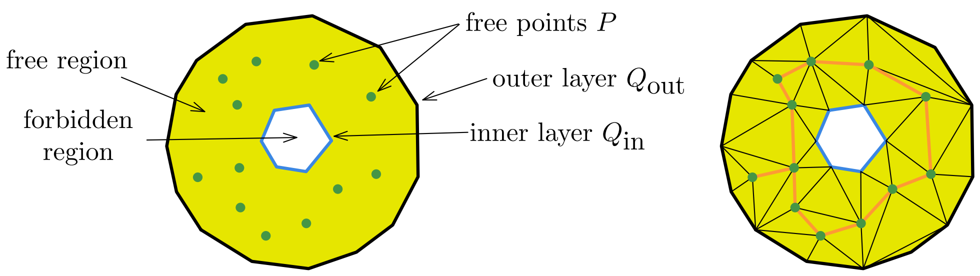


Figure 4 At the left a layer-unconstrained ring subproblem is depicted. At the right a valid triangulation of the ring subproblem is drawn. It consists of three layers: The outer layer drawn in black, the "middle" layer drawn in orange and the inner layer drawn in blue.

outer layer: The *outer layer* Q_{out} is a simple poylgon.

inner layer: The *inner layer* Q_{in} is a cactus contained in the outer layer. An important special case is that the inner layer might be empty.

layer-constraint vector: This vector c prescribes the size of each layer of the triangulation we are looking for. We do not always specify a layer constraint vector: We distinuish between layer-constrained ring subproblems and layer-*un*constrained ring subproblems.

width: The *width* is a natural number w that specifies how many non-empty cactus layers any valid triangulation may have.

free region: The *free region* is the region "between" the inner and outer layer.

free points: The *free points* P are the points of the original point set inside the free region.

Given a layer-unconstrained ring subproblem $\mathcal{S}$ and a layer-constraint vector c, we denote by $\mathcal{S}(c)$ the layer-constrained ring subproblem defined by them.

▶ **Definition 8** (Valid Triangulation). Given a ring subproblem $\mathcal{S}$, consider a graph T extending the graph formed by $Q_{\text{in}} \cup Q_{\text{out}}$ and the free points. We assume that all faces in the free region are triangles and there are no edges outside the free region. The graph T can be decomposed into cactus layers L_i as explained in Section 1.1. We call such a graph T of $\mathcal{S}$ a *valid triangulation of* $\mathcal{S}$ if the following conditions are satisfied:

1. The inner layer Q_{in} corresponds indeed with the innermost cactus layer of T.
2. The number of layers equals the width w.
3. In case that a layer constraint vector c is given, we require that each layer has the size given by c.

Note that Condition 3 subsumes Condition 2, in case a layer-constraint vector is given.

▶ **Theorem 9.** *There exists an algorithm that given an annotated layer-unconstrained ring subproblem $\mathcal{S}$ with n free points computes the number of all triangulations of $\mathcal{S}$ in time $n^{(11+o(1))\sqrt{n}}$.*

Given a set of points S, we can define a set of ring subproblems, such that each triangulation of S is a valid triangulation of exactly one of the ring subproblems. We use the convex hull as outer layer and the empty graph as inner layer. We do not need layer constraints. It is technical, but straightforward to show that this works indeed.

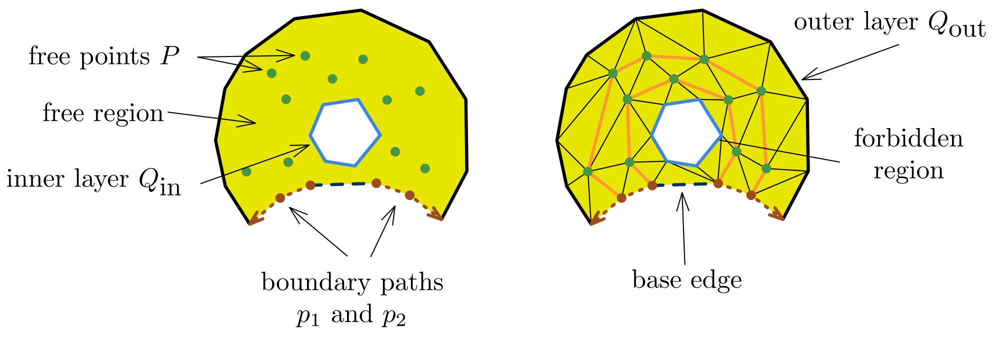


Figure 5 Left: A nibbled ring subproblem $\mathcal{S}$ consisting of a base edge depicted in dark blue dashed; two boundary paths, displayed in dotted brown from the base edge to the outer layer Q_{out}, displayed in black; the free region depicted in yellow; containing free points, depicted in green; The forbidden region is depicted in white. Right: A nibbled ring subproblem together with a valid triangulation. It consists of four layers. The middle two layers are colored orange. The base edge belongs to the third layer.

3 Thin Rings

This section sketches the proof of the following theorem, which gives an algorithm for solving ring subproblems with a certain width w. This algorithm will be invoked by the main algorithm for values $w \leq \sqrt{n}$. We will sketch the main ideas of the algorithm and its runtime analysis.

▶ **Theorem 10.** *There exists an algorithm that given a (layer-constrained or layer-unconstrained) ring subproblem $\mathcal{S}$ with width w and at most n free points, computes the number of all valid triangulations of $\mathcal{S}$ in time $n^{(5+o(1))w}$.*

Theorem 10 implies easily Theorem 2 in a similar fashion as Theorem 9 implies Theorem 3. We use path-separators for this algorithm. This requires a yet more specialized definition of subproblems for our dynamic programming scheme: *nibbled ring subproblems.*

We start with the key ingredients of *nibbled ring problems* omitting the parts that are equivalent to ring subproblems, see Figure 5 for an illustration of the nibbled ring subproblems. We omit the inner layer, the width, the free region, the free points and the layer constraints.

▶ **Definition 11** (Nibbled Ring Subproblem)**.** The outer layer is always a polygonal chain (i.e. , a plane graph that is a path as a graph.). We have additionally two disjoint *boundary paths* p_1 and p_2 that end on the outer layer and start at the base edge. Note that the boundary paths are allowed to have length zero. The base edge, the boundary paths and the polygonal chain form a cycle that bounds the free region, see Figure 5.

Next we define valid triangulations for a nibbled ring subproblems. As for ring problems, we also want to define a cactus layer structure for nibbled ring problems that is consistent with the previous definition. The definition of cactus-layers does not generalize in a straightforward way to nibbled ring subproblems. Recall that the length of the shortest path from any vertex of the i-th layer to the boundary of the convex hull (outer layer) has length $i-1$. Therefore, we will define the layers in terms of distance to the outer layer. This way of defining layers is equivalent for triangulations of point sets and carries over to nibbled ring subproblems.

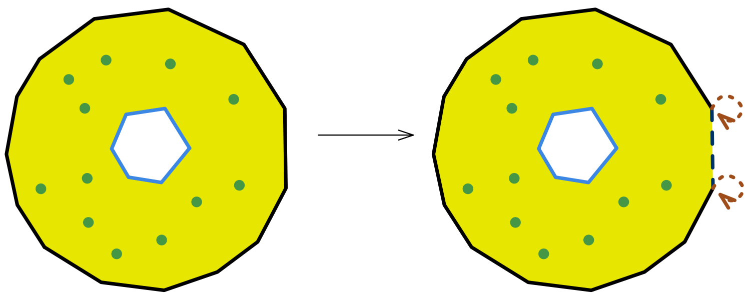


Figure 6 Left: a ring subproblem Right: the transformed nibbled ring subproblem.

Another important ingredient is the concept of *order labels* of the vertices. Each vertex has a distinct order label from $1, \ldots, n$. The importance here is that the order label is defined as a preprocessing step. It is *never* altered during the algorithm.

▶ **Definition 12** (Valid Triangulation)**.** Given a nibbled ring subproblem $\mathcal{S}$ consider a triangulation T of $\mathcal{S}$. For each vertex $v \in V(T)$, we define $d(v)$ as the length of the shortest path to the outer-layer Q_{out}. We call T a *valid triangulation of* $\mathcal{S}$ if the conditions for ring subproblems and the following additional condition are satisfied:

4. For any vertex v_i of any boundary path $p = (v_1, \ldots, v_k)$, we have that $d(v_i) = d(v_{i+1})+1$ holds. Furthermore, we require that the neighbor v_{i+1} of v_i be the neighbor of v_i in T with the smallest order label among the neighbors with smaller distance to the outer layer.

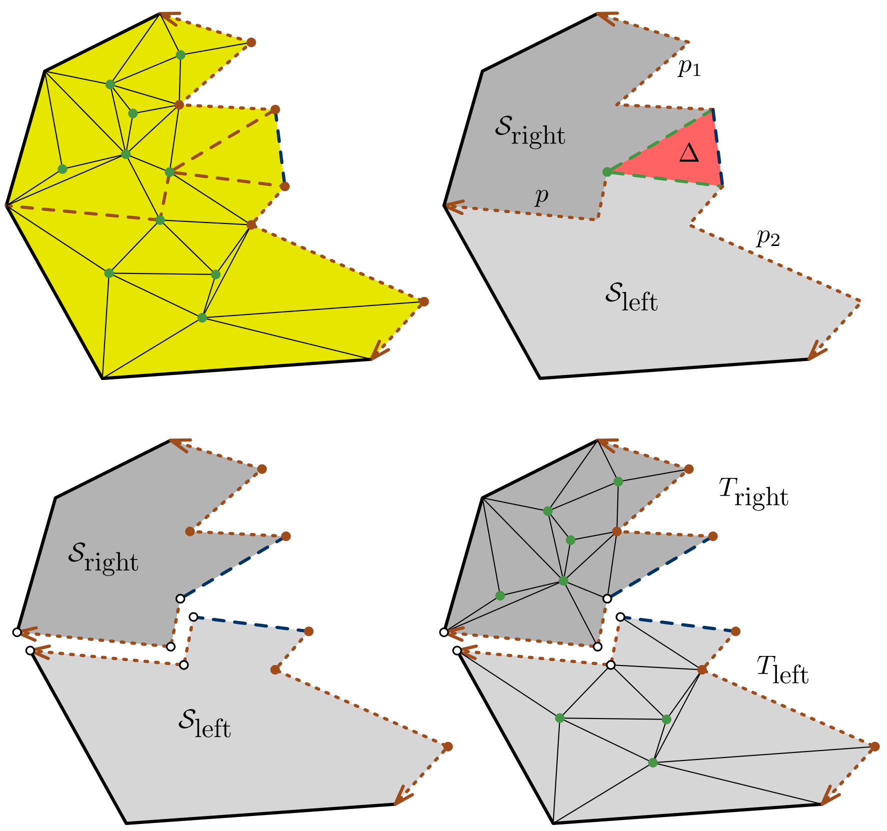
▶ **Theorem 13.** *There exists an algorithm that given a nibbled ring subproblem $\mathcal{S}$ with width w computes the number of all valid triangulations of $\mathcal{S}$ in time $n^{(5+o(1))w}$.*

Theorem 13 easily implies Theorem 10, using a simple transformation from ring subproblems to nibbled ring subproblems, see Figure 6.

We start with a description of the separators, see Figure 7. Given a valid triangulation T of a nibbled ring subproblem $\mathcal{S}$, recall that every vertex on layer i has a neighbor w.r.t. T in layer $i-1$. Furthermore, there is a unique triangle Δ incident to the base edge. From vertex v of Δ, which is not incident to the base edge, there exists a path to the outer layer of $\mathcal{S}$ by always choosing an adjacent vertex closer to the outer layer. There is *exactly one* such path p, if we further require that the vertex with lowest order label is taken. (Recall that the order label is fixed in advance.) Such paths are called *canonical outgoing paths.*

Let us now move to the algorithmic scheme using the paths as separators. We recurse on a nibbled ring subproblem by guessing all potential such triangles Δ incident to the base edge and all potential canonical paths p as described above. For each such path, we can define two subproblems $\mathcal{S}_{\text{right}}(p)$ and $\mathcal{S}_{\text{left}}(p)$, see Figure 7. We can restrict our triangulation T to these subproblems and receive two new triangulations T_{left} and T_{right}, and conversely, given two triangulations T_{left} and T_{right}, we can combine it to a triangulation T. This is the point where property 4 of Definition 12 becomes relevant: if T_{left} and T_{right} satisfy this property, then it will be true for T that the path p is the canonical outgoing path starting at vertex v of Δ.

Thus we count recursively the number of valid triangulations of subproblems $\mathcal{S}_{\text{right}}$ and $\mathcal{S}_{\text{left}}$ and get *exactly* the number of valid triangulations of $\mathcal{S}$ where Δ is the triangle incident to the base edge and p is the canonical outgoing path starting at vertex v of Δ. More



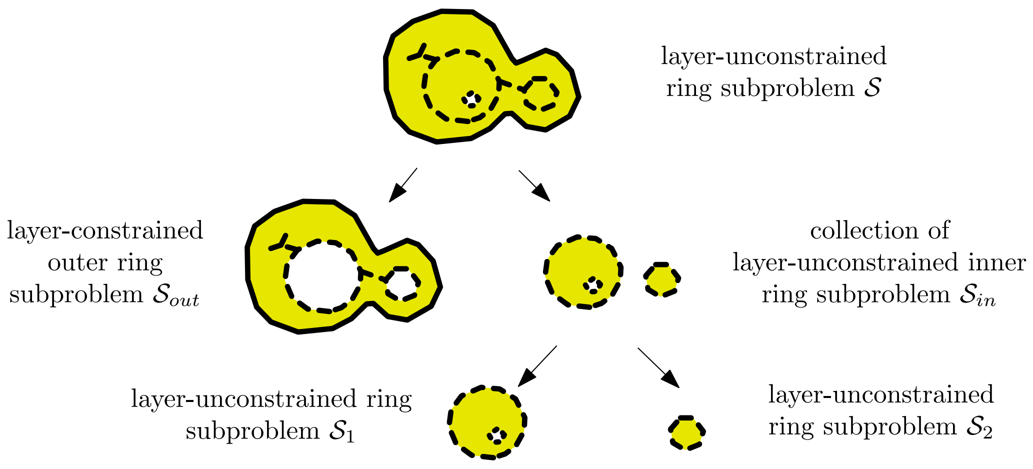
p_1
$\mathcal{S}_{\text{right}}$
Δ
p
p_2
$\mathcal{S}_{\text{left}}$
$\mathcal{S}_{\text{right}}$
$\mathcal{S}_{\text{left}}$
T_{right}
T_{left}

Figure 7 On the top left is a nibbled ring subproblem $\mathcal{S}$ together with a valid triangulation T. There is a unique triangle Δ adjacent to the base edges. From the vertex v of Δ that is not incident to the base edge exists a path p to the outer layer. The triangle Δ and the path p are drawn dashed brown. The path p is uniquely determined if we always use the vertex with the smallest order label among all available choices. On the top right the nibbled ring problem $\mathcal{S}$ is depicted together with the separator path p that splits it into two subproblem $\mathcal{S}_{\text{right}}$ and $\mathcal{S}_{\text{left}}$. At the bottom left, both subproblems are displayed. The three white vertices are shared. At the bottom right the restricted triangulations T_{left} and T_{right} are displayed.

precisely, we sum over all potential canonical outgoing paths p and multiply the number of valid triangulations for $\mathcal{S}_{\text{left}}(p)$ with the number of valid triangulations for $\mathcal{S}_{\text{right}}(p)$.

If there are layer constraints in $\mathcal{S}$, then we have to do some more work. Let d, d_{left}, and d_{right} be the vectors that indicate the size of the layers for T, T_{left}, and T_{right} respectively. Except for the vertices shared by T_{left} and T_{right}, it holds that d equals $d_{\text{left}} + d_{\text{right}}$. Now, let us go back to our subproblems $\mathcal{S}_{\text{left}}$ and $\mathcal{S}_{\text{right}}$. Let c be the layer-constraint vector of $\mathcal{S}$. Then, we define all pairs of *compatible* layer-constraints $(c_{\text{left}}, c_{\text{right}})$ such that two valid triangulations for $\mathcal{S}_{\text{right}}(c_{\text{right}})$ and $\mathcal{S}_{\text{left}}(c_{\text{left}})$ respectively give a triangulation for $\mathcal{S}$ with the correct number of vertices on each layer. Thus for each pair $\mathcal{S}_{\text{left}}$ and $\mathcal{S}_{\text{right}}$ of nibbled ring subproblems we sum over all pairs of compatible layer constraint vectors c_{left} and c_{right} and multiply the number of valid triangulations of $\mathcal{S}_{\text{left}}(c_{\text{left}})$ with the number of valid triangulations of $\mathcal{S}_{\text{right}}(c_{\text{right}})$.

Here, the technical difficulty is to take into consideration the vertices shared by both subproblems. Further we need to ensure that vertices in the i-th layer of $\mathcal{S}_{\text{right}}$ will also be in the i-th layer of $\mathcal{S}$. We recurse on all subproblems occurring in this way.



layer-unconstrained ring subproblem $\mathcal{S}$
layer-constrained outer ring subproblem $\mathcal{S}_{out}$
collection of layer-unconstrained inner ring subproblem $\mathcal{S}_{in}$
layer-unconstrained ring subproblem $\mathcal{S}_1$
layer-unconstrained ring subproblem $\mathcal{S}_2$

Figure 8 On the top is a ring subproblem $\mathcal{S}$ depicted with the outer layer drawn with solid lines, the inner layer in dotted and a separator layer is dashed. In the middle left the outer ring subproblem is depicted and on the middle right is the inner ring subproblem depicted. At the bottom two layer-unconstrained ring subproblems are depicted. The free region is marked yellow in all cases.

Proof sketch (of Theorem 13). The correctness of the algorithm follows from the correctness of the recursion. The bound on the running time follows from bounding the time required to solve a subproblem times the number of subproblems. We save our intermediate results in a search tree in order to prevent to handle any subproblem more than once. The bound on the number of subproblems follows from the fact that all of their items are fixed by at most two path separators, and from the fact that a separator has at most length w, thus there are at most $n^{O(w)}$ of them. The number of layer constraints is bounded by the assumption that at most w layers are constrained. The time for the recursive steps for one subproblem can be asymptotically bounded by the number of recursions, which in turn depends only on the number of potential canonical paths and ways to split the layer constraints in a compatible way. ◀

4 Layer-Unconstrained Ring Subproblems

We sketch the main algorithm in this section and estimate its running time.

The way we solve general ring subproblems is to distinguish two cases. In the case that the ring subproblem is thin, that is, has only few layers ($\leq \sqrt{n}$), we will use the algorithm of Theorem 10 as explained in Section 3. In case that we have many layers ($w > \sqrt{n}$), we know that one of the outermost $\sqrt{n}$ layers must be of size $\leq \sqrt{n}$ by the pigeon hole principle. We use this layer as a separator that splits the problem into a thin outer part and an inner part.

In the course of the algorithm, we will *never* add layer-constraint vectors to ring subproblems with more than $\sqrt{n}$ layers. Thus we do not have to deal in this part of the algorithm with layer-constraint vectors.

We start with a description of the *layer-separators*: To be more explicit, let $\mathcal{S}$ be a ring subproblem and let T be a valid triangulation of $\mathcal{S}$. The width w tells us already the exact number of layers that T has. Consider the case that T has more than $\sqrt{n}$ layers. Then among the $\sqrt{n}$ layers closest to the outer layer, one must have size less than or equal to $\sqrt{n}$. Note that the layer L that is actually closest to the outer layer of $\mathcal{S}$ is *uniquely* determined.

This layer is our separator layer.

The algorithm works as follows: We guess all potential separator layers L, which requires guessing at most $\sqrt{n}$ points and $O(\sqrt{n})$ edges. Each guess defines an inner ring subproblem $\mathcal{S}_{\text{in}}$ and an outer ring subproblem $\mathcal{S}_{\text{out}}$, as depicted in Figure 8. We recurse on all $\mathcal{S}_{\text{in}}$ and $\mathcal{S}_{\text{out}}$ created in this way. To compute the number of valid triangulations of $\mathcal{S}$, we sum over all separator layers L and multiply the number of valid triangulations of $\mathcal{S}_{\text{in}}$ with the number of valid triangulations of $\mathcal{S}_{\text{out}}$, for each appearing L.

In case the cactus layer L is disconnected or has disconnected bounded faces, we deal with a collection of inner ring subproblems. In this case, we recurse on each inner ring subproblem separetly, and multiply the numbers of valid triangulations of each ring subproblem.

We can restrict T to $\mathcal{S}_{\text{out}}$ to attain a triangulation T_{out}. It is clear that all layers different from L and Q_{out} in T_{out} have size larger than $\sqrt{n}$. Therefore, we want to count only those triangulations of $\mathcal{S}_{\text{out}}$ that have all layers (except L and Q_{out}) of size larger than $\sqrt{n}$. We use layer constraints for this purpose: we solve $\mathcal{S}_{\text{out}}$ with every possible layer constraint where every layer is require to have size greater than $\sqrt{n}$. Thus, we sum over all such layer-constraint vector c the number of valid triangulations of $\mathcal{S}_{\text{out}}(c)$. Also note that $\mathcal{S}_{\text{out}}$ has at most $\sqrt{n}$ layers and thus the number of constraint layers are at most $\sqrt{n}$ holds by induction. Note that we do not recurse on the outer ring subproblems by the main algorithm, but rather solve them with the algorithm of Section 3. The runtime is given in Theorem 10.

The running time can be estimated by bounding the total number of ring subproblems times the time spent per ring subproblem. Each layer-unconstrained ring subproblem is defined by an inner and an outer layer. In the course of the algorithm only inner and outer layers of size less than or equal to $\sqrt{n}$ are guessed, and there are at most $n^{O(\sqrt{n})}$ of them. In the recursion step, we either guess all potential layer separators or we deal with rings of width smaller than $\sqrt{n}$. In the first case there are at most $n^{O(\sqrt{n})}$ many guesses. In the second case, the runtime is given by Theorem 10. Thus the total running time is bounded by $n^{O(\sqrt{n})} \cdot n^{O(\sqrt{n})} = n^{O(\sqrt{n})}$.

5 Counting Annotated Triangulations

Here we describe how we adapt the algorithm to also count annotated triangulations with given annotation system L. The adaptation consists of a few straightforward steps. Most prominently, whenever we guess a separator, we guess an annotation for all k_1 vertices and k_2 edges that we guessed. It is easy to see that there are at most $|L|^{k_1+k_2}$ possible guesses. The next step is to check each time we guess a triangle whether the triangle belongs to L. The last step is more subtle. We need to guess also annotations on the outer layer of our initial ring subproblems. However, the initial outer layer is the convex hull, which might contain a linear number of vertices. To circumvent this, we use a standard trick to add a triangle Δ containing the whole point set. This adds some additional constraints on the triangulations we want to count, but circumvents the problem of guessing the annotations of the boundary of the convex hull.

6 Applications for Counting Other Structures

In this section, we develop a framework for counting non-crossing straight line graphs (e.g., non-crossing matchings or non-crossing Hamiltonian cycles), based on our algorithms for counting annotated triangulations. The key idea here is to introduce a new color and extend the graph into a triangulation by filling it with edges of this new color in a *canonical* way.

For these filler-edges, we use the constrained Delaunay triangulation, in a similar fashion as previous authors have done already [2, 4]. We give only a sketch of the argument here.

As a running example of this section, let us consider the problem of counting non-crossing perfect matchings. We show how this problem can be reduced to counting annotated triangulations and then Theorem 3 can be invoked to obtain a subexponential-time algorithm.

▶ **Theorem 14** (Counting Perfect Matchings). *There exists an algorithm that, given a set S of n points in the plane, counts the total number of non-crossing perfect straight line matchings in $n^{O(\sqrt{n})}$ time.*

The obvious idea of reducing counting perfect matchings to counting annotated triangulations is the following. Let us say that the edges appearing in a perfect matchings have red color and let us extend a perfect matching to a triangulation with edges of color blue. It is not very difficult to construct an annotation system enforcing that each edge of the triangulation is either red or blue, and each vertex has exactly one red edge incident to it. One way of doing this would be to annotate each vertex v with the index of the other endpoint v' of its red edge and then to enforce this interpretation by forbidding any triangle incident to v that contains an edge separating v from v'. Now counting triangulations with this annotation system *overcounts* the number of perfect matchings: each perfect matching can be extended into one or more triangulations satisfying the annotation system. Thus we need to further restrict the annotation system in a way that ensures that each perfect matching has exactly one, canonical extension.

Let G be a straight line graph on a set of n points and T be a triangulation extending G on the same set of points. We say that T is the *constrained Delaunay triangulation* of G if every edge $e \in E(T) \setminus E(G)$ satisfies the *Delaunay Condition*, that is, the circumference of neither adjacent triangle contains the other adjacent triangle. (In case that e is on the boundary of the convex hull of S, then we define that the Delaunay Condition is automatically satisfied for e.) It is known that each G has a unique constrained Delaunay triangulation.

▶ **Theorem 15** ([29]). *Given a straight line graph G on a set of n points, there exists exactly one constrained Delaunay triangulation T extending G.*

It is not very difficult to construct an annotation system that enforces that the Delaunay Condition holds for every blue edge. First, we annotate each blue edge with a pair of vertices: a pair containing the third vertex of each of the two triangles incident to the edge. This interpretation is easy to enforce. Then we allow only those annotations where the two triangles of the edge satisfy the Delaunay Condition: neither of them is contained in the circumscribed circle of the other. It is clear that now any valid annotated triangulation has the property that it is the Delaunay triangulation of the red edges and in particular any set of red edges has a unique extension into a valid annotated triangulation.

Combining the two annotation systems, we arrive to an annotation system where the valid annotated triangulations are exactly the Delaunay triangulations of non-crossing perfect matchings. Thus counting the number of annotated triangulations using the algorithm of Theorem 3 counts the number of non-crossing perfect matchings. This proves Theorem 14.

In a similar way, we can construct annotation systems requiring the red edges form other non-crossing structures such as Hamiltonian cycles, spanning trees, quadrangulations, and many more. Combining these annotation systems with the annotation system enforcing the Delaunay Condition on the blue edges, we can construct annotation systems counting these structures as well. This allows us to prove Theorem 6 as stated in the introduction.

References

1 O. Aichholzer. The path of a triangulation. In *SoCG'99*, pages 14–23. ACM, 1999.

2 V. Alvarez, K. Bringmann, R. Curticapean, and S. Ray. Counting triangulations and other crossing-free structures via onion layers. *D&C*, 53(4):675–690, 2015.

3 V. Alvarez, K. Bringmann, and S. Ray. A simple sweep line algorithm for counting triangulations and pseudo-triangulations. *CoRR*, abs/1312.3188, 2013.

4 V. Alvarez, K. Bringmann, S. Ray, and R. Seidel. Counting triangulations and other crossing-free structures approximately. *Comput. Geom.*, 48(5):386–397, 2015.

5 V. Alvarez and R. Seidel. A simple aggregative algorithm for counting triangulations of planar point sets and related problems. In *SoCG'13*, pages 1–8, 2013.

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

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

8 M. Bern and D. Eppstein. Mesh generation and optimal triangulation. *Computing in Euclidean geometry*, 1:23–90, 1992.

9 B. Chazelle. Triangulating a simple polygon in linear time. *D&C*, 6:485–524, 1991.

10 R. H. Chitnis, M. Hajiaghayi, and D. Marx. Tight bounds for Planar Strongly Connected Steiner Subgraph with fixed number of terminals (and extensions). In *SODA*, pages 1782–1801, 2014.

11 M. De Berg, M. Van Kreveld, M. Overmars, and O. C. Schwarzkopf. *Computational geometry*. Springer, 2000.

12 E. D. Demaine, F. V. Fomin, M. T. Hajiaghayi, and D. M. Thilikos. Bidimensional parameters and local treewidth. *SIAM J. Discrete Math.*, 18(3):501–511, 2004.

13 E. D. Demaine, F. V. Fomin, M. T. Hajiaghayi, and D. M. Thilikos. Fixed-parameter algorithms for (k, r)-Center in planar graphs and map graphs. *ACM Transactions on Algorithms*, 1(1):33–47, 2005.

14 E. D. Demaine, F. V. Fomin, M. T. Hajiaghayi, and D. M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *J. ACM*, 52(6):866–893, 2005.

15 E. D. Demaine and M. Hajiaghayi. The bidimensionality theory and its algorithmic applications. *Comput. J.*, 51(3):292–302, 2008.

16 E. D. Demaine and M. Hajiaghayi. Linearity of grid minors in treewidth with applications through bidimensionality. *Combinatorica*, 28(1):19–36, 2008.

17 E. D. Demaine and M. T. Hajiaghayi. Fast algorithms for hard graph problems: Bidimensionality, minors, and local treewidth. In *Graph Drawing*, pages 517–533, 2004.

18 F. Dorn, F. V. Fomin, D. Lokshtanov, V. Raman, and S. Saurabh. Beyond bidimensionality: Parameterized subexponential algorithms on directed graphs. In *STACS*, pages 251–262, 2010.

19 F. Dorn, F. V. Fomin, and D. M. Thilikos. Subexponential parameterized algorithms. *Computer Science Review*, 2(1):29–39, 2008.

20 F. Dorn, E. Penninkx, H. L. Bodlaender, and F. V. Fomin. Efficient exact algorithms on planar graphs: Exploiting sphere cut decompositions. *Algorithmica*, 58(3):790–810, 2010.

21 A. Dumitrescu, B. Gärtner, S. Pedroni, and E. Welzl. Enumerating triangulation paths. *Computational Geometry*, 20(1):3–12, 2001.

22 H. Edelsbrunner and J. Harer. *Computational topology: an introduction.* American Mathematical Soc., 2010.

23 S. Fisk. A short proof of chvátal's watchman theorem. *J. Comb. Theory*, 24(3):374, 1978.

24 F. V. Fomin, D. Lokshtanov, V. Raman, and S. Saurabh. Subexponential algorithms for partial cover problems. *Inf. Process. Lett.*, 111(16):814–818, 2011.

25 F. V. Fomin and D. M. Thilikos. Dominating sets in planar graphs: Branch-width and exponential speed-up. *SIAM J. Comput.*, 36(2):281–309, 2006.

26 P. D. Gilbert. New results on planar triangulations. Technical report, 1979.

27 L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209–233, 1987.

28 J. Hershberger and S. Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *Journal of Algorithms*, 18(3):403–431, 1995.

29 Ø. Hjelle and M. Dæhlen. *Triangulations and Applications*. Springer, 2006.

30 M. Karpinski, A. Lingas, and D. Sledneu. A QPTAS for the base of the number of crossing-free structures on a planar point set. In *ICALP'15*, pages 785–796, 2015.

31 P. N. Klein and D. Marx. Solving Planar k-Terminal Cut in $O(n^{c\sqrt{k}})$ time. In *ICALP (1)*, pages 569–580, 2012.

32 P. N. Klein and D. Marx. A subexponential parameterized algorithm for Subset TSP on planar graphs. In *SODA*, pages 1812–1830, 2014.

33 G. Klincsek. Minimal triangulations of polygonal domains. *Ann. Discrete Math*, 9:121–123, 1980.

34 C. Knauer and A. Spillner. A fixed-parameter algorithm for the minimum weight triangulation problem based on small graph separators. In *WG'06*, pages 49–57, 2006.

35 A. Lingas. Subexponential-time algorithms for minimum weight triangulations and related problems. In *CCCG'98*, 1998.

36 R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

37 D. Lokshtanov, D. Marx, and S. Saurabh. Lower bounds based on the Exponential Time Hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011.

38 D. Marx and T. Miltzow. Peeling and nibbling the cactus: Subexponential-time algorithms for counting triangulations and related problems. *CoRR*, to be announced, 2016.

39 D. Marx and M. Pilipczuk. Optimal parameterized algorithms for planar facility location problems using voronoi diagrams. In *ESA'15*, pages 865–877. Springer, 2015.

40 M. Pilipczuk, M. Pilipczuk, P. Sankowski, and E. J. van Leeuwen. Network sparsification for steiner problems on planar and bounded-genus graphs. In *FOCS'14*, pages 276–285.

41 M. Pilipczuk, M. Pilipczuk, P. Sankowski, and E. J. van Leeuwen. Subexponential-time parameterized algorithm for Steiner Tree on planar graphs. In *STACS*, pages 353–364, 2013.

42 S. Ray and R. Seidel. A simple and less slow method for counting triangulations and for related problems. In *EuroCG'04*, pages 77–80, 2004.

43 M. Sharir and A. Sheffer. Counting triangulations of planar point sets. *Electr. J. Comb.*, 18(1), 2011.

44 D. M. Thilikos. Fast sub-exponential algorithms and compactness in planar graphs. In *ESA*, pages 358–369, 2011.