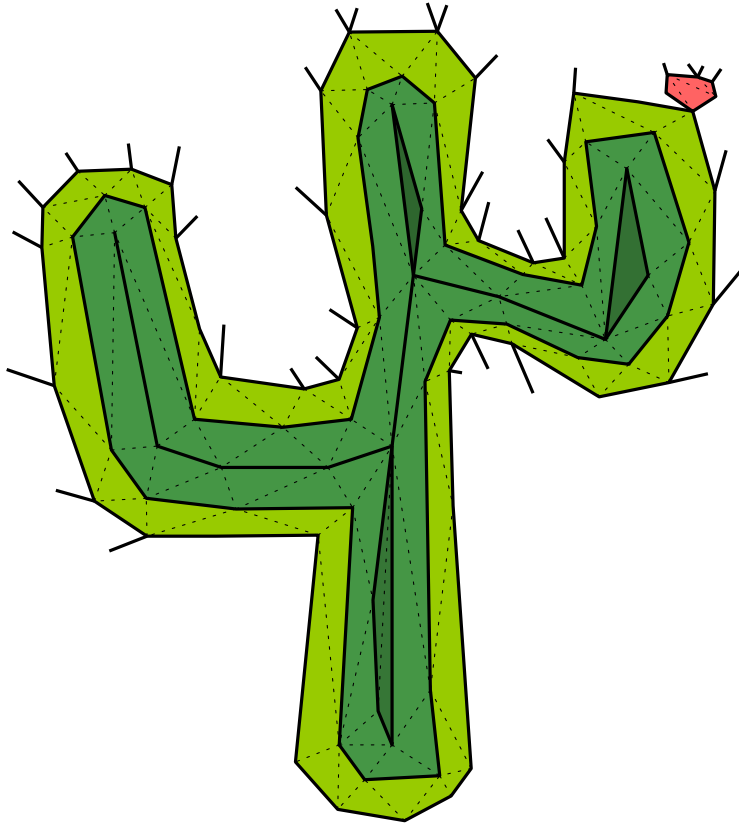# Peeling and Nibbling the Cactus: Subexponential-Time Algorithms for Counting Triangulations and Related Problems.

Dániel Marx   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.

1 Introduction

Given a set 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 [9, 10, 14, 26, 27, 31, 32, 52]. 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 [2–8, 25, 30, 34, 37, 51]. The sequence of improvements culminated in the $O(2^n n^2)$ time algorithm of Alvarez and Seidel [6], winning the best paper award at SoCG 2013. Our main result significantly improves the running time of counting triangulations by making it subexponential:

▶ **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 [43]. 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) [12, 15–20, 22–24, 28, 29, 35, 36, 45, 49, 50, 53].

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 [38, 39]. 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 [40]. The basic idea is to guess a correct $O(\sqrt{n})$-sized separator of a minimum weight triangulation and recurse one all occurring subproblems. 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 [38, 39].

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 full version is committed to give all details in a self-contained way. The reader who is only interested in the main concepts of the algorithm is refered to the conferenc version [44].

1.1 Preliminaries

We interpret a collection of points S and non-crossing segments E in $\mathbb{R}^2$ as a plane graph, if every segment $e \in E$ shares exactly its endpoints with S. We usually identify points with vertices and edges with segments. For convenience, we will use the term graph almost always, to indicate plane straight line graphs and silently assume we mean a plane straight line graph. The number n always denotes the size of the underlying point set S. We assume S to be in general position, that is, no three points lie on a line and no four points lie on a common circle.

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 the *outer face*. We call an edge *inner edge* of G if it is not on the outer face of G. A cactus graph is outerplane, but some outerplane graphs are not cacti, as they might have inner edges. 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 v uniquely the *index*, denoted by layer-index(v), of the layer it is contained in.

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.
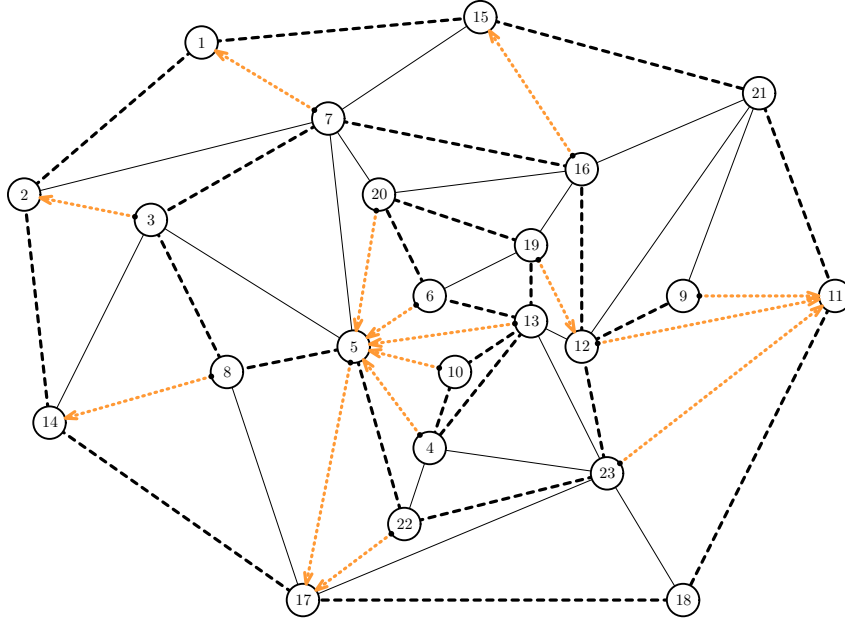
Figure 1 A set of points, each with a distinct *order label*, and a triangulation. The edges of layers are dashed. For each vertex, we drew a directed edge to the unique neighbor with smallest order label, among the vertices with smaller distance to the boundary.

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.

We attach to each point $p \in S$ a different number $l(p) \in \{1, \ldots, n\}$ and refer to this number as the *order label*.

▶ **Lemma 2.** *Given a valid triangulation T of some set S of n points in the plane. Then each vertex v with* layer-index$(v) > 1$ *has a neighbor w with* layer-index$(w) = $ layer-index$(v) - 1$, *vertex v has no neighbor w with* layer-index$(w) < $ layer-index$(v) - 1$. *Among the neighbors w as described above, there is exactly one with smallest order label.*

Proof. Consider a vertex v with $i = $ layer-index$(v) > 1$. We denote by $w_1, \ldots, w_k$ the neighbors of v in cyclic order. We will show one of the neighbors has layer-index layer-index$(v) - 1$. We know after layer-index$(v) - 1$ iterations of removal of vertices on the outer face of G, v is on the outer face of G_i. This implies one of the faces adjacent to v becomes part of the outer face of G_i. This implies one of the edges (w_j, w_{j+1}) is removed (indices taken modulo k.). This implies w_j or w_{j+1} was removed from G_{i-1}. The index of a removed vertex is by definition $i - 1$. This shows the first claim.

To see the second claim, assume there exists a vertex w with the above mentioned index. After the removal of w the vertex v is on the outside face and thus layer-index$(v) \leq$ layer-index$(w) + 1 < $ layer-index$(v) - 1 + 1 = $ layer-index(v) — a contradiction. ◀

In light of Lemma 2, we can define layers alternatively using the distance to the vertices on the outer face. For this purpose, define an artificial vertex v_∞ adjacent to all vertices on the outer face. Let $d(v)$ denote the *distance of v to v_∞*, that is the length of the shortest path to
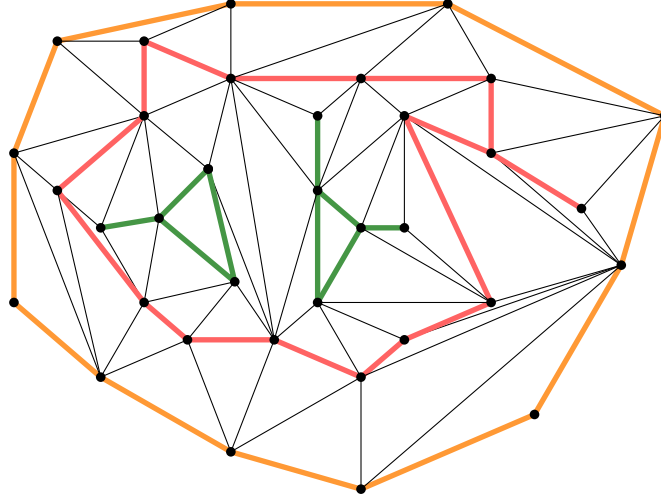
Figure 2 A triangulation with three cactus layers. Layer 1 is colored orange, Layer two is colored red and Layer 3 is colored green. Note that the layers do not need to be connected.

v_∞. By Lemma 2, $d(v) = $ layer-index(v), and layer i has as vertices $V(L_i) = \{v : d(v) = i\}$. Note that the graph induced on $V(L_i)$ is *not* layer i. Here we use the convention that the length of a path equals the number of its edges.

▶ **Definition 3.** 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. We say a triangle is *feasible*, if it belongs to the list. The size of L is the number of triangles it contains and denoted by $|L|$.

Given an annotation system L, we call the T a *valid annotated triangulation* of S if for every annotated triangle Δ of T is feasible($\Delta \in L$). Further $\mathcal{T}^A(L)$ denotes the set of all valid annotated triangulations belonging to L.

We denote with $[a, b]$ the *integer interval* $\{a, a+1, \ldots, b\}$.

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 the interior 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 use the so-called ring subproblems for the layer separators and nibbled rind subproblems for the path separators.

As a byproduct of this algorithmic scheme, we can efficiently count triangulation with a small number of layers. This is similar to previous work on finding a minimum weight triangulation [7] and counting triangulations [3] for point sets with a small number of onion layers.
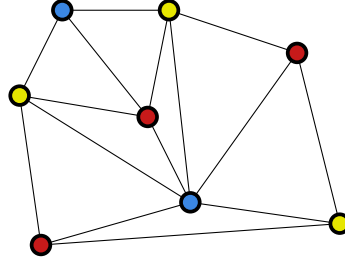
■ **Figure 3** A three colorable triangulation.

▶ **Theorem 4** (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. 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 5** (Counting Annotated Triangulations). *Given an annotation system L and a set S of n points in the plane, we can count all valid annotated triangulation in $n^{(11+o(1))\sqrt{n}} \cdot |L|^{(12+o(1))\sqrt{n}}$ time.*

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

▶ **Theorem 7.** *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 $n^{O(\sqrt{n})}$ time.*

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 think of a geometric graph on a point set as a 2-edge-colored triangulation, with one color forming the graph itself and the other color representing non-edges. 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 nontrivial, as a given geometric graph can be extended into a a 2-edge-colored triangulation in many different ways. Similarly to previous work [3, 5], we use the notion of constrained Delaunay triangulation (see [33]) 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.
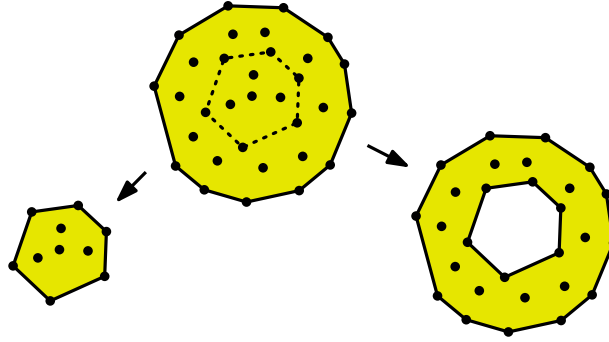
■ **Figure 4** Cactus layes are ideal separators, as they separate in a simple way the inside from the outside and we can easily *peel off* large layers from the outside.

The following theorem states some important examples to demonstrate the applicability of our approach.

▶ **Theorem 8** (Counting Geometric Structures)**.** *The following non-crossing structures can be counted in $n^{O(\sqrt{n})}$ time 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 8 uses the algorithm of Theorem 5 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 5. 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 Key Ideas

Our algorithm is based on separators similar to almost all previous algorithms. There are two major differences: the separators are defined not in terms of the input (point set), but in terms of the output (triangulation). This gives us higher flexibility so that we are not restricted to use one kind of separator, but are able to design an algorithm scheme with *two* types of separators.

Given a triangulation with outerplanar index $k > \sqrt{n}$, there exists a cactus layer of size at most $\sqrt{n}$ by the pigeonhole principal, see Figure 4. This cactus layer is an ideal candidate for a separator. As cactus 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 5. (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 [7] demonstrated this for finding a minimum weight triangulation, using onion layers instead of cactus layers, but the algorithmic idea is essentially the same. Alvarez, Bringmann, Curticapean and Ray showed how to make these separators canonical
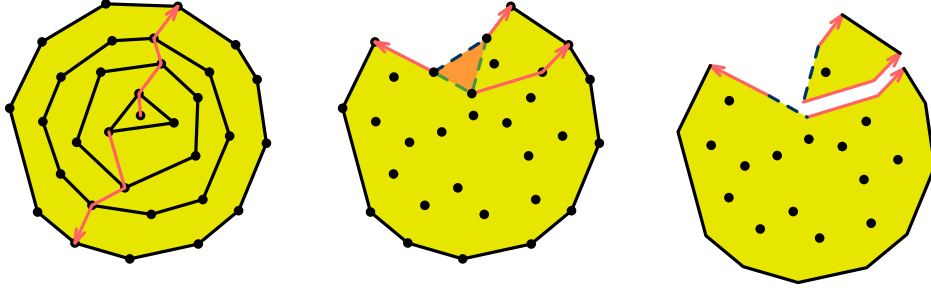
Figure 5 Paths from the interior to the outside are short separators, if the triangulation does not have too many layers. The resulting dynamic programming scheme is a little unintuitive, as it *nibbles off* the algorithmic problem from the outside.

and thus suitable for counting problems [3], 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 path separators in an, at first, unintuitive way. We fix a special edge $e = uv$ on $\partial CH(S)$ and guess all triangles $\Delta = \Delta(u, v, w)$ and a short path 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.

1.4 Related Work

year	contribution	idea	cite
1979	polynomial time algorithm for polygons	divide and conquer	[30, 37]
1993	$n^{O(h)}$ MWT	use short legal paths	[7]
1996	$O(t(P)\text{poly}(n))$ enumeration algorithm	reverse search technique	[8]
1999	$O(tr(P)^2)$ time counting algorithm	D&C + triangulation paths	[2]
2001	triangulation path enumeration + lower bound	reverse search	[25]
2004	empirically fast algorithm	generalize polygon algorithm	[51]
2012	$n^{O(h)}$ time algorithm	legal paths unique using labels	[3]
2013	$O(tr(P))$ time algorithm triangulation path upper bound	sweep instead of divide and conquer	[4]
2013	$n^{O(\sqrt{n})}$ time algorithm with $n^{\left(n^{4/3}\right)}$ approximation-ratio	guess $O(\sqrt{n})$ sized separators solve small subproblmes exact	[5]
2013	$O(2^n n^2)$ time algorithm	sweep using x-monotone curves	[6]
2015	QPTAS	techniques of Adamszek & Wiese	[34]

Figure 6 Summary of important algorithms to count triangulations.

Many authors have worked on finding efficient algorithms for counting all triangulations on a given set of points. Some algorithms have been developed with the purpose of finding a minimum weight triangulation, but it turned out that they can be used or at least adapted to also count triangulations.

We found it very interesting to observe how ideas used to compute a minimum weight triangulation can actually be used to count triangulations instead. Here, we survey the relevant work on counting and we also try to point out where these ideas have been used before to compute the minimum weight triangulation.

In 1979, Peter D. Gilbert showed in his master thesis [30] how to compute a minimum weight triangulation of a simple polygon P using divide and conquer. In 1980, Klincsek [37] showed the same result independently. The idea of the divide and conquer scheme is based on the observation that any edge e of any triangulation has a triangle Δ adjacent to e. The algorithm guesses all potential Δs and recurses on the two arising subproblems. The number of occuring subproblems is bounded by n^2 and on each a linear number of recursion steps is needed. Thus the running time is $O(n^3)$. It is easy to see that this algorithm can be used as well to count triangulations.

Only much later, in 2004, Saurabh Ray and Raymund Seidel showed how to use this idea as an algorithm for counting triangulations for points in the plane [51]. While their algorithm seems to be reasonable fast in practice, they did not supply a run time analysis. However, they observed a runtime of $\sqrt{t(S)}$ empirically, where $t(S)$ is the total number of triangulations of S, a set of n points in the plane.

In 1996, the first enumeration algorithm for counting triangulations was published by David Avis and Komei Fukuda [8]. They developed the so called *reverse search technique* and applied it to numerous problems. The runtime of the enumeration algorithm on a set of points S is $t(S) n^{O(1)}$. To see how it works for enumerating triangulations note that the *flip graph* G of the triangulations is connected. Let Δ_1 and Δ_2 be two adjacent triangles and assume that their union $\square$ is convex. Then a *flip* is a replacement of the shared edge by the other diagonal of $\square$. The vertices of the flip graph correspond to the set of triangulations and two triangulations are adjacent if one can be attained from the other by a flip. The strategy of the reverse search technique is to identify a rooted spanning tree $\tau \subseteq G$ and traverse it. The root of τ is the Delaunay triangulation of the underlying point set. It is known that any sequence of *Lawson-flips* will turn any triangulation to the Delaunay triangulation. This defines an acyclic graph with the Delaunay triangulation as unique sink. To make the outdegree of each triangulation 1, we give an order to all potential edges and we always flip the Lawon-edge with highest priority. This defines τ. In order to traverse the tree in a depth first manner, the constrained Delaunay triangulation is needed to keep track of edges, which we do not want to flip.

In 1993 Efthymios Anagnostou and Derek Corneil presented an algorithm to compute the minimum weight triangulation of a given point set in $O(n^{3h+1})$ time [7]. Here h denotes the number of *onion layer*s of the given point set. The idea of their algorithm is to use *legal paths* from the most inner onion layer to the boundary of the convex hull. The paths are required to visit each layer at most once. These paths can be used as separators. It is easily seen that there are at most $O(n^h)$ many such paths. Each subproblem in their dynamic programming approach is defined by two legal paths and they use one legal path to split their subproblem. This yields the bound on the running time. Ketan Dalal [13] showed in 2004 that the expected number of onion layers is $\Theta(n^{2/3})$. This makes their algorithm subexponential for random point sets on average.

19 years later in 2012 Victor Alvarez, Karl Bringmann, Radu Curticapean and Saurabh

Ray presented an algorithm with the same idea, which could also count the number of triangulations [3]. The problem of the previous algorithm by Anagnostou and Corneil was, that it might potentially over count. To prevent this, they need to make sure that for any vertex and any triangulation a unique legal path is defined. To do this they gave every vertex a label and always choose the vertex with smallest label to extend their legal path. With a refined analysis they could show an upper bound of $O(c^n)$ in the worst-case, with $c \approx 3.1414$.

In 1999 Oswin Aichholzer attacked the problem of counting triangulations from a different angle by introducing the concept of a *triangulation paths* [2]. Given a triangulation T and a line ℓ intersecting this triangulation a triangulation path is a sequence of segments of T twining around ℓ with some additional technical conditions that we do not mention here. The remarkable property of the triangulation path is that it is unique and thus eligible to be used as a potential separator for divide & conquer. The algorithm guesses all potential paths and recurses on all arising subproblems. We denote by $tr(S)$ the number of all potential triangulation paths of S and by $tr(n) = \max_{|P|=n} tr(P)$. Aichholzer showed that the running time $T(n)$ adheres the recursion $T(n) = O(tr(n)T(n/2))$ and thus solves to $O(tr(n)^2)$. Thus the running time of Aichholzers algorithm is bounded by $O(tr(n)^2)$.

In 2001, Adrian Dumitrescu, Bernd Gärtner, Samuele Pedroni and Emo Welzl reduced the hope that triangulation paths can be used for efficient algorithms [25]. They provide a simple lower bound example showing $tr(n) \geq 4^{n-\Theta(\log n)}$. On the positive side they apply the reverse search technique to enumerate all triangulation paths efficiently. They used again the constrained Delaunay triangulations.

Despite hope being diminished, in 2013, Victor Alvarez, Karl Bringmann, and Saurabh Ray presented an algorithm that could count triangulations in $O(tr(n))$ [4]. This is a great improvement over Aichholzers algorithm. Their main idea is to use *sweeping* instead of divide and conquer. This idea is actually more interesting than the running time, as it was later employed for a more efficient algorithm. Their algorithm is technically non-trivial as it demands structural insight to triangulation paths in order to identify all potential successors of a triangulation path. They also provide an upper bound of $O(9^n)$ on $tr(n)$.

All these algorithms were subsumed in 2013 by Victor Alvarez and Raimund Seidel [6], winning the best paper award on the Symposium of Computational Geometry. Their algorithm has a worst-case running time of $O(2^n n^2)$. Remarkable is their algorithm, because it was the first algorithm that provably always counted triangulations faster than the number of triangulations itself. Further the algorithm is simple. Their idea is to use x-monotone curves as separators together with sweeping. It is easy to see that the number of x-monotone curves on S is bounded by 2^n. Each x-monotone curve c defines the subproblem of computing the number of triangulations below c. The interaction between subproblems is elegantly encoded into an algebraic shortest path problem.

In 2013 Victor Alvarez, Karl Bringmann, Saurabh Ray and Raimund Seidel presented a simple approximation algorithm [5]. The running time of their algorithm is $n^{O(\sqrt{n})}$. It has an approximation factor of $n^{O(n^{4/3})}$. This is huge but as our aim is to approximate a number of exponential size it might be not too bad actually. The idea is to use simple cycle separators originally from G.L. Miller [46, 47] and later improved by H.N. Djidjev and S.M. Venkatesan [21]. of size $O(\sqrt{n})$. Here any triangulation can be counted many times, but the total over counting can be bounded because after a certain number of rounds the arising subproblems are so small, that they can be solved fast by an exact exponential time algorithm. In a recursive call of the algorithm all potential $n^{O(\sqrt{n})}$ separators are guessed. As there is no unique balanced separator for each triangulation every triangulation might be

counted at most $n^{O(\sqrt{n})}$ times. To bound the approximation factor, the algorithm computes the number of triangulations exactly as soon as the subproblems get small enough. This algorithm is also able to compute the minimum weight triangulation and has roughly the same runtime as previous algorithms in the worst case [38, 39].

In 2015, an approximation scheme was presented by Marek Karpinski, Andrzej Lingas, and Dzmitry Sledneu [34]. Their running time is quasi-polynomial that is $2^{(\log n/\varepsilon)^{O(1)}}$. The approximation factor is of the form $2^{\varepsilon n}$. Thus, this algorithm has a better running time, but a worse approximation factor. They employ the technique developed by Adamszek and Wiese [1]. The technique main idea is to use separators, that are very small and thus all of these separators can be guessed in quasi polynomial time. The separators have the disadvantage that certain solutions are not accounted for.

A similarity of almost all algorithms is that they use some kind of separator for a dynamic programming/divide & conquer scheme. In 1979, Richard J. Lipton and Robert Endre Tarjan described in their seminal paper a simple algorithmic paradigm that has been employed countless times [40, 41]: "Three things are necessary for the success and efficiency of divide-and-conquer: (i) the subproblems must be of the same type as the original and independent of each other (in a suitable sense); (ii) the cost of solving the original problem given the solutions to the subproblems must be small; and (iii) the subproblems must be significantly smaller than the original." Let us call this the Lipton-Tarjan-paradigm. It turns out that (iii) can be replaced by another property. Namely, that we can bound all potential subproblems that can occur. Denote by $s(n)$ the number of all potential separators. If each subproblem is defined by at most c separators, the number of potential subproblems is bounded by $s(n)^c$ and the running time is bounded by $T(n) \leq s(n)^{c+1}$. This strategy requires that each subproblem is defined by at most a constant number of separators. It goes particularly well with sweeping as every subproblem is defined by only one separator.

The algorithm in [2, 5] follow the Lipton-Tarjan-paradigm; the algorithms in [3, 4, 6, 7, 30, 37] follow the second approach.

Regarding the separators used for the exact algorithms, we want to mention that they are designed in a way that each triangulation admits exactly one such separator. To guarantee uniqueness often additional work is required. To illustrate the importance of uniqueness, recall that this was the main property of triangulation paths proved by Aichholzer [2]. We cannot think of a principal reason, why it should not be possible that any kind of separator can be made unique.

Most algorithms that are able to count triangulations can also be adopted to count other kind of geometric structures. Studied structures are the set of all potential geometric graphs, perfect matchings, spanning trees, cycle partitions, convex partitions and spanning cycles. This was shown for spanning trees, perfect matchings and spanning cycles by Alvarez, Bringmann, Curticapean and Ray in [3] and again by Alvarez, Bringmann and Ray [5]. In 2014, Manuel Wettstein showed how to adapt the algorithm by Alvarez and Seidel to count all the structures mentioned above, see [54, 55]

The basic strategy is to use annotations on the separators. These annotations give additional information on the kind of structures to be counted. The approach of Wettstein uses the specific structure of the separators of the algorithm by Alvarez and Seidel, namely being x-monotone curves. The approach by the other group of authors is more general and can be applied more broadly. The first idea is that each geometric structure $\mathcal{X}$ is in one to one correspondence to the constrained Delaunay triangulations containing $\mathcal{X}$. Thus it is sufficient to count those constrained Delaunay triangulations. In order to be able to guarantee that two constrained Delaunay triangulations compose to one, the separators are
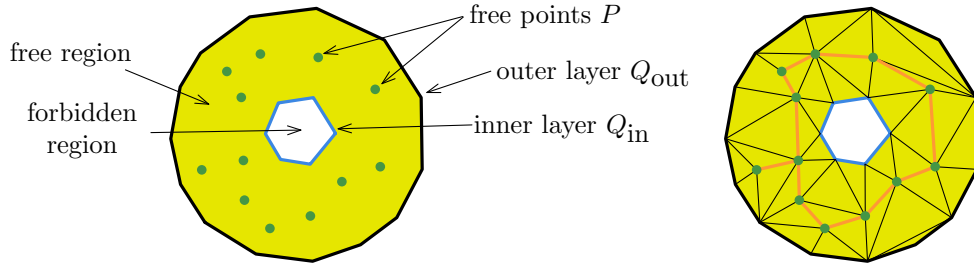
Figure 7 A simple layer-unconstrained ring subproblem and a valid triangulation. The width of the ring subproblem is three and the triangulation has three cactus layers.

made 'fat'.

Up to date there exist no lower bounds on the counting problems mentioned so far. However, as all the afore mentioned algorithms are also able to solve 'decomposable' problems, we review lower bounds for them.

The most prominent decomposable problem is that of finding a minimum weight triangulation. It was a major break-through in 2006, when Wolfgang Mulzer and Günter Rote presented their, by now famous, NP-hardness proof [48]. Their proof needed computer assistance, to check that wires and other gadgets worked as intended. The major insight is that the β-skeleton is always part of a minimum weight triangulation, for certain values of β. With this in mind, it is possible to construct point sets that enforce certain edges to form 'tunnels' and 'walls'. Eventually, this becomes complex enough to build suitable gadgets, to encode any planar positive 1-In-3-SAT formula.

Another, interesting problem is that of computing a triangulation in case that only a subset of the edges are eligible and the others are forbidden. It was shown by Errol Lynn Lloyd in 1977 that this problem is NP-hard [42]. This result was strengthened to #W[2]-hardness, by Alvarez, Bringmann, Curticapean and Ray in case that the input point set consist of k onion-layers [3].

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.

In this section we present the definition of the ring subproblems used by the algorithm and show how an algorithm that can solve those problems implies Theorem 1 and Theorem 5 for counting triangulations and annotated triangulations respectively.

The following definition formally describes ring subproblems, which serve as the basis of the dynamic programming, see Figure 7 for an illustration.

▶ **Definition 9** (Ring Subproblems). A *ring subproblem* $\mathcal{S}$ consists of the following:

outer layer: The *outer layer* is the non-crossing union of some simple polygons $Q_{\text{out}} = Q_{\text{out}}(\mathcal{S}) = (q_1, \ldots, q_a)$. For clarification, $q_1, \ldots, q_a$ are a different simple polygons. And their union forms a cactus graph. This is: we require that all edges of Q_{out} are incident to the outer face of Q_{out}. We will call $\mathcal{S}$ *simple* if Q_{out} consists of only one polygon.

inner layer: The *inner layer* $Q_{\text{in}} = Q_{\text{in}}(\mathcal{S})$ is a cactus contained in Q_{out}. We allow the inner layer to be empty.

inner/outer layer index and width: The *inner(outer) layer index* is some positive integer denoted by in-index($\mathcal{S}$) (resp., out-index($\mathcal{S}$)) and associated with the inner (resp., outer) layer. The width of a ring subproblem is defined as $w = w(\mathcal{S}) = \text{in-index}(\mathcal{S}) - \text{out-index}(\mathcal{S}) + 1$. The meaning of the width is a little tricky. In case that the inner layer is empty it indicates an *upper* bound on the number of layers to be inserted. Otherwise, if the inner layer is non-empty it gives the *precise* number of layers.

free region: The region inside Q_{out} excluding the bounded faces of Q_{in} is the *free region*.

free points: A set of points P in the interior of the free region.

layer-constraints: This is just a vector $c = (i_1, \ldots, i_n)$ of length n, with its entries $i_j \in \{0, 1, 2, 3, \ldots, n\}$. We refer to c as the *layer-constraint vector* of $\mathcal{S}$. The j-th entry of c, denoted by $c(j) = i_j$, indicates that the j-th layer should have exactly i_j vertices. We assume that $c(j) = 0$, for all $j \notin [\text{out-index}(\mathcal{S}), \text{in-index}(\mathcal{S})]$. (Here, n denotes the size of the underlying point set.)

If no layer-constraint vector is specified, we speak of the *layer-unconstrained* ring subproblem. Otherwise, we speak of a *layer-constrained* ring subproblem. Given a layer-constraint vector c and a layer-unconstrained ring subproblem $\mathcal{S}$, we define $\mathcal{S}(c)$ as the layer-constrained ring subproblem appended with the layer-constraint vector c.

For the more general algorithm that is also able to count annotated triangulations, additional informations need to be maintained.

boundary annotation: Some string s on each vertex and edge of the inner and outer layer.

annotation system: An *annotated* triangle is a 9-tuple consisting of 3 points, 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.

The size $|L|$ of the annotation system L is defined as the total number of annotated triangles. We assume that the length of each string is in $n^{O(1)}$.

▶ **Definition 10** (Valid Triangulation). Given a ring subproblem $\mathcal{S}$, consider a graph T extending the graph formed by $Q_{\text{in}} \cup Q_{\text{out}} \cup P$. The graph T can be decomposed into cactus layers L_i as explained in Section 1.1. Here, we slightly change the indexing, by requiring that the first layer L_j has index $j = \text{out-index}(\mathcal{S})$ and the second layer has index $\text{out-index}(\mathcal{S}) + 1$ and so on. We denote by $d(v)$ the index of each vertex defined in this way. We call such a graph T of $\mathcal{S}$ a *valid triangulation of $\mathcal{S}$* if the following conditions are satisfied:

1. All faces in the free region are triangles and there are no edges outside the free region.
2. The graph L_j with $j = \text{in-index}(\mathcal{S})$ is the inner layer Q_{in}.
3. All layer-constraints on the layers are satisfied, that is $|V(L_i)| = c(i)$.

We drop the last condition in case that $\mathcal{S}$ is a layer-unconstrained ring subproblem, that is, $\mathcal{S}$ has no layer-constraints.

Furthermore, a triangulation comes together with an annotations of the edges and the vertices.

4. Each empty annotated triangle Δ of the triangulation T is in the annotation system L.

Condition 2 implies that there are exactly w distinct layers in case that the inner layer is non-empty, because of the way we defined the indexing. In case that the inner layer is empty this condition merely implies that there are at most w non-empty layers.

We denote by $t(\mathcal{S})$ the number of valid triangulations of $\mathcal{S}$.
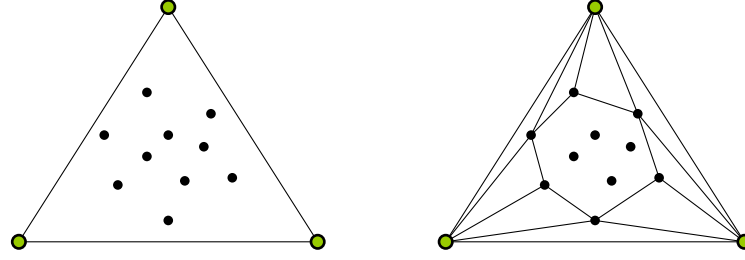
■ **Figure 8** left: The point set S in black and three extra points in green around
it. Right: Triangulation T^* of the part outside $CH(S)$.

▶ **Theorem 11.** *There exists an algorithm that, given an annotated layer-unconstrained ring
subproblem $\mathcal{S}$ on n vertices and an annotation system L, computes the number of all valid
triangulations of $\mathcal{S}$ in $n^{(11+o(1))\sqrt{n}} \cdot |L|^{(12+o(1))\sqrt{n}}$ time.*

To prove Theorem 5 using Theorem 11, we start with a ring subproblem where the inner
layer is empty and the outer layer is the convex hull of the point set. There is a technical
issue here: the definition of the ring subproblem requires a fixed annotation on the outer
layer. This means that we can use the algorithm of Theorem 11 only if we try all possible
annotations on the boundary, which could be a prohibitively large number of possibilities.
Therefore, we use the standard trick of extending the point set to make the convex hull a
triangle.

Proof of Theorem 5 by Theorem 11. We define a ring subproblem $\mathcal{O}$ such that the trian-
gulations of S and $\mathcal{O}$ stay in one to one correspondence. Consider 3 points forming a triangle
Δ^+ containing S, see Figure 8. The point set S^+ is defined as $\Delta^+ \cup S$.

We define another annotation system L^* as follows. Fix a triangulation T^* of the region
enclosed by Δ^+ and $\partial CH(S)$. Given an edge $e = vw$ of $\partial CH(S)$ and a feasible triangle
$\Delta \in L$, with $e \in \Delta$. Let a_v, a_w and a_e be the annotation of v, w and e of Δ respectively.
We define Δ^* as the triangle of T^* incident to e annotated with a_v, a_w and a_e for v, w
and e, and empty strings otherwise. To attain L^*, we add all triangles of Δ^* to L, as
above. Recall that $\mathcal{T}^A(L^*)$ is the set of annotated triangulations with respect to L^*. It
holds $|\mathcal{T}^A(L)| = |\mathcal{T}^A(L^*)|$. With this set up, we are ready to use Theorem 11 by defining
an appropriate annotated layer-unconstrained ring subproblem $\mathcal{O}$. Then the algorithm to
count all annotated triangulations of S is to count all triangulations of $\mathcal{O}$. This takes
$n^{O(\sqrt{n})} \cdot |L^*|^{O(\sqrt{n})}$ time. From $|L^*| \leq 2|L|$ follows the running time bound as stated in the
theorem. We define $\mathcal{O}$ as follows.

outer layer: $Q_{\text{out}} = \Delta^+$ annotated with empty strings.
inner layer: The inner layer Q_{in} is empty.
free points: The free points P are exactly S.
inner/outer layer index out-index$(\mathcal{O}) = 0$ and in-index$(\mathcal{O}) = n + 1$.
annotation system: L^*

Given a set of points S in the plane, it is easy to verify that $\mathcal{O}$ is a well defined simple
annotated layer-unconstrained ring subproblem. Any annotated triangulation T of S can be
extended by T^* to a triangulation T_{ring} of $\mathcal{O}$. We need to check the Conditions 1, 2 and 4 of
Definition 10 to confirm that T_{ring} is indeed a *valid* triangulation. Condition 1 merely states
that T_{ring} is a triangulation. Condition 2 states that the $(n+1)$-st layer of T_{ring} is supposed
to be empty. This is true as no triangulation has more than $(n/3 + 1)$ cactus layers. Note

that we don't have to check Condition 3 as we have *not* specified a layer-constraint. For Condition 4, we have to show that each empty triangle of T_{ring} belongs to L^*. This is the case for the triangles outside $\partial CH(S)$ by the definition of L^* and T_{ring}. For each triangle inside $\partial CH(S)$, it follows from the fact that it was true for T and L.

It is straight-forward that two different triangulations T and T' of S induce indeed two different triangulations T_{ring} and T'_{ring} of $\mathcal{O}$, even if T and T' differ only by their annotations.

At last, we need to show that every valid triangulation T_{ring} of $\mathcal{O}$ comes from some triangulation T of S. The triangulation T comes from restricting T_{ring} to S.   ◄

We can easily derive Theorem 1 from Theorem 5.

Proof of Theorem 1 by Theorem 5. Choosing L to be the set of all empty triangles of S annotated by empty strings yields the claim. Every triangle of every triangulation of S is contained in L by definition and thus a valid *annotated* triangulation. Given two annotated triangulations T_1 and T_2. If T_1 and T_2 are different as annotated triangulations, then they are also different as *plain* triangulations, as every vertex and edge is annotated with the same empty string.

To see that the constant 11 is achievable, it is necessary to observe that the algorithm presented in this paper also works if it deals without annotation systems. In this case the multiplicative factor $|L|^{(7+o(1))\sqrt{n}}$ is omitted.   ◄

3   Thin Rings

This section presents 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 $w \le \sqrt{n}$.

▶ **Theorem 12.** *There exists an algorithm that given a simple (layer-constrained or layer-unconstrained) ring subproblem $\mathcal{S}$ with width w and n free points, computes the number of all valid annotated triangulations of $\mathcal{S}$ in time $n^{(5+o(1))w} \cdot |L|^{(6+o(1))w}$.*

Theorem 12 implies Theorem 4 in a similar fashion as Theorem 11 implies Theorem 5.

Proof of Theorem 4 via Theorem 12. We describe an algorithm that computes the number $t_{\le}(S, k)$ of all valid triangulations with outerplanar index at most k. We can compute the number $t_{=}(S, k)$ of valid triangulations with outerplanar index exactly k, by $t_{=}(S, k) = t_{\le}(S, k) - t_{\le}(S, k-1)$.

We define a layer-unconstrained ring subproblem $\mathcal{O}$ such that valid triangulation of $\mathcal{O}$ with outerplanar index k and triangulations of S with outerplanar index k are in one to one correspondence. We describe each component of $\mathcal{O}$ explicitly. Then we just compute the number of valid triangulations of $\mathcal{O}$ using the RingSec algorithm, as stated in Theorem 12.

outer layer: The outer layer Q_{out} is the boundary of the convex hull of S. The *outer layer index* equals one.

inner layer: the *inner layer* Q_{in} is empty, and its *inner layer index* is $k + 1$.

free points: the *free points* P are the points of S not on $\partial CH(S)$.

boundary annotations: Each edge and vertex of the inner layer, the outer layer, the boundary paths and the base edge is annotated with the empty string.

annotation system: all possible empty triangles annotated with empty strings are feasible.

As we have specified all components, it is clear that $\mathcal{O}$ is indeed a layer-unconstrained nibbled ring subproblem. The one to one correspondence between triangulations of S and $\mathcal{O}$ follows easily. Let us only emphasize that Condition 2 of Definition 10 ensures that any valid triangulation of $\mathcal{O}$ has *at most k* layers. As we require the $(k+1)$-st layer to be empty. ◄

We use path-separators for the algorithm in Theorem 12. This requires a yet more specialized definition of subproblems for our dynamic programming scheme: *nibbled ring subproblems.*

We give a complete self-contained definition of *nibbled ring problems*, see Figure 9. As we want the definition to be self contained, we repeat also the parts that are equivalent to ring subproblems. These are the width, the inner and outer index, the free region, the free points, the layer constraints, the boundary annotations and the annotation system.

▶ **Definition 13** (Nibbled Ring Subproblem). A *nibbled ring subproblem* $\mathcal{S}$ can be considered as a plane graph consisting of several components with some extra conditions. The edges incident to the outer face form a simple polygon, denoted as *boundary polygon* Q_{bound}.

outer layer: The *outer layer* is a simple non-crossing connected polygonal chain $Q_{\text{out}} = Q_{\text{out}}(\mathcal{S})$ that is a connected subset of the boundary polygon Q_{bound}.

boundary paths: Two directed non-crossing paths $p_1(\mathcal{S}) = u_1 \ldots u_b'$ and $p_2(\mathcal{S}) = v_1 \ldots v_b$ ending at the different end vertices of the outer layer Q_{out}. Both paths are part of the boundary polygon and edge-disjoint from the outer layer. The length of the two boundary paths must not differ by more than one.

inner layer: A cactus $Q_{\text{in}} = Q_{\text{in}}(\mathcal{S})$ called *inner layer*. We explicitly allow the inner layer to be empty, but the inner layer index should be always defined. The inner layer may have one or zero components on the boundary, but it is always disjoint from the outer layer and does not share an edge with the boundary paths. To be most precise, the intersection of the inner layer and the boundary must be connected or empty.

inner/outer layer index and width: The *inner(outer) layer index* is some positive integer denoted by in-index($\mathcal{S}$) (resp., out-index($\mathcal{S}$)) and associated with the inner (resp., outer) layer. The width of a ring subproblem is defined as $w = w(\mathcal{S}) = $ in-index($\mathcal{S}$) − out-index($\mathcal{S}$) + 1. The meaning of the width is a little tricky. In case that the inner layer is empty it indicates an *upper* bound on the number of layers to be inserted. Otherwise, if the inner layer is non-empty, it indicates the *precise* number of layers.

forbidden regions/free regions: The bounded faces of the inner layer are called *forbidden region*. The remaining part inside the boundary polygon is the free region. We demand that all edges of the inner layer are incident to the free region. For an example, where this is violated, see Figure 9 d) and the edge that is marked red of the inner layer.

base edge: One simple segment called *base edge* of $\mathcal{S}$, denoted by $b = \text{base}(\mathcal{S})$. The base edge is always an edge of the boundary polygon and edge-disjoint from the outer layer and the boundary paths. There is a singular exception. The base edge might coincide with one boundary path in case the free region is empty, see Figure 9 g). There are three possible configurations of the base edge together with the inner layer, see Figure 9 c) d) and e).

free points: A set of points $P = P(\mathcal{S})$ in the interior of the free region.

vertices: The vertices of a nibbled ring subproblem are all the vertices of its components together with the free points.

In case, the subproblem $\mathcal{S}$ is clear from the context, we will suppress it in the notation, that is, we will write Q_{in} instead of $Q_{\text{in}}(\mathcal{S})$ and so on.
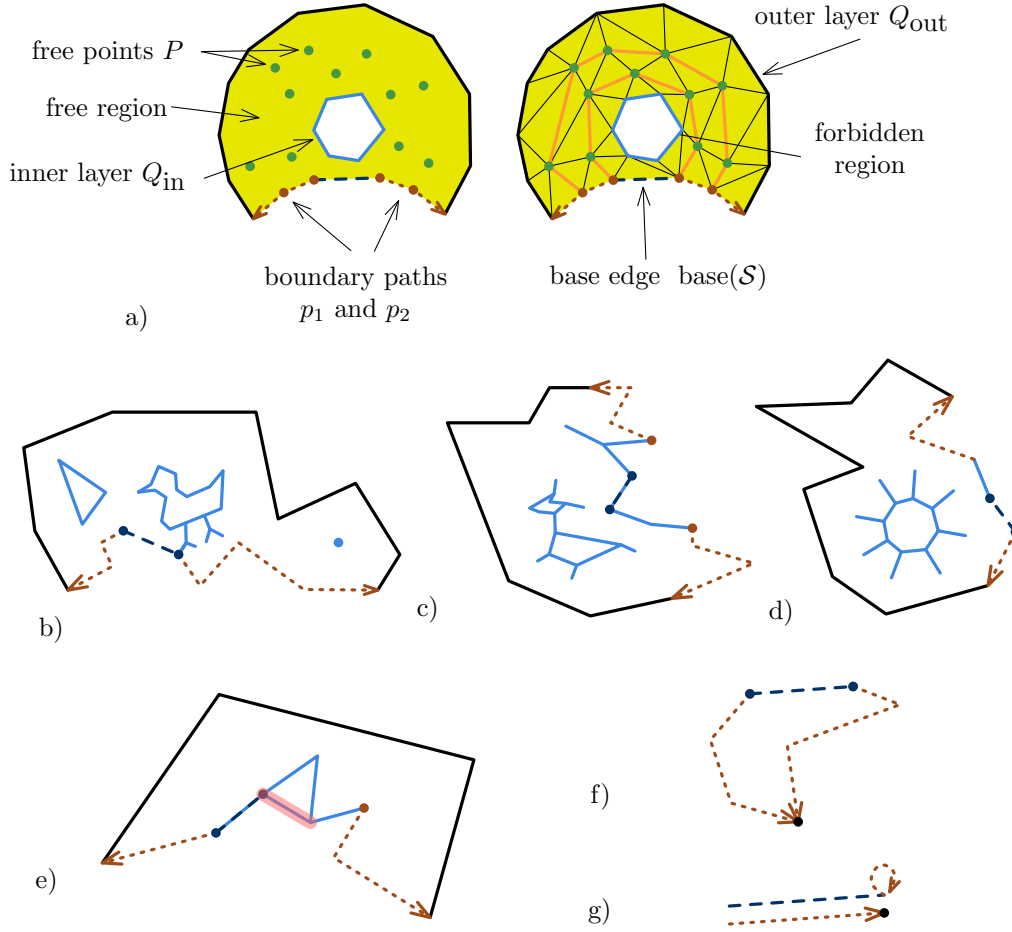
Figure 9 a) A nibbled ring subproblem $\mathcal{S}$ consisting of a base edge $base(\mathcal{S})$ depicted in dark blue dashed; two boundary paths, displayed in dotted brown from $base(\mathcal{S})$ to the outer layer Q_{out}, displayed in black; the free region depicted in light yellow; containing free points, depicted in green; The forbidden region is depicted in white. b) The inner layer is a cactus and it may consist of several components. It might happen that the base edge shares a vertex with the inner layer. c) The special case that the base edge lies on a component D of the inner layer. The boundary paths start, in this situation, from some vertex of D. The component D is supposed to have only edges on the free region of $\mathcal{S}$. d) The special case that a part of the inner layer and the base edge is on the boundary polygon. In this case however, the inner layer and the base edge are edge disjoint. e) In this case, the edge marked red is *not* adjacent to the free region and thus this is not a valid nibbled ring subproblem. f) The special case that the outer layer consists of a single point. g) The special case that $\mathcal{S}$ has an empty free region. The boundary path p_1 coincides with the base edge and the outer layer consists of a single point. The boundary path p_2 consists of a single vertex. This case is called degenerate subproblem.
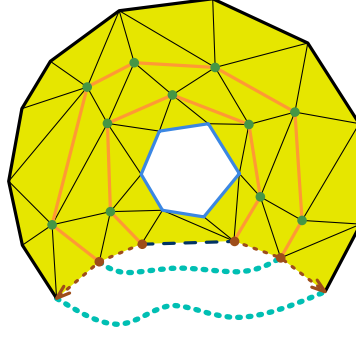
Figure 10 The layers of nibbled ring problems are defined with the help of auxiliary edges (drawn in turquise dotted), which represent the part of the layer of the original triangulation that is already removed ("nibbled") by previous recursion steps.

layer-constraints: This is just a vector $c = (i_1, \ldots, i_n)$ of length n, with its entries $i_j \in \{0, 1, 2, 3, \ldots, n\}$. We refer to c as the *layer-constraint vector* of $\mathcal{S}$. The j-th entry of c, denoted by $c(j) = i_j$, indicates that the j-th layer should have exactly i_j vertices. We assume that $c(j) = 0$, for all $j \notin [\text{out-index}(\mathcal{S}), \text{in-index}(\mathcal{S})]$.

If no layer-constraint vector is specified, we speak of the *layer-unconstrained* nibbled ring subproblem. We denote by $\mathcal{S}(c)$ the layer-unconstrained nibbled ring subproblem $\mathcal{S}$ appended with the layer-constraint vector c.

For the more general algorithm that is also able to count annotated triangulations, additional informations need to be maintained.

boundary annotation: Some string s on each vertex and edge of the inner layer, outer layer, the boundary paths and the base edge.

We should think of vertex and edge annotations as information that are attached to our subproblem, which is used later to impose some constraints on valid triangulations. It being a string is just a way to encode them.

annotation system: An *annotated* triangle is a 9-tuple consisting of 3 points, 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.

Consider the case that all potential triangles annotated with the empty string form the annotation system L then no actual restriction is imposed.

The size of the annotation system is defined as the length of the list and denoted by $|L|$. From the list of feasible triangles, we can derive for each vertex and edge a *list of feasible annotations*. It is defined by considering all annotations of feasible triangles with that specific vertex or edge.

▶ **Definition 14** (Valid Triangulation). Given a nibbled ring subproblem $\mathcal{S}$, consider the boundary paths $p_1(\mathcal{S}) = u_1 \ldots u_{b'}$ and $p_2(\mathcal{S}) = v_1 \ldots v_b$ and assume without loss of generality $b \leq b'$. Then, we define b auxiliary abstract edges $(u_{b'}, v_b), (u_{b'-1}, v_{b-1}), \ldots$, see Figure 10. We draw all the edges (not necessarily straight line) in a non-crossing manner so that the outer layer remains incident to the outer face. Let T be a triangulation extending the graph
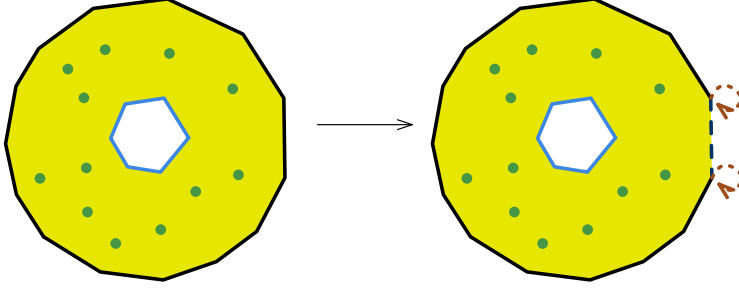
■ **Figure 11** Left: a simple ring subproblem Right: the transformed nibbled ring subproblem.

formed by $Q_{\text{in}} \cup Q_{\text{out}} \cup P \cup p_1 \cup p_2 \cup base$. We define T' as the triangulation T together with the auxiliary edges, which we just described.

The graph T' can be decomposed into cactus layers L_i as explained in Section 1.1. Here, we slightly change the indexing, by requiring that the first layer L'_j has index $j = \text{out-index}(\mathcal{S})$ and the second layer has index $\text{out-index}(\mathcal{S})+1$ and so on. The layers of T are defined by removing the auxiliary edges again. We denote by $d(v)$ the index of each vertex defined in this way. It is easy to see that $d(v)$ corresponds to the distance to the outer layer plus the outer-layer index. We call such a graph T of $\mathcal{S}$ a *valid triangulation of $\mathcal{S}$* if the following conditions are satisfied:

1. All faces in the free region are triangles and there are no edges outside the free region.
2. The graph L_j with $j = \text{in-index}(\mathcal{S})$ is the inner layer Q_{in}.
3. All layer-constraints are satisfied, that is

$$c(i) = |V(L_i)|.$$

4. For any vertex v_i of a boundary path p, the successor of v_i (v_{i+1}) must satisfy $d(v_i) = d(v_{i+1})+1$. Further v_{i+1} must be the neighbor of v_i in T with smallest order label among the neighbors with smaller distance to the outer layer.

We will later see that Condition 3 is always required, as we do not deal with nibbled ring subproblems without layer-constraints.

Further, a triangulation comes together with annotations of the edges and the vertices.

5. Each annotated colored triangle Δ of the triangulation T is in the annotation system, i.e. $\Delta \in L$.

We denote by $t(\mathcal{S})$ the number of valid triangulations of $\mathcal{S}$.

In the remainder of this section, we will show the following result.

▶ **Theorem 15.** *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 $n^{(5+o(1))w} \cdot |L|^{(6+o(1))w}$ time.*

Theorem 15 easily implies Theorem 12, using a simple transformation from ring subproblems to nibbled ring subproblems, see Figure 11. The algorithm indicated in Theorem 15 is called NIBBLING for later reference. The algorithm indicated in Theorem 12 is also called NIBBLING as it only transforms the problem into a nibbled ring subproblem.

Proof of Theorem 12 by Theorem 15. Given a ring subproblem $\mathcal{S}$, we define a nibbled ring subproblem $\mathcal{S}$' as follows:
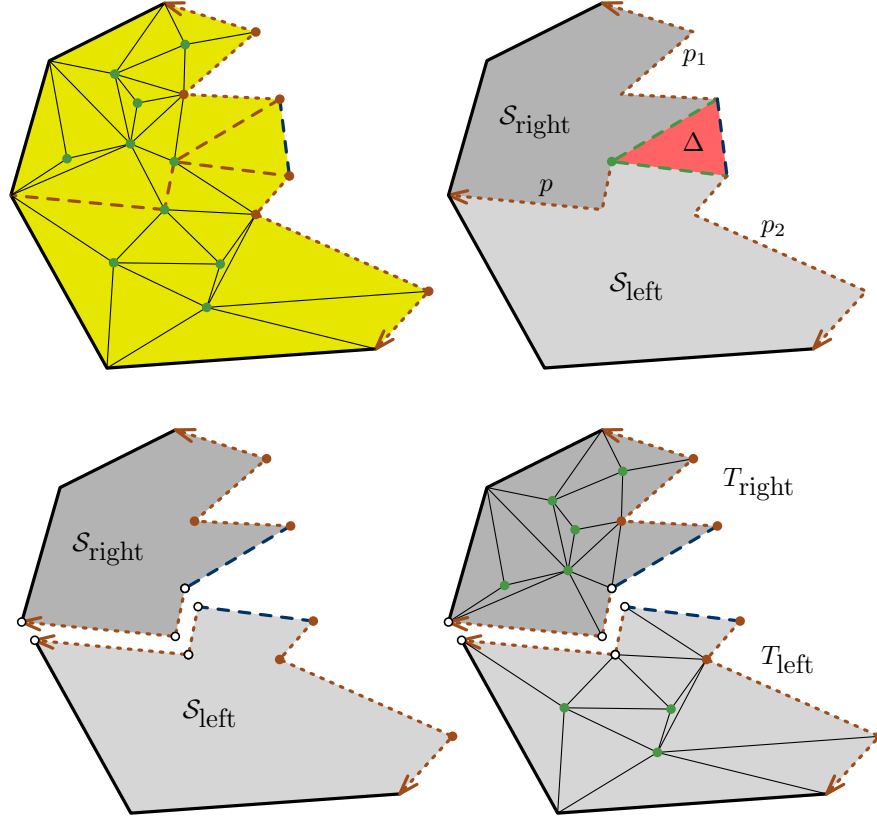
■ **Figure 12** On the top left is a nibbled ring subproblem $\mathcal{S}$ together with a valid triangulation T. There is a unique triangle Δ adjacent to the base edges. From the vertex v of Δ that is not incident to the base edge exists a path p to the outer layer. The triangle Δ and the path p are 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 $\mathcal{S}_{\mathrm{right}}$ and $\mathcal{S}_{\mathrm{left}}$. 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.

base edge: Pick any edge on $Q_{\mathrm{out}}(\mathcal{S})$ and define it to be the base edge $\mathrm{base}(\mathcal{S}')$ of $\mathcal{S}$'.
outer layer: Let $Q_{\mathrm{out}}(\mathcal{S}')$ be the simple polygonal chain remaining after deletion of $\mathrm{base}(\mathcal{S}')$.
boundary paths: Are defined to be the endpoints of the base edge and have length zero.

Layer-constraints, indices, the inner layer, free points, annotations and the annotation system carry over without changes, see Figure 11.

Claim(Transformation) Let $\mathcal{S}$ be a simple ring subproblem and $\mathcal{S}'$ be its transformation as defined above. Then $t(\mathcal{S}) = t(\mathcal{S}')$.

We have to show T is a valid triangulation of $\mathcal{S}$ if and only if it is a valid triangulation of $\mathcal{S}'$. The free region, the inner layer and the layer-constraints of $\mathcal{S}$ and $\mathcal{S}'$ are the same and thus Conditions 1, 2 and 3 of Definition 10 and 14 are equivalent. Condition 4 of Definition 14 is trivially satisfied as both boundary path have length zero. Also the condition about feasbile triangles does not change. ◄

▶ **Definition 16** (Degenerate subproblems)**.** The subproblems that are not solved recursively are called *degenerate subproblems*, see Figure 9 f). To be explicit they consist of one base edge, which coincides with one of the boundary paths. The other boundary path has length zero and the outer layer also consists of a single vertex only. Both edges and vertices have some feasible annotation. The number of triangulations of an degenerate problem is 1. It must be checked that each layer has the intended size to see if it is valid. (The intended size is given by the layer-constraint vector.) All other conditions are always satisfied. (Note that this is the only place, where the algorithm actually checks if the constraint condition is satisfied.) The number of degenerate subproblems on a set of n points in the plane is quadratic in n and $|L|$.

For the remainder of this section, we formally define how we split a nibbled ring subproblem into two nibbled ring subproblems, see also Figure 12. As a first step, we define separator paths. Their definition is motivated by canonical outgoing paths. Canonical outgoing paths are defined in terms of a given triangulation T. This makes them unsuitable as a separator for nibbled ring subproblems, as we do not know the triangulations of our nibbled ring subproblems. Separator paths are defined only in terms of the nibbled ring subproblem. However, we need to make sure that every canonical path is also a separator path. Thereafter, we are ready to define the split of a nibbled ring subproblem $\mathcal{S}$ into two nibbled ring subproblems $(\mathcal{S}_{\text{left}}, \mathcal{S}_{\text{right}})$ using a separator path p. We also have to show how we split the layer-constraints. At last we will show that this indeed leads to a recursion that counts the number of triangulations correctly. This boils down to show that a valid triangulation T splits into two valid triangulations $(T_{\text{left}}, T_{\text{right}})$ and reversely two valid triangulations $(T_{\text{left}}, T_{\text{right}})$ can be combined to a valid triangulation T.

For the following definition, see the top left of Figure 12 for an illustration.

▶ **Definition 17** (Canonical Outgoing Paths)**.** Given a valid triangulation T of a nibbled ring subproblem $\mathcal{S}$, we define its *base triangle* Δ as the unique triangle incident to the base edge. The vertex v_c of the base triangle that is not incident to the base edge is the *base vertex* of T. Then the *canonical outgoing path* $p = w_1 \ldots w_a$ of T is a directed path in T such that:

1. The triangle Δ is empty, feasible and it is formed by w_1 and the base edge.
2. The number of vertices on the path a is bounded by the width w. (In short: $a \leq w$.) And the last vertex is on the outer layer.
3. All edges and vertices are annotated with a feasible annotation.
4. Any edge e of Δ with both endpoints shared with the inner layer must also be on the inner layer.

There are two more conditions that we separate as they can be only formulated with the help of the underlying triangulation.

outwardness condition: The distance to the outer layer decreases, when one "goes" along the path. More precisely: $d(w_{i+1}) = d(w_i) - 1$, for all $i = 1, \ldots, a - 1$.
order label condition: For each vertex w_i the vertex w_{i+1} is the neighbor with smallest order label among all neighbors with smaller distance to the outer layer.

Property 1 ensures that Δ is defined as wished. Property 2 is important for the runtime analysis. It follows implicitly from the outwardness condition and the definition of the width. Property 3 is again important for the runtime analysis. (We only have to guess feasible annotations.) Property 4 ensures that the inner layer remains correct. This is Condition 2 of Definition 14.

Every vertex v, has some adjacent vertices with lower layer-index according to Lemma 2. Among all vertices with this property, there is one vertex with smallest order label. Following these edges from the base vertex shows the existence of the canonical outgoing path. The outwardness condition implies the upper bound on the length of the path. The order label condition ensures uniqueness. For later use, we summarize this in the following lemma.

▶ **Lemma 18.** *Given a valid triangulation T for some nibbled ring subproblem $\mathcal{S}$. Then there exists exactly one canonical outgoing path $p^*(T)$.*

The outstanding property of canonical outgoing paths is that they *separate* one triangulation into two triangulations in a *canonical* way. The technical difficulty is that they are defined in terms of triangulations. In order to use them as separators, we have to define a similar notion *without* reference to a triangulation. This is necessary as our algorithm does not know the triangulations of $\mathcal{S}$ usually. These first 4 conditions are identical to the one of canonical outgoing paths, as they do not rely on some underlying triangulation. We replace the outwardness condition and the order label condition appropriately.

▶ **Definition 19** (Separator Path of Nibbled Ring Subproblems)**.** Let $\mathcal{S}$ be a nibbled ring subproblem, Δ be a triangle incident to the base edge and $p = w_1 \ldots w_a$ a path. We say Δ and p form a *separator path* of $\mathcal{S}$ if the following conditions are met:

1. The triangle Δ is empty, feasible and it is formed by w_1 and the base edge.
2. The number of vertices on the path a is bounded by the width w. (In short: $a \leq w$.) And the last vertex is on the outer layer.
3. All edges and vertices are annotated with a feasible annotation.
4. Any edge e of Δ with both endpoints shared with the inner layer must also be on the inner layer.

Further, we define the *index* of each vertex of p inductively as follows: $\text{index}(w_a) = \text{out-index}(\mathcal{S})$ and $\text{index}(w_i) = \text{index}(w_{i+1}) + 1$. We define the index in the same manner for the boundary paths. For a vertex v on the outer layer (resp. inner layer) $\text{index}(v) = \text{out-index}(v)$ (resp. $\text{index}(v) = \text{in-index}(v)$). Here, the index plays the role of the distance to the outer layer $d(v)$ as defined in Definition 14. We denote by G the graph formed by Δ, p and $\mathcal{S}$. We define $N_G(v)$ as the neighbors of v in G.

replaced outward condition: The index can be consistently defined for each vertex on the inner layer, outer layer, the boundary paths and the separator path. In particular for the shared vertices. No two adjacent vertices in G differ in their index by more than one.

replaced order label condition: Let x_i and x_{i+1} be two adjacent vertices on either one of the boundary paths or the separator paths. Then x_{i+1} has the smallest order label in the set $\{\, y \in N_G(v) : \text{index}(x_{i+1}) = \text{index}(y) \,\}$.

We denote by $\mathcal{P}(\mathcal{S})$ the set of all possible separator paths of $\mathcal{S}$.

When we define separator paths, we have no access to a triangulation we could refer to. In particular we have no function $d(v)$. We replace it by giving each vertex v an index, which essentially plays the role of $d(v)$. The replaced outwards condition ensures that the index is at least consistent on G. We will later see that this is sufficient for the algorithm to work correctly. Morally, when we never insert an edge that is inconsistent locally, the final triangulation(at the end of the algorithm) is also consistent locally.

Note that Condition 1 is the only place where we actually check if a triangle is feasible. However, this is also the only place where we actually insert triangles. The important part
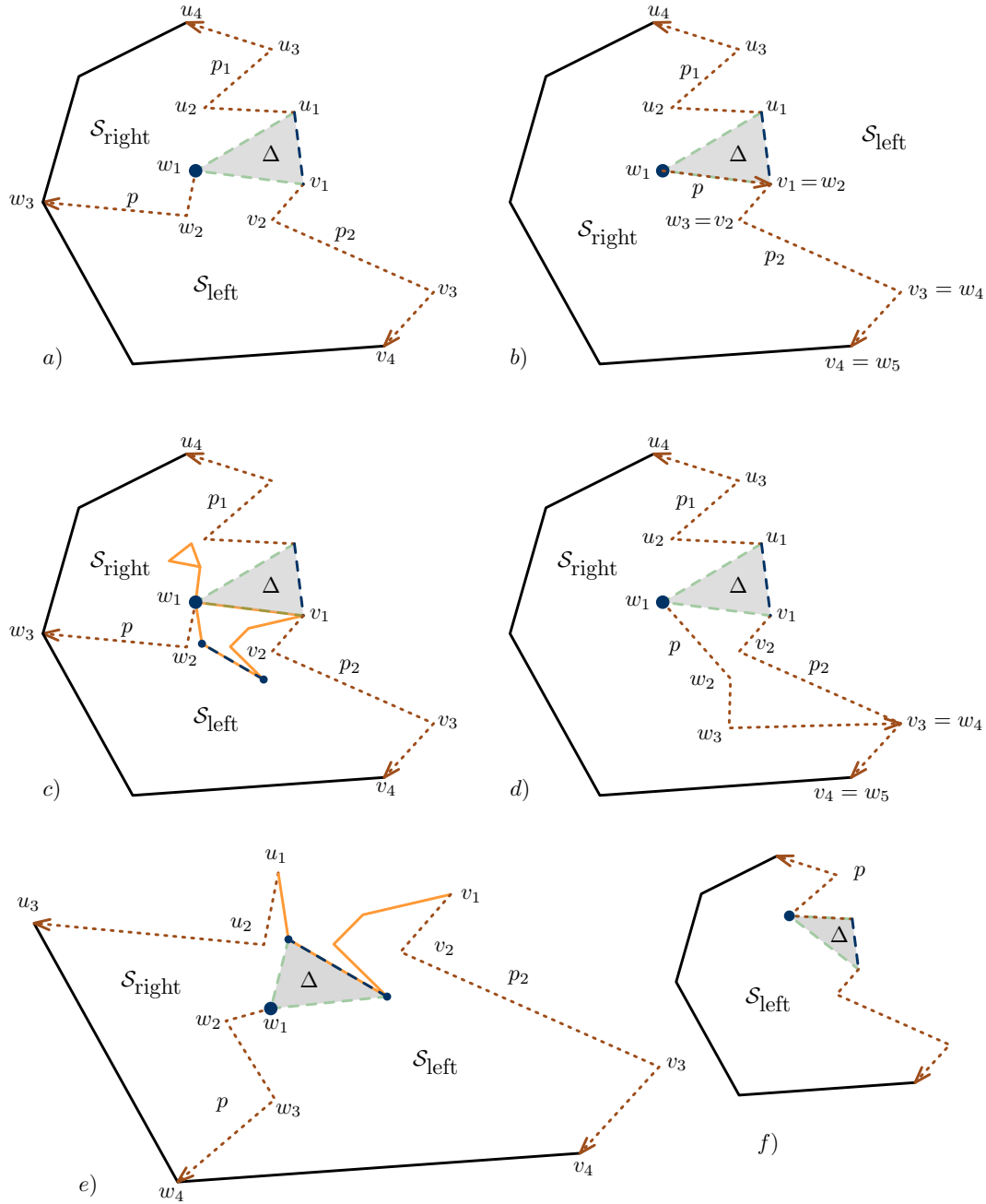
Figure 13 a) A split in the common way. i.e., the separator path p uses only free points of $\mathcal{S}$. b) The separator path p coincides with one of the edges of Δ. Thus $\mathcal{S}_{\text{left}}$ is an degenerate problem. c) One of the edges of Δ agrees with an edge of Q_{in}. Therefore we pick any edge of this component of Q_{in} as base edge of $\mathcal{S}_{\text{left}}$. Also note that we split this component D in this case. d) The path p merges at some point with p_2. For the subproblem $\mathcal{S}_{\text{left}}$ we shorten both boundary paths and define their first common vertex as outer layer Q_{out} of $\mathcal{S}_{\text{left}}$ consisting of exactly one vertex. e) The subproblems have the base edge adjacent to Q_{in}, but not on it. f) One side of the triangle Δ might coincide with one of the boundary paths. In this case, the separator path is the separator path shortened by one.

of this definition is that the set of separator paths include the set of all potential canonical paths.

The replaced outward condition and the replaced order label condition have some interesting consequences. We will not need to make use of them explicitly in order to show correctness, but we hope to give the reader a better understanding of the replaced order label condition.

1. The boundary path p has *at most* one vertex on the outer layer.

2. Assume the separator path p and one of the boundary paths, say p_1, share a vertex v, then all forthcoming vertices of p and p_1 are also shared, see Figure 13 b), d) and f).

3. If the separator path p shares a vertex with the inner layer, it must be the first vertex and p must have length $w - 1$ (i.e., w vertices.)

4. At last it implies a very technical property. Assume that the start vertex w_1 of the separator path and the start vertex u_1 of one of the boundary paths are adjacent. Further assume $\text{index}(u_1) = \text{index}(w_1) - 1$. The first consequence is that the separator path is one edge longer than the boundary path. More interestingly, in case that the two paths do not share a vertex, we have that the order label of w_2 is smaller than the order label of u_1.

Now, we are ready to show how we split a nibbled ring into two nibbled rings using separator paths. For the following definitions, see Figure 13 for illustrations. Later, we will generate a large number of nibbled ring subproblems arising from them by imposing different layer-constraints on them.

▶ **Definition 20** (Splitting via Separator Paths)**.** Given a path $p = w_1 \ldots w_c$ and a triangle Δ forming a separator path of a nibbled ring subproblem $\mathcal{S}$, we can define two layer-unconstrained nibbled ring subproblem $\mathcal{S}_{\text{left}} = \mathcal{S}_{\text{left}}(p)$ and $\mathcal{S}_{\text{right}} = \mathcal{S}_{\text{right}}(p)$ as follows.

The region bounded by $\mathcal{S}$ naturally decomposes into Δ, A_{left} and A_{right} as in Figure 13. (Edges can be shared and it is allowed for A_{left} and A_{right} to have no interior points.) We specify the components of subproblem $\mathcal{S}_{\text{right}}$ as follows:

boundary paths: Its boundary paths are p and p_1. If they are not disjoint only include them up to the first vertex, where they agree.

base edge: Its base edge $\text{base}(\mathcal{S}_{\text{right}})$ is usually the edge e of Δ that is incident to A_{right}.

There is a very specific exception. It might be the case that e is incident to a face f of the inner layer. In this case delete e and choose any other edge on f, see Figure 13 c) and e).

outer layer: Its outer layer $Q_{\text{out}}(\mathcal{S}_{\text{right}})$ is the part on $Q_{\text{out}}(\mathcal{S})$ between the endpoints of p and p_1. Except if p and p_1 have a common vertex, then only this vertex forms $Q_{\text{out}}(\mathcal{S}_{\text{right}})$. Note this vertex might not be contained $Q_{\text{out}}(\mathcal{S})$. In the first case we define the outer layer index by $\text{out-index}(\mathcal{S}_{\text{right}}) = \text{out-index}(\mathcal{S})$, in the second case the outer layer index is defined by the index of the first common vertex of p and p_1.

inner layer: Every connected component of $Q_{\text{in}}(\mathcal{S})$ that lies in A_{right}. It might be that one component D lies in A_{right} and in A_{left}. In this case, we split D in the obvious way. Note that exactly one vertex of D is shared by the two subproblems in this case.

It might happen that one edge is not adjacent to the region A_{right}, see Figure 13 e). In this case, we delete this edge. The inner layer index carries over from $\mathcal{S}$.

free points: The free points $P(\mathcal{S}_{\text{right}})$ are the points $P(\mathcal{S})$ contained in the region A_{right}.

boundary annotations: Carry over from $\mathcal{S}$ and the separator path p.

annotation system: This is exactly the same annotation system as for $\mathcal{S}$.

The subproblem $\mathcal{S}_{\text{left}}$ is defined in the same way, with the replacement of p_1 by p_2 and A_{right} by A_{left}. It is easy to check that $\mathcal{S}_{\text{left}}$ and $\mathcal{S}_{\text{right}}$ are both layer-unconstrained nibbled ring subproblems.

▶ **Definition 21** (Compatible Layer-Constraints). Let $\mathcal{S}$ be a nibbled ring subproblem with layer-constraint c and $p \in \mathcal{P}(\mathcal{S})$ a separator path. We define the *layer-constraint vector of* p, denoted by c_p, as follows. Let $p = w_1 \dots w_a$ and $p' = w_1 \dots w_{a'}$ be the part of p that is shared by $\mathcal{S}_{\text{left}}$ and $\mathcal{S}_{\text{right}}$, see Figure 13 b), d) and f) for examples, where $p \neq p'$. Further let I be the set of indices of the vertices in p'. Formally I is defined as $I = \{\, \text{index}(v) : v \text{ is a vertex of } p' \,\}$. Recall that the index was defined in Definition 19. We define c_p as the indicator vector of I, to be more explicit:

$$c_p(i) = \begin{cases} 1 & \text{, if } i \in I \\ 0 & \text{, otherwise.} \end{cases}$$

The purpose of the vector c_p is to ensure that vertices that are shared by the two subproblems are not accounted for twice. Further the set of *compatible layer-constraints* $\mathcal{C}(\mathcal{S}, p)$ is defined by

$$\mathcal{C}(\mathcal{S}, p) = \{\, (c_1, c_2) : c_1 + c_2 = c + c_p \text{ and } c_i \in \{0, 1, \dots, n\}^n \,\}$$

We denote by $\mathcal{S}_{\text{left}}(p, c_1)$ and $\mathcal{S}_{\text{right}}(p, c_2)$ the left and right nibbled ring subproblems appended with the layer-constraint vector c_1 and c_2 respectively.

▶ **Lemma 22** (Recursion). *Let $\mathcal{S}$ be a nibbled ring subproblem. Then holds*

$$t(\mathcal{S}) = \sum_{p \in \mathcal{P}(\mathcal{S})} \sum_{(c_1, c_2) \in \mathcal{C}} t(\mathcal{S}_{\text{left}}(p, c_1)) \cdot t(\mathcal{S}_{\text{right}}(p, c_2)).$$

Proof. "$\leq$": Consider some valid triangulation T of $\mathcal{S}$ and let Δ_c be the canonical triangle and v_c the canonical vertex. By Corollary 18 there exists a unique canonical outgoing path $p = p^*(T)$ starting at v_c. Note that p satisfies Definition 19 and thus can be used as a separator path. Thus, it splits $\mathcal{S}$ geometrically into $\mathcal{S}_{\text{left}}$ and $\mathcal{S}_{\text{right}}$ as in Definition 20. Denote by T_{left} and T_{right} the triangulations that arise, when T is restricted to $\mathcal{S}_{\text{left}}$ and $\mathcal{S}_{\text{right}}$ respectively. Then there exists exactly one pair of layer-constraint vectors (c_1, c_2) with $c_1 + c_2 = c + c_p$ such that T_{left} and T_{right} satisfy c_1 and c_2 respectively. (Vertices on p are accounted for on T_{left} and T_{right}.) It is easy to see that two different valid triangulations T and T' lead two two different pairs of valid triangulations $(T_{\text{left}}, T_{\text{right}})$ and $(T'_{\text{left}}, T'_{\text{right}})$

We have to show that T_{left} and T_{right} are valid triangulations of $\mathcal{S}_{\text{left}}$ and $\mathcal{S}_{\text{right}}$ respectively. It suffices to do this for T_{left}. We check all conditions of Definition 14 explicitly. Condition 1 follows trivially. Condition 3 follows from the definition of the layer-constraint vectors (c_1, c_2). Condition 4 follows from the fact the the condition holds for T and the replaced outward condition and the replaced order label condition in the definition of separator paths. Condition 5 follows from the fact that every triangle of T_{left} is also a triangle of T.

Condition 2 is more intricate. We show first by induction that every vertex v in T_{left} has the same distance to the outer layer as in T. This is true for the vertices on the outer layer of $\mathcal{S}_{\text{left}}$. Consider the vertex v of $V(T_{\text{left}})$ with distance d to the outer layer of $\mathcal{S}$. Then there is a set of other vertices $\{x_1, \dots, x_b\} \subset N_T(v)$ of vertices with smaller distance to the outer layer in $\mathcal{S}$. At least one of the vertices $x_1, \dots, x_b$ belongs to T_{left}. (Actually, all of them,

except v is on the separator path.) By the induction hypothesis $x_1, \ldots, x_b$ have distance $d-1$ to the outer layer in T and in T_{left}. Thus v has distance d in T_{left}. Thus also all vertices of the inner layer $Q_{\text{in}}(\mathcal{S}_{\text{left}})$ have the correct distance to the outer layer.

We also have to show that each edge remains on the layer L_j with $j = \text{in-index}(\mathcal{S}_{\text{left}})$ and no other edge appears on L_j. We consider first the case that an edge $e = (v, w)$ belongs to L_j of T_{left}, but it does not belong to L_j of T. The only possibility to belong to the L_j in T_{left} is if both endpoints v and w belong to L_j of T_{left} and thus also v and w belong to the L_j of T. But there is no edge that has both endpoints on the inner layer of T, without also belonging to the inner layer $Q_{\text{in}}(\mathcal{S})$, as this edge needs to be inside some bounded face of $Q_{\text{in}}(\mathcal{S})$.

Now, we assume that e belongs to $Q_{\text{in}}(\mathcal{S})$ and we want to show that e also belongs to the inner layer of T_{left}. At first note that v and w also belong to $Q_{\text{in}}(\mathcal{S})$ and this implies that e is not an edge of the boundary path or the separator path. Further there exists a triangle $\Delta = \Delta(v, w, x)$ in T incident to e such that x belongs to layer $i - 1$. (This condition is necessary and sufficient for an edge to belong to layer i.) It is easy to see that this triangle cannot be the base triangle, see Figure 13 c). And thus Δ is also a triangle of T_{left} and this implies e is also an edge of the inner layer of T_{left}. From this discussion follows Condition 2.

Thus T_{left} and T_{right} are both valid triangulations.

"$\geq$": Fix any separator path $p \in \mathcal{P}(\mathcal{S})$ and any pair of compatible constraint vectors (c_1, c_2). Further let T_{left} and T_{right} be valid triangulations of $\mathcal{S}_{\text{left}}(p, c_1)$ and $\mathcal{S}_{\text{right}}(p, c_2)$ respectively. The valid triangulations T_{left} and T_{right} define a triangulation T by taking the union. We show that T is a *valid* triangulation of $\mathcal{S}$ by going through the complete list of requirements in Definition 14.

First we show that a different pair of valid triangulations $(T'_{\text{left}}, T'_{\text{right}})$ needs necessarily lead to a different triangulation T'. For the sake of contradiction assume that $T = T'$. Then there is a canonical outgoing path p of T, which can be used as a separator path. As $(T'_{\text{left}}, T'_{\text{right}}) \neq (T_{\text{left}}, T_{\text{right}})$ at least one pair, say $(T'_{\text{left}}, T'_{\text{right}})$, comes from $(\mathcal{S}_{\text{left}}(q), \mathcal{S}_{\text{right}}(q))$ with $q \neq p$. Let v be the last vertex still shared by p and q. (As the base triangle is the same there exists at least one such vertex.) And let w be the successor of v on p. Without loss of generality w belongs to T'_{right} and thus T'_{right} is not a valid triangulation of $\mathcal{S}_{\text{right}}(q)$ as Condition 4 is violated.

Now let us show that T is a valid triangulation of $\mathcal{S}$ by checking all conditions explicitly. It is easy to see that all faces are triangular and no edge lies outside the free region. Thus Condition 1 is satisfied. Condition 3 follows as no vertex changes the distance to the outer layer. Condition 4 follows from the definition of the separator path and the fact that Condition 4 holds for T_{left} and T_{right}. Let $p' = v_1, \ldots, v_a$ be one of the boundary paths of our subproblem $\mathcal{S}$. We have to show the following technical condition: For any vertex v_i of p' holds that the successor v_{i+1} must be the neighbor of v_i in T with smallest order label among the neighbors with smaller distance to the boundary. Note that all neighbors of v_i are either in T_{left} or in T_{right} as v_i is a vertex of a boundary path. Thus as Condition 4 holds for T_{left} and T_{right} it also holds for T. Condition 5 follows from the fact that each empty triangle of T is a feasible triangle in either T_{left} or T_{right}. Further by Condition 1 of Definition 19 (separator path) the base triangle is also feasible.

For Condition 2 of Definition 14, we have to show that $Q_{\text{in}}(\mathcal{S})$ is the layer L_j of T with $j = \text{in-index}(\mathcal{S})$. Note that the distance to the boundary of each vertex of T_{left} and T_{right} is the same as in T. Every vertex v shared by T_{left} and T_{right} has the same distance to the boundary $d(v)$ due to the enforced distances by the indices on the separator path p. This implies the vertices of $Q_{\text{in}}(\mathcal{S})$ correspond to the vertices of L_j. It remains to show that the

edges are the same as well. For this purpose let e be an edge of $Q_{\text{in}}(\mathcal{S})$ then it is either an edge of T_{left}, T_{right} or an edge of the base triangle of $\mathcal{S}$. Thus it must also be an edge of L_j by the fact that T_{left} and T_{right} satisfy Condition 2 of Definition 14 and Condition 4 of Definition 19. The reverse direction goes by the same argument. This shows that T satisfies all conditions of Definition 14 and thus is a valid triangulation. ◀

Algorithm 1 Nibbling: Counting triangulations of nibbled rings with width w.

Require: $\mathcal{S}$ to be a nibbled ring subproblem of width w;
 1: Initialize Search tree τ to store all computed values and insert the degenerate subproblems.
 2: **return** Count($\mathcal{S}$)

 3: **function** Count($\mathcal{S}$)
 4:     **if** $\mathcal{S}$ in τ **then**
 5:         **return** $\tau(\mathcal{S})$
 6:     **end if**
 7:     $s = 0$
 8:     **for** separator paths $p \in \mathcal{P}(\mathcal{S})$ **do**
 9:         **for** constraints $c_1 + c_2 = c + c_p$ **do**
10:             Define $\mathcal{S}_{\text{left}}(p, c_1)$ and $\mathcal{S}_{\text{right}}(p, c_2)$ using p and constraints (c_1, c_2).
11:             $s = s + \text{Count}(\mathcal{S}_{\text{left}}(p, c_1)) \cdot \text{Count}(\mathcal{S}_{\text{right}}(p, c_2))$
12:         **end for**
13:     **end for**
14:     insert $(\mathcal{S}, s)$ into τ
15:     **return** s
16: **end function**

We are now ready for the proof of Theorem 15.

Proof Theorem 15. We will first describe the Nibbling algorithm, then show its correctness and finally supply a runtime analysis.

The Algorithm Nibbling uses the memoization technique and is based on dynamic programming. The subproblems of the dynamic programming scheme are the nibbled ring subproblems. Already computed solutions are stored in a search tree denoted by τ. At the beginning all degenerate subproblems are inserted into τ with its correct value. Otherwise each nibbled ring subproblem is solved recursively, by the recursion of Lemma 22. The pseudocode of Nibbling is depicted as Algorithm 1.

It is easy to see that there exists exactly one triangulation for each degenerate problem. In order to check if this triangulation is valid, we have to check only if the layer-constraint are satisfied. All other conditions of Definition 14 are trivially satisfied. And thus the algorithm will return the correct output for these subproblems. By induction, all other subproblems are computed correctly as well. The induction step is done in Lemma 22.

The algorithm runs in $n^{O(w)} \cdot |L|^{O(w)}$ time. To see this, we give an upper bound on the total number of subproblems and the total number of recursive calls that can potentially appear. We also have to account for searches in the searchtree τ. But each search has costs of $\log(n^{O(w)}) = w \log n = O(n^2)$. We add these costs to the recursive calls.

Denote by $\mathcal{S}$ the initial input. Then all subsequent subproblems appearing are defined by the two boundary paths together with the base edge and some constraints. In the case that the outer layer of some subsequent subproblem does not coincide with the outer layer of the initial nibbled ring subproblem appears only if the two boundary paths share their

last vertex. Thus also in this case the outer layer is completely defined by the boundary paths.

Note that every boundary path consists of at most $w + 1$ vertices and w edges. And thus there are at most $n^{w+O(1)} \cdot |L|^{2w+O(1)}$ possible annotated boundary paths possible. The number of possible constraint vectors is bounded by n^w, as there are at most w entries that are not predetermined to be zero, the size of the outer layer or the size of the inner layer. Thus the total number of potential nibbled ring subproblems is bounded by $n^{3w+O(1)} \cdot |L|^{4w+O(1)}$

The number of recursive calls per ring subproblem is bounded by the number of separator paths times the number of pairs of compatible constraint vectors (c_1, c_2). Given c_1, there exists only one constraint vector c_2 compatible to it. Thus there are at most n^w compatible pairs.

The number of separator paths is bounded by by $n^{w+O(1)} \cdot |L|^{2w+O(1)}$ in the same way as we bounded the number of boundary paths. Thus the number of recursive calls is bounded by $n^{2w+O(1)} \cdot |L|^{2w+O(1)}$. This remains true even if we add the costs for the searches in τ.

The total running time is thus $O(n^{5w+O(1)} \cdot |L|^{6w+O(1)}) = n^{O(w)} \cdot |L|^{O(w)}$ as claimed. ◀

4 General Layer-Unconstrained Ring Subproblems

Here, we give describe the algorithm to count Layer-Unconstrained Ring Subproblems. We start, to define formally the layers, we are aiming to use as separators and show first that every triangulation has exactly one. Then we define a set of all separators, we want to recurse on. Further, we show how to split a ring subproblem using a layer separator into an inner and outer ring subproblem. For the outer ring subproblem, we will define appropriate layer-constraints, to ensure that the layer separator we used is indeed canonical for all triangulations, we will count henceforth. We finish with a full description of the algorithm, a proof of correctness and an upper bound on the running time.

▶ **Definition 23** (Peripheral Layers). Given a valid triangulation T of a ring subproblem $\mathcal{S}$, we say Layer L_i of T is m-peripheral if out-index$(\mathcal{S}) < i \leq$ out-index$(\mathcal{S}) + m$. Lemma 24 and the definitions following thereafter depend on a parameter m, which will be chosen later.

▶ **Lemma 24** (Layer Separation). *Let $\mathcal{S}$ be a ring subproblem of width $w = w(\mathcal{S}) \geq m + 2$ with at most n free points. And let T be a valid triangulation of $\mathcal{S}$. Then there exists*

(a) an m-peripheral layer L of T of size $\leq \left\lfloor \frac{n}{m} \right\rfloor$,

(b) the smallest index i^* with $|V(L_{i^*})| \leq \left\lfloor \frac{n}{m} \right\rfloor$ of T is unique and

(c) the layer L_{i^*} separates the inner layer from the outer layer. This implies every connected component of $Q_{\text{in}}(\mathcal{S})$ is in some bounded face of L_{i^*}.

(d) All m-peripheral layers further outside of L_{i^*} have size at least $\left\lfloor \frac{n}{m} \right\rfloor + 1$.

(e) All edges and vertices of L_{i^*} have a feasible annotation.

Proof. (a) There are exactly m potential layers, each pair of layers is vertex disjoint and there are at most n potential vertices. The claim follows by the pigeonhole-principle. (b) This is immediate from the definition. (c) This follows from the way we defined our layers. (d) This follows from the definition of L_{i^*}. (e) follows from the fact that every edge of a valid triangulation has some feasible annotation. ◀

▶ **Definition 25** (Peripheral Layer Separators). Given a ring subproblem $\mathcal{S}$, we define a *peripheral layered separator* L of $\mathcal{S}$ as a cactus with an index, denoted by index(L), such that
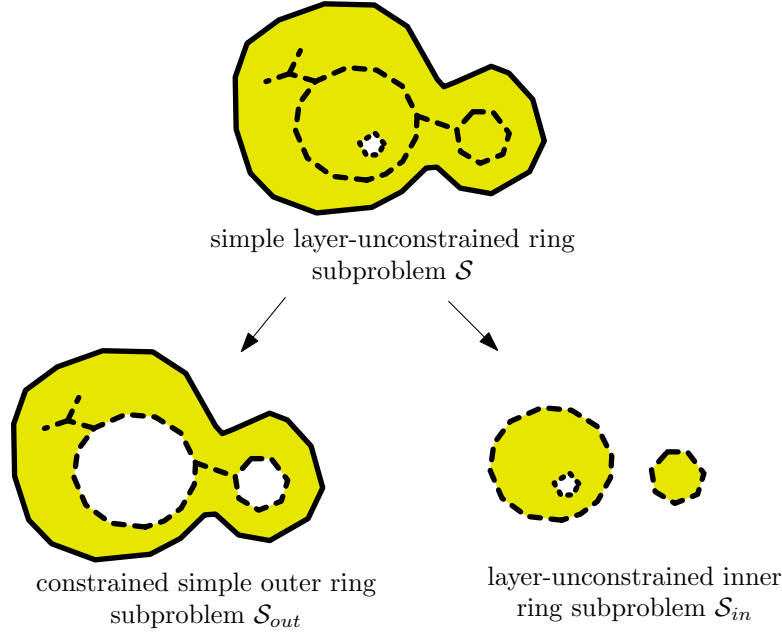
Figure 14 Illustration how to *peel off* the outer ring subproblem.

1. L does not induce any crossings with $\mathcal{S}$ and $V(L) \subseteq P(\mathcal{S})$.

2. Every connected component of $Q_{\mathrm{in}}(\mathcal{S})$ is inside some bounded face of L and L is contained in the interior of the outer layer $Q_{\mathrm{out}}(\mathcal{S})$.

3. The index of L satisfies out-index$(\mathcal{S}) <$ index$(L) \leq$ out-index$(\mathcal{S}) + m$.

4. $|V(L)| \leq \left\lfloor \frac{n}{m} \right\rfloor$.

5. All edges and vertices of L have a feasible annotation.

We denote with $\mathcal{L}(\mathcal{S})$ the set of all peripheral layer separators of $\mathcal{S}$.

In Definition 26, we will define an inner and outer layer-unconstrained ring subproblem $\mathcal{S}_{\mathrm{in}}$ and $\mathcal{S}_{\mathrm{out}}$, given a simple layer-unconstrained ring subproblem $\mathcal{S}$ and a peripheral separator layer L. Thereafter in Definition 27, we will define a set of constraints $\mathcal{C}_{\mathrm{out}}$ for the layer-unconstrained ring subproblem $\mathcal{S}_{\mathrm{out}}$. At last Lemma 28 describes a recursion and asserts that this recursion can be used to correctly count all valid triangulations of $\mathcal{S}$.

▶ **Definition 26** (Split by Layers). Given a *layer-unconstrained, simple*, ring subproblem $\mathcal{S}$ together with a peripheral layered separator $L \in \mathcal{L}(\mathcal{S})$, we are now ready to describe the two arising layer-unconstrained ring subproblems. One of them is called the *outer problem* and the other is called the *inner problem*. We define the layer-unconstrained outer problem $\mathcal{S}_{\mathrm{out}} = \mathcal{S}_{\mathrm{out}}(L)$ as follows:

outer layer: $Q_{\mathrm{out}}(\mathcal{S}_{\mathrm{out}}) = Q_{\mathrm{out}}(\mathcal{S})$.

inner layer: $Q_{\mathrm{in}}(\mathcal{S}_{\mathrm{out}}) = L$.

outer/inner layer index: out-index$(\mathcal{S}_{\mathrm{out}}) =$ out-index$(\mathcal{S})$ and in-index$(\mathcal{S}_{\mathrm{out}}) =$ index(L).

free points $P(\mathcal{S}_{\mathrm{out}})$ is the subset of $P(\mathcal{S})$ in the outer face of L.

boundary annotations: Carry over from $\mathcal{S}$ and the separator layered separator L.

annotation system: This is exactly the same annotation system as for $\mathcal{S}$.

We do not specify the free region, vertices and the width as they arise from the components given above. It might have seemed at first a little unmotivated that we allowed $Q_{\text{out}}(\mathcal{S})$ to consists of more than one polygon, but now we need it as our layers might have more than one component. We define the layer-unconstrained inner problem $\mathcal{S}_{\text{in}} = \mathcal{S}_{\text{in}}(L)$ as follows:

outer layer: The outer layer $Q_{\text{out}}(\mathcal{S}_{\text{in}})$ is defined by the simple polygons defined by the bounded faces of L. The outer layer index out-index($\mathcal{S}_{\text{in}}$) = index(L).

inner layer: $Q_{\text{in}}(\mathcal{S}_{\text{in}}) = Q_{\text{in}}(\mathcal{S})$, with in-index($\mathcal{S}_{\text{in}}$) = in-index($\mathcal{S}$).

free points: $P(\mathcal{S}_{\text{in}})$ is the subset of $P(\mathcal{S})$ in the bounded faces of L.

boundary annotations: Carry over from $\mathcal{S}$ and the separator layered separator L.

annotation system: This is exactly the same annotation system as for $\mathcal{S}$.

Note that if $Q_{\text{in}}(\mathcal{S})$ is empty then $Q_{\text{in}}(\mathcal{S}_{\text{in}})$ will be empty as well. We only want to count triangulations T in $\mathcal{S}_{\text{out}}$ that have all their layers larger than $\lfloor \frac{n}{m} \rfloor$.

▶ **Definition 27.** Given a simple layer-unconstrained ring subproblem $\mathcal{S}$ and an peripheral layer separator $L \in \mathcal{L}(\mathcal{S})$, we define a set of outer constraints $\mathcal{C}_{\text{out}}$ as follows. We say $(i_1, \ldots, i_n) \in \mathcal{C}_{\text{out}}$ if the following conditions on i_j are satisfied for all j.

1. $i_j = |Q_{\text{out}}(\mathcal{S})|$, in case that $j = $ out-index($\mathcal{S}$).
2. $i_j = |V(L)|$, in case that $j = $ index(L).
3. $i_j = 0$, in case that $j \notin [\text{out-index}(\mathcal{S}), \text{index}(L)]$.
4. $i_j \geq \lfloor \frac{n}{m} \rfloor$, if $j \in [\text{out-index}(\mathcal{S}) + 1, \text{index}(L) - 1]$.

The following lemma describes a recursion and asserts its correctness.

▶ **Lemma 28** (Correctness SPLITBYLAYERS)**.** *Let $\mathcal{S}$ be a simple layer-unconstrained ring subproblem and L some separation layer of $\mathcal{S}$. Then $\mathcal{S}_{\text{out}}$ and $\mathcal{S}_{\text{in}}$ are layer-unconstrained ring subproblems, in particular $\mathcal{S}_{\text{out}}$ is again simple. It holds*

$$t(\mathcal{S}) = \sum_{L \in \mathcal{L}(\mathcal{S})} t(\mathcal{S}_{\text{in}}(L)) \cdot \left(\sum_{c \in \mathcal{C}_{\text{out}}} t(\mathcal{S}_{\text{out}}(L, c)) \right).$$

Proof. By definition, $\mathcal{S}_{\text{out}}$ and $\mathcal{S}_{\text{in}}$ are layer-unconstrained ring subproblems and it holds that $\mathcal{S}_{\text{out}}$ is simple as it has the same outer layer as $\mathcal{S}$. It remains to show the recursion.

"$\leq$" Given a triangulation T of $\mathcal{S}$, by Lemma 24, the graph L_{i^*} satisfies all conditions of Definition 24 and we can split $\mathcal{S}$ as described into two layer-unconstrained subproblems $\mathcal{S}_{\text{in}}$ and $\mathcal{S}_{\text{out}}$. The triangulation T decomposes naturally into two valid triangulations T_{in} and T_{out}. In particular there exists exactly one $c \in \mathcal{C}_{\text{out}}$ such that T_{out} is a valid triangulation of $\mathcal{S}_{\text{out}}(c)$. We check explicitly all conditions of Definition 10 to show that T_{out} and T_{in} are indeed valid triangulations. Recall that Condition 1 only asks that a valid triangulation has no crossings or edges outside the free region. Thus Condition 1 is trivially true by the way the triangulations are defined. Recall that Condition 2 asks for the inner layer of the subproblem to coincide with the most inner layer of the triangulation with the correct index. As L_{i^*} is defined to be the first small layer. It is in particular the inner layer of $\mathcal{S}_{\text{out}}$ with index i^*. In other words Condition 2 is satisfied for T_{out}. By the way we defined cactus layers, the layer structure of T_{in} is inherited by T and Condition 2 is also satisfied for T_{in}. Recall Condition 3 asks that each layer has the size given by the layer-constraint vector. We have defined the layer-constraint vector c to do match the size of each layer of T. Thus also this condition is satisfied. At last Condition 4 requires that each triangle is feasible. This

follows for T_{out} and T_{in} from the fact that all their empty triangles are triangle of T and the fact that T satisfies Condition 4. Thus T_{out} and T_{in} are valid triangulations.

Also note that a different triangulation $T' \neq T$ would lead to a different pair of triangulations $(T'_{\text{out}}, T'_{\text{in}}) \neq (T_{\text{out}}, T_{\text{in}})$.

"$\geq$" Assume we have given some separator layer $L \in \mathcal{L}(\mathcal{S})$ and some constraint $c \in \mathcal{C}_{\text{out}}$. Further, let T_{in} be a valid triangulation of $\mathcal{S}_{\text{in}}$ and T_{out} some valid triangulation of $\mathcal{S}_{\text{out}}(c)$. Then the union of T_{out} and T_{in} forms a valid triangulation T of $\mathcal{S}$. As $\mathcal{S}$ has no constraints there are only three conditions to be checked.

Condition 1 is again trivially true as T_{out} and T_{in} are nested and thus cannot cross each other. Condition 2 follows from the fact that the layers of T are exactly the layers of T_{out} and T_{in}. Thus Condition 2 follows for T because it was true for T_{in} and the fact that the inner layer of $\mathcal{S}_{\text{in}}$ and $\mathcal{S}$ coincide. We don't need to satisfy Condition 3 as $\mathcal{S}$ is layer-unconstrained. Condition 4 follows again from the fact each triangle of T is a triangle of either T_{out} or T_{in}.

Also note that a different pair of triangulations $(T'_{\text{out}}, T'_{\text{in}}) \neq (T_{\text{out}}, T_{\text{in}})$ would lead to a different triangulation $T' \neq T$. ◄

▶ Remark. We define a *disk subproblem* as a ring subproblem with an empty inner layer. Coincidentally, whenever the SPLITBYLAYERS procedure is applied, it is applied to a disk problem rather than an layer-unconstrained ring subproblem. So big parts of the algorithm do not need ring subproblems in full generality. We could nevertheless not avoid having non-empty inner layers and have to define ring subproblems anyway, so we decided to present an algorithm that is capable of solving ring subproblems. Another reason for this decision is that the additional complication is negligible.

We also want to be able to simplify ring subproblem $\mathcal{S}$, this is to split $\mathcal{S}$ into ring subproblems $\mathcal{S}_1, \ldots, \mathcal{S}_a$ in such a way that the outer layer of each $\mathcal{S}_i$ consists of only one polygon. This changes the problem not geometrically, but only composes it to smaller parts. Luckily, we have to do that only for layer-unconstrained subproblems. It can be done also for layer-constrained problems, but is technically more demanding and requires the extra condition of small width.

Let $\mathcal{S}$ be a layer-unconstrained ring subproblem with outer layer $Q_{\text{out}} = q_1, \ldots, q_a$, with $a \geq 2$. We define *layer-unconstrained* ring subproblems $\mathcal{S}_1, \ldots, \mathcal{S}_a$ as follows: $\mathcal{S}_i$ consists of q_i and all connected components of $Q_{\text{in}}(\mathcal{S})$ that are contained in q_i and likewise $P(\mathcal{S}_i)$ consists of all free points of $P(\mathcal{S})$ that are inside q_i.

▶ **Lemma 29** (Correct Counting of SPLITBYCOMPONENTS). *Given a layer-unconstrained ring subproblem $\mathcal{S}$ and assume it is split into $\mathcal{S}_1, \ldots, \mathcal{S}_a$. Then $\mathcal{S}_1, \ldots, \mathcal{S}_a$ are all simple layer-unconstrained ring subproblem and it holds:*

$$t(\mathcal{S}) = t(\mathcal{S}_1) \cdot \ldots \cdot t(\mathcal{S}_a).$$

Proof. Any valid triangulation T of $\mathcal{S}$ decomposes naturally into several valid triangulations $T_1, \ldots, T_a$ of $\mathcal{S}_1, \ldots, \mathcal{S}_a$ respectively. Just restrict T to $\mathcal{S}_i$ to get T_i.

Similarly the union of any collection of valid triangulations $T_1, \ldots, T_a$ of $\mathcal{S}_1, \ldots, \mathcal{S}_a$ is a valid triangulation of $\mathcal{S}$.

Note that the outerplanar index of T_i is at most the outer planar index of T, but it could be smaller. In the later case the inner layer of $\mathcal{S}_i$ is empty. This is the reason, it is convenient that the width is only an upper bound on the number of layers. (See Condition 2 of Definition 10.) ◄

We are now ready for the proof of Theorem 11.

Algorithm 2 PEELING: Counting layer-unconstrained ring problems

---

Require: Layer-unconstrained Ring Subproblem $\mathcal{S}$
 1: Initialize search tree τ to store all computed values.
 2: **return** COUNT($\mathcal{S}$)

 3: **function** COUNT($\mathcal{S}$)
 4:     **if** $\mathcal{S}$ in τ **then**
 5:         **return** $\tau(\mathcal{S})$
 6:     **else if** $\mathcal{S}$ simple layer-unconstrained ring subproblem and $w(\mathcal{S}) \geq m + 2$ **then**
 7:         result = SPLITBYLAYERS($\mathcal{S}$)
 8:     **else if** $\mathcal{S}$ layer-unconstrained non-simple ring subproblem **then**
 9:         result = SPLITBYCOMPONENTS($\mathcal{S}$)
10:     **else if** $\mathcal{S}$ simple ring subproblem and $w(\mathcal{S}) \leq m + 1$ **then**
11:         result = NIBBLING($\mathcal{S}$)
12:     **end if**
13:     insert ($\mathcal{S}$, result) into τ
14:   **return** result
15: **end function**

16: **function** SPLITBYCOMPONENTS($\mathcal{S}$)
17:     Split $\mathcal{S}$ into $\mathcal{S}_1, \ldots, \mathcal{S}_a$
18:     **return** COUNT($\mathcal{S}_1$) $\cdot \ldots \cdot$ COUNT($\mathcal{S}_a$)
19: **end function**

20: **function** SPLITBYLAYERS($\mathcal{S}$)
21:     $s_1 = 0$
22:     **for** peripheral separation layer $L \in \mathcal{L}(\mathcal{S})$ **do**
23:         $s_2 = 0$
24:         Define $\mathcal{S}_{\text{out}}(L)$ and $\mathcal{S}_{\text{in}}(L)$ by splitting $\mathcal{S}$ using L
25:         **for** constraint vector $c \in \mathcal{C}_{\text{out}}(L)$ **do**
26:             $s_2 = s_2 + $ COUNT($\mathcal{S}_{\text{out}}(L, c)$)
27:         **end for**
28:         $s_1 = s_1 + $ COUNT($\mathcal{S}_{\text{in}}(L)$) $\cdot s_2$
29:     **end for**
30:     **return** s_1
31: **end function**
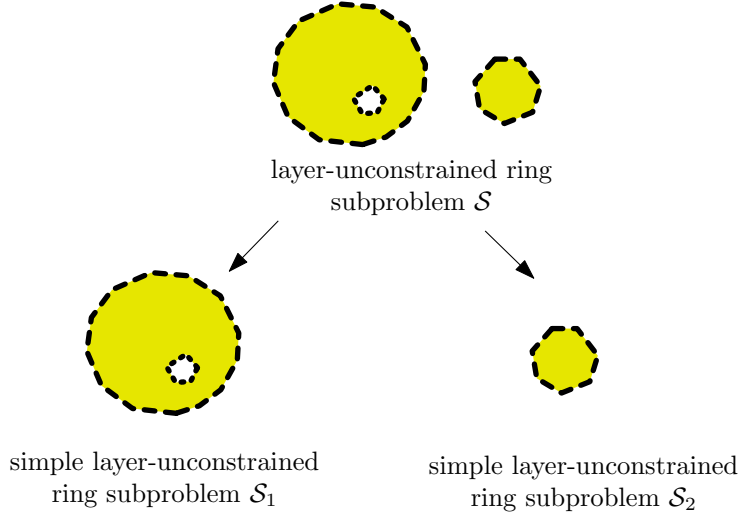
Figure 15 Illustration of the procedure SPLITBYCOMPONENTS.

Proof of Theorem 11. We will first describe the PEELING algorithm, then argue its correctness and finally supply a runtime analysis.

The Algorithm PEELING uses the memoization technique and is based on dynamic programming. The subproblems of the dynamic programming scheme are the ring subproblems. Each ring subproblem is solved recursively, by either the SPLITBYLAYERS, SPLITBYCOMPONENTS or NIBBLING procedure. Which procedure will be called depends on the properties of the subproblem. Already computed solutions are stored in and retrieved from the search tree τ. This prevents repeated computations. The pseudocode of PEELING is depicted as Algorithm 2.

To see correctness, we have to show that one of the cases in the main routine COUNT will be called. It is clear that each routine works correctly by Lemma 28 and 29 and Theorem 12. To be more precise COUNT will not have a valid value for the variable result if COUNT will be called with a non-simple constrained ring subproblem or with a constrained simple ring subproblem with width $w \geq m + 2$. So we have to show that this will not happen. Initially, COUNT will be called by a layer-unconstrained ring subproblem. The subroutine NIBBLING does not call COUNT at all. The subroutine SPLITBYCOMPONENTS does call COUNT only with layer-unconstrained simple ring subproblems. The subroutine SPLITBYLAYER defines a simple outer subproblem $\mathcal{S}_{\text{out}}$ and calls calls COUNT with it. But this is fine as the width of $w(\mathcal{S}_{\text{out}}) \leq m$. The inner subproblem $\mathcal{S}_{\text{in}}$ is layer-unconstrained. This covers all cases and shows correctness.

In order to bound the running time, we will bound the total number of possibly occurring subproblems. Further, we bound the total costs for each subproblem. This is either the total number of recursive calls or the time spend by NIBBLING. As a first step we bound the number of annotated cactus graphs on l points in the plane. Given a cactus graph on l points Q in the plane, it is well known that G has at most $2l - 3$ edges. Thus there are trivially at most

$$\binom{l^2}{2l - 3} + \binom{l^2}{2l - 4} + \ldots + \binom{l^2}{1} + \binom{l^2}{0} \leq l^{4l-6} + l^{4l-8} + \ldots + 1 \leq l^{4l}$$

cactus graphs on Q. Given a set P of n points there are

$$\binom{n}{l} + \binom{n}{l-1} + \ldots + \binom{n}{0} \leq n^l + n^{l-1} + \ldots + 1 \leq n^{l+1}$$

point sets Q of size at most l. Thus there are at most $n^l \cdot l^{4l}$ cactus graphs of size at most l on a set of n points. There are at most $|L|^{3l-3}$ ways to annotated all l vertices and $2l - 3$ edges. Given a set S of points, every ring subproblem on S is completely defined by the outer layer, the inner layer, and the inner layer and outer layer index. If we choose $m = \lfloor \sqrt{n} \rfloor$ the total number of possible annotated inner layers equals $n^{(3+o(1))\sqrt{n}} \cdot |L|^{(3+o(1))\sqrt{n}}$ and the total number of outer layers is bounded by the same number. Note that the boundary of the initial layer-unconstrained ring subproblem might also appear as outer layer in some subsequent subproblems, but this is the only possible outer layer with potentially more than $\lceil \frac{n}{m} \rceil$ vertices. Thus the total number of subproblems is $n^{(6+o(1))\sqrt{n}} \cdot |L|^{(6+o(1))\sqrt{n}}$. The time to solve a subproblem called by NIBBLING takes $n^{(5+o(1))\sqrt{n}} \cdot |L|^{(6+o(1))\sqrt{n}}$ time by Theorem 12. The number of recursive calls by the procedure SPLITBYCOMPONENTS is polynomial in n and does not depend on L. The number of recursive calls by the procedure SPLITBYLAYERS equals $n^{(4+o(1))\sqrt{n}} \cdot |L|^{(3+o(1))\sqrt{n}}$. To see this recall that the set $\mathcal{L}(\mathcal{S})$ has size $n^{(3+o(1))\sqrt{n}} \cdot |L|^{(3+o(1))\sqrt{n}}$ and the set $\mathcal{C}_{\text{out}}(L)$ has size $n^{(1+o(1))\sqrt{n}}$. Thus the total running time is bounded by $n^{(11+o(1))\sqrt{n}} \cdot |L|^{(12+o(1))\sqrt{n}} = n^{O(\sqrt{n})} \cdot |L|^{O(\sqrt{n})}$. ◄

A more careful analysis would probably give a slightly better constant. Possible improvements might be based on a different design of the algorithm, careful choice of m and an improved estimate on the number of cactus graphs on n vertices.

5 Applications for Counting other Structures

In this section, we give a complete description of the framework discussed in Section 5 for counting non-crossing straight line graphs. In Section 5.1, we explain formally how to go from counting triangulations to colored graph classes. As we show how to do this in full generality and not just for a specific graph class, the description becomes a little abstract and technical. We use the example of straight-line perfect matchings to illustrate these definitions. We attain the following result.

▶ **Theorem 30** (Counting Perfect Matchings). *Given a set S of n points in the plane, there exists an algorithm that counts the total number of non-crossing perfect straight line matchings in $n^{O(\sqrt{n})}$ time.*

Another key result of this section is an operation $\oplus$ that can be used to combine annotation systems in order to get the intersection of the graph classes described by them.

In Section 5.2, we apply this framework to a rich selection of important graph classes. We describe annotation systems for them and prove their correctness. This serves to show the power of the framework and also explains how to use the framework. In particular, we hope that certain tricks to design annotation systems become clear and might be helpful to design annotation systems for graph classes not covered by our examples.

We start with the proof of Theorem 6 about counting 3-colorable triangulations.

Proof of Theorem 6. Given a triangulation T that is 3-colorable, this coloring is unique up to a permutation of the colors. It becomes unique as soon as the coloring of two adjacent vertices are fixed. Given a point set S an edge $e^* = vw$ on $\partial CH(S)$, we define a list L_{3c} of feasible empty triangles such that for each triangle Δ holds

1. All three colors (red, yellow, blue) appear exactly once on the vertices of Δ.

2. v is always red and w is always yellow.

We denote by $\mathcal{T}_3$ the set of 3-colorable triangulations on S and with $\mathcal{T}^A(L_{3c})$ the set of annotated triangulations such that each triangle belongs to the list of L_{3c}. It holds

$$|\mathcal{T}^A(L_{3c})| = |\mathcal{T}_3|.$$

The size of this annotation system can be upper bounded by $|L_{3c}| \leq \binom{n}{3} \cdot 6! = O(n^3)$. Now, we apply Theorem 5 to count all triangulations with this annotation system. ◄

5.1 Tools

In this section, we try to develop some general tools for annotation systems. Some of the definitions are a little abstract, so we try to illustrate our definitions with the example of perfect matchings. In this section, we show how to count structures that are not triangulations. We do this with the help of constrained Delaunay triangulations. We are able to abstract out our arguments so that the reader will not have to worry about constrained Delaunay triangulation ever again after this section. For this goal we develop a machinery to combine annotation systems, which is useful in its own respect.

▶ **Definition 31.** Let S be a set of n points in the plane and let c_V and c_E be two integers. We define $\mathcal{G}_{\text{color}}$ to be the set of non-crossing straight line graphs on S, such that

1. each vertex has a color $i \in \{1, \ldots, c_V\}$ and

2. each edge has a color $i \in \{1, \ldots, c_E\}$.

For later convenience, we assume that none of the colors is denoted by purple: the color purple is reserved for "non-edges." We call subsets $\mathcal{C} \subseteq \mathcal{G}_{\text{color}}$ a *classes of non-crossing colored graphs*. We also use the term *property* to describe a class of colored graphs; we say G has property $\mathcal{P}$, if $G \in \mathcal{P}$.

An example of a colored graph class is the class of non-crossing straigth line perfect matchings $\mathcal{M}$ on S. This class has only one color, say orange. We want to keep this example in mind for the forthcoming somewhat technical definitions.

As the algorithm of Theorem 5 counts annotated triangulations, we have to triangulate the matchings and we have to do it in a unique way. We will follow the ideas of previous authors and use the constrained Delaunay triangulation for this purpose. See [3] and [5] on how constrained Delaunay triangulations were used earlier in order to count perfect matchings and other crossing-free structures. The text book by Hjelle and Dæhlen gives a detailed introduction to constrained Delaunay triangulations [33] (Chapter 6).

▶ **Definition 32** (Constrained Delaunay Triangulation)**.** Let $G \in \mathcal{G}_{\text{color}}$ be a graph on a set S of n points and a let T be a triangulation extending G on the same set of points. We say an edge e in a triangulation T satisfies the *Delaunay Condition* if the circumference of neither adjacent triangle contains the other adjacent triangle, see Figure 16. 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*. We also demand that no edge $e \in E(G)$ is colred purple and every other edge is colored purple. (In case that e is on the boundary of the convex hull of S, then we say that e satisfies the Delaunay Condition as well.) We denote by $CDT(\mathcal{P})$ the set of triangulations such that all purple edges satisfy the Delaunay Condition and the remaining colored graph belongs to $\mathcal{P}$.
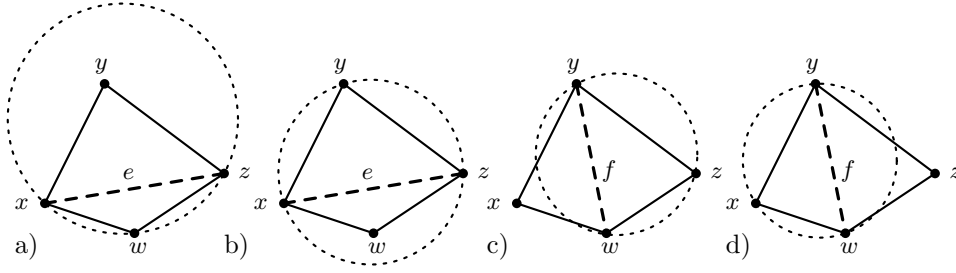
Figure 16 Diagonal e of $xyzw$ does not satisfy the Delaunay condition: as shown in a) and b), circle xzw contains xyz and circle xyz contains triangle xzw. On the other hand, diagonal f does satisfy the Delaunay condition: as shown in c) and d), circle yzw does not contain triangle xyw and circle xyw does not contain triangle yzw.

See Figure 17 for an example of a constrained triangulation of a matching. It is noteworthy that the Delaunay condition is symmetric. That is, let Δ_1 and Δ_2 be the two triangles adjacent to the edge e and C_1 and C_2 be the respective circumferences. Then it holds that C_1 contains Δ_2 if and only if C_2 contains Δ_1. We assume that there are no 4 points on a common circle. This can be achieved in polynomial time by a small affine transformation. The first algorithm to compute the constrained Delaunay triangulation in optimal $O(n \log n)$ time was given by Chew [11]. We will use the fact that the constrained Delaunay triangulation is unique.

▶ **Theorem 33** ([33]). *Given $G \in \mathcal{G}$, there exists exactly one constrained Delaunay triangulation T extending G.*

We get the following corollary.

▶ **Corollary 34** (folklore). *For every point set S and every property $\mathcal{P}$ of colored graphs of S, we have $|\mathcal{P}| = |CDT(\mathcal{P})|$.*

Let us return to perfect matchings, that is, the property $\mathcal{M}$. The way we count $\mathcal{M}$ is to count $CDT(\mathcal{M})$. The most straightforward way would be to define an appropriate annotation system enforcing the matching property for the orange edges and the Delaunay Condition for the purple edges. However, we will go a different path and define two annotation systems. Later on, we will show how two annotation systems can be combined to a new one.

▶ **Definition 35** (Annotations for Matchings). We want to define the annotation system L_{matching} in a way that all the orange edges are forced to form a matching. Consider a triangulation, where the orange edges form a perfect matching and imagine that every vertex v is labeled by the unique vertex $w \neq v$ that is incident to the same orange edge. We define L_{matching} to be the set of all annotated triangles arising this way. In particular, for every triangle Δ that annotates vertex v with w, Δ must not block visibility between v and w. It is easy to see that the orange edges of any valid triangulation annotated by L_{matching} must form a perfect matching.

▶ **Definition 36** (Annotations for Delaunay-Condition). We want to define the annotation system L_{CDT} in a way that all the purple edges satisfy the Delaunay Condition. Consider a triangulation, where the purple edges satisfy the Delaunay Condition and imagine that every edge e is labeled by the unique pair of vertices v_1 and v_2 that are incident to the same
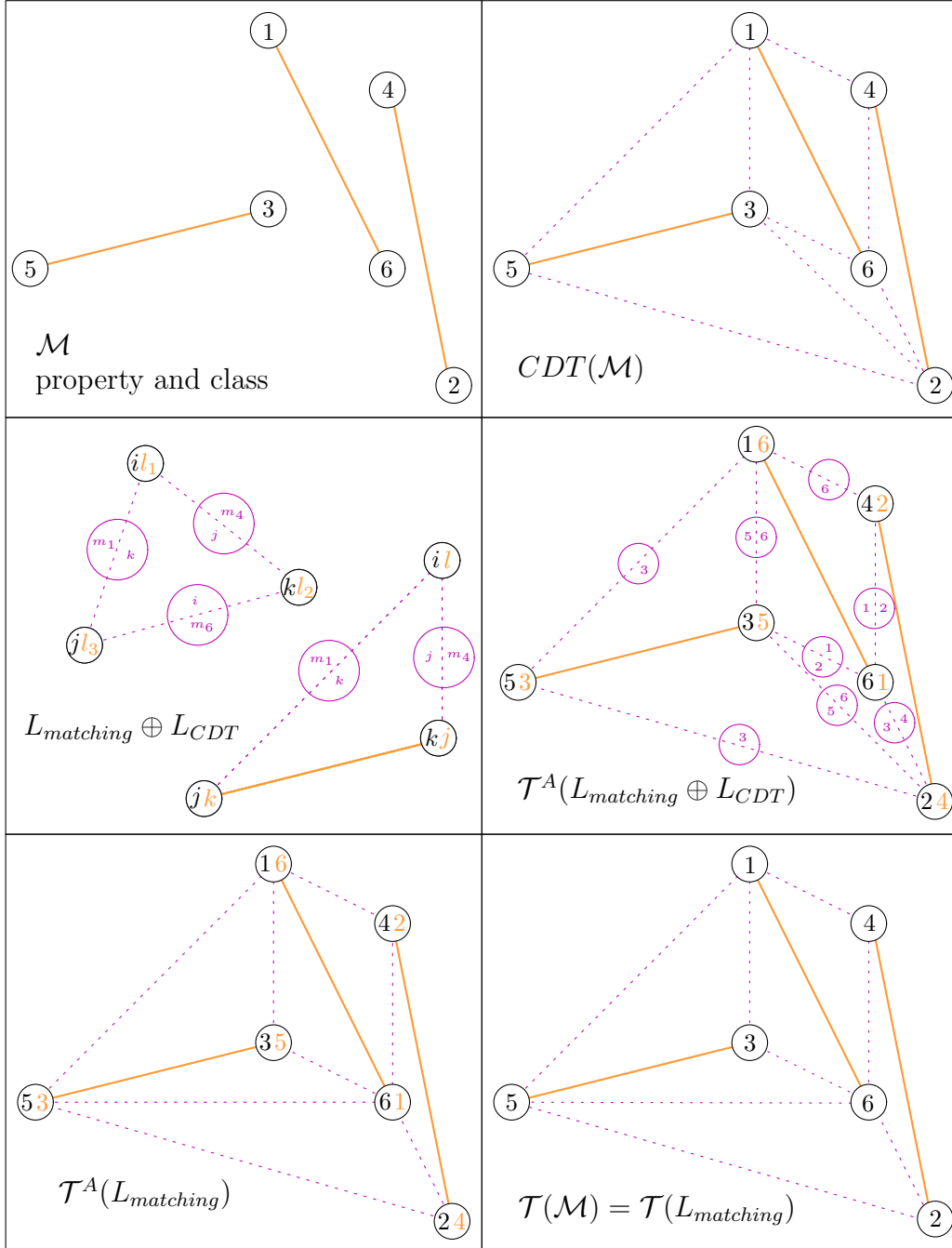
Figure 17

triangles as e. (There might be only one such vertex in case that e on $\partial CH(S)$.) We define L_{CDT} to be the set of all annotated triangles arriving in this way. It is easy to see that the purple edges of any valid triangulation annotated by L_{CDT} must satisfy the Delaunay Condition. It is easy to see that $|L_{\mathrm{CDT}}| \leq n^6$, as there are at most $\binom{n}{3}$ empty triangles and at most n annotations are possible for each edge of each triangle (Note that there are two labels on an edge, but one of them is determined by the triangle itself).

We deliberatley do not specify the number of colors. We leave this open for later, as the description works for different number of colors.

There are several advantages of defining two annotation systems instead of just one. The first advantage is that we have to think only about one property at a time to show correctness. The second advantage is that we can reuse annotation systems, which is particularly interesting for L_{CDT}. The following definitions serve to define the combination of two annotation systems.

▶ **Definition 37** (Color Annotation)**.** Given integers c_V and c_E, we define the *complete color annotation* L_{color} of a point set S as the set of all empty triangles Δ such that

1. each vertex is annotated by a number $i \in \{1, \ldots, c_V\}$ and

2. each edge is annotated by a number $i \in \{1, \ldots, c_E\}$.

We refer to these numbers usually as colors and us names like "red," "magenta," "salmon" and so on. In case c_V or c_E equals 0, we say that the vertices or edges are are not colored. We also call these feasible triangles *feasibly colored triangles*. The annotation system L_{color} has size $|L_{\mathrm{color}}| \leq \binom{n}{3} c_V^3 c_E^3$.

Note that we strictly distinguish between an edge colored by a specific color and an edge having the annotation of that color. Nevertheless, the intention of an annotation with a certrain color is to interpret this as that color.

The set $\mathcal{T}^A(L_{\mathrm{color}})$ is exactly the set of triangulations of S where each vertex and each edge is colored with a color from $\{1, \ldots, c_V\}$ and $\{1, \ldots, c_E\}$ respectively.

▶ **Definition 38** (Extensions)**.** We say that an annotation system L is an *extension of a color annotation* if each annotation consists of two components and L restricted to the first component is a subset of the color annotation. The first component is also called the *color component*. In the following, we restrict ourselves of annotation systems that are extensions of the color annotation L_{color}.

We assume that the numbers c_V and c_E are implicitly clear. Usually we omit to specify them as the description for the annotation system would work for various values of c_V and c_E. Further, we will assume that c_V and c_E are constants and thus they only add a multiplicative constant to the total size of any extension of the color annotation.

Going back to our matching example, annotation system L_{matching} can be regarded as an extension of L_{color}, where we have two edge colors and no vertex color. We denote the two colors with orange and purple.

▶ **Definition 39** (Annotations Describing Properties)**.** Given an annotation system L, it defines a set of valid annotated triangulations $\mathcal{T}^A(L)$. When we restrict the triangulations $T \in \mathcal{T}^A(L)$ to the colored component, we receive a *multi-set* of colored triangulations denoted by $\mathcal{T}_L$. To do this, we interpret the color annotation as the color itself. If each colored triangulation appears only once, it holds $\mathcal{T}(L) \subseteq \mathcal{G}_{\mathrm{color}}$. Given a property $\mathcal{P}$, we define $\mathcal{T}(\mathcal{P})$ to be the set of all those triangulations T such that the graph G_T that remains

after removing all purple edges satisfies $\mathcal{P}$. We say that an annotation system L *describes* a property $\mathcal{P}$, if $\mathcal{T}(\mathcal{P}) = \mathcal{T}(L)$ holds. (Note that we assume here that L has one edge color more than $\mathcal{P}$. This additional color is denoted by purple.)

▶ **Definition 40** (Combining Annotations)**.** Let L_1 and L_2 be annotation systems extending L_{color}. We define

$$L = L_1 \oplus L_2$$

as the set of triangles such that each annotation of a vertex and edge has three components (c_1, c_2, c_3) satisfying the following properties:

1. each triangle restricted to the first component c_1 is feasibly colored,

2. each triangle restricted to the first and second component (c_1, c_2) belongs to L_1, and

3. each triangle restricted to the first and third component (c_1, c_3) belongs to L_2.

In the following, whenever we will use the $\oplus$ operation, we will *implicitly* assume that both annotation systems extend the same colored annotation L_{color}. Then L can be regarded again as an extension of L_{color} by regarding the second and third component as one "meta-component."

▶ **Theorem 41.** *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two properties described by L_1 and L_2 respectively. Then $\mathcal{P}_1 \cap \mathcal{P}_2$ is described by $L_1 \oplus L_2$, and it holds $|L_1 \oplus L_2| \leq |L_1| \cdot |L_2|$.*

Proof. We will show

$$\mathcal{T}(\mathcal{P}_1 \cap \mathcal{P}_2) \overset{(1)}{=} \mathcal{T}(\mathcal{P}_1) \cap \mathcal{T}(\mathcal{P}_2) \overset{(2)}{=} \mathcal{T}(L_1) \cap \mathcal{T}(L_2) \overset{(3)}{=} \mathcal{T}(L_1 \oplus L_2).$$

We start with (1). Let $T \in \mathcal{T}(\mathcal{P}_1 \cap \mathcal{P}_2)$ and G be the graph remaining after removing all purple edges. Then G satisfies $\mathcal{P}_1$ and $\mathcal{P}_2$. Thus $T \in \mathcal{T}(\mathcal{P}_1)$ and $T \in \mathcal{T}(\mathcal{P}_2)$. This shows "$\subseteq$". Conversely, let $T \in \mathcal{T}(\mathcal{P}_1) \cap \mathcal{T}(\mathcal{P}_2)$ and G be the graph that is left after removing all purple edges. Then G satisfies $\mathcal{P}_1$ and $\mathcal{P}_2$ and thus T belongs to $\mathcal{T}(\mathcal{P}_1 \cap \mathcal{P}_2)$. This shows (1).

Statement (2) follows immediately from the assumption $\mathcal{T}(\mathcal{P}_i) = \mathcal{T}(L_i)$, for $i = 1, 2$.

We finish with (3). Let $T \in \mathcal{T}(L_1) \cap \mathcal{T}(L_2)$ be a colored triangulation. Then there exists two annotations A_1 and A_2 such that T annotated with A_i belongs to $\mathcal{T}^A(L_i)$, for $i = 1, 2$. Now, we annotate T with A_1 *and* A_2 carefully. So let T' be an annotated triangulation such that the first component represents its color, the second component is given by A_1 and the third component is given by A_2. By definition T' belongs to $\mathcal{T}^A(L_1 \oplus L_2)$. Thus T belongs to $\mathcal{T}(L_1 \oplus L_2)$. This shows "$\subseteq$". The reverse direction is easier. Let $T \in \mathcal{T}(L_1 \oplus L_2)$ be a colored triangulations corresponding to the annotated triangulation $T' \in \mathcal{T}^A(L_1 \oplus L_2)$. Then by definition of $\oplus$, T' restricted to the second component belongs to $\mathcal{T}^A(L_1)$ and T' restricted to the third component belongs to $\mathcal{T}^A(L_2)$. Thus T' belongs to $\mathcal{T}^A(L_1) \cap \mathcal{T}^A(L_2)$.

The size bound follows easily from the construction. ◀

▶ **Lemma 42.** *Let $\mathcal{P}$ be a property described by an annotation system L. Then*

$$|\mathcal{P}| = |\mathcal{T}^A(L \oplus L_{\text{CDT}})|.$$

Proof. From Corollary 34 follows $|\mathcal{P}| = |CDT(\mathcal{P})|$. We know that $\mathcal{T}(L)$ contains precisely the graphs in $\mathcal{P}$ and thus $\mathcal{T}^A(L)$ represents precisely all possible triangulations of all graphs of $\mathcal{P}$ with purple edges. Further, we know that all purple edges of $\mathcal{T}^A(L \oplus L_{\text{CDT}})$ satisfy the Delaunay Condition, which implies, that each graph in $\mathcal{P}$ is uniquely triangulated. ◀

▶ **Theorem 43** (Counting Colored Graph Classes). *Let $\mathcal{P}$ be a colored graph class and L an annotation system describing it. There exists an algorithm that counts the cardinality of $\mathcal{P}$ in $n^{O(\sqrt{n})} \cdot |L|^{O(\sqrt{n})}$.*

Proof. In order to count the members of $\mathcal{P}$, it is sufficient to count $\mathcal{T}^A(L \oplus L_{\text{CDT}})$. It remains to prove the upper bound on the running time. From Theorem 5, we have the bound of $n^{O(\sqrt{n})} \cdot |L \oplus L_{\text{CDT}}|^{O(\sqrt{n})}$ on the running time. Note that $|L \oplus L_{\text{CDT}}| \leq |L| \cdot |L_{\text{CDT}}| \leq |L| \cdot n^6$. Thus the whole running time is bounded by $n^{O(\sqrt{n})} \cdot |L|^{O(\sqrt{n})} \cdot |L_{\text{CDT}}|^{O(\sqrt{n})} \leq n^{O(\sqrt{n})} \cdot |L|^{O(\sqrt{n})}$. ◄

To see the usefulness of Theorem 43, observe that now the proof of Theorem 30 boils down to the fact that L_{matching} describes the class of orange perfect matchings and has polynomial size.

5.2 Examples

▶ **Definition 44** (Degree Constrains). Given a family $\mathcal{D} = \{D_v : v \in V\}$ of sets $D_v \subseteq \{1, \ldots, n\}$. We say a triangulation *satisfies the degree constraint* given by $\mathcal{D}$ if $\deg(v) \in D_v$ for each $v \in V$. And a triangulation T satisfying the degree constrained imposed by $\mathcal{D}$ is called *degree constrained triangulation.*

We say a colored graph *satisfies the colored degree constraint in color i* given by $\mathcal{D}$ if $\deg_i(v) \in D_v \in \mathcal{D}$ for each $v \in S$ (The expression $\deg_i(v)$ denotes the number of edges in color i incident to v.). And a graph G satisfying the colored degree constraints imposed by $\mathcal{D}$ is called *colored-i degree constrained graph.*

▶ **Lemma 45.** *There are polynomial sized annotation systems for the following colored non-crossing graph classes:*

1. *L_{triang} colored triangulations*

2. *L_{TriDeg} degree constrained triangulations*

3. *L_{Deg} colored degree constrained non-crossing graphs*

4. *L_{SpanTree} red non-crossing spanning tress*

5. *L_{Connect} blue connected non-crossing graphs*

6. *L_{Size} k-green vertices*

7. *L_{Ind} orange vertices form an independent set*

8. *L_{Dom} brown vertices form a dominating set*

9. *L_{FaceDeg} face-degree constrained non-crossing graphs*

Proof. The way, we describe the annotation systems always follows the same scheme. We start with a graph class $\mathcal{P}$ that we want to enforce. Then, we describe a procedure to annotate it in an *unambiguous* way. The annotation system is implicitly defined by all possible annotated empty triangles, that might arise in this way. We will *usually* not make it explicit, as it is only technical and not enlightening. See Figure 17 for an explicit example for perfect matchings. We have to show that the annotation system describes $\mathcal{P}$. For this step, it is necessary and sufficient that the annotation procedure is unambiguous and enforces the property $\mathcal{P}$. The last step is to give a size bound on the size of the annotation system.

Recall that the order label was attached to the points as a preprocessing step. This labeling helps to set priorities between vertices and makes possible choices unambiguous.
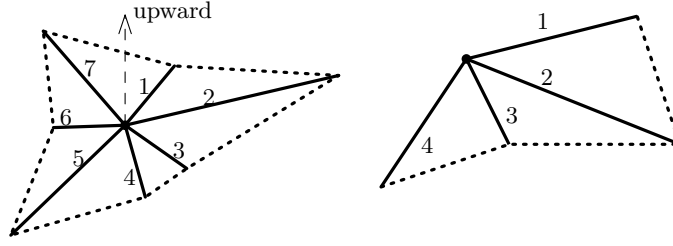
Figure 18
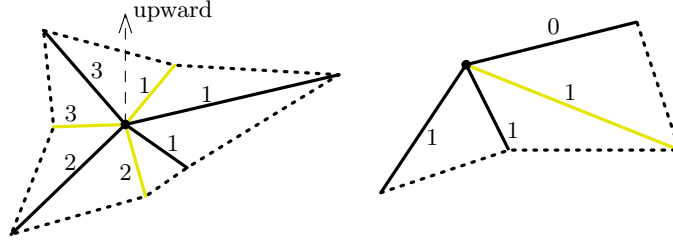
Figure 19

Colored Triangulations Define $L_{\text{triang}} \subseteq L_{\text{color}}$, as the list of all those annotated triangles, that do not use the color purple reserved for "non-edges".

Degree Constraints We start to show how to enforce the degree on one vertex. Let T be a triangulation and v a vertex of T with degree d. We annotate the edges around v in clockwise order with $1, \ldots, d$. We start with the edge, which comes first after the upward direction. This is unambiguous and enforces degree d. See also Figure 18. Note that the triangle containing the upward direction has the annotation d and 1 in clockwise order around v. In case that v is on $\partial CH(S)$, it is even simpler, as we start and end our counterclockwise numbering on $\partial CH(S)$. This is we force the edges incident to v to carry the annotations 1 and d respectively.

Given a set $D \subset \{1, \ldots, n\}$, we can take the union of all possible annotations enforcing a degree $d \in D$. The size of this annotation system is still pretty small, as every triangle adjacent to v gets added only two numbers, where the second number is defined by the first one.

Now, given a set $\mathcal{D}$ as in the definition of degree constrained triangulations, we can define an annotation for each vertex as above. We can combine all of these annotations without increasing the size too much, as every triangle has to remember only two numbers for each of its three vertices. This amounts to a total of $O(n^{3+6})$ many triangles.

Colored Degree Constraints We show how to enforce a yellow degree constrained on one vertex. The generalization to more possible degrees and all vertices simultaneously work in the same way as for uncolored degree constrained triangulations. Generalizing it to more colors can be done using the $\oplus$ operation. Given a colored graph G we can assume that G is completely triangulated by edges not belonging to G (purple edges). Let $v \in V(G)$ be a vertex with d yellow incident edges. We annotate each edge(also non-yellow edges) with the number of yellow edges in the counterclockwise interval between the upwards direction and this edge, including the edge itself, see Figure 19. In case that v is on $\partial CH(S)$, it is even simpler, as we start and end our counterclockwise numbering on $\partial CH(S)$.

If more than one color should be constraint the corresponding annotation systems can
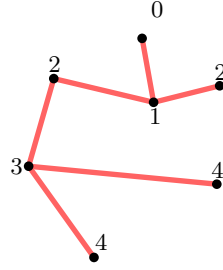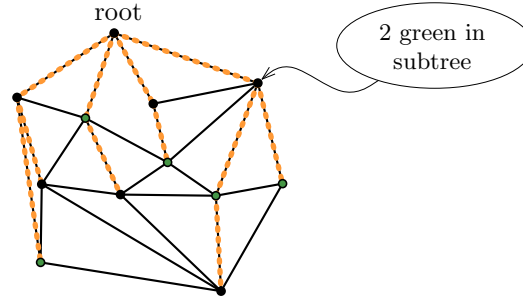
Figure 20
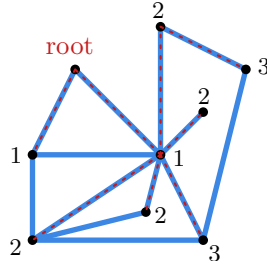
Figure 21

be combined with the $\oplus$ operator. Using the colors, the resulting size of the annotation system is $n^{O}(c)$.

Spanning Trees Let G be a red spanning tree of S. Then we can think of G as a rooted tree. We choose the root to be the vertex with lowest order label. We annotate each vertex v with the distance to the root and the label of its parent. It is easy to see that this gives an annotation in an unambiguous way. The set of feasible triangles defined by this annotation procedure does not include triangles Δ, where a vertex v is annotated with a vertex w, but the vision between v and w is blocked by Δ. We will show, it also enforces the red edges to form a spanning tree. Every vertex, except the root, has exactly one vertex with lower distance to the root. This is enforced by the label on the vertex. Further following these edges defines a path to the root. This shows connectedness. At last, we have to show that this annotation system excludes cycles. To see this, assume, for the purpose of contradiction there is a cycle C in G and let $v \in V(C)$ be the vertex with largest distance to the root. By the discussion above it has at most one neighbor with smaller distance to the root. a contradiction to the fact that every vertyex in C has two neighbors. See Figure 20.

To bound the size of L_{SpanTree} note that every vertex is annotated with two numbers and an upper bound of $O(n^{3+6})$ follows.

Connectedness Let G be a blue connected graph. We will define a spanning tree T_G in an unambiguous way. For this recall that we can assume that all vertices have an order label. This order label is just a number from 1 to n and gives arbitrary preferences. We will use it to single out a particular spanning tree among all spanning trees of G. It is important to keep in mind that this order label is *not* part of the annotation, but given beforehand.

The root of our spanning tree is the vertex with lowest order label. We annotate each vertex with the distance to the root. And we annotate each vertex with the neighbor that has the smallest order label among all neighbors in G closer to the root.

Figure 22

Now every vertex has a unique neighbor closer to the root and this neighborhood relation defines unambiguously a spanning tree of G. See Figure 21.

To avoid confusion let us be more explicit in this case, which properties annotated empty triangles of L_{Connect} must have. Well, every vertex is annotated with its distance to the root and its parent. Further, there better be a blue edge between each vertex v and its parent $p(v)$. Also the parent should have smaller order label. It is maybe more subtly to see, that there must not be a blue edge between a vertex v and another vertex w that has smaller order label than $p(v)$. Further, a triangle Δ is not feasible, if there exists a vertex $v \in \Delta$, which is annotated with a vertex w, but the vision between v and w is blocked by Δ.

The size of L_{Connect} is bounded by $O(n^{3+6})$.

Count Vertices Given a triangulation T on S, we want to ensure that T has exactly k vertices colored green. We ignore all other colors for this description. We have seen in **Connectedness** how we can identify a spanning tree U_T in T in an unambiguous way and find an annotation identifying U_T. Additionally, we annotate each vertex with the number of green vertices in its subtree. We can interpret this number as the weighted indegree within U_T. In order to keep the information consistent we use the annotation system of colored degree, to check that the actual indegree adds up to the number written on each vertex. See Figure 22.

Correctness follows from the correctness of the previous constructions. The polynomial size bound follow in the same way.

Independent Set Given a graph G colored green with some vertices being colored orange, forming an independent set. This means that there is no edge incident to two brown vertices.

Thus L_{Ind} consists of all feasible triangles except those which have two adjacent brown vertices.

Dominating Set Given a graph G colored green with some vertices being colored brown, forming a dominating set. We annotate every non-brown vertex with the adjacent brown vertex with smallest order label.

L_{Dom} consists of all feasible triangles arising in this way. In particular, consider the case that a vertex v is annotated with w. then v must not be part of a triangle Δ blocking visibility between v and w. Further, v should not be adjacent to a brown vertex with lower order label and any feasible triangle with v and w should color w brown in case that v is annotated with w.

The size of L_{Dom} is bounded by $O(n^{3+3})$, as every vertex is annotated with only one number.
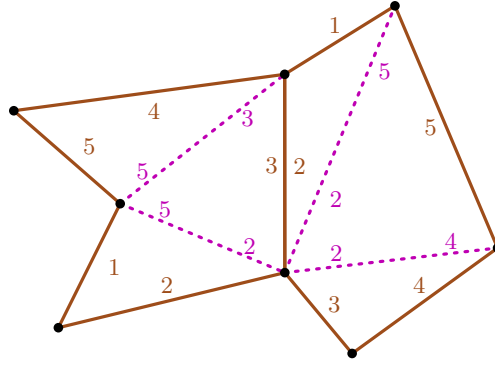
■ **Figure 23**

Face Degree Constraints Given a brown graph G with every bounded face of degree d. Here, we need again to make use of the triangulation T containing G. Consider one specific face f surrounded by the vertices $v_1, \ldots, v_d$ and edges $e_1, \ldots, e_d$ with $e_i = \{v_i, \ldots, v_{i+1}\}$ in counterclockwise order. Then there exists a vertex with lowest order label; without loss of generality assume it is v_1. Annotate every brown edge with 1. We annotate edge e_i with i, meaning that it is the i-th edge of this face. Further, we annotate the edges inside f around v_i with i. (These edges do not belong to G.) In this way, every brown edge receives three annotations(being incident to two faces) and every other edge receives also two annotations, as it is incident to two vertices.

The annotation system L_{FaceDeg} consists of all triangles arising in this way.

The size of L_{FaceDeg} is bounded by $O(n^{3+9})$, as every edge is annotated with at most three numbers. See Figure 23.

This finishes the proof. ◀

We are now ready to proof Theorem 8. For convenience, we restate the theorem.

▶ **Theorem 8** (Counting Geometric Structures). *The following non-crossing structures can be counted in $n^{O(\sqrt{n})}$ time 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.*

Proof. The proof is just a combination of annotation systems that are given in Lemma 45.

All Graphs To count the set of all graphs, we use an annotation system with exactly two edge colors and two vertex colors, namely red and purple. We enforce that all purple edges satisfy the constrained Delaunay Condition.

Perfect Matchings Enforce degree one on all vertices.

Cycle Decompositions Enforce that all vertices have degree two.

Hamilton cycles, Hamilton paths For the first ensure that all vertices have degree two and the graph is connected. For the second ensure that the graph has max degree two and is connected.

Euler-Tours Ensure that the graph has even degree.

Spanning Trees See Lemma 45.

d-**regular graphs** See Lemma 45.

Quadrangulations Enforce that each face has degree four and that the graph is spanning.
◀

▶ Remark (On the scope of the framework). The scope of the structures that we are capable to count is not limited by the examples we have given. It would be nice to encapture more precisely to which structures our procedure applies and to which it doesn't. However, there are some very simple structures that we were not able to give a descriptive annotation system for. Examples are 3-colorable graphs, graphs with diameter exactly k and graphs with degree constraints on more than constantly many colors. The problem of 3-colorable graphs is that there might be more than one 3-coloring and we don't know how to ensure that we count each graph only once. The diameter seems to be a too complex graph property. Known algorithms to compute the diameter usually compute the distance of all pairs of vertices. If someone finds a way how to compute the diameter of a planar graph in a simpler way, this algorithm might also lead to an appropriate annotation system. The reason, we don't know how to give degree constraints to more than a constant number of colors with the above running time is that, we have to maintain too many informations on the edges to keep the informations consistent.

▶ Remark (On the geometry). In general all structures that we are interested in are purely defined by combinatorial means and not so much by geometry. To be more precise: given a set of points S the *order type* of S is defined as an oracle that returns for every triple of points whether they are oriented in clockwise or counterclockwise order. This information is considered as combinatorial, as it does not depend on the concrete coordinates of the set of points. In particular there is only a finite number of possible order types for n points in the plane. Nevertheless, we make use of the geometry at several occasions, in order to make possible choices unique. It is not clear to us, if this could be avoided.

Acknowledgment

References

1 A. Adamaszek and A. Wiese. Approximation schemes for maximum weight independent set of rectangles. In *FOCS'13*, pages 400–409. IEEE Computer Society, 2013.

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

3 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.

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

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

6 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.

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

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

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

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

11 L. P. Chew. Constrained delaunay triangulations. *Algorithmica*, 4(1):97–108, 1989.

12 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.

13 K. Dalal. Counting the onion. *Random Struct. Algorithms*, 24(2):155–165, 2004.

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

15 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.

16 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.

17 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.

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

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

20 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.

21 H. Djidjev and S. M. Venkatesan. Reduced constants for simple cycle graph separation. *Acta Inf.*, 34(3):231–243, 1997.

22 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.

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

24 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.

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

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

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

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

29 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.

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

31 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.

32 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.

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

34 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.

35 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.

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

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

38 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.

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

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

41 R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980.

42 E. L. Lloyd. On triangulations of a set of points in the plane. In *FOCS'77*, pages 228–240.

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

44 D. Marx and T. Miltzow. Peeling and Nibbling the Cactus: Subexponential-Time Algorithms for Counting Triangulations and Related Problems. In *SOCG'16 to appear.*

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

46 G. L. Miller. Finding small simple cycle separators for 2-connected planar graphs. In R. A. DeMillo, editor, *SODA'84*, pages 376–382. ACM, 1984.

47 G. L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *J. Comput. Syst. Sci.*, 32(3):265–279, 1986.

48 W. Mulzer and G. Rote. Minimum-weight triangulation is NP-hard. *J. ACM*, 55(2), 2008.

49 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.

50 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.

51 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.

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

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

54 M. Wettstein. Counting and enumerating crossing-free geometric graphs. In *SOCG'14*.

55 M. Wettstein. Algorithms for Counting Crossing-free Configurations. Master's thesis, ETH, Eidgenössische Technische Hochschule Zürich, Zürich Switzerland, 2013.