A Simple Aggregative Algorithm for Counting Triangulations of Planar Point Sets and Related Problems

Victor Alvarez[*] Raimund Seidel[†]

Abstract

We give an algorithm that determines the number $\mathsf{tr}(S)$ of straight line triangulations of a set S of n points in the plane in worst case time $O(n^2 2^n)$. This is the the first algorithm that is provably faster than enumeration, since $\mathsf{tr}(S)$ is known to be $\Omega(2.43^n)$ for any set S of n points. Our algorithm requires exponential space.

The algorithm generalizes to counting all triangulations of S that are constrained to contain a given set of edges. It can also be used to compute an optimal triangulation of S (unconstrained or constrained) for a reasonably wide class of optimality criteria (that includes e.g. minimum weight triangulations). Finally, the approach can also be used for the random generation of triangulations of S according to the perfect uniform distribution.

The algorithm has been implement and is substantially faster than existing methods on a variety of inputs.

1 Introduction

For the purposes of this paper a *plane graph* on a set S of points in the plane is a set E of *straight* line segments joining points in S that do not intersect except in common endpoints. We will feel free to switch without much ado between the geometric view of such a graph and the discrete view afforded by the naturally associated abstract graph. Let $\mathcal{PG}(S)$ denote the set of all plane graphs on S. The set $\mathcal{PG}(S)$ has received considerable research attention along with some of its subsets, like the set of all *plane perfect matchings* (all 1-regular graphs in $\mathcal{PG}(S)$), the set of all *plane cycle covers* (all 2-regular graphs in $\mathcal{PG}(S)$), the set of all *plane spanning cycles* (all connected 2-regular graphs), the set of all *plane spanning trees* (all connected graphs in $\mathcal{PG}(S)$ with a minimal number of edges), and finally the set of all *plane triangulations* (all graphs in $\mathcal{PG}(S)$ with a maximal number of edges). We will refer to those different subsets of $\mathcal{PG}(S)$ as classes.

For all these classes exponential lower and upper bounds have been proven for their sizes [4, 24, 13, 3, 18, 27, 26, 14, 25, 23, 2], i.e. statements of the form "for any planar set S of n points the number of plane perfect matchings is $\Omega(c_1^n)$ and $O(c_2^n)$." However, typically c_1 and c_2 are (too) far apart. In some cases the general lower bounds are trivial: e.g. points in convex position admit only one plane spanning cycle.

There has been some interesting work [10, 9, 14, 20, 26] on relating the sizes of those classes: E.g. it is easy to see that the number of plane graphs of S is not more than 2^{3n} times the number of plane

[*]Fachrichtung Informatik, Universität des Saarlandes, alvarez@cs.uni-saarland.de

[†]Fachrichtung Informatik, Universität des Saarlandes, rseidel@cs.uni-saarland.de

triangulations of S. The fact proven by Razen and Welzl [20] that the number plane graphs of S is at least $2^{3n/2}$ times the number of plane triangulations of S, however, is far from trivial. The same can be said for the result of Sharir, Sheffer, and Welzl that the number of plane perfect matchings of S is at most $O(1.1067^n)$ times the number of plane triangulations of S. Still, our understanding of these relationships is quite rudimentary.

A different line of research has addressed the problem of computing the sizes of those various classes of plane graphs for a given point set S. Efficient enumeration algorithms are known for all these classes, where "efficient" means polynomial overhead per produced graph. But the interesting problem is to determine the size N of such a class in time that is substantially less than N. So far this has been only accomplished for the class of all plane graphs in a recent paper of Razen and Welzl [20], who manage to determine $|\mathcal{PG}(S)|$ in time roughly $O\left(|\mathcal{PG}(S)|/2^{3n/2}\right)$. However, their method does involve enumerating all plane triangulations of S.

This paper deals with the class of plane triangulations of S. We will refer to it by $\mathcal{T}(S)$ and we will use $\mathsf{tr}(S) = |\mathcal{T}(S)|$. There has been considerable work determining lower and upper bounds for $\mathsf{tr}(S)$. Currently the best bounds known are [23, 26]

$$\Omega(2.43^n) \leq \mathsf{tr}(S) \leq O(30^n)\,,$$

where $n = |S|$.

There has also been a fair number of papers on computing $\mathsf{tr}(S)$ given S: The reverse search method of Avis and Fukuda [7] can be applied to enumerate (see [8] for a particularly fast realization), Katoh and Tanigawa [15] consider more general enumeration problems, Aichholzer [1] proposed a divide-and-conquer method base on so-called triangulation paths, Ray and Seidel [19] exploited dynamic programming, and Alvarez, Bringmann, and Ray [6] applied a sweep approach based on triangulation paths and also proposed a method that exploited the onion layer structure of a point set, see [5]. This last approach achieved the so far best worst case running time with a bound of $O(3.1414^n)$. It should be noted that it is unlikely that a polynomial time counting method will be found, since a closely related problem was shown to be NP-hard [16, 22] and even $W[2]$-hard [5].

In this paper we give an algorithm that determines $\mathsf{tr}(S)$ in worst case time $O(n^2 2^n)$ and space $O(n2^n)$. This running time can well be called substiantially less than $\mathsf{tr}(S)$, since, as already mentioned, $\mathsf{tr}(S) \geq \Omega(2.43^n)$. The algorithm is suprisingly simple given how long this problem has been studied already. We have implemented the new method and report some experimental results. Our approach can also be used to count all triangulations of S that contain a prescribed set of edges, to find an "optimum" triangulation with respect to certain "decomposable" optimization criteria, or to generate triangulations of S uniformly at random.

2 The Algorithm

Let $S = \{p_1, \ldots, p_n\}$ be a set of n points in the plane. We assume that S is not contained in a straight line, otherwise there would be no triangulations to count. For the sake of ease of exposition we assume that no three points in S are colinear and no two points lie on a common vertical line. Thus we can assume without loss of generality that the points in S are indexed by increasing x-coordinate. A *monotone chain for S* is a polygonal chain that connects the leftmost point p_1 and the rightmost point p_n, contains only points of S as vertices, and intersects every vertical line at most once.

Let T be some plane triangulation of S. From now on we will omit the adjective "plane." A *monotone chain in triangulation* T is a monotone chain for S all whose segments are edges of the triangulation T.

For a monotone chain C in T let $\Delta_T(C)$ be the set of triangles of T that lie below C. We call a triangle t of T an *advance* for C if it lies above C and $\Delta_T(C) \cup \{t\} = \Delta_T(C')$ for some monotone chain C' in T. We say that in this case C advances to C'. Note that there are two different types of advances. Let the advance triangle t be spanned by points p_i, p_j, p_k, with $i < j < k$, which also means p_i is to the left of p_j which in turn is to the left of p_k. Either p_j lies above the segment $[p_i, p_k]$. In this case t intersects C in that segment and C' is obtained from C by replacing segment $[p_i, p_k]$ by the two segments $[p_i, p_j], [p_j, p_k]$. Thus such an advance increases the length of the chain and includes a new vertex. Otherwise p_j lies below $[p_i, p_k]$. In this case t intersects C in $[p_i, p_j], [p_j, p_k]$ and C' is obtained from C by replacing those two segments by $[p_i, p_k]$. This type of advance decreases the length of the chain and expels vertex p_j from the chain.

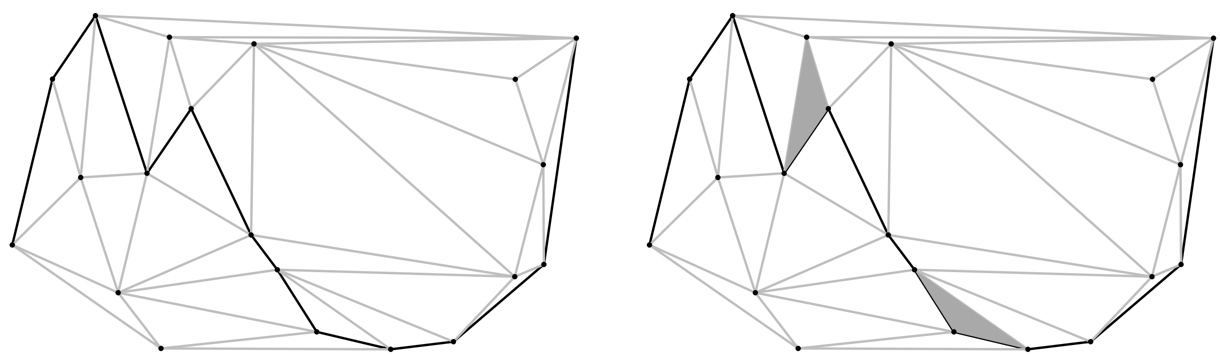

Figure 1: A monotone chain in a triangulation and two possible advances.

The following is folklore:

Lemma 1. *Let T be some triangulation of S. For any monotone chain C in T there is an advance triangle, unless C consists only of edges of the upper boundary of the convex hull of S.*

Proof. Let C be a monotone chain in T, let $\overline{C}$ be a maximal subchain of C containing no "upper hull edges," and let p_h and p_ℓ the left and right endpoints of $\overline{C}$. Note that for each edge $e = [p_i, p_j]$ of $\overline{C}$ there is a unique triangle t_e of T that contains e and lies above $\overline{C}$. Let p be the corner of t_e that is not on e. Assign the orientation `left`, `right`, or `none` to e depending on whether p is to the left of p_i, to the right of p_j or "between" p_i and p_j. If some edge e of $\overline{C}$ has orientation `none` then t_e constitutes an advance triangle. So assume orientation `none` does not occur. Note that if $e = [p_i, p_j]$ is oriented `right` and the next segment $e' = [p_j, p_k]$ is oriented `left`, then $t_e = t_{e'}$ and this triangle constitutes an advance triangle. But note that such a configuration must occur since the leftmost edge of $\overline{C}$ must be oriented `right` and the rightmost edge of $\overline{C}$ must be oriented `left`. ■

Every monotone chain C in T (except for the topmost one) must have some advance. As a matter of fact it can have several of them. But it has a unique leftmost one, t, i.e. no other advance triangle intersects C to the left of t. This means that for every triangulation T there is a unique sequence $C_0, \ldots, C_M$ of monotone chains in T, where C_0 is formed by the lower boundary of the convex hull, C_M by the upper boundary of the convex hull, and each C_ℓ is obtained from $C_{\ell-1}$ by a leftmost advance. Here M is the number of triangles in T, which must be $2n - h - 2$, where $n = |S|$ and h is the number of points in S that are on the boundary of the convex hull of S. We call this unique sequence of chains a *leftmost advancing sweep* of T.

Thus in order to count the number of triangulations of S it suffices to count the number of leftmost advancing sweeps. We will do this by forming a directed acyclic graph G_S in which source-sink paths correspond 1-1 with leftmost advancing sweeps. Counting all such source-sink paths can easily be done in time linear in the size of G_S by traversing its nodes in topological order.

The nodes of G_S will be *marked monotone chains for S*. Such a marked chain is simply a monotone chain for S with one of its edges marked. It can be denoted by the pair (C, ℓ), where C is a monotone chain and ℓ is an integer indicating that the ℓ-th edge e_ℓ of C is marked (counting from left to right).

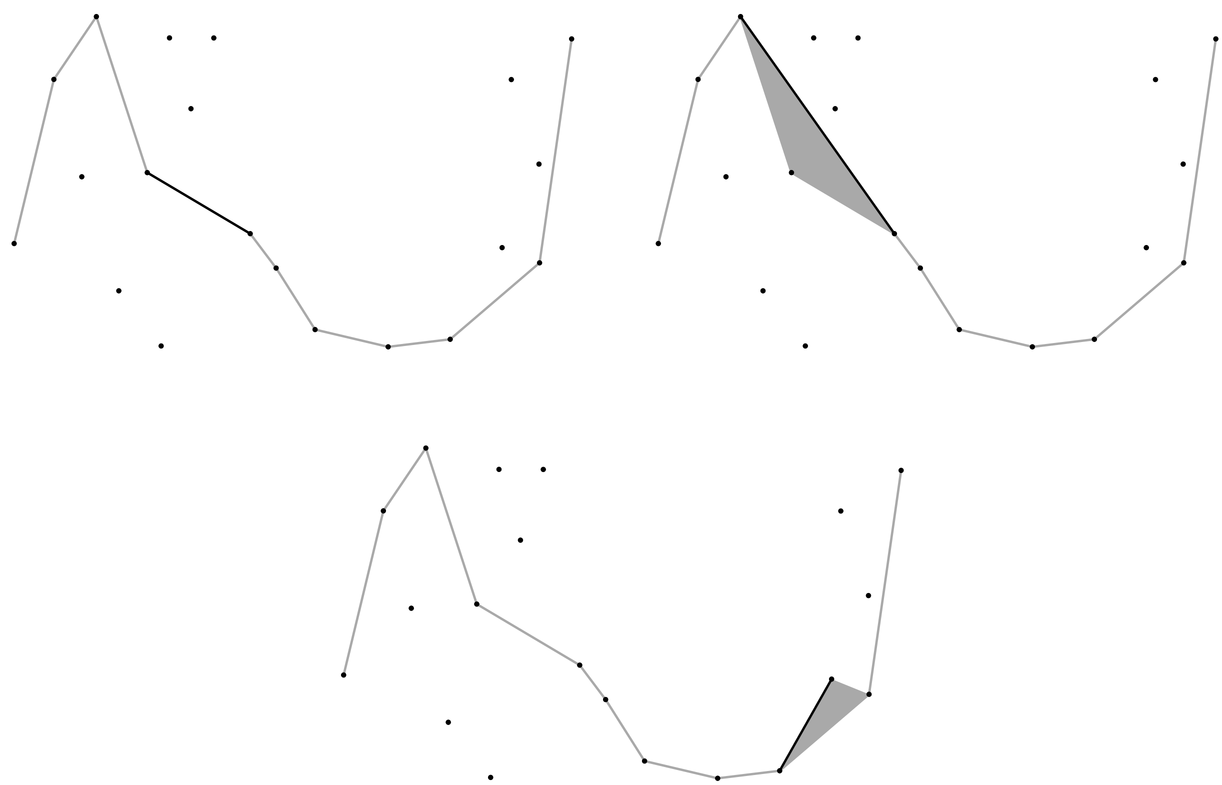

Figure 2: A marked monotone chain for S and two possible successors.

We define a successor relation on such marked monotone chains. Let (C, ℓ) be one such marked monotone chain. Let $e_m = [p_i, p_j]$ with $m \geq \ell$ be the m-th edge of C. First assume there is some p_μ in S that lies "between" p_i and p_j (i.e. $i < \mu < j$), that lies above e_m, and for which the triangle spanned by e_m and p_μ contains no point of S in its interior. Consider the monotone chain C' obtained from C by replacing edge e_m by the sequence of two edges $[p_i, p_\mu], [p_\mu, p_j]$. We define each marked monotone chain (C', m) obtained this way to be a successor of (C, ℓ).

Next assume that chain C contains two consecutive edges e_{m-1}, e_m with $m \geq \ell$ so that the triangle spanned by those two edges lies above C and contains no points of S in its interior. Let p_i, p_j, p_k be the three endpoints of the two segments in left to right order, and consider the monotone chain C' obtained from C by replacing e_{m-1}, e_m by edge $[p_i, p_k]$. We define each marked monotone chain

$(C', m-1)$ obtained this way to be a successor of (C, ℓ).

Our DAG G_S contains an edge from (C, ℓ) to (C', m) iff (C', m) is a successor of (C, ℓ).

Let B be the bottom-most monotone chain for S, i.e. B contains the sequence of edges on the lower boundary of the convex hull of S. Analogously let U be the top-most monotone chain for S, i.e. the sequence of edges of the upper boundary of the convex hull of S. Let C be some monotone chain for S. The following can easily be proven by induction:

Claim 2. *There is a 1-1 correspondence between the directed paths in G_S from $(B, 1)$ to (C, k) and the triangulations of the area A sandwiched between chains B and C whose leftmost advancing sweep has as last advance a triangle t whose leftmost upper boundary edge is the k-th edge of C.*

The triangulation of A must of course include also all points of S as vertices that lie between B and C. Note that for a given chain C all such triangulations have the same number of triangles. We refer to this number as the *level* of C. The induction for proving this claim is on this level number.

Since for any triangulation of S the leftmost advancing sweep has as last advance a triangle whose upper boundary is part of the top-most chain U we get that every path in G_S from $(B, 1)$ to (U, k) for some k corresponds to a leftmost advancing sweep of some triangulations of S. Adding a "top" vertex $\top$ to G_S that has an edge to it from every marked chain (U, k) we get our main theorem:

Theorem 3. *The number of triangulations of S is the number of paths in G_S from source $(B, 1)$ to sink $\top$.*

The number of those source sink paths can be determined in time $O(n^2 2^n)$ and using space $O(n 2^n)$ in the worst case.

Proof. After the preceding discussion we only need to prove the resource bounds.

Determining the number of source-sink paths in a DAG can be achieved by the following algorithm: store a counter with each node that is initialized to 0; initialize the counter of the source to 1; iterate through the vertices in topological order and for each vertex add its countervalue to the counter of each of its successors.

The time necessary for this algorithm is proportional to the number of edges of the DAG G_S. Since each monotone chain for S is uniquely identified by a subset of S there are at most 2^n chains (actually 2^{n-2} since p_1 and p_n are always included) and for each chain there are at most $n-1$ possible markings. Thus the number of nodes in our DAG G_S is $O(n2^n)$. Each node can have at most n successors, since each successor either includes a new vertex into the chain or excludes one. Thus the number of edges in G_S is $O(n^2 2^n)$, which proves the running time bound.

The algorithm does not need to store edges, since given a marked chain, its successors can be computed in $O(n)$ time after polynomial preprocessing. Thus only space for the $O(n2^n)$ nodes of G_S is necessary and the claimed space bound follows. ■

A short remark on the model of computation: since we use exponential space we have to work with a RAM with linear word size. Therefore it is fair to assume that our counters which can take on exponentially large values can be stored in a single word and can be added in constant time.

3 Generalizations

Our approach admits some easy generalizations. For example, it can be used to determine the number of all triangulations of a set S that are constrained to contain a certain set R of edges. It suffices to observe that such constrained triangulations also have a unique leftmost advancing sweep and to adjust the successor relation for the definition of the DAG G_S so that segments that cross edges in R are never considered.

The approach can also be used to compute an "optimal" triangulation of S, as long as the function to be optimzed is sufficiently well-behaved. It suffices that the function is defined for any triangulation of any polygonal domain D, and for any triangle t the optimum value over all triangulations of $D \cup t$ that contain t can be obtained from the optimum value for D and from t. In our algorithm D will be the domain between the bottom chain B and the chain C under consideration and t is the advance triangle sandwiched between C and the successor chain under consideration.

Examples of such well behaved functions are the sum or the maximum of the weights of the triangles (or edges) in a triangulation where the weights are arbitrary numbers assigned to each triangle spanned by three points in S. For instance, you could assign to each triangle t the weight $|\text{area}(t) - \mu)|$, where μ is the area of the convex hull of S divided by M, and M is the number of triangles in any triangulation of S, i.e. $M = 2n - h - 2$, with $n = |S|$ and h the number of points in S that are on the boundary of the convex hull of S. The triangulation that minimizes the sum or the maximum of these weights then consists of triangles whose areas deviate from the mean minimally.

Of course such optimization problems can also be solved for constrained triangulations.

Finally, our approach yields a method for generating a random triangulation of S truly uniformly, unbiased. More precisely, we can preprocess a given point set in time $O(n^2 2^n)$ to produce a data structure of size $O(n^2 2^n)$ from which we can then generate triangulations of S truly uniformly at random at cost $O(n^2)$ per triangulation.

In spite of the large preprocessing cost this can be quite useful. For instance, we have used this approach to estimate the proportion of the triangulations of S that are regular [12, page 55] by taking sufficiently many random samples and testing each sample triangulation for regularity (which just amounts to solving a linear program). This gave us strong empirical evidence that the proportion of regular triangulations of n points, of which k are extreme, goes to 0 with increasing n, when k stays fixed. Details will be described elsewhere.

Here is the random generation method: First we compute the graph G_S along with the counter values for each node. We remove all nodes that are not reachable from the source vertex $(B, 1)$ or from which you cannot reach sink vertex $\top$.

We choose a random triangulation by constructing a random path: choose a random number r between 1 and $\mathsf{tr}(S)$ and construct the r-th source-sink path in reverse order as follows: Starting with $v = \top$ you proceed until $v = (B, 1)$ doing the following: consider the predecessors of v in a canonical order and subtract their counts from r as long as r would stay positive; choose that last predecessor as new v. It is an easy exercise to see that each source-sink path is chosen this way with equal probability and hence each triangulation is chosen with equal probability.

4 Counting other crossing-free structures

The main new ingredient in our method is the reduction to counting source-sink paths in a directed acyclic graph. Wettstein and Welzl [28] have managed to discover similar such reductions for counting or enumerating various other crossing free structures on a planar point set, such as perfect matchings, spanning cycles, spanning trees, or all plane graphs.

5 Dead Ends

We were hoping that our approach could be adapted to determine the number of (pointed) pseudotriangulations of a point set S. So far this has been in vain. The main reason is that Lemma 1 may fail in this context.

Even more importantly, we were hoping that our approach could lead to an improved upper bound for the number of triangulations that every point set can have. The hope was that a combinatorial description of a leftmost advancing sweep would suffice to specify the actual geometric sweep. This combinatorial description would be something of the form: advance chain C by replacing edges e_k, e_{k+1} by a single edge, then advance replacing edge $e_m = [p_i, p_j]$ by the two edges $[p_i, p_\lambda], [p_\lambda, p_j]$ (without specifying p_λ), and so on. If this were possible, then a 27^n upper bound for the number of triangulations of S would follow, since it is relatively easy to show via a labelling argument that there are at most 3^{3n} "combinatorial leftmost advancing sweeps."

However, there is an old example of Günter Rote [21] that shows that there are point sets that admit two different triangulations that are isomorphic even if the edges are all oriented left to right. Figure 3 shows such a point set. But it is easy to see that two such isomorphic triangulations must have leftmost advancing sweeps that have identical combinatorial descriptions, and thus the combinatorial description by itself cannot specify a unique triangulation.

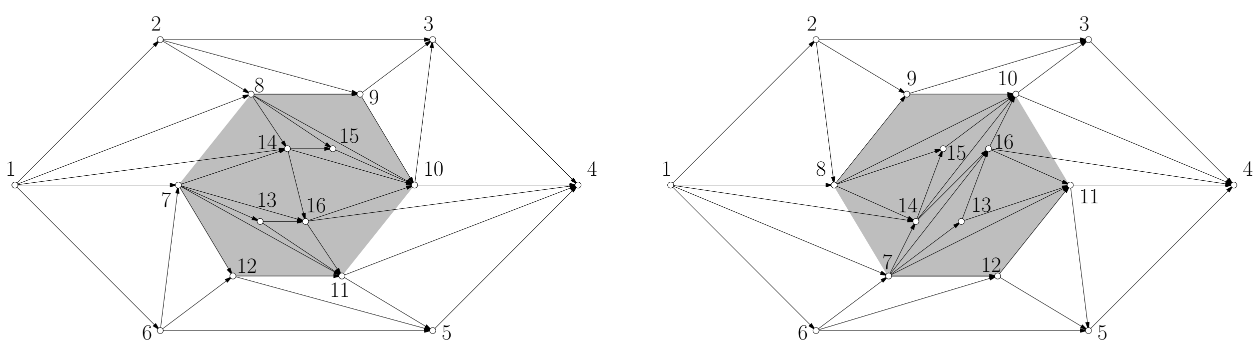

Figure 3: A point set with two different triangulations that are isomorphic even if all edges are oriented left to right.

6 Implementation Issues

Our approach is space-intensive. Thus saving space is of utmost importance.

First of all we never store the entire DAG G_S. We proceed level by level and only store the nodes of at most two consecutive levels. Recall that the level of a chain is the number of triangles in any triangulation below that chain. Say we have proceeded to level s and we have produced the set L_s

of all marked chains of that level that can be reached from source chain $(B, 1)$. We initialize an empty dictionary for L_{s+1}. We iterate over L_s and for each marked chain (C, k) we generate its successors and for each successor we insert it into L_{s+1} if it is not there already and we add (C, k)'s counter value to the counter value of the successor. After this we can delete (C, k).

Note that a marked chain may not have any successors and may form a "dead end" in G_S. It is of course desirable to avoid such dead ends as much as possible. So we need conditions that tell us that a given marked chain (C, k) cannot possibly reach $\top$ in G_S so that we can remove (C, k) from further consideration right away. A sufficient condition for dismissal of (C, k) is the following: Among the leftmost $k - 1$ edges of C there is one that is only visible by points of S that are on or above C and that are left of the k-th edge e_k. The only obstacle for visibility is the chain C. Clearly in such a situation any leftmost advancing sweep would have had to advance on a triangle left of e_k before, and hence (C, k) cannot occur in such a sweep.

The entries of the dictionaries L_s are of the form `(key, value)`, where `key` is a bit vector representing (C, k) and `value` is a possibly large integer. (C, k) can be represented by a bit vector of length $n + \log_2 n$ with n bits specifiying which points of S are vertices of C and the other bits giving the binary representation of k. For all we know `value` could be an integer as large as 30^n, at least this is the best upper bound currently known. Instead of reserving so much space for counters we can resort to **Chinese Remaindering** and count modulo some prime P, for which we only need a fixed number of bits. By running the algorithm several times for different primes and applying the Chinese Remainder Theorem to the resulting count remainders we can recover the true final count.

The dictionaries L_s must allow fast search, insertion, and enumeration. Hashing is a natural choice for the realization of such dictionaries. We have found simple hashing with chaining to be too wasteful in space because of the need for pointers. We finally used cuckoo hashing which is reasonably fast and gets by without pointers.

It is possible to avoid dictionaries alltogether. When processing L_k simply forego searching and generate copies of `(key,value)` pairs. In the end we sort on `key` and aggregate the values of duplicates to one value. This has proven to be too wasteful in terms of space when run on a single machine. However, this way of proceeding is a typical "map-reduce" step [11]. Thus if we are interested in solving large instances using clusters of computers this may well be the simplest way to proceed. Currently the map-reduce paradigm is mostly used for a single step. Our algorithm would need iterated map-reduce steps.

7 Experiments

We have implemented our sweep approach and have compared it to two other algorithms, namely the dynamic programming algorithm of [19] and the sn-paths algorithm of [5]. The latter algorithm is tuned for point sets that have few onion layers, i.e. they can be decomposed into the vertex sets of few nested convex polygons. All algorithms were run on a machine with 128 Gigabytes of main memory.

Tables 1 through 3 give some comparisons of running times and space usage. The first table considers n points chosen uniformly at random in a square, the second table considers point sets that essentially are nested triangles, the last table considers points chosen randomly from $k = 3$ concentric circles. In these tables h denotes the number of extreme points of the respective set and

k denotes the number of its onion layers. The term "Base" denotes the n-th root of the number to the left; "# Sub-problems" in "sweep-algo" denotes the maximum number of marked chains encountered at any level.

The sweep algorithm is particularly superior for random input instances, where it could solve problems with up to 50 points, which was completely out of reach before. As expected, the sn-path algorithm performs extremely well for point sets with few onion layers.

The sweep algorithm performs very poorly if all points are in convex position. Figure 4 gives a brief account of this phenomenon.

We also made preliminary experiments with the sweep algorithm for the constrained version of the problem. Figure 5 illustrates those results with two examples. One has a perfect matching as constraint set, the other one the edeges of the minimum spanning tree. Note that the latter example could of course also be solved using dynamic programming, which would take only $O(n^3)$ time.

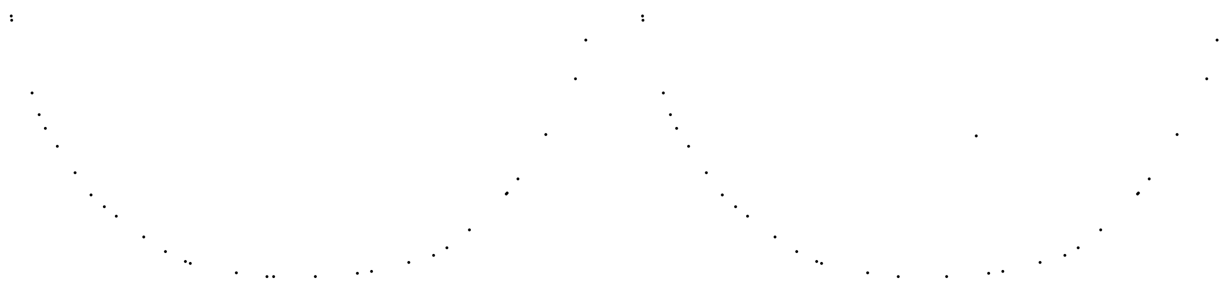

Figure 4: To the left 30 points in convex position which forces the worst-case behavior of the algorithm. It took our program almost 17 minutes and 7050 MB of RAM memory to complete this instance. To the right the same configuration but with one point moved to the interior of the convex hull. This reduces the complexity of the algorithm significantly: almost 9 minutes and 3633 MB of RAM memory.

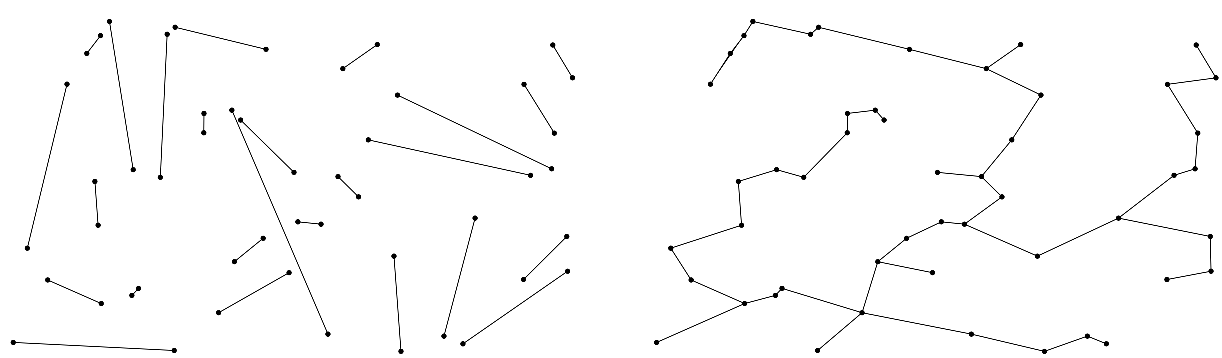

Figure 5: Both shown sets of points are the same. They correspond to the first set of 50 points of Table 1, having 113071115010855074515830603921337 $\approx 4.37^{50}$ triangulations. The number of triangulations containing the shown matching is 1214781483428889112873468 $\approx 3.03^{50}$, which constitutes approximately a $1.074 \cdot 10^{-8}$ fraction of the total number of triangulations. The number of triangulations containing the minimum weight spanning tree is 3963846555354291101173440 $\approx 3.10^{50}$, which is about a $3.505 \cdot 10^{-8}$ fraction of the total number of triangulations.

n	k	h	#Triangulations	Base	Time in hhh:mm:ss.ms			RAM in MB			# Sub-problems		
					dyn-prog	sn-paths	sweep-algo	dyn-prog	sn-paths	sweep-algo	dyn-prog	sn-paths	sweep-algo
30	5	9	29762284427845618	≈ 3.54	7.60	3:56.61	5.02	130	141	8	854579	947262	88490
	6	7	54648952555202115	≈ 3.61	30.69	16:17.12	11.28	470	535	16	3150228	3590878	233038
33	5	11	8830953374442248378	≈ 3.75	34.80	14:47.26	28.84	643	394	79	4245399	2554063	549137
	6	7	23407918365649149382	≈ 3.86	15.10	1:10:34	53.16	288	1292	65	1907449	8731943	723316
37	5	11	8317197892568798832050	≈ 3.91	3:35.27	1:15:00	8:44.55	2796	1524	615	18477670	9735430	3944618
	5	13	15347609782987966767248	≈ 3.97	15:23.53	2:16:32	9:21.30	10707	1957	437	70483691	12535632	7147625
40	6	12	1146138971033715203926926	≈ 3.99	25:42.43	13:29:43	1:00:49.56	18525	8889	2139	121049523	56587195	35007849
	7	10	5050493282169462429012536	≈ 4.14	1:35:45	46:49:41	23:38.37	54128	25533	776	354717051	155716531	12694463
43	6	10	981403313298259834292202925	≈ 4.24	3:20:54	107:48:48	1:39:51.84	116506	37407	4578	752596823	239084256	49968377
	7	8	707769153122173171028848193	≈ 4.21			3:43:21			10614			115902214
47	7	7	362038516572525577788093397554	≈ 4.46			8:22:12			11427			149735651
	6	10	1789292526793886024349584939752	≈ 4.40			15:27:04			22351			292928200
50	6	11	113071115010855074515830603921337	≈ 4.37			18:53:56			30637			401522629
	6	10	147019897942999105259582587551602	≈ 4.39			26:41:00			47378			620962165

Table 1: n random points in a square.

k	n	#Triangulations	Base	Time in hh:mm:ss.ms			RAM in MB			# Sub-problems		
				dyn-prog	sn-paths	sweep-algo	dyn-prog	sn-paths	sweep-algo	dyn-prog	sn-paths	sweep-algo
10	28	134806114688321888	≈ 4.09	1:04.33	39:29:17	33.36	1130	56197	43	8015023	347448787	440822
	28	259051751512786147	≈ 4.18	58.55	32:31:58	13.44	903	41745	21	6303203	266661064	209627
11	33	1360909298406546057232	≈ 4.36	33:23.12		13:07.23	26618		474	181487896		6186341
	33	1862373658387735722566	≈ 4.41	1:22:42		45:24.99	52344		1866	353976704		15271920
12	35	68231356546945667957547	≈ 4.49	1:56:47		1:20:25.55	78356		2758	528028609		30094238
	34	4657839362065190027715	≈ 4.33	1:45:20		2:35:44.46	73150		4541	495166380		59490749
13	37	1075218822593378348459037	≈ 4.46			10:59:33			22268			243201755
	37	1374291968080852706936837	≈ 4.49			16:11:21			31168			408493620

Table 2: n random points having $k = \lceil \frac{n}{3} \rceil$ onion layers.

n	h	#Triangulations	Base	Time in hh:mm:ss.ms			RAM in MB			# Sub-problems		
				dyn-prog	sn-paths	sweep-algo	dyn-prog	sn-paths	sweep-algo	dyn-prog	sn-paths	sweep-algo
30	10	161014656152655441	≈ 3.74	20.18	55.77	7.18	303	33	25	2050514	215732	139697
	10	312513373686594183	≈ 3.82	1:07.72	1:09.17	16.46	1160	38	34	7879754	246657	344201
33	11	32155601714553665796	≈ 3.90	3:50.96	2:19.19	3:02.25	2762	62	293	18992928	405580	2178760
	11	68598010833407738067	≈ 3.99	58.43	2:30.94	2:21.51	857	62	265	5812991	410357	1725109
37	12	9334947679230323509429	≈ 3.92	1:53.60	3:43.51	9:47.13	1521	90	395	10027300	575255	5144450
	12	31113068813012076443512	≈ 4.05	3:20.41	4:42.24	11:19.98	2465	97	664	16250100	626274	7229246
40	14	2642143054680217856074126	≈ 4.07	18:12.16	8:10.26	31:33.81	12557	131	1601	82635240	866278	17458086
	15	2903778262295075928823011	≈ 4.08	7:01.40	9:50.98	1:02:46.86	4325	149	3586	28333612	982791	39139805
43	14	452371697808162396583055656	≈ 4.16	1:34:48	21:39.66	1:50:7.77	53263	243	5008	347603518	1604269	54673756
	14	461550214764369881018564051	≈ 4.16		19:25.53	2:56:09.84		242	8612		1591423	94039067
47	16	157759710540671985436621922639	≈ 4.18		32:46.68	10:02:04		363	17507		2287764	229438083
	15	341037585238678346710372748758	≈ 4.24		39:33.42	5:32:04		420	8971		2720786	117544183
50	16	54782168649020627430413001433261	≈ 4.31		1:06:53	57:16:32		606	71609		3631525	938562001
	16	158997592723683977758501079915910	≈ 4.40		1:07:19	30:07:44		553	46872		3998798	614328639

Table 3: n random points on three concentric circles, each having $\approx \frac{n}{3}$ points.

8 Acknowledgements

Special thanks to Günter Rote for letting us use his original eps-file for Figure 3, and many thanks to Bernd Finkbeiner for letting us use extensively the large main memory machine of his group.

References

[1] O. Aichholzer, "The path of a triangulation", in *Symposium on Computational Geometry*, pp. 14–23, 1999. One citation on page 2.

[2] O. Aichholzer, F. Aurenhammer, H. Krasser, and B. Speckmann, "Convexity minimizes pseudo-triangulations", *Comput. Geom.*, vol. 28, no. 1, pp. 3–10, 2004. One citation on page 1.

[3] O. Aichholzer, T. Hackl, B. Vogtenhuber, C. Huemer, F. Hurtado, and H. Krasser, "On the number of plane geometric graphs", Graphs and Combinatorics, vol. 23(1), pp. 67–84, 2007. One citation on page 1.

[4] M. Ajtai, V. Chvátal, M. Newborn, and E. Szemerédi, "Crossing-free subgraphs", in *Theory and Practice of Combinatorics A collection of articles honoring Anton Kotzig on the occasion of his sixtieth birthday* (G. S. Peter L. Hammer, Alexander Rosa and J. Turgeon, eds.), vol. 60 of *North-Holland Mathematics Studies*, pp. 9 – 12, North-Holland, 1982. One citation on page 1.

[5] V. Alvarez, K. Bringmann, R. Curticapean, and S. Ray, "Counting crossing-free structures", in *Symposium on Computational Geometry* (T. K. Dey and S. Whitesides, eds.), pp. 61–68, ACM, 2012. 3 citations on pages 2 and 8.

[6] V. Alvarez, K. Bringmann, and S. Ray, "A simple sweep line algorithm for counting triangulations and pseudo-triangulations", *Submitted*, 2012. One citation on page 2.

[7] D. Avis and K. Fukuda, "Reverse search for enumeration", *Discrete Applied Mathematics*, vol. 65, no. 1-3, pp. 21–46, 1996. One citation on page 2.

[8] S. Bespamyatnikh, "An efficient algorithm for enumeration of triangulations", *Comput. Geom.*, vol. 23, no. 3, pp. 271–279, 2002. One citation on page 2.

[9] K. Buchin and A. Schulz, "On the number of spanning trees a planar graph can have," in *ESA* (M. de Berg and U. Meyer, eds.), vol. 6346 of *Lecture Notes in Computer Science*, pp. 110–121, Springer, 2010. One citation on page 1.

[10] K. Buchin, C. Knauer, K. Kriegel, A. Schulz, and R. Seidel, "On the number of cycles in planar graphs," in *COCOON* (G. Lin, ed.), vol. 4598 of *Lecture Notes in Computer Science*, pp. 97–107, Springer, 2007. One citation on page 1.

[11] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8*, pp. 137–150, USENIX Association, 2004. One citation on page 8.

[12] J. A. De Loera, J. Rambau, F. Santos, *Triangulations: Structures for Algorithms and Applications*, Algorithms and Computation in Mathematics, Vol. 25, Springer-Verlag One citation on page 6.

[13] P. Flajolet and M. Noy, "Analytic combinatorics of non-crossing configurations," *Discrete Mathematics*, vol. 204, no. 1-3, pp. 203–229, 1999. One citation on page 1.

[14] M. Hoffmann, M. Sharir, A. Sheffer, C. D. Tóth, and E. Welzl, "Counting plane graphs: Flippability and its applications", in *WADS* (F. Dehne, J. Iacono, and J.-R. Sack, eds.), vol. 6844 of *Lecture Notes in Computer Science*, pp. 524–535, Springer, 2011. One citation on page 1.

[15] N. Katoh and S.-I. Tanigawa, "Fast Enumeration Algorithms for Non-crossing Geometric Graphs", Discrete & Computational Geometry, vol. 42, no. 3, pp. 443–468, 2009. One citation on page 2.

[16] E. L. Lloyd, "On triangulations of a set of points in the plane," in *FOCS*, pp. 228–240, IEEE Computer Society, 1977. One citation on page 2.

[17] P. McCabe and R. Seidel, "New Lower Bounds for the Number of Straight-Edge Triangulations of a Planar Point Set", in *European Workshop on Computational Geometry*, 2004. No citations.

[18] A. G. Olaverri, M. Noy, and J. Tejel, "Lower bounds on the number of crossing-free subgraphs of K_n", *Comput. Geom.*, vol. 16, no. 4, pp. 211–221, 2000. One citation on page 1.

[19] S. Ray and R. Seidel, "A simple and less slow method for counting triangulations and for related problems", in *European Workshop on Computational Geometry*, 2004. 2 citations on pages 2 and 8.

[20] A. Razen and E. Welzl, "Counting plane graphs with exponential speed-up", in *Rainbow of Computer Science* (C. S. Calude, G. Rozenberg, and A. Salomaa, eds.), vol. 6570 of *Lecture Notes in Computer Science*, pp. 36–46, Springer, 2011. 3 citations on pages 1 and 2.

[21] G. Rote, Private Communication. One citation on page 7.

[22] A. Schulz, "The existence of a pseudo-triangulation in a given geometric graph," in *EuroCG*, 2006. One citation on page 2.

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

[24] M. Sharir and A. Sheffer, "Counting plane graphs: Cross-graph charging schemes," *CoRR*, vol. abs/1209.0194, 2012. One citation on page 1.

[25] M. Sharir, A. Sheffer, and E. Welzl, "On degrees in random triangulations of point sets", *J. Comb. Theory, Ser. A*, vol. 118, no. 7, pp. 1979–1999, 2011. One citation on page 1.

[26] M. Sharir, A. Sheffer, and E. Welzl, "Counting plane graphs: perfect matchings, spanning cycles, and kasteleyn's technique", in *Symposium on Computational Geometry* (T. K. Dey and S. Whitesides, eds.), pp. 189–198, ACM, 2012. 3 citations on pages 1 and 2.

[27] M. Sharir and E. Welzl, "On the number of crossing-free matchings, cycles, and partitions", *SIAM J. Comput.*, vol. 36, no. 3, pp. 695–720, 2006. One citation on page 1.

[28] M. Wettstein, "Algorithms for counting crossing-free configurations", M.Sc. Thesis, Dept. of Computer Science, ETH Zürich, 2013. One citation on page 7.