A Simple and Less Slow Method for Counting Triangulations and for Related Problems

Extended Abstract

Saurabh Ray [a,1] and Raimund Seidel [b]

[a] *Max-Planck-Institut für Informatik Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany*

[b] *Universität des Saarlandes, Im Stadtwald, 66123 Saarbrücken, Germany*

Abstract

We present a simple dynamic programming based method for counting straight-edge triangulations of planar point sets. This method can be adapted to solve related problems such as finding the best triangulation of a point set according to certain optimality criteria, or generating a triangulation of a point set uniformly at random.

We have implemented our counting method. It appears to be substantially less slow than previous methods: instances with 20 points, which used to take minutes, can now be handled in less than a second, and instances with 30 points, which used to be solvable only by employing several workstations in parallel over a substantial amount of time, can now be solved in about one minute on a single standard workstation.

1. Introduction

In recent years there has been some interest in studying the set $\mathcal{T}(S)$ of straight-edge triangulations associated with a finite planar point set S. Typical problems are *counting,* i.e. given S determine $|\mathcal{T}(S)|$, *random generation,* i.e. given S randomly generate a triangulation in $\mathcal{T}(S)$ with uniform probability, or *optimization*, i.e. given S find the triangulation in $\mathcal{T}(S)$ that satisfies some optimality criterion.

The counting problem is at this point not known to be in P. It is known that $|\mathcal{T}(S)|$ is exponential in $n = |S|$ with the best currently known lower [4] and upper [14] bounds of roughly 2.33^n and 59^n. Some previous work on the counting problem addressed cases where S has some special structure [13,10,11,6]. An algorithm for the general case was given by Avis and Fukuda [5] which was based on their reverse-search paradigm and hence counts via enumeration. For another enumerative method see Rambau's TOPCOM page [12]. The so far best algorithm was given by Aichholzer [2]. It is a divide-and-conquer type algorithm based on the notion of so-called paths of a triangulation. Erkinger [8] implemented a prototype of this algorithm and subsequently Gimpl [9] wrote a "production type" implementation that even includes parallelization in the sense of distributing work to several machines. For usage see *"The Triangulation Homepage"* [3].

The random generation and the optimization problem can clearly be solved, though expensively, using the enumeration based method of Avis and Fukuda [5]. Aichholzer's method can as well be adapted to solve the random generation problem and also certain versions of the optimization problem, namely those that are "decomposable" in the following sense: Suppose one wishes to find the optimum triangulation of a polygon P with interior points and subject to the constraint that some set E of edges must be included in the triangulation. Suppose that E is such that it induces a partition of P into two polygons P_1 and P_2. Then the optimum triangulation of P subject ot E must be computable from the optimum triangulations of P_i subject to $E \cap P_i$, with $i = 1, 2$. Examples of optimization problems that satisfy this decomposability condition are MinMax-Area [7][p. 142] or minimum total edge-length (a.k.a. minimum weight).

Email addresses: saurabh@mpi-sb.mpg.de (Saurabh Ray), rseidel@cs.uni-sb.de (Raimund Seidel).

[1] The first author is supported by the International-Max-Planck-Research-School in Saarbrücken.

We present a simple dynamic-programming based algorithm for the counting problem. It can be viewed as a divide-and-conquer algorithm that in addition stores computed results of subproblems for later reuse. The algorithm can be naturally adapted to solve the random generation problem and to solve optimization problems that are decomposable in the sense described above. We report on a prototype implementation of the counting algorithm.

2. The Algorithm

We consider the slightly more general problem of computing the number $T(P)$ of triangulations of a simple polygon P that contains the point set S and whose corners are all in S. In the original problem polygon P is just the convex hull of S.

In courses on algorithm design the case that P has no "interior points" (i.e. the corners of P are precisely S) is a common homework problem for practicing the application of dynamic programming: Choose an arbitrary edge $e = [a, b]$ of P as "base edge." Let $C(e)$ be the set of all "candidate" vertices c of P so that the triangle spanned by a, b, c is contained in P. Consider some $c \in C(e)$. The diagonals $[a, c]$ and $[b, c]$ partition polygon P into three parts: the trianlge (a, b, c) and two subpolygons P_{ac} and P_{cb} (which may be trivial, i.e. an edge). The number of triangulations of P that include triangle (a, b, c) is then given by the product $T(P_{ac})T(P_{cb})$. Since in any triangulation of P the base edge $[a, b]$ has to be part of some triangle we get

$$T(P) = \sum_{c \in C(e)} T(P_{ac})T(P_{cb}) \,.$$

Of course the subproblems like $T(P_{ac})$ are solved recursively using the same method and using $[a, c]$ as base edge. This choice of base edge ensures that all subproblems ever considered only involve subpolygons of P formed by cutting P along just one diagonal. Since there are only $O(n^2)$ such subpolygons, only $O(n^2)$ many subproblems need to be solved in total and computed results can be stored and reused. This leads to an $O(n^3)$ "time" and $O(n^2)$ "space" algorithm, where the quotation marks are to remind the reader that the stated bounds only hold in a model where the cost of storing of and operating on arbitrarily large integers is constant. In our case we will be dealing with integers not larger than 59^n, i.e. representable by $O(n)$ many bits. This leads to an $O(n^4 \log n)$ time bound and an $O(n^3)$ space bound in the more appropriate word model of computation.

Let us now consider the general case where there are "interior point," i.e. S consists of more than just the corners of P. We will proceed as in the method outlined above: Choose some base edge $e = [a, b]$ from the boundary of P. The candidate set $C(e)$ now must contain all points $c \in S$ that together with e can span a triangle in some triangulation of P. Thus $C(e)$ consists of all $c \in S$ so that the triangle D_c spanned by a, b, c is contained in P and no point of S lies in the interior of triangle D_c. (In this abstract we assume nondegeneracy.) If a candidate point c is a corner of P we can proceed exactly as in the simple case outlined above and compute the number of triangulations of P containing the triangle D_c by solving two recursive subproblems. If c lies in the interior of P the number of triangulations of P containing D_c is given by the number of triangulations of the polygon $P_c = P \setminus D_c$, i.e. the polygon P with edge $[a, b]$ replaced by the chain of two edges $[a, c], [c, b]$. Thus we only need to solve one recursive subproblem, namely finding the number of triangulations of P_c. Note that this problem is easier than the original problem in the sense that there are one fewer interior points (and if there are no interior points we know how to solve the problem). Thus our algorithm proceeds as follows:

(i) Choose some base edge $e = [a, b]$.
(ii) Determine the candidate set $C(e)$.
(iii) For each $c \in C(e)$ that is not a corner of P compute $T(P_c)$ recursively.
(iv) For each $c \in C(e)$ that is a corner of P solve two recursive subproblems and compute $T(P_{ac})T(P_{cb})$.
(v) Return the sum of the computed numbers.

In this procedure some subpolygon Q of P may be considered many times. In the usual top-down dynamic programming fashion we will compute $T(Q)$ only once and store the result for later reuse. Unfortunately the arising subpolygons do not have in general such a nice form as they do in the simple case without interior points. Thus we cannot use an array for storage but need to resort to a hash table with a canonic sequence of vertex names around polygon Q serving as the key for Q.

n	h	m	avg. #triangulations	avg. time(seconds) Dynamic Programming	avg. time(seconds) Path Method
13	3	10	117017.2	0.009	0.250
18	3	15	434650561.2	0.064	43.512
23	3	20	2175541362109.0	0.982	*
28	3	25	17295702671778911.6	24.973	*
33	3	30	90644388799559900031.2	537.890	*
18	15	3	61156327.0	0.030	1.570
23	15	8	148363536731.6	0.178	262.232
28	15	13	596631344845165.6	2.569	*
33	15	18	2615696070967273559.2	55.623	*
23	20	3	49106130174.0	0.077	69.634
28	20	8	124263097179506.8	0.480	*
33	20	13	1303793620633224385.4	10.793	*

Table 1: Results for $n = h + m$ points, with m points randomly chosen in a convex h-gon; average taken over 5 runs. An asterisk indicates that the program did not complete a single instance of that size within 90 minutes.

3. Heuristics

The algorithm outlined above makes no restricions on the choice of the base edge. We have found the following two heuristics profitable for this choice.

If P has an edge e with $|C(e)| \leq 2$, then choose e as base edge.

Otherwise fix a line ℓ that has approximately one half of the points of S on each side and as long as interior points are considered as candidate points choose as base edge always a boundary edge of the current polygon that intersects ℓ. For dividing lines of subproblems always choose lines parallel to the initial ℓ.

The reasonability of the first heuristic is obvious. The second heuristic is reminiscent of the path method of Aichholzer [2]. It aims to steer the algorithm towards a rapid breakup of the polygons considered.

4. Preliminary Experimental Results

We have implemented our dynamic programming algorithm in C++ using the g++ Compiler and STL libraries. Oswin Aichholzer [1] kindly provided us with Gimpl's C-implentation [9] of his path-based method. Thus direct runtime comparisons were possible.

Our experiments were run on a single machine with a 2.40 GHz Intel Pentium 4 Processor, with 512 MB memory and 512 KB Cache, running under Linux 2.4.21.4.p4. We did not attempt to use the parallel feature of Gimpl's program that allows to spread work over several machines.

We compared the two implemtations in a set of experiments where we considered m points distributed uniformly at random in a convex h-gon. We report here the results for pairs (h, m) with $h \in \{3, 15, 20\}$ and $h+m \in \{13, 18, 23, 28, 33\}$. For each pair of parameters Table 1 reports the average over 5 runs of the computed number of triangulations and the average time (in seconds) taken for the computation by each program. An asterisk indicates that the program did not solve a single instance of that size within 90 minutes.

We also tried our method on the largest examples that Gimpl reported on, namely a set of 32 points representing European capitals and a set of 30 random points [3]. Gimpl ran these examples on clusters of machines with a 1GHz Athlon Thunderbird Processor and with 256 MB memory running under Linux. He did not report any explicit running times for these example but stated that these was the largest he could solve in "reasonable time" employing parallelism and several machines. We have been told that "reasonable time" in this context is to mean "several weeks."

Our implementation solved the 32 point problem in 70.6 seconds and the 30 point problem in 42.9 seconds.

The largest example we tried was a set of 35 points, with 32 points distributed randomly in a triangle. On our standard machine this example led to excessive paging due to the large size of the required hash table. However on a SPARC compute server we could complete this example in 15 minutes 32 seconds using one processor and 4 GB of main memory. Adjusting hash table size to available main memory may lead to a more graceful degradation of performance of our algorithm when the input size increases.

5. Discussion

We will not discuss in this abstract how our method can be applied to the random generation problem and to the optimization problem. The adaptions that need to be made are fairly standard.

Our method does not parallelize as easily as Aichholzer's method since reusing already computed results may require non-trivial communication between processors.

We are in the process of analyzing the running time of our method. Empirically we noticed that the number of recursive calls when computing $T(S)$ is always about $\sqrt{T(S)}$. So far we have been unable to prove this or any other non-trivial statement about the running time.

Of course determining the true computational complexity of plane triangulation counting still remains an open problem.

6. Acknowledgments

We would like to thank Oswin Aichholzer for providing valuable information about the existing implementations and also for kindly providing the implementations themselves.

References

[1] O. Aicholzer, Private Communication.

[2] O. Aicholzer, The path of a triangulation, in *Proc. 15th Symp. on Comp. Geometry 1999*, pp. 14–23.

[3] `http://www.cis.tugraz.at/igi/oaich/triangulations/counting/counting.html`

[4] O. Aichholzer, F. Hurtado and M. Noy, On the Number of Triangulations Every Planar Point Set Must Have, in *Proc. 13th Annual Canadian Conference on Computational Geometry CCCG 2001*, Waterloo, Canada, 2001, pp. 13–16. An improved version is to appear in CGTA.

[5] D. Avis and K. Fukuda, Reverse Search for Enumeration. *Discrete Appl. Math. 65*, 1996, pp. 21–46.

[6] R. Bacher, Counting triangulations of configurations. *Manunscript*, `http://arxiv.org/abs/math.CO/0310206`.

[7] H. Edelsbrunner, **Geometry and Topology for Mesh Generation.** Cambridge University Press, 2001.

[8] B. Erkinger, Struktureigenschaften von Triangulierungen. *Master's Thesis*, TU-Graz, 1998.

[9] J. Gimpl, Enumeration von Triangulierungen. *Master's Thesis*, TU-Graz, 2002.

[10] F. Hurtado and M. Noy, Counting triangulations of almost convex polygons. *Ars Combinatorica 45*, 1997, pp. 169–179.

[11] V. Kaibel and G.M. Ziegler, Counting Lattice Triangulations. To appear in: *"British Combinatorial Surveys"* (C. D. Wensley, ed.), Cambridge University Press.

[12] `http://www.zib.de/rambau/TOPCOM/`

[13] D. Randall, G. Rote, F. Santos, and J. Snoeyink, Counting triangulations and pseudo-triangulations of wheels, in *Proc. 13th Annual Canadian Conference on Computational Geometry CCCG 2001*, Waterloo, Canada, 2001, pp. 149–152.

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