New Algorithms for Minimum Area k-gons

David Eppstein

Department of Information and Computer Science
University of California, Irvine, CA 92717

Tech. Report 91-59

July 15, 1991

Abstract

Given a set P of n points in the plane, we wish to find a set $Q \subset P$ of k points for which the convex hull $conv(Q)$ has the minimum area. We solve this, and the related problem of finding a minimum area convex k-gon, in time $O(n^2 \log n)$ and space $O(n \log n)$ for fixed k, almost matching known bounds for the minimum area triangle problem. Our algorithm is based on finding a certain number of *nearest vertical neighbors* to each line segment determined by two input points. We use a classical result of Ramsey theory to prove that these nearest neighbors suffice to determine the minimum convex k-gon.

1 Introduction

One of the initial results of Ramsey theory [12] was the discovery that for every k there is some $n = f(k) = 2^{O(k)}$, so that if n points are given in general position, a subset of k points can be found forming the vertices of a convex k-gon. This does not work for empty convex polygons: arbitrarily large point sets are known that do not contain an empty 7-gon [13, 14].

This naturally raises the question whether such a k-gon can be computed efficiently; several papers study this problem [3, 6, 14]. Because of Ramsey theory, finding a convex k-gon takes time depending on k but not on n; therefore it can be solved in constant time for fixed k. The best known algorithm for empty convex k-gons takes time $O(T(n))$ where $T(n)$ is the number of empty triangles in the set, which varies between $O(n^2)$ and $O(n^3)$.

Here we study the related *geometric optimization* problems of finding a k-gon minimizing or maximizing a certain objective function. A celebrated result in this area is that a minimum area triangle can be found in time $O(n^2)$ by using geometric duality to transform the problem into one of searching a line arrangement [7, 8]. Algorithms are also known for optimizing other functions including minimum perimeter [1, 5, 9] and maximum perimeter and area [2, 4].

For some time it remained open whether the minimum area triangle result could be generalized to finding minimum area k-gons. There are actually four reasonable ways of generalizing this: one could search for (1) a minimum area k-gon, (2) a minimum area *convex* k-gon, (3) a minimum area *empty* convex k-gon, or (4) a minimum area polygon that is the convex hull of k points. All of these problems can be solved trivially in $O(kn^k)$ time, but this is not very satisfactory. Problem 1 remains open (except for $k = 4$, for which it can be solved in $O(n^2)$ time using the same methods as in the minimum triangle problem), but in a recent breakthrough by Eppstein et al. [9], $O(kn^3)$ time algorithms were developed for problems 2, 3, and 4. However these algorithms are a factor of $O(n)$ away from the time for the triangle problem.

In this paper we solve the minimum area convex k-gon problem in time $O(n^2 \log n + 2^{6k} n^2)$, which for fixed k is an improvement by almost a factor of n over the previous algorithm and is only a factor of $O(\log n)$ away from the minimum triangle algorithm. We use Ramsey theory to prove that the minimum k-gon can be found in a small set of points, the *nearest vertical neighbors* of a segment determined by two of the input points. The k-gon can then be found by applying the algorithm of Eppstein et al. to these small sets. This is similar to the approach of Aggarwal et al. [1], who find small sets for minimum perimeter problems using high-order Voronoi diagrams.

We also solve the related problem of finding a set of k points with minimum area convex hull, in time $O(n^2 \log n + k^3 n^2)$. We again use nearest vertical neighbors, but fewer of them, and the proof no longer needs Ramsey theory. It is a curious fact that, for the algorithms presented here, the minimum k-point set problem is easier than the minimum k-gon, whereas in the algorithms of Eppstein et al. the difficulty of the problems was reversed (the minimum k-point set used an extra $O(k)$ factor in space).

Unfortunately our methods do not suffice to solve the third problem treated by Eppstein et al., finding a minimum area *empty* convex k-gon, except for the special cases $k = 4$ and $k = 5$. This happens because of the lack of an appropriate Ramsey theorem, due to the counterexamples described by Horton [13].

2 Nearest vertical neighbors

We begin with the problem of *nearest vertical neighbors* for points and line segments; we use this as a subroutine in our minimum k-gon algorithm.

Given a point x and a non-vertical line l, the *vertical distance* $d(x, l)$ is simply the length of a vertical line segment connecting x and l. The *nearest vertical neighbor* to l from a point set P is the point $x \in P$ minimizing the vertical distance to l.

The connection between this concept and minimum area polygons is as follows. If a triangle is formed by connecting point x to the endpoints of a line segment s, where s is contained in line l, the area of the triangle is $c \cdot d(x, l)$, where c is half the length of the horizontal projection of s. Therefore the point in P forming the minimum area triangle with s is the nearest vertical neighbor of l. This observation was used to develop $O(n^2)$ algorithms for the minimum triangle problem [7, 8].

We can tighten this characterization as follows. Let xyz be the minimum area triangle, and assume that the horizontal projection of y is between those of x and z. Then as before y is the nearest neighbor of line xz, but the vertical segment connecting y and line xz actually touches segment xz. In other words, y is within the *slab* defined by vertical lines through x and z. In general we say x is a neighbor of segment s if it appears vertically above or below s, as in this case y appears above or below segment xz. Then the triangle problem can be solved by finding, for each segment xz, the nearest neighbors above and below xz. Computing nearest neighbors to segments is harder than the original problem of nearest neighbors to lines

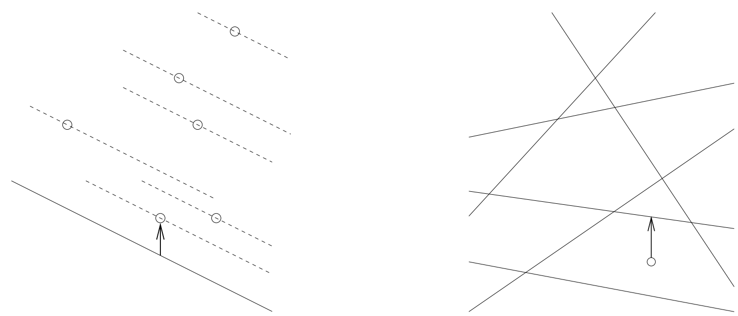

Figure 1. Finding nearest neighbors between lines and points: (a) primal, nearest point neighbor to a line; (b) dual, vertical ray shooting in a line arrangement.

but it generalizes to minimum area k-gons in a way that does not work for nearest neighbors to lines.

Given a set P of n points, and a segment s (which may not be determined by two of the points) we say that the k *vertical nearest neighbors* above (below) s are those k points above (below) the line through s, and within the slab defined by vertical lines through the endpoints of s, such that no other point with the same restrictions is closer to s. We compute these k nearest neighbors as follows.

First, suppose we only care about the restriction that the points be above or below the line l through s; this is the earlier problem of nearest neighbors to a line. By geometric duality, we can transform the line to a point and the point set to a line arrangement; Figures 1(a) and 1(b) illustrate this transformation. Point-line vertical distances are the same in the dual as they were in the primal. Therefore the dual nearest neighbor is the first line encountered by a vertical ray sent from the dual point l. The k nearest neighbors are the first k lines encountered. Thus we can find these neighbors by *ray-shooting* in the dual arrangement.

This does not quite solve the problem we posed, because we have not dealt with the requirement that points be above or below the segment s, only that they be above or below the line containing s. To solve this we sort the points set by their horizontal projections. As well as building an arrangement and ray-shooting data structure for the entire point set, we also build structures for the first and second halves of the sorted lists of points, and so on recursively. Each data structure corresponds to a vertical slab of the points; for each such structure there are two smaller structures corresponding to slabs containing half as many points.

Then for each query segment s, we can find a set of $O(\log n)$ slabs that contain exactly those points above and below s. We can solve the k nearest vertical neighbors problem by performing ray-shooting queries within each slab, and selecting the best k points found.

We now describe the data structures which allow us to perform these ray-shooting operations and therefore find the nearest vertical neighbors. Rather than build a data structure that allows line segments to be tested in arbitrary order, we test the segments in left-to-right order of the points dual to the lines containing them. This allows us to perform our algorithm as a *plane sweep* of the dual line arrangement; i.e. we sweep a vertical line from left to right across the dual plane, and perform line segment neighbor finding queries and data structure updates as the vertical line crosses appropriate features in the arrangement. Such a plane sweep could be transformed into a static data structure using persistence techniques; however we do not need these techniques for our algorithms.

At any point in the algorithm, the sweep line will cross all of the n dual lines. Our data structure for a single slab (corresponding to a single dual line arrangement) simply consists of an array of n elements, listing those lines in the vertical order of their points of intersection with the sweep line. Then a vertical ray shooting query along the sweep line could be performed by a binary search in the array, to locate the starting point among the dual lines. Successive queries would then take $O(1)$ time by simply moving up to the next element in the array. The order of the dual lines changes exactly when the sweep line crosses an intersection between two dual lines. The change consists simply of swapping two adjacent elements in the array. The rays at which we wish to shoot correspond to segments xy, which are also found as the intersection of two lines dual to x and y. Each successive intersection can be found in $O(\log n)$ time, by keeping a priority queue of the $n - 1$ possible intersections between adjacent elements in the array.

Now let us consider putting several slabs together again. Starting each ray shooting query by binary searching in each slab separately would take $O(\log^2 n)$ time per segment. We can reduce this time, by using another data structure to relate locations in different slabs to each other. Recall that for each slab, corresponding to an arrangement of some m dual lines, there are two smaller slabs with $m/2$ dual lines each. The sweep line in the large slab is divided into $m + 1$ regions by the m lines crossing it. Each region in the large slab corresponds to part of a region in each of the smaller slabs. We keep another array, of $m + 1$ elements, listing the correspondence between regions of the large and small slabs. This correspondence only changes when an intersection occurs in the large slab; other intersections will rearrange the correspondence of lines to regions but will not change the numbering of the regions. For each intersection of two lines, we only need to update the correspondence for the region between the lines. Thus again each update takes $O(1)$ time per slab.

Now we can use these arrays to locate the point dual to each segment in the $O(\log n)$ slabs we wish to search. The point is an intersection in the outer slab containing all n points, and its location will already be known when the sweep line crosses that intersection. Then, while we have a location within a slab that contains points not above or below the queried segment, we need to move to locations in the two child slabs. This can be done simply by looking in the appropriate arrays, in constant time per move. In this way it takes $O(\log n)$ time to find the initial locations for ray shooting in each of the $O(\log n)$ appropriate slabs.

Once we have found the initial locations for vertical ray shooting, we can find each successive vertical neighbor of the segment simply by moving from element to adjacent element in the appropriate array. But we must somehow combine the neighbors found in different slabs. To do this, we use a final array, which tells us for each line the position of that line as it crosses the sweep line in the root slab of all n points. This array is updated as before by swapping two elements per intersection encountered. Using this array, we can compare neighbors from different slabs, by examining their positions in the sweep line. Each vertical neighbor for the line segment must be found by selecting among $O(\log n)$ candidates, one from each slab, each of which can be thought of as an integer having $O(\log n)$ bits, representing the position in the sweep line. This selection can be performed in constant time per operation, using the *atomic heap* data structure of Fredman and Willard [11].

Theorem 1. *Given n points, we can enumerate all point sets found as the k nearest vertical neighbors of each segment formed by a pair of points, in total time $O(kn^2 + n^2 \log n)$ and space $O(n \log n)$.* □

Proof: Each intersection of two dual lines, causing a search from the corresponding segment as well as updates to the data structures, can be selected in time $O(\log n)$ from a priority queue of possible intersections. Each search for k nearest neighbors takes time $O(\log n)$ to find the initial positions for the ray shooting, and $O(1)$ time per neighbor found. Each update takes constant time per slab, and involves changes in $O(\log n)$ slabs (only those slabs containing both dual lines that intersect to cause the update). Therefore the total time for searches and updates is $O(n^2(\log n + k))$. Each dual

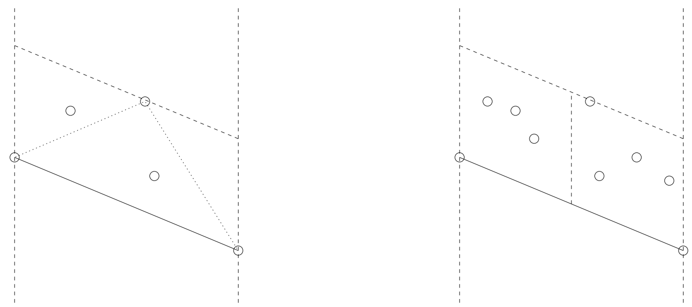

Figure 2. Nearest neighbors to xy: (a) five point set with triangle to furthest point; (b) seven nearest neighbors have five-point subset in one parallelogram.

line is involved in $O(\log n)$ slabs, and uses constant space per slab, so the total space is $O(n \log n)$. □

3 Minimum k-point sets

We now describe how to use the data structure of the previous section to find k-point sets with minimum area convex hulls. This problem is easier than that of finding minimum area k-gons, and serves as a warm-up to the k-gon question.

Lemma 1. *Let Q be the minimum area k-point set of some n point set P, and let x and y be the leftmost and rightmost points of Q respectively. Then each point in Q is one of the $2k-4$ nearest neighbors above or below line segment xy.*

Proof: Let z be the point with largest distance above xy in Q. Then the area of $conv(Q)$ is at least that of triangle xyz. Figure 2(a) shows a set of five points, with triangle xyz outlined.

Suppose z is not one of the $2k-4$ nearest neighbors. Then at least $2k-1$ points (including x, y, and z) are contained in the parallelogram with two vertical sides through x and y, one side equal to segment xy, and the remaining side parallel to xy and through z. Therefore at least k points must be contained in one of the two smaller parallelograms formed by dividing the large parallelogram in half vertically. Figure 2(b) shows $7 = 2 \cdot 5 - 3$ points above xy, five of which are in one small parallelogram. But the small parallelogram has area equal to that of triangle xyz, and therefore at most that of the original k-point set. So the convex hull of the points within the parallelogram has smaller area, and x, y, and z could not all be part of the minimum k-point set. □

This immediately gives us our algorithm for finding the minimum area k-point set. We simply examine each possible segment xy, and find the $2k-4$ neighbors above and below it. That gives us a set of $4k-6$ points, in which we can find the minimum k-point set in time $O(k^4)$ using the algorithm of Eppstein et al. [9]. Thus we achieve a total time of $O(k^4 n^2 + n^2 \log n)$.

The $O(k^4)$ term in this bound can be improved. First note that the convex hull of the minimum k-point set is the union of the convex hulls above and below xy, and the area of the convex hull is the sum of the areas above and below xy. Each of these two hulls must be a minimum j-point set, among those sets containing both x and y, for j equal to the number of points in the respective sets. So if we can compute all minimum such j-point sets, for $j < k$, we can combine the possible top and bottom sets in a further step taking time $O(k)$. In fact the algorithm of Eppstein et al. computes all these sets. Further, it works by trying all possible choices of the bottommost point in the set; for each such choice the algorithm takes $O(k^3)$ time. But in our situation we know the bottommost point: it is either x or y. Therefore we can compute the optimum k-point set among the neighbors of segment xy in time $O(k^3)$.

Theorem 2. *Given a set of n points, we can compute the k points minimizing the area of their convex hull, in time $O(n^2 \log n + k^3 n^2)$ and space $O(n \log n + k^3)$.* □

This improves the previous time bound of $O(kn^3)$ [9] whenever $k = O(\sqrt{n})$. The space used is always an improvement over the previous $O(kn^2)$.

4 Minimum convex k-gons

The proof of Lemma 1 was based on the fact that, if enough points are grouped in a small parallelogram, we can find a k-point set with convex hull area smaller than that of the parallelogram. For this purpose k points are enough, since any such k points have convex hull contained in the parallogram containing the points.

We wish to extend this result to the minimum convex k-gon problem. However it will clearly not always be the case that k points in a parallelogram form a convex set. Hence we need a result of the form that, if enough points are given, some subset of them forms a convex set. Such bounds are given by Ramsey theory; indeed the following was one of the seminal results in the development in Ramsey theory [12].

Lemma 2. (Erdös and Szekeres [10, 12]) *Given* $n = f(k) \leq \binom{2k-4}{k-2} + 1$ *points in general position, then some* k *points form the vertices of a convex* k*-gon.* □

This gives a bound of $f(k) = O(4^k)$. Graham et al. [12] also note a lower bound of $f(k) = \Omega(2^k)$, and cite as an open problem tightening the gap between these bounds. An improvement in the bound for $f(k)$ would also lead to a speed up in our algorithms.

Corollary 1. *If* x *and* y *are the leftmost and rightmost vertices of the minimum area convex* k*-gon formed by a set of points, then the other* $k-2$ *points are among the* $2f(k)-4$ *nearest neighbors above or below segment* xy*.*

Proof: The proof is the same as in Lemma 1: let z be the farthest point from xy, and suppose z is not one of the $2f(k)-4$ nearest neighbors. Then the area of the k-gon is at least that of triangle xyz, which is equal to that of two parallelograms, one of which contains at least $f(k)$ points. But then those $f(k)$ points contain a subset of k points forming a convex k-gon with smaller area than triangle xyz. □

Theorem 3. *The minimum area* k*-gon determined by a set of* n *points can be found in time* $O(n^2 \log n + 2^{6k} n^2)$*.*

Proof: As in the k-point set problem, we can use Corollary 1 to reduce the problem to one of finding minimum k-gons in $O(n^2)$ sets of $O(f(k))$ points each. As in the k-point set problem, this can be further reduced to combining j-gons, $j < k$, from each side of each segment xy. As in the k-point set problem, all minimum j-gons can be computed in time $O(f(k)^3)$, and the results can be combined in time $O(k)$. Therefore the total time is $O(n^2 \log n + f(k)^3 n^2) = O(n^2 \log n + 2^{6k} n^2)$. □

This is an improvement over the previous $O(kn^3)$ algorithm [9] when $k < (\log_2 n)/6$. The total space complexity is $O(n \log n + f(k)^2)$, improving the previous $O(n^2)$ bound.

References

[1] A. Aggarwal, H. Imai, N. Katoh and S. Suri. Finding k points with minimum diameter and related problems. *5th ACM Symp. Comput. Geom.* (1989) 283–291.

[2] A. Aggarwal, M.M. Klawe, S. Moran, P. Shor and R. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica* 2 (1987) 195–208.

[3] D. Avis and D. Rappaport. Computing the largest empty convex subset of a set of points. *1st ACM Symp. Comput. Geom.* (1985) 161–167.

[4] J.E. Boyce, D.P. Dobkin, R.L. Drysdale and L.J. Guibas. Finding extremal polygons. *SIAM J. Comput.* 14 (1985) 134–147.

[5] D.P. Dobkin, R.L. Drysdale and L.J. Guibas. Finding smallest polygons. *Adv. Computing Research, Vol. 1*, JAI Press (1983) 181–214.

[6] D.P. Dobkin, H. Edelsbrunner and M.H. Overmars. Searching for empty convex polygons. *4th ACM Symp. Comput. Geom.* (1988) 224–228.

[7] H. Edelsbrunner and L.J. Guibas. Topologically sweeping in an arrangement. *18th ACM Symp. Theory of Computing* (1986) 389–403.

[8] H. Edelsbrunner, J. O'Rourke and R. Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM J. Comput.* 15 (1986) 341–363.

[9] D. Eppstein, M. Overmars, G. Rote, and G. Woeginger. Finding minimum area k-gons. *Discrete Comput. Geom.*, to appear.

[10] P. Erdös and G. Szekeres. A combinatorial problem in geometry. *Compositio Math.* 2 (1935) 463–470.

[11] M.L. Fredman and D.E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *31st IEEE Symp. Found. Computer Science* (1990) 719–725.

[12] R.L. Graham, B.L. Rothschild, and J.H. Spencer. *Ramsey Theory*. Wiley, 1980.

[13] J.D. Horton. Sets with no empty convex 7-gons. *Canad. Math. Bull.* 26 (1983) 482–484.

[14] M.H. Overmars, B. Scholten and I. Vincent. Sets without empty convex 6-gons. *Bull. EATCS* 37 (1989) 160.