Discrete Comput Geom 7:45–58 (1992)

Discrete & Computational Geometry

Finding Minimum Area k-gons*

David Eppstein,[1] Mark Overmars,[2] Günter Rote,[3] and Gerhard Woeginger[3]

[1] Department of Computer Science, University of California, Irvine, CA 92717, USA

[2] Department of Computer Science, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, The Netherlands

[3] Institut für Mathematik, Technische Universität Graz, Kopernikusgasse 24, A-8010 Graz, Austria

Abstract. Given a set P of n points in the plane and a number k, we want to find a polygon $\mathscr{C}$ with vertices in P of minimum area that satisfies one of the following properties: (1) $\mathscr{C}$ is a convex k-gon, (2) $\mathscr{C}$ is an empty convex k-gon, or (3) $\mathscr{C}$ is the convex hull of exactly k points of P. We give algorithms for solving each of these three problems in time $O(kn^3)$. The space complexity is $O(n)$ for $k = 4$ and $O(kn^2)$ for $k \geq 5$. The algorithms are based on a dynamic programming approach. We generalize this approach to polygons with minimum perimeter, polygons with maximum perimeter or area, polygons containing the maximum or minimum number of points, polygons with minimum weight (for some weights added to vertices), etc., in similar time bounds.

1. Introduction

Given a set P of points in the plane, many papers have studied problems of determining subsets of points in P that form polygons with particular properties. One such problem deals with finding empty convex k-gons in a set of points (i.e., polygons that contain no points of P other than the k vertices). It is well known [12] that such k-gons might not exist for $k \geq 7$. Algorithms to find such k-gons have been presented in [4], [7], and [14]. The best-known result works for

---

* This paper includes work done while David Eppstein was at Columbia University, Department of Computer Science, and while Günter Rote and Gerhard Woeginger were at the Freie Universität Berlin, Fachbereich Mathematik, Institut für Informatik. Research was partially supported by the ESPRIT II Basic Research Actions Program of the EC under Contract No. 3075 (project ALCOM).

arbitrary k in time $O(T(n))$ where $T(n)$ is the number of empty triangles in the set, which varies between $O(n^2)$ and $O(n^3)$ [7].

Boyce *et al.* [5] treated the problems of finding maximum perimeter and maximum area convex k-gons. Their algorithms work in linear space and $O(kn \log n + n \log^2 n)$ time. Aggarwal *et al.* [2] improved these results to $O(kn + n \log n)$.

In applications like statistical clustering and pattern recognition, minimization problems tend to play a more important role than maximization problems. Minimization problems seem to be computationally harder than maximization problems in this context. Finding minimum perimeter k-gons was studied by Dobkin *et al.* [6]. Their $O(k^2 n \log n + k^5 n)$ algorithm was recently improved to $O(n \log n + k^4 n)$ by Aggarwal *et al.* [3]. This recent paper also studies problems like finding minimum diameter k-gons and minimum variance k-gons.

In this paper we concentrate on the problem of finding minimum area polygons. For the case $k = 3$ the problem asks for the minimum area empty triangle. An $O(n^2)$-time and $O(n^2)$-space algorithm for finding this triangle in a set of n points in the plane is given in Chapter 12.4 of Edelsbrunner's book [8]. The storage requirements of this method can be reduced to $O(n)$ using topological sweeping [9]. For $k > 3$ the best-known result was $O(n^k)$. In Problem 4(b) of [1] and Problem 12.10 of [8] the existence of an $o(n^4)$ algorithm for finding a minimum area convex 4-gon is stated as an open problem. (The problem of finding a minimum area quadrilateral without requiring convexity was also listed as open in [1] but this is trivially solved by finding, for each possible diagonal d, a minimum area triangle on each side of d, and then joining the two triangles to form a quadrilateral.) When $k > 3$ we have to define the problem more carefully. We can distinguish between the following three problems:

(1) Find a convex polygon $p_1, p_2, \ldots, p_k$ in P with minimum area.
(2) Find an *empty* convex polygon $p_1, p_2, \ldots, p_k$ in P with minimum area.
(3) Find a point set $\{p_1, p_2, \ldots, p_k\} \subseteq P$ such that the area of the convex hull of this subset is minimal

Note that when $k = 3$ all three problems are the same. (The smallest area triangle is obviously empty.) Note also that in the third problem this convex hull will only contain the points in the subset and no other points. So the subset is a kind of cluster.

Our new results (combined with the known results for $k = 3$) are summarized in Table 1.

Our algorithms are based on the dynamic programming approach.

The paper is organized as follows. Section 2 shows how to calculate, for all triangles determined by P, the numbers of points in their interiors. Section 3 describes space-efficient algorithms for $k = 4$, and Section 4 describes algorithms for general k. In Section 5 our technique is generalized to finding convex k-gons (and solving the other two problems) that minimize or maximize some general weight criterion. In this way we obtain solutions to, e.g., the minimum perimeter problem with the same time bounds as stated above, which is better than previous solutions [3] for large k. Another application finds the convex k-gon containing

Table 1

	$k = 3$		$k = 4$		$k \geq 5$	
	Time	Space	Time	Space	Time	Space
(1) Convex k-gon	n^2	n	n^3	n	kn^3	n^2
(2) Convex empty k-gon	n^2	n	n^3	n	kn^3	n^2
(3) Convex hull of k points	n^2	n	n^3	n	kn^3	kn^2

the smallest or largest number of points. Finally, in Section 6 we give some concluding remarks and directions for further research.

The following notations is used throughout this paper. By $l(p_1, p_2)$ we denote the directed line through the points p_1 and p_2 and by $\overline{p_1 p_2}$ we denote the line segment from p_1 to p_2. $conv(P)$ is the convex hull of the point set P. $\triangle p_1 p_2 p_3$ is the convex hull of the three points p_1, p_2, and p_3 (i.e., the triangle) and $\square p_1 p_2 p_3 p_4$ is the quadrangle formed by p_1, p_2, p_3, p_4 in clockwise order. $\angle p_1 p_2 p_3$ denotes the angle with apex p_2.

We assume throughout that the set of input points is in general position; i.e., no two points have the same x-coordinate, and no three points are in a line. The first restriction can be treated simply by rotating the coordinate system so that the new axes are not parallel to any line between two points; such a rotation can be found and performed in linear time. Relaxing the second restriction requires us to decide whether a convex k-gon is allowed to have some of its k vertices in line with each other; either choice is reasonable, and leads to different optimal polygons. We also repeatedly sort points by their angles around another point; this order is not well defined when three points are in line, and we have to specify what to do in this case. These modifications will be spelled out as appropriate.

2. The Number of Points in All Triangles

In this section we show how to preprocess the point set P in $O(n^2)$ time and $O(n^2)$ space such that the number of points inside any triangle in P can be determined in constant time. This result is used in Section 4. The structure derived by the preprocessing step is an array $stripe[p_i, p_j]$ that stores, for each pair of points (p_i, p_j) in P, the number of points in the vertical stripe below the line segment $\overline{p_i p_j}$. For point sets not in general position, we also store the number of points lying exactly on the line segment $\overline{p_i p_j}$. For a triangle $\triangle xyz$ with leftmost point x and rightmost point z, the number of points in it is equal to the absolute value of $stripe[x, y] + stripe[y, z] - stripe[x, z]$ (for an illustration, see Fig. 1).

To calculate the values in the array $stripe[*, *]$, we treat the line segments from left to right, according to their right endpoint. Line segments with the same right endpoint are treated in clockwise order. This gives the following algorithm (the cases (d1) and (d2) are illustrated in Fig. 2):

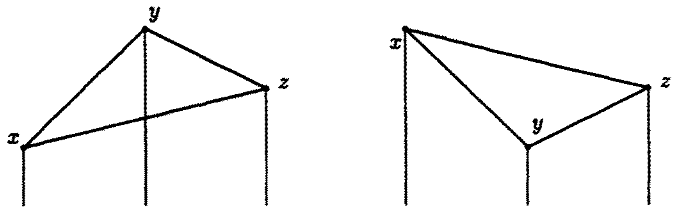

Fig. 1. The two possibilities for the triangle $\triangle xyz$.

Algorithm 1 (Calculating the number of points below each segment)

(a) Initialization. Set all the elements *stripe*$[*, *]$ to zero.
(b) Sort the points in P by x-coordinate from left to right. This gives the sequence $p_1, p_2, \ldots, p_n$.
(c) For each point $p_i \in P$, sort all the points lying left of p_i in clockwise order around p_i. This gives the sequences $p_1^i, p_2^i, \ldots, p_{i-1}^i$.
(d) For $p_i := p_2$ to p_n do
  For $j := 2$ to $i - 1$ do
  (d1) If p_j^i lies to the left of p_{j-1}^i then
    $stripe[p_j^i, p_i] := stripe[p_{j-1}^i, p_i] + stripe[p_j^i, p_{j-1}^i] + 1.$
  (d2) If p_j^i lies to the right of p_{j-1}^i then
    $stripe[p_j^i, p_i] := stripe[p_{j-1}^i, p_i] - stripe[p_j^i, p_{j-1}^i].$
  endfor.
endfor.

The correctness of Algorithm 1 is obvious from Fig. 2: As the points p_j^i are sorted in clockwise order around p_i and p_j^i is the direct successor of p_{j-1}^i in this ordering, the triangle $\triangle p_i p_j^i p_{j-1}^i$ must be empty. Hence, *stripe*$[p_j^i, p_i]$ is either the sum (in the case (d1)) or the difference (in the case (d2)) of *stripe*$[p_{j-1}^i, p_i]$ and *stripe*$[p_j^i, p_{j-1}^i]$. In the case (d1), the additional 1 appearing as a term in the sum corresponds to the point p_{j-1}^i. Moreover, for the calculation of some element in *stripe*, only values calculated previously are needed. The base cases of the recurrence, entries *stripe*$[p_1^i, p_i]$, are initialized to zero as part of step (a). Note that step (d) of the algorithm only fills the entries *stripe*$[p_i, p_j]$ with p_i left of p_j. The other entries can be filled at the same time.

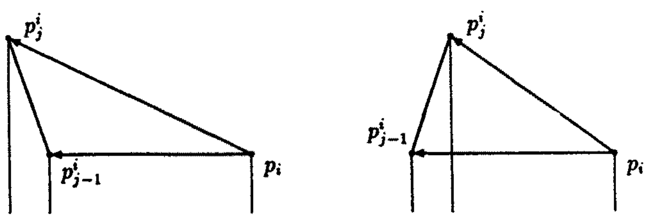

Fig. 2. The cases (d1) and (d2) in Algorithm 1.

For point sets not in general position, the sorting by clockwise order is not well defined. In this case we place nearby points earlier in the sorted order than farther points. Then if p_j^i and p_{j-1}^i are collinear with p_i, the computation of stripe $[p_j^i, p_i]$ should be performed as in case (d1) except that $+1$ is not added. We also remember the number of points on line segment $\overline{p_j^i p_i}$; if p_j^i and p_{j-1}^i are collinear this is one plus the number for $\overline{p_{j-1}^i p_i}$, otherwise it is zero.

Next, we consider the time and space complexity: step (a) takes $O(n^2)$ time and space and step (b) takes $O(n \log n)$ time and linear space. Applying the results of Edelsbrunner *et al.* [10], step (c) can be performed using only quadratic time and space. Finally, step (d) consists of two nested for-loops, and each substep in the loop is a simple addition or subtraction. Hence, step (d) costs at most $O(n^2)$ time and space, too.

Thus we have proved the following theorem:

Theorem 2.1. *We can preprocess a point set P in the plane in $O(n^2)$ time and space, such that afterward, for each triangle in P, the number of points in it can be determined in constant time.*

3. Finding a Minimum-Area Four-Point Set

We now solve the minimum area convex k-gon problem for $k = 4$. We first consider problem (3): Let P be a set of n points in the plane. Find a subset $Q \subset P$ of four points such that the area of $conv(Q)$ is minimized. The following two observations reduce the number of candidates for the set Q:

Observation 3.1. *Let Q be the area-minimizing set of four points for the point set P. Then the convex hull of Q does not contain any point of $P - Q$.*

Proof. Assume that $conv(Q)$ would contain at least five points in P. If we remove some extreme point from Q, we get a smaller area set. □

Observation 3.2. *The smallest area triangle in P containing at least one point is the same as (one of) the smallest area triangle(s) in P containing exactly one point, provided such triangles exist.*

Proof. Suppose triangle $\triangle abc$ contains at least two points d and e. Then the point e would lie in one of the smaller triangles $\triangle abd$, $\triangle acd$, or $\triangle bcd$. □

Hence, it suffices to search for smallest empty quadrangles and for smallest triangles with at least one other point in them. We first give a rough outline of the algorithm:

Algorithm 2 (Calculation of the minimum-area four-point set)

(A) For each $p \in P$, sort all other points by angle around p.
(B) For all pairs (x, y) of points in P do
Let P' be the set of points to the right of $l(x, y)$.
(B1) Find the smallest area triangle $\triangle xyz$ with $z \in P'$ and at least one other point u of P' in its interior.
(B2) Find the smallest area quadrangle $\square xuyz$ with $u \notin P'$, $z \in P'$, and all points above a horizontal line through y.
(C) Select the smallest area configuration of all the triangles $\triangle xyz$ and all the quadrangles $\square xuyz$ found in step (B).

Step (A) can be performed in time $O(n^2)$, but that algorithm uses $O(n^2)$ space. To reduce the space, we separately sort the points around each p, in time $O(n \log n)$; therefore the total time is $O(n^2 \log n)$. At most one such sorted ordering will be needed at a time. In the rest of this section we show how to carry out steps (B1) and (B2) in linear time, using the results of the preprocessing step (A). Since steps (B1) and (B2) are performed for each pair of points, step (B) takes $O(n^3)$ time all together. Step (C) uses $O(n^2)$ time, as the minimum of $O(n^2)$ values is calculated. Therefore, the whole algorithm runs in time $O(n^3)$. For correctness, note that for part (B1) there will always be two points x, y in the correct answer with the other points of the answer to the right of $l(x, y)$, and for part (B2) we can choose $\overline{xy}$ to be the diagonal through the bottommost point of the appropriate quadrilateral.

Problem (B1) is easy to solve: We are given a line segment $\overline{xy}$ and a point set P'. We want to find a point $z \in P'$ such that $\triangle xyz$ contains at least one point of P' and such that z minimizes the area of $\triangle xyz$ under this condition. $\triangle xyz$ contains another point $w \in P'$ exactly when (1) w is counterclockwise from z around x, and (2) w is clockwise from z around y. We step through the possible points z in clockwise order around x, starting at y, so we will have previously seen exactly those points satisfying condition (1). If any point w also satisfies condition (2), it will be the one with the smallest angle $\angle xyw$; this smallest angle can be easily maintained as we step through the points. Therefore, our algorithm is as follows:

Algorithm 2.1 (Solution of problem (B1))

Initialization: Set $MinAngle := \pi$ and $MinArea := \infty$.
For each point $z \in P'$ in clockwise order around x do
  $MinAngle :=$ min($MinAngle$, $\angle xyz$);
  If $MinAngle < \angle xyz$, then
    $MinArea :=$ min($MinArea$, area of $\triangle xyz$);
  endif;
endfor.

In problem (B2) we are given a segment $\overline{xy}$, and we wish to find the smallest area convex quadrangle $\square xuyz$ with u to the left of $l(x, y)$, z to the right of $l(x, y)$, and all points above a horizontal line through y. For each choice of z, the u forming the minimum area quadrangle is found simply by selecting the point to the left of $l(x, y)$ giving the smallest area for $\triangle xuy$. For the quadrangle to be convex, we need angles $\angle zxu$ and $\angle uyz$ to be convex; i.e., they must be less than π.

The requirement for $\angle uyz$ will always be satisfied if points x, u, and z are above y. The remaining requirement is that $\angle zxu$ be convex; this will be dealt with by processing all points, u and z together, in order by the slope of lines $l(x, u)$ or $l(x, z)$. If we sort these lines counterclockwise by slope, starting with lines nearly parallel to $l(x, y)$, then, for each point z to the right of $l(x, y)$, the points u already processed will be exactly those ones forming a convex quadrangle with z. Thus we need merely remember the u giving the smallest area triangle $\triangle xuy$, and this will also give the smallest quadrangle $\square xuyz$. The sorted order of line slopes we use is not the same as the order of points around x, but one order can be generated from the other in linear time; it also takes linear time to filter out those points below y.

We now describe our algorithm more formally.

Algorithm 2.2 (Solution of problem (B2))

Initialization: Set $TotalMinimum := \infty$ and $MinArea := \infty$.
Generate list of points z above y sorted by slope of $l(x, z)$.
For each point z in sorted order do
  If z is to the right of $l(x, y)$, then
    $TotalMinimum :=$ min($TotalMinimum$, $MinArea$ + area of $\triangle xyz$);
  else
    $u := z$;
    $MinArea :=$ min($MinArea$, area of $\triangle xuy$);
  endif;
endfor.

The actual optimal quadrilateral can be found by maintaining which value of u led to the current value of *MinArea*, and maintaining which values of u and z led to the current value of *TotalMinimum*.

Hence, we can give the following summarizing theorem:

Theorem 3.3. *Let P be a set of n points in general position in the plane. There is an algorithm that finds, in $O(n^3)$ time and $O(n)$ space, a subset Q of P of size four such that the area of conv(Q) is minimized.*

Proof. The time complexity is obvious, but the way the algorithm is stated above requires $O(n^2)$ storage. To get the claimed space complexity, we note that each computation for segment $\overline{xy}$ only requires the sorted order of points around point x. So for each point x we sort the other points in time $O(n \log n)$, then process all segments $\overline{xy}$ in time and space $O(n)$ each. This gives the claimed complexity. □

If the points are not in general position, the minimum k-point set may be four points in a line, or a triangle with a point on one of the sides. Such a set can be found analogously to the minimum area triangle algorithm [9], [10]. We compute the line arrangement dual to the point set; then four collinear points correspond to four coincident lines. A minimum area triangle with a point on one side corresponds to three coincident lines and the nearest line directly above or below their point of intersection. These configurations can be found in $O(n^2)$ time and $O(n)$ space using a topological sweep algorithm [9]. Algorithm 2.1 needs no modification for this possibility, and Algorithm 2.2 needs only to ignore pairs (x, y) having a third point on the segment $\overline{xy}$.

To find the smallest area convex 4-gon, we simply skip step (B1). This also gives the smallest area *empty* convex 4-gon, because the smallest area convex 4-gon is necessarily empty (a property that does not hold for $k > 4$).

4. Finding Minimum-Area Convex k-gons

In this section we first show how to find a smallest area convex k-gon in $O(kn^3)$ time and $O(n^2)$ space. This result is then extended to empty convex k-gons and to convex hulls of k points. The algorithm is based on the following observation. Let $\mathscr{C}$ be the minimum area convex k-gon. Let p_1 be the bottommost vertex of $\mathscr{C}$ and let p_2 and p_3 be the next two vertices in counterclockwise order. Now we can decompose $\mathscr{C}$ into the triangle $\triangle p_1 p_2 p_3$ and the remaining $(k-1)$-gon $\mathscr{C}'$. Now obviously $\mathscr{C}'$ is minimal among all $(k-1)$-gons with p_1 as bottommost vertex, p_3 as next vertex, and all points on one side of the line $l(p_3, p_2)$. So we could compute $\mathscr{C}'$ for any possible p_1, p_2, and p_3 and take the minimum of all possibilities. This suggests a dynamic programming approach.

To be precise, we construct a four-dimensional array AR such that the element $AR[p_i, p_j, p_l, m]$ contains the area of the smallest convex m-gon $\mathscr{C}$ such that (see Fig. 3):

- point p_i is the bottommost vertex,
- point p_j is the next vertex in counterclockwise order, i.e., all points of $\mathscr{C}$ lie to the left of the line $l(p_i, p_j)$, and
- all points of $\mathscr{C}$ lie on the same side of $l(p_j, p_l)$ as p_i.

The minimum area convex k-gon is just the minimum of the $O(n^3)$ values $AR[*, *, *, k]$. Thus, our goal is to fill this array up to $m = k$. This is done in the following way:

In the initialization step we set all array elements $AR[*, *, *, 2]$ to zero (as the

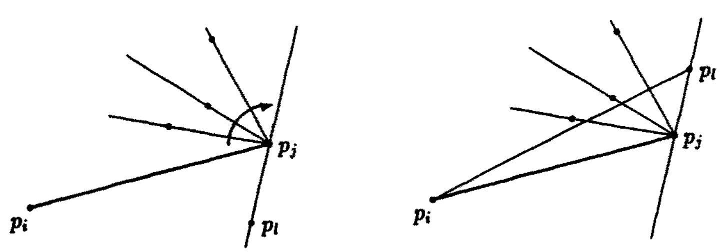

Fig. 3. How to treat the point p_l.

area of a 2-gon is always zero). Moreover, we sort, for each point p in P, all the other points in clockwise order around p and store these orderings. As in Algorithm 2.2, we do not sort the *half-lines* going from p through p_i by direction but sort the *lines* going through p and p_i by slope, i.e., we do not distinguish on which side of p the point p_i lies on $l(p, p_i)$.

Now assume we have already filled all entries in AR with last index $\leq m - 1$. We describe how to fill all the entries for m for some fixed points p_i and p_j in linear time (that means only the point p_l is left to vary). Obviously, this leads to an $O(kn^3)$ total time bound.

We treat the possible points p_l in clockwise order around p_j (in the ordering by the slope of lines calculated in the initialization step). We start with the successor of p_i. The basic idea is that when we treat a point in this ordering as candidate for p_l, the minimum m-gon corresponding to $AR[p_i, p_j, p_l, m]$ is either the same as for p_l's predecessor in the ordering or it involves p_l as a new neighbor of p_j. These two possible cases are shown in Fig. 3.

- If p_l lies to the right of the line segment $\overline{p_i p_j}$, then no new point can be used. $AR[p_i, p_j, p_l, m]$ is equal to $AR[p_i, p_j, pred(p_l), m]$ and nothing changes.
- If p_l lies to the left of $\overline{p_i p_j}$, then p_l might be a vertex of the minimum area polygon. In this case the area is composed of the triangle $\triangle p_i p_j p_l$ and the minimum area $(m - 1)$-gon in $AR[p_i, p_l, p_j, m - 1]$. Hence, we set $AR[p_i, p_j, p_l, m]$ to the minimum of this value and $AR[p_i, p_j, pred(p_l), m]$.

Thus, for each point p_l, we have to do $O(1)$ work checking the two areas and this gives a total amount of $O(n)$ time. Summarizing, the algorithm is as follows:

Algorithm 3 (Finding the smallest k-gon)

```
TotalMinimum := ∞;
For all points p_i do
  AR[p_i, *, *, 2] := 0;
  For m := 3 to k do
    For all points p_j above p_i, in clockwise order around p_i, do
      MinArea := ∞;
      For all points p_l, in clockwise order of the directions of the lines
           l(p_j, p_l), as described in the text, do
        If p_l is to the left of ‾p_i p_j‾, then
          MinArea := min(MinArea,  AR[p_i, p_l, p_j, m − 1] + area  of
                 △p_i p_j p_l);
        endif;
        AR[p_i, p_j, p_l, m] := MinArea;
      endfor;
    endfor;
  endfor;
  TotalMinimum := min(TotalMinimum, min AR[p_i, *, *, k]);
endfor.
```

Theorem 4.1. *The minimum area* (1) *convex k-gon, or* (2) *empty convex k-gon can be found in $O(kn^3)$ time and $O(n^2)$ space. The minimum area* (3) *convex hull of k points can be found in $O(kn^3)$ time and $O(kn^2)$ space.*

Proof. (1) The time complexity of $O(kn^3)$ follows from above. For the space complexity we observe that we do not have to store the complete four-dimensional array AR: For the calculation of the values $AR[p_i, *, *, m]$ only the values $AR[p_i, *, *, m-1]$ are needed. After having computed $AR[p_i, *, *, *]$ we can compute the minimum of $AR[p_i, *, *, k]$ and recover the optimal solution by backtracking the computation that lead to the optimal value. However, this would lead to $O(kn^2)$ space complexity. If we are only interested in the optimal *value*, we can forget $AR[p_i, *, *, m-1]$ after computing $AR[p_i, *, *, m]$, which reduces the storage by a factor of k. Alternatively, we could use a trick in order to get the optimal solution with $O(n^2)$ space: In a first pass, we compute the optimal value as above. This tells us the indices i, j, and l for which $AR[p_i, p_j, p_l, k]$ is optimal. Then we must backtrack through AR one step at a time in order to reconstruct the optimal k-gon. Backtracking from $AR[p_i, *, *, m]$ can be accomplished by recomputing $AR[p_i, *, *, m-1]$, in time $O(kn^2)$. The total time to reconstruct the solution is $O(k^2n^2) = O(kn^3)$. The bound could be reduced to $O(kn^2 \log k)$ by backtracking from m to $m/2$ and recursively solving two smaller backtracking problems, as in [13], but this is not necessary here.

(2) The only difference from case (1) is that we have to take care that the polygons we get are empty. Applying the results of Section 2, we preprocess the point set P in $O(n^2)$ time and space. Afterward, only empty triangles are used to compose the minimum area polygons, i.e., the line $(*)$ in the algorithm is executed only when the triangle $\triangle p_i p_j p_l$ is empty.

(3) In this case, the meaning of $AR[p_i, p_j, p_l, m]$ has to be changed: The first three indices have the same significance as previously, but m is the total number of points contained in the polygon, i.e., vertices and points inside. We again preprocess P in $O(n^2)$ time and space to be able to determine the number of points in each triangle in constant time. Then, in a similar way as above, we can calculate the values $AR[p_i, *, *, m]$ from the values $AR[p_i, *, *, m']$, with $m' < m$. Line $(*)$ in the algorithm is replaced by

$$(*')\quad \begin{cases} s := \text{the number of points inside } \triangle p_i p_j p_l; \\ \text{if } s \le m \text{ then} \\ \quad MinArea := \min(MinArea, AR[p_i, p_l, p_j, m-s] + \text{area of } \triangle p_i p_j p_l); \\ \text{endif}; \end{cases}$$

This time we have to store the whole three-dimensional array $AR[p_i, *, *, *]$ under all circumstances, even if we are only interested in the value of the optimum. □

Let us finally examine the possibility that the point set is not in general position. The main question is how to sort points p_l around p_j by the slopes of the lines

$l(p_j, p_l)$, when some of those lines may be identical. It only matters whether to sort the points left of $l(p_i, p_j)$ before the points to the right, or vice versa; this is because points on the same side of $l(p_i, p_j)$ do not interact with each other. Placing the left points first corresponds to allowing points to be counted as polygon vertices when they are in the middle of a side. Placing right points first corresponds to only counting vertices that have angles less than π. For the k-point set and empty convex k-gon problems, we check if the line segments bounding the possible polygons contain any points, as described in Section 2, and take appropriate action depending on whether we want to count such points as inside or outside the polygon.

5. Other Weight Functions

The method presented in the previous section can be used to solve many other types of minimization and maximization problems as well. To this end let W be some weight function that assigns a real weight to any (convex) polygon $\mathscr{C}$.

Definition 5.1. A weight function W is called *decomposable* iff for any polygon $\mathscr{C} = \langle p_1, \ldots, p_m \rangle$ and any index $2 < i < m$

$$W(\mathscr{C}) = \diamond(W(\langle p_1, \ldots, p_i \rangle), W(\langle p_1, p_i, p_{i+1}, \ldots, p_m \rangle), p_1, p_i),$$

where $\diamond$ takes constant time to compute. W is called *monotone decomposable* iff $\diamond$ is monotone in its first (and, hence, in its second) argument.

In other words, when W is decomposable we can cut the polygon $\mathscr{C}$ in two subpolygons along the line segment $\overline{p_1 p_i}$ and obtain the weight of $\mathscr{C}$ from the weights of the subpolygons and some information on the cut segment. For example, the area of a polygon is a monotone decomposable weight function with $\diamond(x, y, p, q) = x + y$. Many different monotone decomposable weight functions exist. For example:

- The perimeter. Here $\diamond(x, y, p, q) = x + y - 2|\overline{pq}|$.
- Sum of the internal angles. $\diamond(x, y, p, q) = x + y$. This turns out to be simply $(k - 2)\pi$, so optimizing this quantity is not very interesting.
- Number of points of some set in the interior. $\diamond(x, y, p, q) = x + y$.
- Adding a weight $w(p)$ to each point p we can take as the weight of a polygon the sum of the weights of its vertices. $\diamond(x, y, p, q) = x + y - w(p) - w(p)$. Similarly we can take the maximal or minimal weight of the points as weight of the polygon.

Theorem 5.2. *Let W be a monotone decomposable weight function. Let P be a set of n points. The* (1) *convex k-gon,* (2) *empty convex k-gon,* (3) *convex hull of k points in P that minimizes or maximizes W can be computed in time $O(kn^3 + G(n))$ time, where $G(n)$ is the time required to compute W for the $O(n^3)$ possible triangles in the set.*

Proof. The method is the same as in the previous section with the obvious modifications. The time bound follows. To prove the correctness, assume that some polygon $\mathscr{C}$ is the optimal solution. Let $\mathscr{C} = \langle p_1, \ldots, p_k \rangle$ with p_1 the bottommost vertex. Now split $\mathscr{C}$ in $\mathscr{C}' = \langle p_1, p_3, \ldots, p_k \rangle$ and the triangle $\triangle p_1 p_2 p_3$. Because W is monotone $\mathscr{C}'$ must be optimal among all $k-1$ gons with p_1 and p_3 as first two vertices, to the left of $l(p_2, p_3)$. Hence, the method correctly finds $\mathscr{C}$. □

Note that monotonicity of W is essential for the method to be correct. Decomposibility is also necessary: we can use various data structures to maintain nondecomposible functions such as the diameter or the smallest angle, but the proof above will fail because $\mathscr{C}'$ need not be optimal.

The next result follows immediately from Theorem 5.2.

Corollary 5.3. *The minimum perimeter* (1) *convex k-gon,* (2) *empty convex k-gon,* (3) *convex hull of k points can be determined in time $O(kn^3)$.*

The method can also maximize area or perimeter but the bounds will be worse than the methods of [2].

Corollary 5.4. *Given a set P of n points, the convex k-gon with vertices in P containing the minimum or maximum number of points of P in its interior can be determined in time $O(kn^3)$.*

Proof. From Theorem 2.1 it follows that $G(n) = O(n^3)$. □

All other weight functions listed above can also be minimized or maximized. In all cases the time bound will be $O(kn^3)$. Storage for all these problems can be kept to $O(n^2)$ for problems (1) and (2), and to $O(kn^2)$ for problem (3).

With some slight modifications weight functions like the length of the longest or shortest edge can also be treated (although they are not decomposable) in the same bounds. The key insight is that, in our dynamic program, all edge lengths of previous polygons are preserved except for the bottom right edge. If we maintain the extremal edge length or angle among those in the rest of the polygon, this will behave like a decomposable function for the purposes of our algorithm. Then optimization of the bottom right edge can be included only in the last stage of the dynamic program.

6. Conclusions

In this paper we have given $O(kn^3)$ algorithms for solving three different types of minimum area k-point set problems. The methods use $O(n)$ storage when $k = 4$ and $O(n^2)$ or $O(kn^2)$ storage when $k > 4$. The methods are based on the dynamic programming technique, using some special properties of minimum area polygons.

The technique was generalized to solve a large class of minimization (and maximization) problems involving some weight function on the polygons obtained. In this way, for example, solutions were obtained for the minimum perimeter problem.

Many open problems remain. Although our method can solve many types of minimization problems, as shown in Section 5, some problems cannot be solved with it. In particular, problems with a nonlocal weight criterion, such as the minimum diameter, do not fit in the scheme. It is open whether dynamic programming can be used to solve those problems as well.

It is unclear whether our algorithms are optimal. The only lower bounds known for the problems are $\Omega(n \log n)$. If all n points are extreme, it is easy to see that the minimum area k-point sets can be found in $O(kn^2)$ time. So improvement might be possible. Also improving the space bound to $O(n)$ for all k is open. Recently the first author has made some progress: the minimum area k-point set and convex k-gon problems can be solved for any fixed k in time $O(n^2 \log n)$ and space $O(n \log n)$ [11]. The minimum area empty convex k-gon problem, and the problems of optimization with other weight functions, remain open.

A final open problem concerns nonconvex polygons. Rather than asking for the minimum area convex k-gon we could simply ask for the minimum area k-gon. For $k > 3$ this indeed need not be convex. As noted in the introduction, for $k = 4$ this problem is easy to solve in time $O(n^2)$. At first glance a method similar to the one proposed in Section 4 might seem to work for general k but this is not true. The problem is that the polygon might become self-overlapping. Indeed, it is easy to find examples where the smallest k-gon is self-overlapping. Avoiding these polygons seems very hard.

Acknowledgments

We would like to thank Helmut Alt and Emo Welzl for many helpful discussions.

References

1. A. Aggarwal and J. Wein, *Computational Geometry*, Lecture Notes for 18.409, MIT Laboratory for Computer Science, 1988.
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. A. Aggarwal, H. Imai, N. Katoh, and S. Suri, Finding k points with minimum diameter and related problems, *Proc. 5th ACM Symp. on Computational Geometry*, 1989, pp. 283–291.
4. D. Avis and D. Rappaport, Computing the largest empty convex subset of a set of points, *Proc. 1st ACM Symp. on Computational Geometry*, 1985, pp. 161–167.
5. J. E. Boyce, D. P. Dobkin, R. L. Drysdale, and L. J. Guibas, Finding extremal polygons, *SIAM J. Comput.* **14** (1985), 134–147.
6. D. P. Dobkin, R. L. Drysdale, and L. J. Guibas, Finding smallest polygons, In: *Advances in Computing Research*, Vol. 1, JAI Press, Greenwich, CT, 1983, pp. 181–214.
7. D. P. Dobkin, H. Edelsbrunner, and M. H. Overmars, Searching for empty convex polygons, *Proc. 4th ACM Symp. on Computational Geometry*, 1988, pp. 224–228.
8. H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, EATCS Monographs on Theoretical. Computer Science, Springer-Verlag, Berlin, 1987.

9. H. Edelsbrunner and L. J. Guibas, Topologically sweeping in an arrangement, *Proc. 18th ACM Symp. on Theory of Computing*, 1986, pp. 389–403.
10. H. Edelsbrunner, J. O'Rourke, and R. Seidel, Constructing arrangements of lines and hyperplanes with applications, *SIAM J. Comput.* **15** (1986), 341–363.
11. D. Eppstein, New algorithms for minimum area k-gons, *Proc. 3rd ACM/SIAM Symp. on Discrete Algorithms*, 1992, to appear.
12. J. D. Horton, Sets with no empty convex 7-gons, *Canad. Math. Bull.* **26** (1983), 482–484.
13. J. I. Munro and R. J. Ramirez, Reducing space requirements for shortest path problems, *Oper. Res.* **30** (1982), 1009–1013.
14. M. H. Overmars, B. Scholten, and I. Vincent, Sets without empty convex 6-gons, *Bull. EATCS* **37** (1989), 160–160.

Received April 6, 1989, and in revised form July 1, 1991.