Finding minimum area k-gons

M.H. Overmars, G. Rote, G. Woeginger

RUU-CS-89-7
March 1989

Utrecht University

Department of Computer Science

Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : ... + 31 - 30 - 531454

Finding minimum area k-gons

M.H. Overmars, G. Rote, G. Woeginger

Technical Report RUU-CS-89-7
March 1989

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands

ISSN:0924-3275

Finding Minimum Area k-gons

Mark Overmars *
Günter Rote †‡
Gerhard Woeginger †

Abstract

Given a set P of n points in the plane and a number k, we want to find a polygon C with vertices in P of minumum area that satisfies one of the following properties: (1) C is a convex k-gon, (2) C is an empty convex k-gon, or (3) 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 the dynamic programming approach. We also generalize this approach to polygons with many other properties like e.g. minimum perimeter, maximum perimeter and area, containing the maximum or minimum number of points, of 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 of such problems deals with finding empty convex k-gons in a set of points. It is well-known ([9]) that such k-gons might not exist for $k \geq 7$. Algorithms to find such k-gons have been presented in [3, 6, 11]. The best known result works for 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 $O(n^3)$.

Boyce, Dobkin, Drysdale, and Guibas [4] 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, Klawe, Moran, Shor, and Wilber [1] improved these results to $O(kn + n \log n)$.

*Department of Computer Science, University of Utrecht, P.O.Box 80.089, 3508 TB Utrecht, the Netherlands

†Fachbereich Mathematik, Freie Universität Berlin, Arnimallee 2-6, D-1000 Berlin 33

‡on leave from the Institut für Mathematik, Technische Universität Graz, Austria

In application 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, Drysdale and Guibas [5]. Their $O(k^2 n \log n + k^5 n)$ algorithm was recently improved to $O(n \log n + k^4 n)$ by Aggarwal, Imai, Katoh, and Suri [2]. This recent paper also studies problems like finding minimum diameter k-gons and minimum variance k-gons.

In this paper we will 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 [7]. For $k > 3$ the best known result was $O(n^k)$. (In [7] the existence of an $o(n^4)$-algorithm for finding a minimum area convex 4-gon is stated as an open problem (Problem 12.10).) When $k > 3$ we have to define the problem more carefully. We can distinguish between the following three problems (throughout the paper, let P be a set of n points in the plane in general position):

(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 the following table.

	$k = 3$	$k = 4$	$k \geq 5$
	time / space	time / space	time / space
(1) convex k-gon	n^2 , n^2	n^3 , n	kn^3 , kn^2 (n^2)
(2) convex empty k-gon	n^2 , n^2	n^3 , n	kn^3 , kn^2 (n^2)
(3) convex hull of k points	n^2 , n^2	n^3 , n	kn^3 , kn^2

In the case $k \geq 5$, the space requirement is smaller if we only want to compute the *value* of the smallest area, not the polygon itself. This is indicated in parentheses in the last column.

The paper is organized as follows.

In section 3 we show how to solve the three problems in the case $k = 4$ in the stated time and space bounds. The results are based on the concept of a *type-0 chain* that is introduced in section 2.

Section 4 shows how to calculate for all triangles determined by P the number of points in their interior. This will be used in section 5 to obtain our efficient algorithm for $k > 4$.

The method is based on the dynamic programming approach. In section 6 this 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 [2] for large k. Another application finds the convex k-gon containing the smallest or largest number of points.

Finally, in section 7 we give some concluding remarks and directions for further research.

The following notations will be 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. $\Delta p_1 p_2 p_3$ 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 counter-clockwise order. $\angle p_1 p_2 p_3$ denotes the angle with apex p_2.

2 Constructing Type-0 Chains

We start this section by giving a key definition of the paper.

Definition 2.1 *Let $\overline{xy}$ be a line segment. Let P be a set of points lying to the left of $l(x, y)$. We say, a point $p \in P$ is of* type i *with respect to $\overline{xy}$ and P, iff the triangle Δxyp contains exactly i points of P in its interior.*

In this section we will show how the set of all type-0 points for a given line segment $\overline{xy}$ can be computed efficiently. We start with deriving a number of properties for type-i point sets. First, let us fix the line segment $\overline{xy}$ and the point set P. Thus if we speak of a type-i point in the following, we always mean a type-i point with respect to this $\overline{xy}$ and P. W.l.o.g. we assume that $\overline{xy}$ is horizontal and P consists of points lying above $\overline{xy}$.

Lemma 2.2 *(i) If p_1 is a type-i point, then the triangle Δxyp_1 does not contain any type-j point p_2 in its interior, with $j \geq i$.*

(ii) For two type-i points p_1, p_2 in P, $\angle p_1 xy > \angle p_2 xy$ holds if and only if $\angle p_1 yx < \angle p_2 yx$.

(iii) If we sort the set of type-i points by decreasing angle $\angle pxy$, we get the same sequence as if we sort them by increasing angle $\angle pyx$.

Proof:

(i) If the type-j point p_2 lies in the interior of Δxyp_1, the whole triangle Δxyp_2 is a subset of Δxyp_1. Hence, the j points in Δxyp_2 together with the point p_2 lie in Δxyp_1. This implies that $i \geq j+1$ must hold.

(ii) Assume, $\angle p_1xy > \angle p_2xy$ and $\angle p_1yx > \angle p_2yx$ holds. But then, obviously, p_2 must lie in the interior of Δxyp_1 and this is a contradiction to (i).

(iii) A straightforward consequence of (ii). □

Because of (iii), the following definition makes sense.

Definition 2.3 *Let $\langle c_1^i, c_2^i, \ldots, c_m^i \rangle$ be the sequence of type-i points, sorted by decreasing $\angle pyx$ or by increasing $\angle pxy$. We connect the points c_j^i and c_{j+1}^i by straight line segments. The polygonal curve we obtain is called the* type-i chain *C^i of P with respect to $\overline{xy}$.*

We now give a sequential algorithm that constructs the type-0 chain for a point set P of n points and a segment $\overline{xy}$ in $O(n \log n)$ time. The points are sorted around x and are then visited sequentially. The type-0 chain C^0 under construction is stored as a linear list. The variable *low* always contains the point p with smallest angle $\angle xyp$ among the points considered so far.

Algorithm 1 (Construction of type-0 chains)

(a) Sort the points in P around x by increasing angle $\angle pxy$. This gives the sequence $p_1, p_2, \ldots, p_n$.

(b) Initialization: $C^0 := \langle p_1 \rangle$, $low := p_1$.

(c) For $p := p_2$ to p_n do

If *low* is not contained in Δxyp then add p to the end of C^0.

If $\angle xyp < \angle(x, y, low)$ then $low := p$.

Step (a) takes $O(n \log n)$ time, Step (b) takes constant time. Step (c) consists of n substeps and each substep can be performed in constant time. Hence, the algorithm has an overall running time of $O(n \log n)$.

It remains to show that the algorithm works correctly and indeed finds the type-0 chain. Obviously, the point p_1 is a type-0 point: For any point q in the interior of Δxyp_1, the angle $\angle qxy$ has to be smaller than $\angle p_1xy$. As p_1 has the smallest angle there is no such point and, hence, Δxyp_1 is empty.

Assume the algorithm has reached some point p and *low* is maintained as indicated. Consider the point p and its relation to the point *low*. The two possible cases are shown in Figure 1. If *low* lies in the interior of Δxyp, then p is clearly at least of type 1. If *low* does not lie in the interior of Δxyp, then p is of type 0: Any point q in the interior of Δxyp must have a smaller angle $\angle qxy$ and a smaller $\angle qyx$. But *low* is the point with smallest angle $\angle xyq$ among the points q with $\angle qxy < \angle pxy$.

Theorem 2.4 *Let $\overline{xy}$ be a line segment, let P be a set of points all lying to the left of $l(x,y)$. Then the type-0 chain can be constructed in $O(n \log n)$ time. If the points in P are given in sorted order around x, then the type-0 chain can be constructed in $O(n)$ time.*

3 Finding a Minimum-Area Four-Point-Set

We will 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 of 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: Assume, the smallest area triangle Δabc containing at least one point would contain at least two points d, e. Then the point e would lie in one of the smaller triangles Δabd, Δacd or Δbcd. □

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

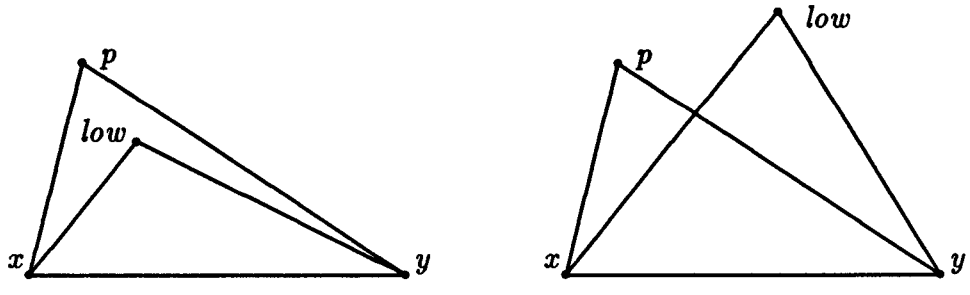

Figure 1: The two possibilities for the point p_{k+1} in Algorithm 1

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 left of $l(x, y)$.

(B1) Find the smallest area triangle Δxyz with $z \in P'$ and at least one other point u of P' in its interior.

(B2) Find the smallest area quadrangle $\square xyzu$ with $z, u \in P'$ and no other point of P' in its interior.

(C) Select the smallest area configuration of all the triangles Δxyz and all the quadrangles $\square xyzu$ found in step (B).

Step (A) takes time $O(n^2 \log n)$. In the rest of this section, we will 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 the correctness note that there will always be two points x, y in the correct answer with the other points of the answer to the left of $l(x, y)$.

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 Δxyz contains at least one point of P' and such that z minimizes the area of Δxyz under this condition. As the length of the triangle's baseline $\overline{xy}$ is fixed, its area becomes minimal iff its height becomes minimal. The triangle's height is the distance of the point z from $l(x, y)$. Therefore, our algorithm is as follows:

Algorithm 2.1 (Solution of problem B1)
Construct the type-0 chain C^0 for P'.
Find the point z in $P' - C^0$ with the smallest distance from $l(x, y)$.
Output the area of Δxyz.

Problem (B2) is more complicated. Of course, we can use the type-0 chain again. As $\square xyzu$ must be empty, the only potential candidates for z and u are the points on the type-0 chain $C^0 = \langle c_1, c_2, \ldots, c_m \rangle$. But now there are $O(n^2)$ possibilities to choose z and u, whereas we want to do it in $O(n)$ time.

Assume, the point $z = c_i$ has already been chosen and we want to find the point u that minimizes the area of $\square xyzu$. Let $C'(z) = \langle c_1, c_2, \ldots, c_{i-1} \rangle$ denote the points on C^0 that lie before z. The area of $\square xyzu$ is composed of the areas of Δxyz and Δxzu (see Figure 2). As z is fixed, the area of Δxyz and the length of $\overline{xz}$ are fixed, too. Consequently, the area of $\square xyzu$ is minimized iff the distance of u to the line $l(x, z)$ is minimized. Hence, our problem is:

For each point z on $C^0 = \langle c_1, c_2, \ldots, c_m \rangle$, find the point u on $C'(z)$ that minimizes the distance of u to $l(x, z)$.

It is easy to see that only the points on the convex hull of $C'(z)$ are candidates for the point u. This suggests the following approach: We treat the points in C^0 in rotational order and maintain the convex hull of the points passed. Determining the point on the convex hull nearest to $l(x, z)$ is easy. Unfortunately this would require $O(n \log n)$ time. To avoid this we restrict our set of possible candidates even further.

First, we can throw away the upper chain of the convex hull, (the convex hull is divided into the *upper chain* and the *lower chain* by the first point c_1 and the last point c_{i-1}) as there is always some point on the lower chain that is closer to the sweepline than a point on the upper chain. Second, consider the following situation: While sweeping over the chain C^0, the angle $\angle zxy$ decreases from π down to 0. Now at some fixed moment, the sweepline becomes parallel to some edge $\overline{c_i c_j}$ of the hull. In the next moment, the slope of the line decreases and from now on, the left endpoint c_i will always be closer to the sweep line than the right endpoint c_j. Hence, we can throw away the right point c_j ("left" and "right" always refer to the positions on C^0).

For an example, see Figure 2: The convex hull of $C'(z)$ consists of the points $\langle c_1, c_5, c_6, c_3, c_2 \rangle$, the point c_4 lies in the interior. $\langle c_2, c_3 \rangle$ constitutes the upper chain of the hull and, hence, can be removed. At some moment, the sweepline has been parallel to $\overline{c_5 c_6}$. Consequently, we can throw away the point c_6 and the remaining points $\langle c_1, c_5 \rangle$ form the pruned sequence $C''(z)$.

$C''(z)$ has some nice properties: It consists of a convex sequence of line segments. It starts at c_1. The slope of the segments increases from left to right, but does not exceed the slope of the line $l(x, z)$. It is easy to see that the rigthmost point of $C''(z)$ is the closest point to $l(x, z)$.

This leads to the following algorithm: $C''(z)$ is stored in a stack $\mathcal{S}$. The variable *last* always contains the last element of the stack $\mathcal{S}$, variable *lbo* contains the last but one element of $\mathcal{S}$. We will use the standard operations PUSH and POP. Variable

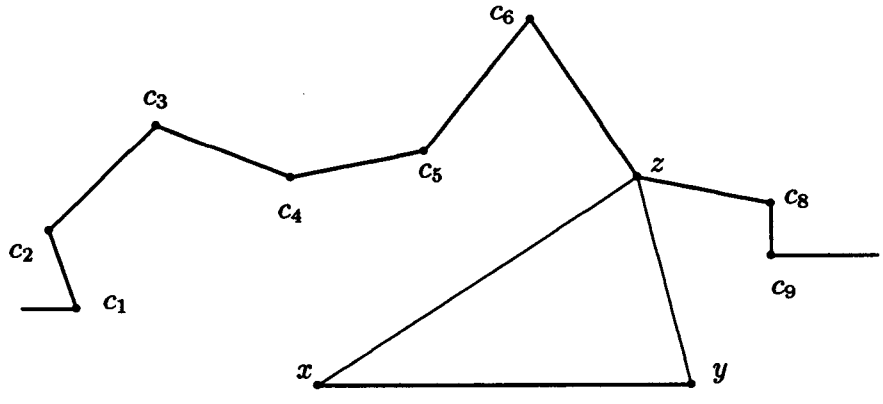

Figure 2: The segment $\overline{xy}$, its type-0 chain and the triangle Δxyz

Minarea stores the smallest area of configurations checked until now:

Algorithm 2.2 (Solution of problem B2)

(1) Initialization: PUSH($\mathcal{S}, c_1$) and set *Minarea* := ∞.

(2) For $z := c_2$ to c_m do

(2.1) while [*lbo* is closer to $l(x,z)$ than *last*] do POP($\mathcal{S}$);
(2.2) *Minarea* := min(*Minarea*, area of $\square(x,y,z,last)$;
(2.3) while [*last* lies to the left of $l(lbo,z)$] do POP($\mathcal{S}$);
(2.4) PUSH($\mathcal{S}, z$);

endfor.

By the preceeding observations the correctness of Algorithm 2.2 is easily established: The while-loop in (2.1) throws away points of segments that were parallel to the sweep line since the previous point z treated; the while-loop (2.3) removes points that do not lie on the lower chain of the convex hull. Also the time complexity is easy to determine. Every point in C^0 is pushed, respectively popped, at most once. Hence, the algorithm runs in linear time, provided the type-0 chain is available (which it is because we computed C^0 in Algorithm 2.1).

The last thing that remains to show is that the quadrangle $\square xyzu$ found by Algorithm 2.2 is indeed empty. Remember: Type-0 points z and u only guarantee that the triangles Δxyz and Δxyu are empty. But $\square xyzu$ does not consist of only these two triangles (see Figure 3).

Fortunately, we can show that the remaining part is empty, too. Assume, it were not. Consider the point q in the part that is closest to $l(x,y)$. Obviously, Δxyq is empty and thus q is of type 0. q is contained in Δxzu and is therefore closer to $l(x,z)$ than u. But this contradicts with the fact that u is the type-0 point closest to $l(x,z)$.

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.*

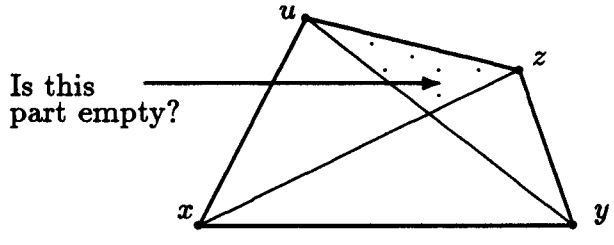

Figure 3: The quadrangle $\square xyzu$ that has been found by Algorithm 2.2

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 mix the constructions of the type-0 chains and the area-calculations. First, we treat all segments $\overline{p_1 p_i}$ incident to p_1. Observe, that the space used in this step is linear, as we only have to store the sequence of points sorted around p_1. Then we treat all segments incident to p_2, p_3 and so on. This gives the claimed space complexity. □

To find the smallest area convex 4-gon, we simply skip Step (B1). It is easy to see that 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 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, afterwards, the number of points inside any triangle in P can be determined in constant time. This result will be used in Section 5. The strucure derived by the preprocessing step is a two-dimensional 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 $\overline{p_i p_j}$. Obviously, for a triangle Δxyz 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 Figure 4).

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 Figure 5):

Algorithm 3 (Calculating the number of points below each line 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 3 is obvious from Figure 5: 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 $\Delta p_i p_j^i p_{j-1}^i$ 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 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. 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 of course be filled at the same moment.

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, O'Rourke, and Seidel [8], 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 4.1 *We can preprocess a point set P in the plane in $O(n^2)$ time and space, such that afterwards for each triangle in P, the number of points in it can be determined in constant time.*

5 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 $O(kn^2)$ 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 C be the minimum area convex k-gon. Let p_1 be the bottommost vertex of C and p_2 and p_3 the next two vertices in counterclockwise order. Now we can decompose C into the triangle $\Delta p_1 p_2 p_3$ and the remaining $(k-1)$-gon C'. Now obviously C' is minimal among all $(k-1)$-gons with p_1 as bottommost vertex, p_3 as next vertex and

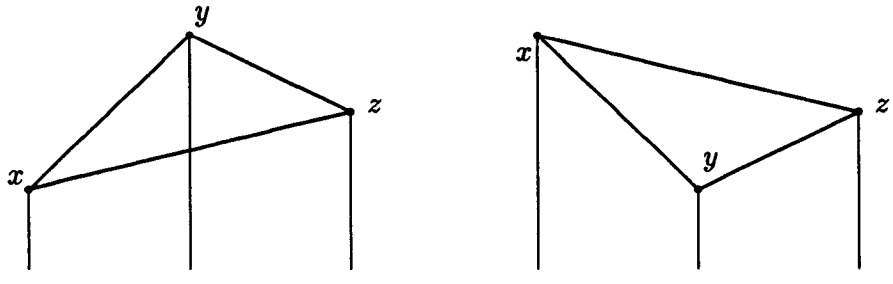

Figure 4: The two possibilities for the triangle Δxyz

all points on one side of the line $l(p_3, p_2)$. So we could compute 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 will 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 C such that (see Figure 6)

- point p_i is the bottommost vertex,
- point p_j is the next vertex in counterclockwise order, i. e., all points of C lie to the left of the line $l(p_i, p_j)$, and
- all points of 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 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. In contrast to the previous sections we do not sort the *halflines* 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 already filled all entries in AR with last index $\leq m - 1$. We will 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 ist left to vary). Obviously, this will lead 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 new neighbor of p_j. These two possible cases are shown in Figure 6.

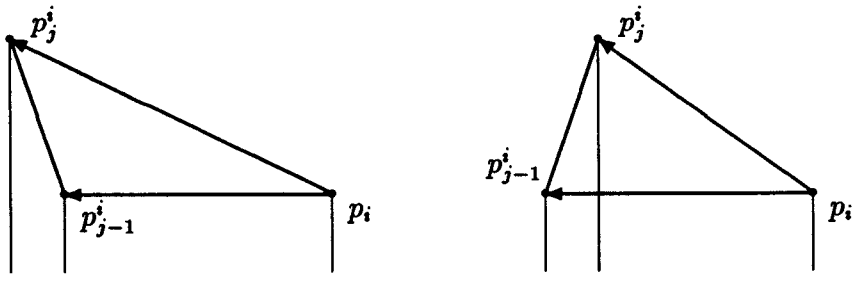

Figure 5: The cases (d1) and (d2) in Algorithm 3

- If p_l lies to the right of the line segment $\overline{p_i p_j}$, then nothing changes. No new point can be used and $AR[p_i, p_j, p_l, m]$ is equal to $AR[p_i, p_j, pred(p_l), m]$.
- If p_l lies to the left of the line segment $\overline{p_i p_j}$, then p_l might be a vertex of the minimum area polygon. In this case, the area is composed of the triangle $\Delta p_i p_j p_l$ 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 4 (Finding the smallest k-gon)

```
TotalMinimum := ∞;
for all points p_i do
  AR[p_i, *, *, 2] := 0;
  for m := 3 to k do
    Minarea := ∞;
    for all points p_j above p_i, in clockwise order around p_i, do
      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 Δp_i p_j p_l);
        endif;
        AR[p_i, p_j, p_l, m] := Minarea;
      endfor;
    endfor;
  endfor;
  TotalMinimum := min(TotalMinimum, minimum of AR[p_i, *, *, k]);
endfor.
```

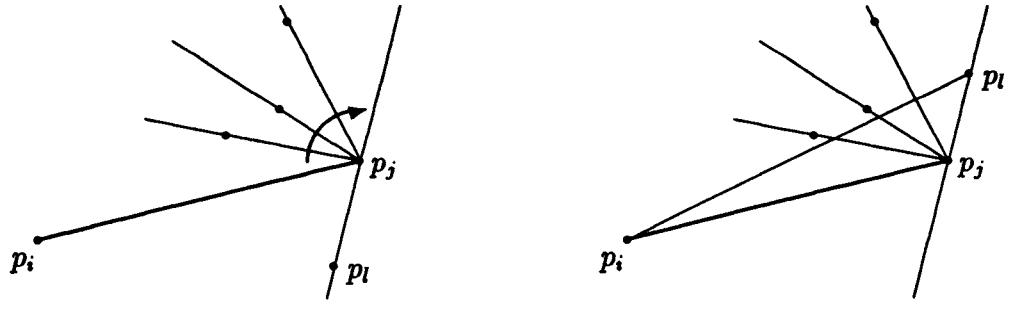

Figure 6: How to treat the point p_l

Theorem 5.1 *(1) The convex k-gon, (2) the empty convex k-gon, or (3) the convex hull of k points, with the smallest area can be found in $O(kn^3)$ time and $O(kn^2)$ space. If only the area of the smallest convex k-gon or of the smallest empty convex k-gon is required, we need only $O(n^2)$ 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. 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, at the expense of a time increase by a factor of $\log k$ (cf. [10]): In a first pass, we only compute the area of the optimal solution and the two "mid-points" $p_{(\lfloor k/2 \rfloor)}$ and $p_{(\lfloor k/2 \rfloor+1)}$ of the optimal polygon $\langle p_{(1)} = p_i, p_{(2)}, \ldots, p_{(k)} \rangle$. Then we recursively compute the left half $\langle p_{(1)}, p_{(2)}, \ldots, p_{(\lfloor k/2 \rfloor)}, p_{(\lfloor k/2 \rfloor+1)} \rangle$ and the right half $\langle p_{(1)}, p_{(\lfloor k/2 \rfloor)}, p_{(\lfloor k/2 \rfloor+1)}, \ldots, p_{(k)} \rangle$ of the optimal polygon.

(2) The only difference to case (1) is that we have to take care that the polygons we get are empty. Applying the results of Section 4, we preprocess the point set P in $O(n^2)$ time and space. Afterwards, only empty triangles are used to compose the minimum area polygons, i. e., the line $(*)$ in the algorithm is executed only when the triangle $\Delta p_i p_j p_l$ 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

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

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. □

6 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 C.

Definition 6.1 *A weight function W is called* decomposable *iff for any polygon $C = \langle p_1, \ldots, p_m \rangle$ and any index $2 < i < m$*

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

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

In other words, when W is decomposable we can cut the polygon C in two subpolygons along the line segment $\overline{p_1 p_i}$ and obtain the weight of 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 $\diamondsuit(x, y, p, q) = x + y$. Many different monotone decomposable weight functions exist. For example

- The perimeter. Here $\diamondsuit(x, y, p, q) = x + y - 2|\overline{pq}|$.
- Sum of the internal angles. $\diamondsuit(x, y, p, q) = x + y$. [1]
- Number of points of some set that lie in the interior. $\diamondsuit(x, y, p, q) = x + y$.
- 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. $\diamondsuit(x, y, p, q) = x + y - w(p) - w(q)$. Similar we can take the maximal or minimal weight of the points as weight of the polygon.

Not all weight functions are decomposable. For example the diameter is not decomposable. Also the smallest internal angle is not decomposable.

Theorem 6.2 *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 C is the optimal solution. Let $C = \langle p_1, \ldots, p_k \rangle$ with p_1 the bottommost vertex. Now split C in $C' = \langle p_1, p_3, \ldots, p_k \rangle$ and the triangle $\Delta p_1 p_2 p_3$. Because W is monotone 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 C. □

Note that the monotonicity of the weight function is essential for the method to be correct. The following result on the perimeter follows immediately.

[1] The sum of the angles can be minimized or maximized in an easier way. This will be pursued in a subsequent paper.

Corollary 6.3 *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 [1].

Corollary 6.4 *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 4.1. 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 $O(kn^2)$. With some slight modifications also weight functions like the length of the longest or shortest edge can be treated (although they are not decomposable) in the same bounds.

7 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 $O(kn^2)$ 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. 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.

Although our method can solve many types of minimization problems, as shown in section 6, some problems can not be solved with it. In particular, problems with a non-local weight criterion, like e.g. the minimum diameter, do not fit in the scheme. It is open whether dynamic programming can be used to solve those problems as well.

A final open problem concerns non-convex 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. At first glance a method similar to the one proposed in section 5 might seem to work but this is not true. The problem is that the polygon might become self-overlapping this way. Indeed, it is easy to find examples where the smallest area k-gon is self-overlapping. Avoiding these polygons seems very hard.

Acknowledgements. We would like to thank Helmut Alt and Emo Welzl for many helpful discussions and Herbert Edelsbrunner for a useful comment simplifying and improving the algorithm for $k = 4$.

References

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

[2] A. Aggarwal, H. Imai, N. Katoh, and S. Suri, Finding k points with minimum diameter and related problems, *Proc. 5th Annual Symp. on Computational Geometry*, Saarbrücken, 1989, to appear.

[3] D. Avis and D. Rappaport, Computing the largest empty convex subset of a set of points, *Proc. Symp. on Computational Geometry*, Baltimore, 1985, 161–167.

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

[5] D.P. Dobkin, R.L. Drysdale, and L.J. Guibas, Finding Smallest Polygons, In: *Advances in Computing Research, Vol. 1*, JAI Press, 1983, 181–214.

[6] D.P. Dobkin, H. Edelsbrunner, and M.H. Overmars, Searching for empty convex polygons, *Proc. 4th Annual Symp. on Computational Geometry*, Urbana-Champaign, 1988, 224–228.

[7] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, EATCS Monographs on Theor. Computer Science, Springer-Verlag, Berlin, 1987.

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

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

[10] J.I. Munro and R.J. Ramirez, Reducing space requirements for shortest path problems, *Operations Research* **30** (1982), 1009–1013.

[11] M.H. Overmars, B. Scholten, and I. Vincent, Sets without empty convex 6-gons, *Bull. of the EATCS* **37** (1989), to appear.

Finding minimum area k-gons

M.H. Overmars, G. Rote, G. Woeginger

Technical Report RUU-CS-89-7
March 1989

Utrecht University

Department of Computer Science

Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : ... + 31 - 30 - 531454

Finding minimum area k-gons

M.H. Overmars, G. Rote, G. Woeginger

RUU-CS-89-7
March 1989

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands

ISSN:0924-3275

Finding Minimum Area k-gons

Mark Overmars *
Günter Rote †‡
Gerhard Woeginger †

Abstract

Given a set P of n points in the plane and a number k, we want to find a polygon C with vertices in P of minumum area that satisfies one of the following properties: (1) C is a convex k-gon, (2) C is an empty convex k-gon, or (3) 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 the dynamic programming approach. We also generalize this approach to polygons with many other properties like e.g. minimum perimeter, maximum perimeter and area, containing the maximum or minimum number of points, of 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 of such problems deals with finding empty convex k-gons in a set of points. It is well-known ([9]) that such k-gons might not exist for $k \geq 7$. Algorithms to find such k-gons have been presented in [3, 6, 11]. The best known result works for 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 $O(n^3)$.

Boyce, Dobkin, Drysdale, and Guibas [4] 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, Klawe, Moran, Shor, and Wilber [1] improved these results to $O(kn + n \log n)$.

*Department of Computer Science, University of Utrecht, P.O.Box 80.089, 3508 TB Utrecht, the Netherlands
†Fachbereich Mathematik, Freie Universität Berlin, Arnimallee 2-6, D-1000 Berlin 33
‡on leave from the Institut für Mathematik, Technische Universität Graz, Austria

In application 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, Drysdale and Guibas [5]. Their $O(k^2 n \log n + k^5 n)$ algorithm was recently improved to $O(n \log n + k^4 n)$ by Aggarwal, Imai, Katoh, and Suri [2]. This recent paper also studies problems like finding minimum diameter k-gons and minimum variance k-gons.

In this paper we will 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 [7]. For $k > 3$ the best known result was $O(n^k)$. (In [7] the existence of an $o(n^4)$-algorithm for finding a minimum area convex 4-gon is stated as an open problem (Problem 12.10).) When $k > 3$ we have to define the problem more carefully. We can distinguish between the following three problems (throughout the paper, let P be a set of n points in the plane in general position):

(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 the following table.

	$k = 3$	$k = 4$	$k \geq 5$
	time / space	time / space	time / space
(1) convex k-gon	n^2 , n^2	n^3 , n	kn^3 , kn^2 (n^2)
(2) convex empty k-gon	n^2 , n^2	n^3 , n	kn^3 , kn^2 (n^2)
(3) convex hull of k points	n^2 , n^2	n^3 , n	kn^3 , kn^2

In the case $k \geq 5$, the space requirement is smaller if we only want to compute the *value* of the smallest area, not the polygon itself. This is indicated in parentheses in the last column.

The paper is organized as follows.

In section 3 we show how to solve the three problems in the case $k = 4$ in the stated time and space bounds. The results are based on the concept of a *type-0 chain* that is introduced in section 2.

Section 4 shows how to calculate for all triangles determined by P the number of points in their interior. This will be used in section 5 to obtain our efficient algorithm for $k > 4$.

The method is based on the dynamic programming approach. In section 6 this 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 [2] for large k. Another application finds the convex k-gon containing the smallest or largest number of points.

Finally, in section 7 we give some concluding remarks and directions for further research.

The following notations will be 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. $\Delta p_1 p_2 p_3$ 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 counter-clockwise order. $\angle p_1 p_2 p_3$ denotes the angle with apex p_2.

2 Constructing Type-0 Chains

We start this section by giving a key definition of the paper.

Definition 2.1 *Let $\overline{xy}$ be a line segment. Let P be a set of points lying to the left of $l(x, y)$. We say, a point $p \in P$ is of* type i *with respect to $\overline{xy}$ and P, iff the triangle Δxyp contains exactly i points of P in its interior.*

In this section we will show how the set of all type-0 points for a given line segment $\overline{xy}$ can be computed efficiently. We start with deriving a number of properties for type-i point sets. First, let us fix the line segment $\overline{xy}$ and the point set P. Thus if we speak of a type-i point in the following, we always mean a type-i point with respect to this $\overline{xy}$ and P. W.l.o.g. we assume that $\overline{xy}$ is horizontal and P consists of points lying above $\overline{xy}$.

Lemma 2.2 *(i) If p_1 is a type-i point, then the triangle Δxyp_1 does not contain any type-j point p_2 in its interior, with $j \geq i$.*

(ii) For two type-i points p_1, p_2 in P, $\angle p_1 xy > \angle p_2 xy$ holds if and only if $\angle p_1 yx < \angle p_2 yx$.

(iii) If we sort the set of type-i points by decreasing angle $\angle pxy$, we get the same sequence as if we sort them by increasing angle $\angle pyx$.

Proof:

(i) If the type-j point p_2 lies in the interior of Δxyp_1, the whole triangle Δxyp_2 is a subset of Δxyp_1. Hence, the j points in Δxyp_2 together with the point p_2 lie in Δxyp_1. This implies that $i \geq j+1$ must hold.

(ii) Assume, $\angle p_1xy > \angle p_2xy$ and $\angle p_1yx > \angle p_2yx$ holds. But then, obviously, p_2 must lie in the interior of Δxyp_1 and this is a contradiction to (i).

(iii) A straightforward consequence of (ii). □

Because of (iii), the following definition makes sense.

Definition 2.3 *Let $\langle c_1^i, c_2^i, \ldots, c_m^i \rangle$ be the sequence of type-i points, sorted by decreasing $\angle pyx$ or by increasing $\angle pxy$. We connect the points c_j^i and c_{j+1}^i by straight line segments. The polygonal curve we obtain is called the* type-i chain C^i *of P with respect to $\overline{xy}$.*

We now give a sequential algorithm that constructs the type-0 chain for a point set P of n points and a segment $\overline{xy}$ in $O(n \log n)$ time. The points are sorted around x and are then visited sequentially. The type-0 chain C^0 under construction is stored as a linear list. The variable *low* always contains the point p with smallest angle $\angle xyp$ among the points considered so far.

Algorithm 1 (Construction of type-0 chains)

(a) Sort the points in P around x by increasing angle $\angle pxy$. This gives the sequence $p_1, p_2, \ldots, p_n$.

(b) Initialization: $C^0 := \langle p_1 \rangle$, $low := p_1$.

(c) For $p := p_2$ to p_n do

If *low* is not contained in Δxyp then add p to the end of C^0.

If $\angle xyp < \angle(x, y, low)$ then $low := p$.

Step (a) takes $O(n \log n)$ time, Step (b) takes constant time. Step (c) consists of n substeps and each substep can be performed in constant time. Hence, the algorithm has an overall running time of $O(n \log n)$.

It remains to show that the algorithm works correctly and indeed finds the type-0 chain. Obviously, the point p_1 is a type-0 point: For any point q in the interior of Δxyp_1, the angle $\angle qxy$ has to be smaller than $\angle p_1xy$. As p_1 has the smallest angle there is no such point and, hence, Δxyp_1 is empty.

Assume the algorithm has reached some point p and *low* is maintained as indicated. Consider the point p and its relation to the point *low*. The two possible cases are shown in Figure 1. If *low* lies in the interior of Δxyp, then p is clearly at least of type 1. If *low* does not lie in the interior of Δxyp, then p is of type 0: Any point q in the interior of Δxyp must have a smaller angle $\angle qxy$ and a smaller $\angle qyx$. But *low* is the point with smallest angle $\angle xyq$ among the points q with $\angle qxy < \angle pxy$.

Theorem 2.4 *Let $\overline{xy}$ be a line segment, let P be a set of points all lying to the left of $l(x,y)$. Then the type-0 chain can be constructed in $O(n \log n)$ time. If the points in P are given in sorted order around x, then the type-0 chain can be constructed in $O(n)$ time.*

3 Finding a Minimum-Area Four-Point-Set

We will 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 of 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: Assume, the smallest area triangle Δabc containing at least one point would contain at least two points d, e. Then the point e would lie in one of the smaller triangles Δabd, Δacd or Δbcd. □

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

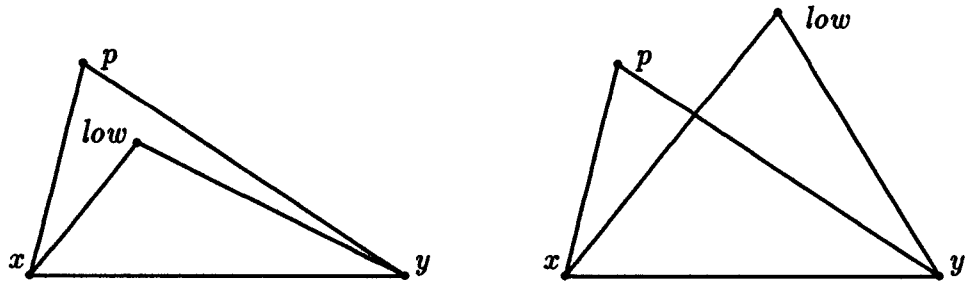

Figure 1: The two possibilities for the point p_{k+1} in Algorithm 1

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 left of $l(x, y)$.

(B1) Find the smallest area triangle Δxyz with $z \in P'$ and at least one other point u of P' in its interior.

(B2) Find the smallest area quadrangle $\square xyzu$ with $z, u \in P'$ and no other point of P' in its interior.

(C) Select the smallest area configuration of all the triangles Δxyz and all the quadrangles $\square xyzu$ found in step (B).

Step (A) takes time $O(n^2 \log n)$. In the rest of this section, we will 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 the correctness note that there will always be two points x, y in the correct answer with the other points of the answer to the left of $l(x, y)$.

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 Δxyz contains at least one point of P' and such that z minimizes the area of Δxyz under this condition. As the length of the triangle's baseline $\overline{xy}$ is fixed, its area becomes minimal iff its height becomes minimal. The triangle's height is the distance of the point z from $l(x, y)$. Therefore, our algorithm is as follows:

Algorithm 2.1 (Solution of problem B1)
Construct the type-0 chain C^0 for P'.
Find the point z in $P' - C^0$ with the smallest distance from $l(x, y)$.
Output the area of Δxyz.

Problem (B2) is more complicated. Of course, we can use the type-0 chain again. As $\square xyzu$ must be empty, the only potential candidates for z and u are the points on the type-0 chain $C^0 = \langle c_1, c_2, \ldots, c_m \rangle$. But now there are $O(n^2)$ possibilities to choose z and u, whereas we want to do it in $O(n)$ time.

Assume, the point $z = c_i$ has already been chosen and we want to find the point u that minimizes the area of $\square xyzu$. Let $C'(z) = \langle c_1, c_2, \ldots, c_{i-1} \rangle$ denote the points on C^0 that lie before z. The area of $\square xyzu$ is composed of the areas of Δxyz and Δxzu (see Figure 2). As z is fixed, the area of Δxyz and the length of $\overline{xz}$ are fixed, too. Consequently, the area of $\square xyzu$ is minimized iff the distance of u to the line $l(x, z)$ is minimized. Hence, our problem is:

For each point z on $C^0 = \langle c_1, c_2, \ldots, c_m \rangle$, find the point u on $C'(z)$ that minimizes the distance of u to $l(x, z)$.

It is easy to see that only the points on the convex hull of $C'(z)$ are candidates for the point u. This suggests the following approach: We treat the points in C^0 in rotational order and maintain the convex hull of the points passed. Determining the point on the convex hull nearest to $l(x, z)$ is easy. Unfortunately this would require $O(n \log n)$ time. To avoid this we restrict our set of possible candidates even further.

First, we can throw away the upper chain of the convex hull, (the convex hull is divided into the *upper chain* and the *lower chain* by the first point c_1 and the last point c_{i-1}) as there is always some point on the lower chain that is closer to the sweepline than a point on the upper chain. Second, consider the following situation: While sweeping over the chain C^0, the angle $\angle zxy$ decreases from π down to 0. Now at some fixed moment, the sweepline becomes parallel to some edge $\overline{c_i c_j}$ of the hull. In the next moment, the slope of the line decreases and from now on, the left endpoint c_i will always be closer to the sweep line than the right endpoint c_j. Hence, we can throw away the right point c_j ("left" and "right" always refer to the positions on C^0).

For an example, see Figure 2: The convex hull of $C'(z)$ consists of the points $\langle c_1, c_5, c_6, c_3, c_2 \rangle$, the point c_4 lies in the interior. $\langle c_2, c_3 \rangle$ constitutes the upper chain of the hull and, hence, can be removed. At some moment, the sweepline has been parallel to $\overline{c_5 c_6}$. Consequently, we can throw away the point c_6 and the remaining points $\langle c_1, c_5 \rangle$ form the pruned sequence $C''(z)$.

$C''(z)$ has some nice properties: It consists of a convex sequence of line segments. It starts at c_1. The slope of the segments increases from left to right, but does not exceed the slope of the line $l(x, z)$. It is easy to see that the rigthmost point of $C''(z)$ is the closest point to $l(x, z)$.

This leads to the following algorithm: $C''(z)$ is stored in a stack $\mathcal{S}$. The variable *last* always contains the last element of the stack $\mathcal{S}$, variable *lbo* contains the last but one element of $\mathcal{S}$. We will use the standard operations PUSH and POP. Variable

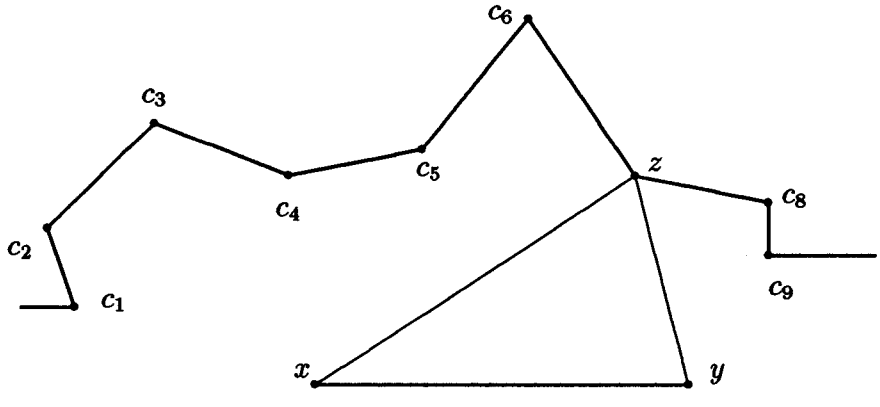

Figure 2: The segment $\overline{xy}$, its type-0 chain and the triangle Δxyz

Minarea stores the smallest area of configurations checked until now:

Algorithm 2.2 (Solution of problem B2)

(1) Initialization: PUSH($\mathcal{S}, c_1$) and set *Minarea* := ∞.

(2) For $z := c_2$ to c_m do

(2.1) while [*lbo* is closer to $l(x,z)$ than *last*] do POP($\mathcal{S}$);
(2.2) *Minarea* := min(*Minarea*, area of $\square(x,y,z,last)$;
(2.3) while [*last* lies to the left of $l(lbo,z)$] do POP($\mathcal{S}$);
(2.4) PUSH($\mathcal{S}, z$);

endfor.

By the preceeding observations the correctness of Algorithm 2.2 is easily established: The while-loop in (2.1) throws away points of segments that were parallel to the sweep line since the previous point z treated; the while-loop (2.3) removes points that do not lie on the lower chain of the convex hull. Also the time complexity is easy to determine. Every point in C^0 is pushed, respectively popped, at most once. Hence, the algorithm runs in linear time, provided the type-0 chain is available (which it is because we computed C^0 in Algorithm 2.1).

The last thing that remains to show is that the quadrangle $\square xyzu$ found by Algorithm 2.2 is indeed empty. Remember: Type-0 points z and u only guarantee that the triangles Δxyz and Δxyu are empty. But $\square xyzu$ does not consist of only these two triangles (see Figure 3).

Fortunately, we can show that the remaining part is empty, too. Assume, it were not. Consider the point q in the part that is closest to $l(x,y)$. Obviously, Δxyq is empty and thus q is of type 0. q is contained in Δxzu and is therefore closer to $l(x,z)$ than u. But this contradicts with the fact that u is the type-0 point closest to $l(x,z)$.

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.*

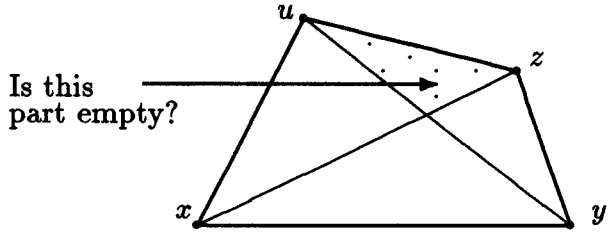

Figure 3: The quadrangle $\square xyzu$ that has been found by Algorithm 2.2

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 mix the constructions of the type-0 chains and the area-calculations. First, we treat all segments $\overline{p_1 p_i}$ incident to p_1. Observe, that the space used in this step is linear, as we only have to store the sequence of points sorted around p_1. Then we treat all segments incident to p_2, p_3 and so on. This gives the claimed space complexity. □

To find the smallest area convex 4-gon, we simply skip Step (B1). It is easy to see that 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 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, afterwards, the number of points inside any triangle in P can be determined in constant time. This result will be used in Section 5. The strucure derived by the preprocessing step is a two-dimensional 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 $\overline{p_i p_j}$. Obviously, for a triangle Δxyz 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 Figure 4).

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 Figure 5):

Algorithm 3 (Calculating the number of points below each line 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 3 is obvious from Figure 5: 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 $\Delta p_i p_j^i p_{j-1}^i$ 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 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. 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 of course be filled at the same moment.

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, O'Rourke, and Seidel [8], 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 4.1 *We can preprocess a point set P in the plane in $O(n^2)$ time and space, such that afterwards for each triangle in P, the number of points in it can be determined in constant time.*

5 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 $O(kn^2)$ 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 C be the minimum area convex k-gon. Let p_1 be the bottommost vertex of C and p_2 and p_3 the next two vertices in counterclockwise order. Now we can decompose C into the triangle $\Delta p_1 p_2 p_3$ and the remaining $(k-1)$-gon C'. Now obviously C' is minimal among all $(k-1)$-gons with p_1 as bottommost vertex, p_3 as next vertex and

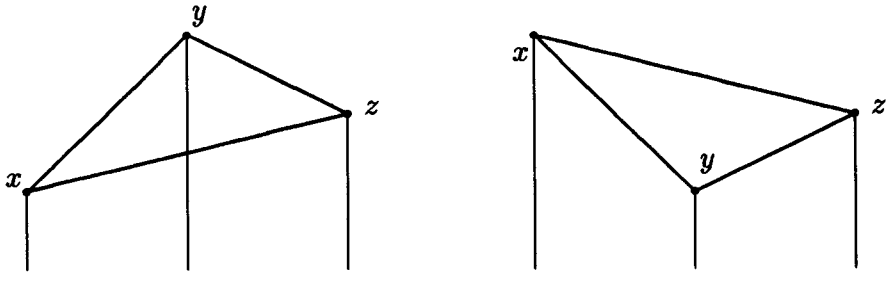

Figure 4: The two possibilities for the triangle Δxyz

all points on one side of the line $l(p_3, p_2)$. So we could compute 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 will 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 C such that (see Figure 6)

- point p_i is the bottommost vertex,
- point p_j is the next vertex in counterclockwise order, i. e., all points of C lie to the left of the line $l(p_i, p_j)$, and
- all points of 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 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. In contrast to the previous sections we do not sort the *halflines* 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 already filled all entries in AR with last index $\leq m - 1$. We will 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 ist left to vary). Obviously, this will lead 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 new neighbor of p_j. These two possible cases are shown in Figure 6.

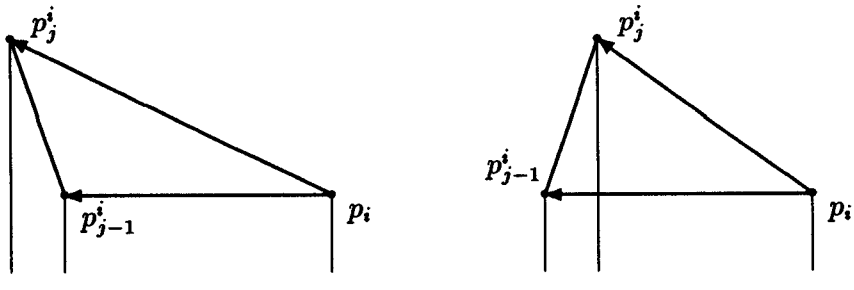

Figure 5: The cases (d1) and (d2) in Algorithm 3

- If p_l lies to the right of the line segment $\overline{p_i p_j}$, then nothing changes. No new point can be used and $AR[p_i, p_j, p_l, m]$ is equal to $AR[p_i, p_j, pred(p_l), m]$.
- If p_l lies to the left of the line segment $\overline{p_i p_j}$, then p_l might be a vertex of the minimum area polygon. In this case, the area is composed of the triangle $\Delta p_i p_j p_l$ 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 4 (Finding the smallest k-gon)

```
TotalMinimum := ∞;
for all points p_i do
  AR[p_i, *, *, 2] := 0;
  for m := 3 to k do
    Minarea := ∞;
    for all points p_j above p_i, in clockwise order around p_i, do
      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 Δp_i p_j p_l);
        endif;
        AR[p_i, p_j, p_l, m] := Minarea;
      endfor;
    endfor;
  endfor;
  TotalMinimum := min(TotalMinimum, minimum of AR[p_i, *, *, k]);
endfor.
```

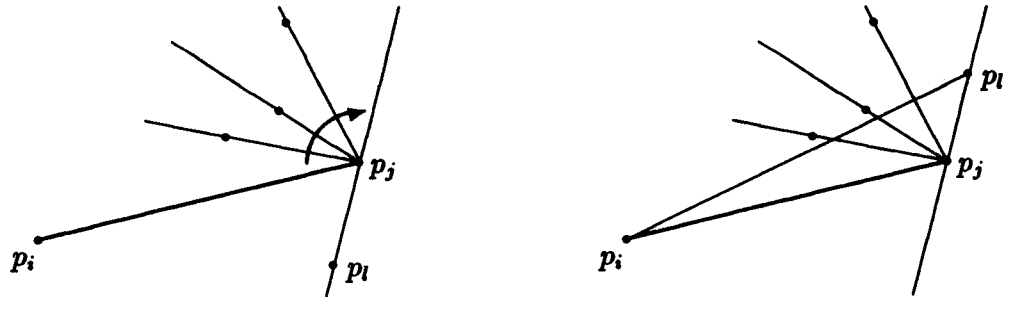

Figure 6: How to treat the point p_l

Theorem 5.1 *(1) The convex k-gon, (2) the empty convex k-gon, or (3) the convex hull of k points, with the smallest area can be found in $O(kn^3)$ time and $O(kn^2)$ space. If only the area of the smallest convex k-gon or of the smallest empty convex k-gon is required, we need only $O(n^2)$ 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. 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, at the expense of a time increase by a factor of $\log k$ (cf. [10]): In a first pass, we only compute the area of the optimal solution and the two "mid-points" $p_{(\lfloor k/2 \rfloor)}$ and $p_{(\lfloor k/2 \rfloor+1)}$ of the optimal polygon $\langle p_{(1)} = p_i, p_{(2)}, \ldots, p_{(k)} \rangle$. Then we recursively compute the left half $\langle p_{(1)}, p_{(2)}, \ldots, p_{(\lfloor k/2 \rfloor)}, p_{(\lfloor k/2 \rfloor+1)} \rangle$ and the right half $\langle p_{(1)}, p_{(\lfloor k/2 \rfloor)}, p_{(\lfloor k/2 \rfloor+1)}, \ldots, p_{(k)} \rangle$ of the optimal polygon.

(2) The only difference to case (1) is that we have to take care that the polygons we get are empty. Applying the results of Section 4, we preprocess the point set P in $O(n^2)$ time and space. Afterwards, only empty triangles are used to compose the minimum area polygons, i. e., the line $(*)$ in the algorithm is executed only when the triangle $\Delta p_i p_j p_l$ 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

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

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. □

6 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 C.

Definition 6.1 *A weight function W is called* decomposable *iff for any polygon $C = \langle p_1, \ldots, p_m \rangle$ and any index $2 < i < m$*

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

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

In other words, when W is decomposable we can cut the polygon C in two subpolygons along the line segment $\overline{p_1 p_i}$ and obtain the weight of 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 $\diamondsuit(x, y, p, q) = x + y$. Many different monotone decomposable weight functions exist. For example

- The perimeter. Here $\diamondsuit(x, y, p, q) = x + y - 2|\overline{pq}|$.
- Sum of the internal angles. $\diamondsuit(x, y, p, q) = x + y$. [1]
- Number of points of some set that lie in the interior. $\diamondsuit(x, y, p, q) = x + y$.
- 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. $\diamondsuit(x, y, p, q) = x + y - w(p) - w(q)$. Similar we can take the maximal or minimal weight of the points as weight of the polygon.

Not all weight functions are decomposable. For example the diameter is not decomposable. Also the smallest internal angle is not decomposable.

Theorem 6.2 *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 C is the optimal solution. Let $C = \langle p_1, \ldots, p_k \rangle$ with p_1 the bottommost vertex. Now split C in $C' = \langle p_1, p_3, \ldots, p_k \rangle$ and the triangle $\Delta p_1 p_2 p_3$. Because W is monotone 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 C. □

Note that the monotonicity of the weight function is essential for the method to be correct. The following result on the perimeter follows immediately.

[1] The sum of the angles can be minimized or maximized in an easier way. This will be pursued in a subsequent paper.

Corollary 6.3 *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 [1].

Corollary 6.4 *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 4.1. 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 $O(kn^2)$. With some slight modifications also weight functions like the length of the longest or shortest edge can be treated (although they are not decomposable) in the same bounds.

7 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 $O(kn^2)$ 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. 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.

Although our method can solve many types of minimization problems, as shown in section 6, some problems can not be solved with it. In particular, problems with a non-local weight criterion, like e.g. the minimum diameter, do not fit in the scheme. It is open whether dynamic programming can be used to solve those problems as well.

A final open problem concerns non-convex 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. At first glance a method similar to the one proposed in section 5 might seem to work but this is not true. The problem is that the polygon might become self-overlapping this way. Indeed, it is easy to find examples where the smallest area k-gon is self-overlapping. Avoiding these polygons seems very hard.

Acknowledgements. We would like to thank Helmut Alt and Emo Welzl for many helpful discussions and Herbert Edelsbrunner for a useful comment simplifying and improving the algorithm for $k = 4$.

References

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

[2] A. Aggarwal, H. Imai, N. Katoh, and S. Suri, Finding k points with minimum diameter and related problems, *Proc. 5th Annual Symp. on Computational Geometry*, Saarbrücken, 1989, to appear.

[3] D. Avis and D. Rappaport, Computing the largest empty convex subset of a set of points, *Proc. Symp. on Computational Geometry*, Baltimore, 1985, 161–167.

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

[5] D.P. Dobkin, R.L. Drysdale, and L.J. Guibas, Finding Smallest Polygons, In: *Advances in Computing Research, Vol. 1*, JAI Press, 1983, 181–214.

[6] D.P. Dobkin, H. Edelsbrunner, and M.H. Overmars, Searching for empty convex polygons, *Proc. 4th Annual Symp. on Computational Geometry*, Urbana-Champaign, 1988, 224–228.

[7] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, EATCS Monographs on Theor. Computer Science, Springer-Verlag, Berlin, 1987.

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

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

[10] J.I. Munro and R.J. Ramirez, Reducing space requirements for shortest path problems, *Operations Research* **30** (1982), 1009–1013.

[11] M.H. Overmars, B. Scholten, and I. Vincent, Sets without empty convex 6-gons, *Bull. of the EATCS* **37** (1989), to appear.