ELSEVIER

Computational Geometry 7 (1997) 187–200

Computational Geometry
Theory and Applications

Sequential and parallel algorithms for finding a maximum convex polygon ☆

Paul Fischer [1],

Lehrstuhl Informatik II, Universität Dortmund, D-44221 Dortmund, Germany

Communicated by B. Chazelle; submitted 19 August 1994; accepted 18 September 1995

Abstract

This paper investigates the problem where one is given a finite set of n points in the plane each of which is labeled either "positive" or "negative". We consider bounded convex polygons, the vertices of which are positive points and which do not contain any negative point. It is shown how such a polygon which is maximal with respect to area can be found in time $O(n^3 \log n)$. With the same running time one can also find such a polygon which contains a maximum number of positive points. If, in addition, the number of vertices of the polygon is restricted to be at most M, then the running time becomes $O(M n^3 \log n)$. It is also shown how to find a maximum convex polygon which contains a given point in time $O(n^3 \log n)$. Two parallel algorithms for the basic problem are also presented. The first one runs in time $O(n \log n)$ using $O(n^2)$ processors, the second one has polylogarithmic time but needs $O(n^7)$ processors. Instead of using the area or the number of positive points contained in the polygon as the quantity to be maximized one may also use other measures fulfilling a certain additive property, however, this may affect the running time.

Keywords: Computational geometry; Convex polygons

1. Introduction

There are various algorithms for determining the convex hull of a set of points in the Euclidean plane, see, e.g., [12]. Here we deal with the more complicated situation where we have two types of points, positive and negative ones and the objective is to find a maximum (with respect to area, number of positive points contained or other measures) convex polygon the vertices of which are positive points that does not contain any negative point. A related problem of finding a *minimum* area polygon in a set of unlabeled points has been considered in [5], the problem of finding maximum *empty rectangles* in [1] and [2]. In [4] an $O(n^3)$ algorithm for the following problem is presented:

☆ Supported by Deutsche Forschungsgemeinschaft grant We 1066/6-2.

[1] E-mail: paulf@goedel.informatik.uni-dortmund.de.

SSDI 0925-7721(95)00035-6

given a set S of n points in the plane, find a convex polygon the vertices of which are from S, and which has a maximum number of vertices.

The investigation of our problem is motivated by applications in statistical clustering, pattern recognition, data compression and PAC-learning. The positive points may, e.g., be interpreted as the pixels of some objects while the negative ones do not belong to the objects. The objective then is to (approximately) recover the shapes of the objects from the pixels, which can be done by successively removing large convex areas containing positive points only.

For the learning application the result of this paper can be employed to give a more efficient learning algorithm for the class of unions of convex polygons than the one based on triangulations, for details see [8]. A modification of this algorithm has been used to efficiently solve the *minimum disagreement problem* for convex polygons (see [9] and [10]). This means that one is looking for a convex polygon P which minimizes the number of positive points not in P plus the number of negative points in P. It is known, that efficiently minimizing disagreements is sufficient for so called *agnostic learning*. In this learning model one is no longer looking for a "perfect" explanation of the observed data, but settles for a "good" explanation of a specific syntactic form.

We present an algorithm which solves the basic problem (finding a maximum area polygon the vertices of which are positive points and that does not contain any negative point) in $O(n^3 \log n)$ time where n is the number of points given. We also show how to find in time $O(n^3 \log n)$ a maximum convex polygon which contains a given point. These time bounds hold in the cases where the quantity to be maximized is the Euclidean area or the number of positive points contained. We then discuss the influence of other measures on the running time. It is also shown how to modify the algorithm in case one is looking for convex polygons with a bounded number of vertices, say at most M. The running time then becomes $O(M n^3 \log n)$. Finally we present a parallel algorithm which runs in time $O(n \log n)$ using $O(n^2)$ processors and a polylogarithmic algorithm which uses $O(n^7)$ processors. The latter is based on a formulation of the problem as a longest path problem in a directed acyclic graph (DAG), see Section 4 for details. Using this DAG approach for the sequential solution would lead to a running time of $O(n^4)$.

2. Definitions

Let POS and NEG be disjoint finite sets of points in $\mathbb{R}^2$. We call their elements *positive* and *negative points*, respectively. Let $n_1 := |\text{POS}|$, $n_2 := |\text{NEG}|$, and $n = n_1 + n_2$. A PN-*convex polygon* is a bounded convex polygon the vertices of which are positive points and which does not contain a negative point in its interior or on its boundary. Because of boundedness the area of a PN-convex polygon is finite. Let f be a function which assigns a nonnegative real number to each bounded convex polygon and satisfies the following *additive property*: if A and B are disjoint convex polygons, then $f(A \cup B) = f(A) + f(B)$. The aim is to find a PN-convex polygon P such that $f(P)$ is maximum. The running time of the algorithm depends on the time needed to evaluate f on triangles. If f is the Euclidean area then this time is constant. In the following we shall always refer to f as the *area* and assume that f can be evaluated in constant time for triangles. In Section 4 we discuss other choices for f. Then, our objective can be stated as Problem A.

Problem A. *Given* POS *and* NEG, *find a* PN-*convex polygon of maximum area.*

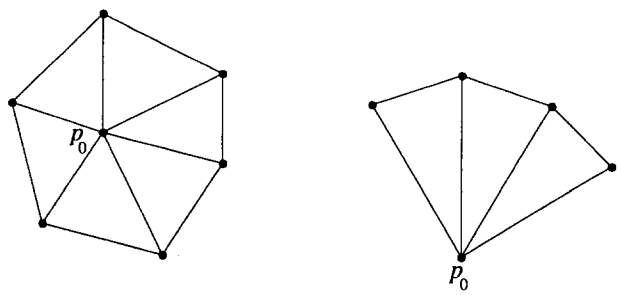

Fig. 1. Two triangulations of a convex polygon containing p_0.

In the following we present an $O(n^3 \log n)$ algorithm for solving this problem by a dynamic programming approach. In terms of the sizes of the sets of negative and positive points the running time is $O(n_1^2(n_1 \log n_1 + n_2 \log n_2))$. In the following we shall always state the bound in terms of both n and n_1, n_2.

In order to solve Problem A we shall solve a number of more restricted problems. We proceed by giving some definitions that are necessary to formulate this and that are also used in the description of the algorithm.

Given points $q_0, q_1, q_2 \in \mathbb{R}^2$ let $\triangle(q_0, q_1, q_2)$ denote the triangle formed by these points. If two or three points coincide or are collinear, then the triangle is only a line segment or a point and has area 0. The points q_0, q_1, q_2 form a *(strict) right turn* if q_2 is (strictly) to the right of the line through q_0 and q_1, where the orientation is from q_0 to q_1. A *(strict) left turn* is defined analogously. By the *angle formed by* q_0, q_1, q_2 we mean the angle between the two lines given by q_0, q_1 and q_1, q_2. Given q_0, q_1, q_2 we say that $\triangle(q_0, q_1, q_2)$ is *good* if it does not contain a negative point (neither in its interior nor on its boundary).

From now on we fix $p_0 \in$ POS. Note that any PN-convex polygon containing p_0 can be divided into triangles in such a way that each triangle has p_0 as a vertex and two different triangles have at most one edge in common. See Fig. 1 for an illustration. Let the points of POS $- \{p_0\}$ be ordered counterclockwise according to their polar angle with respect to p_0. If two points are at the same angle they are ordered arbitrarily. If necessary we renumber the points in such a way that their indices reflect this ordering. If not indicated otherwise, all orderings are counterclockwise and the positive direction of the x-axis has angle 0.

Given a triangle $\triangle(p_0, p_j, p_k)$, then $\triangle(p_0, p_i, p_j)$ is called a *(counterclockwise) continuation of* $\triangle(p_0, p_j, p_k)$ *with respect to* p_0 if p_0, p_j, p_k form a right turn and p_0, p_j, p_i form a left turn. If, moreover, p_i, p_j, p_k form a right turn we speak of a *convex continuation*. See Fig. 2 for an example. Given a positive point p_1, a *fan around* p_0 *originating* at p_1 is a sequence $T_0, T_1, \ldots, T_k$ of good triangles such that each has p_0 as a vertex, p_1 is that vertex of T_0 which is not a vertex of T_1, T_i is a *convex* continuation of T_{i-1} with respect to p_0, $i = 1, \ldots, k$, and no two different triangles in the sequence have more than an edge in common. The latter condition prevents the fan from wrapping around p_0 more than a full turn. Note that though a fan consists of successive convex continuations it may not be convex itself, see Fig. 3. Let $F = T_0, T_1, \ldots, T_k$ be a fan. We say that F *ends* at p_l if p_l is that vertex of T_k which is not a vertex of T_{k-1}. A triangle T is a *convex continuation* of fan

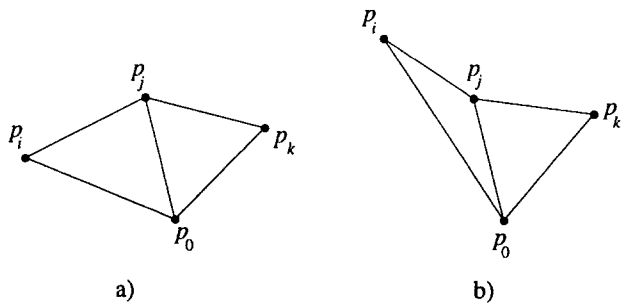

Fig. 2. (a) A convex continuation and (b) a nonconvex continuation of $\triangle(p_0, p_j, p_k)$.

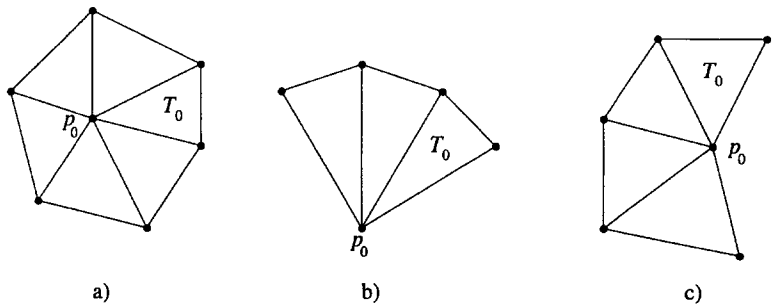

Fig. 3. Examples of fans around p_0. The fan in c) does not form a convex polygon.

F if it is a convex continuation of T_k and its intersection with T_0 is the point p_0 or an edge. In the following we shall in general drop the phrases "with respect to p_0" in connection with continuations and "around p_0 originating at p_1" in connection with fans.

For a convex polygon P we say that a vertex is a *bottom vertex* if it has minimal y-coordinate among all vertices. Using the standard definition of a convex polygon there are at most two bottom vertices. However, in our application we also deal with polygons that are given by a redundant presentation, allowing additional vertices on the edges.

We are now in a position to formulate the restricted problem.

Problem B. *Given* POS, NEG *and* $p_0 \in$ POS, *find a* PN*-convex polygon of maximum area having* p_0 *as bottom vertex.*

Below we shall show how Problem B can be solved in time $O(n^2 \log n)$ $(O(n_1^2 \log n_1))$ given some data which can be computed in a preprocessing step. This preprocessing step, as presented here, supplies the data for all positive points p in time $O(n^3 \log n)$ $(O(n_1^2 n_2 \log n_2))$. However, it can be modified in such a way to produce the necessary data for fixed $p_0 \in$ POS in time $O(n^2 \log n)$ $(O(n_1 n_2 \log n_2))$, whence Problem B can be solved in time $O(n^2 \log n)$ $(O(n_1(n_2 \log n_2 + n_1 \log n_1)))$.

Note that Problem B can be reformulated in terms of fans as follows: given $p_0 \in$ POS, find the maximum fan around p_0 that begins at some positive point p_j and ends at positive point p_i, such that the polar angles ϕ_j and ϕ_i of p_j and p_i with respect to p_0 satisfy $0 \leqslant \phi_j \leqslant \phi_i \leqslant \pi$. The fan simply is a triangulation of a maximum polygon where all triangles have vertex p_0 in common, see also Fig. 4.

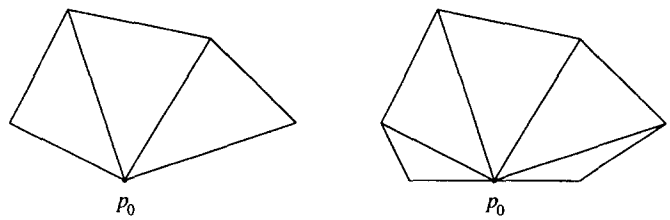

Fig. 4. Two examples of bottom vertices and the corresponding triangulations.

We shall also use algorithms and data structures for the following two problems. Let $x_1, \ldots, x_n$ be real numbers. Given i, $1 \leqslant i \leqslant n$, we want to find the *prefix minimum* of i, $\min\{x_k \mid 1 \leqslant k \leqslant i\}$, and also an index $M[i]$ where this minimum is assumed. Given a pair (i, j), $1 \leqslant i \leqslant j \leqslant n$, we want to find the *range minimum* of the interval $[i, j]$, $\min\{x_k \mid i \leqslant k \leqslant j\}$, and again an index $M[i, j]$ where this minimum is assumed. As both problems will have to be solved multiply on the same data, preprocessing helps to speed up the queries. In fact, the query time of the prefix minimum problem is constant using an $O(n)$ preprocessing. Queries for the range minimum problem can be answered in logarithmic time using a data structure similar to a segment tree, as defined by Bentley, see, e.g., [12]. The preprocessing time is $O(n)$. Clearly, the prefix *maximum* and range *maximum* problem can be solved analogously.

In order to efficiently handle the case where the area function f is the number of positive points contained, we need another data structure. One is given n pairs (x_k, y_k) of real numbers. We want to answer questions of the following type. Given real numbers a_1, b_1, a_2, b_2, what is the number of pairs (x_k, y_k) in the rectangle $[a_1, b_1] \times [a_2, b_2]$, i.e., $a_1 \leqslant x_k \leqslant b_1$ and $a_2 \leqslant y_k \leqslant b_2$. This problem is called the *rectangle counting problem* and it can be solved in time $O(\log n)$ given an $O(n \log n)$ preprocessing, see [3].

3. The algorithm

3.1. Preprocessing

The algorithm does some preprocessing first. Fix $p \in \mathrm{POS}$. We compute the polar angles of the points in $\mathrm{POS} - \{p\}$ with respect to p. Then the counterclockwise ordering of $\mathrm{POS} - \{p\}$ is computed and stored. Doing this for all $p \in \mathrm{POS}$ requires $O(n^2 \log n)$ time $(O(n_1(n_1 \log n_1)))$.

Now fix $p, p' \in \mathrm{POS}$. Let ϕ_k (ϕ'_k) denote the polar angle of negative point s_k with respect to p (p'). Form the pairs (ϕ_k, ϕ'_k), $k = 0, \ldots, n_2 - 1$, and order them according to first component. Then build a range minimum data structure on the second component. For each pair (p, p') it requires $O(n_2)$ time to compute the pairs (ϕ_k, ϕ'_k). Ordering the pairs requires $O(n_2 \log n_2)$. Then the range minimum data structure can be build in time $O(n_2)$. Hence the time for all pairs (p, p') of positive points is $O(n^3 \log n)$ $(O(n_1^2(n_2 \log n_2)))$.

Next, we want to compute a list of all good triangles. Fix $p, p' \in \mathrm{POS}$. There are $O(n_1)$ triangles having p and p' as vertices. Consider the triangle $\triangle(p, p', p'')$, where the points are located as in Fig. 5. After renumbering, let ϕ_k, $k = 0, \ldots, n_2 - 1$, be the ordering of polar angles of the negative points with respect to p. Let λ and ρ be the polar angles of p' and p'' with respect to p, and assume

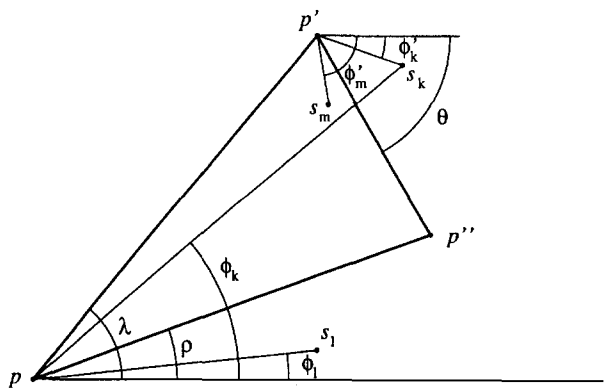

Fig. 5. Testing whether triangle $\triangle(p,p',p'')$ is good. Negative point s_l is not considered because the corresponding angle ϕ_l is not between ρ and λ. s_k is not in the triangle because ϕ'_k is greater than θ, while s_m is inside as $\phi'_m < \theta$.

wlog that $0 \leqslant \rho \leqslant \lambda < 2\pi$. Find the position of ρ and λ in the ϕ_k, i.e., let $l = \min\{v \mid \rho \leqslant \phi_v\}$ and $r = \max\{v \mid \lambda \geqslant \phi_v\}$. Then the negative points corresponding to the ϕ_k, for $l \leqslant k \leqslant r$, are exactly those "lying in the angle" formed by p', p, p''. Now we use the range minimum data structure for (p,p') to determine the minimum angle μ in the interval $[l,r]$ with respect to p', i.e., $\mu = \min\{\phi'_k \mid l \leqslant k \leqslant r\}$. If μ is less than or equal to the angle θ of p'' with respect to p', then $\triangle(p,p',p'')$ contains a negative point, otherwise it is good.

The cases of other relative positions of p, p' and p'' are treated similarly.

The time for checking an individual triangle is $O(\log n)$ ($O(\log n_2)$). The angles ρ, λ and θ can be computed in constant time. The indices l and r can be found in logarithmic time using binary search. The query on the range minimum structure to find μ needs logarithmic time. Hence, the time for checking all triangles is $O(n^3 \log n)$ ($O(n_1^3 \log n_2)$). For each triangle its area is computed in constant time and both informations are stored in an array (of size $O(n_1^3)$). Then we are able to check in constant time whether a triangle is good and find its area. The negative points are no longer needed.

In the case where f is not the Euclidean area, the time depends also on the time needed to evaluate f on triangles. In this case let t be the maximum time to compute f on a triangle. Note, that t may depend on n. Then t appears as a factor in the above preprocessing times.

3.2. *The algorithm for Problem B*

Fix $p_0 \in \mathrm{POS}$. We want to find the maximum PN-polygon having p_0 as bottom vertex. Thus we only have to consider those points of POS which have y-coordinates no less than the one of p_0. For notational convenience, assume that this is true for all points of POS, that $p_1, \ldots, p_{n_1-1}$ is the counterclockwise ordering of $\mathrm{POS} - \{p_0\}$ with respect to p_0, and that p_1 has minimum angle.

The algorithm starts a counterclockwise scan through the points from p_1 to p_{n_1-1} using the precomputed cyclic order. It keeps track of some fans originating at p_1. There can be exponentially many such fans, but we shall see that we have to memorize only a polynomial number without missing those which still have the potential to become a maximum one. Informally the procedure can be described as follows. Let $1 < i \leqslant n_1 - 1$ and suppose we know, for each $j < i$, some "large" fans originating at p_1 and ending at p_j. For each such j we want to extend one such fan to p_i, such that the extended

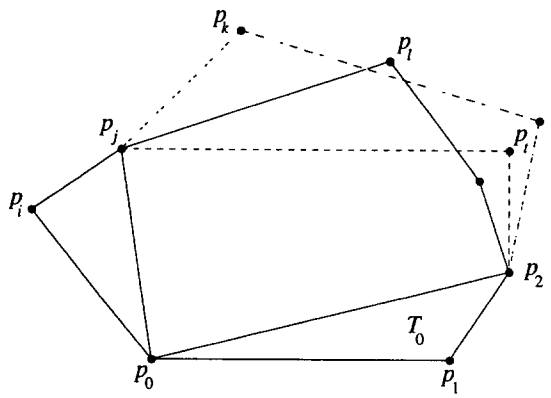

Fig. 6. $\triangle(p_0, p_i, p_j)$ is a convex continuation of the fans ending with $\triangle(p_0, p_j, p_l)$ (solid) and $\triangle(p_0, p_j, p_t)$ (dotted) but not of the one ending with $\triangle(p_0, p_j, p_k)$ (dashed). The area of the solid fan is larger than the one of the dotted one. The predecessor is defined as $P[i, j] := l$ and the area is defined as $F[i, j] = F[j, l] + f(\triangle(p_i, p_j, p_0))$. Hence the dotted fan is forgotten.

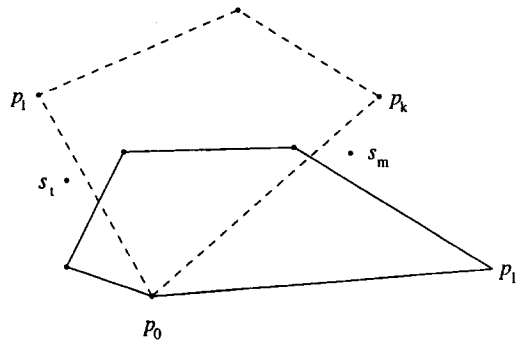

Fig. 7. The dashed fan from p_k to p_l is larger than the solid one but cannot be extended either way due to negative points s_t and s_m.

fan is maximum, provided that $\triangle(p_0, p_i, p_j)$ is good. Among the fans stored at p_j we select one with maximum area such that $\triangle(p_0, p_i, p_j)$ is a convex continuation. We then memorize the area of the extended fan and its last vertex before p_j. See Fig. 6.

Note that the fan of maximum area is not necessarily the one that covers most of the angle around p_0, see Fig. 7. However, the algorithm has stored for each intermediate point the information about which of the fans ending at the point has the largest area. Thus the current maximum is always available. Moreover, new fans are started during the scan, for example at p_k in Fig. 7. The crucial point is that we do not have to remember all possible fans ending at p_j, but only one for each predecessor of p_j because looking back from p_i to p_j, it is unimportant "how the fan looks between p_1 and p_j". One only has to know its area and the vertex before p_j, which we call the *predecessor* of the pair (i, j), denoted $P[i, j]$. The area of the fan is stored in $F[i, j]$. In Fig. 6, $P[i, j] = l$ and $F[i, j] = F[j, l] + f(\triangle(p_i, p_j, p_0))$.

In order to make finding the predecessor more efficient we arrange the areas $F[j, k]$ of fans ending at p_j in a prefix-maximum data structure. We associate with each $F[j, k]$ the slope $S[j, k]$ of the outer

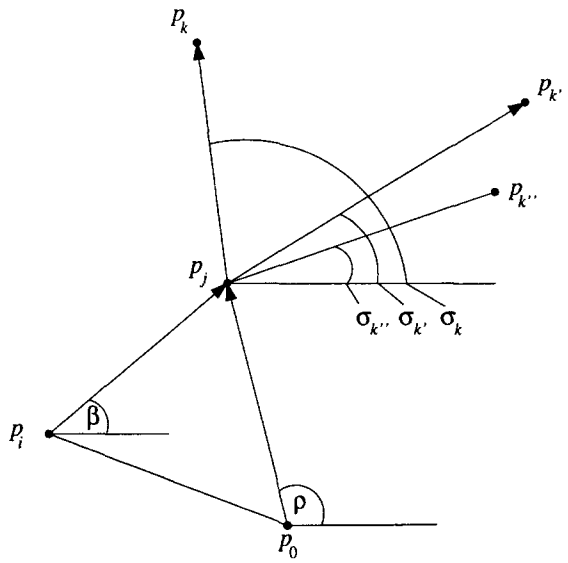

Fig. 8. $\sigma_k > \beta$ hence the fan ending with $\triangle(p_0, p_j, p_k)$ cannot be extended to p_i. The fans ending with $\triangle(p_0, p_j, p'_k)$ and $\triangle(p_0, p_j, p''_k)$ can both be extended to p_i. The value of $M[k']$ tells us which one has the larger area, $P[i,j] = M[k']$.

edge of $\triangle(p_0, p_k, p_j)$, i.e., the angle of the vector $p_j p_k$ with respect to the positive x-axis. Fix j and denote $S[j,k]$ by σ_k, $F[j,k]$ by f_k and let ρ be the slope of $p_0 p_j$ (see Fig. 8).

Then

$$\sigma_k \in [0, \rho] \cup [\rho + \pi, 2\pi).$$

Let us for simplicity assume that all σ_k are in $[0, \pi]$. We order the pairs (σ_k, f_k) (increasingly) according to their first component. Then we build a prefix maximum data structure on the second components. All this requires time $O(n_1 \log n_1)$.

Now, given the slope β of $p_i p_j$, we find the position of β among the σ's using binary search, i.e., $\sigma_k \leqslant \beta < \sigma_{k+1}$. Then the prefix maximum data structure allows to find an index $M[k]$ of the prefix maximum in constant time. We select the fan associated with $M[k]$, i.e., $P[i,j] = M[k]$. All this requires time $\log n_1$. Note that p_i, p_j and $p_{M[k]}$ form a right turn because $\sigma_k \leqslant \beta$, whence the continuation is convex.

The above description of the algorithm is more formally summerized in Fig. 9. Note that by setting $F[i,i] = 0$ at the beginning of loop 1 it is guaranteed that maximum computed in loop 3 is always defined.

We proceed by proving the correctness of the algorithm and establishing the running time of $O(n_1^2 \log n_1)$.

Let us first analyze the running time of the algorithm. The loop in line 0 is executed $(n_1(n_1-1))/2$ times, the time for each execution is constant, as is the time for the two instructions following the loop, whence the time for this part is $O(n_1^2)$. The nested loops marked 1 and 2 are each executed at most $n_1 - 1$ times each. The IF-instruction at 3 can be performed in time $O(\log n_1)$. Part 4 is outside loop 2 and requires $O(n_1 \log n_1)$ time at most. Hence the total time spent in loop 1 is $O(n_1^2 \log n_1)$. The time for the output part 5 is $O(n_1)$, noting that the indices i_0 and j_0 of the maximum F-value

Algorithm for Problem B

INPUT: Points $p_0, \ldots, p_{n_1-1}$, such that $p_1, \ldots, p_{n_1-1}$ are ordered counterclockwise around p_0, and the y-coordinates of p_i, $i = 1, \ldots, n_1 - 1$ are no less than the one of p_0.

AVAILABLE: For each triangle with positive vertices the information whether it is good and its "area" $f(\triangle(p_i, p_j, p_k))$, both in constant time.

OUTPUT: A maximum area fan of good triangles around p_0.

```
0  FOR i,j := 1,...,(n_1 - 1), i ⩾ j DO
       F[i,j] := P[i,j] := undefined;
   END (* FOR i,j *)
   F[1,1] := 0; (* initialization *)
   Build Prefix-Maxima data structure on F[1,1];
1  FOR i := 2,...,(n_1 - 1) DO   (* counterclockwise scan *)
       F[i,i] := 0; (* initialization of fans starting at p_i *)
2      FOR j := 1,...,(i - 1) DO   (* look back to p_j *)
3          IF △(p_i,p_j,p_0) is good
             THEN
                 Use Prefix-Maxima data structure at j to find
                 max := max_{k⩽j}{F[j,k] is defined and
                                  such that p_i,p_j,p_k form a right turn};
                 k_max := the corresponding index;
                 F[i,j] := f(△(p_i,p_j,p_0)) + max; P[i,j] = k_max;
       END (* FOR j *)
4      Compute the slopes S[i,k] and build Prefix-Maxima data structure
           on the pairs (S[i,k],F[i,k]) for the defined F[i,k], k < i;
   END (* FOR i *)
5  (* Output *)
   Find i_0 and j_0 such that F[i_0,j_0] := max{F[i,j] is defined | 1 ⩽ j ⩽ i ⩽ (n_1 - 1)};
   RETURN F[i_0,j_0]; (* Area of a maximum fan *)
   WHILE P[i_0,j_0] defined DO
       RETURN i_0;
       next := P[i_0,j_0]; i_0 := j_0; j_0 := next;   (* compute vertices of the fan *)
   END (* while *)
   RETURN i_0.
```

Fig. 9. Algorithm for solving Problem B.

can be updated at 3. The time for the algorithm thus is $O(n_1^2 \log n_1)$. The correctness of the algorithm is proved by establishing the following five claims.

1. Any sequence $P[i,j]$, $P[j,P[i,j]]$, ... induces a fan, provided that $P[i,j]$ is defined.
2. If $P[i,j]$ is defined then $F[i,j]$ is the area of the induced fan having $\triangle(p_0,p_i,p_j)$ as its last triangle.
3. For all i, j, $i \geqslant j$, if $F[i,j]$ is defined then it is less than or equal to the area of a maximum fan ending at p_i.
4. For all i, $F[i,j'] = \max\{F[i,j] \mid 1 \leqslant j \leqslant i,\ F[i,j] \text{ is defined}\}$ is the area of a maximum fan ending at p_i .

5. $F[i',j'] = \max\{F[i,j] \mid 1 \leqslant j \leqslant i \leqslant n_1 - 1,\ F[i,j] \text{ is defined}\}$ is the area of a maximum fan and hence of a maximum PN-polygon with bottom vertex p_0.

Proof. Claim 1. If $P[i,j]$ is defined, say $P[i,j] = k$, then, by part 3 of the algorithm, p_i, p_j, p_k form a right turn and $\triangle(p_0, p_i, p_j)$ is good. The argument inductively applies to $P[j,k]$, ..., until at some point $P[j',k'] = k'$. Then $P[k',k']$ is undefined, whence $\triangle(p_0, p_{j'}, p_{k'})$ is the first triangle of this fan.

Claim 2. By part 3 of the algorithm.

Claim 3. Clear by Claim 2.

Claim 4. Fix i. Let $p_{i_1}, \ldots, p_{i_m}$ be the vertices of a maximum fan ending at i, where $i_1 < \cdots < i_m = i$. We show by induction on j that for all i_j, $j = 1, \ldots, m$, $F[i_j, i_{j-1}]$ (respectively $F[i_1, i_1]$ for $j = 1$) is the area of the corresponding *initial segment* of the maximum fan, i.e., the fan with vertices $p_0, p_1, p_2, \ldots, p_{i_j}$. When processing p_{i_1}, the variable $F[i_1, i_1]$ is initialized to 0. $P[i_1, i_1]$ is set "undefined" in part 0.

Now, let $j \geqslant 2$. At p_{i_j} the algorithm looks also back at $p_{i_{j-1}}$. By inductive hypothesis, $F[i_{j-1}, i_{j-2}]$ (respectively $F[i_1, i_1]$ for $j = 2$) is defined, and its value is the area of the corresponding initial segment of the maximal fan. The triangle $\triangle(p_0, p_{i_j}, p_{i_{j-1}})$ is good, and $p_{i_j}, p_{i_{j-1}}, p_{i_{j-2}}$ form a right turn, because they are vertices of the maximum fan. Note that it cannot happen that $F[i_{j-1}, k] > F[i_{j-1}, i_{j-2}]$ and $p_{i_j}, p_{i_{j-1}}, p_k$ form a right turn, because this would give rise to a fan strictly larger than the maximum one. Hence $F[i_j, i_{j-1}]$ is set to $F[i_{j-1}, i_{j-2}] + f(\triangle(p_0, p_{i_j}, p_{i_{j-1}}))$.

It can, however, happen, that the predecessor $P[i_j, i_{j-1}]$ is not set to i_{j-2}. This can be the case if another fan ending at $p_{i_{j-1}}$ has the same area as the maximal one and $\triangle(p_0, p_{i_j}, p_{i_{j-1}})$ is a convex continuation for both.

Claim 5. Clear by Claim 4.

3.3. Algorithm for Problem A

The algorithm for Problem A does the preprocessing described in Section 3.1. Then it simply calls the algorithm for Problem B described in Section 3.2 for all possible choices of $p_0 \in$ POS. If the point chosen is a bottom vertex of some maximum PN-polygon, then the fan produced is a PN-polygon with the same area. The running time of this algorithm of Fig. 9 clearly is $O(n^3 \log n)$ $(O(n_1^2(n_1 \log n_1 + n_2 \log n_2)))$. The time for computing the good triangles and their areas in the preprocessing also is $O(n^3 \log(n))$ $(O(n_1^3 \log(n_2)))$. As indicated in Section 3.1 the time for the preprocessing depends on the time needed to evaluate f on triangles.

We summarize this as Theorem 1.

Theorem 1. *Let* POS, NEG $\subset \mathbb{R}^2$, $n_1 := |\text{POS}|$, $n_2 := |\text{NEG}|$ *and* $n = n_1 + n_2$. *Let* f *be a nonnegative real-valued function satisfying: if* A *and* B *are disjoint convex polygons then* $f(A \cup B) = f(A) + f(B)$, *and let* $t = t(n)$ *be the maximum time to compute* $f(T)$ *for some triangle* T.

Then a maximum (with respect to f*),* PN*-polygon can be found in time*

$$O\big(tn^3 \log n\big)\big(O\big(tn_1^2(n_1 \log n_1 + n_2 \log n_2 + n_1 \log n_2)\big)\big),$$

including the preprocessing. If f *is the Euclidean area then* t *is constant.*

4. Further results

We have shown that the problem of finding a maximum PN-polygon can be solved in time $O(n^3 \log n)$ if the area function f measures the Euclidean area. If we want to maximize a different quantity the running time might become larger by a factor proportional to the time to evaluate this quantity for a triangle.

We discuss the situation where one wants to maximize the number of positive points contained and show that in this case the order of the running time is maintained. This is achieved by an additional preprocessing step, which is similar to the one used to find the good triangles. We now want to compute for each triangle the number of positive points that are contained in it. Again, fix p, $p' \in \mathrm{POS}$, but now let ϕ_k and ϕ'_k be the polar angle of *positive* point p_k with respect to p and p', respectively, $p_k \in \mathrm{POS} - \{p, p'\}$. We then build a rectangle counting data structure on pairs (ϕ_k, ϕ'_k). This requires time $O(n_1 \log n_1)$. Doing this for all possible pairs of positive points requires $O(n_1^3 \log n_1)$.

Now consider triangle $T = \triangle(p, p', p'')$ as shown in Fig. 5 and note that a point p_k is contained in T if $\rho \leqslant \phi_k \leqslant \lambda$ and $(\pi + \lambda) \leqslant \phi'_k \leqslant (2\pi - \theta)$. Hence we may use the rectangle counting data structure for the pair (p, p') to find out in time $O(\log n_1)$ how many positive points meet these two conditions. As the points p and p' are not considered in the rectangle counting data structure we have to add 2 to get the desired number. Again if the relative location of p, p', p'' is different from the one in Fig. 5, one has to make straightforward modifications. As there are $O(n_1^3)$ good triangles the total time for this part is also $O(n_1^3 \log n_1)$.

Corollary 1. *Let* $\mathrm{POS}, \mathrm{NEG} \subset \mathbb{R}^2$, $n_1 := |\mathrm{POS}|$, $n_2 := |\mathrm{NEG}|$ *and* $n = n_1 + n_2$. *Then a* PN*-polygon containing a maximum number of positive points can be found in time* $O(n^3 \log n)$ $(O(n_1^2(n_1 \log n_1 + n_2 \log n_2 + n_1 \log n_2)))$, *including the preprocessing.*

The problem of finding a maximum area PN-polygon that has a given positive point as bottom vertex can be solved in time $O(n^2 \log n)$. This requires straightforward modifications of the preprocessing and only one execution of the algorithm for Problem B. The dependence on the time to evaluate f on a triangle is again multiplicative.

The running times given in Theorem 1 and Corollary 1 are, up to a logarithmic factor, proportional to the number of triangles defined by positive points.

We now address the problem of finding a maximum area PN-polygon (or one that covers a maximum number of positive points) that contains a given positive point p^* (but not necessarily as bottom vertex). It is not sufficient to execute the algorithm of Fig. 9 for all choices of the bottom vertex $p_0 \in \mathrm{POS}$ and to check after each execution, whether p^* is in the maximum polygon P found by the algorithm. Indeed, even if p_0 is a bottom vertex of a maximum PN-convex polygon P^* containing p^*, it can happen that the polygon P found by the algorithm does not contain p^*. Of course the areas of P and P^* are equal.

In [7] it is shown that the problem of finding a maximum area PN-polygon (or one that covers a maximum number of positive points) that contains a given positive point (but not necessarily as bottom vertex) can be solved in time $O(n^3 \log n)$. The algorithm is somewhat different from the one presented here, and we only sketch it. Again, one considers fans, but this time around the given point p^*. As p^* might lie in the interior of the maximum polygon, one has to consider also fans that wrap around this point completely. But then there is the problem of "closing the fan correctly". This is

handled by running the algorithm $O(n_1)$ times. At each execution a different positive point is chosen as the starting point for the fans. The algorithm succeeds if the starting point is a vertex of some maximum area polygon containing p^*. It is not clear whether this problem can also be solved in time $O(n^3)$ or better if f is the Euclidean area. If f cannot be evaluated in constant time on a triangle then the running time may again depend on that time.

Corollary 2. *Let* POS, NEG $\subset \mathbb{R}^2$, $n_1 := |\text{POS}|$, $n_2 := |\text{NEG}|$ *and* $n = n_1 + n_2$. *Let* f *be a nonnegative real-valued function satisfying: if* A *and* B *are disjoint convex polygons then* $f(A \cup B) = f(A) + f(B)$, *and let* $t = t(n)$ *be the maximum time to compute* $f(T)$ *for some triangle* T. *Let* $p^* \in \text{POS}$.

Then a maximum (with respect to f*),* PN*-polygon containing* p^* *can be found in time* $O(tn^3 \log n)$. *If* f *is the Euclidean area or the number of positive points contained then the running time is* $O(n^3 \log n)$.

The number of vertices in a maximum PN-convex polygon $P_{\max}$ found by the algorithm can be of order $\Omega(n)$, although there is another maximum PN-convex polygon with fewer vertices. For the purpose of image compression $\Omega(n)$ too large. An algorithm for finding minimum separating convex polygons due Edelsbrunner and Preparata [6] can sometimes reduce the number of vertices: given two finite sets of points A and B such that the convex hull of A does not contain a point from B, this algorithm finds a convex polygon Q with a minimum number of vertices such that Q contains A but no point from B. It runs in time $m \log m$, where $m = |A| + |B|$. In our case choose $A = P_{\max} \cap \text{POS}$ and $B = \text{NEG}$.

It can of course happen, that the application of the algorithm of Edelsbrunner and Preparata does not reduce the number of vertices. However, the algorithm of Section 3.2 can be modified to find a maximum PN-convex polygon with a small number of vertices, say at most M. To this end one has to keep track of the number of vertices of the fans constructed. We only sketch how this is done. Another index is added to the arrays of the F- and P-values. Then $F[i,j,l]$, $2 \leqslant l \leqslant M$, is the area of the fan which ends at p_i, has p_j as the vertex before p_i, and consists of $l-2$ triangles (hence has l vertices). Analogously $P[i,j,l]$ is the vertex of this fan before p_j. If the triangle $\triangle(p_i, p_j, p_0)$ is good then the update rule of the F-values is

$$F[i,j,l] := f\big(\triangle(p_i,p_j,p_0)\big) + max,$$
$$\text{where } max := \max_{k \leqslant j}\{F[j,k,l-1] \text{ is defined and } p_i, p_j, p_k \text{ form a right turn}\}.$$

The running time is increased by a factor at most M, because the update has to be done for all l, $2 \leqslant l \leqslant M$. Note that a PN-convex polygon with at most M vertices will in general have smaller area than a PN-convex polygon without this restriction. The preprocessing is not affected by this modification.

Corollary 3. *Let* POS, NEG $\subset \mathbb{R}^2$, $n_1 := |\text{POS}|$, $n_2 := |\text{NEG}|$ *and* $n = n_1 + n_2$, *and let* $M \in \mathbb{N}$. *Let* f *be a nonnegative real-valued function satisfying: if* A *and* B *are disjoint convex polygons then* $f(A \cup B) = f(A) + f(B)$, *and let* $t = t(n)$ *be the maximum time to compute* $f(T)$ *for some triangle* T.

Then a maximum (with respect to f*),* PN*-polygon which has at most* M *vertices can be found in time* $O(tn^3 + Mn^3 \log n)$. *If* f *is the Euclidean area this becomes* $O(n^3 + Mn^3 \log n)$, *if* f *is the number of positive points contained it becomes* $O((M+1)n^3 \log n)$.

The algorithm presented in Section 3.2 can be parallelized in a rather straightforward manner. The n_1 many positions of the sweep-ray are nevertheless treated sequentially. At each position n processors compute the $F[\cdot, \cdot]$- and $P[\cdot, \cdot]$-values and build the prefix maximum data structure in parallel. This requires $O(\log n)$ time at each position of the sweep-ray. Hence the total time for the problem is $O(n \log n)$ using $O(n)$ processors. This is done in parallel for each $p_0 \in$ POS. The preprocessing described in Section 3.1 can also be parallelized to run in time $O(n \log n)$ with n^2 processors. Hence we have the following.

Corollary 4. *With the notation of Theorem 1 a maximum (with respect to f),* PN*-polygon can be found in time $O(tn \log n)$ using n^2 processors.*

For the sake of completeness we mention that a parallel solution is possible in polylogarithmic time if one is willing to use $O(n^7)$ many processors. The preprocessing can be performed in constant time with n^4 processors (on a CRCW-PRAM) in case that f is the Euclidean area. Then, one models the problem as a maximum path problem in a weighted directed acyclic graph $G = (V, E, w)$ as follows. Fix the bottom vertex p_0 and assume that no other positive point has y-coordinate less than p_0. The graph has a vertex v_T for each good triangle T that has p_0 as a vertex and an additional vertex start. Let $v_T, v_S \in V$, then (v_T, v_S) is a directed edge if S is a counterclockwise continuation of T with respect to p_0. The weight $w((v_T, v_S))$ is the area of of S, i.e., $f(S)$. Moreover, for each $v_T \in V$ we have an edge (start, v_T) with weight $F(T)$. Clearly paths in G correspond to fans with respect to p_0 and vice versa and G does not contain any directed cycles. Now one computes the nth power of the adjacency matrix of G using fast exponentiation and *maximum* as operation in order to find the longest path. (One has to store some additional information to be able to recover a path and not only its length.) Note that the size of the adjacency matrix of G is of order $n^2 \times n^2$, whence fast matrix multiplication can be done in sequential time $O(n^6)$ and hence in time $O(\log n)$ with $O(n^6)$ processors, see, e.g., [11]. Note, that in order to use faster methods for matrix multiplication the operations used must form a ring. The parallel construction of the adjacency matrix is possible in constant time with n^4 processors. Again the problem has to be solved for all choices of p_0 whence one gets the following corollary.

Corollary 5. *A maximum area* PN*-polygon can be found in polylogarithmic time using $O(n^7)$ processors.*

Acknowledgements

I would like to thank an anonymous referee for helpful comments on a previous version of this paper.

References

[1] A. Aggarwal and S. Suri, Fast algorithms for computing the largest empty rectangle, in: Proc. 3rd Symp. Comput. Geom. (1987) 278–290.

[2] B. Chazelle, R. Drysdale and D. Lee, Computing the largest empty rectangle, SIAM J. Comput. 15 (1986) 300–315.

[3] B. Chazelle, A functional approach to data structures and its use in multidimensional searching, SIAM J. Comput. 17 (1988) 427–462.
[4] V. Chvátal and G. Klincsek, Finding largest convex subsets, Congressus Numerantium 29 (1980) 453–460.
[5] D. Eppstein, M. Overmars, G. Rote and G. Woeginger, Finding minimum area k-gons, Discrete Comput. Geom. 7 (1992) 45–58.
[6] H. Edelsbrunner and F.P. Preparata, Minimal polygonal separation, Inform. and Comput. 77 (1988) 218–232.
[7] P. Fischer, Finding maximum convex polygons, in: Z. Esik, ed., Proc. Fund. Comput. Theory (FCT'93), Lecture Notes in Computer Science 710 (Springer, Berlin, 1993) 234–243.
[8] P. Fischer, Learning unions of convex polygons, in: J. Shawe-Taylor and M. Anthony, eds., Computational Learning Theory: Euro-Colt'93 (1994) 61–67.
[9] P. Fischer, More or less efficient agnostic learning of convex polygons, in: Proc. 8th Conf. Comput. Learning Theory (ACM, New York, 1995) 337–344.
[10] P. Fischer and S. Kwek, Minimizing disagreements for geometric concepts using dynamic programming, in preparation, 1995.
[11] J. JáJá, An Introduction to Parallel Algorithms (Addison Wesley, Reading, MA, 1992).
[12] F. Preparata and M. Shamos, Computational Geometry, An Introduction (Springer, New York, 1985).