Implicit Convex Polygons

Francisco Gómez[1] Ferran Hurtado[2] Suneeta Ramaswami[3] Vera Sacristán[2]
Godfried Toussaint[4]

July 19, 2000

Abstract

Convex polygons in the plane can be defined explicitly as an ordered list of vertices, or given implicitly, for example by a list of linear constraints. The latter representation has been considered in several fields such as GIS, robotics and computer graphics.

In this paper, we investigate many fundamental problems for implicitly represented polygons and give simple and fast algorithms that are easy to implement. We uncover an interesting partition of the problems into two classes: those that exhibit an $\Omega(n \log n)$ lower bound on their complexity, and those that yield $O(n)$ time algorithms via prune-and-search methods.

1 Introduction

In many GIS applications, linear constraints are considered to be a good representation of geometric objects because it is practically viable in terms of storage and efficiency of operations. In this representation, queries on geometric objects are expressed as operations on linear inequalities. The development of detailed and efficient algorithms for standard geometric operations in the linear constraint representation model is an important topic of research for GIS and graphics systems. For instance, in [8, 9, 10], the authors pose several questions on the development of efficient algorithms for fundamental geometric problems in this model.

In this paper, we design simple and fast algorithms for many fundamental operations on spatial data. In this model, the input is a convex polygon given as a set of linear constraints, i.e., by the intersection of a possibly redundant set of halfplanes. Given a set $H = \{H_1, \ldots, H_n\}$ of halfplanes, we consider various computation and optimization problems for the convex polygon $P(H) = \cap_{i=1}^{n} H_i$. Some examples are computing the two supporting lines of $P(H)$ from an external point, and computing the minimum distance to the polygon from an external point. We also consider problems for which the input consists of two convex polygons. Some examples are finding the two separating tangents of two disjoint polygons, as well as their two common external tangents, and computing the minimum distance between two disjoint polygons. We will also briefly discuss analogous results when the input is a polygon given as the convex hull of a set of unordered points.

[1]Dep. de Matemática Aplicada, E.U. Informática, Universidad Politécnica de Madrid, Ctra. de Valencia Km 7, 28031 Madrid, Spain. E-mail: `fmartin@eui.upm.es`

[2]Dep. Matemàtica Aplicada II, Universitat Politècnica de Catalunya, Pau Gargallo 5, 08028 Barcelona, Spain. E-mail: `hurtado@ma2.upc.es, vera@ma2.upc.es` Partially supported by CUR Gen. Cat. 1999SGR0356 and Proyectos DGES-MEC PB96-0005-C02-02 and PB98-0933.

[3]Dep. Computer Science, 322 Business and Science Building, Rutgers University, Camden, NJ 08102, USA. E-mail: `rsuneeta@crab.rutgers.edu`

[4]School of Computer Science, McGill University, 3480 University, Montréal, Québec, Canada H3A 2A7. E-mail: `godfried@cs.mcgill.ca`

Most of these problems can be solved very efficiently, particularly when the polygons are known to be convex, in $O(n)$ time, for example, with "rotating calipers" [20]. However, many of these algorithms assume that the order of the polygon vertices is given. These problems have not been investigated for the situation when the vertices of the convex polygon are not known, or not specified in order. Obviously, one approach to these problems is to "construct" the polygon; that is, obtain the ordered list of the vertices of the polygon by intersecting the halfplanes that define it, and then use existing algorithms for "ordered" polygons to solve the problems. However, computing halfplane intersections implies that such a procedure will always be $\Omega(n \log n)$. The more interesting question is to determine which of these problems can be solved in optimal $\Theta(n)$ time, and which have an inherent complexity $\Omega(n \log n)$. We show, for example, that computing the two supporting lines from an external point, or computing the minimum distance between two polygons takes linear time, whereas computing a diametral pair of points, or the minimum enclosing circle of $P(H)$ are $\Omega(n \log n)$.

Our optimal linear time algorithms are based on a prune-and-search strategy. The prune-and-search paradigm was first introduced in a geometric context by Megiddo [17, 18] and Dyer [6, 7] to solve the linear programming problem. This paradigm has been used to solve a wide variety of problems. Some examples are Megiddo's and Dyer's linear time algorithm for the Euclidean 1-center problem [7, 17], Kirkpatrick and Seidel's optimal planar convex hull algorithm [14], and Hurtado et al.'s optimal algorithms for some facility location problems [13]. In this paper, we demonstrate further applications of this strategy to give optimal linear time algorithms for a wide variety of fundamental geometric problems.

Each of these algorithms applies an oracle (the search step) that allows it to discard a constant fraction of the halfplanes H_i in linear time (the prune step). By recursively applying the prune-and-search strategy, we obtain a linear-time algorithm. In order to guarantee that a constant fraction of the input halfplanes is eliminated at each step, it is usually necessary to compute a median value, which can be obtained in linear time [4, 11]. The linear time search procedure is specific to each problem, whereas the prune step is always the same. Identifying linear time search procedures for various fundamental geometric problems is the main contribution of this work. When such a search procedure does not exist, $\Omega(n \log n)$ lower bounds are proved. Our lower bound proofs are based on reductions from elementary problems such as element uniqueness, set intersection and max-gap on the circle.

The paper is organized as follows: Section 2 describes the general technique used to obtain optimal linear time algorithms for problems involving one implicit polygon given as the intersection of halfplanes. In particular, the prune procedure, which is general and applies to all the problems, is described there. Section 3 discusses problems involving one implicit polygon. The search procedures are described in Section 3.1, whereas Section 3.2 contains superlinear complexity problems. Section 4 discusses problems involving two implicit polygons. In Section 5, we describe prune and search methods for an alternative implicit representation of convex polygons, namely, as the convex hull of a set of points. We conclude with Section 6, which contains a summary of our results.

2 The technique for halfplanes

As stated earlier, our algorithms follow a prune-and-search strategy inspired by Megiddo's algorithm for solving the linear programming problem in [17]. The strategy of Megiddo in [17, 18] was independently studied and improved by Dyer in [6, 7]. The basic idea of the paradigm is the following:

1. *Search step*: Solve the problem constrained to a line, if such a problem has a solution. In

any case, determine which side of the line, i.e., which halfplane, contains the solution to the original problem.

2. *Prune step*: Recursively apply the search procedure to reduce the size of the original problem.

In order for the algorithm to run in linear time, it is essential that the search step is done in linear time and the prune step eliminates a fixed fraction of the input halfplanes. In other words, if r is the fraction of halfplanes discarded at each step, the overall cost of the algorithm is

$$T(n) = O(n) + O((1-r)n) + O((1-r)^2 n) + \ldots + O((1-r)^i n) + \ldots = O(n).$$

2.1 The prune procedure

Let $P(H)$ be a polygon defined as the intersection of a set H of halfplanes. We start by presenting the prune step of the algorithm, because it is the same for all the problems. In order to do that, we will assume the existence of an oracle, to be described later: a linear time algorithm which is able to decide, given a line ℓ, which side of ℓ contains the point(s) of $P(H)$ that solves the problem (for example, the point of $P(H)$ that lies closer to a given external point p, or the x-coordinate that gives a longer vertical chord of $P(H)$). If the solution lies on the line, the oracle detects it and computes it. As pointed out earlier, this search procedure will be specific to each problem.

Given such an oracle, the prune procedure is as follows. Consider the halfplanes of the input set H. Halfplanes of the form $y \leq ax + b$ will be called upper halfplanes, and those of the form $y \geq ax + b$ will be called lower halfplanes. We will be comparing the slopes of the upper halfplanes, as well as those of the lower halfplanes. In both cases, the comparison will be well defined, as we will consider the slopes to increase counterclockwise.

1. The process starts by pairing up the halfplanes. Upper halfplanes are always paired with upper halfplanes, and lower with lower. Vertical halfplanes can be considered as being upper or lower, as necessary. This process allows to pair up all the halfplanes, except at most two of them. If a pair consists of halfplanes defined by two parallel lines, one of them is redundant and may be discarded. Now consider the set C of intersection points of the pairs of lines just formed.

2. Compute the median value y_m of the y-coordinates of all the points in C and apply the oracle to the line $y = y_m$ to determine whether the solution to the problem lies on, above or below the line. If the solution lies on the line $y = y_m$, the oracle actually finds it, and the algorithm ends. Otherwise, the solution lies in one of the two open halfplanes. Without loss of generality, assume that the solution point lies to the left of the line $y = y_m$.

3. For each crossing point $c_i \in C$ located in the opposite halfplane, i.e. right of the line in our example, it will be possible to discard one of the two halfplanes that form it in the following way (see Figure 1):

   - If the two halfplanes are lower halfplanes, the one that has bigger slope is irrelevant to the left of c_i and, hence, to the left of the line ℓ, and it can be eliminated from the input because it cannot determine or affect the solution point.
   - Analogously, if the two halfplanes are upper halfplanes, the one that has smaller slope is irrelevant and it can be eliminated from the input.

- Notice that vertical halfplanes can be used as if they were either lower or upper, because they can only be irrelevant once the oracle has decided which side of the line ℓ the solution lies. Notice also that for pairs formed by parallel lines, one of the pair was already discarded in the first step.

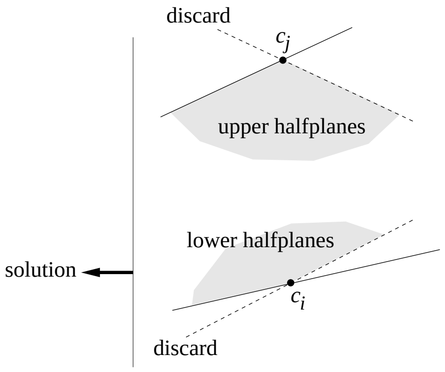


Figure 1: How to discard a halfplane for each crossing point c_i lying in the halfplane opposite to the solution.

Notice that at least a fourth of the input is discarded during the prune step. Therefore, the algorithm runs in linear time if the oracle does, because it only consists of pairing up the halfplanes, computing the intersection points of each pair, computing a median, and determining if each halfplane can be discarded or not.

It is important to notice that the previous procedure does not depend on the problems to be solved, and that the specificity of each problem will appear in the search procedure.

2.2 The search procedure

The search procedure always consists of deciding, given a vertical line ℓ, which side of ℓ contains the points of the polygon P that determine the solution to the given problem. If the points lie on the line, the procedure should detect it and return the points.

In general, it can be said that the search algorithm determines the solution to the given problem restricted to the line ℓ. In other words, it finds the point in $P \cap l$ (if any) that optimizes the solution and uses it to determine which side of ℓ would improve it. Of course, when $P \cap l = \emptyset$, the solution lies on the same side of ℓ as P. The search procedure is the part of the algorithm that depends on the problem and will be described in the following section.

The search procedure must have linear time complexity, for the overall algorithm to be linear. As we will see, some problems admit such a search procedure, and some do not. We will provide examples of the first class of problems in Section 3.1, while examples of the last class are presented in Section 3.2.

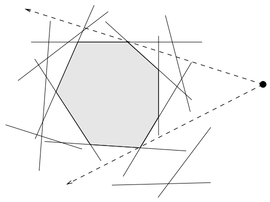
3 One Implicit Polygon given as Intersection of Halfplanes

In this section, we discuss problems that involve one implicit polygon specified as the intersection of n halfspaces. We start with linear complexity problems that can be solved using the prune and search technique. We then discuss some $\Omega(n \log n)$ lower bound results for implicit polygons in the algebraic computation tree model.

3.1 Linear complexity problems

We present several problems that can be solved in optimal linear time using the prune-and-search technique described in the previous section. The prune procedure described in Section 2.1 is the same for all the problems, and the proofs of this section only specify the search procedure.

Supporting lines: *Given a non-empty polygon $P(H)$ defined as the intersection of a set H of n halfplanes, and a point p external to $P(H)$, compute the two supporting lines from p to $P(H)$.*



First notice that the two restrictions in the statement of the problem can be avoided: the intersection of a set of halfplanes can be tested to be empty in linear time using Megiddo's algorithm for linear programming [17]; checking if a point belongs to an intersection of halfplanes is trivially done in linear time.

We describe how the search procedure works for this problem. We will describe the procedure for the supporting ray from p that has $P(H)$ to its right (see Figure 2). The procedure for the other supporting ray works similarly. Let ℓ be a vertical line. The goal is to decide which side of ℓ

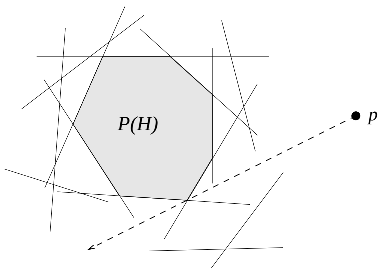


Figure 2: The supporting ray that has $P(H)$ to its right.

contains the point(s) of $P(H)$ that determine the supporting line. This is done by maintaining the intersection of ℓ with the successive halfplanes.

1. First consider the intersection of line ℓ with all the halfplanes determined by lines parallel to ℓ. Notice that, in fact, only two of these halfplanes can be relevant in determining the polygon $P(H)$. If the intersection of ℓ and these two halfplanes is empty, the polygon itself is empty. If only one of the two intersections is empty, it is easy to determine the side of ℓ that contains the constraint polygon. If none of the two intersections is empty, proceed to the next step.

2. Intersect the line ℓ with all the upper halfplanes. The result is a ray whose origin is determined by one or at most two halfplanes, depending on whether ℓ intersects the associated polygonal region at an edge or at a vertex. In the event that ℓ intersects a vertex that is common to more than two halfplanes, all but the extremal ones can be discarded (see Figure 3). Proceed the same way with the lower halfplanes.

3. If the two rays are disjoint, ℓ does not intersect the polygon. The intersection of the at most four halfplanes that determine the two rays shows which side of ℓ contains the polygon $P(H)$ (in some cases, it may even show that the polygon is empty).

4. If the two rays have a common segment, then the line ℓ intersects polygon $P(H)$, and the intersection segment bt is found (refer to Figure 3). The decision is then taken by determining the position of the point p with respect to the halfplanes that determine the endpoints b and t:

   - If p lies in the region labeled A (this region can collapse to a ray, if the top endpoint t of $\ell \cap P(H)$ is determined by a single halfplane), then the tangent from p is supported by t, and the problem is solved.
   - If p lies in the region labeled B, then the tangent from p is supported by a point lying to the right of ℓ.
   - If p lies in the region labeled C (this region can collapse to a ray, if the bottom endpoint b of $\ell \cap P(H)$ is determined by a single halfplane), then the tangent from p is supported by b, and the problem is solved.
   - If p lies in the region labeled D, then the tangent from p is supported by a point lying to the left of ℓ.

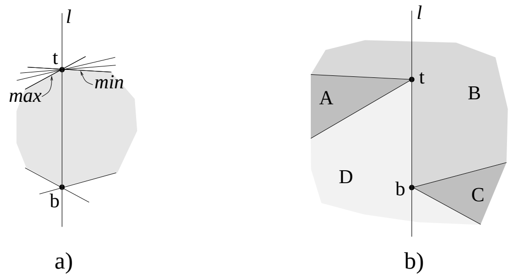
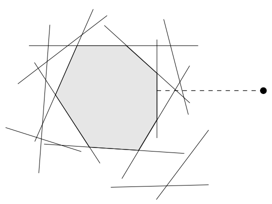


Figure 3: a) The halfplanes of H that determine the intersection of ℓ and $P(H)$. All except the extremal ones, `max` and `min`, can be discarded. b) The induced decomposition of the plane.

The above procedure requires finding intersections of ℓ with the n halfplanes and finding the extremal halfplanes, giving the following result.

Theorem 1 *The supporting lines problem for an implicit polygon defined as the intersection of n halfplanes can be solved in optimal $\Theta(n)$ time.*

Minimum distance to a point: *Given a non-empty polygon $P(H)$ defined as the intersection of a set H of n halfplanes and a point p, compute the minimum distance from $P(H)$ to p.*



Start by detecting whether the polygon contains the point. If this is the case, the distance is zero. Otherwise, the minimum distance from $P(H)$ to p is achieved at a point that belongs to the chain of the boundary of $P(H)$ that is visible from p. Moreover, the distance from p is a unimodal function over that chain. Hence, the algorithm only needs to deal with those halfplanes that may form the visible chain.

The algorithm first computes the two supporting lines from p to $P(H)$. Let h_l and h_r be the two halfplanes of H that determine the supporting lines and do not contain p in their interior. In other words, h_l and h_r are the two halfplanes that determine the leftmost and the rightmost edges of the visible chain as seen from p. The halfplanes of H that may form the visible chain from p are the halfplanes that do not contain p and have slopes between those of h_l and h_r. These form the input to the algorithm.

Given a vertical line ℓ, the search procedure is as follows:

1. As described for the previous problem, detect the intersection of the line ℓ with the polygon $P(H)$. If the intersection is empty, detect the relative position of the polygon with respect to the line.

2. If the intersection is not empty, it must be a segment or a ray. Each of its endpoints must be determined by either one or two halfplanes, depending on whether it is a point interior to an edge or a vertex of the visible chain.

   For each halfplane h determining an endpoint of $\ell \cap P(H)$, let p' be the orthogonal projection of p onto h. Since the distance function from p is unimodal along the visible chain of P, it follows that the relative position of p' with respect to the line ℓ gives the position of the solution (see Figure 4):

   - If the endpoint is determined by a single halfplane (as in the upper examples of Figure 4), then the solution lies on the same side of ℓ as p'.
   - If the endpoint is determined by two halfplanes (as in the lower examples of Figure 4), then either the projections of p onto the halfplanes lie on two different sides of ℓ, in which case the solution has been found, or they both lie on the same side of ℓ, in which case the solution also lies on that side.

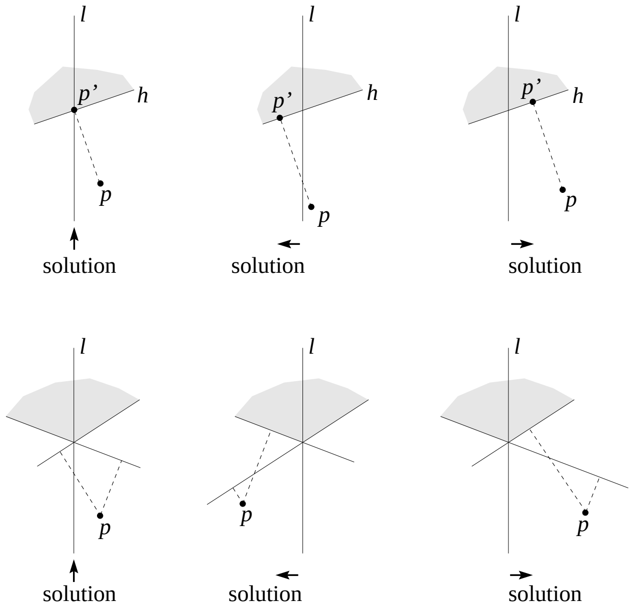
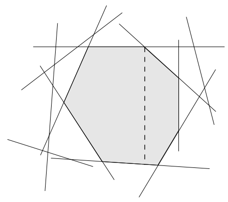


Figure 4: How to detect the relative position of the solution with respect to a line ℓ that intersects the visible chain.

This procedure requires computing the supporting lines, intersecting ℓ with the resulting halfplanes, computing extremal planes, and projecting p onto at most four lines. We thus have the following result.

Theorem 2 *The minimum distance to a point problem for an implicit polygon defined as the intersection of n halfplanes can be solved in optimal* $\Theta(n)$ *time.*

Longest vertical chord: *Given a non-empty polygon* $P(H)$ *defined as the intersection of a set* H *of* n *halfplanes, compute a longest vertical chord in* $P(H)$*.*



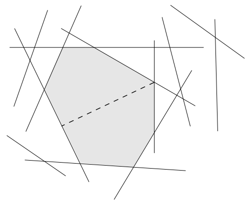
The function that assigns to every x-coordinate the length of the corresponding vertical chord in $P(H)$ is unimodal [5]. This allows us to solve the problem using the following search procedure. Given a vertical line ℓ, compute its intersection with $P(H)$. If it is empty, the solution, i.e. the x-coordinate that determines the chord, lies to the left or right of ℓ, depending on the relative position of $P(H)$ with respect to ℓ. If the intersection $\ell \cap P(H)$ is not empty, there are at most four halfplanes that determine it. The slopes of the lines that define these halfplanes can be used to determine the side of ℓ that locally increases the length of the chord, which must be the side containing the solution because of the unimodality of the vertical chord length. We thus have the following result.

Theorem 3 *The longest vertical chord problem for an implicit polygon defined as the intersection of n halfplanes can be solved in optimal* $\Theta(n)$ *time.*

3.2 Superlinear complexity problems

In this section, we present some problems that are $\Omega(n \log n)$ when the input is a convex polygon specified as the intersection of n halfplanes. In other words, any solution of these problems is at least as expensive as "constructing" the polygon and then applying the known algorithms for polygons given by the ordered list of their vertices. The lower bounds will hold in the algebraic computation tree model, as defined in [2].

Width: *Compute the width of a non-empty polygon* $P(H)$ *defined as the intersection of a set* H *of* n *halfplanes.*



Recall that once the vertices of the polygon are known and in order, the width can be computed in $O(n)$ time using the rotating calipers method [12]. Hence, the problem can be solved in $O(n \log n)$ time.

The lower bound $\Omega(n \log n)$ is proved by reduction from *set disjointness*: given two sets A and B of n real positive numbers, determine whether or not $A \cap B = \emptyset$. The reduction is as follows: Let $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_n\}$, and consider the following points on the unit circle (see Figure 5):

- for each $a_i \in A$, construct $p_i = \left(\dfrac{1 - a_i^2}{1 + a_i^2}, \dfrac{2a_i}{1 + a_i^2} \right)$;

- for each $b_i \in B$, construct $q_i = \left(-\frac{1-b_i^2}{1+b_i^2}, -\frac{2b_i}{1+b_i^2}\right)$.

Each point p_i (respectively q_i) is the intersection of the line through the origin that forms an angle $2\arctan a_i$ (resp. $2\arctan b_i$) with the horizontal axis, and the first (resp. third) quadrant of the unit circle centered at the origin . Consider the line through each point p_i and q_i, tangent to the

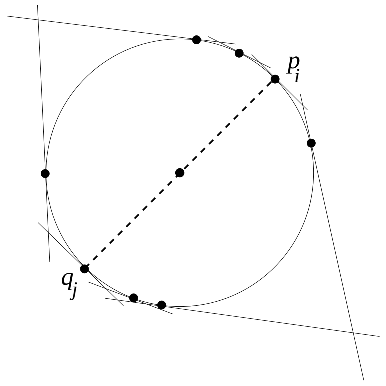


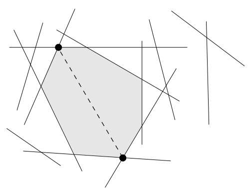
Figure 5: If the width is 2, then $p_i = -q_j$ for some i, j. Hence $a_i = b_j$ and $A \cap B \neq \emptyset$.

circle. Among the two halfplanes that each of these lines define, consider the one containing the circle. Apply the *width* algorithm to this intersection of halfplanes. If the width equals 2, then $A \cap B \neq \emptyset$, otherwise (i.e., if the width is greater than 2) A and B are disjoint.

As the reduction is trivially made in $O(n)$ time, we obtain the following result.

Theorem 4 *The width problem for an implicit polygon defined as the intersection of n halfplanes has complexity $\Theta(n \log n)$.*

Diameter: *Compute a diametral pair of points of a non-empty polygon $P(H)$ defined as the intersection of a set H of n halfplanes.*



It is well known that, once the vertices of the polygon are known and in order, a diametral pair of points can be computed in $O(n)$ time [19]. Hence, the problem can be solved in $O(n \log n)$ time.

The lower bound $\Omega(n \log n)$ is proved by reduction from *maximum gap on a quadrant of a circle.* This problem was proved to have complexity $\Omega(n \log n)$ by Lee and Wu [15] in the algebraic computation tree model. The problem is stated as follows: Given n points on the first quadrant of the unit circle, find the maximum straight line distance between neighboring points on the unit circle.

The reduction is as follows: Given n points on the first quadrant of the unit circle, $p_1, \ldots, p_n$, compute the diametrically opposite points $q_i = -p_i$, $i = 1, \ldots, n$. Consider the line through each point p_i and q_i, tangent to the circle (refer to Figure 6). Among the two halfplanes that each of these lines define, consider the one that contains the circle. Add two new lines to the set, namely the line through the rightmost points p_1 and q_1, together with the line through the leftmost points p_n and q_n. Among the two halfplanes that each of these two lines define, take the one that contains all the points p_i, q_i.

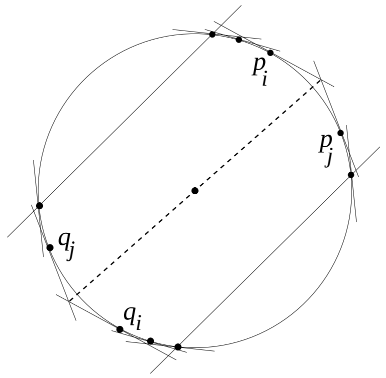
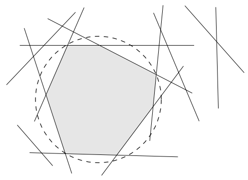


Figure 6: The diametral pair is reached at the intersection of the tangent lines at the two points p_i, p_j that define the maximum gap.

Apply the *diameter* algorithm to the intersection of all these halfplanes. The diameter is reached at the intersection of the two tangent lines to the circle at the points p_i, p_j (and q_i, q_j) that define the maximum gap in $p_1, \ldots, p_n$. Since the reduction is done in linear time, we obtain the following result.

Theorem 5 *The diameter problem for an implicit polygon defined as the intersection of n halfplanes has complexity $\Theta(n \log n)$.*

Smallest enclosing circle: *Compute the smallest enclosing circle of a non-empty polygon $P(H)$ defined as the intersection of a set H of n halfplanes.*



Once the vertices of the polygon are known, the smallest enclosing circle can be computed in linear time using Megiddo's [17] and Dyer's [7] prune-and-search algorithm. In this case, there is no need to know the order of the vertices of the polygon. In fact, the algorithm solves the smallest enclosing circle problem for any set of points in the plane. Hence, the problem can be solved in $O(n \log n)$ time.

The lower bound $\Omega(n \log n)$ is proved by reduction from *maximum gap on a quadrant of a circle.* The construction is the same as in the previous reduction (refer to Figure 7).

Apply the *minimum enclosing circle* algorithm to a set of halfplanes defined exactly as in the previous problem. The minimum enclosing circle is determined by two diametrically opposite vertices of the implicit polygon that identify the maximum gap in $p_1, p_2, \ldots, p_n$. In other words, the vertices that define the minimum enclosing circle are the intersection of the two tangent lines through the points p_i, p_j and q_i, q_j, where (p_i, p_j) gives the maximum gap. This is because the polygon is symmetric about the origin, and any circle with center other than the origin must have a larger enclosing radius. Since the reduction is done in linear time, we obtain the following result.

Theorem 6 *The smallest enclosing circle problem for an implicit polygon defined as the intersection of n halfplanes has complexity $\Theta(n \log n)$.*

It is worth noting here that that the *largest enclosed circle* problem (finding the largest circle enclosed in $P(H)$) can be formulated as a linear program in one more dimension (see [1] and [22]), and hence can be solved in linear time using a prune-and-search algorithm.

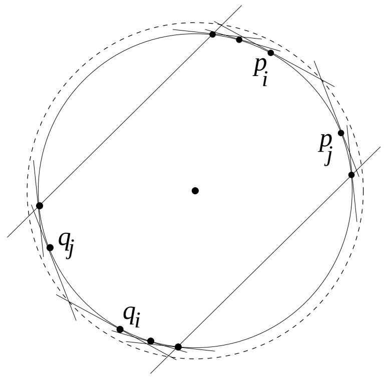


Figure 7: The minimum enclosing circle is determined by the points p_i, p_j that define the maximum gap.

4 Two Implicit Polygons given as Intersection of Halfplanes

The problems addressed in this section involve two polygons $P = P(H_1)$ and $Q = P(H_2)$, each given as the intersection of halfplanes. Section 4.1 describes linear-time algorithms for some problems, whereas Section 4.2 describes a problem with an $\Omega(n \log n)$ lower bound. Each of the linear complexity problems can be reformulated as finding some line or strip whose slope is defined by the problem at hand. As before, all the linear-time algorithms use the prune-and-search strategy, but the prune step here differs from the prune step for problems involving only one polygon. We start by presenting the general form of the prune and search algorithms for two implicit polygons.

1. Find the chain of each polygon that determines the solution (upper, lower, right or left, depending on the problem and on the relative position of the polygons). This step, which is part of the search procedure, finds the leftmost and the rightmost vertices of the two polygons and compares their relative positions. Once we have only one chain per polygon, say $chain(P)$ and $chain(Q)$, a total order is defined on the slopes of the halfplanes that may form the chains (the slopes are increasing in counter-clockwise order around the chain), which allows us to prune the input.

2. Compute the median slope m of all the halfplanes that may form the chains.

3. Find t_P and t_Q, the lines that have slope m and are tangent to $chain(P)$ and $chain(Q)$ at some points p and q respectively.

4. Use the search step to decide if the solution slope is equal to, bigger or smaller than m, that is, if t_P and t_Q should be rotated counterclockwise or clockwise in order to find the solution.

5. This is the prune step, which is illustrated in Figure 8 with an example.

   - If the solution slope is m, the problem is solved and the algorithm ends.
   - If the solution slope is bigger than m, all the halfplanes of H_1 and H_2 that have smaller slope can be eliminated.
   - If the solution slope is smaller than m, all the halfplanes of H_1 and H_2 that have bigger slope can be eliminated.

   In the two last cases, p and q replace the eliminated endpoints of the corresponding chain.

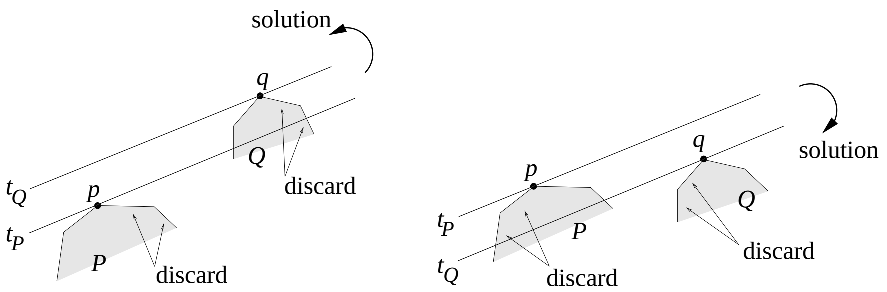


Figure 8: Discarding halfplanes when the solution slope is bigger than m (left) or when it is smaller than m (right).

Since half the input is pruned at every step of the algorithm, the overall complexity will be linear as long as the search procedure is linear. Notice that all the other steps are linear, including finding the leftmost and rightmost vertices by linear programming (step 1), finding the median (step 2), and computing the tangents by linear programming (step 3). We are now ready to present the search procedure for each problem.

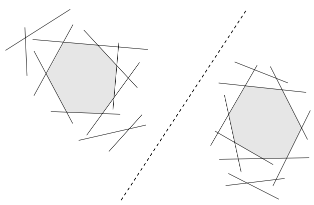
In general, the search procedure has two main steps.

1. Decide which chains are relevant to the problem.
2. Decide if the solution slope is equal to, bigger or smaller than a given slope m.

We will discuss how to solve both steps in linear time for each problem.

4.1 Linear complexity problems

Separating line: *Given two non-empty and disjoint polygons $P(H_1)$ and $P(H_2)$, each defined as the intersection of n halfplanes, compute a line s such that $P(H_1)$ and $P(H_2)$ belong to different halfplanes with respect to s.*



First notice, as with the previous problems, that the two restrictions of the problem statement are, in fact, unnecessary, because it is possible to check in linear time whether or not $P(H_1)$, $P(H_2)$ and $P(H_1) \cap P(H_2) = P(H_1 \cup H_2)$ are empty, using Megiddo's algorithm for linear programming [17].

1. *Decide which chains are relevant to the problem.* Determine the relative positions of $P = P(H_1)$ and $Q = P(H_2)$ by using the leftmost and the rightmost points of both P and Q:
   - If the vertical strips containing the two polygons are disjoint, then P and Q are separable by any vertical line located between the two strips and the problem is solved.
   - If the two vertical strips containing the two polygons are not disjoint, find the polygon that lies above the other in the common vertical strip. Suppose that Q lies above P. In this case, P will be below any separating line and Q will be above it. Any line separating the upper chain of P from the lower chain of Q will be a separating line for the polygons.
2. *Decide if the solution slope is equal to, bigger or smaller than a given slope m.*

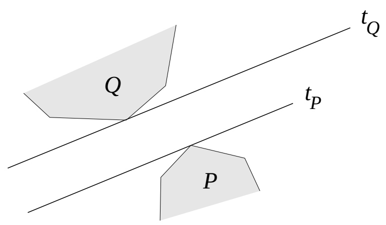


Figure 9: If Q is above P and t_Q is above t_P, the strip $t_P t_Q$ separates the polygons.

- If t_Q and t_P coincide or if t_Q lies above t_P (Figure 9), a separating strip has been found, and the problem is solved.
- If t_Q lies below t_P, we have two cases (see Figure 10):
  - If p lies to the left of q, then the separating line has slope smaller than m.
  - If p lies to the right of q, then the separating line has slope bigger than m.

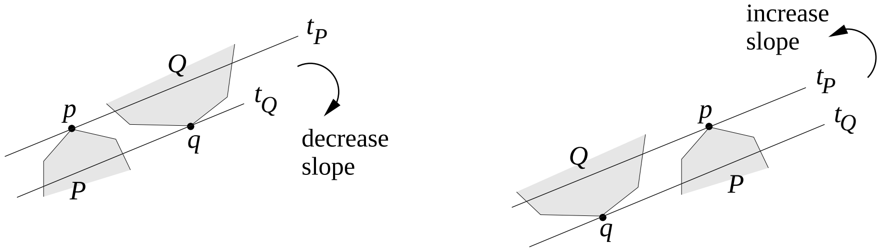
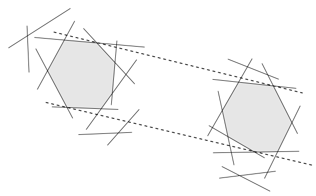


Figure 10: If p is to the left of q, m must be decreased (left). If p is to the right of q, m must be increased (right).

It is obvious that the search algorithm for this problem runs in linear time. Hence, we obtain the following result.

Theorem 7 *The separating line problem for two implicit polygons, each defined as the intersection of n halfplanes, can be solved in optimal $\Theta(n)$ time.*

Common external tangents: *Given two non-empty and disjoint polygons $P(H_1)$ and $P(H_2)$, each defined as the intersection of n halfplanes, compute their common external tangents.*



1. *Decide which chains are relevant to the problem.* This can again be done by comparing the relative positions of the leftmost and rightmost points of P and Q, but a simpler approach is to compute a separating line and, by a change of reference, assume it to be vertical. In this case, one of the tangents will connect the upper chains of P and Q, while the other one will connect the two lower chains.

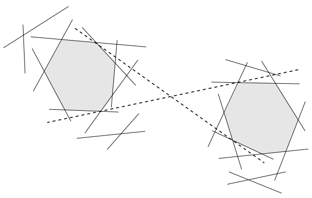
2. *Decide if the solution slope is equal to, bigger or smaller than a given slope m.* Suppose that we are computing the upper common tangent, and that P is left of Q (the two polygons are vertically separated).

- If t_P and t_Q coincide, the tangent has been found.
- If t_P lies above t_Q, then the common tangent has slope smaller than m.
- If t_P is below t_Q, then the common tangent has slope bigger than m.

We thus have the following theorem.

Theorem 8 *The common external tangents problem for two implicit polygons, each defined as the intersection of n halfplanes, can be solved in optimal $\Theta(n)$ time.*

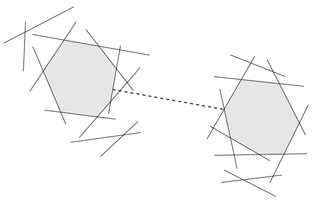
Common Separating tangents: *Given two non-empty and disjoint polygons $P(H_1)$ and $P(H_2)$, each defined as the intersection of n halfplanes, compute their common separating tangents.*



The algorithm that solves this problem is analogous to the previous one, the only difference being the fact that each separating tangent of two vertically separated polygons is determined by the upper chain of one polygon and the lower chain of the other.

Theorem 9 *The common separating tangents problem for two implicit polygons, each defined as the intersection of n halfplanes, can be solved in optimal $\Theta(n)$ time.*

Minimum distance between two polygons: *Given two non-empty polygons $P(H_1)$ and $P(H_2)$, each defined as the intersection of n halfplanes, compute the minimum distance between them.*



The minimum distance between two polygons is zero if they intersect. If they are disjoint, the minimum distance between them is the width of the largest separating strip. The largest separating strip is achieved at points on the chains defined by the separating tangents of the two polygons. The function that assigns to a given slope, the width of the separating strip of that slope is unimodal. Therefore, any local improvement (increase) in the width of the strip is also a global improvement. Hence, this problem can be solved by first computing the separating tangents and then applying the usual prune and search strategy. Suppose again that the two polygons are vertically separated. The search procedure is as follows:

1. *Decide which chains are relevant to the problem.* As described above, this can be done by computing the separating tangents.

2. *Decide if the solution slope is equal to, bigger or smaller than a given slope m.* In this problem, the lines t_P and t_Q always define a separating strip. The prune step is determined by the direction, if any, that increases the width of the strip. Consider the points p and q and the slope h perpendicular to the segment pq (see Figure 11).

    - If $h = m$, then m is the solution slope and $d(p, q)$ is the minimum distance between P and Q (case a).

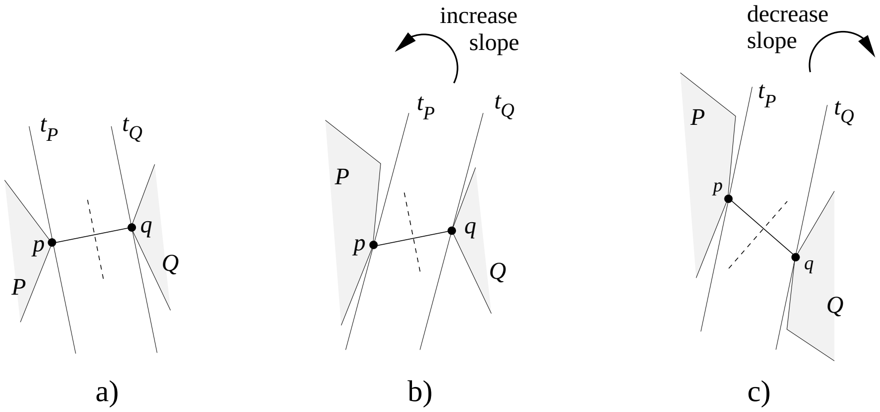


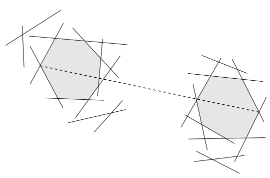
Figure 11: The search procedure when computing the minimum distance between P and Q.

- If $h > m$, then the widest separating strip has slope bigger than m (case b).
- If $h < m$, then the widest separating strip has slope smaller than m (case c).

Since all the decisions are taken locally, we obtain the following result.

Theorem 10 *The minimum distance between two implicit polygons, each defined as the intersection of n halfplanes, can be computed in optimal $\Theta(n)$ time.*

4.2 Superlinear complexity problem



Maximum distance between two polygons: *Given two non-empty polygons $P(H_1)$ and $P(H_2)$, each defined as the intersection of n halfplanes, compute two points, one in $P(H_1)$ and one in $P(H_2)$, that achieve the maximum distance between the two polygons.*

Once the vertices of the polygon are known and in order, the maximum distance between them can be computed in $O(n)$ time using the rotating calipers method [3, 21]. Hence, the problem can be solved in $O(n \log n)$ time.

The lower bound $\Omega(n \log n)$ is proved by reduction from *diameter* (see above). Given a polygon $P = P(H)$, compute its leftmost and rightmost vertices, l and r respectively. Note that the diameter of P is always achieved in two points of P located in opposite chains (we include l and r both in the upper and in the lower chain), because the diameter is the maximum distance between any pair of parallel supporting lines. Let H_1 be the set of all the upper halfplanes of H, plus the lower halfplane defined by the line lr. Let H_2 be the set of all the lower halfplanes of H, plus the upper halfplane defined by the line lr. The maximum distance between $P(H_1)$ and $P(H_2)$ determines the diameter of P. Since the reduction is done in linear time, we obtain the following result:

Theorem 11 *Computing the maximum distance between two implicit polygons, each defined as the intersection of n halfplanes, has complexity $\Theta(n \log n)$.*

5 Implicit Polygons given as Convex Hulls of Points

In this section we present some analogous results for convex polygons that are implicitly defined as the convex hull of sets of points in the plane. Many of the problems discussed so far, for both the one and two polygon cases, have well-known solutions when the input is a set of points. In particular, linear time prune-and-search strategies were used by Kirkpatrick and Seidel [14] to find supporting lines, and by Megiddo and Dyer [7, 17] to compute the smallest enclosing circle. On the other hand, computing the diameter [19], or the width [12, 15] of a set of n points takes $\Theta(n \log n)$ time. In Sections 5.1 and 5.2, we present straightforward extensions of our prune-and-search technique, as well as $\Omega(n \log n)$ lower bound results, for some problems involving one and two implicit polygons, respectively, represented as convex hulls of sets of points.

5.1 One polygon

As in the case of polygons implicitly defined as intersections of halfplanes, we first present the prune step of the algorithms, which is common to all of them. Assume the existence of a search procedure that, given a vertical line $x = a$, decides in linear time whether the solution point lies on the line, to its left or to its right.

1. Compute x_m, the median value of the x-coordinate of all the points in the input set S.
2. Compute the two edges, known as bridges, of the polygon $P = CH(S)$ that intersect the line $x = x_m$.
3. Apply the search procedure to the line $x = x_m$.
4. Prune step:
    - If the solution point lies on the line, the search procedure has found it.
    - If the solution point lies to the left of the line, all the points of S lying to the right can be eliminated, except the endpoints of the bridges.
    - If the solution point lies to the right of the line, all the points of S lying to the left can be eliminated, except the endpoints of the bridges (see Figure 12).

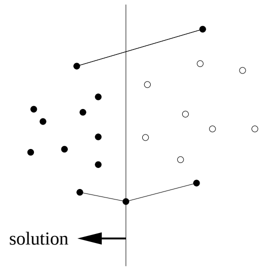
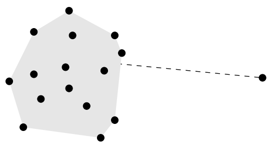


Figure 12: Pruning the white points from the set S.

Assuming that the search step can be done in linear time, this algorithm has complexity $O(n)$, because it eliminates $n/2 - 2$ points of the input by computing a median value, and the two bridges. The bridges can be found in linear time using Kirkpatrick and Seidel's prune-and-search algorithm [14].

Minimum distance to a point: *Given a polygon $P = CH(S)$ defined as the convex hull of a set S of n points, and a point p external to P, compute the minimum distance from P to p.*



Once again, notice that p need not be external to P, since it is easy to check in linear time if p is inside P. Given a vertical line ℓ, let t and b be the top and bottom endpoints of the segment $\ell \cap P$. Consider the rays lying outside the polygon, originating at t and b and perpendicular to the corresponding bridges. They induce a decomposition of the plane that determines the relative position of the point of $P = CH(S)$ that minimizes the distance to p (refer to Figure 13):

- If p lies in the region A, then the minimum distance is achieved at t.
- If p lies in B, then the solution point lies to the right of ℓ.
- If p lies in C, then the minimum distance is achieved at b.
- If p lies in D, then the solution point lies to the left of ℓ.

When either t or b are not points of the original set S, the regions A or C collapse to a ray, but the argument holds.

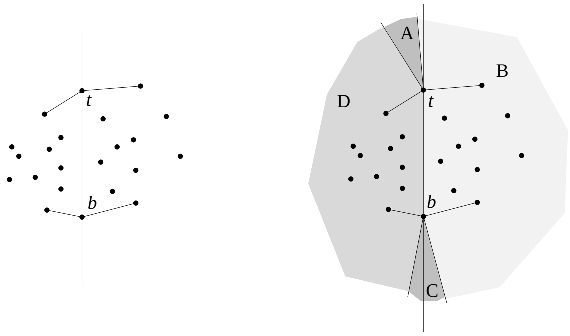
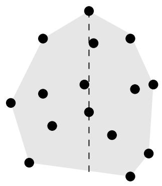


Figure 13: The bridges that intersect the vertical line, and the induced decomposition of the plane.

Hence, we obtain the following result.

Theorem 12 *The minimum distance to a point problem for an implicit polygon defined as the convex hull of n points can be solved in optimal $\Theta(n)$ time.*

Longest vertical chord: *Given a polygon $P = CH(S)$ defined as the convex hull of a set S of n points, compute a longest vertical chord in P.*

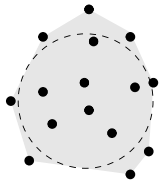


With the same notation as in the previous problem, comparing the angles formed by the bridges at t and b allows us to decide the direction of local increase in the length of the vertical chord tb. The unimodality of the chord length guarantees the correctness of the solution.

As a consequence, we have the following result.

Theorem 13 *The longest vertical chord problem for an implicit polygon defined as the convex hull of n points can be solved in optimal $\Theta(n)$ time.*

Largest enclosed circle: *Given a polygon $P = CH(S)$ defined as the convex hull of a set S of n points, compute the largest circle enclosed in P.*



This problem has complexity $\Theta(n \log n)$. Once the lines supporting the edges of the polygon are known, the problem can be solved in linear time (this is true even if the order of the edges on the boundary is not known) [1, 22]. Hence the problem can be solved in $O(n \log n)$ time by constructing the convex hull and computing the largest enclosed circle.

The lower bound can be proved by reduction from *maximum gap on a circle.* The construction is analogous to Lee and Wu's lower bound result for width computation [15]. Given n points on the first quadrant of the unit circle, $p_1, \ldots, p_n$, compute their diametrically opposite points $q_i = -p_i$, $i = 1, \ldots, n$ (see Figure 14). Add two more points, p_0 and q_0, that are computed in the following way: Let $p_1 = (x_1, y_1)$ the leftmost point among the p_i, and let $p_n = (x_n, y_n)$ be the rightmost point. Then, p_0 is the intersection of the lines tangent to the circle at p_1 and q_n, while $q_0 = -p_0$. That is,

$$p_0 = \left(-\frac{y_n + y_1}{x_n y_1 - x_1 y_n}, \frac{x_n + x_1}{x_n y_1 - x_1 y_n} \right), \qquad q_0 = -p_0.$$

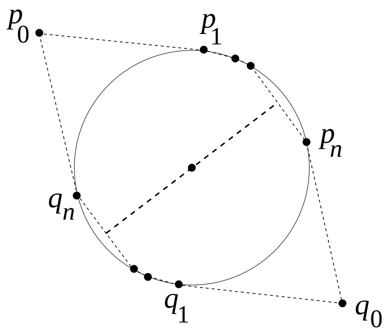


Figure 14: The largest enclosed circle detects the maximum gap.

Apply the *largest enclosed circle* algorithm to this set of points. The circle is tangent to the edges of the convex hull through the points p_i, p_j (and q_i, q_j) that define the maximum gap in $p_1, \ldots, p_n$. Thus we have proved the following

Theorem 14 *The largest enclosed circle problem for an implicit polygon defined as the convex hull of n points has complexity $\Theta(n \log n)$.*

5.2 Two polygons

Again, the algorithms can be presented in such way that only the search step is different for each problem. Assume that the two polygons $CH(S_1)$ and $CH(S_2)$ are vertically separable (checking whether two sets of points are linearly separable and, if so, finding a separating line is done in linear time by linear programming [17]; once a separating line is found, it can be made vertical by a change of reference). Let P be the polygon lying to the left of the separating vertical line, and Q the polygon lying to its right. The algorithm is as follows:

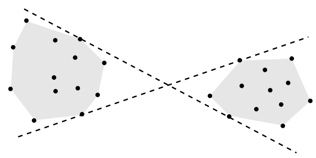
1. Pair up the points in P. Pair up the points in Q.
2. Depending on the problem, the search procedure will orient the segments formed by the pairs. Each endpoint of the segment will be designated as the head or the tail, so that the orientation goes from the tail to the head. In addition, some segment slopes will be declared as feasible, and the rest as infeasible.
3. The first prune step: for each infeasible slope segment, discard the head if the segment is in P, and discard the tail if the segment is in Q.
4. Compute the median slope, m, of the remaining segments, all of which have feasible slopes.
5. The search procedure will decide whether the solution slope is equal to, bigger or smaller than m.
6. The second prune step:
   - If the solution slope is equal to m, the search procedure must have found the solution.
   - If the solution slope is bigger than m, then for each segment having slope smaller than or equal to m, one endpoint may be discarded, namely the tail if the segment belongs to P, or the head if it belongs to Q.
   - If the solution slope is smaller than m, then for each segment having slope bigger than, or equal to m, one endpoint may be discarded, namely the head if the segment belongs to P, or the tail if it belongs to Q.

Notice that at each iteration the algorithm eliminates a half of the endpoints of segments with infeasible slopes, and a fourth of the endpoints of segments with feasible slopes. This is done by pairing up the points and computing a median value. Hence, the algorithm is linear if the search procedure is linear. The search procedure has two steps:

1. Orienting the segments and identifying feasible and infeasible slopes.
2. Given a feasible slope m, deciding whether the solution slope is equal to, bigger, or smaller than m. In the first case, the solution to the problem must be found as well.

We provide such a procedure, which runs in linear time, for our problems.

Common separating tangents: *Compute the common separating tangents of two polygons $P = CH(S_1)$ and $Q = CH(S_2)$, each defined as the convex hull of a set of n points.*



Recall that P and Q are supposed to be separated by a vertical line that has P to its left and Q to its right. We present the search procedure for computing the separating tangent that has P above it and Q below it, as shown in Figure 15.

1. *Orient segments and determine feasible slopes.* All non-vertical slopes are feasible slopes, and they get oriented left to right. The only infeasible slope is the vertical, and it gets oriented bottom to top.

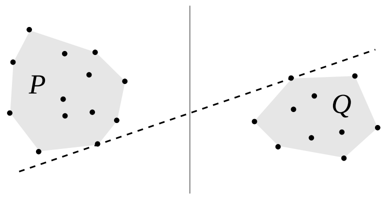


Figure 15: Assuming that P is to the left of Q and that they are vertically separated, one of their separating tangents has P above it and Q below it.

2. *Decide if the solution slope is equal to, bigger or smaller than a given feasible slope m.* Compute t_P, the line tangent to P with slope m that has P on its left (recall that any slope is oriented now). Compute t_Q, the line tangent to Q with slope m that has Q to its right (see Figure 16).
   - If $t_P = t_Q$, then m is the solution slope and $t_P = t_Q$ is the separating tangent.
   - If t_P is above t_Q, then the solution slope is smaller than m (Figure 16, left).
   - If t_P is below t_Q, then the solution slope is bigger than m (Figure 16, right).

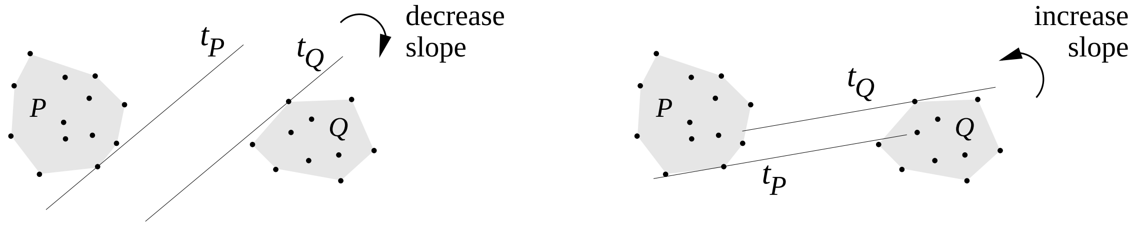
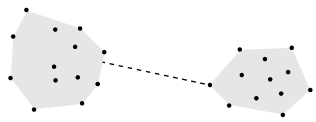


Figure 16: Deciding if the slope should be increased or decreased.

Computing the two tangents is easily done in linear time. Hence we obtain the following result.

Theorem 15 *The common separating tangents problem for two implicit polygons, each defined as the convex hull of n points, can be solved in optimal $\Theta(n)$ time.*

Minimum distance between polygons: *Compute the minimum distance between two polygons $P = CH(S_1)$ and $Q = CH(S_2)$, each defined as the convex hull of a set of n points.*



Recall that P and Q are supposed to be separated by a vertical line that has P to its left and Q to its right. Also recall that the minimum distance between P and Q is given by the maximum width separating strip of P and Q. The algorithm is as follows:

1. *Orient segments and determine feasible slopes.*
   - First compute the separating tangents. They determine two wedges, one containing the two sets of points, the other free of points.
   - Feasible slopes are those that lie in the range defined by the wedge free of points. All feasible slopes get oriented such that any separating line with such a slope will have P to its left and Q to its right. Figure 17 (left) illustrates the orientation for segments having feasible slopes.

- Infeasible slopes are those that lie in the range defined by the wedge that contains the points. All infeasible slopes get oriented such that any line with such a slope will intersect the part of the wedge containing Q before (in the oriented order) the part of the wedge containing P. Figure 17 (right) illustrates the orientation for segments having infeasible slopes.

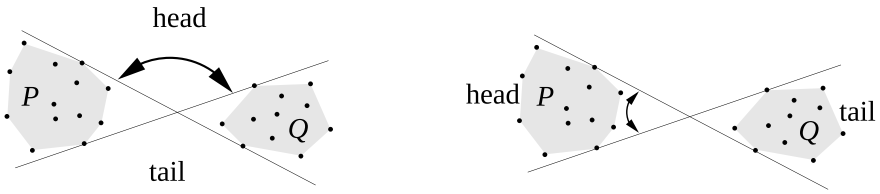


Figure 17: Head and tail of the segments.

2. *Decide if the solution slope is equal to, bigger or smaller than a given feasible slope m.* Compute t_P, the line tangent to P with slope m that has P on its left (recall that any slope is oriented now). Call its supporting point p. Compute t_Q, the line tangent to Q with slope m that has Q to its right. Call its supporting point q. The strip $t_P t_Q$ separates P and Q. We must determine the direction that increases the width of the strip (see Figure 18). Since the strip width is a unimodal function of the angle of the strip, the direction of increase may be determined locally. Let h be the slope perpendicular to the segment pq.

   - If $h = m$, then m is the solution slope and $d(p, q)$ is the minimum distance between P and Q.
   - If $h > m$, then the solution slope is bigger than m.
   - If $h < m$, then the solution slope is smaller than m.

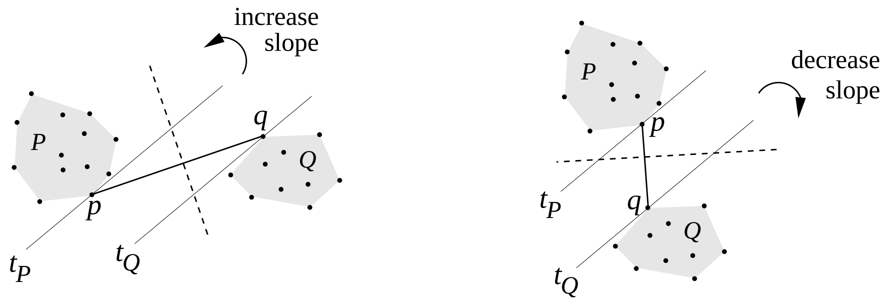
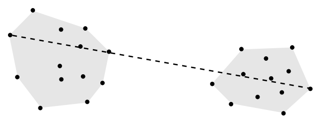


Figure 18: Deciding if the slope should be increased or decreased.

Given that the separating tangents can be computed in linear time, and so can the tangents t_P and t_Q, the search procedure for this problem is linear, and we obtain the following result.

Theorem 16 *The minimum distance between two implicit polygons, each defined as the convex hull of n points, can be computed in optimal $\Theta(n)$ time.*

Maximum distance between two polygons: *Given two polygons $CH(S_1)$ and $CH(S_2)$, each defined as the convex hull of a set of n points, compute the maximum distance between them.*



This problem has complexity $\Theta(n \log n)$. As mentioned earlier, once the edges of the polygon are known, the problem can be solved in linear time, using the rotating calipers method [3, 21]. Hence, the problem can be solved in $O(n \log n)$ time.

The lower bound $\Omega(n \log n)$ is proved by reduction from *diameter*. The construction is as follows. Let S be a set of points. Compute l and r, its leftmost and rightmost points. Let $S_1 \subset S$ be all the points that lie above or on the line lr. Let $S_2 \subset S$ be all the points that lie below or on the line lr. Apply the *maximum distance between two polygons* algorithm to S_1 and S_2. The result is the diameter of S. The correctness of the reduction is based on the fact that S_1 and S_2 share two points, l and r, that form an antipodal pair of S. Thus we have the following:

Theorem 17 *Computing the maximum distance between two implicit polygons, each defined as the convex hull of n points, has complexity $\Theta(n \log n)$.*

6 Concluding Remarks

This paper demonstrates the effectiveness of the prune-and-search strategy in solving numerous problems involving polygons that are represented implicitly. The implicit representation of a convex polygon as the intersection of a set of halfplanes, or as the convex hull of a set of points, is a natural choice in many different applications.

Our optimal linear-time algorithms, which are extremely easy to implement, requiring no preprocessing and no special data structures, are very useful in a practical setting as well. We ran some experiments on some of the algorithms, namely those to compute supporting lines, the minimum distance to a point, and the longest vertical chord. These algorithms were implemented in `C++`, using the CGAL (`www.cs.ruu.nl/CGAL`) and LEDA (`www.mpi-sb.mpg.de/LEDA`) libraries on a 266 MHz, Pentium II running Linux. The results for computing the minimum distance to a point are shown in Table 1. Recall that this algorithm must perform the test to check if the point is in the polygon, and also compute the supporting lines before finding the minimum distance. For each input set of halfplanes, the minimum distance to 5 different points was computed. The table shows the size of the input, the average run-times and number of times the search step was executed. Each input was chosen randomly from a data set of 20,000 halfplanes, which were generated manually to ensure a non-empty intersection. The experiments were run with two different methods of finding the median: the deterministic linear time algorithm [4], and the expected linear time algorithm [11]. As seen from the data, the randomized median finding algorithm gives better results and is extremely fast, taking less than 5 seconds to find the minimum distance to a point from a polygon represented implicitly by 19,200 halfplanes.

In addition to the linear time algorithms, there are problems for which the prune-and-search strategy does not work, and for which $\Omega(n \log n)$ lower bounds are known, or are shown in this paper. Table 2 summarizes our results for problems involving one polygon, and Table 3 contains the results for problems involving two polygons. For the sake of completeness, these tables also include some results that are already known, which are indicated by including references where the results can be found.

A natural open problem is to obtain a characterization of the threshold between problems that are solvable in linear time, and those that have an $\Omega(n \log n)$ lower bound. We would like to observe that in all the cases that we studied, the prune-and-search strategy is effective in obtaining an optimal linear-time algorithm when the problem involves finding the minimum or maximum value of a unimodal function, either on the boundary of the implicit polygon (in the one polygon case), or on the slope of a separating strip (in the two polygons case). For example, computing

Size	Deterministic Median		Random Median	
	time(sec.)	*#searches*	*time(sec.)*	*#searches*
75	0.034	7.2	0.036	8.6
150	0.056	11.2	0.046	10.8
300	0.104	15.0	0.084	14.6
600	0.194	15.2	0.164	15.6
1200	0.348	16.2	0.302	17.8
2400	0.7	18.4	0.576	19.0
4800	1.366	25.2	1.194	24.4
9600	2.784	23.4	2.402	26.6
19200	5.784	27.0	4.936	27.2

Table 1: Average running times.

the minimum distance to a point from a polygon falls in the first category, and computing the minimum distance between two polygons falls in the second.

Acknowledgments We are grateful to Valenti Boira, of the Facultat d'Informàtica de Barcelona, at the Universitat Politècnica de Catalunya for the implementation results in Table 1.

References

[1] A. Aggarwal, L. J. Guibas, J. Saxe and P. Shor. A linear time algorithm for computing the Voronoi diagram of a convex polygon. *Proc. 19th Annu. ACM Sympos. Theory Comput.*, 39-45, 1987.

[2] M. Ben-Or. Lower bounds for algebraic computation trees. *Proc. 15th Annual Symposium on Theory of Computing*, 80–86, 1983.

[3] B. Bhattacharya and G. Toussaint. Efficient algorithms for computing the maximum distance between two finite planar sets. *J. Algorithms*, 4:121-136, 1983.

[4] M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *J. Comput. and Syst. Sci.*, 7:448–461, 1973.

[5] S. W. Dharmadhikari and K. Jogdeo. A characterization of convexity and central symmetry for planar polygonal sets. *Israel J. Math.*, 15:356–366, 1973.

[6] M. E. Dyer. Linear time algorithms for two- and three-variable linear programs. *SIAM J. Comput.*, 13:31–45, 1984.

[7] M. E. Dyer. On a multidimensional search technique and its application to the Euclidean one-centre problem. *SIAM J. Comput.*, 15:725–738, 1986.

[8] A. Frank, P. Haunold, W. Kuhn and G. Kuper. Representation of geometric objects as set of inequalities. *Proc. 12th European Workshop on Computational Geometry*, 51–56, 1996.

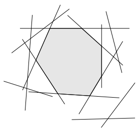
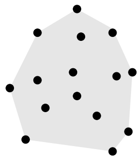
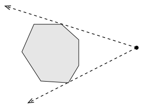
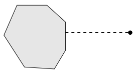
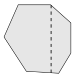
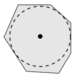
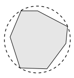
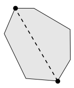
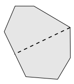
	Intersection of n halfplanes 	*Convex hull of n points* 
 Supporting lines	$\Theta(n)$	$\Theta(n)$
 Min distance to a point	$\Theta(n)$	$\Theta(n)$
 Longest vertical chord	$\Theta(n)$	$\Theta(n)$
 Largest enclosed circle	$\Theta(n)$ [22]	$\Theta(n \log n)$
 Smallest enclosing circle	$\Theta(n \log n)$	$\Theta(n)$ [7, 17]
 Diameter	$\Theta(n \log n)$	$\Theta(n \log n)$ [19]
 Width	$\Theta(n \log n)$	$\Theta(n \log n)$ [12, 15]

Table 2: Summary of results for one polygon.

[9] P. Haunold, A. Frank, S. Grumbach, G. Kuper and Z. Lacroix. Geometric objects represented by inequalities. *Proc. Angewandte Geographische Informationsverarbeitung IX (AGIT)*, Salzburger Geographische Materialen, Vol. Heft 26, 77–86, 1997.

[10] P. Haunold, S. Grumbach, G. Kuper and Z. Lacroix. Linear constraints: geometric objects represented by inequalities. *Proc. International Conference COSIT '97, Spatial Information Theory, A Theoretical Basis for GIS*, 429–440, 1997.

[11] C. A. R. Hoare. Algorithm 63 (partition) and algorithm 65 (find). *Communications of the ACM*, 4(7):321–322, 1961.

[12] M. E. Houle and G. T. Toussaint. Computing the width of a set. *IEEE Trans. Pattern Anal. Mach. Intell.*, 10:761–765, 1988.

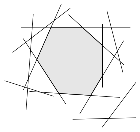
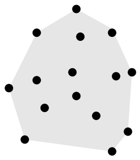
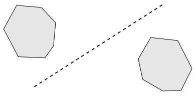
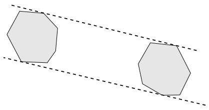
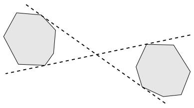
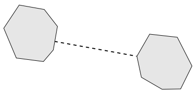
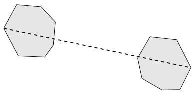
		Intersection of n halfplanes 	*Convex hull of n points* 
	Separating line	$\Theta(n)$	$\Theta(n)$ [17]
	Common external tangents	$\Theta(n)$	$\Theta(n)$ [14]
	Common separating tangents	$\Theta(n)$	$\Theta(n)$
	Min distance b/w polygons	$\Theta(n)$	$\Theta(n)$
	Max distance b/w polygons	$\Theta(n \log n)$	$\Theta(n \log n)$

Table 3: Summary of results for two polygons.

[13] F. Hurtado, V. Sacristán and G. Toussaint. Some constrained minimax and maximin location problems. *Studies in Locational Analysis*, to appear.

[14] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm. *SIAM J. Comput.*, 15:287–299, 1986.

[15] D. T. Lee and Y. F. Wu. Geometric complexity of some location problems. *Algorithmica*, 1:193-211, 1986.

[16] Liang, Y. and B. Barsky. A new concept and method for line clipping. *ACM Transactions on Graphics*, 3(1):1-22, 1984.

[17] N. Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM J. Comput.*, 12:759–776, 1983.

[18] N. Megiddo. Linear programming in linear time when the dimension is fixed. *J. ACM*, 31:114–127, 1984.

[19] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, NY, 1985.

[20] G. Toussaint. Solving geometric problems with rotating calipers. MELECON, Greece, 1983.

[21] G. Toussaint and M.A.McAlear A simple $O(n \log n)$ algorithm for finding the maximum distance between two finite planar sets. *Pattern Recogn. Lett.*, 1:21-24, 1982.

[22] E. Welzl. LP with Small d - Algorithms and Applications. Manuscript, 1994.