GEOMETRIC INTERSECTION PROBLEMS †

Michael Ian Shamos
Departments of Computer Science and Mathematics
Carnegie-Mellon University
Pittsburgh, PA 15213

and

Dan Hoey
Department of Computer Science
Yale University
New Haven, CT 06520

Abstract

We develop optimal algorithms for forming the intersection of geometric objects in the plane and apply them to such diverse problems as linear programming, hidden-line elimination, and wire layout. Given N line segments in the plane, finding all intersecting pairs requires $O(N^2)$ time. We give an O(N log N) algorithm to determine whether any two intersect and use it to detect whether two simple plane polygons intersect. We employ an O(N log N) algorithm for finding the common intersection of N half-planes to show that the Simplex method is not optimal. The emphasis throughout is on obtaining upper and lower bounds and relating these results to other problems in computational geometry.

I. Introduction

The task of computational geometry is to relate the geometric properties of figures to the complexity of algorithms that manipulate them. One important benefit of such a study is the development of efficient algorithms for applications in which geometry plays a major role, such as computer graphics and operations research. In an earlier paper[1] on closest-point problems, we showed how a number of seemingly unrelated questions could be answered by a single algorithm based on a common underlying geometric theme. In this paper we will again try to be economical and exploit one unifying idea to solve a variety of problems.

Geometric applications frequently involve finding the intersection of objects. A plane polygon is simple if and only if no two of its edges intersect, a printed circuit can be realized without crossovers if no two conductors cross, and a pattern can be cut from a single piece of stock if no two parts of the layout overlap. In computer graphics, an object to be displayed obscures another if their projections on the viewing plane intersect. The importance of efficient algorithms for these problems is becoming more and more apparent as industrial applications grow increasingly ambitious: a single integrated circuit may contain tens of thousands of components, a complicated scene for graphic display can involve a hundred thousand vectors, and data bases for architectural design must be able to store upwards of one million distinct elements. For these purposes even quadratic algorithms are impractical.

The common theme of the above problems is that they can all be solved quickly if a fast algorithm is available for detecting whether any two of N line segments in the plane intersect. The first part of this paper deals with the development of such an algorithm.

In some cases we must form the common intersection of N objects. The convex hull of a figure is the intersection of its supporting half-spaces. The kernel of a polygon (the locus of points from which all points of the polygon are visible) is the intersection of the half-planes determined by its sides[2]. Determining whether N linear inequalities are simultaneously satisfiable and linear programming are further examples. When the objects are convex, forming their common intersection is particularly easy and we borrow an algorithm for the intersection of N half-planes to solve a number of problems of this type. The emphasis throughout is on relating these results to basic questions in geometric complexity.

†This work was supported in part by the Office of Naval Research under contracts N00014-75-C-0450 and N00014-76-C-0829. For a portion of the time during which this research was in progress the first author held an IBM Fellowship at Yale University.

II. Problems and Methods

In this section we will pose four fundamental intersection problems and outline the methods that will be used to obtain upper and lower bounds on the time required for their solution.

I1. *Given N objects, do any two intersect ?*

Lower bounds: In general, we will obtain O(N log N) lower bounds on this problem by showing that either sorting or element uniqueness is reducible to it. The element uniqueness problem is to determine whether any two of N real numbers are equal and the lower bound is proved in a model allowing comparisons between linear functions of the inputs[3].

Upper bounds: Suppose that two of the objects can be checked for intersection in constant time. Then determining whether any two of the N given objects intersect can be done naively in $O(N^2)$ time. In fact, if the only operation allowed is pairwise checking for intersection, no better algorithm is possible. This is analogous to the problem, described by Reingold[4], of determining whether two sets A and B, each containing N real numbers, are equal. If comparisons are not allowed between two elements of A or between two elements of B, then we are unable to take advantage of the fact that A and B can be totally ordered, and $O(N^2)$ comparisons are required. On the other hand, if the sets can be ordered, then O(N log N) comparisons suffice. In the intersection problem, we will reduce the running time to O(N log N) by defining an order relation on the objects and using a height-balanced tree to manipulate them.

I2. *Given N objects, how many pairs intersect ?*

Lower bounds: Since I2 answers the question posed in I1, the same lower bounds apply.

Upper bounds: Even though all pairs of objects may intersect, it may be possible to count the number of intersections without explicitly listing them. A similar situation arises in counting the number of inversions of a permutation, which can be accomplished with O(N log N) comparisons even if there are $O(N^2)$ inversions.

I3. *Given N objects, find all intersecting pairs.*

Lower bounds: If k pairs intersect, then a lower bound of $O(\max(k, N \log N))$ follows from I1.

Upper bounds: The outstanding question is whether $O(\max(k, N \log N))$ time suffices. This problem will not be treated in any detail but is included for the sake of completeness.

I4. *Given N objects, form their common intersection.*

Lower bounds: If the objects are half-planes, then an O(N log N) lower bound can be obtained by showing that any such algorithm can be used to sort. This result can be extended to circles and a variety of other objects.

Upper bounds: We make use of the associative property of the intersection operation to rearrange the order in which the operation is applied so that a divide-and-conquer strategy may be used. In this way we will obtain O(N log N) algorithms.

The model of computation we shall assume is the random-access machine (RAM)[5], generalized to allow real arithmetic and the storage of real numbers. If the reader is troubled by this generalization she may prefer to work with a traditional RAM operating on objects whose coordinates are integral or rational, but our lower bounds will then not apply.

The question of how objects are to be represented arises frequently in computational geometry but causes no difficulty. We will restrict our attention to the Cartesian plane but any other coordinate system in which it is possible to transform a point to Cartesian coordinates in constant time may be imposed without affecting the asymptotic order of our results. Even problem-dependent systems such as center-of-mass coordinates may be used without fear because N points in such a system can be transformed to rectangular coordinates in linear time. The authors are unaware of any coordinate system for which this is not the case.

A line segment is represented by the coordinates of its endpoints. A polygon is represented by a list of its vertices, in the order in which they occur as the boundary is traversed. Note that while a polygon is a plane figure consisting of a polygonal boundary and its interior, we represent it unambiguously by specifying only the boundary. A polygon is simple if no non-consecutive segments comprising its boundary intersect. (Checking for this property is a non-trivial matter.) A point lies above a line segment if it lies above the line containing the segment. It is below if it lies below that line. Some special cases arise, which we may settle by definition: If the line segment s is vertical and the point p is not collinear with it, we say by convention that p is above s if it lies in the right half-plane determined by s. If s is vertical and p is collinear, then p lies above s only if it lies above all points of s. Similarly, if s is horizontal and p is collinear with it, then by conven-

tion p lies above s if it is entirely to the right of s. Of course, it is possible that a point lies neither above or below a segment, but on it. The preceding definition is an example of the painful and boring attention to detail that must be paid if our theorems are to be strictly correct. In this paper we shall largely ignore such special cases with the promise that no results will be substantially affected. It seems pointless to fill these pages with intricate definitions that do not contribute materially to the questions at hand. Unfortunately, implementors of the algorithms will be compelled to supply the details.

III. Intersection of Line Segments

Suppose we are given N intervals on the real line and are asked whether any two overlap. This can be answered easily in $O(N^2)$ time by inspecting all pairs of intervals. We may make use, however, of the fact that there is a natural order relation on intervals: either A is to the left of B, A is to the right of B, or they overlap. Let us sort the endpoints of the intervals in O(N log N) time and set up a (very simple) data structure that can contain just one object: the "current" interval. We now scan the endpoints from left to right, inserting an interval into the structure when its left endpoint is encountered and deleting it when its right endpoint is encountered. If an attempt is ever made to insert an interval when the data structure is non-empty, an overlap has occurred; otherwise, no overlap exists. Since there are 2N endpoints and the processing of each endpoint (after the sort) requires only constant time, the total time required is O(N log N). No asymptotic improvement is possible:

Theorem 1. *O(N log N) is a lower bound on the time required to determine whether N intervals on a line are pairwise disjoint.*

Proof: Let all the intervals degenerate into single points and if there is no overlap then all the points are distinct; the result follows from the O(N log N) lower bound for the element uniqueness problem[3]. □

In two dimensions the lower bound still applies, but there is no geometrically-induced total order on line segments in the plane and the above algorithm fails. We are obliged to define a new order relation and employ a more complicated data structure. To clarify the discussion, from this point on we assume that no segment is vertical and that no three segments meet in a single point. These assumptions do not change the asymptotic running times of the algorithms.

Consider two non-intersecting line segments A and B in the plane. We say that A and B are comparable if there exists a vertical line, say at x, that intersects both of them. In case A and B are comparable, we shall say that A is above B at x, written $A \underset{x}{\uparrow} B$, if the intersection of A with the vertical line lies above the intersection of B with that line. In Figure 1, we have the following relations among the line segments A, B, C and D: $B \underset{u}{\uparrow} D$, $A \underset{v}{\uparrow} B$, $B \underset{v}{\uparrow} D$, $A \underset{v}{\uparrow} D$, and C is not comparable with any other segment.

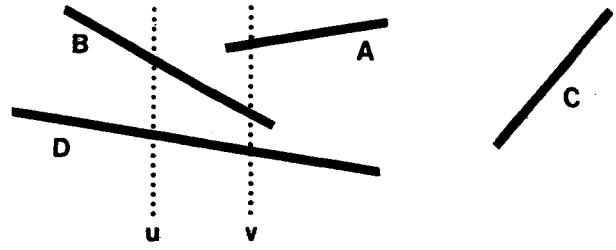

Figure 1. An order relation between line segments.

Note that the relation $\underset{x}{\uparrow}$ is transitive and hence defines a total order. Imagine sweeping a vertical line from left to right across a field of line segments. At each point the segments intersected by the line are totally ordered by $\underset{x}{\uparrow}$. When can this ordering change? Certainly when the moving line L encounters the left endpoint of a new segment, that segment must enter the total order. Also, when L passes to the right of the right endpoint of a segment, that segment must leave the total order. The case of interest to us, however, is when L passes through the intersection of two segments, for then the positions of the two segments in the total order are reversed. It is important to realize that if segments A and B intersect, then there is some x for which A and B are consecutive in the total order $\underset{x}{\uparrow}$. If we are able to maintain the total order as x moves along, we will certainly detect all intersections. However, since there may be $O(N^2)$ intersection points, it must take at least $O(N^2)$ operations to do the updating. Below we describe an algorithm which detects the presence of an intersection if there exists one, but does not go through the expense of keeping complete information about the ordering.

The algorithm proceeds in a manner similar to the one-dimensional algorithm given earlier. We will sort the 2N segment endpoints and march from left to right, inserting a segment in the data structure when its left endpoint is encountered and deleting a segment when its right endpoint is passed, checking segments for intersection when they become consecutive in the total order. This will require a structure T to maintain the ordering, on which we can perform the following operations:

1. INSERT(s,T) inserts the segment s into the total order maintained by T.
2. DELETE(s,T) deletes segment s.
3. ABOVE(s,T) returns the name of the segment immediately above s in T.
4. BELOW(s,T) returns the name of the segment immediately below s.

The algorithm will be given in pseudo-Algol and is based on an earlier, unpublished version[6].

Algorithm 1. (Intersection of line segments)

```
SORT the endpoints lexicographically on x and y so
   that POINT[1] is leftmost and POINT[2N] is
   rightmost.
FOR I:=1 UNTIL 2N DO BEGIN
   P:=POINT[I];   Let S be the segment of which p
                  is an endpoint.
   IF P is the left endpoint of S THEN BEGIN
      INSERT(S,T);
      A:=ABOVE(S,T);
      B:=BELOW(S,T);
      IF  A intersects S  THEN RETURN(A,S);
      IF  B intersects S  THEN RETURN(B,S);
                                      END;
   ELSE (P is the right endpoint of S) BEGIN;
      A:=ABOVE(S,T);
      B:=BELOW(S,T);
      IF  A intersects B  THEN RETURN(A,B);
      DELETE(S,T);
                                      END;
                END;
```

If an intersection exists, Algorithm 1 returns the names of two intersecting segments; otherwise it returns nothing.

Theorem 2. *Algorithm 1 finds an intersection if one exists.*

Proof: We show that the algorithm successfully finds the leftmost intersection point. Since Algorithm 1 only reports an intersection if it finds one, it will never falsely claim the existence of an intersection, and we may turn our attention to the possibility that an intersection exists but remains undetected. If an intersection exists, then there is a leftmost one at point p. If several leftmost intersections exist, let p be the one with least y-coordinate. The half-plane to the left of p is free of intersections. (Figure 2) As a vertical line moves from $-\infty$ to p, the total order being maintained in T only changes when segments are inserted or deleted. Since no intersections occur before p, the order represented by T is correct at all points to the left of p. Let A and B be the segments

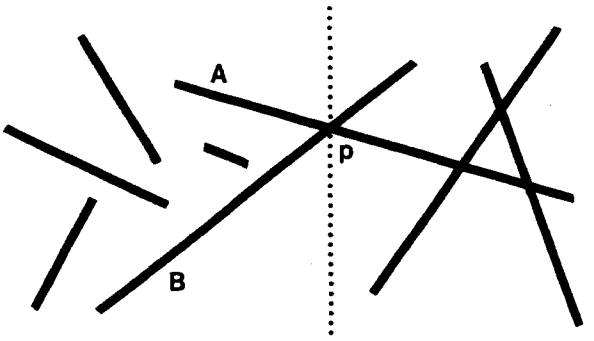

Figure 2. Finding the leftmost intersection.

whose intersection is p. Consider the leftmost point at which A and B become consecutive in the total order. This can occur in only two ways:

1. Either A or B is inserted and the other is either immediately ABOVE or BELOW it in T. This case is detected by the first IF block in Algorithm 1.
2. A and B are already in T and an intervening segment is deleted, leaving A and B consecutive. This case is successfully detected by the ELSE block in algorithm 1.

Thus in either event the intersection is found. □

Even though Algorithm 1 is simple, it has some curious properties. First of all, even though the leftmost intersection is found, it is not necessarily the first intersection to be found. Second, since the algorithm only makes O(N) intersection tests, it may fail to find some intersections. The reader may occupy some idle moments constructing illustrative examples. Figure 3 shows how Algorithm 1 operates in finding an intersection. There are five segments, labeled A, B, C, D and E, in order by left endpoint.

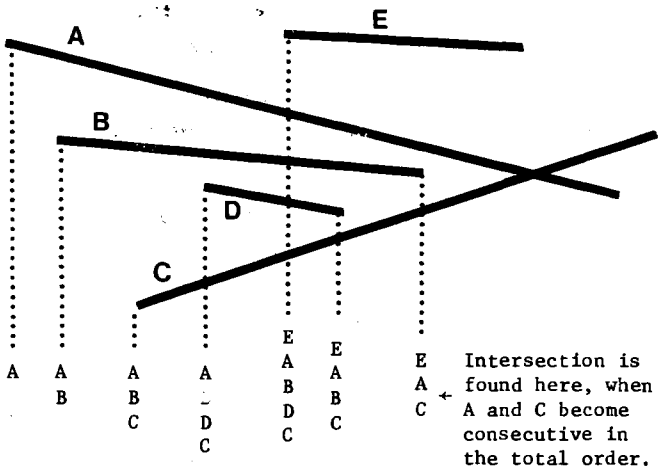

Figure 3. Operation of Algorithm 1.

Theorem 3. *Whether any two of N line segments in the plane intersect can be determined in O(N log N) time.*

Proof: We show that Algorithm 1 can be implemented in O(N log N) time. The sorting of the 2N endpoints can

be accomplished in O(N log N) time. Using a balanced tree scheme we may implement the operations INSERT, DELETE, ABOVE, and BELOW so that each can be performed in O(log N) time. The FOR-loop of Algorithm 1 contains a constant number of these operations and the loop is executed 2N times, hence O(N log N) time suffices. □

IV. Applications of the Intersection Algorithm.

Algorithm 1 is of direct use in testing wire and integrated circuit layouts. Normally, circuit design is performed by human engineers or heuristic design programs,neither of which is guaranteed to produce circuit with no intersections. The cost of testing each conductor against every other, even mechanically, is prohibitive,but Algorithm 1 performs an equivalent test quite rapidly. Likewise, we can determine in O(N log N) time whether a straight-line embedding of a planar graph is proper (that is, has no intersecting edges).

Detecting whether or not a polygon is simple is of some importance because some geometric algorithms can operate much more quickly on simple polygons than on non-simple ones. For example, the convex hull of a simple N-gon can be found in O(N) time [7] [8] but forming the hull of a non-simple polygon requires O(N log N) time. A direct application of Theorem 3 gives

Theorem 4. *Whether a plane N-gon is simple or not can be determined in O(N log N) time.* □

Computing the intersection of two polygons is a fundamental operation in the hidden-line problem and in certain optimization problems. The intersection of two convex N-gons can be found in O(N) time, but the intersection of general (non-convex) polygons may consist of $O(N^2)$ disjoint parts containing a total of $O(N^2)$ edges, so we have a trivial quadratic lower bound on the time needed to intersect such polygons. If we abandon the requirement of finding the entire intersection and are satisfied with knowing merely whether the intersection is non-empty, the situation is brighter:

Theorem 5. *Whether two simple plane N-gons intersect can be determined in O(N log N) time.*

Proof: Let the polygons be A and B. If they intersect, then either A contains B, B contains A, or some edge of A intersects an edge of B. Since A and B are simple, any edge intersection that exists must be between edges of different polygons. Execute Algorithm 1 on the set of 2N line segments comprising the edges of A and B. If an intersection is found, then A and B intersect and we are done. If no intersection is found, then test any vertex of A to see whether it is interior to B. This can be done, for an arbitrary polygon, in O(N) time[8]. If a vertex of A is interior to B and there are no edge intersections, then A itself lies wholly within B (Jordan Curve Theorem). Perform an identical test to see whether B lies within A. □

Even though $O(N^2)$ time is required to form the intersection of polygons in the worst case, it is wasteful to spend this much time if the polygons are in fact disjoint. The algorithm of Theorem 5 may be used as a pre-screen to determine whether the more expensive algorithm will be necessary.

A useful generalization is the problem of determining whether any two of N k-gons intersect. We shall treat the convex case first. In earlier times (i.e. more than two years ago), when no algorithm for intersecting two convex k-gons in less than $O(k^2)$ time was known and no algorithm faster than the obvious quadratic one was known for detecting whether any two of N objects intersect, this problem would have required $O(N^2k^2)$ time. We will make use of both improvements to reduce the running time drastically.

The crucial observation is that an order relation can be defined on polygons, not just on line segments. A vertical line intersects the boundary of a convex polygon in at most two points (unless an edge of the polygon is vertical, but this is a minor detail). We say that two non-intersecting convex polygons are comparable if they are both intersected by some vertical line. Given comparable polygons A and B, either A lies above B, B lies above A, or one contains the other. Furthermore, the intersection points of A and B with the vertical line L partitions L into intervals which lie either wholly within or entirely outside a polygon. This relationship is illustrated in Figure 4.

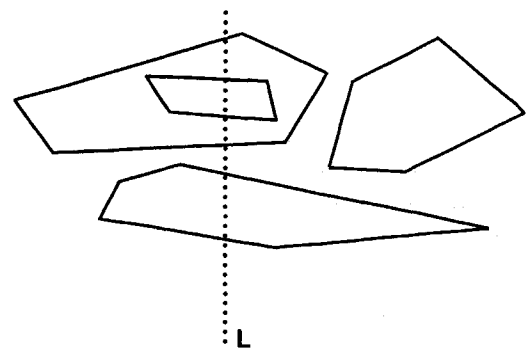

Figure 4. Order relations among convex polygons

We may thus proceed by analogy to Algorithm 1, depending on the fact that, if two polygons intersect, they must at some point be consecutive in the total order. The only complication is that one polygon may contain

another.

Theorem 6. *Whether any two of N convex k-gons intersect can be determined in O(N(k + log N log k)) time.*

Proof: We will first sort the polygons by leftmost and rightmost extreme points, then proceed from left to right, inserting and deleting polygons, and performing intersection tests whenever two polygons become adjacent.

1. Sorting. There are a total of Nk edges in all of the polygons. For each polygon we can find the leftmost and rightmost vertices in O(k) time, for a total of O(Nk) time overall. The result is 2N vertices, which we may sort in O(N log N) time, so this step requires at most O(Nk + N log N) time.
2. Insertions. We insert a polygon when its leftmost endpoint is encountered. To do this, we must be able to tell whether a point lies above, below, or within a convex polygon. This test can be performed on a convex k-gon in O(log k) time by binary search[2], if O(k) preprocessing is allowed. If we use a balanced tree to maintain the order of the polygons, O(log N) such tests will suffice to accomplish an insertion or deletion. Since at most N insertions and N deletions will be made, O(Nk + N log N log k) time is sufficient.
3. Inclusion and intersection tests. When a polygon is inserted, it must be tested against its immediate neighbors for inclusion or intersection. As we have seen before, each such test can be done in O(k) time. Similarly, when a polygon is deleted, the two polygons that become consecutive must be tested. There are at most O(N) tests overall, for a cost of O(Nk) operations.

The desired result is obtained by adding the times required by steps 1, 2, and 3. The correctness of this algorithm can be proved by methods nearly identical to those used in Theorem 2: the leftmost intersection is certain to be found. □

Theorem 7. *Whether any two of N simple (but possibly non-convex) k-gons intersect can be determined in O(Nk log Nk) time.*

Proof: This is a straightforward modification of algorithm 1, using the ideas of Theorem 6. We can sort the endpoints of all the edges of the polygons in O(Nk log Nk) time. Perform the same scan as before, inserting and deleting the edges (not polygons) separately. The reason that we do not attempt to insert whole polygons is that non-convex polygons are not totally ordered by the "above" relation, as can be seen from Figure 5. The region of the plane lying above a segment is either inside some polygon or outside all polygons. As segments are inserted into the data structure, we associate with each segment two pieces of information, UP and DOWN. Each of these is either zero or the number of the polygon to which the segment belongs, zero indicating "outside". If, proceeding upwards from the segment s, we pass out of its polygon, then UP(s) is zero. If, on the other hand, we move into polygon P, then UP(s) = P . On inserting or deleting a segment we perform the same intersection tests as in Algorithm 1, with the following check for inclusion: if a segment is inserted into a region belonging to a different polygon then trouble has occurred. Suppose that segment s, belonging to polygon P, is to be inserted below segment t and above segment u, both belonging to polygon Q. Then either Q contains P entirely, or they intersect. Thus Nk insertions and Nk deletions will be made and O(Nk) segment intersection tests performed. The height-balanced tree never holds more than Nk segments, so O(Nk log Nk) time suffices. □

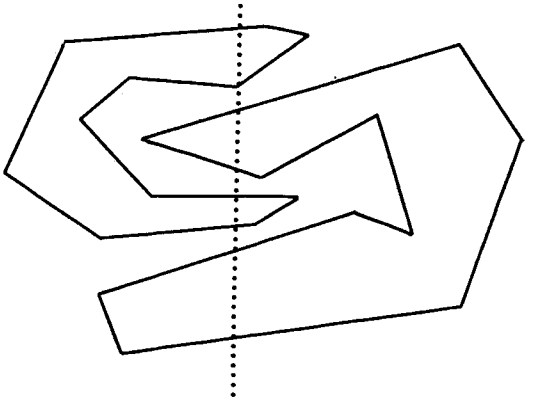

Figure 5. Non-convex polygons in the plane.

Theorem 8. *Whether any two of N circles in the plane intersect can be determined in O(N log N) time, and this is optimal.*

Proof: Non-intersecting circles are totally ordered by the "above" relation and Algorithm 1 can be used with one modification. A circle is inserted when its leftmost point is encountered and deleted when its rightmost point is encountered. As usual, circles which become consecutive are checked for intersection. When inserting a circle it is necessary to see whether that circle is contained in either of its neighbors in the total order. To show that O(N log N) is optimal, allow the circles to shrink to single points and the element uniqueness problem results. □

Theorem 8 may be applied to the facilities location problem of determining whether any of a number of radio transmitters will interfere and to checking the layout of circular patterns.

V. Common Intersection

Theorem 9. *The common intersection of N half-planes can be found in O(N log N) time, and this is optimal.*

Proof: The intersection of k half-planes is a convex polygonal region of at most k sides. Let H_i be the half planes, so we are trying to form the N-fold intersection $\bigcap_{i=1}^{N} H_i$. Using the associative property of the intersection operator, we may rearrange terms to give

$$(H_1 \cap \ldots \cap H_{N/2}) \quad \cap \quad (H_{1+N/2} \cap \ldots \cap H_N) \quad .$$

The term on the left (in parentheses) is an intersection of N/2 half-planes and hence is a convex polygonal region of at most N/2 sides. The same is true of the term on the right. Since two convex polygonal regions of k sides can be intersected in O(k) time[2], the middle intersection operation above can be performed in O(N) time. If T(N) represents the time sufficient to form the intersection of N half-planes, then we have $T(N) = 2T(N/2) + O(N)$, so $T(N) = O(N \log N)$. This is a textbook example of divide and conquer.

To prove the lower bound, we will show that any algorithm which finds an intersection of half-planes can sort. Given N real numbers $x_1, \ldots, x_N$, let H_i be the half-plane containing the origin defined by a line of slope x_i that is tangent to the unit circle. The intersection of these half-planes is a convex polygonal region whose successive edges are ordered by slope. Once the intersection polygon is formed, we may immediately read off the x_i in sorted order, so O(N log N) comparisons must have been performed. □

Pavlidis[9] [10] uses half-planes as primitives for describing the syntax of patterns and decomposes figures into convex subsets by taking various intersections of half-planes. Theorem 9 applies directly to these problems.

Theorem 10. *The kernel of polygon can be found in O(N log N) time[2].*

Proof: Each side of the polygon determines a half-plane within which the kernel must lie. The kernel itself is the intersection of these half-planes, and Theorem 9 applies. □

If the polygon is simple, the lower bound of Theorem 9 does not apply directly, since the edges cannot be in arbitrary positions. We currently have no better than a linear lower bound for the kernel problem. Note that a non-simple polygon cannot have a kernel.

Theorem 11. *Whether N linear inequalities in two variables are consistent can be determined in O(N log N) time.*

Proof: The inequalities are of the form $a_i x + b_i y \leq c_i$ where a_i, b_i and c_i are constants. The inequalities are consistent iff there exists at least one pair x,y satisfying all inequalities simultaneously. Each inequality determines a half-plane within which such a point (x,y) must lie. If the intersection of all of these half-planes is non-empty, the inequalities are consistent. Thus Theorem 9 applies. □

Theorem 12. *A linear program in two variables and N constraints can be solved in O(N log N) time.*

Proof: The feasible region of a two-variable linear program is the intersection of the half-planes defined by the constraints. By Theorem 9, this region may be constructed explicitly in O(N log N) time. Since the objective function attains a maximum at some vertex of the feasible region, we need only compute it at most N times. □

The Simplex method requires $O(N^2)$ time to solve a two-variable problem because it finds extreme points the feasible region one at a time rather than collectively. Thus, Simplex is not optimal. In fairness, however, we must point out that, in higher dimensions, forming the entire feasible polytope (which Simplex avoids) is a distinctly inferior method because the number of extreme points may grow exponentially with dimension (the number of variables). It is not yet known whether Simplex can be beaten in three dimensions. We would like to proceed along the lines of Theorem 9, finding the intersection of N half-spaces, but this requires a fast algorithm for intersecting two convex polyhedra, and no such algorithm is known.

Returning to the two-dimensional problem, it is often of interest to solve the same linear program many times, with the same set of constraints but different objective functions. Having solved the linear program once, Simplex may spend $O(N^2)$ time maximizing a new objective function. If we have found the feasible region as in Theorem 12 (using only O(N log N) time), a new objective function can be maximized in O(log N) time by finding a line of support of the feasible region parallel to the line defined by the new constraint[2].

Theorem 13. *The common intersection of N circles can be found in O(N log N) time, and this is optimal.*

Proof: Substantially identical to the proof of Theorem 9. A linear algorithm for the intersection of two convex circular polygons (figures whose boundaries are piecewise circular) is a direct generalization of the convex polygon intersection algorithm. □

Theorem 14. *The common intersection of N convex k-gons can be found in O(Nk log N) time.*

Proof: There are a total of Nk polygon edges. Each of these defines a half-plane within which the final intersection must lie. By Theorem 9, the intersection of Nk half-planes can be accomplished in O(Nk log Nk) time. We can reduce the work required by operating on polygons as units, rather than manipulating single edges. Let T(N,k) be the time sufficient to solve the problem. The intersection of N/2 convex k-gons is a convex polygon having at most Nk/2 sides. Two of these can be intersected in time ckN, for some constant c. Thus, by recursively splitting the problem, we have $T(N,k) = 2T(N/2,k) + ckN$, giving $T(N,k) = O(Nk \log N)$.

VI. Conjectures and Unsolved Problems

1. Prove that O(N log N) time is required to determine whether a polygon is simple.

2. Prove that detecting whether two simple polygons intersect requires O(N log N) time.

3. Suppose that the intersection of two simple N-gons has k edges. Is there an $O(\max(k, N \log N))$ algorithm to construct it ?

4. Given N line segments in the plane, how much time is required to count the number of intersecting pairs?

5. Of N line segments, suppose that k pairs intersect. Is there an $O(\max(k, N \log N))$ algorithm to list them?

6. Prove that finding the kernel of a simple polygon requires O(N log N) operations.

7. Prove that determining whether N linear inequalities in two variables are consistent requires O(N log N) time.

8. Prove that solving a linear program in two variables and N constraints requires O(N log N) time. It is true that Theorem 9 proves that O(N log N) time is required to form the feasible region, but this may not be necessary to solve the linear program.

9. Find an algorithm for intersecting two three-dimensional convex polyhedra, each having N vertices, in less than $O(N^2)$. Such an algorithm could be used for linear programming in three variables. Very little is known about the complexity of linear programming in more than two variables.

VII. Summary

Intersection problems play a fundamental role in computational geometry, computer graphics, and linear programming. They seem to divide naturally into two classes: those requiring the detection of intersection and those requiring the formation of a common intersection. Fast algorithms for the former can be obtained by imposing an order relation on the objects to speed processing. In the latter class, the associative property of intersection permits a divide and conquer strategy to be employed effectively. Lower bounds are obtained by demonstrating a reducibility with known problems.

VIII. Acknowledgements.

Stan Eisenstat was the first to suggest that a linear algorithm for the intersection of two convex polygons would lead to an O(N log N) algorithm for intersecting half-planes, and this observation was the starting-point of our work. As always, the first author's wife, Julie, provided the spark necessary to to keep inspiration kindling.

References

1 Shamos, M.I., and Hoey, D. *Closest-Point Problems.* Proc. Sixteenth Annual Symposium on Foundations of Computer Science (1975), 151-162.

2 Shamos, M.I. *Geometric Complexity.* Proc. Seventh Annual ACM SIGACT Symposium (1975), 224-233.

3 Dobkin, D. and Lipton, R. *On the Complexity of Computations under Varying Sets of Primitives.* Yale Computer Science Research Report #42 (1975).

4 Reingold, E.M. *On the Optimality of Some Set Algorithms.* JACM 19,4(1972), 649-659.

5 Aho, A.V., Hopcroft, J.E. and Ullman, J.D. The Design and Analysis of Computer Algorithms. Addison-Wesley(1974), 470 pp.

6 Hoey, D. *On Determining Whether Line Segments in the Plane are Disjoint.* (1975) unpublished ms.

7 Sklansky, J. *Measuring Concavity on a Rectangular Mosaic.* IEEE Trans. Comp. C-21(1972), 1355-1364.

8 Shamos, M.I. Problems in Computational Geometry. Springer-Verlag, to appear.

9 Pavlidis, T. *Analysis of Set Patterns.* Pattern Recognition 1(1968), 165-178.

10 Pavlidis, T. *Representation of Figures by Labeled Graphs.* Pattern Recognition 4(1972), 5-17.