COMMON INTERSECTIONS OF POLYGONS

Laurence BOXER*
Department of Computer and Information Sciences, Niagara University, Niagara University, NY 14109, USA

Russ MILLER**
Department of Computer Science, 226 Bell Hall, State University of New York, Buffalo, NY 14260, USA

Communicated by Fred B. Schneider
Received 19 June 1989
Revised 14 August 1989

An algorithm is given for the detection of a common intersection with respect to a set of vertically convex planar polygons. A serial implementation is given, as well as parallel implementations for the CREW PRAM, hypercube, and mesh computers. Given an input of size n, the algorithm runs in $\Theta(n \log n)$ serial time and in $\Theta(\log n)$ time on a CREW PRAM, $\Theta(\log^2 n)$ time on a hypercube, and $\Theta(n^{1/2})$ time on a mesh, where all parallel machines are configured with n processors.

Keywords: Computational geometry, parallel algorithms, convex, common intersection problem

1. Introduction

In [8], a serial algorithm is given to detect the common intersection of a set of planar convex objects. The algorithm runs in optimal $\Theta(n)$ time when the objects are n circles. An implementation for convex polygons is also given. In this paper, we give an algorithm to solve a slightly more general problem—our algorithm works for vertically convex polygons—and give an efficient implementation of the algorithm on serial, CREW PRAM, hypercube, and mesh architectures. We say a planar object $P$ is vertically convex if for every pair of points $a, b \in P$, if $a$ and $b$ are on the same vertical line, then the line segment from $a$ to $b$ is contained in $P$.

Our algorithm depends, in part, on creating a description of the minimum function (also called the lower envelope)

$$h(t) = \min\{f_0(t), \ldots, f_{n-1}(t)\}$$

from descriptions of n functions $f_0, \ldots, f_{n-1}$ of a real variable. This represents a significant departure from the algorithm of [8] that is brought about by the greater generality of the problem we seek to solve. Notice that $h(t)$ is described by an ordered list of “pieces”, where each piece consists of an interval of the real line and a description of the function that is the minimum on the interval, and where the intervals of the pieces have disjoint interiors. This approach has been used previously for solving problems in dynamic computational geometry, where the input is a set of functions describing the motions of objects with respect to time, and the output is a description of a geometric property (such as nearest neighbor of a fixed object in the system) as a function of time. Serial algorithms using this approach for problems in dynamic computational geometry are given in [1], and parallel solutions of dynamic computational geometry problems are given in [2–4]. In this
paper, we show that a similar approach yields efficient parallel solutions to problems in static computational geometry.

The format of the paper is as follows. In Section 2, we present background information on the parallel models of computation. In Section 3 we give fundamental results concerning description of the min function. We present our algorithm for detecting common intersections in Section 4 and analyze its running times for planar polygons in Section 5. A summary is given in Section 6.

2. Parallel models of computation

In this section we define the parallel models of computation used in this paper. We will use the terms processor and processing element (PE) interchangeably.

2.1. CREW PRAM

A parallel random access machine (PRAM) is a computer with multiple processors that share memory. The PRAM model that we use in this paper is the concurrent read, exclusive write (CREW) PRAM, which permits several processors to read from the same memory location simultaneously, but permits only one processor at a time to attempt to write to a given memory location.

2.2. Mesh-connected computer

A (two-dimensional) mesh-connected computer (mesh) is a computer with multiple processors arranged as a square lattice. Each generic processor shares a bidirectional communication link with each adjacent processor in its row and in its column. Therefore, a mesh of size $n$ has $n$ processors arranged as an $n^{1/2} \times n^{1/2}$ lattice, with each generic processor connected to its northern, southern, eastern, and western neighbors.

A mesh of size $n$ has communication diameter $\Theta(n^{1/2})$, meaning that the maximum number of communication links separating any pair of PEs in the mesh is $\Theta(n^{1/2})$. Therefore, if a problem on a mesh of size $n$ requires the possibility of two processors that are $\Theta(n^{1/2})$ communication links apart to communicate, and the running time of an algorithm to solve the problem is $O(n^{1/2})$, then the algorithm is optimal.

2.3. Hypercube computer

A hypercube of size $n$, where $n$ is a nonnegative integral power of 2, has $n$ PEs indexed by the integers $\{0, 1, \ldots, n-1\}$. If we view each integer in the index range as a $(\log n)$-bit string, two PEs are connected by a bidirectional communication link if and only if their indices differ in exactly one bit.

The communication diameter of a hypercube of size $n$ is $\Theta(\log n)$. Therefore, if a problem in a hypercube of size $n$ requires the possibility of two PEs that are $\Theta(\log n)$ communication links apart to communicate, and the running time of an algorithm to solve the problem is $O(\log n)$, then the algorithm is optimal.

3. The min function

3.1. Pieces and the function $\lambda$

Input to problems in this paper consists of descriptions of planar curves that are graphs of equations of the form $y = f(t)$, where the functions $f(t)$ require $\Theta(1)$ memory to describe and where if $f_1(t)$ and $f_2(t)$ are two such functions, then all solutions of $f_1(t) = f_2(t)$ may be found in serial $\Theta(1)$ time. These assumptions are used in [1–4] and are consistent with [8]. The planar curves represented by the input will be edges of polygons, or, more generally, edges of planar objects whose common intersection is in question.

Given a set of real-valued functions $\Phi = \{f_0, \ldots, f_{n-1}\}$ defined on some interval $I_0$ of the real line, it will often be useful to construct the minimum function $h(t)$ defined in equation (1). Define a piece of the minimum function generated by $\Phi$ to consist of a description of some $f_j$ and an interval $I \subset I_0$ such that $h = f_j$ identically on $I$ and such that $h$ is not identically equal to any $f_j$ over any interval $J \subset I_0$ such that $I$ is properly contained in $J$. A piece of the maximum function generated by $\Phi$ is defined similarly.
If \( h_1(t) \) and \( h_2(t) \) are real-valued functions defined on \( I_0 \) whose pieces are generated by a family of functions \( \Phi \), then a piece of \( h_1 - h_2 \) generated by differences of members of \( \Phi \) consists of a description of a function \( g \) and an interval \( I \subseteq I_0 \) such that

1. there exist \( g_1, g_2 \in \Phi \) such that \( g = g_1 - g_2 \) identically on \( I_0 \),
2. \( h_1 - h_2 = g \) identically on \( I \), and
3. \( h_1 - h_2 \) is not identically equal to \( g \) on any interval \( J \subseteq I_0 \) such that \( I \subseteq J \) is a proper subset of \( J \).

Let \( \Phi = \{ f_0, \ldots, f_{n-1} \} \) be a set of continuous real-valued functions defined on an interval \( I_0 \), no pair of whose graphs intersect more than \( s \) times. Then an upper bound for the number of pieces of the minimum function \( h(t) \) described in equation (1) is given [1] by a function \( \lambda(n, s) \) associated with the maximal length of a Davenport–Schinzel sequence [5]. Although \( \lambda(n, s) \) is not, in general, linear in \( n \) [6], we have the following.

**Proposition 3.1** [5]. \( \lambda(n, 1) = n \) and \( \lambda(n, 2) = 2n - 1 \).

In fact, for \( s \) a fixed positive integer, \( \lambda(n, s) \) is “almost” linear in \( n \) [6,9].

Two intervals have a nondegenerate intersection if and only if their intersection contains more than one point. The next result gives a useful bound on the number of pieces in a “combined” function.

**Proposition 3.2** [2]. Let \( f(t) \) and \( g(t) \) be real-valued functions defined for all \( t \geq 0 \). Let \( m \) and \( n \) be positive integers. Suppose \( f(t) \) has \( m \) pieces and \( g(t) \) has \( n \) pieces. Then the intervals of pieces of \( f(t) \) have, altogether, at most \( m + n \) nondegenerate intersections with the intervals of pieces of \( g(t) \).

### 3.2. Constructing the min function

In this section, we give results from [1–4] for efficient construction of a description of the minimum function in parallel.

Let \( \Phi \) be a family of real-valued functions, each defined on some interval of the real line. Let \( s \) be a fixed positive integer. We say \( \Phi \) is \( s \)-like if the following are satisfied.

1. \( f \in \Phi \) implies \( f \) is continuous on its domain.
2. \( f \in \Phi \) implies \( f \) has a \( \Theta(1) \) storage description.
3. If \( f \in \Phi \) and \( t \) is a real number, then it may be decided in \( \Theta(1) \) serial time whether \( f(t) \) is defined, and if so, \( f(t) \) may be evaluated in \( \Theta(1) \) serial time.
4. If \( f, g \in \Phi \), then there are at most \( s \) solutions to the equation \( f(t) = g(t) \), and the solutions may be calculated in \( \Theta(1) \) serial time.

Note that a family of distinct polynomials of degree at most \( s \) is \( s \)-like.

**Proposition 3.3.** Let \( s \) be a fixed positive integer. Let \( \Phi \) be an \( s \)-like family of functions. Let \( f(t) \) and \( g(t) \) be real-valued functions defined by pieces generated by \( \Phi \). Suppose \( n \) is a positive integer such that the total number of pieces of \( f \) and \( g \) is at most \( n \). Suppose the pieces of \( f \) and the pieces of \( g \) are ordered in global memory of a serial computer or of a CREW PRAM with \( n \) PEs, or at most one piece per PE in a hypercube of size \( n \) or a mesh of size \( n \).

Then the function \( h(t) = \min\{ f(t), g(t) \} \) has \( O(n) \) pieces, and they may be described by the serial computer in \( O(n) \) time, by the PRAM or the hypercube in \( O(\log n) \) time, and by the mesh in \( O(n^{1/2}) \) time.

**Proposition 3.4.** Let \( s \) be a fixed positive integer. Let \( \{ f_0, \ldots, f_{n-1} \} \) be an \( s \)-like family of functions. Suppose descriptions of \( f_0, \ldots, f_{n-1} \) are stored in global memory of a serial computer or of a CREW PRAM with \( \lambda(n, s) \) PEs, or \( \Theta(1) \) per PE in either a hypercube with \( \lambda(n, s) \) PEs or a mesh with \( \lambda(n, s) \) PEs. Then the minimum function \( h(t) \) can be constructed by the serial computer in \( O(\lambda(n, s)\log n) \) time, by the PRAM in \( \Theta(\log n) \) time, by the hypercube in \( \Theta(\log^2 n) \) time, and by the mesh in \( O(\lambda^{1/2}(n, s)) \) time. At the end of the algorithm, the description of \( h(t) \) is given by \( O(\lambda(n, s)) \) pieces ordered by their intervals, stored in the cases of the hypercube and the mesh \( \Theta(1) \) pieces per PE.

The min operation may be replaced by max in Propositions 3.3 and 3.4, and by difference in Proposition 3.3.
4. The algorithm

In this section, we present a general algorithm to determine whether or not planar figures have a common intersection. Suppose we have \( k \leq n \) planar objects \( D_0, D_1, \ldots, D_{k-1} \) that can be described by \( n \) data items requiring \( \Theta(1) \) memory apiece. For example, we may have \( k \) polygons with a total of \( \sum \) edges. When the model of computation we use is the serial or PRAM computer, descriptions of these data items are stored in global memory. When the model of computation is the mesh or hypercube, descriptions of the data items are distributed \( \Theta(1) \) per PE. Each data object \( Q \) has a boundary made up of a lower curve \( L_\gamma \) and an upper curve \( U_\gamma \) that are determined by a leftmost and rightmost boundary point of \( D_i \) (see Fig. 1).

Assume real-valued functions of a real variable are expressed in the notation \( y = f(t) \). Let the functions \( L_i(t) \) and \( U_i(t) \) be defined by (see Fig. 1)

\[
L_i(t) = \begin{cases} 
  y, & \text{if there exists a } y \\
  \infty, & \text{such that } (t, y) \in L_i,
\end{cases}
\]

and

\[
U_i(t) = \begin{cases} 
  y, & \text{if there exists a } y \\
  -\infty, & \text{such that } (t, y) \in U_i,
\end{cases}
\]

If \( t_1 < t_r \), halt, as there is no common intersection. Otherwise, notice that if a common intersection exists, its first coordinate is in \( [t_l, t_r] \).

**Step 1: Range-finding.** Determine left and right bounds \( t_l \) and \( t_r \), respectively, for any common intersection, as follows (see Fig. 2). Determine for each \( D_i \) its leftmost \( t \)-value \( l_i \) and its rightmost \( t \)-value \( r_i \). Next, compute the rightmost of the left values

\[
t_l = \max\{ l_i : i = 0, 1, \ldots, k - 1 \}
\]

and the leftmost of the right values

\[
t_r = \min\{ r_i : i = 0, 1, \ldots, k - 1 \}.
\]

**Step 2: Maximize lowers.** For \( t_l \leq t \leq t_r \), construct a description of the function

\[
L(t) = \max\{ L_i(t) : i \in \{ 0, 1, \ldots, k - 1 \} \}.
\]

**Step 3: Minimize uppers.** For \( t_l \leq t \leq t_r \), construct a description of the function

\[
U(t) = \min\{ U_i(t) : i \in \{ 0, 1, \ldots, k - 1 \} \}.
\]

**Step 4: Subtract.** Construct a description of the function

\[
F(t) = L(t) - U(t).
\]

Our algorithm is given below.

**Step 1: Range-finding.** Determine left and right bounds \( t_1 \) and \( t_2 \), respectively, for any common intersection, as follows (see Fig. 2). Determine for each \( D_i \) its leftmost \( t \)-value \( l_i \) and its rightmost \( t \)-value \( r_i \). Next, compute the rightmost of the left values

\[
t_l = \max\{ l_i : i = 0, 1, \ldots, k - 1 \}
\]

and the leftmost of the right values

\[
t_r = \min\{ r_i : i = 0, 1, \ldots, k - 1 \}.
\]

If \( t_1 > t_2 \), halt, as there is no common intersection. Otherwise, notice that if a common intersection exists, its first coordinate is in \( [t_1, t_2] \).

**Step 2: Maximize lowers.** For \( t_1 \leq t \leq t_2 \), construct a description of the function

\[
L(t) = \max\{ L_i(t) : i \in \{ 0, 1, \ldots, k - 1 \} \}.
\]

**Step 3: Minimize uppers.** For \( t_1 \leq t \leq t_2 \), construct a description of the function

\[
U(t) = \min\{ U_i(t) : i \in \{ 0, 1, \ldots, k - 1 \} \}.
\]

**Step 4: Subtract.** Construct a description of the function

\[
F(t) = L(t) - U(t).
\]
Step 5: Detect. Determine $M = \min \{ F(t); t_1 \leq t \leq t_0 \}$. There is a common intersection if and only if $M \leq 0$ [8].

We observe that the algorithm of [8] does not require our Subtract step. This is because when the $D_i$ are all convex, the function $F(t)$ is a convex function, which allows its minimum value to be determined via data elimination methods not available in our more general situation.

5. Detecting a common intersection of polygons

If $P$ is a vertically convex planar object whose boundary contains no vertical line segments, then the boundary of $P$ is the union of two curves, $U$ and $L$, that meet at unique leftmost and rightmost points of $P$ such that each of $U$ and $L$ is the graph of a function of the variable $r$ (which is represented by the plane's horizontal axis). We assume that input to the problem consists of $n$ labeled edges so that two edges have the same label if and only if they bound the same polygon.

Theorem 5.1. Let $D_0, D_1, \ldots, D_{k-1}$ be polygonal disks in the Euclidean plane having a total of $n$ boundary edges. Assume each edge of $D_i$ is labeled $i$. Assume the edges are stored in the global memory of a serial computer or of a CREW PRAM with $n$ PEs or are arbitrarily distributed $\Theta(1)$ per PE among the $n$ PEs of a hypercube or mesh. Assume each of the $D_i$ is vertically convex. Then the existence of a common intersection of $D_0, D_1, \ldots, D_{k-1}$ can be detected by the serial computer in $\Theta(n \log n)$ time, by the PRAM in $\Theta(\log n)$ time, by the hypercube in $\Theta(\log^2 n)$ time, and by the mesh in $\Theta(n^{1/2})$ time.

Proof. There is no loss of generality in assuming that each $D_i$ has no vertical boundary edges, as a simple preprocessing step can handle the existence of vertical edges within the allotted time. All running times are worst-case.

The Range-finding step is done as follows. Sort the edges by polygonal disk label and mark the first and last edge within every label in $\Theta(n \log n)$ time in serial, $\Theta(\log n)$ time on the PRAM, $\Theta(\log^2 n)$ time on the hypercube, or $\Theta(n^{1/2})$ time on the mesh [7]. Within every polygon $D_i$, determine the leftmost and rightmost $r$-values $l_r$ and $r_r$, respectively, for the polygon via min and max operations on the endpoints of the edges. This can be performed using a semigroup operation that takes $\Theta(n)$ times in serial, $\Theta(\log n)$ time on the PRAM and hypercube and $\Theta(n^{1/2})$ time on the mesh. Now use a global semigroup operation on these values to compute $t_r$ and $t_l$ in $\Theta(n)$ time in serial, in $\Theta(\log n)$ time for the PRAM and hypercube, and in $\Theta(n^{1/2})$ time for the mesh.

The Maximize lowers and Minimize uppers steps require that each edge in $D_i$, for all $i$, first determines whether it belongs to $U_i$ or to $L_i$. This can be done as follows. Create two records per edge (one corresponding to each endpoint), using $\Theta(n)$ time in serial, $\Theta(1)$ time in parallel. Within every polygon, sort these records according to $r$-coordinates of the endpoints. For the polygon $D_i$, the first two edges have left endpoint with $r$-coordinate $l_r$. In serial $\Theta(n)$ time or in parallel $\Theta(1)$ time, determine for each such pair which edge of this pair belongs to $U_i$ and which belongs to $L_i$, according to which has greater slope. Using another sort step, reunite the two records representing each edge. (Now each edge knows the two edges it is adjacent to.) A (parallel) prefix operation on the edge records within each label is now used so that each edge of each polygon $D_i$ knows whether it belongs to $U_i$ or $L_i$. Sorting requires $\Theta(n \log n)$ time in serial, $\Theta(\log n)$ time on the PRAM, $\Theta(\log^2 n)$ time on the hypercube, and $\Theta(n^{1/2})$ time on the mesh. Prefix computations require $\Theta(n)$ time in serial, $\Theta(\log n)$ time for the PRAM and hypercube, $\Theta(n^{1/2})$ time for the mesh.

Now descriptions of the functions $L(t)$ and $U(t)$ can be constructed, using the algorithm and running times associated with Proposition 3.4.

By Proposition 3.3, the function $F(t)$ may be described (the Subtract step) in serial $\Theta(n)$ time, by the PRAM and by the hypercube in $\Theta(\log n)$ time, and by the mesh in $\Theta(n^{1/2})$ time.

By Propositions 3.2 and 3.1, the function $F(t)$ has $\Theta(n)$ pieces generated by differences of members of the family of functions whose graphs are edges of $\{D_0, D_1, \ldots, D_{k-1}\}$. Since the minimum value on a given piece of $F(t)$ can be found in
serial $\Theta(1)$ time, it follows that the Detect step can be performed in serial $\Theta(n)$ time, by a PRAM or hypercube in $\Theta(\log n)$ time, and by a mesh in $\Theta(n^{1/2})$ time.

Therefore, the algorithm runs in serial $\Theta(n \log n)$ time, on a PRAM in $\Theta(\log n)$ time, on a hypercube in $\Theta(\log^2 n)$ time, and on a mesh in $\Theta(n^{1/2})$ time.

Minor modifications in the proof of Theorem 5.1 yield the following.

**Corollary 5.2.** Let $s$ be a positive integer. Let $D_0, D_1, \ldots, D_{k-1}$ be disks in the Euclidean plane having a total of $n$ boundary edges. Suppose each edge is an arc of the graph of an equation of the form $y = f(r)$, where $f$ is a polynomial of degree at most $s$. Assume each edge of $D_i$ is labeled $i$. Assume the edges are stored in the global memory of a serial computer or of a CREW PRAM with $\lambda(n, s)$ PEs or are arbitrarily distributed at most $\Theta(1)$ per PE among the $\lambda(n, s)$ PEs of a hypercube or mesh. Assume each of the $D_i$ is vertically convex. Then the existence of a common intersection of $D_0, D_1, \ldots, D_{k-1}$ can be detected by the serial computer in $\Theta(\lambda(n, s) \log n)$ time, by the PRAM in $\Theta(\log n)$ time, by the hypercube in $\Theta(\log^2 n)$ time, and by the mesh in $\Theta(\lambda^{1/2}(n, s))$ time.

**6. Summary**

We have given a general algorithm, with implementations for serial, CREW PRAM, hypercube, and mesh computers, to detect the existence of a common intersection of vertically convex planar polygons, or, more generally, vertically convex planar disks whose boundaries are arcs of polynomial graphs. The algorithm makes use of techniques developed for the study of problems in dynamic computational geometry.

Our algorithm runs in optimal $\Theta(n^{1/2})$ time for the mesh. The optimality of our running times of $\Theta(n \log n)$ in serial, $\Theta(\log n)$ for the CREW PRAM, and $\Theta(\log^2 n)$ for the hypercube, are open questions. If the common intersection problem we have studied reduces to the min problem, then it reduces to sorting [4]. It would then follow that our running times are optimal for the serial and CREW PRAM models. Sorting in optimal time is an open problem for the hypercube. Thus our algorithms may be improved on the hypercube by discovery of a faster hypercube sorting algorithm.

**References**