

ON COMPUTING SIMPLE CIRCUITS ON A SET OF LINE SEGMENTS

David Rappaport †
Hiroshi Imai ‡
Godfried T. Toussaint †

† School of Computer Science
McGill University
805 Sherbrooke St. W.
Montreal, Canada H3A 2K6

‡ Department of Mathematical Engineering
and Instrumentation Physics
University of Tokyo
Bunkyo-Ku, Tokyo, Japan

1. Introduction.

Given a set of non-intersecting line segments in the plane, we are required to connect the line segments such that they form a simple circuit (a simple polygon). However, not every set of segments can be so connected. Figure 1 shows a set of segments that does not admit a simple circuit.

This leads to the challenging problem of determining when a set of segments admits a simple circuit, and if it does, then find such a circuit. It has been shown [Rappaport] that in general, to determine whether a set of segments admits a simple circuit is NP-complete. In this paper an optimal algorithm is presented to determine whether a simple circuit exists, and deliver a simple circuit, on a set of line segments, where each segment has at least one endpoint on the convex hull of the segments (a CH-connected set of segments). Furthermore this technique can be used to determine a simple circuit of minimum length, or a simple circuit that bounds the minimum area, with no increase in computational complexity.

The rest of the paper is summarized. In section 2 of this paper, the preliminary definitions and notation are introduced. In section 3, the geometric properties of the set of segments are used to transform the segments into an associated graph.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1986 ACM 0-89791-194-6/86/0600/0052 \$00.75,

A Hamiltonian circuit in this graph is then used to deliver the connections of segments that form the boundary of a simple polygon. In section 4, a linear algorithm is introduced which finds, if there is one, a Hamiltonian circuit in graphs of the class obtained by the transformation discussed in section 3. This algorithm actually computes the minimum weight matching on an extremely structured bipartite graph. From this the result on minimal simple circuits follows immediately. Section 5, relates the details of a necessary step in the segment to graph transformation. This involves the intersection of line segments in the plane. The paper is summarized in section 6, where the proof of optimality of the algorithm is given.

2. Definitions and Notation.

A set of non-intersecting line segments S is represented as $S = (s_0, s_1, \dots, s_{n-1})$ (to be referred to from now on as segments). The endpoints of S will be represented by the set of $2n$ points, $P = (p_0, p_1, \dots, p_{2n-1})$.

Define a *simple circuit* as a sequence of points in the plane that lie in clockwise order on the boundary of a simple closed curve. A simple circuit can be represented by a set of segments, the edges of the simple circuit. A *simple circuit on a set of segments, S* , is a simple circuit representable by a superset of S .

Given a set of non-intersecting segments, represented by S it is sometimes possible to find a simple circuit. If this circuit exists we say that S admits a simple circuit. Denote R a set of n non-intersecting segments whose endpoints are in P such that $R \cup S$ represents a simple circuit and R and S

are disjoint sets. We will refer to R as the set of *augmenting segments* of S . In Figure 2 a set of segments is shown in solid lines, with its corresponding set of augmenting segments in broken lines.

The convex hull $CH(P)$ of a set of points P is the smallest convex region enclosing P . Note that if S admits a simple circuit this circuit is enclosed by the convex hull of the endpoints of S , $CH(P)$. If S contains a segment s such that both endpoints of s lie on $CH(P)$ and the interior of s lies in the interior of $CH(P)$, then denote s a *cutting segment*.

Theorem 2.1: If S contains a cutting segment then S does not admit a simple circuit.

Proof: Assume S admits a simple circuit and contains a cutting segment s . Let p_i and p_j , in P be points on different sides of s . Since every simple circuit on S is enclosed by $CH(P)$ then every path on the simple circuit from p_i to p_j must pass through s . It is well known that for every pair of points x, y on a circuit there exists two disjoint paths from x to y . Since every path from p_i to p_j must pass through s , S cannot admit a simple circuit. ■

This result suggests an easy way to determine whether a set of segments may not admit a simple circuit. Given S and P , we compute $CH(P)$ and then examine segments of S to see if any are cutting segments. The convex hull of a set of points can be computed in $O(|P| \log |P|)$ time, [Graham], [Tousaint], where $|P|$ represents the cardinality of P .

It should be noted, however, that even though a set of segments does not have a cutting segment, it still may not admit a simple circuit. (Figure 3)

For the remainder of the discussion we will constrain the domain of the set of segments. Define a set of segments as *CH-connected* if for every segment $s \in S$, at least one of the endpoints of s lies on $CH(P)$. We will also assume that the set S in the ensuing discussion contains no cutting segments, and $|S| > 4$.

3. Geometric Results.

The approach that will be taken is to associate a CH-connected set of segments represented by S to a graph $G = (V, E)$. If the endpoints of S , $P = \{p_0, p_1, \dots, p_{2n-1}\}$ and the vertices of G , $V = \{v_0, v_1, \dots, v_{2n-1}\}$ then p_i corresponds to v_i . Similarly a segment referred to as (p_i, p_j) has its corresponding image, the edge (v_i, v_j) . Using the geometric properties of S , we arrive at the appropriate set of edges E so that solving a combinatorial problem in G , leads to a solution to our original problem.

In the search for augmenting segments to form a simple circuit one must consider likely *candidates* for being augmenting segments. Clearly it simplifies matters if the pool of candidates is small.

Initially there are $O(n^2)$ candidates i.e. $P \times P$ such that $(p_i, p_j) \notin S$ and $p_i \neq p_j$. However the upcoming key lemma reduces the number of candidates drastically. An intuitive description will be given, before stating the lemma formally.

Denote segments of S whose endpoints are adjacent on $CH(P)$ as *neighbors*. The following lemma proves that all the endpoints of augmenting segments are endpoints of neighbors in S .

Lemma 3.1: Given a CH-connected set of segments S that admits a simple circuit represented by the sequence of points $T = (t_0, t_1, \dots, t_{2n-1})$. Let $B = (b_0, b_1, \dots, b_m)$ be the sequence of points representing $CH(P)$. Assume without loss of generality that $b_0 = t_0$. For every such sequence T a simple circuit on S , the sequence B is a subsequence of T .

Proof: Assume b_1, b_3, b_2 is a subsequence of T . (See Figure 4) This creates a polygonal chain from b_1 to b_3 that separates b_2 from the remaining points. By using arguments similar to those of theorem 2.1 we see that this sequence must lead to a non-simple circuit, a contradiction. ■

The pool of $O(n)$ candidates can be reduced further by considering segment intersection of candidates. Clearly an augmenting segment cannot intersect any segment in S . By a naive algorithm it would require at most $O(n^2)$ time to determine which of the current $O(n)$ candidates intersect any of the n segments of S . However using a variant of Shamos and Hoey's line sweep technique [Shamos and Hoey], and a careful decomposition of the segments involved this can be done in $O(n \log n)$ time. To avoid a lengthy digression from the current discussion a detailed description of this algorithm is postponed until section 5.

As was stated earlier it is desirable to put this problem into a purely combinatorial setting. By associating a graph to the original points and segments, the goal will be to determine the existence of a Hamiltonian circuit in the graph, that corresponds to a simple circuit in the underlying segments. A *Hamiltonian circuit* is a simple closed path through all the nodes of a graph. Of course the Hamiltonian circuit in G requires the inclusion of the edges that correspond to the segments S . Let E_s represent the edges in G that correspond to segments of S . Denote an E_s -required Hamiltonian circuit, H , a Hamiltonian circuit of G such that $H \cap E_s = E_s$.

If the current pool of candidates is used as the edge set in G and an E_s -required Hamiltonian circuit is found in G , then we are not ensured that the resulting circuit of segments is non-intersecting. This is because among the pool of candidates so far described, one notices that, there may be intersections between pairs of candidates. It is useful to distinguish between three types of these intersections.

Let a, b be two candidates.

Case 1: All four endpoints of candidates a and b are endpoints of only two of the segments of S (See Figure 5). In this case we can allow the images of both a and b to appear in the final graph G . Any E_s -required Hamiltonian circuit of G cannot contain both a and b . We would visit both endpoints of the segments connected by a and b , before visiting the rest of the segments of S , therefore, a and b cannot appear together in a Hamiltonian Circuit.

Case 2: The four endpoints of a and b lie on three different segments of S . (See Figure 6). Therefore one of the segments of S has a candidate at both of its endpoints. Denote the segment t incident to both a and b with p_a the endpoint of t on a and p_b the endpoint of t on b . Denote the neighbors of t as t^- and t^+ . At least one endpoint of t is on $\text{CH}(P)$, so one endpoint of a or b must also be on $\text{CH}(P)$. Without loss of generality assume p_a is on $\text{CH}(P)$, and a 's other endpoint is on t^- . Because a and b intersect they cannot be edges of $\text{CH}(P)$. Observe that the candidates connecting p_b with each of the endpoints of t^+ must intersect a . Therefore p_b is isolated from the segment t^+ by a . This implies that a cannot be an augmenting segment.

It is convenient to label the segments in S so that s_i is a neighbor of s_{i+1} for all $i=0, \dots, n-1$ (addition modulo n). Let c be a candidate with endpoints on the segments s_i and s_{i-1} . Define c a *blocking candidate* if either the segments s_i and s_{i+1} are on opposite sides of a chain (s_{i-1}, c) , or the segments s_{i-1} and s_{i-2} are on opposite sides of a chain (s_i, c) . The candidate a described above is a blocking candidate.

Lemma 3.2: A blocking candidate cannot be an augmenting segment.

Proof: This follows immediately from the preceding discussion. ■

From the definition of blocking candidates it should be clear that $O(n)$ operations are sufficient to determine all blocking candidates.

Case 3: The four endpoints of a and b lie on four

different segments of S . Let a be a candidate with endpoints on s_i and s_{i+1} , and let b be a candidate with endpoints on s_j and s_{j+1} . Let h_i denote the convex hull edge from s_i to s_{i+1} , and let h_j denote the convex hull edge from segment s_j to s_{j+1} . Therefore the quadrilaterals $Q_i = (s_i, a, s_{i+1}, h_i)$ and $Q_j = (s_j, b, s_{j+1}, h_j)$ intersect. (Observe that if one of the endpoints a or b is on h_i or h_j , then we must consider triangles rather than quadrilaterals, however this does not effect the argument.) Two intersecting polygons intersect in at least two points. The intersection of a and b accounts for one of the intersections. Since, no edge can intersect a convex hull edge and none of the segments of S intersect, we must conclude that one of the candidates intersects a segment of S . But this type of intersection has been previously determined and the offending candidate removed.

Lemma 3.3: If two candidates a and b intersect, and the four endpoints of a and b lie on four different segments of S , then a or b must intersect one of those four segments.

Proof: Follows immediately from the preceding discussion. ■

The construction of a graph with the property that the original segments admit a simple circuit if and only if the graph admits an E_s -required Hamiltonian, circuit can now be obtained. Let C represent the set of candidates, with endpoints on neighbors, that do not intersect any segment in S , and are non-blocking. The edges E of G correspond to the line segments $S \cup C$.

Lemma 3.4: The segments S admit a simple circuit if and only if $G = (V, E)$ has an E_s -required Hamiltonian circuit.

Proof: Suppose S admits a simple circuit. It is required to show that the augmenting segments have their counterparts in G . It was shown that all augmenting segments were of the type used to obtain the set C above. Every segment in C has a counterpart edge in E so G must have an E_s -required Hamiltonian circuit.

Suppose G admits an E_s -required Hamiltonian circuit. Edges in E are easily mapped back to segments. The resulting circuit (of line segments) must be simple by the way candidates were chosen. ■

It is well known that in general, determining whether there is a Hamiltonian circuit in a graph is NP-complete. However in the next section a linear time algorithm is presented to determine whether an

E_s -required Hamiltonian circuit is present in G . Furthermore the same algorithm is used to determine minimum weight E_s -required Hamiltonian circuits.

4. Finding E_s -required Hamiltonian circuits.

Let the graph G can be characterized as follows: $G = (V, E)$ where $V = (v_0, v_1, \dots, v_{2n-1})$ and $E = E_s \cup E_c$ where:

$$E_s = \{(v_{2i}, v_{2i+1}), i = 0, \dots, n-1\}$$

and

$$E_c \subseteq \{(v_i, v_{i+2}), i = 0, \dots, 2n-1\}$$

$$\cup \{(v_{2i}, v_{2i+3}), i = 0, \dots, n-1\}$$

$$\cup \{(v_{2i+1}, v_{2i+2}), i = 0, \dots, n-1\}$$

(all index additions are modulo $2n$).

Denote G_c as the the graph $G = (V, E_c)$. The Graphs G and G_c have cyclic structures. It is more convenient to designate a vertex a start vertex and a vertex an end vertex and 'break' the cyclic structure. Remove from E_c (if they exist) the edges (v_0, v_2) , (v_0, v_3) and (v_1, v_{2n-1}) , (v_1, v_{2n-2}) . Call the resulting graph $G_c' = (V, E_c')$. To find an E_s -required Hamiltonian circuit in G , we will use the graphs G_c' .

A *matching* in a graph is a set of edges no two of which share a vertex. A *maximal matching* is a matching on the maximum number of vertices in the graph. A matching is said to be *complete* if a maximal matching in the graph contains all vertices of the graph.

The following theorem immediately leads to an algorithm for finding an E_s -required Hamiltonian circuit in G .

Theorem 4.1: Given the graph G and G_c' as described above. If a maximal matching in the graph G_c' is a complete matching then there is an E_s -required Hamiltonian circuit in G .

Proof: Every complete matching in G_c' must match v_1 with either v_2 or v_3 . Choosing either of these edges in the matching and deleting edges on matched vertices we are left with a graph having the same structure as our original graph. This gives the complete matching M the property that every edge $m \in M$ connects two edges $\in E_s$, and there are no disjoint cycles. The edges $E_s \cup M$ comprise a Hamiltonian circuit in G . ■

To determine whether there is an E_s -required Hamiltonian circuit in G simply compute the

maximal matching of G_c' . If the matching is complete then an E_s -required Hamiltonian circuit in G can be easily constructed. If there is no complete matching then reunite the edges removed from G_c' . Now remove from G_c' (if it exists) the edges (v_1, v_2) , (v_1, v_3) and (v_0, v_{2n-1}) , (v_0, v_{2n-2}) to obtain $G_c'' = (V, E_c'')$. Compute the maximal matching in G_c'' . If there is a complete matching then there is an E_s -required Hamiltonian circuit in G . Otherwise it can be concluded that there is no Hamiltonian Circuit in G .

A *bipartite graph* is a graph whose vertices V can be divided into disjoint subsets W and U such that every edge in the graph has one endpoint in W and one endpoint in U . It is easy to see that G_c' and G_c'' are bipartite graphs. The vertices $\{v_0, v_1, v_4, v_5, v_8, v_9 \dots\} \subseteq W$, and $\{v_2, v_3, v_6, v_7, v_{10}, v_{11} \dots\} \subseteq U$. In [Hopcroft and Karp] an efficient algorithm based on a network flow algorithm is given to find maximal matchings in bipartite graphs. The complexity of this algorithm is $O(|V|^{1/2} |E|)$. In the problem considered here the edges and the vertices are both of cardinality $O(n)$ so the running time is $O(n^{3/2})$.

The structure of the graphs G_c' and G_c'' permit a more efficient method of determining whether there is a complete matching. This algorithm will now be discussed.

A *weighted graph* has a real value $w(e)$ assigned to each edge e of the graph. This algorithm computes the minimum weight matching in the weighted graph $G_s = (V, E_s)$. V is defined as above and:

$$E_s = \{(v_i, v_{i+2}), i = 0, \dots, 2n-1\}$$

$$\cup \{(v_{2i}, v_{2i+3}), i = 0, \dots, n-1\}$$

$$\cup \{(v_{2i+1}, v_{2i+2}), i = 0, \dots, n-1\}.$$

Assign weights $w(e) = 1$ if $e \in E_c'$ and $w(e) = 2$ if $e \notin E_c'$. The minimum weight complete matching in G_s is computed as follows:

ALGORITHM MATCH

```

ω[2] ← w(v1, v2); ω[3] ← w(v1, v3);
for i ← 2 to n-1
  do begin
    ω[2i] ← min(ω[2i-1] + w(v2i-2, v2i),
                ω[2i-2] + w(v2i-1, v2i));
    ω[2i+1] ← min(ω[2i-1] + w(v2i-2, v2i),
                  ω[2i-2] + w(v2i-1, v2i));
  end ;
Ω ← min(ω[2n-1] + w(v2n-2, v0),
        ω[2n-2] + w(v2n-1, v0));

```

The cost of the minimum weight matching is kept in Ω . If the cost is n then a complete matching exists in G_c' , otherwise there is no complete matching. To prove the algorithm correct and show that $O(n)$ operations are used is straight forward. To obtain in $O(n)$ time, the edges of the complete matching $M = (m_1, m_2, \dots, m_n)$, the following simple procedure can be used.

ALGORITHM OBTAIN

```

k ← 0;
for i ← n downto 1
  if  $\omega[2i-2] = \Omega - w(v_{2i-1}, v_k)$ 
    then
       $m_i \leftarrow (v_{2i-1}, v_k); \Omega \leftarrow \Omega - 1; k \leftarrow 2i-2$ 
    else
       $m_i \leftarrow (v_{2i-2}, v_k); \Omega \leftarrow \Omega - 1; k \leftarrow 2i-1;$ 

```

Assume a complete matching in G_* has been found. A direct consequence of the previous result is that a simple circuit on S , that has minimum perimeter can be determined. A weighted graph $G_l = (V, E_l)$ is constructed. E_l is defined as E_* except for the weight assignments. Let the Euclidean length of a candidate be the weight given to the corresponding edge $\in E_l$, and, $\in E_c'$. For any edge $\in E_l \notin E_c'$ assign a weight of ∞ . Finding a minimum weight complete matching, in G_l reveals a simple circuit of minimum perimeter.

The simple circuit which encloses the minimum area can also be found by using a weighted graph. The weights assigned to edges in G hinge on the observation that the area of a simple polygon Q , is the area of $CH(Q)$ less the sum of the areas of the polygonal regions that constitute the difference between $CH(Q)$ and Q . Denote the polygonal regions that constitute the difference between $CH(Q)$ and Q as *convex deficiency polygons* of Q . The convex deficiency polygons of every simple circuit on S , consist of two segments s_i, s_{i+1} and the augmenting segment connecting s_i and s_{i+1} . (If the augmenting segment happens to connect two convex hull vertices we can conveniently define this as a zero area convex deficiency polygon.) Therefore every candidate describes an unique convex deficiency polygon. Assign weights to G , giving the weighted graph G_a where edges in G_a corresponding to candidates are given weights equal to the negation of area of the deficiency polygon described by that candidate. For edges $\notin E_c'$ assign a weight of 1. A complete matching in G_a with minimum weight, is a simple circuit that encloses the smallest area.

5. Finding Intersections of Candidates and

Segments.

Determining candidate-segment intersections is a necessary step to obtain the final set of candidates, as described in section 3. It was stated in section 3. that this could be computed in $O(n \log n)$ time. In this section the details of this algorithm are described.

One possibility to consider is to compute all segment intersections. Given a set of n line segments in the plane the algorithm of Bentley and Ottmann [Bentley and Ottmann] can be used to report all pairwise intersections, in $O(n \log n + k \log n)$ time, where k represents the number of pairwise intersections found. Unfortunately the number of pairwise intersections, may be large. In fact k may be as large as $O(n^2)$. In Figure 7 an example illustrating this phenomenon is shown. This example can be generalized, showing that as many as $O(n^2)$ intersections may occur.

It is not necessary to compute all pairwise segment intersections for the problem considered here. All that is required is to find candidates that are intersected by segments. Since there are a linear number of candidates the output is at most linear. Clearly we need not compute all $O(n^2)$ pairwise intersections.

Consider two sets of disjoint line segments A and B . It will be useful to be able to report in $O(n \log n)$ time all segments of A that are intersected by any segment of B .

An algorithm used to accomplish this, is based on the line sweep technique of Shamos and Hoey [Shamos and Hoey]. The algorithm scans a vertical line from left to right while maintaining a balanced tree that represents the order in the y direction of the segments intersected by the scanning line. Denote this as the y -order of the segments. The balanced tree allows insert and delete operations on the y -order in $O(\log n)$ time. Intersecting line segments will be adjacent in this ordering. The y -order changes when; the left endpoint of a segment is encountered, and the segment is inserted into the y -order; the right endpoint is encountered, and the segment is deleted from the y -order; or two segments cross thus interchanging their relative position in the y -order. In the problem of our concern, any time an intersection is found one of the intersected edges can be dispensed with. So the case of segments changing their relative position in the y -order does not occur. A pseudo code algorithm follows:

ALGORITHM SEGMENT INTERSECTION

step 1:

traverse the endpoints of A and B from left to right;

for each endpoint **do**

case left endpoint of segment s :

 insert s into y-order;

 Check if segments above or below in y-order intersect s ;

if intersection found **then begin**

 report intersection;

if $s \in B$ **then** $s \leftarrow$ edge $\in A$ that was intersected; **goto** step 2

end;

case right endpoint of segment s :

goto step 2;

step 2:

repeat

 remove s from y-order; test segments above and below it for intersection;

if intersection found **then** report intersection;

$s \leftarrow$ edge $\in A$ that was intersected;

until no intersection found;

goto step 1;

The proof of correctness and complexity of $O(n \log n)$ is a straight forward extension of the result of Shamos and Hoey. Now it will be shown how this algorithm can be applied to the candidate-segment intersection problem.

Recall there are segments $S = (s_0, s_1, \dots, s_{n-1})$, with endpoints in $P = (p_0, p_1, \dots, p_{2n-1})$, where each segment $\in S$ has at least one endpoint on $CH(P)$. As before assume that segments s_i and s_{i+1} , $i = 0, \dots, n-1$, (addition modulo n) are neighbors on $CH(P)$. Denote the endpoints of each segment s_i by $s_i = (s_{i_k}, s_{i_k})$ where s_{i_k} denotes an endpoint of S on $CH(P)$. The candidates considered for intersection can now be expressed as $C = C_0 \cup C_1 \cup C_2 \cup C_3$, where $C_0 = (s_{i_k}, s_{i+1_k})$, $C_1 = (s_{i_k}, s_{i+1_k})$, $C_2 = (s_{i_k}, s_{i+1_k})$, $C_3 = (s_{i_k}, s_{i+1_k})$, $i = 0, \dots, n-1$.

Candidates from C_0 do not have to be tested for intersection, since they are on the convex hull. Handling candidates from the other classes requires an examination of different cases of candidate intersections. The terminology of section three will be used to distinguish candidate intersections. It is easy to see that candidates from within the same class C_i , $i=1,3$ cannot intersect in a case 1 intersection. Blocking candidates, those candidates which intersect in a case 2 intersection can be predetermined and eliminated using the method suggested in section 3. Thus after all blocking candidates have been removed the only way two candidates from within the same class C_i can intersect is in a case 3 intersection.

Recall in lemma 3.3 it was shown that two candidates involved in a case 3 intersection necessarily intersect a segment $\in S$. Furthermore the segment $\in S$ is one of four segments namely the segments connected by the intersecting candidates. Therefore the decomposition of C into the classes C_1 , C_2 and C_3 can be used to determine candidate-segment intersections. We can use a slightly modified ALGORITHM SEGMENT INTERSECTION. With an input of candidates in C_i $i = 1,3$, and S any intersection found is either a candidate-segment intersection which can easily be handled, or a candidate-candidate intersection of case 3. We are assured one of these candidates also intersects a segment $\in S$, and in constant time we can determine this candidate. Any candidate-candidate intersection we may encounter is also a candidate-segment intersection and can be easily handled as such.

Therefore we can conclude that all intersections of candidates and segments can be determined in $O(n \log n)$ time.

An alternate method to compute candidate-segment intersections has been proposed by Suri [Suri]. By using a clever observation and the triangulation algorithm of Tarjan and Van Wyk, Suri can determine all candidate-segment intersections in $O(n)$ time.

In the next section the results of this paper are summarized.

6. Summary.

The main result of this paper is: Given a set of CH-connected segments S an $O(n \log n)$ algorithm is presented that returns a simple circuit on the segments, if such a simple circuit is admitted by S .

ALGORITHM SIMPLE CIRCUIT

Input : A set of segments S with endpoints P .

Output: A set of augmenting segments R , where $T = R \cup S$ represents a simple circuit. If there is no simple circuit on S then report this.

step 1:

 Compute the corresponding graph G and get G' 's subgraphs G_c , G_c' and G_c'' ;

step 2:

 Compute a maximal matching M in G_c' ;

if M is not a complete matching **then**

 Compute a maximal matching M in G_c'' ;

if M is a complete matching **then**

$E_c \cup M$ is a Hamiltonian circuit

 in G , and R corresponds to the edges M in G ;

otherwise report no simple circuit;

Theorem 6.1: Given a set S , of n CH-connected segments in the plane it can be determined whether S admits a simple circuit, in $O(n \log n)$ operations, and the circuit will be delivered in the same time bound.

The results of the previous sections lead up to the proof of this theorem. It will now be shown that:

Theorem 6.2: $O(n \log n)$ is necessary to deliver a simple circuit on a CH-connected set of segments.

Proof: The problem will be reduced to sorting real numbers. Given a set of n distinct reals, r_i , $i=0, \dots, n-1$. We can determine the minimum and maximum values, denoted by r_l and r_r , respectively, in $O(n)$ time. Construct n vertical line segments s_i , $i=0, \dots, n-1$, where s_i has endpoints $(r_i, 0)$, $(r_i, 1)$, except where $i = l, r$ the endpoints are $(r_l, 0), (r_l, 2)$ and $(r_r, 0), (r_r, 2)$. By inspection one sees that these segments are CH-connected and they admit a simple circuit. Traversing the segments in the order dictated by the augmenting segments, a cyclic permutation of the real numbers in sorted order, is obtained. It is well known that the lower bound for sorting is $O(n \log n)$. Therefore $O(n \log n)$ is necessary to deliver a simple circuit on a CH-connected set of segments. ■

Theorem 6.3 ALGORITHM SIMPLE CIRCUIT is optimal.

Proof: Follows immediately from theorems 6.1 and 6.2. ■

Acknowledgements

We are indebted to the attendants of a seminar in Computational Geometry, held at McGill in the fall of 1984, where this problem was originally discussed. In particular we thank Minou Mansouri who first showed the example in Figure 3, and Hossam ElGindy who inspired the concept of blocking segments. Finally a discussion between the first author and Ryan Hayward led to ALGORITHM MATCH.

References

[Bentley and Ottmann] Bentley, J. L., and T. A. Ottmann, "Algorithms for Reporting and Counting Geometric Intersections", *IEEE Transactions on Computers*, vol. c-28 No. 9, (1979), 643-647.

[Graham] Graham, R. L., "An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set",

Information Processing Letters, vol. 1, (1972), 132-133.

[Hopcroft and Karp] Hopcroft, J. E., and R. M. Karp, "An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs", *SIAM Journal on Computing*, vol. 2 (1973), 225 - 231.

[Rappaport] Rappaport, David, "Computing Simple Circuits on a Set of Line Segments is NP-Complete" *Technical Report No. SOCS-86.6* McGill University 1986.

[Shamos and Hoey] Shamos, M. I. and D. Hoey, "Geometric Intersection Problems", *Proceedings of the 17th FOCS*, Oct. 1976, 208 - 215.

[Toussaint] Toussaint, G. T., "A Historical Note on Convex Hull Finding Algorithms", *Technical Report No. SOCS-83.14* McGill University (1983).

[Suri] Suri, Subhash, *Personal communication*.

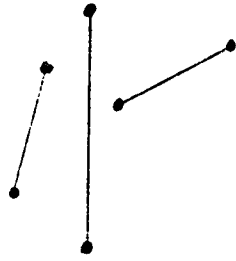


Figure 1.

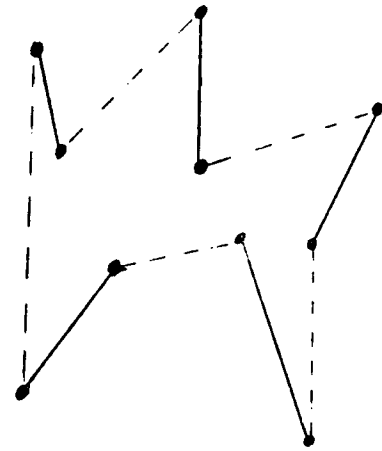


Figure 2.

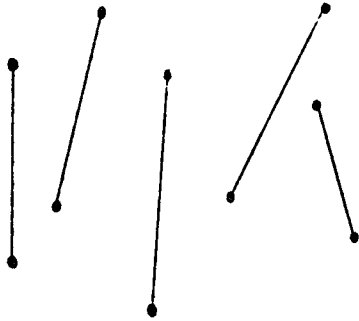


Figure 3.

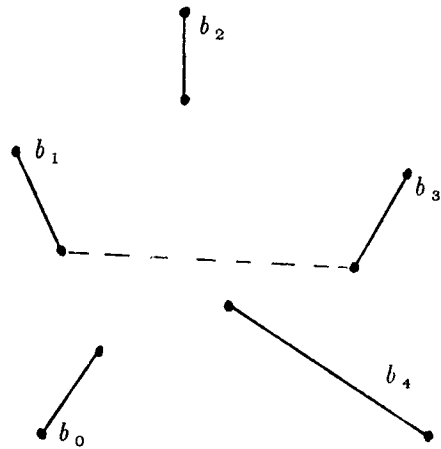


Figure 4.

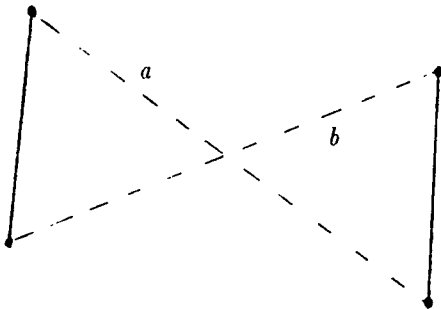


Figure 5.

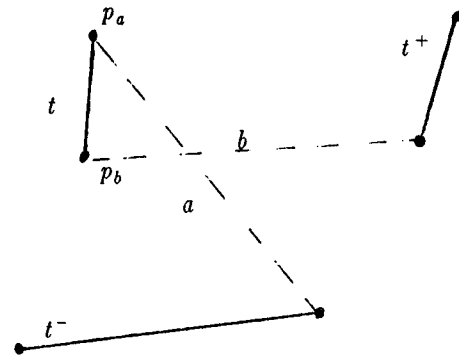


Figure 6.

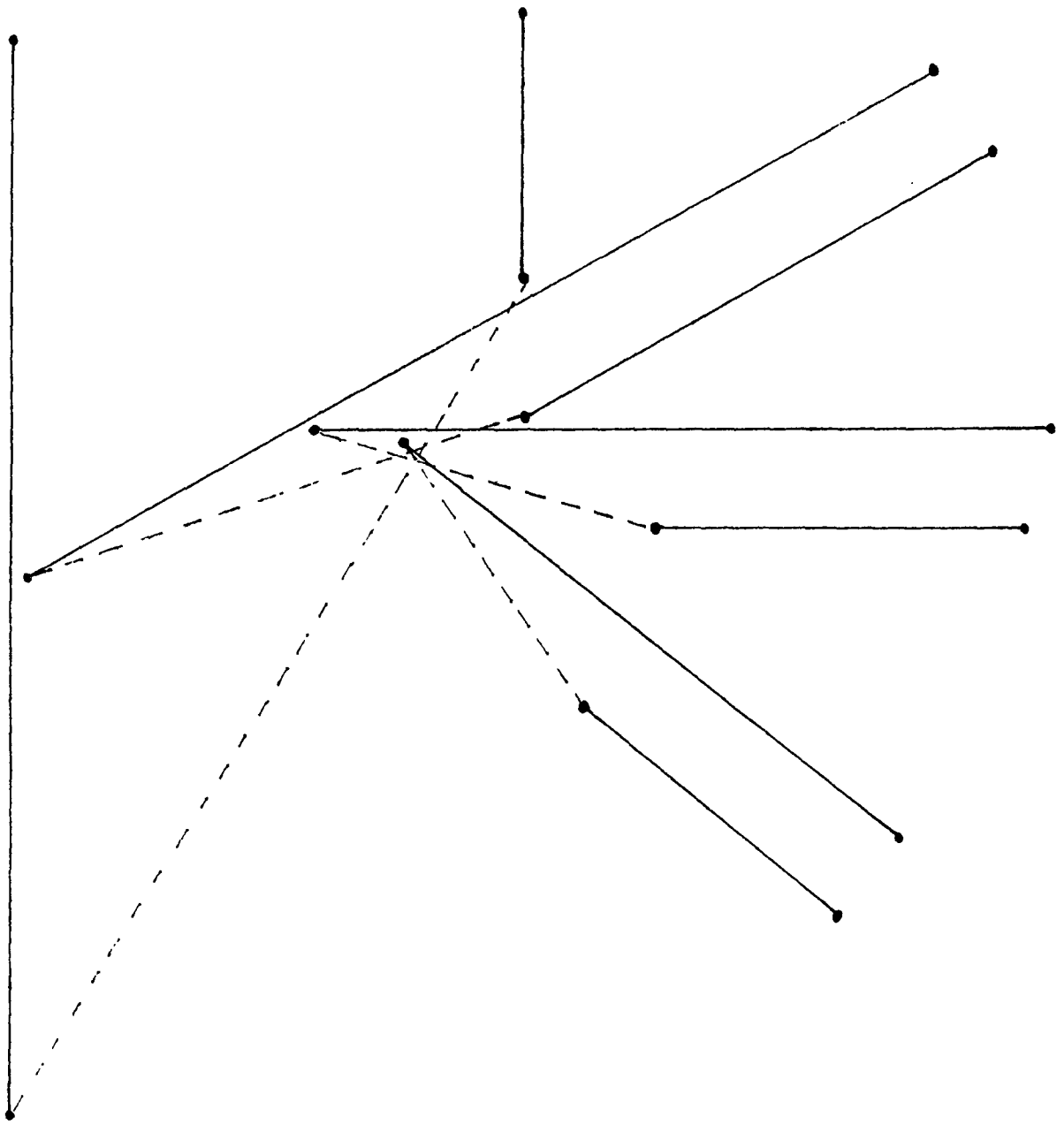


Figure 7.