Computing Simple Circuits from a Set of Line Segments

David Rappaport[*] *Hiroshi Imai*[†] *Godfried T. Toussaint*[‡]

Published in DISCRETE AND COMPUTATIONAL GEOMETRY, vol. 5, 1990, pp. 289-304.

Abstract

We address the problem of connecting line segments to form the boundary of a simple polygon — a simple circuit. However, not every set of segments can be so connected. We present an $O(n \log n)$-time algorithm to determine whether a set of segments, constrained so that each segment has at least one endpoint on the boundary of the convex hull of the segments, admits a simple circuit. Furthermore, this technique can also be used to compute a simple circuit of minimum perimeter, or a simple circuit that bounds the minimum area, with no increase in computational complexity.

1 Introduction

It is always possible to construct a simple polygon passing through every point of a planar point set, and this task can be accomplished in $\Theta(n \log n)$ time [12]. However, for a set of line segments, it is *not* always possible to obtain a simple circuit passing through every line segment. An example is shown in Fig. 1. If we can find a simple circuit that passes through every segment of a set of line segments, then we say that the set *admits* a simple circuit. In general it has been shown that to determine whether a set of line segments admits a simple circuit is NP-complete [10] and [11].

In [10] and [11] it is shown that deciding wheteher a set of segments admits a simple circuit is polynomially reducible to deciding whether a planar graph has a Hamiltonian circuit. A *Hamiltonian circuit* is a simple closed path through all the nodes of a graph. To determine whether a planar graph has a Hamiltonian circuit is an NP-complete problem [4]. Given a planar graph, a configuration of line segments is constructed in polynomial time so that a Hamiltonian circuit can be found in the graph if and only if the configuration of segments

[*]Department of Computing and Information Science, Queen's University, Kingston, Ontario, Canada K7L 3N6

[†]Department of Computer Science and Communication Engineering, Faculty of Engineering, Kyushu University, Hakozaki, Fukuoka 812 Japan

[‡]School of Computer Science, McGill University, 3480 University Street, Montreal, Quebec, Canada H3A 2A7

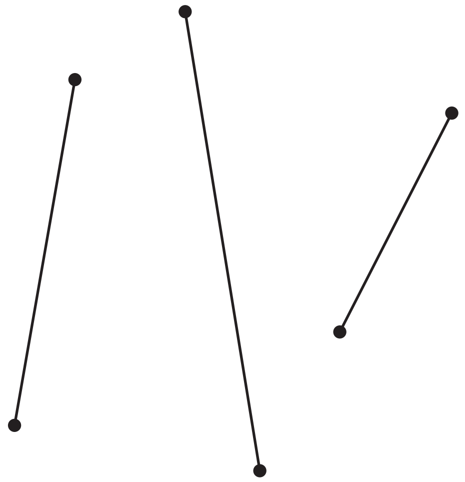

Figure 1:

admits a simple circuit. This construction requires that some line segments intersect at their endpoints. Therefore, for an input of line segments with disjoint closed intervals the NP-completeness result does not necessarily hold. To remain consistent with the results in [10] and [11] we allow the possiblity of some segments intersecting at their endpoints. We denote a sequence of segments with disjoint interiors but intersecting at their endpoints as a *chain of segments*.

We consider a constrained version of the simple circuit problem. An $O(n \log n)$-time algorithm is presented to determine whether a set of line segments, where each line segment has at least one endpoint on the boundary of the convex hull of the segments, admits a simple circuit. Furthermore, this technique can also be used to compute a simple circuit of minimum perimeter, or a simple circuit that bounds the minimum area, with no increase in computational complexity.

A set of line segments S is represented as $S = (s_0, s_1, \ldots, s_{n-1})$ (to be referred to from now on as segments). The endpoints of S are represented by the set of $2n$ points, $P = (p_0, p_1, \ldots, p_{2n-1})$. A *polygon* is defined as a sequence of distinct points in the plane, called *vertices*, where each consecutive pair x_i, x_{i+1} (modulo n) is connected by a straight line segment, or *edge*. A polygon is *simple* if no point in the plane belongs to more than two edges and the only points of the plane that do belong to more than one edge are endpoints of the edges, that is, the *vertices*. In the literature it is common for the term simple polygon to denote a boundary and the area it encloses, the *interior*. In order to clearly distinguish between the two, we use *simple circuit* to denote only the boundary of a simple polygon. We say that S admits a simple circuit if it is possible to construct a simple circuit $R \cup S$, where R is a set of nonintersecting segments whose endpoints *augmenting segments* of S. In Fig. 2 a set of segments is shown as solid lines with a set of augmenting segments as dashed lines. The convex hull of a set of points, P, is the smallest convex region enclosing P. Let $CH(P)$ denote the vertices (in clockwise order) of the convex hull of P. We define a set of segments

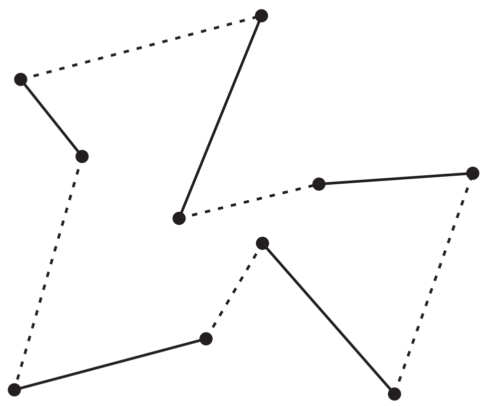

Figure 2:

as *CH-connected*, if for every segment $s \in S$, at least one of the endpoints of s is in $CH(P)$.

In Section 2 some preliminary observations regarding S are made. In Section 3 some geometric properties of S are established. We show that every simple circuit from a CH-connected set of segments passes through the segments in the same order that the segments appear on $CH(P)$. We also show that we can obtain a set of candidates that includes the set of all potential augmenting segments. A graph is then constructed to reflect these properties. In Section 4 a linear-time algorithm is introduced which finds, if there is one, a Hamiltonian circuit in the graph we have created. The structure of the graph allows us to find a Hamiltonian circuit by solving an easy matching problem. Section 5 relates some computational details regarding finding intersections of augmenting segments and the original segments S. The paper is summarized in Section 6 where a proof of optimality is given.

2 Preliminaries

Consider a set, S, of n line segments. There are some preliminary tests that can be made to determine whether S does not admit a simple circuit. It is obvious that if any pair of segments intersect in both their interiors, then clearly S cannot admit a simple circuit. Three further conditions that preclude the admission of a simple circuit are: if an endpoint of one segment meets the interior of another, if three or more segments meet at the same endpoint, or a subset of the segments intersect at their endpoints to form a simple circuit. We can test for the four forbidden conditions above in $O(n \log n)$ time, by using the line-sweep technique for determining line-segments intersections [13]. If any two segments intersect in their interiors we immediately reject the set. Similarly, if we encounter an endpoint at the intersection of three segments, or in the interior of another segment, we also reject the set. If the line-sweep procedure terminates without rejecting the segments, then we traverse the segments that intersect at their endpoints to determine whether a circuit is present. If any circuits are found we can again reject the set of line segments.

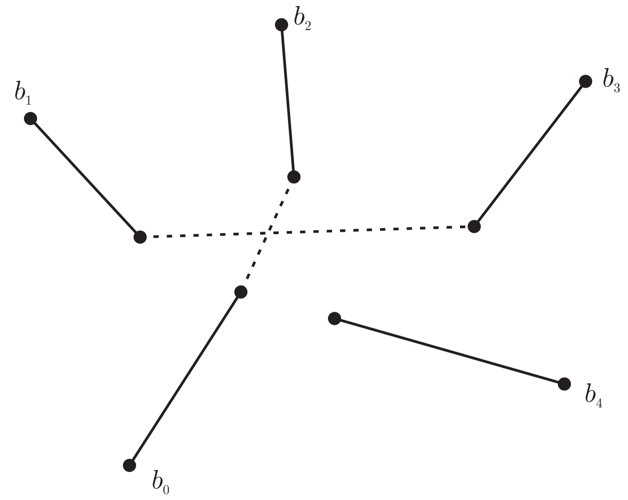

Figure 3:

There is yet another simple preliminary test that can be undertaken to determine if a set of segments does not admit a simple circuit, as suggested by the following lemma.

Lemma 2.1 *Let S be a set of segments that admits a simple circuit represented by a sequence of vertices $T = \{t_0, t_1, \ldots, t_{2n-1}\}$. Let $B = \{b_0, b_1, \ldots, b_m\}$ be the sequence of vertices representing $CH(T)$. B is a subsequence of every simple circuit T.*

Proof: Assume that B is not a subsequence of T, so at least one pair of points in the sequence B are interchanged. Without loss of generality we can assume that b_2 and b_1 are interchanged, and $\{b_0, b_2, b_1, b_3\}$ is a subsequence of T (see Fig. 3). Since $\{b_0, b_1, b_2, b_3\}$ is a subsequence of B, $\{b_0, b_1, b_2, b_3\}$ are the vertices of a convex quadrilateral Q. Let $C(x, y)$ denote the polygonal chain formed by a contiguous subsequence of T beginning at x and ending at y. The polygonal chains $C(b_0, b_2)$ and $C(b_1, b_3)$ must intersect the interior of Q, since every simple circuit is contained in $CH(P)$. Because b_0, b_2 and b_1, b_3 are diagonals of a convex quadrilateral, the polygonal chains $C(b_0, b_2)$ and $C(b_1, b_3)$ must intersect each other. Therefore $\{b_0, b_2, b_1, b_3\}$ cannot be a subsequence of a simple circuit, and therefore T cannot be a simple circuit, which is a contradiction. ∎

It should be noted that the above lemma is not restrictive to CH-connected sets of segments, but applies to all sets of segments. We say that S contains a *cutting chain*, if there exists a chain of segments in S, that intersects the interior of the convex hull of S, and also intersects two nonadjacent vertices on the boundary of the convex hull of S. As a direct consequence of Lemma 2.1 a set of segments that contains a cutting chain does not admit a simple circuit. An algorithm to determine whether a set of segments contains a cutting chain is readily available. Given S and P, we first compute $CH(P)$. A linear scan can then test all chains of segments (as computed above) to see if any are cutting chains. The convex

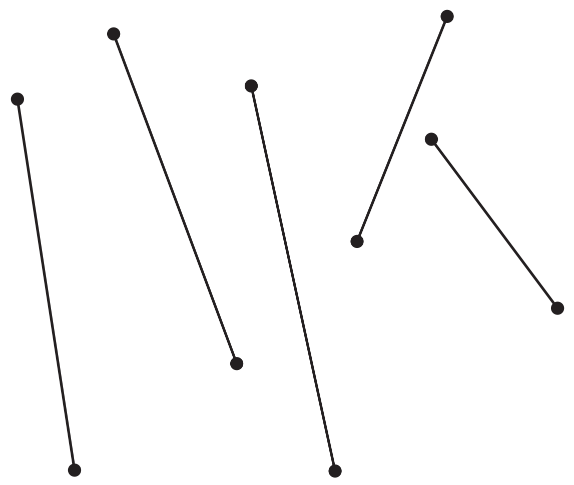

Figure 4:

hull of a set of n points can be computed in $O(n \log n)$ time [5] or $O(n \log h)$ time [8] and [9] where h is the cardinality of the convex hull points. See [16] for a historical account of the planar convex hull problem.

We have given some preliminary methods to reject sets of segments that do not admit a simple circuit. However, a set of segments may pass the above tests and still not admit a simple circuit. A set of five CH-connected segments that illustrate this phenomenon is shown in Fig. 4. We assume in the ensuing discussion that S is a set of four or more CH-connected segments such that:

1. the interiors of the segments are disjoint,
2. no endpoint of one segment lies in the interior of another,
3. no more than two segments meet at an endpoint,
4. no circuits are formed by subsets of the segments, and
5. there are no cutting chains.

3 Geometric Results

We examine the geometric properties of S, to arrive at an appropriate graph representation G. We then solve a combinatorial problem using G to determine whether our original segments S admit a simple circuit. The vertices of G correspond to the endpoints of S. For every endpoint p_i of a segment there is a corresponding vertex v_i in G. The edges in G are a combination of edges corresponding to the segments S, and edges corresponding to a set of candidates for augmenting segments. Denote these edges as E_s and E_c, respectively. In

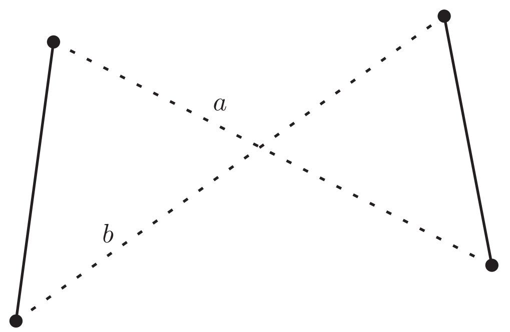

Figure 5:

this section we show how an appropriate set of candidates leading to the edges E_c, can be obtained from S. Denote by an E_s-*required* Hamiltonian circuit a Hamiltonian circuit of G that passes through every edge in E_s. The edges E_c are chosen so that we will be able to prove that S admits a simple circuit if and only if G has an E_s-required Hamiltonian circuit.

Initially we can consider as candidates all segments $P \times P$ such that $(p_i, p_j) \notin S$ and $p_i \neq p_j$. However, this is far too many to consider. Denote segments of S whose endpoints are adjacent on $CH(P)$ as neighbors. It is convenient to label the segments of S so that s_i is a neighbor of s_{i+1} for all $i = 0, \ldots, n-1$ (addition modulo n). As a consequence of Lemma 2.1 we see that augmenting segments must connect the endpoints of neighbors. Therefore, we only have to consider edges connecting neighboring segments as *candidates*. This pool of $O(n)$ candidates is further reduced. Consider the case where two segments intersect at their endpoints. Clearly, the only admissible candidate between two segments that meet at an endpoint is the degenerate candidate formed by the coincident endpoints. Including any other candidates connecting such segments is redundant, so we call these candidates *redundant candidates*. All redundant candidates can be eliminated. We can also eliminate any candidate that intersects the interior of any segment in S. Denote these as *segment-intersecting candidates*. 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 [13] and a careful decomposition of the segments 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. We must also ensure that if an E_s-*required* Hamiltonian circuit is found in G, then no pair of edges from E_c correspond to a pair of candidates that intersect. It is useful to distinguish between three types of these intersections.

Let a and b be two candidates that are not redundant or segment-intersecting.

Case 1. All four endpoints of candidates a and b are endpoints of only two of the segments of S (see Fig. 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

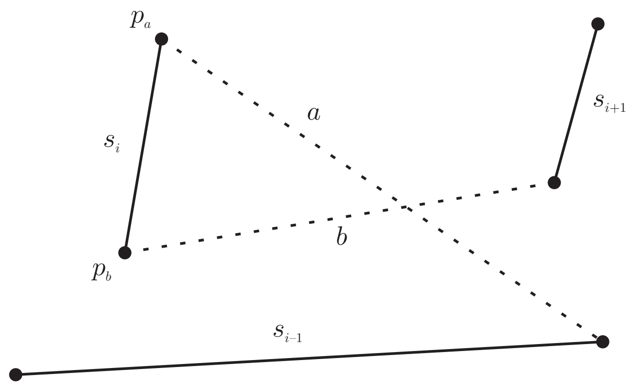

Figure 6:

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 Fig. 6). Therefore, one of the segments of S has a candidate at both of its endpoints. It will be shown that we can eliminate at least one of these candidates. Consider a candidate c connecting segments s_i and s_{i-1}. Let σ_i and σ_{i-1} be the respective endpoints of s_i and s_{i-1} that are not endpoints of c. If every candidate adjacent to σ_i that connects s_i to s_{i+1} intersects c, or if every candidate adjacent to σ_{i-1} that connects s_{i-1} to s_{i-2} intersects c, then call c a blocking candidate. In Fig. 6, a is a blocking candidate. Obviously a blocking candidate cannot be an augmenting segment. Every blocking candidate can be identified in constant time. Therefore, all blocking candidates can be eliminated from the pool of candidates in $O(n)$ time.

Lemma 3.1 *If two candidates a and b intersect and both of them are incident to the same segment, then either a or b (or both) is a blocking candidate.*

Proof: Let s_i denote the segment incident to both a and b, with p_a being the endpoint of s_i on a, and p_b being the endpoint of s_i on b (see Fig. 6). At least one endpoint of s_i is on $CH(P)$, so one endpoint of a or b (or both) must also be on $CH(P)$. Without loss of generality assume p_a is on $CH(P)$, and the other endpoint of a is on s_{i-1}. Because a and b intersect they cannot be edges of $CH(P)$. Observe that the candidates connecting p_b with each of the endpoints of s_{i+1} must intersect a. Therefore, a is a blocking candidate. ■

Case 3. The four endpoints of a and b lie on four different segments of S.

Lemma 3.2 *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: 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} (see Fig. 7). Let h_i denote the convex hull edge from s_i to s_{i+1}, and

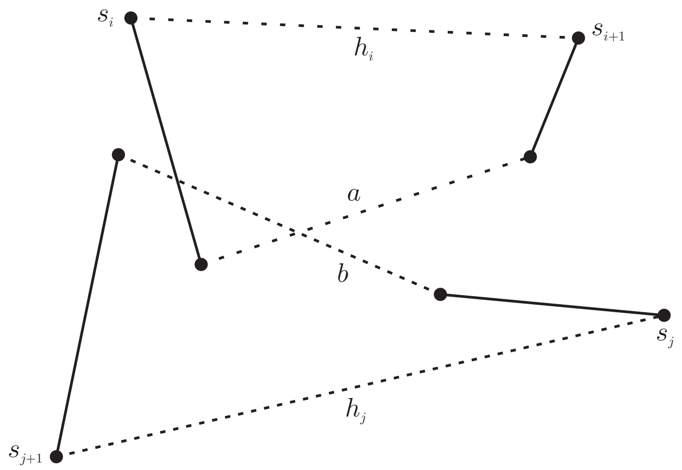

Figure 7:

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 affect the argument.) Two intersecting circuits 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. ∎

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 are not segment-intersecting, are not redundant and not blocking. The edges in G we call E_c correspond to the set of candidates C. The edges of G are $E = E_c \cup E_s$.

Lemma 3.3 *The segments S admit a simple circuit if and only if $G = (V, E)$ has an E_s-required Hamiltonian circuit.*

Proof: Assume S admits a simple circuit. It is required to show that the augmenting segments have their counterparts in G. From the preceding lemmas we know that all augmenting segments connect neighbors, and are not redundant nor blocking segments. Every edge with these properties has been included in G, therefore G must have an E_s-required Hamiltonian circuit if S admits a simple circuit.

On the other hand, assume G has an E_s-required Hamiltonian circuit. Clearly, every E_s-required Hamiltonian circuit in G is a circuit in S. It remains to show that the circuit is simple. Assume the circuit obtained results in a nonsimple circuit. No segments of S and C can cross. If two segments from C cross then they cross in a Case 1, Case 2, or Case 3 intersection as defined above. However, as was shown above, a Case 1 intersection cannot

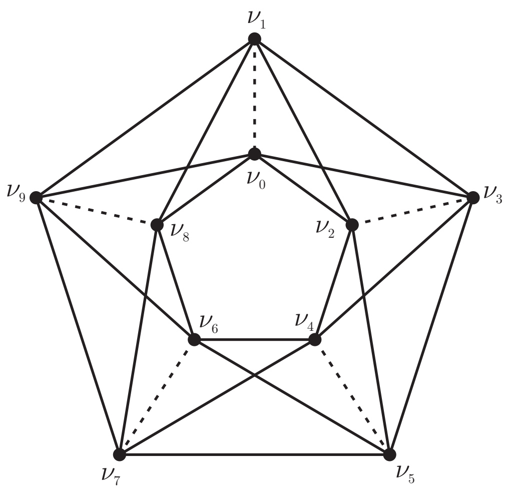

Figure 8: A ten vertex example of the graph G. Dashed edges represent E_s. E_c is a subset of the solid edges.

occur, a Case 2 intersection is prevented by eliminating blocking candidates, and a Case 3 intersection is prevented by eliminating any candidate intersecting a segment. Therefore the circuit must be simple. ∎

In general, it is an NP-complete problem to determine whether there is a Hamiltonian circuit in a graph [7]. 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. These polynomial-time algorithms rely on the special structure of the graph G.

4 Matchings that Lead to Simple Circuits

Let $G = (V, E)$ be a graph as obtained in the previous section, where $V = (v_0, v_1, \ldots, v_{2n-1})$. We denote $E_s = \{(v_{2i}, v_{2i+1}) \mid i = 0, \ldots, n-1\}$ as edges of G corresponding to S and E_c as the edges of G corresponding to C. The edges of E_c are a subset of edges connecting endpoints of s_i to s_{i+1}. See Fig. 8.

Denote G_c as 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), (v_1, v_{2n-1}), and (v_1, v_{2n-2}). Call the resulting graph $G_c' = (V, E_c')$. We also need a subgraph of G_c, G_c''. This graph contains the edges E_c with (v_1, v_2), (v_1, v_3), (v_0, v_{2n-1}), and (v_0, v_{2n-2}) removed (if they exist). We use the graphs G_c' and G_c'' to find an E_s-required Hamiltonian circuit in G.

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_i 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 contained 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. ∎

If both G'_c and G''_c do not contain a complete matching then we can conclude that G_c does not have an E_s-required Hamiltonian circuit. Otherwise, we can use a complete matching from G'_c or G''_c to construct an E_s-required Hamiltonian circuit as suggested by Theorem 4.1. There are several algorithms found in the literature that can be used to compute the maximum matchings in G'_c and G''_c. In particular, since these graphs are bipartite graphs (a bipartite graph is defined as a graph whose vertices can be divided into two disjoint subsets, such that every edge in the graph has an endpoint in each subset) we can use an efficient method based on a network flow algorithm to find maximal matchings in $O(|V|^{1/2}|E|)$ time [6]. In the problem cinsidered here the edges and the vertices are both of cardinality $O(n)$ so the running time is $O(n^{3/2})$. However, the structure of the graphs G'_c and G''_c permit a more efficient method to determine whether there is a complete matching. This algorithm will now be discussed.

Construct a weighed graph $G_* = (V, E_*)$, where V is defined as above and E_* consists of all edges connecting endpoints of s_i to s_{i+1}. We assign weights $w(e) = 1$ to an edge $e \in E_*$, if e is also contained in E'_c, and $w(e) = 2$ if $e \notin E'_c$. The minimum weight complete matching in G_* is computed in stages. Let $\omega[i]$ denote the cost of the minimum weight matching using the vertex v_i in $G_* - \{v_{i+1}, v_{i+2}, \ldots, v_{2n-1}, v_0\}$. The following lemma leads to a simple dynamic programming algorithm.

Lemma 4.2 *The minimum weight matching in G_* is*

$$\min(\omega[2n-1] + w(v_{2n-2}, v_0), \omega[2n-2] + w(v_{2n-1}, v_0))$$

Proof: Since v_0 is only adjacent to v_{2n-1} and v_{2n-2}, every candidate matching must contain either the edge (v_{2n-1}, v_0) or the edge (v_{2n-2}, v_0). The result follows. ∎

The following algorithm is an iterative implementation of the dynamic programming algorithm suggested by Lemma 4.2.

Algorithm MATCH
$\omega[2] \leftarrow w(v_1, v_2)$;
$\omega[3] \leftarrow w(v_1, v_3)$;
for $i \leftarrow 2$ **to** $n-1$
    do begin
    $\omega[2i] \leftarrow \min(\omega[2i-1] + w(v_{2i-2}, v_{2i}), \omega[2i-2] + w(v_{2i-1}, v_{2i}))$;
    $\omega[2i+1] \leftarrow \min(\omega[2i-1] + w(v_{2i-2}, v_{2i}), \omega[2i-2] + w(v_{2i-1}, v_{2i}))$
    end;
$\omega[0] \leftarrow \min(\omega[2n-1] + w(v_{2n-2}, v_0), \omega[2n-2] + w(v_{2n-1}, v_0))$;

The cost of the minimum weight matching is kept in $\omega[0]$. Correctness follows from Lemma 4.2, and the linear running time is obvious. If the cost is n then a complete matching exists in G'_c. Otherwise there is no complete matching. The algorithm OBTAIN-M can be used to obtain, in $O(n)$ time, the edges of the complete matching $M = (m_1, m_2, \ldots, m_n)$. The following algorithm follows the approach used in the proof of Lemma 4.2.

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

Assume a complete matching in G_* has been found. As a direct consequence of the previous result we can determine a minimum perimeter simple circuit from S. A weighted graph $G = (V, E_l)$ is constructed, where E_l is the same as E_* except for the weight assignments. Assign the Euclidean length of a candidate as the weight given to the corresponding edge in $E_l \in E'_c$. For any edge in $E_l \notin E'_c$ assign a weight of ∞. A minimum perimeter simple circuit corresponds to a minimum weight complete matching in G_l.

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 circuit P is the area of $CH(P)$ less the sum of the areas of the polygonal regions that constitute the difference between $CH(P)$ and P. Denote the polygonal regions that constitute the difference between $CH(P)$ and P as *convex deficiency circuits* of P. The convex deficiency circuits of every simple polygon from 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 circuit.) Therefore, every candidate describes a unique convex deficiency circuit. Assign weights to G to obtain 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 circuit described by that candidate. For edges not in 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.

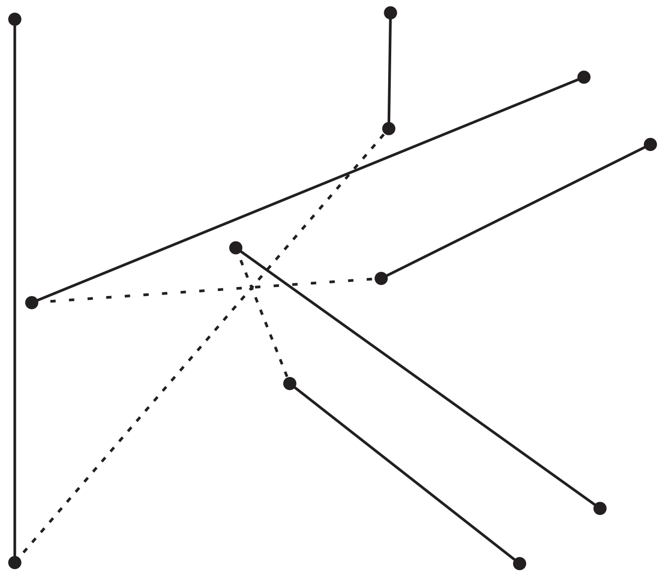

Figure 9:

5 Finding Intersections of Candidates and Segments

As described in Section 3, a necessary step to obtain the final set of candidates is to determine candidate-segment intersections. 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 [2] 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 $O(n^2)$. An example illustrating this phenomenon is shown in Fig. 9. 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. We only need to find candidates that are intersected by segments. Since there are a linear number of candidates the output is at most linear. We need not compute all $O(n^2)$ pairwise intersections.

Consider two sets of line segments A and B, where the interiors of the segments in A are pairwise disjoint, and so are the interiors of the segments in 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 [13]. 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 $O(n \log n)$ 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 our problem, any time an intersection is found, one of the intersected edges can be dispensed with. Therefore, the case of segments changing their relative position in the y-order does not occur. This observation leads to an algorithm that is a straightforward extension of the result of Shamos and Hoey [13]. We now show how this algorithm can be applied to the candidate-segment intersection problem.

Let the endpoints of each segment s_i be denoted by s_{i_h} and s_{i_k} where s_{i_h} 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 $C_0 = \{(s_{i_h}, s_{i+1_h}) | i = 0, \ldots, n-1\}$, $C_1 = \{(s_{i_k}, s_{i+1_k}) | i = 0, \ldots, n-1\}$, $C_2 = \{(s_{i_k}, s_{i+1_h}) | i = 0, \ldots, n-1\}$, and $C_3 = \{(s_{i_h}, s_{i+1_k}) | i = 0, \ldots, n-1\}$. 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 the different types of candidate intersections. The terminology of Section 3 is used to distinguish candidate intersections. It is easy to see that candidates from within the same class $C_i, i = 1, 2, 3$, 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 that in Lemma 3.2 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. With an input of candidates in $C_i, i = 1, 2, 3$, and S, any intersection found is either a candidate-segment intersection which can be easily handled, or a candidate-candidate intersection of Case 3. We are assured one of these candidates also intersects a segment of 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 [14]. Suri has observed that candidates for augmenting segments are a subset of any triangulation of the line segments. If $CH(P)$ is known, the candidates can be obtained in $O(n \log \log n)$ time by applying the triangulation algorithm of Tarjan and Van Wyk [15]. As is shown in the next section $\Omega(n \log n)$ is required for obtaining a simple circuit from a set of CH-connected line segments. Therefore, the contribution of Suri cannot affect the overall complexity of the algorithm unless the segments are already given in their convex hull order.

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

6 Computational Complexity

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 from 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, then report this.
Compute the corresponding graph G and get the subgraphs of G, G_c, G'_c, and G''_c;
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**
$E_s \cup M$ is a Hamiltonian circuit in G, and R corresponds to the edges M in G;
otherwise report no simple circuit;

The results of the previous sections lead to the following theorem.

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.*

A lower bound for delivering a simple circuit from a set of CH-connected segments is given.

Theorem 6.2 $\Omega(n \log n)$ *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, $r_i, i = 0, \ldots, n-1$, 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 $S_i, i = 0, \ldots, n-1$, where S_i has endpoints $(r_i, 0), (r_i, 1)$, except where $i = l$ or r the endpoints are $(r_l, 0), (r_l, 2)$ and $(r_r, 0), (r_r, 2)$. By inspection we see that these segments are CH-connected and they admit a simple circuit. A cyclic permutation of the real numbers in sorted order is obtained by traversing the segments in the order dictated by the augmenting segments. In [3] it is shown that $\Omega(n \log n)$ is a lower bound for sorting if comparisons between arbitrary functions are allowed. Quadratic functions can be used to handle the two-dimensional problems discussed in this section. Therefore, $\Omega(n \log n)$ is necessary to deliver a simple circuit on a CH-connected set of segments. ■

Thus we have shown that our algorithm is optimal. It should be noted that although we have given a lower bound for the case that the line segments are given in arbitrary order, the lower bound does not hold if the segments are given convex hull order. We may argue that since we know that the segments are CH-connected, it is also likely that we have the segments in order. As was pointed out in the previous section an $O(n \log \log n)$ algorithm is possible if the segments are indeed given in order. It would be interesting to determine whether we could do better under those circumstances.

7 Discussion

The paradigm followed in this paper has been to take a geometric problem and convert it to a combinatorial problem by extracting the essential geometric properties and then dealing with the problem in a more "pristine" combinatorial setting. This technique may have introduced some unnecessary complexity during the segment to graph conversion discussion, since in principle we could have used the same methods directly on the segments. However, in the transformed domain all of the geometric issues can be ignored resulting in a very compact algorithm. By using weighted graphs, obtaining the optimum perimeter and optimum area simple circuits is achieved with very little additional work.

Other problems concerning simple circuits from line segments are explored in [1] and [10]. Constrained versions of this problem are examined where polynomial solutions exist. In particular, it is shown that to determine if a set of line segments admits a monotone or star-shaped simple circuit is polynomial. It is also shown that optimum (with respect to area and perimeter) monotone or star-shaped simple circuits from a set of line segments can be obtained in the same time bounds.

Acknowledgments

We are indebted to the attendants of a seminar in Computational Geometry held at McGill University in the fall of 1984, where this problem was originally introduced by the third author. In particular, we thank Minou Mansouri who first showed an example similar to the one in Fig. 4, and Hossam ElGindy who inspired the concept of blocking segments. Finally, a discussion between the first author and Ryan Hayward led to Algorihtm MATCH.

References

[1] D. Avis and D. Rappaport. Computing monotone simple circuits in the plane. In G. Toussaint, editor, *Computational Morphology*, pages 13–23. North-Holland, Amsterdam, 1988.

[2] J.L. Bentley and T.A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, 28(9):643–647, 1979.

[3] N. Friedman. Some results on the effect of arithmetics on comparison problems. In *Proc. 13th IEEE Symp. Switching Automata Theory*, pages 139–142, 1972.

[4] M.R. Garey, D.S. Johnson, and R.E. Tarjan. The planar hamiltonian circuit problem is NP-complete. *SIAM J. Comput.*, 5:704–714, 1976.

[5] R.L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Inform. Process. Lett.*, 1:132–133, 1972.

[6] J.E. Hopcroft and R.M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2:225–231, 1973.

[7] R.M. Karp. On the complexity of combinatorial problems. *Networks*, 5:45–68, 1975.

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

[9] M. McQueen and G.T. Toussaint. On the ultimate convex hull algorithm in practice. *Pattern Recognition Lett.*, 3:29–34, 1985.

[10] D. Rappaport. *The Complexity of Computing Simple Circuits in the Plane*. PhD thesis, McGill University, 1986.

[11] D. Rappaport. Computing simple circuits from a set of line segments is NP-complete. In *Proc. 3rd ACM Symp. Comput. Geom.*, pages 322–330, 1987.

[12] M.I. Shamos. Geometric complexity. In *Proc. 7th ACM Annu. Symp. Theory Comput.*, pages 224–233, 1975.

[13] M.I. Shamos and D. Huey. Geometric intersection problems. In *Proc. 17th IEEE Annu. Symp. Found. Comput. Sci.*, pages 208–215, 1976.

[14] S. Suri. Personal communication, 1986.

[15] R.E. Tarjan and C. Van Wyk. An $O(n \log\log n)$ algorithm for triangulating a simple polygon. *SIAM J. Comput.*, 17:143–178, 1988.

[16] G.T. Toussaint. A historical note on convex hull finding algorithms. *Pattern Recognition Lett.*, 3:21–28, 1985.