Some Problems in Distributed Computational Geometry

(Extended Abstract)

Sergio Rajsbaum*   Jorge Urrutia†

February 14, 2004

Abstract

A *geometric network* is a distributed network where each processor is identified by two numbers, representing the coordinates of the point in the plane where the processor is located. The edges of the network correspond to straight line segments such that no two of them intersect. In this paper we introduce the study of distributed computing in geometric networks. We study several computational geometry problems from the distributed computing point of view, such as finding convex hulls of geometric networks and identification of the external face. In particular, we obtain a $O(n \log^2 n)$ message complexity algorithm to find the convex hull of a planar geometric graph, and a $O(n \log n)$ algorithm to identify the external face of a geometric graph. We also prove that the message complexity of leader election in an asynchronous geometric ring of n processors is $\Omega(n \log n)$.

1 Introduction

A *geometric network* is a distributed network in which each processor has a fixed location on the plane, and its edges are represented by *non-crossing* straight line segments. There are many papers that investigate the complexity of various specific distributed problems. Of special interest is the *message complexity* of an algorithm, which is the total number of messages

*Contact Author. Instituto de Matemáticas, UNAM, Ciudad Universitaria, D.F. 04510, México. email: `rajsbaum@servidor.unam.mx`.

†Supported by NSERC, Canada. SITE, University of Ottawa, Ottawa ON Canada

sent in the worst case during any execution of the algorithm. It turns out that the complexity of solving a problem depends critically on various specific, and sometimes subtle network assumptions, such as, for example, the degree of asynchrony, or the type of failures. The knowledge that processors have about the network can dramatically affect the complexity of a problem; for example, knowledge about its topology (e.g. ring, hypercube), sense of orientation, the number of processors, etc. Detailed treatments and references can be found in textbooks such as [3, 5]. In the asynchronous, failure-free setting, it is known that $\Theta(m + n \log n)$ messages are necessary and sufficient to elect a leader in an arbitrary network of n processors and m links. The same result holds for rings. In this paper we initiate the study of distributed algorithms on geometric networks, and investigate the impact of geometric information on the complexity of the leader election problem. We also obtain distributed algorithms to calculate the convex hull of a geometric graph, as well as for the identification of internal and external faces of a geometric network. We concentrate on the message complexity of asynchronous, failure-free, uniform (processors do not know the size of the network) algorithms. We assume each processor in the network knows the coordinates of a point that represents its position in the plane, and that no two processors are at the same location.

Our motivation to study distributed computational geometry, arises in part from the fact that in real-life networks, computers (the processors of a distributed network) are located at some specific place. Thus computer networks can be modeled as geometric networks. It is also interesting to note that, some of the best network topology maps used by Internet Service Providers and Internet Backbone Networks, such as TEN-34, EuropaNET, Eunet, Qwest Nationwide Network, etc. can be modeled as planar or almost planar graphs; see [1].

From a theorethical point of view, the development of algorithms to calculate fundamentals in computational geometry, such as convex hulls and the faces of a planar map, are basic problems to be addressed in this context.

Results

In Section 3 we prove a lower bound of $\Omega(n \log n)$ on the message complexity of leader election in geometric rings. Since there are $O(n \log n)$ algorithms for rings (not geometrical), this bound is tight, and the geometric information does not help to reduce the message complexity of this problem.

In Section 4, we present a lower bound of $\Omega(n \log n)$ on the message complexity of the external face problem in geometric rings. Then we show that the external face problem can be solved in $O(n \log n)$ messages in a general geometric network. We prove this by showing that if there is already a leader in a geometric network, it takes $O(n)$ messages to solve the external face problem.

Both of our lower bounds hold even if the network lies on a grid; that is, the positions of the processors in the plane are integers and each line segment joining two processors is horizontal or vertical. It will follow from our proof that our result holds even for rings located on grids of area $O(n \log n)$.

We remark that our lower bound for leader election holds even if the processors know the external face of the ring. Therefore, although both problems have message complexity of $\Theta(n \log n)$, in a sense, leader election is strictly harder than external face; if there is a leader, $O(n)$ messages are needed to find the external face, but if the external face is known, $\Omega(n \log n)$ messages are needed to elect a leader.

Finally, in Section 5, for the convex hull problem in geometric networks, we give a $O(n \log^2 n)$ message algorithm which sends only a constant number of identifiers in each message. Notice that once the convex hull has been solved, it takes only $O(n)$ messages to elect a leader, and hence also to find the external face.

Summarizing, we prove that geometric leader election is strictly harder than external face and that convex hull is at least as hard as geometric leader election.

In Section 6 we discuss some issues related to the geometric model and some open questions.

2 Preliminaries

We consider standard asynchronous, failure-free networks where the messages take a finite but arbitrary time to traverse a link [3, 5].

The *message complexity* of a distributed algorithm is the worst-case number of messages sent by the algorithm. For the upper bounds we assume that each message contains a constant number of processor identifiers, otherwise any distributed problem could be solved by first electing a leader, which in turn would gather the topology of the network and solve the problem locally. In our algorithms, each message will have two parts; a constant-length list of processor identifiers, followed by $O(\log n)$ bits. The lower bounds we present hold even if the messages are of unbounded size.

A *geometric network* is a distributed network in which each processor p knows its position on the plane determined by an ordered pair (p_x, p_y) of numbers called the *identifiers* of p. No two processors have the same identifiers. The communication links are straight lines, and no two lines intersect, other than perhaps at their end points.

We will use the partial order "$\prec$" defined on the processors of our networks as follows. Given two processors, p and q,

$$p \prec q \,\text{if}\, p_y < q_y \,\text{or else if}\, p_y = q_y \,\text{and}\, p_x < q_x.$$

Notice that under $\prec$ any two processors are comparable, and thus there is a unique largest processor with respect to $\prec$.

3 Leader Election In Geometric Networks

We start by considering the case of a geometric *convex* ring. Then we deal with arbitrary rings. A ring is called *convex* if its processors are located at the vertices of a convex polygon. We now show that in convex rings, electing a leader can be done using at most $2n$ messages.

Consider the order $\prec$ defined above on the processors of a convex ring C. Let p be a processor of C, and let q and r be its neighbours. Since the ring is convex, p is the maximal processor with respect to $\prec$ if and only if $q \prec p$, and $r \prec p$. Our election algorithm proceeds as follows: A processor either wakes up spontaneously, or upon receiving a message from any of its neighbours. When a processor p wakes up, it sends a message to its two neighbors, say q, r, asking them for their identifiers. Once p obtains this information, it elects itself as leader iff $p \prec q$ and $p \prec r$. It is easy to see that the total number of messages used by this algorithm is $2n$.

At this point, it is a natural question to ask if the additional information provided in geometric rings can be used to elect a leader in better than $O(n \log n)$ messages. As mentioned in the introduction, a leader can be elected in a ring, and hence also in a geometric ring with, $O(n \log n)$ messages. We now show that any asynchronous algorithm to elect a leader in geometric rings has an execution where $\Omega(n \log n)$ messages are sent. The argument incorporates geometry into the classical proof of Burns [2] (see also [3]). Thus, as in [2], we assume *uniform* algorithms that have to work for any ring size n. For the proof, we require that at the end of a leader election algorithm every processor knows the id (x, y) of the processor p with largest (w.r.t. $\prec$) id, so that p is considered the leader. Notice that this lower bound also implies that $\Omega(n \log n)$ messages are needed to elect any leader, with the requirement that only a single processor needs to know the leader; if there exists an algorithm sending fewer messages, once this algorithm terminates, the leader can send a message around the ring to find out the coordinates of the largest processor, and then send another message around the ring to distribute this information; this adds no more than $2n$ messages to the message complexity of the original algorithm.

Consider any leader election algorithm A. The idea of Burns' proof is to consider executions of A that send many messages without communicating with part of the ring. Take a segment (roughly half) of a ring, containing k processors, with endpoints p, q, with *open* edges e_1, e_2 connecting the segment to the rest of the ring. Prove, by induction, that there is an *open* execution for the segment in which $\Theta(k \log k)$ messages are sent, but no message is delivered along e_1, e_2. Then, consider an execution that consists of the open executions for each half of the ring, and show how to force an additional linear number of messages to be sent, in an open execution.

Let C be a geometric ring. We say that C is an *orthogonal* ring if the edges of C are horizontal or vertical line segments, and its processors have integer coordinates. Given an orthogonal ring C, let $R(C)$ be the smallest orthogonal rectangle containing it, and let $n(C)$, $h(C)$, and $w(C)$ respectively be the number of processors in C, the height, and the the width of $R(C)$. We also call $h(C)$, and $w(C)$ the height and width of C

In order to make this argument work, for each point (i, j) on the plane with integer coordinates, we construct recursively a family of orthogonal rings $F_k(i, j)$, $k \geq 0$, which satisfy the following conditions:

1. For every pair of integers (i, j), all the elements of $F_k(i, j)$ have the same width and height, denoted by w_k and h_k respectively. Moreover, if $C \in F_k(i, j)$ and $C' \in F_k(k, l)$ then $n(C) = n(C') = n_k$.

2. For each pair of elements C and C' of $F_k(i, j)$, $R(C) = R(C')$.

3. Each orthogonal ring C in $F_k(i, j)$ has exactly one edge in the bottom and top sides of $R(C)$, called the *top* and *bottom* edges of C. These are the potential open edges (in Burns' proof) of the ring.

4. n_k, w_k, and h_k satisfy the following equations:

   - $n_k = 2n_{k-1} + 8$, with $n_0 = 4$.
   - $h_k = h_k + 4$ with $h_0 = 1$.

- $w_k = 2w_{k-1} + 3$ with $w_0 = 1$.

For every pair of integers (i,j), let $C_0(i,j)$ be the unit square, with four processors in its corners, and let the top and bottom edges of $C_0(i,j)$ be the the top and bottom sides of $C_0(i,j)$ respectively. Let $F_0(i,j) = \{C_0(i,j)\}$.

Having constructed $F_{k-1}(i,j)$ for every (i,j), we now show how to construct $F_k(i,j)$; $i,j \in I$.

Consider the family of rings $F_k(i, j+2)$ and $F_k(i + w_{k-1} + 3, j+2)$, and let $C_1 \in F_k(i, j+2)$ and $C_2 \in F_k(i + w_{k-1} + 3, j+2)$.

Using C_1 and C_2 we obtain four elements in $F_k(i,j)$ called $C_{1,2}(top, top)$, $C_{1,2}(top, bottom)$, $C_{1,2}(bottom, top)$, and $C_{1,2}(bottom, bottom)$ as follows:

First remove the top and bottom edges of C_1 and C_2 respectively, and join the endpoints of these edges by two non-intersecting paths of length 5 as shown in Figure 1(a). C_1 and C_2 are not drawn, only their boxes, represented by dotted squares. Let $C_{1,2}(top, bottom)$ be the resulting ring. The edges e_1 and e_2 are the top and bottom edges $C_{1,2}(top, bottom)$.

To obtain $C_{1,2}(top, top)$ we now remove the top edges of C_1, and C_2 and join their end-vertices by two non-intersecting paths, one of length 3, and the second of length 7 as shown in Figure 1(b).[1]

$C_{1,2}(bottom, bottom)$ and $C_{1,2}(bottom, top)$ are handled in a symmetric way. Clearly n_k, w_k, and h_k satisfy the equations in item 4 above. It also follows that the solutions of these equations yield:

- $n_k = 3 \cdot 2^{k+2} - 8$.
- $h_k = 4k + 1$.
- $w_k = 2^{k+2} - 3$.

Thus the area occupied by an element of any $F_k(i,j)$ is $h_k \cdot w_k$, which is $\Theta(n_k \log n_k)$.

We now prove:

Theorem 3.1 *For any election algorithm A there is an element $C \in F_k(i,j)$ such that to elect a leader in C, algorithm A sends $\Omega(n \log n)$ messages, where $n = n_k = 3 \cdot 2^{k+2} - 8$.*

[1] The actual shape of the paths connecting the top and/or bottom edges of C_1 and C_2 are irrelevant, other than to keep the sizes of the boxes enclosing the elements of $F_k(i,j)$ uniform, and that each element of $F_k(i,j)$ has a unique top and bottom edge.

To prove Theorem 3.1 we will need some preliminary results. Given a ring C and an edge e in it, we call an execution of an election algorithm A on C *open on* e if it is obtained by running A on C, but with the introduction of an infinite delay on e; that is, any message sent along e will never reach its destination. Notice that under these conditions, A may not terminate; nevertheless at some point in time all activity on $C - e$ will stop, either because a leader has been elected, or because A is waiting for the messages sent along e to arrive at their destination.

We now prove the following result:

Lemma 3.2 *Let A be a leader election algorithm. For every $k \geq 0$ and integers i, j there exists a ring $C \in F_k(i, j)$ with an execution of A open either at the top or the bottom edge of C such that A sends at least $\Theta(n_k \log(n_k)$ messages.*

Proof: Our result holds for small values of k. Suppose then that it holds for $k-1$. By induction, there are rings C_1 and C_2 in $F_{k-1}(i, j+2)$ and $F_{k-1}(i+w_{k-1}+3, j+2)$ for which A has an open execution on each of them (open at their bottom or top edge) that sends at least $\Theta(n_{k-1}\log(n_{k-1}))$ messages. Assume w.l.o.g that these executions of A are open at the top edges of C_1 and C_2. We now show that there is an execution of A open at the top or bottom edge of $C_{1,2}(top, top)$ that sends at least $\Theta(n_k \log(n_k))$ messages.

Consider first an execution of A on $C_{1,2}(top, top)$ in which we introduce an infinite delay in all the edges along the paths connecting C_1 with C_2. Notice that this will result in executions of A on C_1 and C_2 in which their top edges are open, and thus in the worst case, A will be forced to send at least $f(k-1)$ messages in each of them, where $f(k-1)$ is $\Theta(n_{k-1}\log(n_{k-1}))$.

Suppose now that we remove the delay on all the edges on the path of $C_{1,2}(top, top)$ containing its top edge, and connecting C_1 with C_2. If this forces A to send an extra $\frac{n_{k-1}}{2}$ messages, then the total number of messages sent by A is $2f(k-1)$ plus $O(n_{k-1})$ which proves our result. Let S_{top} be the set of vertices which send or receive a message.

In a similar way, suppose that we remove the delay assumption on the edges on the path $C_{1,2}(top, top)$ containing its bottom edge. Define S_{bottom} in a similar way to S_{top}, and assume again that A is not forced to send $\frac{n_{k-1}}{2}$ extra messages.

This implies that S_{top} and S_{bottom} do not intersect, and thus by simultaneously removing the delay on all the edges on the paths connecting C_1 and C_2, A could enter a deadlock failing to elect a leader! ∎

Theorem 3.1 follows.

4 The External Face Problem

A geometric network induces a partitioning of the plane into a set of polygonal regions called *faces*. One of these faces is unbounded, and will be referred to as the *external* face. In this section, we study the *external face problem:* each processor should find out whether it is a vertex of the external face, and if so, which of the faces containing it is the external one. We start by proving a $\Omega(n \log n)$ message complexity lower bound for geometric rings, and then present a matching upper bound for general geometric networks.

The lower bound proof is analogous to the lower bound proof in the previous section for leader election, except that instead of using the construction shown in Figure 1, we use that shown in Figure 2. The idea is that the rings $C_1 \in F_{k-1}(i, j+2)$ and $C_2 \in F_{k-1}(i + w_{k-1} + 3, j+2)$ used to generate rings in $F_k(i, j)$ cannot identify the external face of the obtained rings before a message has passed along the (top or bottom) open edge.

If we connect the two rings as in Figure 1(a), then the internal face of C_1 becomes part of the internal face of the new ring, while in Figure 2(b) it becomes part of the external face. Thus we have:

Theorem 4.1 *The message complexity of the external face problem in geometric rings is* $\Omega(n \log n)$.

We now proceed to the upper bound.

Theorem 4.2 *Once a leader has been elected, the external face problem can be solved in geometric networks using* $O(n)$ *messages. Thus the message complexity of this problem is* $O(n \log n)$.

Proof: We recall that in any distributed network, the leader can determine a spanning tree T using $O(E)$ messages (where E is the number of edges in the network), and since the geometric network is planar, it has a linear number of edges. Next, using T, the leader can determine the point with the largest id, p, with at most $2n$ messages; sending a wave down the tree to request this information and a wave up the tree to collect it. Notice that p is in the external face of the geometric network. The leader notifies p that it has the largest id, and asks it to finish the determination of the external face. Observe that p can determine which of the faces incident to it is the external one simply by collecting the coordinates of its neighbours. Processor p then sends a message along one of its links on the external face, notifying the neighbor at the other end of this link that it is also in the external face, and which of the faces incident at this vertex is the external one.

Each time a processor q is notified that it belongs to the external face, by receiving a message along a link e in the external face, it forwards this message along the other link e' incident to q in the external face. When p gets this message back, it notifies all the processors that the external face has been determined. This can all be done using a linear number of messages.

■

5 The Convex Hull Problem

The convex hull of a geometric network is the smallest convex polygon that encloses the network. In this section, we present a distributed algorithm to solve the *convex hull problem:* each processor has to find out whether it is a vertex of the convex hull, and if so, the identities of its neighbors in the convex hull, in both the clockwise and counterclockwise direction.

We first observe that using the results of the previous section, we can reduce the problem of finding the convex hull of a geometric network to that of finding the convex hull of its external face in $O(n \log n)$ messages. Thus, in the rest of this section we concentrate in the problem of finding convex hulls of geometric rings. We prove that the convex hull of a geometric ring C can be found using $O(n \log^2 n)$ messages.

Our algorithm proceeds as follows:

1. In the first iteration, we elect a leader in C.
2. The leader then sends a message along C relabeling its vertices $\{v_1, \ldots, v_n\}$ such that v_1 is the leader.
3. In a recursive way, calculate the convex hulls of $\{v_1, \ldots, v_{\lfloor \frac{n}{2} \rfloor}\}$ and $\{v_{\lfloor \frac{n}{2} \rfloor}, \ldots, v_n\}$. Merge these hulls to obtain $Conv(C)$.

We now proceed to show that merging the convex hulls of $\{v_1, \ldots, v_{\lfloor \frac{n}{2} \rfloor}\}$ and $\{v_{\lfloor \frac{n}{2} \rfloor}, \ldots, v_n\}$ can be done in $O(n \log n)$. This will prove our result.

It is important to observe that in what follows, instead of using all of C, we use only the edges in the path obtained from C by deleting the edge connecting v_1 to v_n. This is important since in the recursive iterations of our algorithm, we are calculating the union of convex hulls of subpaths of C. Thus, rather than considering C as a ring, we will consider it as the path connecting v_1 to v_n. Several preliminary results will be needed. Let P_1 and P_2 be the polygons determined by the convex hulls of $\{v_1, \ldots, v_{\lfloor \frac{n}{2} \rfloor}\}$ and $\{v_{\lfloor \frac{n}{2} \rfloor}, \ldots, v_n\}$ respectively. The following result, given without proof, is an easy consequence of the simplicity of C; see Figure 3.

Lemma 5.1 *The boundaries of P_1 and P_2 intersect along a line segment, or in at most two points. Moreover, if the boundaries of P_1 and P_2 do not intersect, then $P_1 \subset P_2$ or $P_2 \subset P_1$.*

This is important since due to this result to calculate the convex hull of $P_1 \cup P_2$, all we need to do is to decide if $P_1 \subset P_2$ or $P_2 \subset P_1$, and if not find exactly two common supporting lines of P_1 and P_2.

We now prove the next result, which we call *The ray shooting lemma*

Lemma 5.2 (Ray-Shooting) *Let $v_i \in \{v_1, \ldots v_n\}$, and let L be any line through v_i. Then using a linear number of messages v_i can find, if any, the points at which L intersects P_1 and P_2.*

Proof: To prove this, notice that all v_i has to do is to send a message along C containing the equation of L. Each time a processor that corresponds to a vertex of P_1 (resp. P_2) receives this message, it verifies whether L intersects the line segments that join it to its neighbours in P_1 (resp. P_2). If an intersection is detected, a message is sent back to v_i informing it that an intersection was detected, along with the coordinates of the intersection point. ■

Observe that in Lemma 5.2, we can easily substitute a line segment or a ray for L by sending the coordinates of the endpoints of a line segment or the initial point and the direction of a ray along C, instead of the equation of L.

The next result follows from Lemma 5.1:

Corollary 5.3 *Let v_i be any vertex of P_1 (resp. P_2), and L any line through v_i. Then we can determine whether L intersects P_1 (resp. P_2) using a linear number of messages.*

We now prove:

Lemma 5.4 *We can detect if $P_1 \subset P_2$ or $P_2 \subset P_1$ using a linear number of messages.*

Proof: Let i be the largest index such that v_i is a vertex of P_1. Suppose that v_j and v_k are the vertices adjacent to v_i on the boundary of P_1. Two cases arise:

1. $i < \lfloor \frac{n}{2} \rfloor$
2. $i = \lfloor \frac{n}{2} \rfloor$

In the first case, all we have to verify is whether the path connecting $v_{\lfloor \frac{n}{2} \rfloor}$ to v_n intersects either of the line segments connecting v_i to v_j and v_k. If it does, then P_1 is not a subset of P_2, otherwise $P_1 \subset P_2$. In the second case, we verify first whether $v_{\lfloor \frac{n}{2} \rfloor + 1}$ belongs to the interior of P_1. If it does, we then verify whether the path $v_{\lfloor \frac{n}{2} \rfloor + 1}$ to v_n intersects either of the line segments connecting v_i to v_j and v_k. Testing if the path from $v_{\lfloor \frac{n}{2} \rfloor}$ to v_n intersects the line segments connecting v_i to v_j and v_k can be accomplished in a linear number of messages due to the observation at the end of the proof of Lemma 5.2. ∎

Observe that $v_{\lfloor \frac{n}{2} \rfloor}$ belongs to both P_1 and P_2. Let L be the horizontal line through $v_{\lfloor \frac{n}{2} \rfloor}$, and let I_1, I_2 be the intervals at which L intersects P_1 and P_2.

Two cases arise:

1. I_1 and I_2 overlap.
2. $I_1 \subset I_2$ or $I_2 \subset I_1$.

For the first case, assume that the left endpoint of I_1 is to the left of I_2, as in Figure 4.

Let P_1' and P_2' be the polygons obtained by intersecting P_1 and P_2 with the halfplane above L, and let us relabel their vertices by $\{u_1, \ldots, u_r\}$ and $\{w_1, \ldots, w_s\}$ respectively such that u_1 and u_r are the left and right endpoints of I_1, and w_1 and w_s are the left and right endpoints of I_2, and let u_p and w_t be vertices of P_1' and P_2' such that the line segment joining them is an edge of the convex hull of $P_1' \cup P_2'$. See Figure 4.

We now prove:

Lemma 5.5 *u_p and w_t can be found using at most $O(n \ln n)$ messages.*

Proof: We show how to perform a binary search on $\{u_1, \ldots, u_r\}$ to find u_p; w_t can be found in a similar way. Take the mid-vertex $v_{\lfloor \frac{r}{2} \rfloor}$, and consider ray R through $v_{\lfloor \frac{r}{2} \rfloor}$ starting at $v_{\lfloor \frac{r}{2} \rfloor -1}$; see Figure 5. If R intersects $P_2^{'}$, then $u_r \in \{u_1, \ldots, v_{\lfloor \frac{r}{2} \rfloor -1}\}$, else $u_r \in \{v_{\lfloor \frac{r}{2} \rfloor}, \ldots, v_r\}$. Iterating this procedure, we can find u_r in a logarithmic number of iterations. By Lemma 5.2, detecting whether R intersects $P_2^{'}$ can be done using a linear number of messages. Our result follows. ■

In a similar way we can find, using $O(n \log n)$ messages, the missing edge e of the convex hull of the union of the polygons $Q_1^{'}$ and $Q_2^{'}$ obtained by intersecting P_1 and P_2 with the plane below L. If the lines generated by the edges $u_p w_t$, and e are supporting lines of P_1 and P_2, then these are the edges we are seeking to calculate $Conv(P_1 \cup P_2)$. It could happen, however, that one of them, say $u_p w_t$, is not an edge of $Conv(P_1 \cup P_2)$. This could happen if w_t is exactly w_s. See Figure 5. The reader may easily verify that performing a binary search on the chains $u_1, \ldots, u_p$, and $w_{s-1}, \ldots, w_m$, we can find the missing edge in $Conv(P_1 \cup P_2)$, where w_m is the end-vertex of e in $Q_2^{'}$; see Figure 6.

We now show how to solve the case when $I_1 \subset I_2$ or $I_2 \subset I_1$.

Suppose without loss of generality that $I_1 \subset I_2$. Let v_r and v_s be the vertices of P_1 with the smallest and the largest y coordinates, and consider the line L' that passes through them. Then L' intersects P_1 and P_2 at two overlapping intervals $I_1^{'}$ and $I_2^{'}$. Substituting L for L' and proceeding as before, we obtain $Conv(P_1 \cup P_2)$.

The last result that we need to prove is how to calculate the relative order of the processors in the convex hull of C. This order is used to perform the search procedure described in Lemma 5.5.

We now show how to *relabel* the vertices on the convex hull of C as $\{u_1, \ldots, u_m\}$ such that u_i is adjacent to u_{i+1} and u_{i-1}, addition taken *mod* n.

Recall that at the end of the execution of the steps described above, each vertex $v_i \in C$ knows if it belongs to the convex hull of C, and if it does, it also knows the identities, say $v_{l(i)}$ and $v_{r(i)}$, of its left and right neighbours in the convex hull of C. To start the relabeling process the leader, v_1, sends a message along C to find m, the number vertices of $Conv(C)$. Once v_1 knows r, it initializes this relabeling by sending a message containing r that will be forwarded to v_2, v_3, etc. until it reaches the first vertex v_i of C in $Conv(C)$. Now v_i becomes u_1. Notice that at this point, v_i knows that its left neighbour in $Conv(C)$, $v_{l(i)}$ is u_2, and $v_{r(i)}$ is u_m. Processor v_i forwards this information along C until it reaches either of $v_{l(i)}$ or $v_{r(i)}$. Suppose it reaches $v_{l(i)}$ first. Now $v_{l(i)}$ knows that its left neighbour $v_{l(l(i))}$ is u_3. Then $v_{l(i)}$ modifies the message to contain the information that $v_{l(l(i))}$ is u_3, and $v_{r(i)}$ is u_m, and forwards it along C. This procedure continues until all the vertices of $Conv(C)$ have been relabeled. For the polygon shown in Figure 7, the message starting at v_1 will reach v_2 first, and relabel it u_1. It will then reach u_2, then u_6, u_3, u_5, and finally u_4. Clearly the relabeling procedure uses a linear number of messages.

Summarizing, we have:

Theorem 5.6 *The message complexity of the convex hull in geometric networks is $O(n \log^2 n)$ messages.*

6 Discussion

The planarity restriction imposed on geometric networks is essential to our work. Finding non-planar embeddings of distributed networks is trivial, e.g. a processor with id x could simply assume that it is located at point (x, x^2). Finding planar embeddings of distributed networks, on the other hand, is a more challenging problem.

For rings, we know that the problem of finding convex embeddings has $O(n \log n)$ message complexity; first a leader is elected, then the leader sends a message around the ring, renaming the processors with consecutive integers 1 to n. Then if a processor gets value i, it chooses (i, i^2) as its coordinates. To show that this algorithm is optimal, notice that once the embedding has been obtained, we can elect a leader in $O(n)$ messages. However since election in a ring takes $\Theta(n \log n)$ messages, it follows that our convex embedding algorithm for rings is optimal.

Furthermore, geometric information does not help to reduce the message complexity of problems that require $\Omega(n \log n)$ messages in a geometric ring. To prove this, suppose that some problem P can be solved with $\Theta(f(n))$ messages in a geometric ring. To solve P in a ring (not necessarily geometric), we can first find a planar embedding of the ring in $O(n \log n)$ messages, then run the geometric algorithm, solving the problem with $O(f(n)+n \log n)$ messages.

The question remains whether there is a distributed algorithm to find a planar (not necessarily convex) embedding of a ring with smaller message complexity, $o(n \log n)$.

We do not know if the convex hull algorithm outlined in the previous section is optimal. However we venture the following conjecture:

Conjecture 6.1 *The message complexity of the convex hull problem for geometric networks is* $\Omega(n \log^2 n)$*. The same bound holds for geometric trees.*

$\Lambda\psi$

References

[1] An Atlas of Cyberspaces,
$http://www.geog.ucl.ac.uk/casa/martin/atlas/isp\_maps.html$.

[2] James E. Burns, "A formal model for message passing systems," Technical Report TR-91, Computer Science Department, Indiana University, Bloomington, September 1980.

[3] Nancy A. Lynch, *Distributed Algorithms,* Morgan Kaufmann Publishers, Inc. 1996.

[4] F. Preparata, and M.I. Shamos, "Computational Geometry, an introduction", Springer Verlag, (1985).

[5] Gerard Tel, *Introduction to Distributed Algorithms,* Cambridge University Press, 1994.

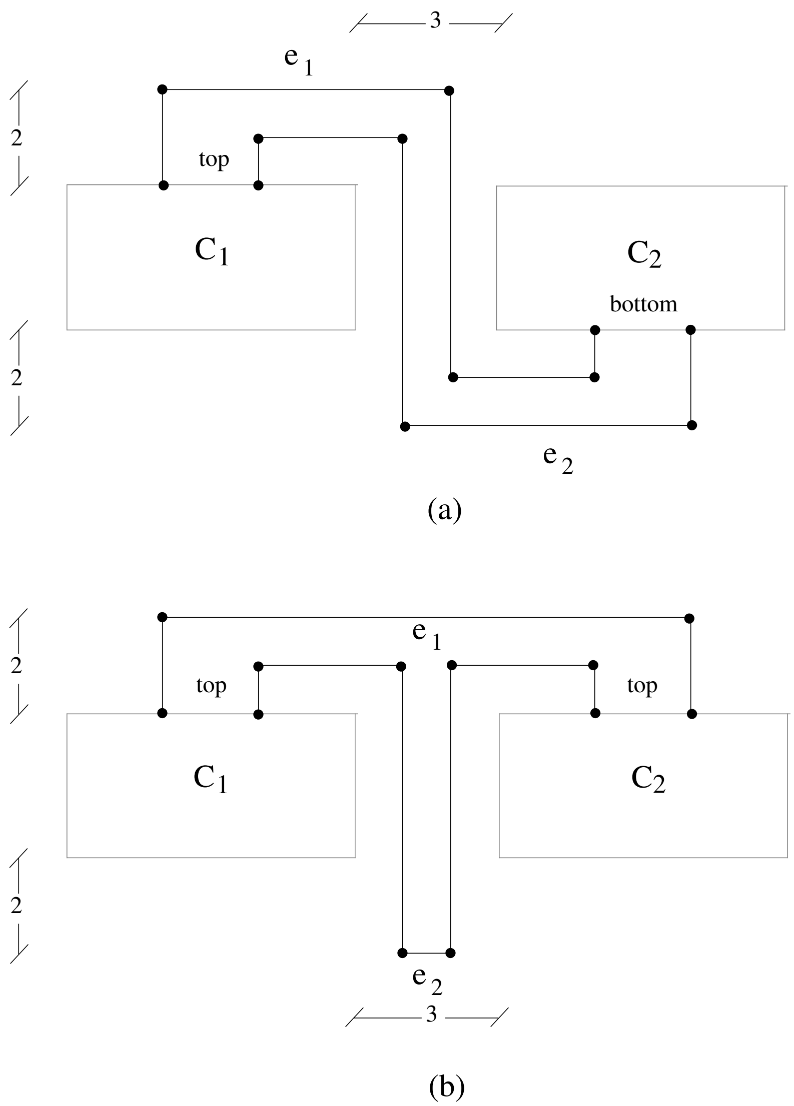

Figure 1: Inductive construction for the leader election lower bound.

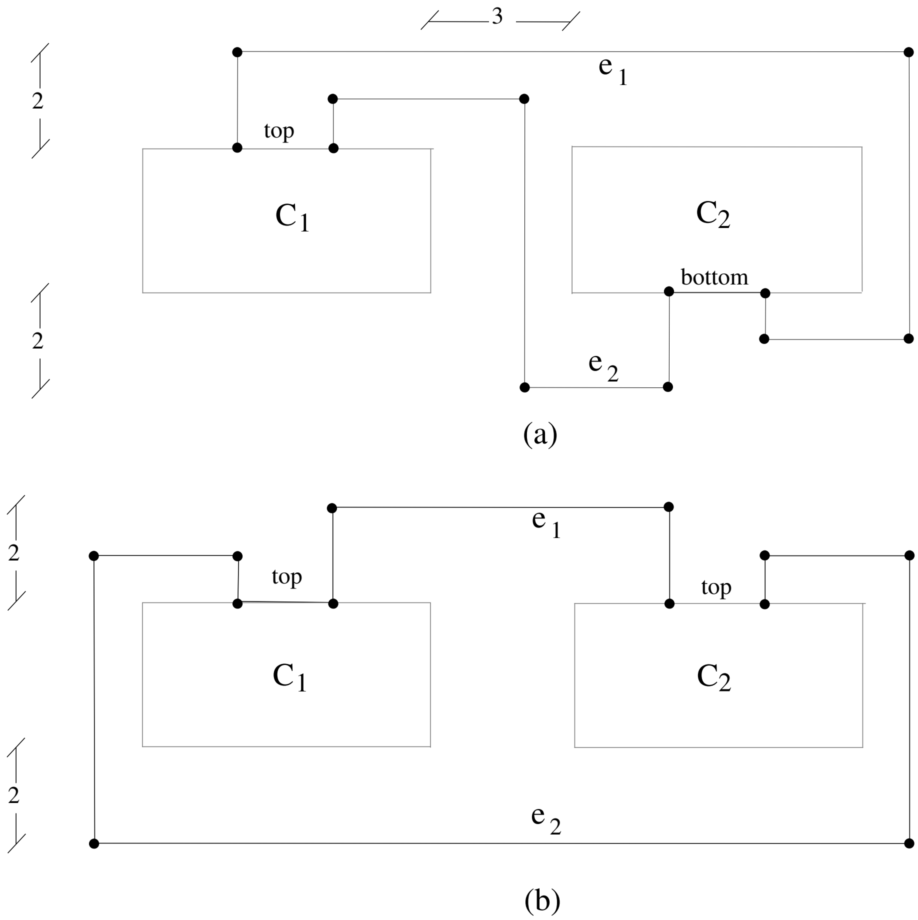

Figure 2: Inductive construction for the external face lower bound.

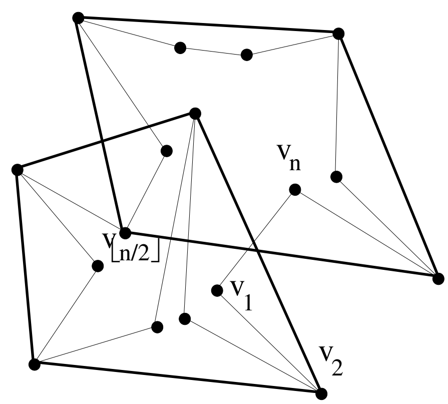

Figure 3: The boundaries of P_1 and P_2 intersect at most twice.

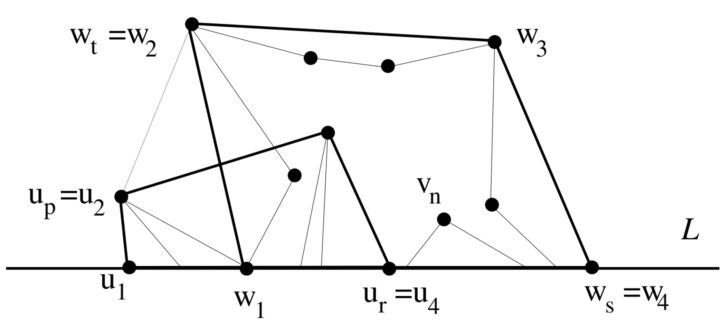

Figure 4: Defining P_1' and P_2'.

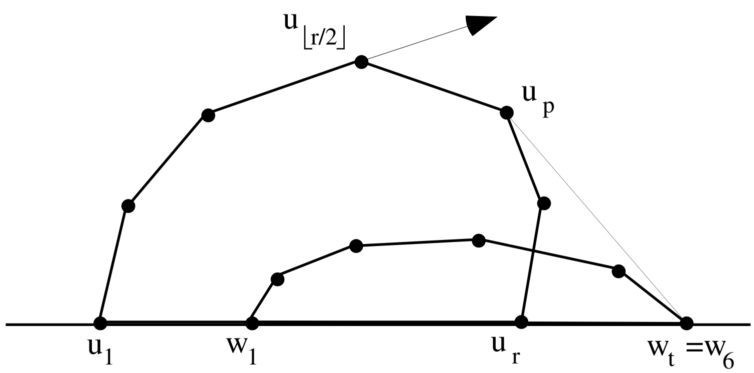

Figure 5: Finding u_p and w_t.

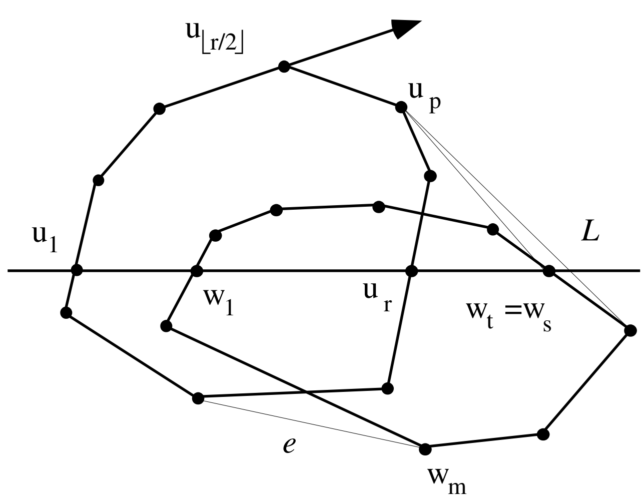

Figure 6: Solving the case $w_t = w_s$.

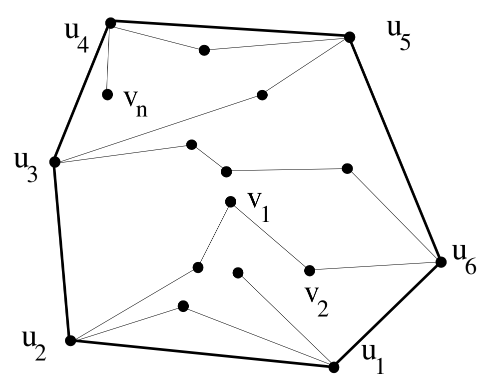

Figure 7: Realbeling the vertices of $Conv(C)$.