# AN IMPROVED UPPERBOUND FOR DISTRIBUTED ELECTION IN BIDIRECTIONAL RINGS OF PROCESSORS

J. van Leeuwen and R.B. Tan

RUU-CS-85-23
August 1985

Rijksuniversiteit Utrecht

**Vakgroep informatica**

Budapestlaan 6 3584 CD Utrecht
Corr. adres: Postbus 80.012 3508 TA Utrecht
Telefoon 030-53 1454
The Netherlands

# AN IMPROVED UPPERBOUND FOR DISTRIBUTED ELECTION IN BIDIRECTIONAL RINGS OF PROCESSORS

J. van Leeuwen and R.B. Tan

Technical Report RUU-CS-85-23

August 1985

Department of Computer Science
University of Utrecht
P.O.Box 80.012, 3508 TA Utrecht
the Netherlands

# AN IMPROVED UPPERBOUND FOR DISTRIBUTED ELECTION IN BIDIRECTIONAL RINGS OF PROCESSORS*

J. van Leeuwen and R.B. Tan

Department of Computer Science, University of Utrecht
P.O.Box 80.012, 3508 TA Utrecht, the Netherlands

Abstract. We present a distributed algorithm for electing a leader (i.e., breaking symmetry) in bidirectional rings of N processors with no global sense of orientation, that uses at most 1.44...NlogN+O(N) messages in the worst case.

Keywords and Phrases : distributed algorithms, bidirectional rings, election, message complexity.

1. Introduction . In a number of distributed problems it is required that the participating processors elect a central coorinator (a "leader"), e.g. as part of a reinitialisation or recovery procedure. It is normally assumed that the processors are distinguished by unique identification numbers but that they are otherwise "identical" and have no global knowledge about the network topology. Following LeLann [4] the election problem has been studied mainly for circular configurations, i.e., rings of processors.

In this paper we consider the case of N processors that act asynchronously and are connected in a bidirectional ring. We assume that the processors have no global sense of orientation on the ring, i.e., their

* This work was done while the second author was visiting the University of Utrecht, supported by a grant of the Netherlands Organization for the Advancement of Pure Research (ZWO).

** Author's address: Department of Computer Science, University of Sciences and Arts of Oklahoma, Chickasha, OK 73018, USA.

notions of "left" and "right" need not be consistent initially. On the other hand, a processor can tell whether a message has arrived via its "left" port or its "right" port, e.g. by maintaining separate message queues. The election problem asks for a distributed algorithm to select a unique processor (the "leader") on the ring, by having processors exchange messages with their neighbors. We present a solution that uses at most 1.44...NlogN+O(N) messages in the worst case, which improves on the previous worst case bound of 1.89...NlogN+O(N) messages of an algorithm due to Santoro, Korach, and Rotem [6] for this type of rings.

It is interesting that the election problem for bidirectional rings appears to be be more complex than for unidirectional rings, which have a very "strong" form of global orientation. (See e.g. Bodlaender and van Leeuwen [1] for a survey of the election problem in rings of processors.) For unidirectional rings the currently best algorithms for the election problem use 1.44...NlogN+O(N) messages (Peterson [5]) and 1.356...NlogN+O(N) messages (Dolev, Klawe, and Rodeh [2]) respectively, in worst case. Our algorithm for the bidirectional case achieves the same worst case bound as Peterson's algorithm, although it is rather more complicated in its details. An interesting feature of our algorithm is that different (asynchronous) runs may result in different leaders being elected, although clearly the leadership may be transfered to any other processor in O(N) additional messages after the election algorithm has come to completion.

The paper is organised as follows. In Section 2 we present the election algorithm under a simplified assumption about the message-handling capability of the links. In Section 3 we analyse the message complexity of the algorithm and prove it to be bounded by 1.44...NlogN+O(N) messages in worst case. In section 4 we show the necessary modifications in order to eliminate the assumption on the links and prove that the worst case message complexity of the resulting algorithm still has the same upperbound*.

* We have recently heard that, independently, another 1.44...NlogN election algorithm was found by S. Moran et.al. (Haifa). See: S. Moran,

## 2. Distributed election in a simplified model of bidirectional rings.

Consider a bidirectional ring of N processors. We assume that each processor has separate queues for the incoming messages from each direction on the ring, and that the links and queues preserve the order in which messages are sent. A processor can send messages in one or two directions simultaneously on the ring. While every processor can distinguish between the two directions on the ring, there is no global sense of orientation at the outset. The distributed election algorithm that we develop is based on a series of message-exchanges between all processors on the ring, and repeatedly generates new information when messages "meet". As conceptually messages should meet in a processor and not bypass each other in a link we make the simplifying assumption that the links are "half-duplex", i.e., the links carry at most one message at a time regardless of its direction. In Section 4 we will rid ourselves from this assumption by a suitable modification of the algorithm.

The distributed election algorithm operates in the following manner. At all times every processor maintains a register ID that contains the name (identifier) of a "large" processor that is still in the game, and a (Boolean) register DIR that contains a "direction" on the ring in which there are processors that still have a "smaller" processor up for election. Typically the "large" candidate will exist on one side of the processor, and DIR will point towards the other side. Messages are generated that contain the name of a "large" candidate, and are sent out (or: passed on) in the direction where a "smaller" candidate is known to be still alive. The idea of the chase is to eliminate the smaller candidate, and force agreement on the larger candidate. Processors that initiate a chase are termed active, and the remaining processors are termed observant ("passive"). After the initialization phase of the algorithm (phase 0), the processors that are local minima are active and all other processors are observant.

---

M. Shalom, and S. Zaks, An algorithm for distributed leader finding in bidirectional rings without common sense of direction, draft, August 1985.

After the current active processors have initiated a chase, the observant processors basically relay messages onwards unless they notice an "unusual" situation on the ring. Active processors immediately go to the observant state after initiating a chase, and some of the observant processors will become active again as a result of their conclusion from the "unusual" situation that was observed. The new active processors will initiate another chase, and the same procedure repeats. In order to distinguish new chases from old, the algorithm will be organized in phases and each processor will maintain a register PNUM containing the phase-number of the most recent phase in which it participated. Initially all PNUM registers are set to 0. Through the phase numbers we keep track of the logical ordering of the chases (as if they were scheduled ring-wide and strictly separated in time). All messages are stamped with the phase number of the generating (active) processor, to separate the phases and to synchronize the processors that did not receive messages for a while and are unaware of the current phase.

There are two "unusual" situations that can arise at the site of an observant processor as the algorithm proceeds (recall that the links are assumed to be half-duplex):

(i) the processor receives a message of the current phase, say via its left link, that contains a value that is less than the current value in its ID register. (This happens when the message has completed its chase of smaller processors and now bumps into a larger one that should take over.) The processor turns active, increments its phase number by 1, and initiates a chase with the value in its current ID in the direction of the message that was received, i.e., out over its left link.

(ii) the processor receives two messages of the same phase from opposite directions (a "collision"). The processor turns active, increments its phase number by 1, and initiates a chase with the largest value contained in the two messages in the direction of the smallest.

It will be shown in the next Section that the number of active processors that can arise in a phase rapidly decreases as the algorithm proceeds, and that in the end precisely one processor will be left. This processor will know that it is elected because either it receives a message of the current phase with a value identical to the one it sent out (and stored in its ID register) or it receives two messages of the same phase from opposite directions that hold identical values. Thus the processor elected by the algorithm is not necessarily the processor with the largest identification number. The correctness of the algorithm will be demonstrated in the next Section as well.

In the remainder of this Section we will describe the algorithm performed by every participating processor in more detail. We assume that each processor starts with the initialization phase (phase 0) and then alternates between the active and the observant state as dictated by the algorithm. Messages are of the form <v, p> where v is a "value" (a processor identification number) and p a phase number.

```
Algorithm E ("Election")
{The algorithm describes the actions of an arbitrary processor on a
bidirectional ring with half-duplex links as required for electing a
leader.}
1. Initialization
  {Let u be the executing processor's identification number. It can
  either begin the algorithm spontaneously or be induced into it by
  one or both of its neighbors.}
  send message <u, o> to both neighbors;
  PNUM := 0;
  wait for the corresponding messages <$u_1$, o> and <$u_2$, o> to come in
  from the two neighbors;
  if $u_1$>u & $u_2$>u then
          begin
            ID := max($u_1$, $u_2$);
            DIR := the direction of min($u_1$, $u_2$);
            goto active
          end
  else goto observant;
  {This ends the initialization phase. A processor continues in either
  the active or the observant state, to perform the further steps of
  the election algorithm.}
```

## 2. Election

{The election proceeds in phases. A processor performs in either the active or the observant state.}

### a. Active

{A processor enters the active state with some value v stored in its ID-register and a phase number p. The phase number p is either stored in PNUM or it is an "update" stored in a temporary register.}

```
PNUM := p+1;
send message <v, PNUM> to the direction DIR;
goto observant;
```

{If a processor has executed this code, it will be called "active" in phase PNUM even though it now went into the observant state. As implied by the code, the change to active will always come with an increase in phase number.}

### b. Observant

{In this state a processor receives messages and passes them on, unless an "unusual" situation is observed that enables it to initiate a new phase.}

```
receive message(s) from one or both directions;
discard any message <v, p> received with p<PNUM;
discard any message that does not have the highest p-value among the
messages;
{The logic of the algorithm will guarantee that at most two messages
now remain.}
case #messages left of
   0: goto observant;
   1: begin
        {Let the one message received be <v, p>, where necessarily
         p≥PNUM.}
          if p=PNUM then
             case "test as indicated" of
                v=ID: goto inaugurate;
                v<ID: begin
                         DIR := the direction from which the
                                message was received;
```

```
                        goto active
                      end;
                v>ID: begin
                      {stop the message, the assignment ID := v is
                       optional}
                        goto observant
                      end
          end
        else
          begin
          {This branch will be followed when p>PNUM and the message
           must be relayed further.}
            PNUM := p;
            ID := v;
            DIR := the direction in which the message was going;
            send message <ID, p> to the direction DIR;
            goto observant
          end;
   2: begin
      {Let the two messages received be <$v_1$, p> and <$v_2$, p>, neces-
       sarily from opposite directions and with $p \geq$PNUM.}
        case "test as indicated" of
          $v_1 = v_2$: begin PNUM := p; goto inaugurate end;
          $v_1 \neq v_2$: begin
              {If this happens we say that a "collision" occurs.}
                ID := max($v_1$, $v_2$);
                DIR := the direction of min($v_1$, $v_2$);
                goto active
              {Note that the update of PNUM occurs in the active
               state, so PNUM always denotes the true phase num-
               ber for a processor.}
              end
      end
end;
```

3. Inauguration

{A transfer to this final phase occurs when the algorithm terminates, and the ID register contains the identity of the (unique) leader. We later show that precisely one processor will eventually make the transition to this state.}

We make a few final comments on the assumption that the links are "half-duplex". Clearly the algorithm requires that the meeting ("collision") of two messages in a processor is recognized as such and is not treated as "twice the receipt of one message". It follows that a message should not be sent and any processing be un-done when there is still an unprocessed incoming message in a queue at the processor, in the direction in which a message would be sent. Strictly speaking, the assumption that messages do not bypass each other in a link while going in opposite directions is only required for messages with the same phase number.

## 3. An analysis of algorithm E

In this Section we will show that algorithm E "terminates" and that it is correct, i.e., upon termination precisely one processor has been elected as a leader. We will also prove that algorithm E uses at most 1.44...NlogN+O(N) messages, by showing that it terminates within 1.44..logN phases that all require at most N messages total (except perhaps the first).

Clearly every processor that whishes to begin the election, must begin with the "initialization phase" (phase 0). The messages <u, 0> of processor u will awaken its two neighbors to the election process as well, if they aren't aware of it yet. Eventually every processor on the ring is woken up and is passing through its "phase 0", and thus every processor will receive two messages with p-value 0 in this phase (one from each neighbor). It follows that all processors follow suit as soon as one processor starts the election, and that precisely 2N messages are exchanged in the initialization phase. The initialization ends by declaring the processors that are "local minima" to be active for the next phase (phase 1) and all other processors to be observant. Clearly at least 1 and at most N/2 processors will be local minima and thus are active in phase 1. Active processors will be separated by observant ones.

As the algorithm proceeds messages are generated and relayed around the ring, and the phase-number (PNUM) of observant processors continues to be updated (increased) whenever necessary. If an observant processor turns active, its phase number is incremented again. Messages with lower phase numbers than current at the receiving processor are ignored and (hence) are not passed on. It follows that at every processor the messages that come in from the same direction, in fact come in with increasing p-value! (Note that all links and queues preserve the order of the messages.) It implies that for the analysis we may as well assume that the algorithm proceeds synchronously, in "increasing" phases. This gives the worst possible bounds on the message-complexity, because it automatically means that no messages will be discarded in this regime. (In the asynchronous case one part of the ring could be "faster" and proceed to higher phase numbers, and thus "overtake" the ongoing process in the other part of the ring in its lower phase. Thus the asynchronous algorithm may, in fact, use fewer message exchanges to complete the election.) Clearly, for the termination and correctness proofs no such restriction can be made, although some aspects of the synchronous regime will be implicit in any version of the algorithm. Let PNUM(u) be the contents of the PNUM-register of processor u at some moment, and ID(u, p) be the contents of the ID-register of processor u at some moment while PNUM(u)=p. Let $p \geq 1$.

Lemma 3.1. If a processor u receives a message $m \equiv \langle v, p \rangle$ with p=PNUM(u), then m must have arrived from a direction (i.e., a link) opposite to the one over which u has last sent out a message.

Proof.

Suppose processor u has last sent a message $m' = \langle v', p' \rangle$ in the direction from which m is arriving. The action could occur when u was active or when it relayed m' onward. In either case we have necessarily p'=PNUM(u). By the assumption on the links, m and m' should have met at a processor u' with $u' \neq u$. (If m' would have been discarded at some processor u" before meeting with m, then certainly m would be discarded at u" or earlier too. We may assume that m and m' are not discarded at u' either, because apparently m still makes it to u.) But u' now either concludes that it is the leader (goto inaugurate) or turns active and increments its phase number to p+1 before stamping it on the next message it sends out. In either case m is stopped at u' and does not reach u. Contradiction. □

The processors u with PNUM(u)=p are said to be in phase p. Phase p is reached either in the active state (by incrementing PNUM to p) or in the observant state (by updating PNUM to p while relaying a message of the form <v, p>). In the former case we say that u is "active in phase p". For every processor u we call the first value of ID(u, p) in phase p the "color" of u in phase p. (Observe that ID(u, p) will be constant during phase p. Only when a message <v, p> is received with v>ID could an update of u's color to v be made, but this is omitted from the given version of algorithm E.)

Definition

(i) $A_p$ is the set of processors u that are active in phase p.

(ii) $\tilde{A}_p$ is the set of colors of processors u that are active in phase p (the "active colors").

The idea of the colors is that we keep track of the identification numbers of outstanding candidates for election.

Lemma 3.2. For all $p \geq 1$, $\tilde{A}_p \supseteq \tilde{A}_{p+1}$.

Proof.

Messages of the form <v, q> for any q<p+1 can exist only if they were generated by a processor that was active in phase q with color v(hence $v \in \tilde{A}_q$). Suppose a processor u enters phase p+1 with color v. There are three possibilities.

Case I: u became active because two messages $<v_1, p>$ and $<v_2, p>$ collided at u, with $v_1 \neq v_2$. Note that $v_1, v_2 \in \tilde{A}_p$ and that the color of u is determined to be $\max\{v_1, v_2\} \in \tilde{A}_p$.

Case II: u became active because one message <v, p> arrived at u while PNUM(u)=p and v<ID(u, p). Arguing inductively one sees that necessarily $ID(u, p) \in \tilde{A}_p$. (Note that it either is the color u got when entering phase p, or it is the color of another active processor from this

phase that it obtained by an update of its previous ID(u, p)-value.) The color of u in phase p+1 is determined to be ID(u, p)$\varepsilon\tilde{A}_p$.

Case III: u receives a message $<v_1, p>$ with p>PNUM(u). Note that u does not turn active now. Again we have $v_1\varepsilon\tilde{A}_p$ and the color of u is determined to $v_1$.

In all cases (viz. I and II) it follows that $v\varepsilon\tilde{A}_p$. Hence $\tilde{A}_p \supseteq \tilde{A}_{p+1}$. □

For every phase p we will distinguish between c-active processors, which became active because of the collision of two messages of the same phase, and d-active processors, which became active because of the receipt of one message of the same phase with smaller value (the "non-collision" case). It is easily seen from algorithm E that there can be no other types of active processors.

Lemma 3.3.

(i) Between every two c-active processors in phase p there are at least two non-active processors in the same phase.

(ii) A c-active processor in phase p cannot be c-active in the next phase.

(iii) A d-active processor in phase p cannot be c-active in the next phase.

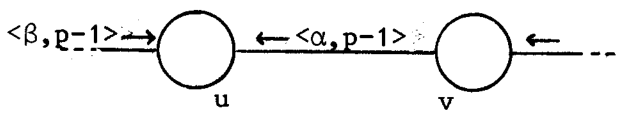
(iv) If two adjacent processors are active in the same phase p, then they do not send messages to each other directly in this phase (i.e., over the joining link and simultaneously).

Proof.

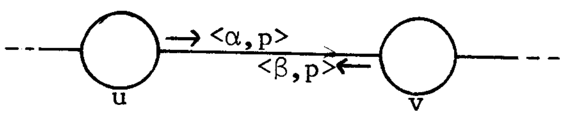
(i) Consider any two consecutive c-active processors u, $u'\varepsilon A_p$. If there is no processor in between u and u', then one of them (say u) can only have become c-active because it received a $<v, p-1>$ message from u' and a message of the same phase "from the other side". By lemma 3.1. u' cannot now receive a message with phase number p-1 from its "u-side", and hence it cannot become c-active in phase p. Suppose next that there is one processor u" in between u and u', and u" is not active in phase p. Again, for one of the processors (say u) to become c-active in phase p it is required that it receives $<*, p-1>$ messages from both directions. But, if u" sends a $<*, p-1>$ message to u, it cannot ever send a message of the same phase in the other direction (to u'). Thus u' cannot become c-active in phase p. It follows that u and u' must be separated by at least two non-active processors, in both directions on the ring.

(ii) and (iii) follow immediately from lemma 3.1.

(iv) Consider the special case of p=1 first. As the active processor must be "local minima", there must be at least one non-active processor in between every two active processors. Thus the lemma is vacuously true in this case. For p>1, suppose u and v are two neighboring processors that turned active in phase p. By (i) u and v cannot both be c-active. We consider two further, essential subcases. First, suppose u is c-active and v is d-active in phase p. (The case with the roles of u and v interchanged is handled completely analoguously.) Let u be c-active because it received a message <α, p-1> from the v-side (thus, <u>from</u> v) and a message <β, p-1> from the other side. Note that v can be d-active only because it received a phase-(p-1) message from its non-u side with a smaller value than the current contents of its ID-register(α). Consequently, at least v will send out its phase-p message out away from u again (in the direction of where the smaller-valued message came from).



Next, suppose that both u and v are d-active in phase p. Suppose by way of contradiction that u and v do send their phase-p message to each other. Thus let u send <α, p> to v, and let v send <β, p> to u. Because u and v are d-active, they will have had color α and β respectively at the end of phase p-1 as well and they can only send their phase-p message to each other if they sent their phase-(p-1) message to each other as well (because it triggered the activation to phase p). But the phase-(p-1) messages must have been <α, p-1> (from u to v) and <β, p-1> (from v to u) respectively, and the activation of u and v can only be triggered when β<α (at u) and α<β (at v). Contradiction. Thus, when u and v are both d-active in phase p, at least one of them must send its phase-p message out away from the other as well. □



Lemma 3.3. shows that there is always "room" during a given phase for messages to collide. The activities of the distributed election algorithm in all phases are triggered by the active processors. Our first concern is to show that the algorithm does not "die", i.e., fall silent, before a leader is found (if ever).

Proposition 3.4 For all $p \geq 1$, if $A_p \neq \emptyset$ and a leader is not found in phase p, then $A_{p+1} \neq \emptyset$.
Proof.

We prove this by way of contradiction. Let p be the largest phase number for which $A_p \neq \emptyset$ (hence $p \geq 1$) but $A_{p+1} = \emptyset$, while a leader is not found in phase p.

Consider any active processor $u \in A_p$. When u turned active in phase p, it sent out a message $\langle \alpha, p \rangle$ in some direction over the ring (with $\alpha$ the color of u). If u is the only active processor on the ring, algorithm E shows that the message is relayed all the way around the ring and that eventually u gets its own message back. In this case u declares itself the leader. Contradiction. Thus $A_p$ must contain k processors $u = u_1, \ldots\ldots u_k$ (in ring order), for some $k \geq 2$. Suppose the processors have colors $\alpha_1, \ldots, \alpha_k$ respectively, and (hence) the $i^{th}$ active processor sent out a message $\langle \alpha_i, p \rangle$ when it turned active in phase p. If any two consecutive active processors sent their phase-p message towards each other, then the two messages must "collide" in some intermediate processor u' (by the assumption on the links) and u' would turn active in phase p+1 or inaugurate. Contradiction. Thus, active processors $u_i$ in phase p necessarily send their $\langle \alpha_i, p \rangle$ message in the same direction on the ring. Define $u_{k+1} \equiv u_1$.

If there is any pair of consecutive active processors $u_i$, $u_{i+1}$ $(1 \leq i \leq k)$ such that $\alpha_i < \alpha_{i+1}$, then the arrival of the $\langle \alpha_i, p \rangle$ message at processor $u_{i+1}$ will turn the processor active on phase p+1. Contradiction. It follows that $\alpha_i \geq \alpha_{i+1}$ for all i and hence, because $\alpha_{k+1} \equiv \alpha_1$, that $\alpha_1 = \ldots = \alpha_k$. Thus all active processors in phase p must have the same color, under the current assumptions. Now the arrival of the $\langle \alpha_i, p \rangle$ message at processor $u_{i+1}$ will lead $u_{i+1}$ in the inauguration state, and it will declare processor $\alpha \equiv \alpha_i \equiv \alpha_{i+1}$ the leader $(1 \leq i \leq k)$. (Note that $\alpha$ is indeed the current value in $u_{i+1}$'s ID-register.) But we assumed no leader in phase p would be found. Contradiction.

It follows that necessarily $A_{p+1} \neq \emptyset$, if $A_p \neq \emptyset$ and no leader is found in phase p. □

Our next concern is to show that the algorithm is "partially correct", i.e., if a processor enters the inauguration state ("declares itself a leader") then it is the only processor to do this and the algorithm terminates. Let $p \geq 1$.

Lemma 3.5 Suppose two active processors u, u' $\varepsilon$ $A_p$ have the same color $\alpha$. Then:

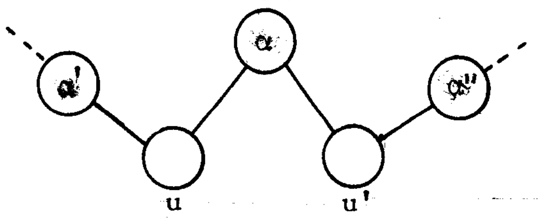
(i) u and u' must be adjacent, i.e., u and u' are separated only by non-active processors,

(ii) all processors in between u and u' (in one direction) have the same value $\alpha$ in their ID-register, and

(iii) u and u' send their phase-p messages out in opposite directions, i.e., "out away from the edges of the $\alpha$-colored interval".

Proof.

We prove this by induction on p. For p=1 the only way two active processors u, u'$\varepsilon A_p$ can have the same color $\alpha$ is when they both received this color from the same processor during the initialization. Necessarily this processor is directly adjacent to both u and u', and its color($\alpha$) is larger than the color of the other neighbors of u and u'. It follows that u and u' will indeed send out their phase-1 messages "out away from each other". Note that the color $\alpha$ does not occur anywhere else outside of the interval from u to u'. (Of course the messages sent out by u and u' will expand the size of the $\alpha$-colored interval in phase 1 to the left and to the right.) The induction basis follows.



Next consider two active processors u, u'$\varepsilon A_{p+1}$ that have the same color $\alpha$. Clearly color $\alpha$ is "inherited" from some processors in phase p. There are only two ways in which u and u' can have gotten color $\alpha$ in phase p+1.

<u>Case</u> I: there are two active processors v, $v' \epsilon A_p$ that have the same color $\alpha$ and that are responsible for $\alpha$-coloring u and u' respectively.

By the induction hypothesis v and v' must be adjacent active processors in phase p (thus, there are no active processors in between them), all processors in between v and v' have the same color $\alpha$, and v and v' send out their $<\alpha, p>$ messages out away from the $\alpha$-colored interval. The only way u and u' can be $\alpha$-colored in the next phase is when the $<\alpha, p>$ message of v collides at u with a smaller-valued message that arrived from the opposite direction and, likewise, the $<\alpha, p>$ message from the other direction. Note that as the $<\alpha, p>$ messages of v and v' travel out to u and u' respectively, all processors in between u and u' are colored $\alpha$ as well. When the $<\alpha, p>$ messages arrive at u and u' and collide with the smaller-valued phase-p messages that arrived "from the other side", it is clear that u and u' will continue to send their phase-(p+1) messages out away from each other (in the direction from which the smaller-valued messages came). In particular no processor in the $\alpha$-colored interval between u and u' can be active in phase p+1. The induction hypothesis follows.

<u>Case</u> II: there is only one active processor $v \epsilon A_p$ that has color $\alpha$ and that is responsible for $\alpha$-coloring u and u'.

When v turned active in phase p (with color $\alpha$) it sent out a $<\alpha, p>$ message in one particular direction on the ring. The only way this message can activate a processor (say u) to phase p+1 and $\alpha$-color it, is when the message collides at u in phase p with a smaller-valued phase-p message that arrived from the other side. Note that as the $<\alpha, p>$ message travels from v to u, all processors in between v and u are colored $\alpha$ as well. Clearly u will send its phase-(p+1) message out from v (in the direction from which the smaller-valued message came). The only way v can be responsible for $\alpha$-coloring another processor u' in phase p+1, is when a smaller valued phase-p message comes in at v from the other side (i.e., over the link different from the one over which v sent its phase-p message) and in fact activates v to phase p+1. Thus $u' \equiv v$. Clearly v will send out its phase-(p+1) message in the direction of the smaller-valued message as well, i.e., out away from the $\alpha$-colored interval. Again the induction hyphothesis follows.

This completes the proof of the induction step. □

Proposition 3.6. If a processor moves to the inauguration state in phase p, then:

(i) it is the only one to do so,

(ii) $|\tilde{A}_p|=1$, and

(iii) all processors on the ring have the same value in their ID-register.

Proof.

Suppose processor u moves to the inauguration state in phase-p. There are only two ways in which this can happen (see algorithm E). We assume u is the first to move.

Case I: u has value $\alpha$ in its ID-register and receives a $<\alpha, p>$ message from one side (the non-collision case).

As $\alpha\varepsilon\tilde{A}_p$ we distinguish two further sub-cases. First, suppose there is only one $\alpha$-colored active processor v in phase p. As v is responsible for any $<\alpha, p>$ message on the ring and (hence) for $\alpha$-coloring any processor in phase p, u cannot have been in phase p or gotten color $\alpha$ in phase p before the $<\alpha, p>$ message arrived unless u=v. It follows that the $<\alpha, p>$ message must have made one full tour around the ring. If there had been any other active processor v' in phase p of some color $\alpha'\neq\alpha$ on the ring, then the $<\alpha, p>$ message would not have made it (either because it was stopped or because it was "replaced" by a message with phase-number p+1 as the result of an activation somewhere). Thus all processors simply relay the $<\alpha, p>$ message and are colored $\alpha$, $|\tilde{A}_p|=1$, and u is the processor to move to the inauguration state when the $<\alpha, p>$ message arrives.

Second, suppose there are two $\alpha$-colored active processors v, $v'\varepsilon A_p$. (By lemma 3.5 there can be no more than two such processors.) By lemma 3.5 there are no active processors in one direction between v and v', all processors in the interval between v and v' have color $\alpha$, and v and v' have send their $<\alpha, p>$ messages out away from each other. As u is already $\alpha$-colored and we are in the "non-collision" case, u must necessarily belong to the interval between v and v'. But this means that the $<\alpha, p>$ message could not have reached u. Thus this subcase cannot occur.

Case II: PNUM(u)$\leq$p, and u receives identical $<\alpha, p>$ messages from both sides (the "collision" case).

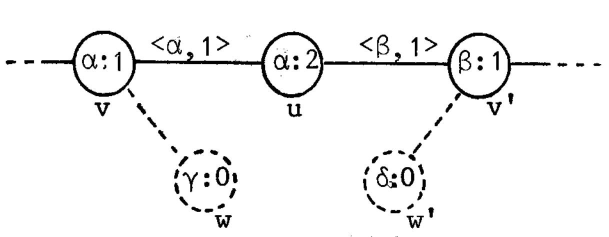
As $\alpha\in\tilde{A}_p$ and u receives $<\alpha, p>$ messages from both sides, there must be two $\alpha$-colored active processors v, v'$\in A_p$ (v$\neq$v') that are responsible for sending the messages. By lemma 3.5 v and v' must be adjacent active processors in phase p, all processors in between v and v' in one direction must have the value $\alpha$ in their ID-register, and v and v' have send their $<\alpha, p>$ messages out away from each other. (By an argument similar to lemma 3.3 one can show that v and v' must be separated by at least one non-active processor on either side, and thus there is "room" for the messages to collide.) It follows that u is located somewhere on the complementary interval between v and v'. As the $<\alpha, p>$ messages approach u from both sides, all processors from v to u and from v' to u become $\alpha$-colored. By the same argument as before there can be no other active processors in phase p, and (hence) $|\tilde{A}_p|=1$. When the messages collide at u, u is the only processor to move to the inauguration state. □

By proposition 3.6, any processor that moves to the inauguration state knows that the election algorithm has terminated. In particular proposition 3.6 implies that the leader is unique, whenever the algorithm terminates. We now prove that algorithm E indeed terminates, by showing that $|\tilde{A}_p|$ gradually decreases for $p\rightarrow\infty$. Let p$\geq$2.

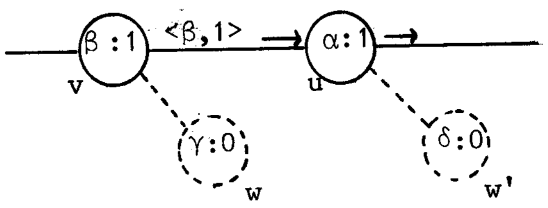
Lemma 3.7 There is a 1-1 mapping from the active colors in phase p to the active colors in phase p-2 that are no longer active in phase p-1.

Proof.

Let u:$\alpha$:p denote that processor u has color $\alpha$ in phase p. We consider the special case of p=2 first. Let $\alpha$ be any active color of phase 2, and let u be an active processor in phase 2 with u:$\alpha$:2. Suppose first

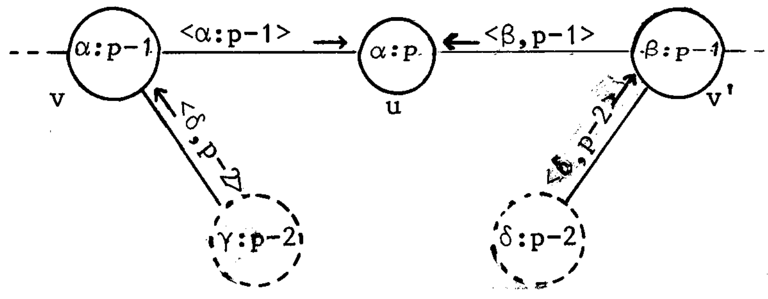


that u is c-active in phase 2, and is activated because it received a message $<\alpha, 1>$ over its left port and a message $<\beta, 1>$ over its right port (see the illustration). Let v and v' be the active processors in phase 1 from which the two messages originate, respectively. We may assume that $\alpha \neq \beta$ and, because u gets color $\alpha$, $\alpha > \beta$. By the way the initialization phase assigns colors to v and v' and observing the direction in which the $\langle \alpha, 1\rangle$ and $\langle \beta, 1\rangle$ messages are sent, v must have a neighbor w on its u-side with w:$\gamma$:0 for some $\gamma < \alpha$ and v' must have a neighbor w' on its u-side with w':$\delta$:0 for some $\delta < \beta$. It is possible that w=w'(=u) and thus that $\gamma = \delta$. Note that there can be no other active processors in phase 1 between v and v', in order that the $\langle \alpha, 1\rangle$ and $\langle \beta, 1\rangle$ messages can both reach u. As these messages travel, the colors $\gamma$ and $\delta$ are eradicated and overrun by $\alpha$ and $\beta$ respectively. Thus we can let $\alpha$ correspond to e.g. $\gamma$, which is an active color from phase 0 that is no longer active in phase 1. Next suppose that u is d-active in phase 2, and is activated because it received a message $\langle \beta : 1\rangle$ from a processor $v \in A_1$ while it was in phase 1 with color $\alpha$ and $\alpha > \beta$ (see the illustration). Arguing as before, v must have a neighbor w on its u-side with w:$\gamma$:0 for some $\gamma < \beta$ and u must have a neighbor w' on the same side with w':$\delta$:0 for some $\delta < \alpha$ (otherwise the phase-1 message of u would collide with the $\langle \beta, 1\rangle$ messsage). Note that



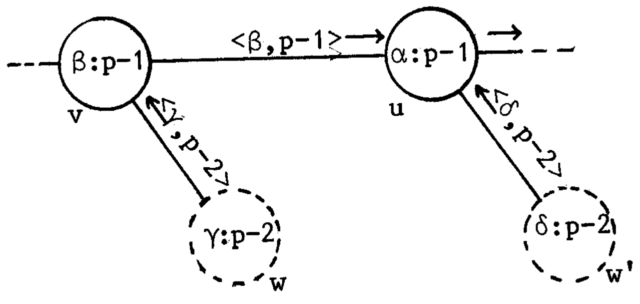
in particular $\alpha > \gamma$ and, by the way the initialization phase colors and activates processors to phase 1, it follows that u$\neq$w. As the $\langle \beta, 1\rangle$ message travels towards u, the color $\gamma$ is eradicated and overrun. Thus we can let $\alpha$ corresponds to $\gamma$ again, because $\gamma$ is indeed an "active" color from phase 0 that is not active in phase 1. In all cases the color $\gamma$ associated with $\alpha$ cannot be associated with any other active color of phase 2, because of the disjointness of the "intervals of discourse" around the active processors u.

For p>2 the argument proceeds in a rather similar way. Again let $\alpha$ be any active color of phase p, and u an active processor of phase p with u:$\alpha$:p. Suppose first that u is c-active in phase p, because it received a message $\langle \alpha, p-1\rangle$ over its left port and a message $\langle \beta, p-1\rangle$ over its right port (see the illustration). Let v and v' be the active processor in phase p-1 from which the messages originate, respectively.



Now because v and v' are active in phase p-1 and both send their phase-(p-1) messages towards u, v must have received a <γ, p-2> messages from the "right" and v' must have received a <δ, p-2> message from the "left" to trigger the activation to phase p-1 and α>γ and β>δ. (Note that the colors γ and δ cannot exist outside of the current interval of discourse.) Let w and w' be the active processors in phase p-2 from which the two messages originate, respectively. It is possible that w=u or u=w', but not both. (Note that w≠w' because an active processor sends its phase message in one direction only.) Clearly the colors γ and δ are overrun by α and β. (By lemma 3.5 one argues that e.g. γ≠β.) Thus we can let α correspond to γ, which indeed disappears as an active color in phase p-1.

Next suppose that u is d-active in phase p, and was activated because it received a <β, p-1> message over (say) its left port while it was in phase p-1 with color α and α>β. Again, let v be the active processor in phase p-1 from which the <β, p-1> message originates. As the <β, p-1> message travels all the way to u, there can be no phase p-1 processor between v and u. Thus u must be active in phase p-1 as well. Observing the direction in which u and v send their phase (p-1) messages, it is necessary that there are processors w, w' ε $A_{p-2}$ to the "right" of v and u respectively, whose phase (p-2) messages are responsible for activating v and u to phase p-1 (together with messages "from the other side"). Let w:γ:p-2 ans w':δ:p-2, where necessarily β>γ and α>δ (this follows because there is no collision of phase-(p-1) messages here and thus u must have decided to send its phase-(p-1) message back in the direction of w'). In particular we conclude that α>γ. Also observe that w≠w' and, by arguing from lemma 3.5, γ≠δ. One easily sees that the color γ is again eradicated and overrun, and (hence) is not active in phase p-1 anymore. Let α correspond to γ. By disjointness of the intervals of discourse, the correspondence so



defined is 1-1. □

Proposition 3.8 Algorithm E always terminates in finitely many phases and (hence) is a correct distributed election algorithm.

Proof.

By lemma 3.2 and lemma 3.7 it follows that for $p \geq 2$ $|\tilde{A}_p| \leq |\tilde{A}_{p-2}| - |\tilde{A}_{p-1}|$. (Note that $|\tilde{A}_0| = N$ and $|\tilde{A}_1| \leq \frac{N}{2}$.) As long as a leader is not found in phase p-1 (thus $|\tilde{A}_{p-1}| > 0$) we have $|\tilde{A}_p| < |\tilde{A}_{p-2}|$, and it follows that necessarily $|\tilde{A}_p| \rightarrow 0$ for $p\infty$. In finitely many phases we must have reached the situation that there is only one active color left and, using lemma 3.5, the election is completed in that very phase. The correctness of algorithm E now follows from propositions 3.4 and 3.6. □

As an interesting aside we note the following property of algorithm E. As algorithm E will "terminate" in precisely one processor u (cf. proposition 3.6), this processor now has the choice to either declare itself as the leader or the processor whose identification number is now stored in its ID-register. In the former case u must send a message carrying its id around the ring to signal that the election is over and to inform every processor of the new leader. In the latter case it is sufficient for u to merely send a 1-bit flag around, as all processors will have the same value in their ID-register already (cf. proposition 3.6). The one processor that finds its own id to be identical to the value in its ID-register upon receipt of the 1-bit flag, will know that it has been elected to be the new leader. We leave it to the reader to choose his favorite among the two possible inauguration procedures.

Finally, we analyse the message complexity of algorithm E for arbitrary bidirectional rings of N processors.

Proposition 3.9 Algorithm E terminates within 1.44..logN phases and uses at most 1.44..logN+O(N) messages.

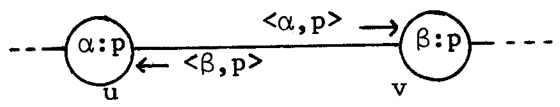
Proof.

Suppose algorithm E terminates in T phases. By proposition 3.6 we have $|\tilde{A}_{T-1}| = 1$. By lemma 3.2 and lemma 3.7 it follows that for $p \geq 2$, $|\tilde{A}_p| \leq |\tilde{A}_{p-2}| - |\tilde{A}_{p-1}|$ and (hence). $|\tilde{A}_{p-2}| \geq |\tilde{A}_{p-1}| + |\tilde{A}_p|$. Let $F_i$ denote the i th Fibonacci number. We claim that for all $0 \leq i \leq T$, $F_i \leq |\tilde{A}_{T-i}|$. The claim is true for i=0 and i=1. Proceeding inductively, suppose the claim is true up to i and $i+1 \leq T$. Then $|\tilde{A}_{T-(i+1)}| \geq |\tilde{A}_{T-i}| + |\tilde{A}_{T-(i-1)}| \geq F_i + F_{i-1} = F_{i+1}$, and the induction step is complete. It follows that $F_T \leq |\tilde{A}_0| = N$ and by known estimates for $F_T$ we conclude that $T \leq \frac{\log N}{\log \psi} + O(1)$, where $\psi = \frac{1}{2}(1+5) = 1.61803...$ (cf. Knuth [3]). Thus algorithm E terminates within about $\frac{\log N}{\log \psi} = 1.44 .. \log N$ phases.

Finally observe that in each phase p, the phase-p messages either collide or are intercepted (stopped) at some processor that is active in the current phase and necessarily sent its phase-p message out away in the other direction. Thus at most one phase-p message will travel over each link in the ring and, consequently, each phase requires at most N messages. The initialization phase is special and requires 2N messages. It follows that algorithm E uses at most 1.44..NlogN+O(N) messages. □

4. <u>Modification of algorithm E for general bidirectional rings</u> . The correctness of algorithm E and its analysis as a distributed election very much depend on the assumption that the links are "half-duplex", i.e., that no two messages bypass each other in a link. (Strictly speaking we only need that no two messages with the same phase-number do not bypass each other in a link.) In reality the processors are completely asynchronous, and two adjacent processors could easily send messages to each other simultaneously, thus violating the assumption on the links. We will demonstrate in this Section that algorithm E can be modified so as to cope with this situation, without any essential increase of the message complexity of the algorithm. We assume the version of algorithm E in which at the time of inauguration, the processor whose id appears in all ID-registers (cf. proposition 3.6) is elected as the unique leader.

In order to describe the necessary modifications, consider a situation in which two phase-p messages "bypass" in a link (see illustration). Thus, suppose a processor u sends a message <α, p> to its neighbor v, while v sends a message <β, p> to u at approximately the same moment. Clearly we have u:α:p and v:β:p. (The notation is from the proof



of lemma 3.7.) Assume that $\alpha>\beta$. Clearly the $<\alpha, p>$ message will do no harm at processor v, as it will be effectively discarded ("stopped") under algorithm E. And processor u will become d-active in phase(p+1) and send a message $<\alpha, p+1>$ to v (where it will overrun $\beta$ and be passed on or otherwise collide). As the messages would have collided at u or v under the earlier assumption, the pattern of communication would have been different but the effect rather the same. However, we will make sure (below) that the same activations occur as could have occured under algorithm E with the link assumption in effect. To achieve this, we analyse the possible situations for u and v (which they can detect). By lemma 3.3(ii) u and v cannot both have been active in phase p. Thus suppose u was active when it sent the $<\alpha, p>$ message and v was not when it sent the $<\beta, p>$ message. In the old situation the $<\alpha, p>$ and $<\beta, p>$ messages would have collided at v, turning v active in phase p+1 with color $\alpha$ and triggering v to send a message $<\alpha, p+1>$ out away from u. (u remains in phase p with color $\alpha$.) The following modification of algorithm E will enable u and v to detect the situation and act so as to make the same steps. Introduce a special extra 1-bit field q in the message format, and let every active processor set the q-field to 1 when it sends out its phase message. The receiving processor will strip off the q-bit and immediately set the field to 0, when its task is merely to relay the message further. (Thus the q-field only remains "active" in the first hop, and serves to inform the receiver that its neighbor is active in the current phase.) Now u and v can spot that their messages bypass and act as follows:

(i) u knows that it is active in phase p and, upon receiving a message $<\beta, p, 0>$ from the link over its phase message was sent out, it will simply discard the incoming message.

(ii) v knows that it is non-active in phase p and, upon receiving a message $<\alpha, p, 1>$ from the link over which it relayed the $<\beta, p, 0>$ message, it knows that u was active and that it must act as if the two messages had collided at v. Thus v turns active in phase p+1 and sends out its phase-(p+1) in the appropriate direction with the q-field to 1 again for one hop. The modification maintains the consistency with algorithm E, and works also when $\alpha<\beta$ and when u is non-active in phase p but v is (i.e., with the corresponding message exchanges).

The same modification works, in fact, also in the remaining case that u and v are both non-active in phase p. Then the <α, p> and <β, p> messages would merely have been relayed, and in the original algorithm the messages would have collided either at u or at v. (In fact, either situation can occur.) As it stands, the "collision" will take place in the form of a type-d activation in the processor with the larger value in its ID-register (u when α>β, and v when α<β) and at the other processor there will be "no effect". Thus consistency is maintained, and no special modification is required. (u and v detect the situation by observing the links over which phase-p messages were send and received, with the q-fields all 0.)

Finally, assume that α=β. In this case u and v will both move into the "inauguration state" according to algorithm E, although only one of them would have done so in the case where the link assumption was in effect. Instead of resolving this case, we note that both u and v will necessarily have the same value in their ID-register and (thus) elect the same processor as a leader. As u and v thus inform the same processor of its leadership, no special further modification is required to avoid conflicts.

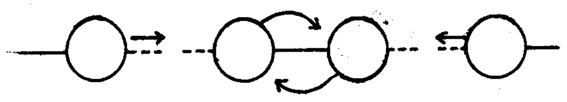
<u>Proposition</u> 4.1 The modified version of algorithm E is a correct distributed election algorithm for fully asynchronous, bidirectional rings.

As the modified algorithm is fully consistent with algorithm E with the link assumption in effect, the number of phases of the modified algorithm is still bounded by 1.44..logN.

<u>Proposition</u> 4.2 The modified version of algorithm E (still) terminates within 1.44..logN phases and uses at most 1.44..NlogN+O(N) messages.

<u>Proof</u>.

We only need to show the bound on the message complexity. For any given phase p, consider the messages exchanged with number p. Normally a link will carry at most one phase-p message (in either the left or the right direction) but in the fully asynchronous case it can happen that two phase-p messages bypass in the same link. In the modified algorithm this is not prevented, but it clear from algorithm E that the two messages will not travel further after bypassing each other. Also, an active



processor in phase p+1 is created. Now consider the ring of processors subdivided into intervals bounded by active processors from phase p that were responsible for sending the phase-p messages. (The intervals travelled by phase-p messages that "ran into" a processor of lower color in the same phase are simply glued on at the end.) If there are k intervals with bypassing messages "in the middle", then there must be at least k+1 separating links in between the intervals over which no phase-p message travelled. Thus the "extra" count of 1 message over the link in the middle of every interval can be charged to one of the separating links, and the total message count in a phase still remains bounded by N. It folows that the message complexity of the modified version of algorithm E is again bounded by 1.44..NlogN+O(N). □

## 5. <u>References</u>.

[1] Bodlaender, H.L., and J. van Leeuwen, New upperbounds for decentralized extrema-finding in a ring of processors, Techn. Rep. RUU-CS-85-15, Dept of Computer Science, University of Utrecht, Utrecht, 1985.

[2] Dolev, D., M. Klawe, and M. Rodeh, An O(nlogn) unidirectional distributed algorithm for extrema finding in a circle, J. Algorithms 3(1982) 245-260.

[3] Knuth, D.E., The art of computer programming, Vol. I: fundamental algorithms, Addison Wesley Publ. Comp., Reading, Mass., 1968.

[4] LeLann, G., Distributed systems-towards a formal approach, in: B. Gilchrist(ed.), Information Processing 77 (IFIP), North-Holland Publ. Comp., Amsterdam, 1977, pp. 155-160.

[5] Peterson, G.L., An O(nlogn) unidirectional algorithm for the circular extrema problem, ACM ToPlaS 4(1982) 758-762.

[6] Santoro, N., E. Korach, and D. Rotem, Decentralized extrema-finding in circular configurations of processors: an improved algorithm, Congr. Numer. 34(1982) 401-412.