# An $O(n \log n)$ Unidirectional Algorithm for the Circular Extrema Problem

GARY L. PETERSON
University of Rochester

Hirschberg and Sinclair recently published a solution to the circular extrema-finding (or election) problem which requires $O(n \log n)$ message passes in two directions around the ring. They conjecture that $\Omega(n^2)$ message passes are required in the unidirectional case. This conjecture is shown to be false. The algorithms presented here are unidirectional, simpler than the Hirschberg and Sinclair solution, use fewer (in fact, optimal) distinct messages, have many fewer total message passes, and require less time.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*distributed applications*; D.4.1 [**Operating Systems**]: Process Management—*synchronization*; F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Distributed algorithms, extrema finding, message passing

## 1. INTRODUCTION

Given $n$ processes in a ring communicating only with message passing to its neighbors, the *circular extrema-finding* (or *election*) *problem* is to select a *maximum* (or *minimum*) process. Each process has a unique value in a set with a total order. These values may be transmitted and compared. All processes are identical (except for their value) and $n$, the number of processes, is not initially known.

Solutions may be compared using several measures. The primary one is the total number of messages transmitted by the processes. Also, one can count the number of distinct messages used and the total message delay time (assuming that each message takes a maximum of one unit of time to be transmitted). In addition, one can consider limiting the number of directions that messages can be transmitted; in this case, a message can be transmitted either unidirectionally or bidirectionally. Obviously, the simplicity of a solution is important.

This research was supported in part by the Office of Naval Research under grant N00014-80-C-0917 and in part by DARPA under grant N00014-78-C-0164.
Author's address: Department of Computer Science, University of Rochester, Mathematical Sciences Building, Rochester, NY 14627.

The original solution to the problem was due to LeLann [5] and required $O(n^2)$ message passes. The algorithm proposed by Chang and Roberts [2] requires $O(n \log n)$ in the average case, but still $O(n^2)$ in the worst case. Both solutions were unidirectional. The algorithm by Hirschberg and Sinclair [4] has $O(n \log n)$ message passes in the worst case, but requires bidirectional communication. Burns [1] has a slightly better bidirectional $O(n \log n)$ algorithm. In addition, Burns formally defines the model and the problem and gives an $\Omega(n \log n)$ lower bound for the bidirectional case. Hirschberg and Sinclair conjecture that any unidirectional solution must be $\Omega(n^2)$. The unidirectional algorithms presented here disprove their conjecture by requiring at most $O(n \log n)$ message passes. Furthermore, these algorithms are better in all respects than the algorithms given in [1, 4].

## 2. THE BASIC ALGORITHM

In this case, consider finding the maximum process and assume all messages are passed to the right. All processes are divided into two categories: *active* and *relay*. All processes are initially active. (The standard assumptions are made about how the election gets started and how it terminates after the maximum is found; see [2].) Active processes operate in phases. Relay processes just pass on any messages they receive. The primary goal is to have the number of active processes be cut at least in half during each phase. Each process starts with its own value as its temporary identifier (*tid*). During a phase, each active process receives the *tid* of its nearest active neighbor to the left and that neighbor's nearest left active neighbor's *tid*. If the first process sees that its left active neighbor has the largest of the three *tids*, then it sets its *tid* to that value and starts the next phase. Otherwise, it becomes a relay process. (In retrospect, this algorithm is a unidirectional simulation of an obvious bidirectional algorithm, using a "sliding" method similar to that given in [6]. Bidirectionally, a process obtains the *tid* of its left and right active neighbors and survives the phase only if it is the largest of the three.)

The text of the election part of the algorithm follows. The procedure send(*msg*) sends *msg* to the right, and receive (*var*) waits until a message has arrived from the left and puts its value into *var*. If the message received is the initial value of the process, then an exception occurs and that process will consider itself elected and will perform whatever action it is required to do. In order to handle the case of two active processes, an active process will never send a *tid* lower than its own.

```
tid := initial value
do forever
begin
  /* start phase */
  send(tid)
  receive(ntid)
  if ntid = initial value then announce elected
  if tid > ntid then send(tid)
                else send(ntid)
  receive(nntid)
  if nntid = initial value then announce elected
  if ntid ≥ max(tid, nntid) then tid := ntid
                            else goto relay
end
```

```
relay:
  do forever
  begin
    receive(tid)
    if tid = initial value then announce elected
    send(tid)
end
```

## 3. ANALYSIS OF THE BASIC ALGORITHM

It is very simple to see that at most $2n\lfloor \log n \rfloor + n$ messages are sent. First one notes that there can be at most $\lfloor \log n \rfloor$ phases to reach one active process. This follows from the fact that any active process survives a phase only if the active process to its left does not survive that phase. Hence of $m$ active processes during a phase, at most $\lfloor m/2 \rfloor$ will survive for $m > 1$. During a phase every process sends (and receives) two messages, and on the last phase the lone active process sends one message which is relayed around to the maximum for a total of $2n\lfloor \log n \rfloor + n$. Hirschberg and Sinclair's solution has an upper bound of $8n\lceil \log n \rceil + 8n$ and Burns's is only slightly better than theirs. Note that if the values of the processes are ordered around the ring, in either direction, then only one process is active after the first phase. Since, in the average case, only $m/4$ processes survive a phase, the average number of messages sent is about $n \log n$ for large $n$.

The correctness of the algorithm follows from two facts: during any phase there is exactly one process whose *tid* is the maximum value, and no process except the maximum will ever see its own initial value. The first fact comes by noting that the process whose *tid* is the maximum value is by definition larger than its two active neighbors, so the active process to its right will have its *tid* equal to the maximum on the next phase. The second fact follows by first noting that, for any process, its value is either the *tid* value of the nearest active process to its right, or its value no longer is being propagated. It therefore follows that in order for some nonmaximum value of a process to cycle around the ring, the largest value must already have passed the process (lest the active process with *tid* equal to the maximum prevents passage of the lower value), but then the smaller value is no longer being propagated since the *tid* value to the right of the process is now the maximum.

Note that only $n$ distinct messages are used. This is obviously the minimum, since we are assuming that the processes are identical (and if one process never sends its value, then none do) and values cannot be broken up into a string of successive messages. The solution in [4] requires $O(n^2 \log n)$ distinct messages, but breaking up messages improves this, at a cost of more total messages sent.

There is a simple upper bound on the message delay time of $O(n \log n)$ which follows from the fact that there are about $\log n$ phases of time $O(n)$ in the worst case. But there is an upper bound of $2n - 1$ that is harder to derive. Consider the maximum valued process, call it $P_0$, and look backward in time starting at the point where it sees its own value for the first time. If there are two processes at the last phase, then one process whose *tid* is not the maximum, call it $P_1$, will wait until the maximum arrives from some other process, call it $P_2$, and retransmit it. $P_2$ cannot send the maximum until it receives the second message in the previous phase from another process ($P_3$) which cannot send it until it receives

the first message from a process further back ($P_4$). In general, there is a process $P_{2j}$ which has to wait until some $P_{2j+1}$ gets its first message from some $P_{2j+2}$ and retransmits it. (If only one process survives at the last phase, then there is no $P_1$.) Clearly the total amount of message delay time for $P_0$ to see its own value is the length of this backward chain, where distance is the number of processes between each $P_{i+1}$ and $P_i$, counting relay processes. The assertion is that the maximum length of this chain is $2n - 1$. This comes from proving that the chain cannot cross $P_0$ twice.

Suppose instead that there are $j$, $k$, $(j > k)$, where $P_0$ lies between $P_{2j}$ and $P_{2j+2}$ and also between $P_{2k}$ and $P_{2k+2}$. If $P_0$ lies between $P_{2j}$ and $P_{2j+1}$, then $P_{2j}$'s *tid* at that time is the maximum, applying the observation used above where the *tid* of the active process to the right of the maximum must be the maximum. Thus $P_{2j-2}$ will see that $P_{2j-1}$'s *tid* is actually *smaller* than $P_{2j}$'s *tid* and will not survive that phase; the chain is broken. Similarly for $P_0$ between $P_{2j+1}$ and $P_{2j+2}$. So the chain can only be broken once (near the end), and the length of the chain is at most $2n - 1$.

For $n$ at least six, the worst case is nearly reachable by having the largest value initially two positions to the left of the second largest, which is two left of the third largest. The length of the backward chain in this case is $2n - 3$. This compares favorably to the worst case upper bound of $6n - 6$ for the algorithm in [4].

If relay processes were allowed to stop intercepting messages to retransmit them (except to note whether the message was their own value), then the number of messages sent would be at most $4n$ and the time would be at worst $2 \log n$.

## 4. IMPROVEMENTS

There is a very simple way to improve the basic algorithm so that even fewer total messages are sent without producing any increases in the other measures. Looking at the bidirectional version of the basic algorithm, it is easy to note that instead of an active process getting the *tid*s of its neighbors at the same time, it can first compare itself to its left neighbor (an A-phase) and then its right neighbor (a D-phase). This means that there are two phases where one sufficed before, but a process can determine earlier that it should become a relay process. Converting back to unidirectional gives the following algorithm:

```
tid := initial value
do forever
begin
  /* Compare to left, an A-phase */
  send(tid)
  receive(ntid)
  if ntid = initial value then announce elected
  if ntid > tid then goto relay
  /* Compare to right, a D-phase */
  send(tid)
  receive(ntid)
  if ntid = initial value then announce elected
  if ntid < tid then goto relay
    else tid := ntid
end
```

```
relay:
  do forever
  begin
    receive(ntid)
    if ntid = intial value then announce elected
    send(ntid)
  end
```

All the major points of correctness and complexity are the same as for the basic algorithm except for the total number of messages sent. The total number of messages is roughly $pn$, where $p$ is the maximum number of phases, since $n$ messages are sent per phase. The following discussion proves that $p$ is at most $c \log n + O(1)$, where $c$ is the inverse of the logarithm (base two) of the golden ratio $\varphi = (1 + \sqrt{5})/2$, and therefore at most $1.440420\ldots\ n \log n + O(n)$ messages are sent. Number the phases in reverse order so that $p$ is the first phase and 1 is the last, after which only one active process remains. Let $m_k$ denote the number of active processes left at the end of the $k$th phase, that is, $m_{p+1} = n$ and $m_1 = 1$. During an A-phase, a process may remain active only if the process to its left became a relay on the previous phase. Similarly for a D-phase, but to the right. This means that the number of processes remaining active at the end of the $k$th phase is at most the number that became relays during the $(k + 1)$st phase. In other words, $m_k \leq m_{k+2} - m_{k+1}$, or $m_{k+2} \geq m_{k+1} + m_k$. This gives a Fibonacci progression, so that $m_k \geq F_{k+1}$, where $F_k$ is the $k$th Fibonacci number. Since $F_{p+1} = \varphi^{p+1}/\sqrt{5} + O(1)$, taking logarithms and solving for $p$ gives the desired result. It is also easy to construct examples which demonstrate that this bound is tight.

The question of lowering the constant factor of $1.44\ldots$ has received some attention. In [3], Dolev et al. apply a technique which originally used the basic algorithm to the improved algorithm and demonstrate a constant factor of $1.356\ldots$, but the number of distinct messages used (and therefore the message size) increases considerably. They point out that the total number of bits, in an appropriate measure, actually increases. While the techniques of [3] can no doubt be further applied to reduce the constant factor, the problem of doing so without increasing the number of bits sent is still open. Note that Burns [1] showed that $\frac{1}{4}n \log n$ messages must be sent when $n$ is a power of two, even if bidirectional communication is allowed.

## REFERENCES


1. Burns, J.E. A formal model for message passing systems. Tech. Rep. 91, Computer Science Dep., Indiana Univ., Bloomington, May 1980.
2. Chang, E., and Roberts, R. An improved algorithm for decentralized extrema-finding in circular configurations of processes. *Commun. ACM 22*, 5 (May 1979), 281–283.
3. Dolev, D., Klawe, M., and Rodeh, M. An $O(n \log n)$ unidirectional distributed algorithm for extrema finding in a circle. IBM Research Rep. RJ3185, IBM Corp., San Jose, Calif., July 1981.
4. Hirschberg, D.S., and Sinclair, J.B. Decentralized extrema-finding in circular configurations of processors. *Commun. ACM 23*, 11 (Nov. 1980), 627–628.
5. LeLann, G. Distributed systems—Towards a formal approach. *Information Processing 77*, Elsevier Science, New York, 1977, pp. 155–160.
6. Smith, A.R., III. Cellular automata complexity trade-offs. *Inf. Control 18* (1971), 466–482.


Received December 1980; revised February 1982; accepted February 1982