Abstract

Leader election causes a unique processor to be distinguished from among a collection of processors. As the study of this problem progressed, increasingly efficient, then more general algorithms emerged. Eventually, Las Vegas algorithms for leader election on rings with identifiers and for those without that achieve asymptotically optimal expected message and bit communication complexity emerged (AAGHK).

In this paper, the same results are achieved with a much simpler algorithm than previously proposed. That is, on a ring of size n without identifiers where an N is know that satisfies $N \leq n < 2N - 1$, a leader is elected using $O(n \log n)$ expected bits. If distinct identifiers are available, then the algorithm can be adapted to use $O(nm)$ expected bits even without knowledge of the ring size, where m is the size of the longest identifier. These results are optimal in communication complexity and in generality.

The algorithm's simplicity facilitates not only its proof of correctness, but also its extension to several other problems. An optimal algorithm for ring orientation follows easily even for situations where deterministic orientation is impossible. The algorithm also generalizes to an optimal (expected bit complexity $O(n)$) Las Vegas algorithm for election in an oriented complete graph. This algorithm, in turn, is adapted to an election algorithm in an oriented sparse graph with no degradation in communication complexity.

1 Introduction

Leader election causes a unique processor, from among a specified subset of the processors, *candidates*, to enter a distinguished final state. This problem is fundamental in that its solution forms a building block for many more involved distributed computations. The communication complexity of leader election on distributed rings with various combinations of properties has been well studied. The reader is directed to the introductory section of [1] for a review of results for various versions of the leader election problem for rings. This paper addresses randomized leader election on rings and on related networks either with identifiers or without (*anonymous*).

If processors lack distinct identifiers then, as was first observed by Angluin [3], *deterministic* algorithms are unable to elect leaders, even if ring size is known to all processors. Itai and Rodeh [12] propose the use of *randomized* algorithms to skirt this limitation. They present a randomized algorithm that elects a leader in an asynchronous ring of known size n using $O(n \log n)$ expected messages of $O(\log n)$ bits each. Abrahamson *et al* [1] improve this result with a randomized algorithm to elect a leader on a unidirectional asynchronous ring with complexity matching the message complexity of the Itai and Rodeh algorithm and using a factor of $\log n$ fewer bits. Furthermore, in this algorithm, ring size need only be known to within a factor of two.

Section 2 of this paper describes another algorithm for randomized leader election on an anonymous ring with message and bit complexity matching that of [1]. It is significantly simpler than the previous one and thus provides a strong illustration of the advantages (in terms of simplicity as well as complexity) that can

be achieved through randomization. The algorithm has evolved from the earlier leader election algorithm, and is most naturally described in an analogous manner.

The simplicity of the election algorithm facilitates its extension or adaptation to the solution of related problems. Section 3 presents solutions to orientation problems and to election problems for other networks that are based on the leader election algorithm of section 2.

2 Leader Election, Attrition and Solitude Detection

Initially set each candidate to be a *contender*. A leader election algorithm must (1) eliminate all but one contender by converting some of the contenders to *non-contenders*, and (2) confirm that only one contender remains. This suggests the separation of leader election into two subtasks called attrition and solitude detection respectively (cf.[1]). A procedure solves the *attrition problem* if, when initiated by every candidate, it never makes all candidates noncontenders, and with probability 1 it takes all but exactly one of these candidates into a permanent state of noncontention. Typically an attrition procedure does not terminate but rather enters an infinite loop in which the remaining contender continues to send messages to itself. An algorithm solves the *solitude detection* problem if, when initiated by a set of processors, it terminates with probability 1 and, upon termination, if there is exactly one initiator then the initiator is in state "yes", and if there is more than one initiator then all initiators are in state "no".

The relationship between leader election and the two subtasks, attrition and solitude detection, was pointed out and exploited by Itai and Rodeh [12]. By tightly interleaving the solutions to the two subproblems, the algorithm of Abrahamson *et al* [1] achieves a lower expected bit complexity than the former algorithm. The algorithm described herein employs a different interleaving strategy, which can be implemented with a simpler attrition procedure. As a consequence, the proof of correctness is significantly simplified.

The next three subsections describe, respectively, each of the three tasks: attrition, solitude detection and interleaving for an asynchronous unidirectional ring. Processors are message-driven. First some subset of processors each initiate one message. Computation proceeds by each processor repeatedly executing the steps: (1) block until some message is received, (2) do the specified local processing, and (3) send the message (possibly null) determined by the local processing. The goal is a Las Vegas leader election algorithm, and therefore the solution is required to terminate with probability 1 and upon termination to have selected exactly one leader.

2.1 The attrition procedure

The leader election algorithm describe here employs an extremely simple attrition procedure. Attrition is initiated by all candidates (the initial *contenders*) for leadership. Let the number of candidates be c. The procedure has an implicit round structure. In each round, each contender independently tosses an unbiased coin,

sends the outcome to its contending successor (via the intervening noncontenders) and waits to receive the coin toss generated by its contending predecessor. The processor becomes a noncontender for the remainder of attrition if and only if it sent a tail and received a head. Those processors remaining in contention proceed with the next round.

Let $\pi_1, \ldots, \pi_m$ be the contenders at the beginning of an arbitrary round j of attrition.

Correctness: If all flips in round j are the same then all m processors remain contenders in round $j+1$; if not all flips are the same then those that flipped heads are guaranteed to be contenders in round $j+1$. Therefore, not all processors can become noncontenders. On the other hand, the probability that a given contender sends a tail and receives a head is $1/4$ as long as there is more than one contender. Hence, with probability 1, the number of contenders decreases to one.

Complexity: When only one contender remains, it continually receives the same flip as it last sent. This infinite loop is broken only by the intervention of solitude detection. Therefore, the complexity of concern for the analysis of attrition is the expected number of bits expended until the number of contenders is reduced to 1. Define the random variables $Y_i, 1 \leq i \leq m$, for round j by:

$$Y_i = \begin{cases} 1 & \text{if } \pi_i \text{ is a contender at the beginning of round } j+1 \\ 0 & \text{if } \pi_i \text{ is a noncontender at the beginning of round } j+1 \end{cases}$$

Given that there are at least 2 contenders in round j, the flip generated by contender π_i in round j is independent of that produced by its nearest preceding contender. Therefore, $\mathrm{E}(Y_i | m \geq 2) = 3/4$. Let random variable X_j be the number of contenders at the beginning of round j. Then $X_1 = c$. So, if $m \geq 2$:

$$\begin{aligned} \mathrm{E}(X_{j+1} | X_j = m) &= \mathrm{E}\left(\sum_{i=1}^{m} Y_i\right) \\ &= \sum_{i=1}^{m} \mathrm{E}(Y_i) \\ &= \frac{3}{4}m \end{aligned}$$

And $\mathrm{E}(X_{j+1} | X_j = 1) = 1$. Therefore:

$$\begin{aligned} \mathrm{E}(X_{j+1}) &= \sum_{m \geq 1} \mathrm{E}\left(X_{j+1} | X_j = m\right) \cdot \Pr(X_j = m) \\ &= \sum_{m \geq 2} \mathrm{E}\left(X_{j+1} | X_j = m\right) \cdot \Pr(X_j = m) + \Pr(X_j = 1) \\ &= \sum_{m \geq 2} \left(\frac{3}{4}m \cdot \Pr(X_j = m)\right) + \Pr(X_j = 1) \\ &= \frac{3}{4}\mathrm{E}(X_j) + \frac{1}{4}\Pr(X_j = 1) \end{aligned}$$

Thus, after $r = \log_{\frac{4}{3}} c < 2.41 \log c$ rounds of attrition, $\mathrm{E}(X_r)$, when c candidates initiate attrition, is a small constant (at most 3)[1]. The expected number of rounds required to reduce from 3 contenders to 1 is a constant and each round of attrition requires exactly n bits. Therefore:

Lemma 2.1 *The expected number of bits communicated by the attrition procedure when there are c candidates, up to the point where there is only one remaining contender is at most* $2.41 n \log c + \mathrm{O}(n)$.

2.2 The solitude detection algorithm

On an anonymous ring of known size, solitude can be verified by confirming that the gap between a contender and its nearest preceding contender is equal to the ring size. Suppose that each processor knows the size n of the ring. A simple algorithm for determining solitude has each contender initiate a counter, which is incremented and forwarded by each noncontender until it reaches a contender, π_x. By comparing the received counter with n, π_x knows whether or not it is alone. As shown in [1], this algorithm can be transformed into a solitude detection algorithm that has constant length messages without any increase in bit communication complexity. It is repeated here for completeness.

Each processor π_x, whether contending or noncontending, maintains a counter c_x, initialized to 0. Let $d_x > 0$ denote the distance from π_x to its nearest preceding contender. The algorithm maintains the invariant:

if π_x has received j bits then $c_x = d_x \bmod 2^j$.

Then if π_x reaches a state where $c_x = n$, there must be $n-1$ noncontenders preceding π_x, so π_x can conclude that it is the sole contender. It remains to describe a strategy for maintaining the invariant.

Contenders first send 0. Thereafter, all processors alternately receive and send bits. If π_x is a noncontender, then π_x is required to send the j^{th} low order bit of d_x as its j^{th} message. Contenders continue to send 0. Suppose a processor, π_y, has the lowest order $j-1$ bits of d_y in c_y. A simple inductive argument shows that when π_y receives its j^{th} message (by assumption the j^{th} bit of $d_y - 1$), it can compute the first j bits of d_y and thus can update the value of c_y to satisfy $c_y = d_y \bmod 2^j$.

The algorithm assumes that n is known exactly. Suppose instead that each processor knows an integer N, such that $N \leq n \leq 2N-1$. Then there can be at most one gap of length N or more between neighbouring contenders. Thus, any gap of less than N confirms nonsolitude. Any processor detecting a gap of $m \geq N$ can determine solitude by initiating a single round to check if the next gap is also m. (For the purposes of leader election, it is sufficient for any contender that detects a gap of N or more to declare itself the leader, since it has confidence that no other processor can do the same.) The modified algorithm is a correct solitude detection algorithm when ring size is known to within a factor of less than two.

[1]Logarithms denoted by "log", without explicit bases, are assumed to be base 2.

Lemma 2.2 *If each processor knows a value N such that the ring size n satisfies $N \leq n \leq 2N-1$, then deterministic solitude detection can be achieved using at most $O(n \log n)$ bits with message length only 2 bits.*

2.3 Interleaving attrition and solitude detection

A leader can be elected on a ring of size $n \in [N, 2N-1]$ with just a coarse interleaving of the attrition procedure and a trivial solitude detection algorithm. First, attrition is run until, with overwhelming probability, there is one remaining contender. This is followed by a single round of solitude detection where each remaining contender sends a counter to measure the gap between itself and its nearest preceding contender. In the rare event that nonsolitude is confirmed, these two steps are repeated by the remaining contenders until solitude is verified. Details appear in [11], where it is shown that even this naive algorithm achieves $O(n \log n)$ expected bit complexity. Its shortcoming is that it cannot be adapted to one that provides an efficient solution to leader election when there are distinct identifiers but no knowledge of ring size nor of the number of candidates. The shortcoming is overcome by a tighter interleaving of attrition and solitude detection messages. As an additional advantage, the refined interleaving achieves early stopping. That is, if attrition proceeds more quickly than expected, then this event will be detected and the time used to elect a leader will correspondingly decrease.

Subsection 2.2 describes a solitude detection algorithm for a static configuration of contenders and noncontenders on the ring. However, when solitude detection is interleaved with attrition, gaps between contenders typically combine into longer gaps before complete gap information is collected. (The algorithm just outlined avoids this complication because it collects complete gap information each time solitude is checked.) To eliminate this problem, restart flags are set to signal when a processor's accumulated gap information is no longer valid.

In each round, a message with 3 bits (an attrition bit, a solitude detection bit, and a restart flag initialized to false) is sent by each contender and propagated to the next contender. If a message arrives at a noncontender, π_p, that was a contender in the previous round (a *new noncontender*), then all processors following π_p, up to and including the next contender, have gap information that is no longer correct. Noncontender π_p signals this situation to these successors by setting the restart flag in the message. Any processors that receives a message with the restart flag set, reinitializes its solitude detection variables. Note that a new noncontender following a contender retains correct gap information, since the gap preceding this new noncontender remains unchanged. In previous rounds, this new noncontender accumulated some bits (at least one) of this unchanged gap. Therefore, it can send the first bit of that gap as the required bit in the solitude detection field of its message.

2.4 The leader election algorithm

The leader election algorithm presented in this subsection is designed for asynchronous anonymous rings of size $n \in [N, 2N-1]$. Recall that processor π_i maintains the following two local variables which are described in subsection 2.2. (The gap from processor π_i to its contending predecessor is denoted d_i.)

j_i: count of the number of messages received containing correct gap information.

c_i: gap counter containing the value $d_i \bmod 2^{j_i}$.

The bit position of the outgoing solitude detection bit may lag behind the bit position of the incoming solitude detection bit. Therefore, an additional variable, o_i, representing the position of the outgoing solitude detection bit, is introduced. For clarity, the following functions and procedures (see subsections 2.1 and 2.2) are assumed as subroutines.

leader: a boolean function that returns true if and only if the local variable c_i has a value in $[N, 2N-1]$.

initializesv: sets gap variables to their initial values in preparation for processing the first bit of gap information. ($j_i \leftarrow 0$, $o_i \leftarrow 0$ and $c_i \leftarrow 0$.)

gapupdate(x): Increments the position counters j_i and o_i. Then uses bit x to update the gap information in the counter c_i in order to maintain the invariant $c_i = d_i \bmod 2^{j_i}$.

nextsvbit(b): produces the b^{th} bit of the number d_i given that $c_i = d_i \bmod 2^{j_i}$ and $j_i \geq b$.

random(x): assigns variable x a random coin flip in $\{h, t\}$.

```
Algorithm Ring LE:
     initializesv; contender ←true;
     WHILE contender AND NOT leader DO
          send(<random(myflip), 0, false>);
          receive(<predflip, svbit, restart>);
          IF restart THEN initializesv;
          gapupdate(svbit);
          IF myflip = t and predflip = h THEN
               o ←0;
               contender ←false;
     WHILE NOT contender DO
          receive(<predflip, svbit, restart>);
          IF restart THEN initializesv;
          gapupdate(svbit);
          IF o = 1 THEN restart ←true;
          send(<predflip, nextsvbit(o), restart>).
```

Correctness: The correctness of attrition and solitude detection have been established in Lemmas 2.1 and 2.2. With probability 1, attrition reduces the set of contenders to one, and solitude detection confirms that one contender is left. It only remains to prove that interleaving, using restart flags, correctly maintains gap information. Define round number r of processor π_i to be the interval in the execution of π_i after it has received r messages and before it has received $r+1$ messages. Let variable v_i^r denote the value at the end of round r of processor π_i's local copy of variable v. Let $\gamma_1^r, \ldots, \gamma_{m_r}^r$ be the processors that are contenders at the beginning of round r. Let d_i^r be the distance from π_i to the nearest predecessor in $\{\gamma_1^r, \ldots, \gamma_{m_r}^r\}$.

Claim 2.3 *For every round number, r, and for every $1 \leq i \leq n$, $c_i^r = d_i^r \bmod 2^{j_i^r}$.*

Proof: Clearly the claim holds at the end of round 1, given the correctness of solitude detection. Consider a message from contender γ_{l-1}^r to contender γ_l^r in round r over gap d_l^r and assume that all accumulated gap information is accurate at the end of round $r-1$. If no processor between γ_{l-1}^r and γ_l^r is a new noncontender, then the gap d_i^r is unchanged from d_i^{r-1} for any processor π_i between γ_{l-1}^r and γ_l^r. The correctness of solitude detection ensures that π_i will accumulate one more bit of information about d_i^r as required. Suppose that some processors $\rho_1, \ldots, \rho_p$ between γ_{l-1}^r and γ_l^r are new noncontenders[2]. They each have $o = 0$ and *contender*=`false` at the beginning of round r. Since `gapupdate` increments o, each of $\rho_1, \ldots, \rho_p$ sets the restart flag. The restart flag is first set by ρ_1, so all processors from γ_{l-1}^r to ρ_1 retain their gap information and acquire one more significant bit. Hence, the claim holds up to and including processor ρ_1. Processor ρ_1 has at least the first two bits of its preceding gap in its local copy of variable c. Therefore, it sends the correct first bit to its successor. All remaining processors π_x up to and including γ_l^r receive a message with the restart flag set and a correct first bit of the new gap, d_x^r. Thus, the claim holds for these processors as well. ∎

Complexity: By the complexity of attrition, *Ring LE* expends less than $2.5n \log c + O(n)$ expected messages and $7.5n \log c + O(n)$ expected bits up until one contender remains. At this point, each message sent by the sole remaining contender drives one bit of gap information back to it. After $\lceil \log N \rceil + 1$ more rounds ($3n \log n + O(n)$ bits) its solitude will be confirmed. Therefore,

Theorem 2.4 *Algorithm Ring LE elects a leader on an anonymous ring of n processors where $n \in [N, 2N-1]$ using $O(n \log n)$ expected bits.*

Ring LE elects a leader under the weakest possible condition on an anonymous ring since solitude cannot even be verified if ring size is not constrained as Theorem 2.4 requires [1]. Furthermore, $\Omega(n \log n)$ expected messages are necessary to elect a leader even when ring size is know exactly [9, 11].

[2]For the attrition procedure of subsection 2.1, in a given round there can be at most one new noncontender between any two contenders. For a generalization of the attrition procedure, described in subsection 2.6, however, it will be possible to have a run of more than one new noncontenders between two contenders. Therefore this possibility is included in the current proof.

2.5 Rings with identifiers

On a ring with distinct identifiers, a processor verifies its solitude by confirming that the preceding contender has the same identifier as itself. Algorithm *Ring LE* can easily be adapted to this situation — see [11] for details. The result is a leader election algorithm for rings with distinct identifiers even when ring size is unknown.

Theorem 2.5 *Algorithm Ring LE can be adapted to elect a leader on a ring of size n where processors have distinct identifiers of length at most m bits using $O(nm)$ expected bits.*

2.6 Tuning the leader election algorithm

Since there is only one attrition bit per message, the number of messages used by *Ring LE* is expected to be more than $n \log_{4/3} c > 2.4n \log c$. When all n processors are candidates for leadership, this exceeds the complexity of the leader election algorithm in [10] for rings with distinct identifiers. The algorithm in [10] has message complexity less than $1.356n \log n + O(n)$ and remains the deterministic leader election algorithm with the best known message complexity.

However, the attrition procedure generalizes directly to one that sends a k-bit random number in place of each coin flip (for details see [11]). Contenders then become noncontenders if and only if they sent a number strictly smaller than the one received. The expected *message* complexity of the revised attrition procedure is at most $(1 - \log(1 + 2^{-k}))^{-1} n \log c + O(n)$, which can be brought arbitrarily close to $n \log c$ by increasing k. Since $(1 - \log(1 + 2^{-k}))^{-1} < 1 + 2\log(1 + 2^{-k}) = 1 + (2/\ln 2)\ln(1 + 2^{-k}) < 1 + (2/\ln 2)2^{-k}$, the complexity of the general attrition procedure is less than $(1 + 2^{-k+2})n \log c + O(n)$ expected messages[3]. Similarly, the $n \log n$ messages sent by solitude detection after there remains only one contender, can be reduced by a factor of $l \leq \log n$ by sending l detection bits with each attrition message. By tuning these parameters, the expected message complexity of leader election can be reduced to within a factor of $(1+\epsilon)$ of $n \log n$ for $\epsilon > 0$. For example, choosing parameter $k = 4$, the expected message complexity of attrition drops to less than $1.096n \log n$. By choosing $l = 4$, the solitude detection adds only $(n \log n)/4$ messages for a total of less than $1.346n \log n$ expected messages. These settings result in a Las Vegas leader election algorithm that has lower expected message complexity than the lowest complexity known for a deterministic algorithm, while retaining both constant length messages (only 9 bits) and simplicity. Clearly, the same packaging idea can be employed whether solitude is detected using identifiers or ring size information.

The optimal parameter setting for minimizing *bit* complexity of the leader election algorithm occurs when messages are short. If k bits are used in an attrition field, and l bits are used in the solitude detection field, then the annotated messages of *Ring LE* are $k + l + 1$ bits each. First $\log_{k'} n$ rounds of attrition are expected until one contender remains, where $k' = 2^{k+1}/(2^k + 1)$. This is followed by $\left\lceil \frac{\log n}{l} \right\rceil$

[3] Logarithms denoted by "ln" stand for the natural logarithm, base e.

rounds to confirm solitude. Therefore, the expected total number of bits is $n(k+l+1)(\log_{k'} n + \lceil \frac{\log n}{l} \rceil)$. This is minimized at $k = 2$ and $l = 2$ resulting in an expected complexity for *Ring LE* of fewer than $8.88n \log n$ bits and fewer than $1.98n \log n$ messages. Of course the bit complexity can be reduced further by running pure attrition for several rounds before interleaving with solitude detection, since it is known that at least $\log n$ rounds will be required anyway.

3 Applications of Las Vegas Leader Election

Because a leader can coordinate further computation, some problems that have no deterministic algorithmic solutions, can be solved easily by employing randomized leader election. For some other problems, the efficient randomized leader election algorithm implies an improvement in the communication complexity over that achievable by deterministic algorithms. This section illustrates these phenomena by extending the simple leader election algorithm of subsection 2.3.

3.1 Function evaluation

In a distributed setting, function evaluation refers to the problem of computing the value of a fixed function of n variables where initially each processor in the network has as input, the value of one of the variables. Evaluation of common functions on rings is examined in some detail elsewhere [2, 8, 7, 9, 11]. The problem of determining the minimum possible complexity of evaluating any nontrivial function on a ring is also addressed separately ([3, 15]). It is simply noted here that all functions on rings can by inexpensively evaluated by preprocessing with leader election.

Once a leader is elected, the leader simply circulates a message which collects all the necessary information to compute the given function. When the message returns, the leader computes the function value locally and announces the result. Thus, there exists an algorithm for evaluating any function in any situation where a leader can be elected using only $O(n)$ additional messages.

For example, SUM can be evaluated by a randomized algorithm on any ring that has size known to within a factor of two, with an expected complexity of $O(n \log n)$ messages and $O(n \log n + n \log S)$ bits, where S is the sum. In contrast, SUM cannot be evaluated deterministically on any anonymous ring unless the ring size is known exactly [7]. Computing AND has complexity $\Omega(n^2)$ messages in the deterministic anonymous model whereas, by employing randomized leader election this can be reduced to $O(n \log n)$ expected bits, even if ring size is known only to within a factor of two.

3.2 Ring orientation

Attiya, Snir and Warmuth [7] first introduced the problem of determining a consistent orientation on an anonymous bidirectional ring when processors have only local labels on their incident edges. Let $\pi_1, \ldots, \pi_n$ be a ring of identical processors, such that each processor π_i has two communication channels, a_i and b_i, each connected

to one of its neighbours. A solution to the *ring orientation* problem distinguishes one channel in $\{a_i, b_i\}$ for each processor, such that all the processors, together with the collection of these distinguished channels, form a unidirectional ring.

Attiya *et al* show that there is no deterministic ring orientation algorithm for rings if the ring size is unknow or if it is known and even. They further show that $\Omega(n^2)$ messages are required in the worst case for any deterministic ring orientation algorithm for ring of known odd size. A deterministic orientation algorithm for rings of known and odd size is provided by Syrotiuk and Pachl in [18] with average complexity $O(n^{3/2})$ messages, assuming that all initial configurations of local orientations are equally likely. With the help of randomization both the complexity barrier and the impossibility barrier disappear.

Theorem 3.1 *Ring orientation reduces to leader election in* $O(n)$ *expected bits.*

Proof: Let *Ring LE* be any leader election algorithm for a unidirectional ring. Consider the following algorithm, which employs *Ring LE* as a subroutine. In addition to the messages used by *Ring LE*, the algorithm uses two message types — *leader* messages of the form <*l*, *finished*> (*l* indicates that this is a *leader* message, and *finished* is a boolean flag), and *orientation* messages. Each processor maintains a local *two-leaders* flag initialized to false.

`Algorithm` *Orientation*:

1. Each processor, π, is initially a candidate and initiates *Ring LE* by sending its first message of *Ring LE* on its *a* link.

2. Upon receipt of an *Ring LE* message on link *b*, π proceeds exactly as in algorithm *Ring LE* and sends its response (if there is one) on its *a* link.

3. Upon receipt of an *Ring LE* message on link *a*, π executes the leader election code for a noncontender and forwards its response on its *b* link.

4. When a processor is elected, it sends a *leader* message on its *a* link with the *finished* flag set to true.

5. A leader message is forwarded around the ring until it is received on link *b* by a processor that sent a leader message. (This recipient is necessarily the originator of the message.) However, any processor receiving a leader message on its *a* link, sets the *finished* flag to false and sets its local *two-leaders* flag to true before forwarding the leader message on its *b* link.

6. When a leader message returns to its originator (necessarily on a *b* link), this processor examines the *finished* flag. If *finished* = true then this leader propagates a final *orientation* message on its *a* link. If *finished* = false, then this leader delays until its local *two-leaders* flag is true.

7. When *two-leaders* is true, both leaders exchange independent random coin tosses until they send and receive opposite tosses. The leader sending heads and receiving tails propagates an *orientation* message on its *a* link.

8. The orientation message is forwarded until it returns to the originator of the message. Each recipient sets its *incoming* link to be the one on which the orientation message is received.

Correctness: Let A (respectively, B) be the subset of processors initially consistent with a clockwise (respectively, counterclockwise) orientation. Steps 1 through 3 perform leader election simultaneously on sets A and B. Since the messages of each election propagate in opposing directions, any interleaving of these two elections cannot disturb their progress. Because at most one of sets A and B is empty, eventually either one or two leaders are elected. Step 6 ensures that an elected leader learns which is the situation for the current computation. If either A or B is empty, step 6 ensures that the sole leader's orientation is adopted by all processors. If neither A nor B is empty, then the delay in step 6 ensures that set A and set B have both selected a leader before the algorithm proceeds. It is then straightforward to check that the number of leaders is reduced from two to one (step 7) and that the ring is oriented consistently with this remaining leader (step 8).

Complexity: One orientation message and at most two leader messages, all of constant length, are propagated once each around the ring accounting for $O(n)$ bits. It is expected that a constant number of exchanges of coin tosses are required to select one leader from two, accounting for an additional $O(n)$ expected bits. All other messages are messages of *Ring LE*. ■

Algorithm *Orientation* can be easily modified to work for any ring with distinct identifiers even in the absence of any knowledge of ring size. Therefore:

Corollary 3.2 *There are Las Vegas orientation algorithms for*

1. *anonymous unoriented rings with size bounded to within a factor of two*
2. *unoriented rings with distinct identifiers and no knowledge of ring size*

that have expected complexity $O(n \log n)$ *bits where* n *is the size of the ring.*

Although the algorithm is described for the situation when all processors start simultaneously, it is easily converted to one tolerating arbitrary wake-up. Processors that have not initiated the algorithm when a first message arrives simply adopt the role of a noncontender for leadership.

The orientation algorithm demonstrates that lack of orientation in a bidirectional ring does not significantly complicate the problem of leader election — a leader can still be elected in $O(n \log n)$ expected bits.

3.3 Leader election in oriented complete graphs

The problem of deterministically electing a leader in a complete network of distinct processors has complexity $\Theta(n \log n)$ messages [13, 4]. With an additional condition however, the lower bound can be violated. Loui, Matsushita and West [14] and

Sack, Santoro and Urrutia [16] studied a version of election on a complete network in which edge labels reflect distance information. Consider a complete network, in which each processor labels its incident edges with the numbers 1 through $n-1$. Let $f(\pi, k)$ denote the processor connected to π via π's link numbered k. The labelling is *consistent* if and only if $f(f(\pi,k),l) = f(\pi,(k+l) \bmod n)$ for every processor π and for every $0 \leq k, l \leq n-1$. The model assumed in [14, 16] is an asynchronous, bidirectional complete network of processors with unique identifiers and consistently labelled incident links. The "sense of direction" constraint in [14] is equivalent to the consistent labelling property. Given this model, [14] presents a leader election algorithm with communication complexity only $O(n)$ messages. The messages used in their solution contain both identifiers and link numbers and thus have order $\log n$ bits each.

Call a network *oriented* if all edges that are present in the network satisfy the consistent labelling constraint. Using randomization, leader election can be solved in $O(n)$ expected bits on an oriented complete asynchronous network, even if processors lack unique identifiers. The randomized algorithm is similar to the leader election algorithm for unidirectional rings of subsection 2.3, except that after each round of attrition, the ring is updated so that subsequent messages need not pass through passive processors. Rather, each contender communicates directly with its nearest contending neighbours. Eventually only one contender remains, and the ring is updated to just a self-loop at that survivor. Thus, this processor's solitude is confirmed automatically. No explicit solitude verification is required. Conceptually, each phase of the algorithm has two parts:

1. a round of attrition on the current ring — contenders send and receive a single coin flip, and

2. ring revision to bypass the processors just eliminated in the attrition round.

A more detailed description follows. The algorithm is executed by each processor. The instruction `send`($< m >: l$) sends message $< m >$ on the link with local label l. The instruction `receive`($< m >$) is a blocking receive; the processor waits until a message arrives on some channel. There are two different types of messages — attrition messages containing a random coin toss and a link number, and gap messages containing a link number or "0". Contenders process these messages alternately: if a message of one type arrives while a processor is waiting for the other, it saves this message and continues to wait. It is a consequence of the algorithm that a processor never has more than one message in a "holding" buffer. The function `random`(x) is assumed to return an unbiased random coin toss and to store the result in variable x.

```
Algorithm Complete graph LE:
     sendlink ←1;
     REPEAT:
1.        send(<random(sflip), sendlink> :  sendlink);
2.        receive(<rflip, receivelink>);
```

```
3.        IF NOT (sflip = t AND rflip = h) THEN
4.             send(<0> :  N−receivelink);
5.             receive(<newgap>);
6.             sendlink ←(sendlink + newgap) mod n
      UNTIL (sflip = t AND rflip = h) OR sendlink = 0
7.    IF sendlink = 0 THEN
8.        announce ''elected''
      ELSE
9.        receive(<0>);
10.       send(<sendlink> :  N−receivelink).
```

Correctness: Let r_i be the total number of times that processor π_i enters the repeat loop of algorithm *Complete graph LE*. Define *round* j for π_i , $j \leq r_i$ to be the j^{th} pass of π_i through the loop. Say that π_i is *active for round* j whenever $j \leq r_i$ and that π_i is *active for* r_i *rounds*. If π_i satisfies the condition "(*sflip* = t AND *rflip* = h)" at the end of round r_i, then say that π_i *went passive* in round r_i. Let $\pi_1^j, \ldots, \pi_{n_j}^j$ denote the substring of $\pi_1, \ldots, \pi_n$ consisting of those processors that are active in round j. Then clearly $\pi_1^1, \ldots, \pi_{n_1}^1 = \pi_1, \ldots, \pi_n$ and $\pi_1^{j+1}, \ldots, \pi_{n_{j+1}}^{j+1}$ is a not necessarily contiguous substring of $\pi_1^j, \ldots, \pi_{n_j}^j$.

The following claim means that processors interpret the implicit round structure consistently. It can be established by induction on the round number and by tracing the effect of the communication on each active processor.

Claim 3.3 *The coin flip sent by* π_i^j *in its* j^{th} *round is received by* π_{i+1}^j *in its* j^{th} *round.*

The claim implies that rounds can be considered in isolation. Let x_i^j denote the value of variable x at the beginning of the j^{th} round of processor π_i. The first two lines following REPEAT constitute the attrition procedure of subsection 2.1. Therefore, given that $sendlink_i^j$ is the number of the link connecting π_i^j to π_{i+1}^j, eventually there will remain one active processor. In round 1, all active processors communicate along the links of a ring with $sendlink_i^1 = 1$ for all i. Suppose that π_i^j goes passive in round j. The attrition procedure guarantees that two adjacent active processors cannot become passive in the same round. Therefore, π_{i-1}^j and π_{i+1}^j must be active processors in round $j+1$. Lines 9 and 6 ensure that $sendlink_{i-1}^j$ is updated to connect π_{i-1}^j to π_{i+1}^j. Alternatively, if π_i^j remains active for round $j+1$, then it sends a 0 (line 4) to π_{i-1}^j so that again π_{i-1}^j connects to its successor via link number $sendlink_{i-1}^j$. Finally, when one active processor remains, its *sendlink* value must be 0 ensuring correct termination.

Complexity: On first inspection of the algorithm, it may appear that messages are generally $O(\log n)$ bits long since link numbers are sent as part of the message. However, large link numbers are only sent toward the end of the algorithm, when fewer processors are active. This is enough to save a factor of $\log n$ bits from the naive complexity analysis.

Specifically, in round 1, $sendlink_i^1 = 1$ for all i and it can at most double in each round. Therefore:

Claim 3.4 *At the beginning of round j $sendlink_i^j \leq 2^{j-1}$ for every active processor π_i^j.*

Let X_j be the number of active processors in round j. From the analysis of the attrition procedure in subsection 2.1, $\mathrm{E}(X_j) = \max\{(3/4)^{j-1}n, 1\}$. Each active processor in round j sends at most $1 + 2\log(2^j) < 2(j+1)$ bits. Therefore, the expected bit complexity is bounded above by:

$$\mathrm{E}(\sum_j X_j \cdot 2(j+1)) = \sum_j (\mathrm{E}(X_j) \cdot 2(j+1)) = \sum_j (3/4)^{j-1}n \cdot 2(j+1) = \mathrm{O}(n).$$

The preceding discussion is summarized by:

Theorem 3.5 *There is a Las Vegas algorithm that elects a leader on an anonymous, asynchronous, oriented complete network with n processors in $\mathrm{O}(n)$ expected bits.*

Algorithm *Complete graph LE* does not require that a processor know which link carried an arriving message. The consistency of the labelling allows processors to compute this number from the link number used by the sender and included in the sender's message. If the model provided this information to the processors, it would be unnecessary to send link numbers with the coin tosses at line 1. The bit complexity would then drop by a constant factor but remain $\mathrm{O}(n)$.

In a possible modification to the model, processors execute a *selective* blocking receive — that is, `receive`(k) causes a processor to wait for a message on link k. Any message arriving on link $l \neq k$ is queued. Such a message is processed only when the processor executes `receive`(l). Algorithm *Complete graph LE* can be adapted to apply to the selective blocking model. After each round of attrition, each processor must be informed of the link to its predecessor as well as to its successor so that it can selectively block on the correct link.

3.4 Leader election in oriented sparse graphs

Santoro [17] discusses the impact of topological information on message complexity. Results for oriented rings and oriented complete graphs are cited as evidence that additional topological information can affect the inherent message complexity of a problem. The oriented ring and the oriented complete network can be viewed as the two extremes of a class of graphs with edges labelled in a globally consistent way. In the oriented ring there is a globally consistent sense of left and right. In the oriented complete network this sense of direction is extended; local link number l connects a processor directly to the processor at distance l via the ring links. Attiya, Santoro and Zaks [6] observed that this transition from an oriented ring to an oriented complete graph results in a drop of message complexity from $\Theta(n \log n)$ to $\Theta(n)$. They also show that the $\mathrm{O}(n)$ complexity can in fact be achieved in a oriented graph that is much sparser than the complete graph. Let $\mathcal{H}$ be a network

consisting of a ring $\pi_1, \ldots, \pi_n$ augmented with the chords (π_i, π_{i+j}) and (π_i, π_{i+n-j}) for $2 \leq j \leq t-1$ at each node π_i. Then a leader for $\mathcal{H}$ can be elected in $2\frac{n}{t}\log n + 3n$ messages. Thus, there is an oriented graph consisting of a ring augmented with $\log n$ chords at each node, such that a leader can be elected in $O(n)$ messages [6].

The randomized algorithm, *Complete graph LE* for leader election on an oriented complete graph can also be adapted to a much sparser graph without significantly increasing the expected bit complexity. Let $\mathcal{G}$ be a network of n processors $\pi_1, \ldots, \pi_n$ such that π_i is connected to π_{i+2^k} via a link with label $k, k = 0, \ldots, \lfloor \log n \rfloor$. $\mathcal{G}$ is an oriented graph with degree $\lfloor \log n \rfloor + 1$. For any d, a message can be sent between π_i and π_{i+d} using at most $\lceil \log d \rceil$ hops. By including a more elaborate *send* and *receive* structure which allows for forwarding, algorithm *Complete graph LE* can be converted to an algorithm which only uses the edges of $\mathcal{G}$. The messages need only be augmented with one additional field, *remaining distance*, leaving the other fields of the original message unchanged. Consider the *send* instruction in line 1 of algorithm *Complete graph LE*. A message is sent directly from its source, say π_s, to its destination, say π_d, on link number *sendlink*. To send it to π_d using edges of $\mathcal{G}$, it is sent instead on link labelled a where $2^a \leq$ *sendlink* $< 2^{a+1}$, and the *remaining distance* field is set to *sendlink*-2^a. Any processor (necessarily a passive one) receiving a message with *remaining distance* not equal to 0, knows that the message is not destined for it. Suppose *remaining distance* $= d$, where $2^b \leq d < 2^{b+1}$. The forwarding processor changes the *remaining distance* field to $d - 2^b$ and forwards the message on its link labelled b. A similar strategy can be used to send the messages of lines 4 and 9 which travel in the opposite direction. The required forwarding direction can be resolved by adopting the convention that *remaining distance* is positive for messages sent in line 1 and negative for those in lines 4 and 9.

Complexity: In round j, successive active processors π_i and π_l satisfy $|i - l| \leq 2^{j-1}$. Therefore, the *remaining distance* field can always be encoded in j bits. So a total of fewer that $4(j+1)$ bits are sent by each active processor in round j. Each message requires at most $\lceil \log 2^j \rceil = j$ hops to reach its destination.

If X_j is the number of active processors in round j, the expected bit complexity is bounded above by:

$$\mathrm{E}\left(\sum_j X_j \cdot 4(j+1)j\right) = O(n) \text{ since } \mathrm{E}(X_j) = \left(\frac{3}{4}\right)^{j-1} n$$

Theorem 3.6 *There is a oriented graph with n processors and $n\lceil \log n \rceil$ links on which a leader can be elected in $O(n)$ expected bits by a Las Vegas algorithm.*

References

[1] Karl Abrahamson, Andrew Adler, Rachel Gelbart, Lisa Higham, and David Kirkpatrick. The bit complexity of randomized leader election on a ring. *SIAM Journal on Computing*, 18(1):12–29, 1989.

[2] Karl Abrahamson, Andrew Adler, Lisa Higham, and David Kirkpatrick. Probabilistic evaluation of common functions on rings of known size. Technical Report TR 88-15, University of British Columbia, 1988.

[3] Karl Abrahamson, Andrew Adler, Lisa Higham, and David Kirkpatrick. Randomized function evaluation on a ring. *Distributed Computing*, 3(3):107–117, 1989.

[4] Yehuda Afek and Eli Gafni. Simple and efficient distributed algorithms for election in complete networks. In *Proc. 22nd Ann. Allerton Conf. on Communication, Control, and Computing*, pages 689–698, 1984.

[5] D. Angluin. Local and global properties in networks of processors. In *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, pages 82–93, 1980.

[6] Hagit Attiya, Nicola Santoro, and Shmuel Zaks. From rings to complete graphs — $\theta(n \log n)$ to $\theta(n)$ distributed leader election. Technical Report SCS-TR-109, Carleton University, 1987.

[7] Hagit Attiya, Marc Snir, and Manfred Warmuth. Computing on an anonymous ring. In *Proc. 4th Annual ACM Symp. on Principles of Distributed Computing*, pages 196–203, 1985.

[8] Hagit Attiya and Mark Snir. Better computing on the anonymous ring. In *Proc. Aegean Workshop on Computing*, pages 329–338, 1988.

[9] Hans L. Bodlaender. New lower bound techniques for distributed leader finding and other problems on rings of processors. Technical Report RUU-CS-88-18, Rijksuniversiteit Utrecht, 1988.

[10] Danny Dolev, Maria Klawe, and Michael Rodeh. An $O(n \log n)$ unidirectional distributed algorithm for extrema finding in a circle. *J. Algorithms*, 3(3):245–260, 1982.

[11] Lisa Higham. *Randomized Distributed Computing on Rings.* PhD thesis, University of British Columbia, Vancouver, Canada, 1988.

[12] Alon Itai and Michael Rodeh. Symmetry breaking in distributed networks. In *Proc. 22nd Annual Symp. on Foundations of Comput. Sci.*, pages 150–158, 1981.

[13] E. Korach, S. Moran, and S. Zaks. Tight lower and upper bounds for some distributed algorithms for a complete network of procesors. In *Proc. 3rd Annual ACM Symp. on Principles of Distributed Computing*, pages 199–207, 1984.

[14] Michael Loui, Teresa Matsushita, and Douglas West. Election in a complete network with a sense of direction. *Information Processing Letters*, 22(4):185–187, 1986.

[15] Shlomo Moran and Manfred Warmuth. Gap theorems for distributed computation. In *Proc. 5th Annual ACM Symp. on Principles of Distributed Computing*, pages 131–140, 1986.

[16] J. Sack, Nicola Santoro, and Jorge Urrutia. $o(n)$ election algorithms in complete graphs with sense of direction. Technical Report SCS-TR-49, Carleton University , Ottawa, Ontario, 1984.

[17] Nicola Santoro. Sense of direction, topological awareness, and communication complexity. *SIGACT News*, 16(2):50–56, 1984.

[18] Violet Syrotiuk and Jan Pachl. Average complexity of a distributed orientation algorithm. Technical Report CS-87-23, University of Waterloo, Waterloo, Ontario, 1987.