

## Simple Algorithms for Distributed Leader Election in Anonymous Synchronous Rings and Complete Networks Inspired by Neural Development in Fruit Flies

Lei Xu

*Audaque Data Technology Ltd.*

*Software Building, 9 Gaoxin Middle First Road, Nanshan District  
Shenzhen 518000, China*

*and*

*Department of Computer Science, University of Oxford*

*Wolfson Building, Parks Road*

*Oxford OX1 3QD, UK*

*E-mail: lei.xu@cs.ox.ac.uk*

Peter Jeavons\*

*Department of Computer Science, University of Oxford*

*Wolfson Building, Parks Road*

*Oxford OX1 3QD, UK*

*E-mail: peter.jeavons@cs.ox.ac.uk*

Leader election in anonymous rings and complete networks is a very practical problem in distributed computing. Previous algorithms for this problem are generally designed for a classical message passing model where complex messages are exchanged. However, the need to send and receive complex messages makes such algorithms less practical for some real applications.

We present some simple synchronous algorithms for distributed leader election in anonymous rings and complete networks that are inspired by the development of the neural system of the fruit fly. Our leader election algorithms all assume that only one-bit messages are broadcast by nodes in the network and processors are only able to distinguish between silence and the arrival of one or more messages. These restrictions allow implementations to use a simpler message-passing architecture. Even with these harsh restrictions our algorithms are shown to achieve good time and message complexity both analytically and experimentally.

*Keywords:* Leader election; cell signalling; bio-inspired algorithms; distributed computing; anonymous networks.

### 1. Introduction

A network of processors is said to be *anonymous* if the nodes do not have unique identifiers. Leader election in anonymous networks is a fundamental problem in distributed computing and has been extensively studied<sup>1–6</sup>. Informally, leader election in an

anonymous  $n$ -node network requires that exactly one node identifies itself as the leader and the remaining  $n - 1$  nodes identify themselves as subordinates.

Choosing a leader node can greatly enhance the effectiveness of a network. Thus, leader election serves as a basic building block in many other distributed algorithms, and has many applications

---

\*Corresponding author

in distributed network deployment, load balancing, wireless sensor networks, radio-communication networks, and in particular in regenerating the lost “token” in local area token ring networks<sup>7–11</sup>. With the advent of large scale ad-hoc networks, finding a scalable, highly efficient, and easily implementable solution to the leader election problem is more and more desirable.

Rings and complete networks are the most widely studied network topologies for leader election in both real distributed network deployment and theoretical algorithm analysis. The main reason is that in practice the leader election problem arises most often in local area token ring networks and radio communication networks (which can most simply be viewed as complete networks). From a theoretical point of view, many interesting challenges in distributed computing can be reduced to the special cases of rings and complete networks<sup>12; 13</sup>. Thus, this paper focuses on the leader election problem in rings and complete networks. We shall assume, for simplicity, that our networks operate synchronously, so that communication between nodes happens at fixed time-steps.

There are many randomised algorithms for leader election in anonymous rings and complete networks in the literature<sup>2; 3; 14–17; 6</sup>. However, most previous algorithms are based on a classical message passing model where complex messages, such as random numbers or counters, are exchanged between processors at each time step (see the summary of previous algorithms in Table 1). Relying on the perfect transmission of complex messages makes many previous algorithms difficult to implement in some settings.

Symmetry-breaking processes similar to leader election also exist in the natural world. For example, during the development of many organisms certain cells specialise to become “leader cells” and the surrounding cells become “subordinate cells” that are coordinated by the leader cells.

As a specific example, during the development of the neural system of the fruit fly *Drosophila*, certain cells in the pre-neural clusters of the fly specialise to become sensory organ precursor (SOP) cells, which later develop into cells attached to small bristles (microchaetes) on the fly that are used to sense the environment<sup>18</sup>.

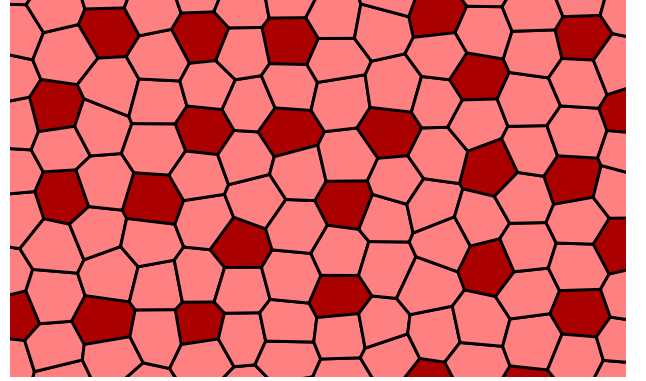


Fig. 1. The selection of SOP cells in the fruit fly: each cell either becomes an SOP or a neighbour of an SOP, and no two SOPs are neighbours.

This process of symmetry breaking in biological systems seems to proceed without using complex messages. In some of the most well-studied systems it is achieved by simple local interactions between certain membrane-bound proteins, notably the proteins Notch and Delta<sup>19</sup>. The transmembrane protein Delta has been shown to have two activities: Delta in one cell can bind to, and transactivate, the transmembrane protein Notch in neighbouring cells<sup>20</sup>; Delta and Notch in the same cell mutually inactivate each other<sup>21; 22</sup>.

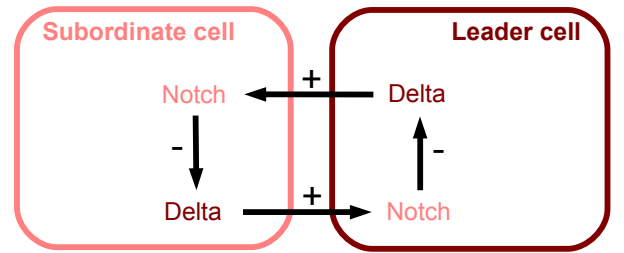


Fig. 2. Notch-Delta signalling constructing a positive feedback to amplify small differences between cells. A slight excess of Delta in the leader cell enables the positive feedback loop that leads to mutually exclusive signalling states in neighbouring cells.

These two interactions constitute a positive feedback mechanism which generates an ultrasensitive switch between two mutually exclusive signalling states: *sending* (high Delta/low Notch) and *receiving* (high Notch/low Delta) (see Figure 2). A slight excess of Delta production in one cell can generate a strong

signalling bias in one direction: the cell becomes a sender (leader) and its neighbours become receivers (subordinates)<sup>23</sup>. At the multicellular level, this lateral inhibition mechanism can break the symmetry among cells and amplify small differences between neighbouring cells arising from random fluctuations.

Inspired by the mechanism of lateral inhibition for selecting local leader cells during the development of the neural system of the fruit fly, we consider solutions to the leader election problem in anonymous rings and complete networks that use only the simplest possible messages, consisting of the presence or absence of a signal. We also impose the restriction that the same message is broadcast to all neighbours, as appears to be the case for inter-cellular signalling in biological networks. Finally, we impose the condition that a node can tell whether one or more neighbours are signalling, but cannot tell which ones or how many, which also appears to be a feature of inter-cellular signalling.

We show that even with these very strict limitations it is possible to perform distributed leader election efficiently in anonymous rings and complete networks. Our approach is similar to the approach used in Ref.<sup>18</sup> to develop a new approach to the maximal independent set selection problem based on the study of the neural system development of the fruit fly. This approach to computing a maximal independent set was further developed in Ref.<sup>24–26</sup>.

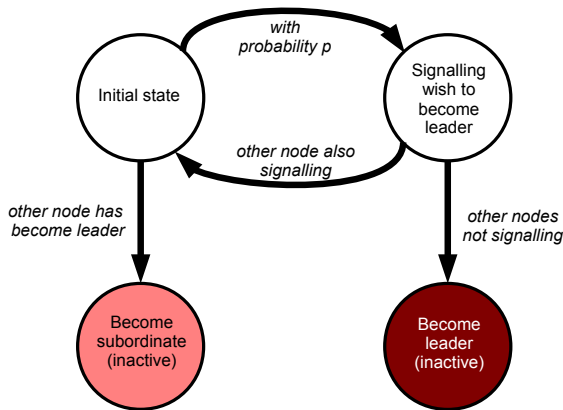


Fig. 3. Automaton description of the computation at each node for leader election.

The computation at each individual node of the network can be described by a simple automa-

ton which is abstracted from the mechanism of the Notch-Delta signalling circuit (see Figure 3). At each step, a node may signal that it wishes to become the leader by moving from its initial state to the state at the top right with probability  $p$ . If some other node also signals at that step, then all the signalling nodes will go back to the initial state; however, if no other node signals at that step, then the signalling node will successfully become the leader with all the remaining nodes becoming subordinates.

Using this approach, we first describe a simple randomised algorithm using single-bit messages that elects a unique leader in an anonymous ring with  $n$  nodes, where the expected number of time steps until the algorithm terminates is  $O(n)$ . Moreover, the expected total number of messages sent by each node is  $O(1)$ . We then extend the same approach to anonymous complete networks. For an anonymous complete network with  $n$  nodes we present three algorithms for distributed leader election that use only single-bit messages. The first algorithm assumes that each of the nodes has knowledge of  $n$ ; the expected number of time steps before termination is  $O(1)$ , and the expected number of time steps at which any given node broadcasts a message is  $O(1/n)$ . The second algorithm assumes that each of the nodes has no knowledge of  $n$ ; the expected number of time steps for this algorithm is  $O(\log^2 n)$ , and the expected number of time steps at which any given node broadcasts a message is  $O(\log n)$ . The third algorithm still assumes no knowledge of  $n$  for each node but improves on the second algorithm by making use of feedback to adjust the probability values used at each node. The expected number of time steps for the third algorithm is  $O(\log n)$ , and the expected number of steps at which any given node broadcasts a message is  $O(1)$ .

The paper is organized as follows: in Section 2, we review previous work on leader election in anonymous rings and complete networks. In Section 3, we describe the model of distributed computation we are using and formally define the leader election problem. In Section 4, we present and analyse an algorithm for distributed leader election in anonymous rings. In Section 5, we present and analyse three algorithms for distributed leader election in anonymous complete networks. In Section 6, we investigate the performance of our algorithms in practice on rings and complete networks of different sizes.

Table 1. Distributed leader election algorithms on anonymous rings and complete graphs with  $n$  nodes

Graph topology	Communication mechanism	Time complexity	Message complexity	Message size (bits)	Information required at each node	Reference
Rings	Asynchronous	$O(n)$	$O(n)$	$\Omega(\log n)$	Graph size	Ref. <sup>2</sup>
					Graph size and FIFO channels	Ref. <sup>3</sup>
		$O(\log n)$	$O(n \log n)$	$\Omega(\log n)$	Graph size	Ref. <sup>14</sup>
		$O(\log n)$	$O(n)$	$\Omega(\log n)$	Graph size and channel names	Ref. <sup>17</sup>
	Asynchronous with bounded expected delay	$O(n)$	$O(n)$	$\Omega(\log n)$	Graph size	Ref. <sup>15</sup>
	Synchronous	$O(\log n)$	$O(n)$	$\Omega(\log n)$	Graph size and channel names	Ref. <sup>17</sup>
		$O(n)$	$O(n)$	1	Graph size and count of active nodes	Ref. <sup>2</sup>
		$O(n)$	$O(n)$	1	<b>Graph size</b>	<b>Table 2</b>
Complete graphs	Asynchronous	$O(\log n)$	$O(n)$	$\Omega(\log n)$	Graph size	Ref. <sup>16</sup>
					Graph size and channel names	Ref. <sup>17</sup>
	Synchronous	$O(\log n)$	$O(n)$	$\Omega(\log n)$	Graph size and channel names	Ref. <sup>17</sup>
		$O(1)$	$O(\sqrt{n} \log^{3/2} n)$	$\Omega(\log n)$	Graph size and channel names	Ref. <sup>6</sup>
		$O(1)$	$O(n)$	1	<b>Graph size</b>	<b>Table 3</b>
		$O(\log^2 n)$	$O(n^2 \log n)$	1	<b>None</b>	<b>Table 4</b>
		$O(\log n)$	$O(n^2)$	1	<b>None</b>	<b>Table 5</b>

## 2. Related Work

Leader election is a symmetry-breaking problem at its core. It was shown in Ref.<sup>27</sup>, using symmetry arguments, that *deterministic* leader election in anonymous rings and complete networks is impossible. Thus, some form of randomised algorithm is required to solve this problem. We now review state-of-the-art randomised algorithms for leader election in both anonymous rings and complete networks, as summarised in Table 1.

### 2.1. Leader election in anonymous rings

The most well-known algorithm for leader election in anonymous rings is the Itai-Rodeh algorithm<sup>2</sup>. The Itai-Rodeh algorithm assumes that the size of the ring is known to all nodes. The synchronous version of the Itai-Rodeh algorithm requires each node of the ring to transmit only one-bit messages to its neighbours and runs in expected  $O(n)$  time using no more than  $2.442n$  bits of messages, on average. We present a slightly simplified version of this algorithm as our first algorithm in Section 4.

The asynchronous Itai-Rodeh algorithm is more complex, because it allows messages to be delivered in the wrong order. In order to avoid being misled by outdated messages, each message contains an additional number, which serves as a "round number". With this idea, the asynchronous Itai-Rodeh algorithm overcomes the difficulty of messages becoming outdated and successfully elects a unique leader in a ring in expected  $O(n)$  rounds with an expected  $O(n)$  messages transmitted around the ring, but the messages are now longer, requiring  $\Omega(\log n)$  bits.

Fokkink and Pang proposed an alternative to the asynchronous Itai-Rodeh algorithm<sup>3</sup>. Their algorithm assumes FIFO channels in order to omit the round numbers used in the asynchronous Itai-Rodeh algorithm. The algorithm by Fokkink and Pang is finite-state which means that the properties of the algorithm can be automatically verified using model checking<sup>3</sup>.

Bakhshi et al. proposed another asynchronous leader election algorithm for anonymous rings<sup>14</sup>. As in the asynchronous Itai-Rodeh algorithm, the algorithm does not make any assumption about the

channel behaviour but assumes that the ring size is known to all nodes. Similarly to the asynchronous Itai-Rodeh algorithm, the algorithm by Bakhshi et al. includes a round number in each message. Thus, the message size is  $\Omega(\log n)$ . The algorithm has been shown to successfully elect a unique leader in a ring with high probability within an expected  $O(\log n)$  rounds using an expected  $O(n \log n)$  messages<sup>14</sup>.

In Ref.<sup>15</sup>, Bakhshi et al. proposed a fast leader election algorithm for asynchronous anonymous rings where there is a known bound on the *expected* message delay. This assumption strengthens the properties of asynchronous networks and hence makes the algorithm fall between the extremes of synchronous and asynchronous. In this algorithm, there is a type of message of size  $\Omega(\log n)$  called the hop-counter that is passed around the network. Each node has a variable  $d(A)$  to store the highest received hop-counter value. As a result, the algorithm successfully elects a unique leader in expected  $O(n)$  rounds using an expected  $O(n)$  messages. Bakhshi et al. claim the algorithm in Ref.<sup>15</sup> as the first leader election algorithm having linear expected time and message complexity in the settings of asynchronous anonymous rings with no fixed bound on the message delay.

## 2.2. Leader election in complete networks

Afek and Matias addressed the problem of electing a leader in an anonymous, asynchronous network of arbitrary topology in Ref.<sup>16</sup>. They also considered the special case of complete networks and proposed an algorithm for leader election in anonymous complete networks. The algorithm has time complexity  $O(\log n)$  and message complexity  $O(n)$ . In the algorithm, each node has knowledge of the graph size and the messages used in the algorithm have a size of  $\Omega(\log n)$ .

In Ref.<sup>17</sup>, Ramanathan et al. proposed an efficient randomised algorithm for leader election in large-scale distributed systems. The algorithm has both synchronous and asynchronous versions and works for a general topology, thus it also works for rings and complete networks. This algorithm guarantees correctness with high probability and has time complexity  $O(\log n)$  and message complexity  $O(n)$ . The algorithm consists of two phases and uses very sophisticated techniques including probabilistic quorums to determine the leader in the second phase.

Nodes in the network are assumed to have knowledge of the size of the network, and they are assumed to be able to distinguish between channels to transmit different messages to different neighbours. The message size used in the algorithm is  $\Omega(\log n)$ .

Recently, Kutten et al. proposed a randomised algorithm for distributed leader election in synchronous anonymous complete networks (see Ref.<sup>6</sup>). In this algorithm, all nodes require knowledge of the network size and each node can sample its neighbours and then send in each round one message of size  $\Omega(\log n)$  to each selected neighbour. The algorithm is simple but also achieves very good worst-case performance: it runs in expected  $O(1)$  rounds and uses only  $O(\sqrt{n} \log^{3/2} n)$  messages. In addition, Kutten et al. also gave a lower bound in Ref.<sup>6</sup> showing that  $\Omega(\sqrt{n})$  messages are needed for any leader election algorithm in complete networks which succeeds with probability at least  $1/e + \varepsilon$  for any constant  $\varepsilon > 0$ . This serves as the first non-trivial lower bound for randomised leader election in complete networks.

## 3. Definitions

As shown in Section 2, previous work on leader election in anonymous rings and complete networks generally relies on a classical message-passing model where complex messages are exchanged between processors of the network. The techniques used are often very complex and sophisticated. However, the complex nature of the algorithms and the requirements for accurate transmission of multi-bit messages can impede the implementation of these algorithms for solving leader election problems that arise in the real world. By contrast, our approach is to investigate the performance of a simple algorithmic framework that uses only single-bit messages.

### 3.1. Distributed computation model

We adopt the widely-used Linial model of distributed computing<sup>28</sup>, where a network is composed of a set of processors together with a set of communication links (channels) between pairs of processors. If there is a link (channel) between two processors, these two processors are said to be *neighbours*.

A distributed network with  $n$  processors corresponds to a directed graph  $G = (V, E)$  with  $n$  vertices. Each vertex holds a processor, which we will refer to as a node in the network, and each (directed)

edge  $(u, v)$  corresponds to a communication link between the processor at node  $u$  and the processor at node  $v$ .

If the graph  $G$  is a cycle, of the form  $(\{1, 2, \dots, n\}, \{(i, i+1) \mid i = 1, 2, \dots, n-1\} \cup \{(n, 1)\})$ , then the network is said to be a ring. If the graph  $G$  is a complete graph, of the form  $(\{1, 2, \dots, n\}, \{(i, j) \mid i, j \in \{1, 2, \dots, n\}, i \neq j\})$ , then the network is said to be a complete network.

Linial's distributed computation model is a synchronous system and all processors operate in a lock-step fashion. We assume that all processors wake up and start their computation at the same time step. During each time step, all processors act in parallel and carry out the following operations sequentially<sup>1</sup>:

- (1) Optionally send a message along one or more outgoing communication links;
- (2) Optionally receive any messages sent by neighbours;
- (3) Perform arbitrary local computation.

The computation is said to be complete only when the local computations at every vertex have terminated. Note that any message that is not received by a node immediately after being sent is assumed to be lost.

In such a model, the main complexity measure for both deterministic and randomised algorithms is the number of synchronous communication rounds required to complete the computation, i.e., the *time complexity*. Another complexity measure considered sometimes is the overall number of messages communicated, i.e., the *message complexity*.

The distributed computation model we use is based on this model, but taking inspiration from cell signalling mechanisms in neural systems we impose the following severe additional conditions:

- (1) At each time step, each processor either keeps silent or broadcasts a fixed signal on all its outgoing communication links;
- (2) At each time step, each processor can tell whether at least one neighbour has broadcast the fixed signal, but cannot tell how many of them have done so, or which ones.

Because of these restrictions, our algorithms can be implemented using very simple communication mechanisms such as radio waves, optical signals, or even chemical signals, as in biological intercellular

signalling<sup>29; 30</sup>. Using a small number of simple messages, allows an implementation to use fewer communication resources and less energy, and this may be crucial in some applications<sup>31</sup>.

### 3.2. The leader election problem

**Definition 1 (Leader Election).** *Given a network of processors, the leader election problem requires exactly one of the nodes to decide to be a leader node, and all remaining nodes to decide to be a subordinate node.*

When a network of processors is solving the leader election problem, each node is in one of the three states: *undecided*, *leader*, *subordinate*. All the nodes begin in the undecided state which is an active state, but eventually terminate in the leader state or the subordinate state which are inactive states. Moreover, exactly one node must terminate in the leader state, and all remaining nodes must terminate in the subordinate state.

**Definition 2 (Anonymous Network).** *A network is called anonymous if the processors in the network cannot distinguish each other by comparing unique identifiers.*

Designing algorithms for anonymous networks is important because transmitting identifiers between processors may be expensive and unreliable in some cases. Moreover, it may become difficult to correctly assign distinct identifiers to a large number of processors in a distributed network<sup>15</sup>.

**Definition 3 (Uniform Algorithm).** *An algorithm  $\mathcal{A}$  on an  $n$ -node network is said to be uniform if it makes no use of any information about the total number of nodes  $n$ , and hence is defined in exactly the same way for all networks, regardless of their size. If the value of  $n$  is used by  $\mathcal{A}$ , then the algorithm  $\mathcal{A}$  is said to be non-uniform.*

It has been shown that no uniform algorithm can elect a leader in an anonymous ring<sup>27; 15</sup>. Hence we will present a non-uniform algorithm for this problem where each node in the network is assumed to know the value of  $n$ .

On the other hand, we will show that it is possible to construct a uniform algorithm for leader election in an anonymous complete graph. Hence we will present both uniform and non-uniform algorithms for

Table 2. An algorithm to execute at each node of an  $n$ -node ring to elect a unique leader

<p><b>Global constants:</b> <math>p = 1/n</math> : probability value for each node;  <b>Local variables:</b> CANDIDATE : Boolean flag, initialised to FALSE.</p> <pre> 1. <b>while</b> active <b>do</b>  2.     With probability <math>p</math>: 3.         set CANDIDATE := TRUE; 4.         <b>send</b> signal on outgoing edge;  5.     <b>for</b> the next <math>n - 1</math> time steps <b>do</b> 6.         <b>receive</b> any signals on incoming edges; 7.         <b>if</b> any signal received <b>then</b> 8.             <b>send</b> signal on outgoing edge; 9.             set CANDIDATE := FALSE;  10.    <b>if</b> CANDIDATE = TRUE <b>then</b> 11.        <b>send</b> signal on outgoing edge; 12.        Terminate as leader  13.    <b>for</b> the next <math>n - 1</math> time steps <b>do</b> 14.        <b>receive</b> any signals on incoming edges; 15.        <b>if</b> any signal received <b>then</b> 16.            <b>send</b> signal on outgoing edge; 17.            Terminate as subordinate </pre>
--

this problem. The non-uniform algorithm we present has a lower time and message complexity, but requires each node to know the value of  $n$ .

#### 4. Leader Election in Anonymous Rings

We first present an algorithm for leader election in anonymous rings, as shown in Table 2. This algorithm is a modified and simplified version of the algorithm first presented in Ref.<sup>2</sup>.

The basic idea of the algorithm in Table 2 is that in each iteration of the while loop each node attempts to elect itself as leader with probability  $1/n$ . If it chooses to do this, then it sends a signal which is passed all around the ring. If more than one node signals, then they will all receive signals from other nodes and give up their current attempt to elect themselves as leader. However, if exactly one node signals, then that node will not receive any signals from other nodes and will terminate as leader after sending one further signal. The remaining nodes will then receive this additional signal as it is passed around the ring and terminate as subordinates.

The computation in each iteration of the while loop consists of four phases. In the first phase, each node signals to its clockwise neighbour with probability  $1/n$ . In the next phase, any signals that have

been sent are passed around the ring for  $n - 1$  steps. If a node receives a signal in this phase then it knows that some other node in the ring has sent a signal and is trying to become the leader, so it abandons any current attempt to become a leader. In the third phase, any node that is still trying to become the leader knows that no other node has signalled, and so it succeeds in becoming the leader. It then sends a final signal which is passed around the ring in the final phase causing each node that receives it to terminate as a subordinate.

**Theorem 4.** *The algorithm described in Table 2 correctly solves the leader election problem on a ring with  $n$  nodes.*

**Proof.** The computation described by the algorithm in Table 2 halts if and only if each node becomes inactive. We need to show that when the computation halts, exactly one node has terminated as leader and all remaining nodes have terminated as subordinates.

To terminate as leader, a node must set CANDIDATE := TRUE in the first phase, and not receive any signals in the second phase. This happens if and only if this node is the only node in the ring that signals in the first phase.

Similarly, to terminate as subordinate, a node must receive a signal in the final phase. This hap-

pens if and only if exactly one other node in the ring signals in the first phase.  $\square$

**Theorem 5.** *When the algorithm described in Table 2 is executed at each node on a ring with  $n$  nodes, the expected total number of time steps before all nodes terminate is  $O(n)$ . Moreover, the expected number of signals sent by each node is  $O(1)$ .*

**Proof.** Note that each iteration of the while loop in line 1 of Table 2 requires a total of  $2n$  time steps (unless it terminates sooner than this). For convenience, we will refer to each iteration of the while loop as a *round* consisting of  $2n$  time steps.

We first consider the time complexity of the algorithm described in Table 2. We have seen that a node running this algorithm will terminate if and only if exactly one node in the ring attempts to become leader in phase one of some round. The probability that this happens is the probability of getting exactly one success in  $n$  independent Bernoulli trials with probability  $p = 1/n$ , which is

$$P_{\text{terminate}} = np(1-p)^{n-1} = (1-1/n)^{n-1} \geq 1/e.$$

Thus, in each round, the computation halts with probability at least  $1/e$ , and hence the expected number of rounds taken before the algorithm halts is  $O(1)$ . Since each round consists of (at most)  $2n$  time steps, the expected number of time steps taken by the algorithm is  $O(n)$ .

To compute the expected number of signals sent by each node we note that the signals sent in phase one will be repeated  $n-1$  times as they are sent around the ring in phase two. The expected total number of signals sent in phase one (across all  $n$  nodes) is one, so the total expected number of signals in phases one and two is  $n$ , and the expected number of signals per node is one. Since the expected number of rounds is constant, by linearity of expectation we obtain that the expected number of signals per node sent in the first and second phases is constant. Finally, each node sends exactly one signal in the third or fourth phase, before terminating.  $\square$

The algorithm described in Table 2 cannot be adapted to give a uniform algorithm because the value of  $n$  is used in an essential way in line 5 and line 13 to ensure that a signal is passed all around the ring (and no further). However, in the next section we will see that variations of this simple algorithm

can be used to obtain both a non-uniform and a uniform algorithm for anonymous complete networks.

## 5. Leader Election in Anonymous Complete Networks

### 5.1. A simple non-uniform algorithm

We first investigate non-uniform leader election in anonymous complete networks where nodes have knowledge of the total number of nodes  $n$ . Our algorithm for this problem is adapted from the algorithm for rings presented in the previous section, and is shown in Table 3.

The basic idea of the algorithm described in Table 3 is similar to the algorithm described in Table 2: each node attempts to elect itself as leader with probability  $1/n$ . If it chooses to do this, then it sends a signal to all other nodes. If more than one node signals, then they will all receive signals from other nodes and give up their current attempt to elect themselves as leader. However, if exactly one node signals, then that node will not receive any signals from other nodes and will send a further signal and terminate as leader. If this happens then at the next step all remaining nodes will receive this signal and terminate as subordinates.

The computation in each iteration of the while loop again consists of four phases. In the first phase, each node signals to all other nodes with probability  $1/n$ . In the second phase, any node that receives a signal knows that some other node in the network has sent a signal and is trying to become the leader, so it abandons any attempt to become a leader. In the third phase, any node that is still trying to become the leader knows that no other node has signalled, and so it succeeds in becoming the leader. It then sends a final signal to all other nodes. Any node that receives such a signal in the final phase terminates as a subordinate.

Note that in a complete network signals are communicated to neighbouring nodes directly in one step, so the algorithm described in Table 3 does not need to relay the signals in order to communicate with all of the nodes, as the previous algorithm did.

**Theorem 6.** *The algorithm described in Table 3 correctly solves the leader election problem on a complete network with  $n$  nodes.*

**Proof.** Similar to the proof of Theorem 4.  $\square$



Table 3. An algorithm to execute at each node of an  $n$ -node complete network to elect a unique leader

**Global constants:**  $p = 1/n$  : probability value for each node;  
**Local variables:** CANDIDATE : Boolean flag, initialised to FALSE.

```

1. while active do
2.     With probability  $p$ :
3.         set CANDIDATE := TRUE;
4.         send signal on all outgoing edges;
5.     receive any signals on incoming edges;
6.     if any signal received then
7.         set CANDIDATE := FALSE;
8.     if CANDIDATE = TRUE then
9.         send signal on all outgoing edges;
10.        Terminate as leader
11.    receive any signals on incoming edges;
12.    if any signal received then
13.        Terminate as subordinate

```

**Theorem 7.** *When the algorithm described in Table 3 is executed at each node of a complete network with  $n$  nodes, the expected total number of time steps before all nodes terminate is  $O(1)$ . Moreover, for any given node, the expected number of time steps where that node chooses to broadcast a signal is  $O(1/n)$ .*

**Proof.** Similar to the proof of Theorem 5, except that each round now consists of only a constant number of time steps.  $\square$

Although the algorithm described in Table 3 requires a knowledge of the value of  $n$ , and hence is non-uniform, it only uses this information to set the value of the probability  $p$ . Hence this algorithm may be easily adapted to obtain a uniform algorithm, as we will now show.

## 5.2. A uniform algorithm

We now adapt our earlier algorithms to derive a uniform algorithm for leader election in anonymous complete networks where nodes require no knowledge at all about the network. The resulting algorithm is shown in Table 4.

**Theorem 8.** *The algorithm described in Table 4 correctly solves the leader election problem on a complete network with  $n$  nodes.*

**Proof.** Note that the algorithm described in Table 4 only differs from the algorithm described in Table 3

in using different probability values which vary over time. Hence the same proof as Theorem 6 can be applied to guarantee correctness.  $\square$

To analyse the time and message complexity we need to consider how the probability values used in the algorithm vary with time. Each node has a variable  $x$  with an initial value of 0 indicating the *stage* of the computation. Each stage  $x$  consists of  $x + 1$  iterations of the while loop in line 5, indexed by the variable  $i$ , which is set to 0 at the start of each stage, and incremented on each iteration. The probability  $p$  that each node attempts to self-select as a leader is set to  $1/2^i$ . Hence the value of the probability  $p$  varies as follows: the value of  $p$  is set to 1 initially, and then gets halved on each successive iteration until the end of the current stage. Thus, the value of  $p$  during the computation will take the following values in successive iterations:  $1, \underline{\frac{1}{2}}, \underline{\frac{1}{2}}, \underline{\frac{1}{4}}, \underline{\frac{1}{2}}, \underline{\frac{1}{4}}, \underline{\frac{1}{8}}, \underline{\frac{1}{2}}, \underline{\frac{1}{4}}, \underline{\frac{1}{8}}, \underline{\frac{1}{16}}, \dots$  (where the underlinings indicate the stages, starting from stage 1).

**Theorem 9.** *When the algorithm described in Table 4 is executed at each node of a complete network with  $n$  nodes, the expected total number of time steps before all nodes terminate is  $O(\log^2 n)$ . Moreover, for any given node, the expected number of time steps where that node chooses to broadcast a signal is  $O(\log n)$ .*

Table 4. A uniform algorithm to execute at each node of an  $n$ -node complete network to elect a unique leader

<p><b>Local variables:</b> <math>x</math> : phase counter;  <math>i</math> : time step counter;  <math>p</math> : local probability value;  CANDIDATE: Boolean flag, initialised to FALSE.</p> <ol style="list-style-type: none"> <li>1. Set <math>x := 0</math>;</li> <li>2. <b>while</b> active <b>do</b></li> <li>3.     Set <math>i := -1</math>;</li> <li>4.     Set <math>x := x + 1</math>;</li> <li>5.     <b>while</b> <math>i &lt; x</math> <b>do</b></li> <li>6.         Set <math>i := i + 1</math>;</li> <li>7.         With probability <math>p = 1/2^i</math>:</li> <li>8.             set CANDIDATE := TRUE;</li> <li>9.             <b>send</b> signal on all outgoing edges;</li> <li>10.         <b>receive</b> any signals on incoming edges;</li> <li>11.         <b>if</b> any signal received <b>then</b></li> <li>12.             set CANDIDATE := FALSE;</li> <li>13.         <b>if</b> CANDIDATE = TRUE <b>then</b></li> <li>14.             <b>send</b> signal on all outgoing edges;</li> <li>15.             Terminate as leader</li> <li>16.         <b>receive</b> any signals on incoming edges;</li> <li>17.         <b>if</b> any signal received <b>then</b></li> <li>18.             Terminate as subordinate</li> </ol>
--

**Proof.** As in the proof of Theorem 4, a node running the algorithm described in Table 4 will terminate if and only if exactly one node in the network attempts to become leader in phase one of some round. As in the proof of Theorem 5, the probability that this happens is the probability of getting exactly one success in  $n$  independent Bernoulli trials with probability  $p$ , which is

$$P_{\text{terminate}} = np(1-p)^{n-1}.$$

To obtain an upper bound on the time complexity, we shall assume that this probability is zero, except when  $p = 1/2^{\lfloor \log_2 n \rfloor}$ .

When  $p = 1/2^{\lfloor \log_2 n \rfloor}$ , we have that  $1/n \leq p \leq 2/n$ , which implies that

$$P_{\text{terminate}} = np(1-p)^{n-1} \geq 1/e^4.$$

Hence, in any stage  $x$  where  $x \geq \lfloor \log_2 n \rfloor$ , the algorithm terminates with probability at least  $1/e^4$ . This means that the expected number of such stages before termination is constant, so the total expected number of time steps for such stages is  $O(\log n)$ .

The total number of time steps before  $x$  reaches the value  $\lfloor \log_2 n \rfloor$  is  $O(\log^2 n)$ . Hence the expected

total number of time steps before termination is bounded by  $O(\log^2 n)$ .

For the message complexity, we know that the expected number of stages before the algorithm halts is  $\lfloor \log_2 n \rfloor + O(1)$  which is  $O(\log n)$ . In each of these stages, for any given node, the expected number of time steps at which it chooses to broadcast a signal is  $1 + 1/2 + 1/4 + 1/8 + \dots \leq 2$ . So, for any fixed node, the expected total number of time steps where that node chooses to broadcast a signal is  $O(\log n)$   $\square$

### 5.3. An improved uniform algorithm

We now propose an improved uniform algorithm for leader election in anonymous complete networks where nodes require no knowledge at all about the network. Unlike the algorithm described in Table 4, the improved algorithm in Table 5 adjusts the local probability value  $p$  based on signals received from the other nodes.

**Theorem 10.** *The algorithm described in Table 5 correctly solves the leader election problem on a complete network with  $n$  nodes.*

Table 5. An improved uniform algorithm to execute at each node of an  $n$ -node complete network to elect a unique leader

**Local variables:**  $p$  : local probability value, initialised to 1;  
 CANDIDATE : Boolean flag, initialised to FALSE.

```

1. while active do
2.   With probability  $p$ :
3.     set CANDIDATE := TRUE;
4.     send signal on all outgoing edges
5.   receive any signals on incoming edges;
6.   if any signal received then
7.     set CANDIDATE := FALSE;
8.     set  $p := p/2$  (halve  $p$ );
9.   else
10.    set  $p = \min\{2p, 1\}$  (double  $p$ )
11.   if CANDIDATE=TRUE then
12.     send signal on all outgoing edges;
13.     Terminate as leader
14.   receive any signals on incoming edges;
15.   if any signal received then
16.     Terminate as subordinate

```

**Proof.** Note that the algorithm described in Table 5 only differs from the algorithm described in Table 3 and the algorithm described in Table 4 in using different probability values. Hence the same proof as Theorem 6 can be applied.  $\square$

Before we determine the time and message complexity of the algorithm described in Table 5, it will be useful to introduce the following Lemma.

**Lemma 11.** *Consider a random walker on the integer lattice  $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$  whose original position is 0. If, in each round, the random walker takes either a unit step left with probability  $p_l$  or a unit step right with probability  $p_r$ , where  $p_l < p_r$  and  $p_l + p_r = 1$ , then with probability 1 the walker will eventually pass position +1, and the expected number of rounds taken for the first passage of the walker to position +1 is  $1/(p_r - p_l)$ .*

**Proof.** This is a well-studied one-dimensional random walk problem<sup>32</sup>.

We consider the generating function

$$\omega(t) = \sum_{n=0}^{\infty} \beta_n t^n$$

where  $\beta_n$  represents the probability that the first passage of the walker to position +1 occurs after exactly

$n$  rounds. Note that

$$\omega(t) = p_r t + p_l t \omega^2(t).$$

Solving this quadratic equation for  $\omega(t)$ , we get

$$\omega(t) = \frac{1 - \sqrt{1 - 4p_l p_r t^2}}{2p_l t}.$$

The probability that the walker ever reaches +1 is given by

$$\sum_{n=0}^{\infty} \beta_n = \lim_{t \rightarrow 1} \omega(t) = \begin{cases} p_r/p_l & \text{if } p_r < p_l \\ 1 & \text{if } p_r \geq p_l \end{cases}$$

The expected number of rounds taken until this event occurs is given by

$$\lim_{t \rightarrow 1} \omega'(t) = \begin{cases} 1/(p_r - p_l) & \text{if } p_r > p_l \\ \infty & \text{if } p_r \leq p_l \end{cases}$$

Thus, when  $p_l < p_r$ , the walker will eventually reach position +1 with probability 1 and the expected number of rounds taken for the first passage of the walker to position +1 is  $1/(p_r - p_l)$ .  $\square$

**Theorem 12.** *When the algorithm described in Table 5 is executed at each node of a complete network with  $n$  nodes, the expected total number of time steps before all nodes terminate is  $O(\log n)$ . Moreover, for any given node, the expected number of time steps where that node chooses to broadcast a signal is  $O(1)$ .*

**Proof.** When exactly one node in the network attempts to become a leader in phase one of some round of the algorithm described in Table 5, the algorithm will terminate with exactly one leader being elected. As in the proof of Theorem 5, the probability that this happens is the probability of getting exactly one success in  $n$  independent Bernoulli trials with probability  $p$ , which is

$$P_{\text{terminate}} = np(1-p)^{n-1}.$$

We denote by

$$P_{\text{double}} = (1-p)^n$$

the probability that no node signals in phase one of some round, and hence the probability value at each node doubles. Finally, we denote by

$$P_{\text{halve}} = 1 - P_{\text{terminate}} - P_{\text{double}}$$

the probability that more than one node signals in phase one of some round, and hence the probability value at each node halves.

The probability value  $p$  at each node will vary over time, however if the algorithm does not terminate, then either all nodes in the network double their probability value  $p$  (when no node signals in phase one) or all halve their probability value  $p$  (when more than one node signals in phase one). Thus all nodes have an equal probability value  $p$  in all rounds.

We divide the possible values for  $p$  into three regions:

**Region 1** :  $R_1 = (0, 1/2n)$

**Region 2** :  $R_2 = [1/2n, 2/n]$

**Region 3** :  $R_3 = (2/n, 1]$

At each round, the value of  $p$  will fall into exactly one of these regions.

We now consider the value of  $P_{\text{terminate}}$  when  $p$  falls into each of these three regions.

We first consider region  $R_2$ , where  $1/2n \leq p \leq 2/n$ . When  $p \in R_2$ , we have

$$P_{\text{terminate}} \geq 1/e^4.$$

Hence the algorithm terminates with probability at least  $1/e^4$  when  $p \in R_2$ .

Now we consider Region 3 where  $2/n < p \leq 1$ . To obtain an upper bound on the time complexity we assume that the algorithm will never terminate when

$p \in R_3$ , i.e., we assume  $P_{\text{terminate}} = 0$  in Region 3. When  $p \in R_3$ , we have

$$P_{\text{double}} \leq 1/e^2.$$

By our assumption, we therefore also have

$$P_{\text{halve}} \geq 1 - 1/e^2.$$

When  $p$  gets closer to 1 within Region 3,  $P_{\text{double}}$  decreases and  $P_{\text{halve}}$  increases.

Hence to obtain an upper bound on the number of steps spent in region  $R_3$  we can use the one-dimensional random walker model considered in Lemma 11 ( $P_{\text{halve}}$  corresponds to  $p_r$  and  $P_{\text{double}}$  corresponds to  $p_l$  where  $P_{\text{halve}} > P_{\text{double}}$  and  $P_{\text{halve}} + P_{\text{double}} = 1$ ). By Lemma 11, in this model the expected number of rounds until  $p$  is halved is bounded by

$$1/(P_{\text{halve}} - P_{\text{double}}) = 1/(1 - 2/e^2).$$

Since  $p \leq 1$ , the expected number of rounds before  $p$  is changed from its initial value in region  $R_3$  to some value in the adjacent region  $R_2$  is at most  $(\log_2 n)/(1 - 2/e^2)$ .

Now we consider the analysis for region  $R_1$  which is similar to the analysis for region  $R_3$ . When  $p \in R_1$ , and  $n \geq 2$ , we have

$$P_{\text{double}} > 0.56.$$

Taking our worst-case assumption that  $P_{\text{terminate}} = 0$  in region  $R_1$ , we have

$$P_{\text{halve}} < 1 - 0.56 = 0.44.$$

When  $p$  gets closer to 0 within region  $R_1$ ,  $P_{\text{double}}$  increases and  $P_{\text{halve}}$  decreases. So at all points in region  $R_1$  we have  $P_{\text{double}} > P_{\text{halve}}$  and  $P_{\text{halve}} + P_{\text{double}} = 1$ . By Lemma 11, when  $p$  first enters Region  $R_1$  from Region  $R_2$ , the expected number of rounds before  $p$  moves back to region  $R_2$  is bounded by

$$1/(P_{\text{double}} - P_{\text{halve}}) \leq 1/(0.56 - 0.44)$$

which is  $O(1)$ .

In summary, considering all three possible regions for probability  $p$  we obtain the following bounds: the expected number of rounds for  $p$  to initially reach  $R_2$  from  $R_3$  is  $O(\log n)$ . Once  $p$  falls into  $R_2$ , the algorithm terminates with probability at least  $1/e^4$  so the expected number of rounds spent in  $R_2$  before terminating is  $O(1)$ . If  $p$  just moves out of  $R_2$  and reaches either  $R_1$  or  $R_3$ , the expected number of rounds until it is pulled back to  $R_2$  is also bounded

by a constant. Hence the expected total number of rounds before all nodes terminate is  $O(\log n)$ .

Each round consists of 2 time steps, so the expected total number of time steps before all nodes terminate is  $O(\log n)$ .

Now we consider the expected number of times that any individual node broadcasts a signal.

Fix an individual node  $v$ . In each round, one of the following 2 things happens:

- Case 1** No signals from other nodes are received at  $v$ , so it doubles its probability value.
- Case 2** One or more signals from other nodes are received at  $v$ , so it halves its probability value.

Now we consider the expected number of times that  $v$  will broadcast a signal during these steps.

Consider first the subsequence of rounds where Case 2 holds, and the new value for  $p$  is the lowest value so far. The expected number of times that  $v$  will broadcast during this subsequence of steps is  $1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots \leq 2$ .

Now consider all the remaining rounds. For each round  $t$  where Case 1 holds, and the probability increases from  $p$  to  $2p$ , we look for a corresponding round  $t' > t$  when the probability next drops back to  $p$  again. (If there is no such round, because the algorithm terminates before the probability returns to  $p$ , then we simply add a dummy round  $t'$  to the end of the sequence.) Note that each round where Case 2 holds is now paired with an earlier round where Case 1 holds.

Now consider the pairs  $(t, t')$ , and define  $B_t$  to be the event that  $v$  broadcasts a signal at time  $t$  or  $t'$ .

If  $B_t$  occurs, then the probability that  $v$  broadcasts at time  $t$  is  $p/(p(1-2p) + 2p(1-p) + 2p^2) > \frac{1}{3}$ . However, if  $v$  broadcasts at time  $t$ , then it will terminate as leader at that step and becomes inactive. Hence the expected number of events  $B_t$  that occur before  $v$  becomes inactive is less than 3.

To sum up, for any given node, the expected number of time steps where that node chooses to broadcast a signal is  $O(1)$ .  $\square$

## 6. Experimental Results

Theorems 5, 7, 9 and 12 provide only worst-case asymptotic upper bounds, and do not tell us the full story of how well these algorithms will perform in

practice. In this section, we investigate the performance of the four algorithms in practice on rings and complete networks of different sizes. The time complexity was measured by the number of time steps, and the message complexity was measured by the number of steps at which an arbitrary node chooses to broadcast a signal. We refer to such a broadcast as a “beep”, and hence consider the number of beeps per node.

We first ran the algorithm described in Table 2 on rings of various sizes. The experiments were conducted on rings of  $n$  nodes where the value of the number of nodes  $n$  was varied from 10 to 200 (in steps of 10). For each value of  $n$ , the algorithm was repeated 100 times. The mean number of time steps and mean number of beeps at an arbitrary node are shown in Figure 4a and Figure 5a respectively. The error bars indicate the standard deviation over 100 trials.

According to these experimental results, the mean number of time steps required to elect a unique leader in an  $n$ -node ring using the algorithm described in Table 2 is approximately  $8n$ , as shown by the black dashed line in Figure 4a, and the standard deviation grows as  $n$  increases. However, the mean number of beeps at an arbitrary node appears to stay constant, and is approximately equal to 4, as shown by the black dashed line in Figure 5a. The standard deviation of this value does not appear to vary significantly with  $n$ .

We also conducted experiments to investigate the performance of the algorithms described in Table 3, Table 4 and Table 5 on complete networks of different sizes. We ran these algorithms on complete networks of  $n$  nodes where the value of the number of nodes  $n$  was varied from 10 to 200 (in steps of 10). For each value of  $n$ , the algorithms were all repeated 100 times. The mean number of time steps and mean number of signals sent by an arbitrary node when running these algorithms are shown in Figure 4b, Figure 4c, Figure 4d and Figure 5b, Figure 5c, Figure 5d respectively. Once again, the error bars indicate the standard deviation over 100 trials.

According to these experimental results, the mean number of time steps required to elect a unique leader in an  $n$ -node complete network using the algorithm described in Table 3 appears to be approximately 8, regardless of the size of the network, as shown in Figure 4b. The standard deviation of this

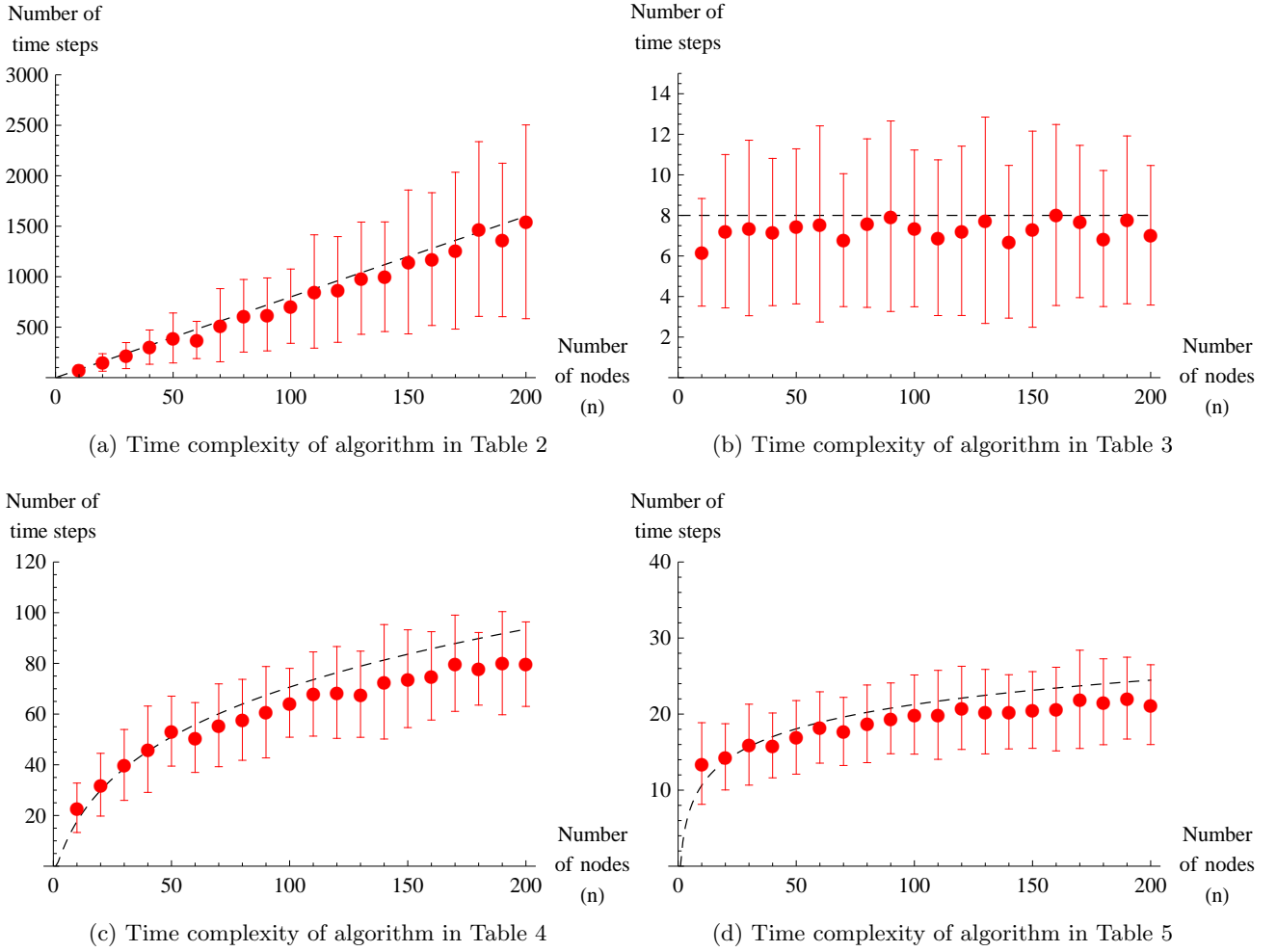


Fig. 4. Experimental results for time complexity on networks of different sizes

value also appears to be small, and independent of  $n$ . For the uniform algorithm described in Table 4 the number of time steps required is much larger, and grows with  $n$ , but appears to be bounded by  $1.6 \log_2^2 n$ , as shown by the dashed line in Figure 4c. Once again the standard deviation of this value appears to be independent of  $n$ . For the uniform algorithm described in Table 5 the number of time steps required grows with  $n$ , but appears to be bounded by  $3.2 \log_2 n$ , as shown by the dashed line in Figure 4d. Once again the standard deviation of this value appears to be independent of  $n$ . The mean number of beeps at an arbitrary node executing the algorithm described in Table 3 appears to be approximately  $4/n$ , as shown by the black dashed line in Figure 5b, and the standard deviation appears to decrease as  $n$  increases. However, the mean number of beeps at

an arbitrary node using the algorithm described in Table 4 appears to be approximately  $1.9 \log_2 n$ , as shown by the black dashed line in Figure 5c, and the standard deviation does not appear to vary significantly with  $n$ . The mean number of beeps at an arbitrary node using the algorithm described in Table 5 appears to be approximately 2, as shown by the black dashed line in Figure 5d, and the standard deviation appears to decrease as  $n$  increases.

It is worth noting that the algorithm described in Table 5 is highly robust, in the sense that it retains its good performance even when various features are changed. To investigate the practical robustness of the algorithm described in Table 5, we denote by  $p_0$  the initial probability value for all nodes, and we denote by  $f$  the increase/decrease factor in each round for all nodes (increase by  $f$  and

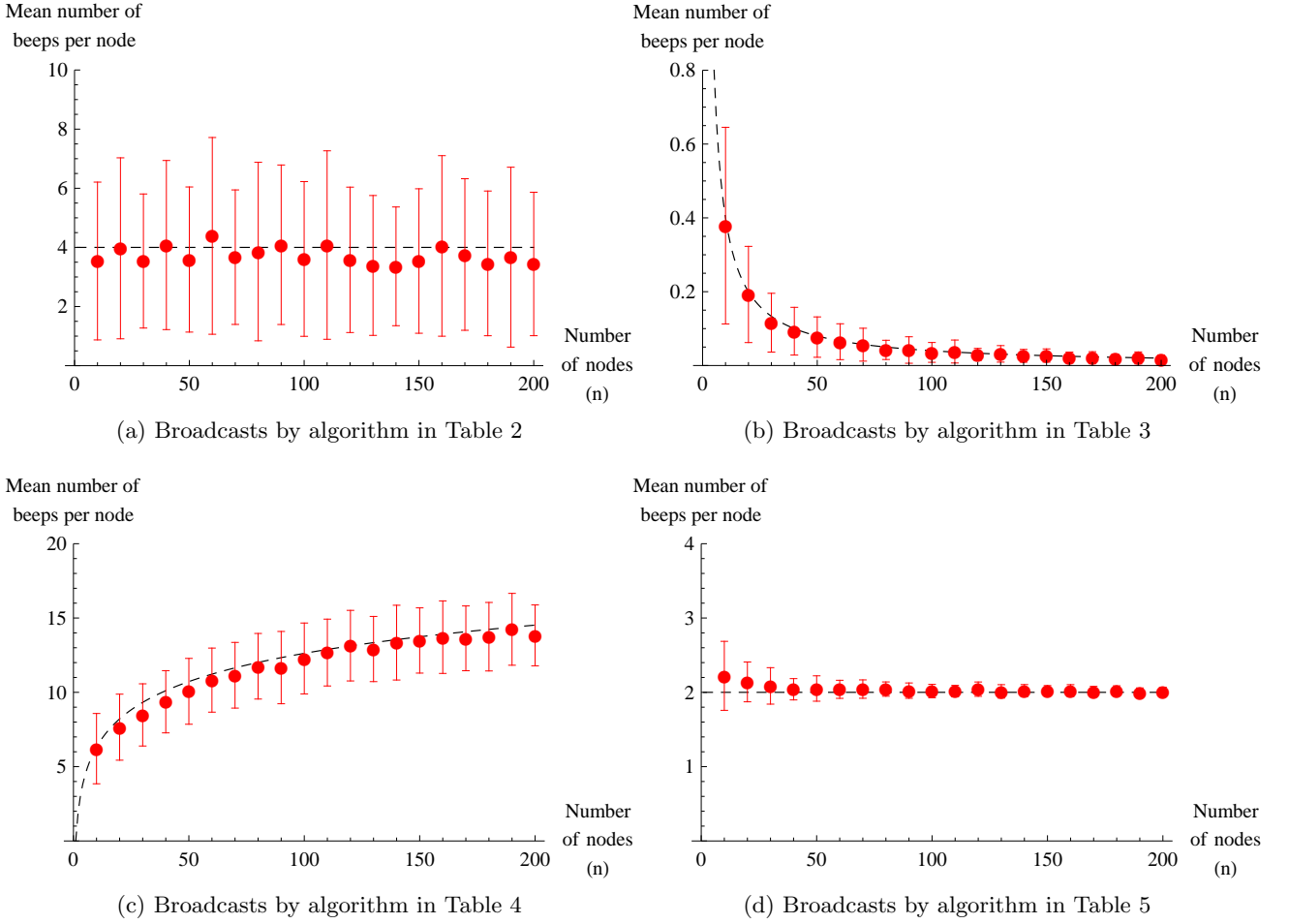


Fig. 5. Experimental results for message complexity on networks of different sizes

decrease by  $1/f$ ). As an example, in the algorithm described in Table 5, we have  $p_0 = 1$  and  $f = 2$ . We conducted experiments of different combinations of  $p_0 \times f = \{1, 1/n, \text{random}\} \times \{2, n, \text{random}\}$  where  $p_0 = \text{random}$  means  $p_0$  is set to an arbitrary random value in  $(0, 1]$  and  $f = \text{random}$  means  $f$  is set to  $1/x$ , where  $x$  is an arbitrary random value in  $(0, 1)$ . For each combination of  $p_0$  and  $f$ , we ran the corresponding algorithms on complete networks of different sizes varying from 10 to 200 (in steps of 10). For each value of  $n$ , the algorithms were all repeated 100 times. Again, the mean number of time steps and mean number of signals sent by an arbitrary node together with their standard deviations are shown in Table 6. For each row in Table 6, we show in block the maximum mean value and show in bold the minimum mean value for all the tested combinations of  $p_0$  and  $f$ .

Whatever the combination of  $p_0$  and  $f$ , the results in Table 6 appear to show that the mean number of time steps never grows faster than logarithmically and the mean number of signals per each node always stays bounded by a small constant as  $n$  increases. Among the tested combinations of  $p_0$  and  $f$ , when  $p_0 = 1, f = n$  and when  $p_0 = 1/n$ , the mean number of time steps appear to be approximately constant. Moreover, the maximum (worst) value for both the mean number of time steps and the mean number of signals per node mostly occurs for the combination of  $p_0 = 1$  and  $f = \text{random}$ ; the minimum (best) value for both the mean number of time steps and the mean number of signals per node mostly occurs for the combination of  $p_0 = 1/n$  and  $f = 2$ .

Table 6. Experimental results demonstrating the practical robustness of the algorithm described in Table 5

Measure	Number of nodes	$p_0 = 1$			$p_0 = 1/n$			$p_0 = \text{random}$		
		$f = 2$	$f = n$	$f = \text{random}$	$f = 2$	$f = n$	$f = \text{random}$	$f = 2$	$f = n$	$f = \text{random}$
Mean number of time steps	10	12.84±5.13	14.06±9.18	13.90±4.39	<b>7.66±4.07</b>	9.58±7.22	8.28±5.21	10.76±5.21	11.38±7.04	10.16±5.13
	20	14.92±4.89	12.44±6.66	14.88±5.69	8.74±5.85	10.54±7.72	<b>8.16±4.73</b>	11.70±5.70	15.38±14.94	13.36±6.04
	30	16.56±4.68	13.18±10.50	15.26±4.54	<b>8.96±6.57</b>	10.22±8.53	9.66±7.20	13.52±6.28	14.16±11.02	13.34±5.26
	40	16.66±7.04	12.54±7.71	17.48±6.01	<b>7.92±4.42</b>	10.14±7.04	8.52±5.60	14.26±5.25	16.02±19.65	14.30±6.69
	50	18.70±5.90	12.32±7.44	18.18±6.38	<b>7.48±4.41</b>	10.62±7.90	8.14±5.38	15.26±6.22	17.66±11.47	15.48±5.72
	60	18.10±5.99	12.76±9.32	18.50±6.31	<b>7.86±5.42</b>	10.80±8.05	8.78±5.89	15.68±7.10	17.24±13.91	15.34±6.06
	70	18.54±5.20	14.46±9.05	19.54±7.00	<b>8.10±5.58</b>	10.90±10.39	8.72±6.13	16.12±5.66	18.62±16.51	16.02±6.49
	80	19.76±6.28	13.64±10.08	19.48±6.68	<b>7.56±5.60</b>	10.06±6.74	8.98±5.33	16.58±6.09	18.08±18.16	16.30±6.23
	90	19.12±5.24	12.24±8.23	20.44±7.23	<b>8.58±5.73</b>	10.74±7.97	9.56±6.78	17.14±6.53	20.98±23.19	16.60±6.21
	100	20.38±6.81	11.68±7.52	19.64±6.27	<b>8.42±5.72</b>	12.56±9.60	9.92±6.53	16.04±6.44	19.16±18.28	16.80±6.47
	110	19.46±5.63	13.30±10.60	20.96±6.12	8.76±5.82	<b>8.40±5.95</b>	9.22±5.49	16.44±4.79	17.70±15.98	16.96±6.02
	120	20.16±5.63	11.88±8.01	19.90±6.20	<b>8.06±5.03</b>	12.12±9.26	9.54±7.19	17.78±6.53	25.74±31.59	17.70±6.21
	130	20.76±5.30	12.32±7.84	21.78±7.53	<b>7.90±5.00</b>	11.12±10.34	10.24±7.30	17.90±6.57	21.10±22.52	17.94±7.40
	140	20.08±4.91	13.62±8.97	21.44±6.47	<b>7.98±4.82</b>	10.20±8.09	10.06±7.52	18.16±6.37	23.30±30.78	18.28±6.80
	150	21.30±6.43	13.20±9.50	21.90±6.78	<b>7.76±6.64</b>	10.04±6.71	10.22±7.92	17.20±4.77	22.50±28.56	20.40±8.47
	160	21.86±6.25	13.44±9.01	21.78±7.13	<b>8.76±5.45</b>	10.60±7.61	9.54±6.36	17.00±5.13	16.50±14.29	18.46±7.29
	170	21.68±5.66	12.82±10.48	21.80±5.90	<b>6.96±3.60</b>	12.44±8.51	10.46±7.50	17.90±6.30	21.66±28.78	20.06±7.65
	180	21.38±4.98	13.56±11.80	23.02±7.75	<b>8.78±5.49</b>	11.38±9.58	10.20±7.89	17.80±6.06	22.14±22.36	18.78±6.80
	190	21.46±5.69	11.62±6.71	21.66±6.91	9.88±7.67	11.38±9.30	<b>9.76±9.32</b>	17.50±6.75	20.16±27.00	19.08±6.27
	200	21.30±5.37	14.16±10.83	22.98±7.26	9.14±6.51	10.48±8.21	<b>8.08±5.50</b>	18.18±6.12	26.26±32.61	20.68±7.58
Mean number of signals per node	10	2.20±0.45	2.17±1.52	2.20±0.56	<b>0.46±0.40</b>	1.62±1.44	0.57±0.54	1.18±0.80	1.45±0.98	1.23±0.72
	20	2.11±0.26	2.19±1.69	2.22±0.36	<b>0.25±0.18</b>	1.16 ±1.83	0.30±0.28	1.10±0.63	1.45±1.05	1.20±0.69
	30	2.06±0.21	1.77±1.14	2.14±0.26	<b>0.16±0.14</b>	0.89±1.01	0.25±0.22	1.13±0.63	1.27±1.05	1.09±0.63
	40	2.06±0.16	2.20±1.89	2.06±0.20	<b>0.13±0.12</b>	1.15±1.55	0.22±0.20	1.05±0.56	1.48±1.09	1.07±0.56
	50	2.03±0.12	2.05±1.50	2.06±0.17	<b>0.08±0.07</b>	0.98±1.13	0.16±0.16	1.02±0.58	1.63±1.28	1.08±0.58
	60	2.02±0.13	1.89±1.17	2.10±0.20	<b>0.08±0.05</b>	0.96 ±1.55	0.15±0.14	0.99±0.62	1.63±1.48	1.14±0.62
	70	2.03±0.14	2.01±1.32	2.06±0.16	<b>0.07±0.05</b>	0.99 ±1.35	0.13±0.12	1.04±0.60	1.64±1.47	0.99±0.54
	80	2.03±0.09	2.05±1.47	2.07±0.15	<b>0.06±0.06</b>	0.99±1.54	0.10±0.10	1.09±0.59	1.57±1.30	1.12±0.62
	90	2.02±0.10	2.27±1.48	2.05±0.12	<b>0.07±0.05</b>	1.10 ±1.31	0.10±0.09	0.96±0.55	1.44±1.09	1.04±0.62
	100	2.05±0.10	2.33±1.61	2.04±0.12	<b>0.05±0.04</b>	1.04±1.42	0.09±0.10	1.01±0.62	1.49±1.18	1.06±0.62
	110	2.04±0.10	2.07±1.54	2.04±0.13	<b>0.05±0.04</b>	1.10 ±1.59	0.08±0.08	1.00±0.59	1.42±1.08	1.00±0.60
	120	2.02±0.09	2.05±1.56	2.05±0.13	<b>0.04±0.03</b>	0.88±1.29	0.06±0.07	1.03±0.52	1.38±1.05	1.07±0.62
	130	2.02±0.07	2.07±1.65	2.05±0.11	<b>0.03±0.02</b>	0.86±1.09	0.08±0.07	1.06±0.58	1.44±1.33	0.99±0.58
	140	2.00±0.08	1.95±1.31	2.06±0.11	<b>0.03±0.03</b>	1.26±1.64	0.09±0.08	1.00±0.60	1.54±1.24	1.04±0.62
	150	2.02±0.08	2.10±1.51	2.02±0.10	<b>0.03±0.03</b>	0.86 ±1.33	0.08±0.07	0.95±0.61	1.22±0.97	0.96±0.56
	160	2.02±0.08	1.98±1.27	2.02±0.09	<b>0.03±0.02</b>	0.98 ±1.28	0.06±0.06	1.03±0.55	1.43±1.31	1.09±0.58
	170	2.02±0.07	2.08±1.36	2.03±0.09	<b>0.03±0.03</b>	0.78 ±1.31	0.06±0.06	1.02±0.58	1.52±1.36	0.97±0.57
	180	2.01±0.07	2.14±1.72	2.04±0.10	<b>0.03±0.02</b>	1.11 ±1.34	0.05±0.06	0.93±0.56	1.55±1.28	1.08±0.57
	190	2.02±0.07	1.86±1.31	2.03±0.10	<b>0.03±0.02</b>	1.24±1.90	0.06±0.05	1.00±0.57	1.51±1.14	1.04±0.62
	200	2.01±0.06	1.87±1.30	2.02±0.08	<b>0.03±0.02</b>	1.06±1.27	0.04±0.05	0.93±0.59	1.52±1.21	1.03±0.56

## 7. Conclusions

Leader election in anonymous rings and complete networks is a fundamental problem of both theoretical and practical importance in distributed computing. In this paper, we have proposed novel algorithms for these problems that are inspired by the intercellular signalling processes in biological systems. These new algorithms are all based on a very harsh model of distributed computing, where only simple fixed signals are broadcast to all neighbours by a small number of processors, as appears to be the case in some inter-cellular processes in living organisms.

The non-uniform algorithms we propose are shown both analytically and experimentally to be very efficient for the leader election problem. The uniform algorithms we propose for anonymous complete networks are slower, but have the advantage that each node requires no global information about the network. All of our algorithms are simple and

easy to implement on a variety of hardware platforms.

Hence we have shown that taking inspiration from biological processes can lead to effective approaches to classical computing problems, as described in Ref.<sup>33; 18; 34</sup>.

## Acknowledgements

The work of Lei Xu is supported by Guangdong Innovative Research Team Program (No. 2011D005), and the Shenzhen Peacock Program (No. 1105100030834361).

## References

1. Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.
2. Alon Itai and Michael Rodeh. Symmetry breaking in distributed networks. *Information and Computation*, 88(1):60–87, 1990.



3. Wan Fokkink and Jun Pang. Variations on Itai-Rodeh leader election for anonymous rings and their analysis in PRISM. *Journal of Universal Computer Science*, 12:981–1006, 2006.
4. Zhenyu Xu and Pradip K. Srimani. Self-stabilizing anonymous leader election in a tree. *International Journal of Foundations of Computer Science*, 17(2):323–336, 2006.
5. Daniel Fajardo-Delgado, José Alberto Fernández-Zepeda, and Anu G. Bourgeois. Randomized self-stabilizing leader election in preference-based anonymous trees. *International Journal of Foundations of Computer Science*, 23(4):853–876, 2012.
6. Shay Kutten, Gopal Pandurangan, David Peleg, Peter Robinson, and Amitabh Trehan. Sublinear bounds for randomized leader election. In Davide Frey, Michel Raynal, Saswati Sarkar, Rudrapatna K. Shyamasundar, and Prasun Sinha, editors, *Distributed Computing and Networking*, volume 7730 of *Lecture Notes in Computer Science*, pages 348–362. Springer Berlin Heidelberg, 2013.
7. Martin Haenggi. Distributed sensor networks: A cellular nonlinear network perspective. *International Journal of Neural Systems*, 13(06):405–414, 2003.
8. Gérard Le Lann. Distributed systems - towards a formal approach. In *IFIP Congress*, pages 155–160, 1977.
9. Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun. The akamai network: a platform for high-performance internet applications. *Operating Systems Review*, 44(3):2–19, 2010.
10. Dariusz R. Kowalski and Andrzej Pelc. Leader election in ad-hoc radio networks: A keen ear helps. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming: Part II*, ICALP '09, pages 521–533, Berlin, Heidelberg, 2009. Springer-Verlag.
11. Jérémie Chalopin, Yves Métivier, and Thomas Morsellino. Enumeration and leader election in partially anonymous and multi-hop broadcast networks. *Fundamenta Informaticae*, 120(1):1–27, 2012.
12. Karl R. Abrahamson, Andrew Adler, Rachel Gelbart, Lisa Higham, and David G. Kirkpatrick. The bit complexity of randomized leader election on a ring. *SIAM Journal on Computing*, 18(1):12–29, 1989.
13. Gurdip Singh. Leader election in complete networks. In *Proceedings of the Eleventh Annual ACM Symposium on Principles of Distributed Computing*, PODC '92, pages 179–190, New York, NY, USA, 1992. ACM.
14. Rena Bakhshi, Wan Fokkink, Jun Pang, and Jaco Van De Pol. Leader election in anonymous rings: Franklin goes probabilistic. In *Fifth IFIP International Conference On Theoretical Computer Science*, pages 57–72. Springer, 2008.
15. Rena Bakhshi, Jörg Endrullis, Wan Fokkink, and Jun Pang. Fast leader election in anonymous rings with bounded expected delay. *Information Processing Letters*, 111(17):864–870, 2011.
16. Yehuda Afek and Yossi Matias. Elections in anonymous networks. *Information and Computation*, 113(2):312–330, 1994.
17. Murali Krishna Ramanathan, Ronaldo A Ferreira, Suresh Jagannathan, Ananth Grama, and Wojciech Szpankowski. Randomized leader election. *Distributed Computing*, 19(5-6):403–418, 2007.
18. Yehuda Afek, Noga Alon, Omer Barad, Eran Hornstein, Naama Barkai, and Ziv Bar-Joseph. A biological solution to a fundamental distributed computing problem. *Science*, 331(6014):183–185, 2011.
19. Omer Barad, Eran Hornstein, and Naama Barkai. Robust selection of sensory organ precursors by the Notch-Delta pathway. *Current Opinion in Cell Biology*, 23(6):663–667, 2011.
20. Omer Barad, Dalia Rosin, Eran Hornstein, and Naama Barkai. Error minimization in lateral inhibition circuits. *Science Signaling*, 3(129):ra51, 2010.
21. Thomas L. Jacobsen, Keith Brennan, A. Martinez Arias, and M. A. Muskavitch. Cis-interactions between Delta and Notch modulate neurogenic signalling in Drosophila. *Development*, 125(22):4531–4540, 1998.
22. Miho Matsuda and Ajay B. Chitnis. Interaction with Notch determines endocytosis of specific Delta ligands in zebrafish neural tissue. *Development*, 136(2):197–206, 2009.
23. David Sprinzak, Amit Lakhmanpal, Lauren LeBon, Leah A. Santat, Michelle E. Fontes, Graham A. Anderson, Jordi Garcia-Ojalvo, and Michael B. Elowitz. Cis-interactions between Notch and Delta generate mutually exclusive signalling states. *Nature*, 465(7294):86–90, 2010.
24. Yehuda Afek, Noga Alon, Ziv Bar-Joseph, Alejandro Cornejo, Bernhard Haeupler, and Fabian Kuhn. Beeping a maximal independent set. *Distributed Computing*, 26(4):195–208, 2013.
25. Lei Xu and Peter Jeavons. Simple neural-like P systems for maximal independent set selection. *Neural Computation*, 25(6):1642–1659, 2013.
26. Alex Scott, Peter Jeavons, and Lei Xu. Feedback from nature: an optimal distributed algorithm for maximal independent set selection. In *ACM Symposium on Principles of Distributed Computing, PODC '13, Montreal, QC, Canada, July 22-24, 2013*, pages 147–156, 2013.
27. Dana Angluin. Local and global properties in networks of processors (extended abstract). In *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, STOC '80, pages 82–93, New York, NY, USA, 1980. ACM.
28. Nathan Linial. Legal coloring of graphs. *Combinatorica*, 6(1):49–54, 1986.
29. Sarah J. Bray. Notch signalling: a simple pathway becomes complex. *Nature Reviews Molecular Cell Biology*, 7(9):678–689, 2006.

30. Joanne R. Collier, Nicholas A.M. Monk, Philip K. Maini, and Julian H. Lewis. Pattern formation by lateral inhibition with feedback: a mathematical model of delta-notch intercellular signalling. *Journal of Theoretical Biology*, 183(4):429–446, 1996.
31. Christoph Lenzen and Roger Wattenhofer. Distributed algorithms for sensor networks. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 370(1958):11–26, January 2012.
32. Charles Miller Grinstead and James Laurie Snell. *Introduction to Probability*. American Mathematical Society, 1998.
33. Saket Navlakha and Ziv Bar-Joseph. Algorithms in nature: the convergence of systems biology and computational thinking. *Molecular Systems Biology*, 7(1), 2011.
34. Gexiang Zhang, Haina Rong, Ferrante Neri, and Mario J. Pérez-Jiménez. An optimization spiking neural P system for approximately solving combinatorial optimization problems. *International Journal of Neural Systems*, 24(05):1–16, 2014.