Gathering and Election by Mobile Robots in a Continuous Cycle

Paola Flocchini
School of Electrical Eng. and Comp. Sci., University of Ottawa, Ottawa, ON, K1N 6N5, Canada.

Ryan Killick
School of Computer Science, Carleton University, Ottawa, ON, K1S 5B6, Canada.

Evangelos Kranakis
School of Computer Science, Carleton University, Ottawa, ON, K1S 5B6, Canada.

Nicola Santoro
School of Computer Science, Carleton University, Ottawa, ON, K1S 5B6, Canada.

Masafumi Yamashita
Dept. of Comp. Sci. and Comm. Eng., Kyushu University, Motooka, Fukuoka, 819-0395, Japan.

Abstract
Consider a set of $n$ mobile computational entities, called robots, located and operating on a continuous cycle $C$ (e.g., the perimeter of a closed region of $\mathbb{R}^2$) of arbitrary length $\ell$. The robots are identical, can only see their current location, have no location awareness, and cannot communicate at a distance. In this weak setting, we study the classical problems of gathering ($\textsc{Gather}$), requiring all robots to meet at a same location; and election ($\textsc{Elect}$), requiring all robots to agree on a single one as the “leader”. We investigate how to solve the problems depending on the amount of knowledge (exact, upper bound, none) the robots have about their number $n$ and about the length of the cycle $\ell$. Cost of the algorithms is analyzed with respect to time and number of random bits. We establish a variety of new results specific to the continuous cycle – a geometric domain never explored before for $\textsc{Gather}$ and $\textsc{Elect}$ in a mobile robot setting; compare Monte Carlo and Las Vegas algorithms; and obtain several optimal bounds.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Cycle, Election, Gathering, Las Vegas, Monte Carlo, Randomized Algorithm

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.11

Acknowledgements Paola Flocchini: University Research Chair
Paola Flocchini, Evangelos Kranakis, Nicola Santoro: Research supported in part by NSERC Discovery grant.
Ryan Killick: Research supported by the NSERC Canada Graduate Scholarship

1 Introduction

1.1 The Framework

Consider a distributed system composed of a set $R$ of autonomous mobile computational entities, called robots, located and operating in an Euclidean space $U$. The robots are identical: without identifiers or distinguishing features, they have the same capabilities and execute the same algorithm. Although autonomous, their goal is to collectively perform some assigned system task or to solve a given problem. Among the important tasks and problems are: gathering ($\textsc{Gather}$), requiring all robots to meet at a same location; and election ($\textsc{Elect}$), requiring all robots to agree on a single one as the “leader”. Indeed, $\textsc{Gather}$ is one of the fundamental problems in theoretical mobile robotics, while $\textsc{Elect}$ is typically solved as an intermediate step in the resolution of many important problems, in particular
pattern formations. Both GATHER and ELECT have been extensively investigated under a variety of assumptions on the capabilities of the robots (e.g., memory, communication, visibility, orientation, speed), on the space in which they operate, and on the power of the adversary. From the point of view of the behaviour of the robots, the two main models are Look-Compute-Move (LCM) and Continuous Time (CT). In LCM the robots operate by cycling through three separate processes: observing the space (Look), executing the algorithm to determine a destination (Compute), and moving towards it (Move). In CT the robots are permanently active and continuously performing all three processes. For a recent overview see [15] and the chapters therein.

In all investigations, in both models, the theoretical concern is to identify the weakest possible conditions that make the problems solvable.

In this paper, we consider GATHER and ELECT by identical robots when the space $\mathcal{U}$ is a continuous cycle $\mathcal{C}$ (e.g., the perimeter of a closed region of $\mathbb{R}^2$). This spatial setting has been investigated in the LCM model with respect to the scattering problem, requiring identical robots to place themselves at uniform distance along the cycle [13]. In the CT model, a continuous cycle has been studied in the context of solving patrolling when the robots are identical [9] and when they have different motorial capabilities [7]; gathering has also been investigated, but only with robots having different motorial capabilities [22].

We study GATHER and ELECT in the CT model in a very weak computational setting: the identical robots can only see their current location and have no location awareness; furthermore they cannot communicate at a distance (i.e., communication is possible only between robots located at the same point at the same time).

It is immediate to observe that, in our setting, both problems are deterministically unsolvable: there is no deterministic algorithm that, in all possible executions of the algorithm by the robots and regardless of the initial position of the robots in the cycle, will always correctly solve the problem within finite time. This is obvious in the case of ELECT because, to render a single robot uniquely different from all others it requires the existence of some asymmetry in the system (e.g., in the initial placement of the robots, in shape of the Euclidean space) if no difference is present among the robots (e.g., distinct ids, different speeds). In our setting the impossibility holds also for GATHER, which does not have such a stringent requirement, and can sometimes be deterministically solved in absence of asymmetries and differences among the robots (e.g. [5]). Further observe that, since visibility is limited to the current robot’s location, in our setting both problems are deterministically unsolvable even if the initial configuration is asymmetric, and the robots are aware of this fact. Summarizing, the only possible solution algorithms are randomized ones.

### 1.2 Main Contributions

In this paper we start the investigation of solving GATHER and ELECT by the set of robots $\mathcal{R}$ deployed in a continuous cycle $\mathcal{C}$. Since GATHER is of easy resolution once a leader has been elected, we primarily focus on ELECT.

We propose both Las Vegas and Monte Carlo decentralized election protocols where: a Las Vegas algorithm correctly terminates with probability one in an unpredictable amount of time; a Monte Carlo algorithm has a fixed termination time but pays for this determinism with a positive – yet bounded – probability that it has terminated incorrectly. In other words, a Las Vegas algorithm “gambles with resources” and a Monte Carlo algorithm “gambles with correctness”.

We evaluate the complexity of the proposed algorithms with respect to two cost measures: the time until the algorithm terminates, and the total number of random bits (coin flips)
required by the algorithm. The costs depend not only on the length \( \ell \) of the cycle and the
number \( n \) of mobile robots (note that \( n \) can be arbitrarily larger than \( \ell \)), but also and more
importantly on the knowledge (none, exact, upper bound) the robots have on \( \ell \) and/or \( n \).

We establish several results. In particular, we prove that, with knowledge of \( \ell \), a leader
can be elected with probability one in optimal time with an optimal number of random bits,
even without any knowledge of (an upper bound on) \( n \). If only an upper bound \( L = O(\ell) \) is
known, then a leader can be elected with high probability in optimal time with an optimal
number of random bits, even without any knowledge of (an upper bound on) \( n \).

The results of the paper are summarized in Tables 1 and 2. As we are analyzing
randomized algorithms, the cost measures are often random variables; when this is the case,
we give both the value achieved in the average and that with high probability.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( \ell )</th>
<th>( T_{exp} )</th>
<th>( B_{exp} )</th>
<th>Type</th>
<th>Algo.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ex.</td>
<td>UB</td>
<td>( O(L) )</td>
<td>( O(n) )</td>
<td>LV</td>
<td>A1</td>
</tr>
<tr>
<td>Ex.</td>
<td>-</td>
<td>( O(n+\ell) )</td>
<td>( O(n+n \log \frac{\ell}{n}) )</td>
<td>LV</td>
<td>A1 + A7</td>
</tr>
<tr>
<td>- Ex.</td>
<td>-</td>
<td>( O(\ell) )</td>
<td>( O(n) )</td>
<td>LV</td>
<td>A1 + A6</td>
</tr>
<tr>
<td>UB</td>
<td>UB</td>
<td>( O(L) )</td>
<td>( O(n) )</td>
<td>MC</td>
<td>A3</td>
</tr>
<tr>
<td>UB</td>
<td>-</td>
<td>( O(N+N \cdot \ell/n) )</td>
<td>( O(n+n \log \frac{\ell}{n}) )</td>
<td>MC</td>
<td>A3 + A7</td>
</tr>
<tr>
<td>- UB</td>
<td>-</td>
<td>( O(L) )</td>
<td>( O(n) )</td>
<td>MC</td>
<td>A3 + A8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( n )</th>
<th>( \ell )</th>
<th>( T_{whp} )</th>
<th>( B_{whp} )</th>
<th>Type</th>
<th>Algo.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ex.</td>
<td>UB</td>
<td>( O(L \log n) )</td>
<td>( O(n \log n) )</td>
<td>LV</td>
<td>A1</td>
</tr>
<tr>
<td>Ex.</td>
<td>-</td>
<td>( O(n + \ell \log n) )</td>
<td>( O(n \log n + n \log \frac{\ell}{n}) )</td>
<td>LV</td>
<td>A1 + A7</td>
</tr>
<tr>
<td>- Ex.</td>
<td>-</td>
<td>( O(\ell \log n) )</td>
<td>( O(n \log n) )</td>
<td>LV</td>
<td>A1 + A6</td>
</tr>
<tr>
<td>UB</td>
<td>UB</td>
<td>( O(L \log N) )</td>
<td>( O(n \log N) )</td>
<td>MC</td>
<td>A3</td>
</tr>
<tr>
<td>UB</td>
<td>-</td>
<td>( O(N+N \cdot \ell/n \cdot \log N) )</td>
<td>( O(n \log n + n \log \frac{\ell}{n}) )</td>
<td>MC</td>
<td>A3 + A7</td>
</tr>
<tr>
<td>- UB</td>
<td>-</td>
<td>( O(L \log N) )</td>
<td>( O(n \log N) )</td>
<td>MC</td>
<td>A3 + A8</td>
</tr>
</tbody>
</table>

The paper is organized as follows. We first consider the case when the robots have some
level of knowledge (exact or upper bound) of both parameters (Section 3). We prove that,
when the robots possess knowledge of \( n \), the knowledge of an upper bound \( L = O(\ell) \) allows
for a LV solution which is optimal with respect to both complexity measures. In case the
robots know only upper bounds on both \( n \) and \( \ell \), we give a Monte Carlo algorithm. In
Section 4 we consider the cases when the robots have no knowledge (exact nor upper bound)
of one of the two parameters. In these cases we provide Las Vegas algorithms by which the
robots can obtain knowledge of the unknown parameter efficiently, and subsequently elect a
leader using the algorithms of Section 3. In Section 5 we demonstrate that unless the robots
know \( n \) and/or \( \ell \) exactly, a Las Vegas algorithm cannot exist that solves ELECT. Extensions,
including the solutions for GATHER using the results for ELECT, and open questions are
discussed in Section 6.
1.3 Related work

There exists an extensive literature on problem solving by \( n \) identical mobile robots in continuous spaces, both within the distributed computing and the control communities; e.g., see the books [4, 14, 15]. In distributed computing, the problem of gathering identical robots has been the focus of intensive investigations under a variety of assumptions on the computational power and communication capabilities of the robots (e.g., [5, 6, 16, 27]). Similarly, the problem of electing a leader and its relationship to asymmetry has been observed, investigated and discussed when studying solvability of a variety of problems by autonomous mobile robots, in particular pattern formations (e.g., [10, 17, 19]). Indeed, a great deal of research has been devoted to the link between degree of symmetries and deterministic problem solving; see [15] and chapters therein for a recent account, in particular [30]. Almost all of this work is on deterministic solutions, with few exceptions (e.g., [20]).

Robots operating specifically in a continuous cycle have been studied in the context of rendezvous and gathering, but only with robots having different motorial capabilities [11, 22]. Other investigated problems in a continuous cycle are: patrolling, studied both when the robots are identical and when they have different motorial capabilities (e.g. see [7, 8, 9]); and scattering, where the robots must place themselves at uniform distance on the cycle [13].

The geometric continuous settings in which the mobile entities can move freely are in general more suitable than discrete settings for distributed computing applications in robotics [4]. This is further enforced by the fact that after a system shut-down in a robot application the participating robots cannot be guaranteed to occupy the vertices of a graph but rather might be placed at arbitrary locations in the underlying geometric domain.

Settings of identical mobile entities operating in discrete spaces (i.e., in graphs) are extremely important as they naturally describe a wide variety of computational environments, including networked systems supporting mobile software agents, and ad-hoc wireless networks. In these settings, the analogue of a set of mobile robots in a continuous cycle is a set of identical mobile agents in a ring of identical nodes. Interestingly, this discrete setting has been extensively studied, especially for rendezvous and gathering; e.g., see the monograph [26]. In absence of distinct features of the agents and of the nodes (e.g., ids, markers, tokens), solutions are necessarily randomized, and their development has been the object of several investigations. In particular Ooshita et al. studied the gathering problem in anonymous unidirectional ring networks for multiple (mobile) agents with limited knowledge and characterized the relation between probabilistic solvability and termination detection [29]. Izumi et al. investigated the feasibility of polynomial-expected-round randomized gathering for \( n \) robots and show that any randomized algorithm has \( \Omega(\text{exp}(n)) \) expected-round lower bound [24].

In the computational universe of static (or stationary) entities connected via a communication network (i.e. the traditional message-passing universe in distributed computing), the computational entities coincide with the network nodes (i.e., the nodes are the active agents). Note that, in this universe, the problem GATHER does not exist; on the other hand, ELECT is a fundamental problem. When the entities are identical, the system is known as an anonymous network, and several researchers have focused on computing in an anonymous ring (e.g., [1, 2, 12]). The problem of electing a leader in an anonymous network, known also as symmetry breaking and for which clearly only probabilistic solutions exist, has been investigated in an anonymous ring network (e.g., [3, 18, 23]). In particular, Itai and Rodeh proposed probabilistic algorithms for both the synchronous and asynchronous case; they considered both cases when the size of the ring may be either known or unknown to the nodes and studied its impact on termination with a nonzero probability [23].

Interestingly, of all the related work, the one closest in spirit to our investigation is that
of symmetry breaking in an anonymous ring, in spite of the fact that the computational
universes are completely different: static entities and discrete space in one while mobile
entities and continuous space in ours.

2 Model

Let $R$ be a set of $n \geq 2$ autonomous mobile computational entities, called robots, located in
a continuous cycle $C$ (e.g., the perimeter of a closed region of $\mathbb{R}^2$) of real length $\ell$ in arbitrary
and pairwise distinct positions.

The robots are identical: without identifiers or distinguishing features, they have the same
(computational, motorial and communication) capabilities and execute the same algorithm.
We assume that all robots move at speed one. Each robot $r \in R$ has a local memory
composed of a finite set of registers, including a special register $\text{state}(r)$ which stores the
current state of $r$; initially, the content of the memory of every robot is the same. Each robot
is in possession of a fair coin which outputs $H$ or $T$ each with probability $1/2$. At any time
a robot may flip its coin and base a decision on the outcome of that flip. For a robot $r$ we
will use the notation $b(r)$ to represent a special register which always contains the outcome
of its most recent coin-flip. We will use the notation $b(r) \leftarrow \text{flip}()$ to represent the action of
flipping a coin and assigning the outcome to $b(r)$.

The robots can only see their current location and have no location awareness. Furthermore
they cannot communicate at a distance; that is, communication is possible only between
robots located at the same point at the same time (face-to-face). A robot may move along
$C$ in either the $cw$ (clockwise) or $ccw$ (counter-clockwise) direction and may stop and/or
reverse its direction of movement at any time. For simplicity, we will assume that the robots
have consistent orientations and argue in Section 6 why this assumption is not necessary.

The robots are permanently active and continuously performing three processes: executing
the algorithm (which might require flipping a coin), moving in a given direction or not at
all (if so prescribed by the algorithm), and communicating with co-located robots. A robot
can distinguish among its co-located robots and is able to instantaneously exchange any
amount of information with each of them. When two robots moving in opposite directions
meet, or a moving robot meets a stopped robot, the two robots become co-located; we call
this an encounter. During an encounter, one of the robots can decide to merge with the
other, thereby committing itself to following all actions of the robot it has merged with. As a
result of this process, robots will form robot stacks with the head of the stack the only robot
actively participating in an algorithm (the stack acts as a single robot). A robot $r$ will keep
track of the number of robots present in its stack in a special register denoted by $\text{cnr}(r)$.

We assume a fully synchronous system in the following sense. Each robot possesses an
identical copy of the same clock and each robot can use their respective clocks to measure
arbitrarily small intervals with respect to the same unit of time (which we may take to be 1
without loss of generality). All robots will begin an algorithm at the same moment and all
robots move with the same speed (which we may also take to be 1 without loss of generality).
This implies that robots can fix a unit length as the distance traveled in one unit of time.

We study how such robots can solve $\text{ELECT}$ and $\text{GATHER}$, and at what cost. The election
problem, $\text{ELECT}$, requires the robots to transition from an initial configuration where each
robot is in an identical state, to one where a single robot can be uniquely distinguished
from the others. When solving this problem, we will assume the robots can be found in
one of the three states CANDIDATE, FOLLOWER, or LEADER. The gathering problem,
$\text{GATHER}$, requires the robots to transition from an initial configuration where each robot is
in an identical state, to one where all robots are co-located and will no longer move. Since
GATHER is of easy resolution once a leader has been elected, we primarily focus on ELECT.

We distinguish between two types of randomized algorithms: those of the Las Vegas type
and those of the Monte Carlo type \cite{28, 21}. An algorithm is of the Las Vegas type, if, for
any problem instance, it is correct when it terminates and it terminates with probability 1.
In contrast, an algorithm is of the Monte Carlo type if, for any problem instance, it always
terminates and it is correct with a probability \( p \) which is bounded away from zero.

The costs of a solution algorithm are evaluated with respect to two measures: 1) time
complexity – the time until the algorithm terminates; and 2) random-bit complexity – the
total number of random bits/coin flips used by the algorithm. The costs depend not only on
the system parameters, the length \( \ell \) of the cycle and the number \( n \) of mobile robots, but also
and more importantly on the type of knowledge available to the robots about the values of
those parameters. As we are analyzing randomized algorithms, these complexity measures
will often be random variables. When this is the case, we will give the value achieved in the
average and with high probability.

\section{Election with knowledge of both \( n \) and \( \ell \)}

In this section we consider ELECT when the robots possess knowledge of both \( n \) and \( \ell \) (either
exact or upper bounds). We begin with the case that the robots have exact knowledge.
Pseudocode for all algorithms can be found in the appendix.

\subsection{Exact knowledge of \( n \) and \( \ell \)}

\begin{theorem}
Let \( n \) and \( \ell \) be known to the robots. There is a Las Vegas algorithm solving
ELECT which terminates in time \( O(\ell) \) on average and in time \( O(\ell \log n) \) with high probability;
and requires \( O(n) \) random bits on average and \( O(n \log n) \) with high probability.
\end{theorem}

The proof is based on the algorithm ELECTLV(\( n, \ell \)). This algorithm is formally described
as Algorithm 1 and takes as inputs the number of robots \( n \) and the length of the cycle \( \ell \).
Initially all robots begin in the same CANDIDATE state and each robot \( r \) has \( \text{cnr}(r) \) set
to 1. The algorithm proceeds in a series of rounds beginning with the round \( t = 0 \). In each
round the CANDIDATE robots will run the procedure ELECTIONROUND(\( D \)) with input
\( D_i = \min\{\frac{\ell}{2}, \frac{\ell}{4}, (4/3)^i\} \), the result of which is that a subset of the robots merge and enter the
FOLLOWER state. This will continue on until only a single CANDIDATE robot remains
with a stack containing all \( n \) robots. As the robots know the value of \( n \), this last remaining
robot will know it is the last and will thus enter the LEADER state.

The procedure ELECTIONROUND(\( D \)) is formally described as Algorithm 2. The idea of
this procedure is as follows. Each robot begins by flipping a coin. Those that flip T will
remain stationary for a time \( 4D \). Those that flip H will: move ccw a distance \( D \); return
to their initial positions; move cw a distance \( D \); and again return to their initial positions.
If ever it occurs that a robot \( r \) who flipped H encounters a robot \( s \) who flipped T then \( s \) will
merge with \( r \) and \( r \) will update the value of \( \text{cnr}(r) \) to reflect this.

We begin our analysis by determining how effective the procedure ELECTIONROUND(\( D \))
is at reducing the number of candidates. This will be the subject of the next two lemmas.

\begin{lemma}
Let \( n \) and \( n' \) respectively represent the number of CANDIDATE robots before
and after ELECTIONROUND(\( D \)) is run with input \( D > 0 \). Then \( E[n'] \leq \frac{n}{2} + \frac{1}{2} \left\lceil \frac{n}{2D} \right\rceil \).
\end{lemma}

\begin{proof}
Partition the cycle into \( \left\lceil \frac{\ell}{2D} \right\rceil \) disjoint intervals such that each interval has length
\( \frac{\ell}{m} \leq 2D \). For each \( i \in [1, m] \) let \( n_i \) and \( n'_i \) respectively represent the number of CANDIDATE
robots contained in the \(i^{th}\) interval at the beginning and end of \(\text{ELECTIONROUND}(D)\). Then it is clear that \(n = \sum_{i=1}^{m} n_i\) and \(n' = \sum_{i=1}^{m} n_i'\). This latter expression allows us to write the expectation of \(n'\) as follows:

\[
E[n'] = \sum_{i=1}^{m} E[n'_i] = \sum_{i=1}^{m} \sum_{x=1}^{n_i} x \Pr[n'_i = x].
\]  

(1)

To determine the probability \(\Pr[n'_i = x]\) consider the \(i^{th}\) interval which initially contains \(n_i > 0\) CANDIDATE robots. If at least one of these \(n_i\) robots flipped H then the number of them that will remain CANDIDATE is exactly the number of them that flipped H. Thus, if we let \(k_i\) represent the random variable which counts the number of CANDIDATE robots that flipped H in an interval \(i\) then we can conclude that \(\Pr[n'_i = x|k_i \geq 1] = 1\) if \(x = k_i\) and 0 otherwise. For \(x \in [1, n_i]\) this implies that \(\Pr[n'_i = x] = \sum_{j=0}^{n_i} \Pr[n'_i = x|k_i = j] \Pr[k_i = j]\) or \(\Pr[n'_i = x] = \Pr[k_i = x] + \Pr[n'_i = x|k_i = 0] \Pr[k_i = 0]\). Using this expression for \(\Pr[n'_i = x]\) we find that \(E[n'_i] = \sum_{x=0}^{n_i} x \Pr[n'_i = x] = \sum_{x=0}^{n_i} x \Pr[n'_i = x|k_i = 0] \Pr[k_i = 0].\) The lemma clearly holds if \(D_k \geq \ell/2\). If this is not the case then \(D_k = \frac{\ell}{n} (\frac{4}{3})^k\) and again it is easy to see that the lemma holds.

Lemma 3. Let \(n_i\) count the number of CANDIDATE robots remaining in round \(t \geq 0\) of \(\text{ELECTLV}(n, \ell)\). Then \(E[n_i] \leq \left(\frac{3}{4}\right)^t n\).

Proof. The proof is by induction on \(t\). The base case \(t = 0\) is clearly true. We assume that the claim holds up to \(t = k\). Using the induction hypothesis and Lemma 2 we can write

\[
E[n_{i+1}] \leq \frac{1}{2} \left(\frac{4}{3} k\right)^k n + \frac{1}{2} \left(\frac{4}{3} \right)^k n \sum_{x=0}^{n_i} x \Pr[k_i = 0] \Pr[k_i = 0].
\]

The lemma clearly holds if \(D_k \geq \ell/2\). If this is not the case then \(D_k = \frac{\ell}{n} (\frac{4}{3})^k\) and again it is easy to see that the lemma holds.

In the next three lemmas (Lemma 4, Lemma 5, and Lemma 6) we bound the number of rounds, time, and random-bits required until only a single candidate robot remains. In order to do so we will employ a useful theorem by Karp [25] concerning the solutions of stochastic recurrence relations. This theorem is described in the appendix as Theorem 22.

Lemma 4. Let \(T\) be the first round of \(\text{ELECTLV}(n, \ell)\) in which only a single CANDIDATE robot remains. Then \(E[T] \leq \left[\log_{4/3}(n)\right] + 1\) and, for any positive integer \(w\),

\[
\Pr\left[T \geq \left[\log_{4/3}(n)\right] + 1 + w\right] \leq \left(\frac{4}{3}\right)^w n \left[\log_{4/3}(n)\right].
\]

Proof. Observe that \(T = T(n)\) satisfies the stochastic recurrence relation \(T(n) = 1 + T(h(n))\) with base condition \(T(1) = 0\) and where the expectation of \(h(n)\) is bounded using Lemma 3, i.e., \(E[h(n)] \leq \left[\frac{4}{3} n\right]\). With this observation the lemma follows easily from Theorem 22.

Lemma 5. Let \(\tau\) be the time required until only a single CANDIDATE robot remains in \(\text{ELECTLV}(n, \ell)\). Then \(E[\tau] \leq 8L\) and, for any positive integer \(w\), \(\Pr[T \geq 2L(4 + w)] \leq \left(\frac{3}{4}\right)^w n \left[\log_{4/3}(n)\right].\)
Proof. Set \( t_L \) as the first round which satisfies \( L/n(4/3)^{t} \geq L/2 \), i.e. \( t_L = \left\lceil \log_{4/3}(n/2) \right\rceil \).

Assume that it takes \( T > t_L \) rounds until only one CANDIDATE robot remains. The time \( \tau \) required to complete these \( T \) rounds is \( \tau = 4L \sum_{t=0}^{t-1} (4/3)^{t} + 2 \sum_{t=t_L}^{T} L \leq 12L (4/3)^{t_L} + 2(T - t_L)L \leq 8L + 2(T - t_L) \). The lemma now follows from Lemma 4.

\[ \blacksquare \]

Lemma 6. Let \( B \) be the random variable which counts the number of coin-flips used in \( \text{ELECTLV}(n, \ell) \). Then \( E[B] \leq 4n \) and, for any positive integer \( w \), \( \Pr[T \geq (4+w)n] \leq \left(\frac{3}{4}\right)^w \).

Proof. Similarly to the proof of Lemma 4 we observe \( B = B(n) \) satisfies the stochastic recurrence relation \( B(n) = n + B(h(n)) \) with base condition \( B(1) = 0 \) and where \( h(n) \) has expectation \( E[h(n)] \leq \left\lceil \frac{3}{4}n \right\rceil \). With this observation the lemma follows easily from Theorem 22.

The proof of Theorem 1 now follows immediately from Lemmas 5, and 6.

3.2 Inexact knowledge of \( n \) and/or \( \ell \)

We now consider the cases that the robots are provided with inexact knowledge (upper bounds) of at least one of \( n \) or \( \ell \). We begin with the case that the robots know \( n \) and an upper bound on \( \ell \).

Observe that nowhere in the proof of Theorem 1 did we require the robots to know exactly the value of \( \ell \). In particular, if the robots were to instead use an upper bound \( L \) on \( \ell \) then the only change we need to make is to replace \( \ell \) with \( L \) in the time complexity. This observation thus easily leads to the following corollary of Theorem 1:

Corollary 7. Let \( n \) and an upper bound \( L \geq \ell \) be known to the robots. There is a Las Vegas algorithm solving this problem which terminates in time \( O(L) \) on average and in time \( O(L \log n) \) with high probability; and requires \( O(n) \) random bits on average and \( O(n \log n) \) with high probability.

The same argument does not work if the robots know \( \ell \) and an upper bound \( N \geq n \) since \( \text{ELECTLV} \) requires the exact value of \( n \) in order to terminate. We will see in the next section that exact knowledge of \( \ell \) however allows the robots to determine \( n \) and we will therefore postpone a discussion of this case until then.

If the robots only possess upper bounds on both \( n \) and \( \ell \) then a Las Vegas algorithm does not exist (see Section 5). We thus provide a Monte Carlo algorithm (Algorithm 3) to solve the problem.

Theorem 8. Let upper bounds \( N \geq n \) and \( L \geq \ell \) be known to the robots. Then, for any positive integer \( w \) there is a Monte Carlo algorithm solving \( \text{ELECT} \) with error probability \( O((3/4)^w) \). This algorithm terminates in time \( O(wL) \) and requires \( O(wn) \) random bits.

Proof. The proof is based on the algorithm \( \text{ELECTMC}(N, L, w) \) which takes as inputs the upper bounds \( N \) and \( L \), and a positive integer \( w \) which controls the runtime. This algorithm is formally described as Algorithm 3. This algorithm is identical to \( \text{ELECTLV}(N, L) \) except that it deterministically terminates on the round \( t_{\infty} = \left\lceil \log_{4/3}(N) \right\rceil + w \). We may therefore reuse many of our previously derived results. In particular, the time \( \tau \) until termination follows from the proof of Lemma 5 and is given by \( \tau = 8L + 2(w + 1)L \). The random-bit complexity follows from Lemma 6. The error probability of the algorithm is also easy to derive. In particular, if we let \( T \) be the number of rounds required until only a single CANDIDATE remains then the probability that the algorithm terminates incorrectly is simply...
the probability \( \Pr[T > t_\infty] = \Pr[T > \lceil \log_{4/3}(N) \rceil + w] = \Pr[T \geq \lceil \log_{4/3}(N) \rceil + 1 + w] \)
and this probability is given by Lemma 4.

## 4 Election with knowledge of either \( n \) or \( \ell \)

In this section we investigate ELECT when the robots are provided with knowledge of only
one of \( n \) or \( \ell \) (exact or upper bounds). In all cases we use the same strategy to solve the
problem: we develop algorithms by which the robots gain knowledge of the unknown of \( n \) or
\( \ell \) and then use the algorithms of the previous section to solve ELECT. Pseudocode for all
algorithms presented can be found in the appendix.

### 4.1 Exact knowledge of \( n \) or \( \ell \)

**Theorem 9.** Let either \( n \) or \( \ell \) be known to the robots. Then there are Las Vegas algorithms solving ELECT. If \( \ell \) is known the algorithm terminates in time \( O(\ell) \) on average and in time
\( O(\ell \log n) \) with high probability; and requires \( O(n) \) random bits on average and \( O(n \log n) \)
with high probability. If \( n \) is known the algorithm terminates in time \( O(n + \ell) \) on average
and in time \( O(n + \ell \log n) \) with high probability; and requires \( O(n + n \log \lceil \frac{\ell}{n} \rceil) \) random bits
on average and \( O(n \log(n) + n \log \lceil \frac{\ell}{n} \rceil) \) with high probability.

As previously stated, our proof strategy is to first develop algorithms by which the robots
can gain knowledge of the unknown of \( n \) or \( \ell \). More specifically, the goal of this section is to
constructively demonstrate the validity of the following two lemmas from which Theorem 9
will easily follow.

**Lemma 10.** Consider \( n \) robots on a cycle of length \( \ell \) and assume the robots know only
the value of \( \ell \). Then there exists a Las Vegas algorithm by which the robots can determine
the value of \( n \). This algorithm terminates in time \( O(\ell) \) on average and with high probability;
and requires \( O(n) \) random bits on average and with high probability.

**Lemma 11.** Consider \( n \) robots on a cycle of length \( \ell \) and assume the robots know only
the value of \( n \). Then there exists a Las Vegas algorithm by which the robots can determine
an \( O(\ell) \) upper bound \( L \) on \( \ell \). This algorithm terminates in time \( O(n + \ell) \) on average and
with high probability; and requires \( O(n + n \log \lceil \frac{\ell}{n} \rceil) \) random bits on average and with high
probability.

We will begin by introducing two procedures which will be used throughout the remainder
of the section. The first procedure will be used by the robots to count coin flips, and the
second is a minimum finding procedure.

**A procedure to count coin flips:** The procedure \( \text{COUNTFLIPS}(D) \) is formally described as
Algorithm 4 and takes as input a distance \( D \). For simplicity in the following description
we will assume that \( D = \ell \). The procedure presumes that each robot \( r \) has flipped a coin
and stored the result in \( b(r) \). It will result in each robot either knowing the total number of
robots or that all robots have flipped the same thing.

At the beginning the robots that flip \( H \) will move \( cw \) a distance \( \ell \) around the cycle and
count each robot they encounter which flipped \( T \). The robots that flipped \( T \) will likewise
wait for a time \( \ell \) and count each robot they encounter that flipped \( H \). Since each moving
robot makes a full traversal of the cycle they are guaranteed to see all stationary robots.
Thus, after the first \( \ell \) time units, each robot will determine the number of robots which
flipped opposite to themselves. In the last $\ell$ time units of the algorithm the robots which
initially flipped $H$ (resp. $T$) will move ccw a distance $\ell$ around the cycle (resp. wait for $\ell$
for $H$) time units). In either case, a robot will determine the total number of robots that flipped the
same as themselves from the first robot they encounter which flipped opposite to themselves.
Thus, after $2\ell$ time units each robot will have determined both the total number of robots
which flipped $H$ and the number that flipped $T$ and from this they can compute $n$. If all
robots flipped the same thing then the robots will know this since each will have determined
that $N_H(r) = N_T(r) = 0$. From this description it is easy to establish the following lemma:

**Lemma 12.** Assume that all robots have flipped a coin. Then in exactly $2\ell$ time units the
procedure $\text{CountFlips}(\ell)$ will result in either each robot knowing $n$ or that all robots have
flipped the same thing.

When an input $D > \ell$ is used in the procedure we claim the following:

**Lemma 13.** Assume that all robots have flipped a coin and that $D \geq \ell$. Then in exactly
$2D$ time units the procedure $\text{COUNTFlips}(D)$ will result in either each robot $r$ computing an
upper bound $N(r) \geq n$ or that all robots have flipped the same thing.

**Proof.** Clearly, if all robots flip the same then each robot will compute $N_H(r) + N_T(r) = 0$.
Thus, assume that at least two robots flip differently. Let $n_T$ and $n_H$ represent the actual
number of robots that flipped $T$ and $H$ respectively, i.e. $n_T + n_H = n$. Since each robot
that flipped $H$ traverses the cycle at least once each such robot is guaranteed to encounter
all robots that flipped $T$. Likewise, each robot that flipped $T$ is guaranteed to encounter
each robot that flipped $H$. It is therefore not possible for a robot $r$ to compute a value of
$N_H(r) < n_H$ or $N_T(r) < n_T$ and thus it is ensured that $N_T(r) + N_H(r) \geq n$ for all robots. $\blacksquare$

Finally, if an input $D < \ell$ is used in the procedure then we claim the following:

**Lemma 14.** Assume that all robots have flipped a coin and that $D < \ell$. Then in exactly $2D$
time units the procedure $\text{COUNTFlips}(D)$ will result in each robot $r$ computing a lower-bound
$N(r) \leq n$.

**Proof.** The only thing we need to demonstrate is that all robots will compute a value
$N(r) \leq n$. Clearly, in order for this not to be true, at least one of the robots must double
count another robot. This, however, is not possible unless a robot traverses the cycle more
than once and this will clearly not be the case if $D < \ell$. $\blacksquare$

**A minimum finding procedure:** The minimum finding procedure $\text{FindMin}(L, N_0)$ is for-
mainly described as Algorithm 5 and takes as input an upper bound $L \geq \ell$ on the cycle
length, and a value $N_0$ (which is specific to each robot). The algorithm results in each robot
computing the minimum of the inputs $N_0$. It assumes that all robots have flipped a coin and
that at least two robots have flipped differently.

Each robot that flipped $H$ will initially move ccw a distance $L \geq \ell$ around the cycle and is
guaranteed to encounter every robot that flipped $T$. Likewise every robot that flipped $T$ will
encounter every robot that flipped $H$. Thus, after the first $L$ time units, every robot that
flipped $H$ (resp. $T$) will know the minimum value of every robot that flipped $T$ (resp. $H$). In
the second $L$ time units the robots that flipped $H$ will move ccw a distance $L$ and will again
encounter every robot that had flipped $T$. They can thus determine the minimum value of
all robots that flipped $H$ from the first robot they encounter that flipped $T$. Likewise, each
robot that flipped $T$ will determine the minimum value of all robots that flipped $T$ from the
first robot they encounter that flipped $H$. The algorithm clearly terminates after $2L$ time
units. We can thus claim the following without proof:
Lemma 15. Assume that all robots have flipped a coin, at least two have flipped differently, and that $L \geq \ell$. Then in exactly $2L$ times units the procedure $\text{FindMin}(L, N_0(r))$ will result in each robot $r$ computing the minimum of all inputs $N_0(r)$.

Computing $n$ using $\ell$: We will now tackle the proof of Lemma 10 which is based off of the algorithm $\text{CountRobots}(\ell)$. This algorithm is formally described as Algorithm 6 and takes as input the length of the cycle. The idea is to repeatedly flip coins and run the procedure $\text{CountFlips}(\ell)$ until the first round in which at least two robots flip differently. When this occurs each robot will compute the total number of robots that flipped T and the total number that flipped H and will thus determine $n$ to be the sum of these values.

Proof. (Lemma 10) The correctness of $\text{CountRobots}(\ell)$ is obvious. The algorithm will terminate on the first round during which at least two robots flip differently. The probability that all robots flip the same is $2^{1-n}$ and therefore the algorithm terminates after an expected probability that the algorithm terminates after $T$ rounds is $2^{(T-1)(1-n)}(1-2^{1-n})$. From this it is clear that the algorithm terminates after $O(1)$ rounds with high probability. The time and random-bit complexities follow from the fact that each round lasts time at most $2\ell$ and in each round all $n$ robots flip their coins.

Computing a $O(\ell)$ upper bound on $\ell$ using $n$: The proof of Lemma 11 is based off of the algorithm $\text{BoundCycle}(n)$. This algorithm is formally described as Algorithm 7 and takes as input the number of robots on the cycle. In each round $t \geq 0$ the robots will employ the procedure $\text{CountFlips}$ in an attempt to determine a strict upper bound on the number of robots using an estimate $L_t = n \cdot 2^t$ for an upper bound on $\ell$. This will result in each robot $r$ computing a value $N_0(r)$. If $L_t < \ell$ then, by Lemma 14, the robots will each compute $N(r) \leq n$ and the algorithm will proceed to the next round. If $L_t \geq \ell$ then the robots will each compute $N(r) \geq n$ and, after performing $\text{FindMin}$, they will all agree on the computed value of $N(r)$. Let $t_\downarrow$ be the first round in which all robots compute $N(r) > n$. The corresponding value of $L_t$ in the round $t_\downarrow$ will then be an upper bound on $\ell$. We reduce $L_t$, by a factor $\frac{1}{2}$ $\left\lceil \frac{N(r)}{n} \right\rceil$ to ensure that the returned upper bound is $O(\ell)$.

Proof. (Lemma 11) To determine the running time we let $t_0$ be the first round for which $L_t > 2\ell$. Then $t_0 = \left\lceil \log \frac{2\ell}{n} \right\rceil$ if $n < 2\ell$ and $t_0 = 0$ if $n \geq 2\ell$. The algorithm will certainly terminate in the first round $t_\downarrow > t_0$ in which at least two robots flip differently. Since the probability that all robots flip the same is $2^{1-n}$ we will have $t_\downarrow = t_0 + O(1)$ with high probability. The algorithm will therefore take at most $\left\lceil \log \frac{2\ell}{n} \right\rceil + O(1)$ rounds. Since the procedures $\text{CountFlips}(L_t)$ and $\text{FindMin}(L_t)$ each take time $2L_t$ to complete, each round of the algorithm lasts time $4L_t = n \cdot 2^{t_\downarrow+2}$. The total time required is thus $\sum_{t=0}^{t_\downarrow} n \cdot 2^{t+2} = 4n(2^{t_\downarrow+1} - 1)$. If $n > 2\ell$ then the above is clearly $O(n)$. If $n \leq 2\ell$ then we have that $4n(2^{t_\downarrow+1} - 1) = 4n \left(2^{\left\lceil \log \frac{2\ell}{n} \right\rceil + O(1)} - 1 \right) = O(\ell)$.

Thus, we can conclude that the algorithm terminates in time $O(n + \ell)$ on average and with high probability. In each round of the algorithm all robots flip a coin and thus the algorithm requires $O(n)$ random bits if $n > 2\ell$ and otherwise $O(n \log \left\lceil \frac{2\ell}{n} \right\rceil)$ when $n \leq 2\ell$.

4.2 Inexact knowledge of $n$ or $\ell$

We now consider the cases that the robots are only provided with an upper bound on $n$ or only an upper bound on $\ell$. The main result follows:
11:12 Gathering and Election by Mobile Robots in a Continuous Cycle

Theorem 16. Let only an upper bound $L \geq \ell$ or an upper bound $N \geq n$ be known to the robots. Then, for any positive integer $w$ there are Monte Carlo algorithms solving ELECT with error probability $O((3/4)^w)$. If the robots know $L \geq \ell$ then the algorithm terminates in time $O(wL)$ and requires $O(wn)$ random bits. If the robots know $N \geq n$ then the algorithm terminates in time $O(N + w \frac{n}{L} \ell)$ and requires $O(wn + n \log \left\lceil \frac{\ell}{n} \right\rceil)$ random bits.

Our goal is again to develop algorithms by which the robots will gain knowledge of the unknown of $n$ or $\ell$ and then employ the algorithm ELECTMC to solve ELECT. We therefore want to demonstrate the following two lemmas:

Lemma 17. Consider $n$ robots on a cycle of length $\ell$ and assume the robots know an upper bound $L \geq \ell$. Then there exists a Las Vegas algorithm by which the robots can determine an upper bound $N = O(\frac{\ell}{n}n)$ on $n$. This algorithm terminates in time $O(L)$ on average and with high probability; and requires $O(n)$ random bits on average and with high probability.

Lemma 18. Consider $n$ robots on a cycle of length $\ell$ and assume the robots know only an upper bound on the value of $n$. Then there exists a Las Vegas algorithm by which the robots can determine an $O(\frac{N}{\ell}n)$ upper bound $L$ on $\ell$. This algorithm terminates in time $O(N + \frac{n}{L} \ell)$ on average and with high probability; and requires $O(n + n \log \left\lceil \frac{\ell}{n} \right\rceil)$ random bits on average and with high probability.

Clearly Theorem 16 will directly follow from the above two lemmas as well as Theorem 8.

We begin with the case that the robots know $L \geq \ell$.

Computing an upper bound on $n$ from an upper bound on $\ell$: Here we will use an algorithm essentially identical to COUNTROBOTS($\ell$) except with the addition of a FINDMIN procedure. The robots will repeatedly flip coins and run the procedure COUNTFLIPS($L$) until at least two robots flip differently. At this point each robot $r$ will know an upper bound $N(r) \geq n$. They will then run the procedure FINDMIN($L, N(r)$) in order to determine the same upper bound. The correctness of the algorithm follows easily from Lemmas 13 and 15. The fact that the robots compute a $O(\frac{\ell}{n}n)$ upper bound follows from the fact that the robots will traverse the cycle $\frac{\ell}{n}$ times. The asymptotic running time of the algorithm is identical to that of COUNTROBOTS with $\ell$ replaced with $L$. The random-bit complexity does not change. Lemma 17 follows without proof from this discussion.

Computing an upper bound on $\ell$ from an upper bound on $n$: Here we simply use the algorithm BOUNDcycle with the input $N \geq n$ instead of $n$.

Proof. The proof is nearly identical to that of Lemma 11 except we replace $n$ with $N$ and require at least $t_0$ rounds where $t_0$ is the first round in which $L_\ell = N \cdot 2^\ell \geq 2 \left\lceil \frac{N}{\ell} \right\rceil \ell$, i.e. $t_0 = \left\lceil \log \left( \left\lceil \frac{N}{\ell} \right\rceil \right) \right\rceil = O(\log \left\lceil \frac{\ell}{n} \right\rceil)$.

5 Impossibility results

In the previous sections we have developed Las Vegas algorithms which solve ELECT when one of $n$ or $\ell$ is known exactly to the robots. We have also developed Monte Carlo algorithms when only upper-bounds on $n$ and/or $\ell$ are known. In the sequel we demonstrate that, unless the robots know at least one of $n$ or $\ell$ exactly, there does not exist a Las Vegas algorithm which solves ELECT.
\begin{proof}
To derive a contradiction suppose that there is a Las Vegas type algorithm $A$ which solves the problem. Consider an instance $I$ in which there are two robots $r_1$ and $r_2$ at antipodal positions on a cycle with circumference $\ell$. Since $A$ solves the problem it terminates with probability 1 in a finite, though unpredictable, amount of time $T$. Let $O_1$ and $O_2$ be the sequence of outcomes of coin flips of $r_1$ and $r_2$.

Consider another instance $I'$ in which there are four robots $r_1, r_2, r'_1,$ and $r'_2$ at equally spaced locations of a cycle with circumference $2\ell$ such that $r_1$ and $r'_1$ (resp. $r_2$ and $r'_2$) are antipodal (see Figure 1). Assume that the pair $r_1$ and $r'_1$ (resp. $r_2$ and $r'_2$) each have the same orientation and each receives the outcome of coin flips $O_1$ (resp. $O_2$). Call an encounter between a pair of robots $r_1$ and $r_2$ a left encounter (resp. a right encounter) if $r_1$ and $r_2$ encounter each other while either $r_1$ is moving ccw and $r_2$ is stationary, $r_2$ is moving cw and $r_1$ is stationary, or $r_1$ is moving ccw and $r_2$ is moving cw (resp. while either $r_1$ is moving ccw and $r_2$ is stationary, $r_2$ is moving ccw and $r_1$ is stationary, or $r_1$ is moving cw and $r_2$ is moving ccw). Then for every left encounter of $r_1$ and $r_2$ in $I$ there is a corresponding identical left encounter between $r_1$ and $r_2$ in $I'$ and between $r'_1$ and $r'_2$ in $I'$. Likewise, for every right encounter of $r_1$ and $r_2$ in $I$ there are corresponding identical right encounters between $r_1$ and $r'_2$ in $I$ and between $r_2$ and $r'_1$ in $I'$. Thus, at time $T$, each of $r_1$ and $r'_1$ (resp. $r_2$ and $r'_2$) in $I'$ must come to the same conclusion as $r_1$ (resp. $r_2$) in $I$. However, this implies that at the end of the execution of $A$ in $I'$ we will have elected two leaders. Since there is a positive probability that $r_1$ and $r'_1$ (resp. $r_2$ and $r'_2$) both get the outcome of coin flips $O_1$ (resp. $O_2$) then there is a positive probability that $A$ incorrectly terminates in time $T$. This contradicts our assumption that $A$ correctly terminates with probability one. \end{proof}

It is not hard to extend this to the situation that the robots know only an upper bound on $n$:

\begin{corollary}
Suppose that the robots only know an upper bound $N$ on $n$. Then there is no Las Vegas type algorithm which solves ELECT.
\end{corollary}

\begin{proof}
To derive a contradiction suppose that there is a Las Vegas type algorithm $A$ for ELECT. We use the instances $I$ and $I'$ given in the proof of Theorem 19. Provided that

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure1.png}
\caption{Left: The instance $I$ with two robots $r_1$ and $r_2$ on a cycle of length $\ell$. Right: The instance $I'$ with four robots $r_1, r_2, r'_1,$ and $r'_2$ on a cycle of length $2\ell$.}
\end{figure}
$N = 5$ is given, consider the execution of $A$ for $I$. Then in time $T$, $A$ terminates in which $O_1$ and $O_2$ are the sequences of outcomes of the coin flips of $r_1$ and $r_2$.

Then $A$ terminates incorrectly in time $T$, when it is executed for $I'$ with $N = 5$, as argued in the proof of Lemma 20, which is a contradiction. 

Proof. (Theorem 19) Assume that a Las Vegas algorithm $A$ exists by which the robots can solve ELECT if they know upper bounds $N$ and $L$ on $n$ and $\ell$ respectively. Now consider an instance of the problem when only an upper bound $N$ on $n$ is known. Then by Lemma 18 there exists a Las Vegas algorithm by which the robots can determine $L$. Once the robots know $L$, they run algorithm $A$ to elect a leader. This implies that there exists a Las Vegas algorithm by which the robots can elect a leader when they only know an upper bound $N$ on $n$. This contradicts the previous result of Corollary 21 which states that such an algorithm cannot exist. We may therefore conclude that a Las Vegas algorithm does not exist if the robots know both upper bounds $N$ and $L$. This further implies that a Las Vegas algorithm does not exist when the robots know only $L$. ▶

6 Extensions and Open Questions

Here we discuss why the consistent orientation assumption is unnecessary; the extension of our election algorithms to the GATHER problem; and other extensions/open problems.

Orientation: In the previous sections we have assumed that the robots have consistent orientations. Here we will argue why this assumption is not required.

First, observe that with the consistent orientation assumption it will never occur that two moving robots encounter each other. By removing this assumption we will have to deal with the extra encounters involving two robots which move in opposite directions. For most of these encounters the solution is simple — the two moving robots will simply ignore each other. A more problematic encounter occurs if two moving robots encounter a stationary robot from opposite directions at the same time. Fortunately, this is also easily remedied — we simply have the stationary robot choose to “process” the moving robot arriving from its, say, cw direction first. We can thus conclude that all of our results still hold if we remove the consistent orientation assumption.

Gathering: In the previous sections our primary goal has been on how to solve ELECT. However, it is easy to see that our algorithms also solve GATHER at no extra cost. Indeed, consider Algorithm 1 where, during the election process, robots only enter a FOLLOWER state when they merge with a remaining CANDIDATE robot. When only a single CANDIDATE remains all other robots will be part of its stack. This is also the case for Algorithm 3, however, since this is a Monte Carlo algorithm, there is a bounded probability that more than one stack remains when the algorithm terminates. Thus, by construction, Algorithm 1 is a Las Vegas algorithm which solves GATHER and Algorithm 3 is a Monte Carlo algorithm which solves GATHER. Clearly, the complexities of these algorithms remain the same when applied to either the ELECT or GATHER problems.

6.1 Discussions and Open Problems

In this paper we have studied the ELECT and GATHER problems for $n$ identical robots in the $\mathcal{CT}$ model on a continuous cycle of length $\ell$. We have established several results including
optimal algorithms with respect to time and random bits when the robots know $\ell$, or an upper bound $L = O(\ell)$ (in the latter case with high probability).

There are a number of open questions remaining. Firstly, we have not considered the possibility (or lack thereof) of a Monte Carlo algorithm when the robots do not possess any knowledge of $n$ or $\ell$. In addition, we have only considered a fully synchronous time model and a natural extension is therefore to study ELECT and GATHER when this assumption is removed. In particular one can consider a model where the robots do not begin an algorithm simultaneously but otherwise their respective clocks tick at the same rate, or a model where even the robots’ clocks are not synchronized.

References

11:16 Gathering and Election by Mobile Robots in a Continuous Cycle


A Description of Karp’s theorem

Consider the stochastic recurrence relation

\[ T(n) = a(n) + T(h(n)) \]  

which describes a process in which we start with an input of size \( n \) and after investing some amount of resources (represented by \( a(n) \)) we are left with a smaller problem of size \( h(n) \) upon which we recurse. As it applies here, \( n \) represents the number of candidate robots, \( a(n) \) will represent the number of rounds/time/random-bits, and \( h(n) \) the expected number of robots remaining after one iteration of a leader election algorithm.

Formally, \( n \) is a nonnegative integer variable; \( a(n) \) a nonnegative real-valued function of \( n \); \( h(n) \) a random variable with support \([0, n]\) and expectation bounded by \( m(n) \); and \( m(n) \) a nonnegative real-valued function of \( n \). The equation \( \tau(n) = a(n) + \tau(m(n)) \) is the deterministic analogue of (2) and, when it exists, has the unique least nonnegative solution \( u(n) \) given by

\[ u(n) = \sum_{k=0}^{\infty} a(m[k](n)) \]  

with \( m[k](n) \) inductively defined by \( m[0](n) = n \) and \( m[k](n) = m(m[k-1](n)), k \geq 1 \). Karp proved the following:

Theorem 22. (Karp [25], Theorems 1.1 and 1.2). Consider the stochastic recurrence (2), a continuous function \( m(n) \) with \( m(n)/n \) non-decreasing, and let \( u(n) \) be given by (3).

1. Suppose there is a constant \( d \) such that \( a(n) = 0, n < d \); and \( a(n) = 1, n \geq d \). Let \( c_k = \min\{n|u(n) \geq k\} \). Then, for every positive integer \( n \) and every positive integer \( w \),

\[ \Pr[T(n) \geq u(n) + w] \leq \left( \frac{m(n)}{n} \right)^{w-1} \frac{m(n)}{c_{u(n)}}. \]

2. Suppose that \( a(n) \) is strictly increasing on \( \{n|a(n) > 0\} \). Then, for every positive integer \( n \) and every positive integer \( w \), \( \Pr[T(n) > u(n) + wa(n)] \leq \left( \frac{m(n)}{n} \right)^{w} \).

B Pseudocode for algorithms of Section 3.1

Alg. 1: \textsc{ElectLV}(n, \ell)

\textbf{Input:} \( n > 0 \) (integer); \( \ell > 0 \) (real); \( \triangleright \) The number of robots and the length of the cycle.

\textbf{Initialize:} \textit{state}(r) \leftarrow \textsc{Candidate}; cnr(r) \leftarrow 1; t \leftarrow 0;

\textbf{Begin:}
1. \textbf{repeat}
2. \textbf{D} \leftarrow \min \left\{ \frac{\ell}{2}, \ell \left( \frac{4}{7} \right)^t \right\} ;
3. \textbf{ElectionRound}(D); t \leftarrow t + 1; \quad \triangleright \text{Run one election round.}
4. \textbf{if} cnr(r) = n \textbf{then} state(r) \leftarrow \textsc{Leader}; \quad \triangleright \text{Stack contains} \ n \text{ robots, terminate.}
5. \textbf{until} state(r) = \textsc{follower} or \textsc{leader}
\textbf{:End}
Algorithm 2 \textsc{ElectionRound}(D)
\begin{enumerate}
\item \textbf{Input:} $D > 0$ (real);
\item \textbf{Begin:} $b(r) \leftarrow \text{flip}()$;
\item \textbf{if} $b(r) = H$ then \hfill \textgreater{} H was flipped
\item \textbf{Move ccw a distance $D$; cw a distance $2D$; ccw a distance $D$;}
\item \textbf{if} a robot $s$ with $b(s) = T$ is encountered while moving then
\item \textbf{CNR}(r) $\leftarrow$ CNR(r) + CNR(s); \hfill \textgreater{} Update CNR(r) since $s$ will merge with $r$.
\item \textbf{else} \hfill \textgreater{} T was flipped
\item \textbf{Remain stationary for time $4D$;
\item \textbf{if} a robot $s$ with $b(s) = H$ is encountered while waiting then
\item \textbf{state}(r) = FOLLOWER;
\item \textbf{Merge with robot $s$;}
\end{enumerate}
\textbf{:End}

\section{Pseudocode for algorithms of Section 3.2}

Algorithm 3 \textsc{ELECTMC}(N, L, w)
\begin{enumerate}
\item \textbf{Input:} $N > 0$ (integer); $L > 0$ (real); $w \geq 0$ (integer); \hfill \textgreater{} upper bounds on $n$ and $\ell$; termination parameter $w$.
\item \textbf{Initialize:} \textbf{state}(r) $\leftarrow$ CANDIDATE; $t \leftarrow 0$; $t_\infty \leftarrow \lceil \log_{4/3}(n) \rceil + w$; \hfill $t_\infty =$ termination round.
\item \textbf{Begin:}
\item \textbf{repeat}
\item \textbf{D}_t \leftarrow \min\left\{ \frac{\ell}{2}, \frac{L}{N} (\frac{3}{4})^t \right\};
\item \textbf{ElectionRound}(\textbf{D}_t); t \leftarrow t + 1; \hfill \textgreater{} Run one election round.
\item \textbf{until} \textbf{state}(r) = FOLLOWER or $t = t_\infty$
\item \textbf{if} \textbf{state}(r) = CANDIDATE then \textbf{state}(r) $\leftarrow$ LEADER;
\end{enumerate}
\textbf{:End}

\section{Pseudocode for algorithms of Section 4.1}

Algorithm 4 \textsc{CountFlips}(D)
\begin{enumerate}
\item \textbf{Input:} $D > 0$ (real); \hfill \textgreater{} An estimate of the length of the cycle.
\item \textbf{Initialize:} $N_H(r) \leftarrow 0$; $N_T(r) \leftarrow 0$; \hfill \textgreater{} To count the robots flipping H and T.
\item \textbf{Begin:}
\item \textbf{if} $b(r) = H$ then \hfill \textgreater{} H was outcome of last coin flip
\item \textbf{Move cw a distance $D$;
\item \textbf{if} a robot $s$ with $b(s) = T$ is encountered while moving then $N_T(r) \leftarrow N_T(r) + 1$;
\item \textbf{Move ccw a distance $D$;}
\item \textbf{if} $N_H(r) = 0$ and a robot $s$ with $b(s) = T$ is encountered while moving then
\item \textbf{N}_H(r) \leftarrow \textbf{N}_H(s); \hfill \textgreater{} Determine $N_H$.
\item \textbf{else} \hfill \textgreater{} T was outcome of last coin flip
\item \textbf{Wait for time $D$;
\item \textbf{if} a robot $s$ with $b(s) = H$ is encountered while waiting then $N_H(r) \leftarrow N_H(r) + 1$;
\item \textbf{Wait for time $D$;
\item \textbf{if} $N_T(r) = 0$ and a robot $s$ with $b(s) = H$ is encountered while waiting then
\item \textbf{N}_T(r) \leftarrow \textbf{N}_T(s); \hfill \textgreater{} Determine $N_T$.
\item \textbf{return} $N_H(r) + N_T(r)$; \hfill \textgreater{} Returns 0 if all robots flipped the same.
\end{enumerate}
\textbf{:End}
Algorithm 5 \textbf{FindMin}(L, N_0)

\textbf{Input}: \(L > 0\) (real); \(N_0\) (real); \(\triangleleft\) upper bound cycle length; quantity to find the minimum of.

\textbf{Initialize}: \(N(r) \leftarrow N_0\); \(\triangleright\) Will contain the minimum of the inputs \(N_0\).

\textbf{Begin}:
1: \textbf{if} \(b(r) = H\) then \(\triangleright H\) was outcome of last coin-flip
2: Move \textit{cw} a distance \(L\) and then move \textit{ccw} a distance \(L\);
3: \textbf{if} robot \(s\) with \(b(s) = T\) is encountered \textbf{then} \(N(r) \leftarrow \min\{N(r), N(s)\}\);
4: \textbf{else} \(\triangleright T\) was outcome of last coin-flip
5: \textbf{Wait for time} \(2L\);
6: \textbf{if} robot \(s\) with \(b(s) = H\) is encountered \textbf{then} \(N(r) \leftarrow \min\{N(r), N(s)\}\);
7: \textbf{return} \(N(r)\);
\textbf{:End}

Algorithm 6 \textbf{CountRobots}(\ell)

\textbf{Input}: \(\ell > 0\) (real); \(\triangleright\) The length of the cycle.

\textbf{Initialize}: \(N(r)\); \(\triangleright\) Will contain the computed value of \(n\).

\textbf{Begin}:
1: \textbf{repeat}
2: \(b(r) \leftarrow \text{flip}()\); \(N(r) \leftarrow \text{CountFlips}(\ell)\);
3: \textbf{until} \(N(r) > 0\)
4: \textbf{return} \(N(r)\);
\textbf{:End}

Algorithm 7 \textbf{BoundCycle}(n)

\textbf{Input}: \(n > 0\) (integer); \(\triangleright\) The number of robots.

\textbf{Initialize}: \(N(r)\); \(t \leftarrow -1\);

\textbf{Begin}:
1: \textbf{repeat}
2: \(t \leftarrow t + 1\);
3: \(L_t = n \cdot 2^{t-1}\);
4: \(b(r) \leftarrow \text{flip}()\);
5: \(N(r) \leftarrow \text{CountFlips}(L_t)\);
6: \(N(r) \leftarrow \text{FindMin}(L_t, N(r))\);
7: \textbf{until} \(N(r) > n\)
8: \textbf{return} \(\frac{2L_t}{\sqrt{N(r)/n}}\);
\textbf{:End}

\section*{E Pseudocode for algorithms of Section 4.2}

Algorithm 8 \textbf{BoundRobots}(L)

\textbf{Input}: \(L > 0\), real \(\triangleright\) upper bound on the length of the cycle.

\textbf{Initialize}: \(N(r)\); \(\triangleright\) Will contain the computed upper bound on \(n\).

\textbf{Begin}:
1: \textbf{repeat}
2: \(b(r) \leftarrow \text{flip}()\); \(N(r) \leftarrow \text{CountFlips}(L)\);
3: \textbf{until} \(N(r) > 0\)
4: \(N(r) \leftarrow \text{FindMin}(L, N(r))\);
5: \textbf{return} \(N(r)\);
\textbf{:End}