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* $\mathcal{C}$ (e.g., the perimeter of a closed region of $\mathcal{R}^2$) of arbitrary length ℓ. 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* (GATHER), requiring all robots to meet at a same location; and *election* (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 ℓ. 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 GATHER and 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 $\mathcal{U}$. 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* (GATHER), requiring all robots to meet at a same location; and *election* (ELECT), requiring all robots to agree on a single one as the "leader". Indeed, GATHER is one of the fundamental problems in theoretical mobile robotics, while ELECT is typically solved as an intermediate step in the resolution of many important problems, in particular

© Paola Flocchini, Ryan Killick, Evangelos Kranakis, Nicola Santoro, and Masafumi Yamashita;
licensed under Creative Commons License CC-BY

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* ($\mathcal{LCM}$) and *Continuous Time* ($\mathcal{CT}$). In $\mathcal{LCM}$ 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 $\mathcal{CT}$ the robots are permanently active and continosly 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 $\mathcal{R}^2$). This spatial setting has been investigated in the $\mathcal{LCM}$ model with respect to the *scattering* problem, requiring identical robots to place themselves at uniform distance along the cycle [13]. In the $\mathcal{CT}$ 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 $\mathcal{CT}$ model in a very weak computational setting: the identical robots can only see their current location and have no location awareness; furthemore 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 **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 teminates 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 ℓ of the cycle and the number n of mobile robots (note that n can be arbitrarily larger than ℓ), but also and more importantly on the *knowledge* (none, exact, upper bound) the robots have on ℓ and/or n.

We estabish several results. In particular, we prove that, with knowledge of ℓ, 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 1 Results according to the knowledge of the robots ("Ex." = exact, "-" = no knowledge, "UB" = upper bound). T_{exp} (resp. B_{exP}) represents the expected time (resp. random-bit) complexity. The column "Type" gives the type of randomized algorithm (LV = Las Vegas, MC = Monte Carlo). The last column gives the corresponding algorithm label in the text. When an upper bound on ℓ (resp. n) is known it is represented by L (resp. N); and the constructed upper bound on n is $\hat{N} = \frac{Ln}{\ell}$.

n	ℓ	T_{exp}	B_{exp}	Type	Algo.
Ex.	UB	$O(L)$	$O(n)$	LV	A1
Ex.	-	$O(n+\ell)$	$O(n + n\log\lceil\ell/n\rceil)$	LV	A1 + A7
-	Ex.	$O(\ell)$	$O(n)$	LV	A1 + A6
UB	UB	$O(L)$	$O(n)$	MC	A3
UB	-	$O(N + N\cdot\ell/n)$	$O(n + n\log\lceil\ell/n\rceil)$	MC	A3 + A7
-	UB	$O(L)$	$O(n)$	MC	A3+A8

Table 2 Same as Table 1 for time and bit complexities with high probability.

n	ℓ	T_{whp}	B_{whp}	Type	Algo.
Ex.	UB	$O(L\log n)$	$O(n\log n)$	LV	A1
Ex.	-	$O(n+\ell\log n)$	$O(n\log n + n\log\lceil\ell/n\rceil)$	LV	A1 + A7
-	Ex.	$O(\ell\log n)$	$O(n\log n)$	LV	A1 + A6
UB	UB	$O(L\log N)$	$O(n\log N)$	MC	A3
UB	-	$O(N + N\cdot\ell/n\cdot\log N)$	$O(n\log n + n\log\lceil\ell/n\rceil)$	MC	A3 + A7
-	UB	$O\left(L\log\hat{N}\right)$	$O\left(n\log\hat{N}\right)$	MC	A3+A8

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 ℓ, 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 ℓ 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 $\Omega(\exp(n))$ 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 $\mathbf{R}$ be a set of $n \geq 2$ autonomous mobile computational entities, called robots, located in a continuous *cycle* $\mathcal{C}$ (e.g., the perimeter of a closed region of $\mathcal{R}^2$) of real length ℓ 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 $r \in \mathbf{R}$ has a local memory composed of a finite set of registers, including a special register $state(r)$ 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 $b(r) \leftarrow flip()$ 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 $\mathcal{C}$ 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 comitting 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 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 ELECT and GATHER, and at what cost. The *election* problem, 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, 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 [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 ℓ 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.

3 Election with knowledge of both n and ℓ

In this section we consider ELECT when the robots possess knowledge of both n and ℓ (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.

3.1 Exact knowledge of n and ℓ

▶ **Theorem 1.** *Let n and ℓ 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.*

The proof is based on the algorithm ElectLV(n, ℓ). This algorithm is formally described as Algorithm 1 and takes as inputs the number of robots n and the length of the cycle ℓ. Initially all robots begin in the same CANDIDATE state and each robot r has 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 $D_t = \min\{\frac{\ell}{2}, \frac{\ell}{n}(4/3)^t\}$, 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 $4D_t$. Those that flip H will: move ccw a distance D_t; return to their initial positions; move cw a distance D_t; 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 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.

▶ **Lemma 2.** *Let n and n' respectively represent the number of CANDIDATE robots before and after* ElectionRound(D) *is run with input $D > 0$. Then $E[n'] \leq \frac{n}{2} + \frac{1}{2}\left\lceil \frac{\ell}{2D} \right\rceil$.*

Proof. Partition the cycle into $m = \left\lceil \frac{\ell}{2D} \right\rceil$ 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 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:

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

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 $E[n'_i] = \sum_{x=0}^{n_i} x \Pr[k_i = x] + \sum_{x=0}^{n_i} x \Pr[n'_i = x | k_i = 0] \Pr[k_i = 0]$.

It is not hard to see that k_i is binomially distributed with parameters n_i and $p = 1/2$ implying that $\sum_{x=0}^{n_i} x \Pr[k_i = x] = n_i/2$, and that $\Pr[k_i = 0] = (1/2)^{n_i}$. The sum $\sum_{x=0}^{n_i} x \Pr[n'_i = x | k_i = 0]$ represents the expected number of CANDIDATE robots surviving in an interval i given that they all flipped T. Clearly this expectation is bounded by n_i and we can thus conclude that $E[n'_i] \leq \frac{n_i}{2} + n_i \left(\frac{1}{2}\right)^{n_i} \leq \frac{n_i}{2} + \frac{1}{2}$.

To bound the expectation of n' we can substitute this inequality into (1) to get $E[n'] = \sum_{i=1}^{m} E[n'_i] \leq \sum_{i=1}^{m} \left(\frac{n_i}{2} + \frac{1}{2}\right) = \frac{n}{2} + \frac{m}{2}$ where we have used the fact that $n = \sum_{i=1}^{m} n_i$ in the last step. Since $m = \left\lceil \frac{\ell}{2D} \right\rceil$ the lemma follows. ◀

▶ **Lemma 3.** *Let n_t count the number of CANDIDATE robots remaining in round $t \geq 0$ of* ElectLV(n, ℓ). *Then $E[n_t] \leq \left\lceil \left(\frac{3}{4}\right)^t n \right\rceil$.*

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 $E[n_{k+1}] \leq \frac{1}{2} \left\lceil \left(\frac{3}{4}\right)^k n \right\rceil + \frac{1}{2} \left\lceil \frac{\ell}{2D_k} \right\rceil$ where $D_t = \min\left\{\frac{\ell}{2}, \frac{\ell}{n}\left(\frac{4}{3}\right)^t\right\}$. The lemma clearly holds if $D_k \geq \frac{\ell}{2}$. If this is not the case then $D_k = \frac{\ell}{n}\left(\frac{4}{3}\right)^k$ 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* ElectLV(n, ℓ) *in which only a single CANDIDATE robot remains. Then $E[T] \leq \left\lfloor \log_{4/3}(n) \right\rfloor + 1$ and, for any positive integer w, $\Pr\left[T \geq \left\lfloor \log_{4/3}(n) \right\rfloor + 1 + w\right] \leq \left(\frac{3}{4}\right)^w \frac{n}{(4/3)^{\left\lfloor \log_{4/3}(n) \right\rfloor}}$.*

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., $E[h(n)] \leq \left\lceil \frac{3}{4} n \right\rceil$. With this observation the lemma follows easily from Theorem 22. ◀

▶ **Lemma 5.** *Let τ be the time required until only a single CANDIDATE robot remains in* ElectLV(n, ℓ). *Then $E[\tau] \leq 8L$ and, for any positive integer w, $\Pr[T \geq 2L(4 + w)] \leq \left(\frac{3}{4}\right)^w \frac{n}{(4/3)^{\left\lfloor \log_{4/3}(n) \right\rfloor}}$.*

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 τ required to complete these T rounds is $\tau = 4\frac{L}{n}\sum_{t=0}^{t_L-1}(4/3)^t + 2\sum_{t=t_L}^{T} L \leq 12\frac{L}{n}(4/3)^{t_L} + 2(T - t_L)L \leq 8L + 2(T - t_L)$. The lemma now follows from Lemma 4. ◀

▶ **Lemma 6.** *Let B be the random variable which counts the number of coin-flips used in* ElectLV(n, ℓ). *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 ℓ

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

Observe that nowhere in the proof of Theorem 1 did we require the robots to know exactly the value of ℓ. In particular, if the robots were to instead use an upper bound L on ℓ then the only change we need to make is to replace ℓ 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 ℓ and an upper bound $N \geq n$ since ElectLV requires the exact value of n in order to terminate. We will see in the next section that exact knowledge of ℓ 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 ℓ 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* 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 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 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 τ 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 $\Pr[T > t_\infty] = \Pr\left[T > \left\lceil \log_{4/3}(N) \right\rceil + w\right] = \Pr\left[T \geq \left\lceil \log_{4/3}(N) \right\rceil + 1 + w\right]$ and this probability is given by Lemma 4. ◀

4 Election with knowledge of either n or ℓ

In this section we investigate ELECT when the robots are provided with knowledge of only one of n or ℓ (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 ℓ 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 ℓ

▶ **Theorem 9.** *Let either n or ℓ be known to the robots. Then there are Las Vegas algorithms solving* ELECT*. If ℓ 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 $O\left(n + n \log \left\lceil \frac{\ell}{n} \right\rceil\right)$ random bits on average and $O\left(n \log(n) + n \log \left\lceil \frac{\ell}{n} \right\rceil\right)$ 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 ℓ. 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 ℓ and assume the robots know only the value of ℓ. 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 ℓ 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 ℓ. This algorithm terminates in time $O(n + \ell)$ on average and with high probability; and requires $O\left(n + n \log \left\lceil \frac{\ell}{n} \right\rceil\right)$ 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 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 ℓ around the cycle and count each robot they encounter which flipped T. The robots that flipped T will likewise wait for a time ℓ 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 ℓ time units, each robot will determine the number of robots which flipped opposite to themselves. In the last ℓ time units of the algorithm the robots which initially flipped H (resp. T) will move CCW a distance ℓ around the cycle (resp. wait for ℓ 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 2ℓ 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 $N_{\mathrm{H}}(r) = N_{\mathrm{T}}(r) = 0$. From this description it is easy to establish the following lemma:

▶ **Lemma 12.** *Assume that all robots have flipped a coin. Then in exactly 2ℓ time units the procedure* CountFlips(ℓ) *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* 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 $N_{\mathrm{H}}(r) + N_{\mathrm{T}}(r) = 0$. 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. $n_{\mathrm{T}} + n_{\mathrm{H}} = n$. 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 $N_{\mathrm{H}}(r) < n_{\mathrm{H}}$ or $N_{\mathrm{T}}(r) < n_{\mathrm{T}}$ and thus it is ensured that $N_{\mathrm{T}}(r) + N_{\mathrm{H}}(r) \geq n$ for all robots. ◀

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* 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 $D < \ell$. ◀

A minimum finding procedure: The minimum finding procedure FindMin(L, N_0) is formally 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 CW 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* FindMin$(L, N_0(r))$ *will result in each robot r computing the minimum of all inputs $N_0(r)$.*

Computing n using ℓ: We will now tackle the proof of Lemma 10 which is based off of the algorithm CountRobots(ℓ). 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 CountFlips(ℓ) 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 CountRobots(ℓ) 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 $\frac{1}{1-2^{1-n}} \leq 2$ rounds. The 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 2ℓ and in each round all n robots flip their coins. ◀

Computing a $O(\ell)$ upper bound on ℓ using n: The proof of Lemma 11 is based off of the algorithm 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 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 ℓ. This will result in each robot r computing a value $N(r)$. 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 FindMin, they will all agree on the computed value of $N(r)$. Let t_* be the first round in which all robots compute $N(r) > n$. The corresponding value of L_t in the round t_* will then be an upper bound on ℓ. We reduce L_{t_*} by a factor $\frac{1}{2}\left\lfloor\frac{N(r)}{n}\right\rfloor$ 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 $t_* > t_0$ in which at least two robots flip differently. Since the probability that all robots flip the same is 2^{1-n} we will have $t_* = t_0 + O(1)$ 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 CountFlips(L_t) and FindMin(L_t) each take time $2L_t$ to complete, each round of the algorithm lasts time $4L_t = n \cdot 2^{t+2}$. The total time required is thus $\sum_{t=0}^{t_*} n \cdot 2^{t+2} = 4n(2^{t_*+1} - 1)$. If $n > 2\ell$ then the above is clearly $O(n)$. If $n \leq 2\ell$ then we have that $4n(2^{t_*+1} - 1) = 4n\left(2^{\left\lceil \log \frac{2\ell}{n} \right\rceil + O(1)} - 1\right) = O(\ell)$.

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 $O\left(n \log \left\lceil \frac{2\ell}{n} \right\rceil\right)$ when $n \leq 2\ell$. ◀

4.2 Inexact knowledge of n or ℓ

We now consider the cases that the robots are only provided with an upper bound on n or only an upper bound on ℓ. The main result follows:

▶ **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 $O\left((3/4)^w\right)$. If the robots know $L \geq \ell$ then the algorithm terminates in time $O\left(wL\right)$ and requires $O\left(wn\right)$ random bits. If the robots know $N \geq n$ then the algorithm terminates in time $O\left(N + w\frac{N}{n}\ell\right)$ and requires $O\left(wn + n\log\left\lceil\frac{\ell}{n}\right\rceil\right)$ random bits.*

Our goal is again to develop algorithms by which the robots will gain knowledge of the unknown of n or ℓ 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 ℓ 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 $N = O\left(\frac{L}{\ell}n\right)$ on n. This algorithm terminates in time $O\left(L\right)$ on average and with high probability; and requires $O\left(n\right)$ random bits on average and with high probability.*

▶ **Lemma 18.** *Consider n robots on a cycle of length ℓ 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 $O\left(\frac{N}{n}\ell\right)$ upper bound L on ℓ. This algorithm terminates in time $O\left(N + \frac{N}{n}\ell\right)$ on average and with high probability; and requires $O\left(n + n\log\left\lceil\frac{\ell}{n}\right\rceil\right)$ 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 ℓ: Here we will use an algorithm essentially identical to CountRobots(ℓ) 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 $O\left(\frac{L}{\ell}n\right)$ upper bound follows from the fact that the robots will traverse the cycle $\frac{L}{\ell}$ times. The asymptotic running time of the algorithm is identical to that of CountRobots with ℓ replaced with L. The random-bit complexity does not change. Lemma 17 follows without proof from this discussion.

Computing an upper bound on ℓ 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 $L_t = N \cdot 2^t \geq 2\left\lceil\frac{N}{n}\right\rceil\ell$, i.e. $t_0 = \left\lceil\log\left(\left\lceil\frac{N}{n}\right\rceil\frac{2\ell}{N}\right)\right\rceil = O\left(\log\left\lceil\frac{\ell}{n}\right\rceil\right)$. ◀

5 Impossibility results

In the previous sections we have developed Las Vegas algorithms which solve ELECT when one of n or ℓ is known exactly to the robots. We have also developed Monte Carlo algorithms when only upper-bounds on n and/or ℓ are known. In the sequel we demonstrate that, unless the robots know at least one of n or ℓ exactly, there does not exist a Las Vegas algorithm which solves ELECT.

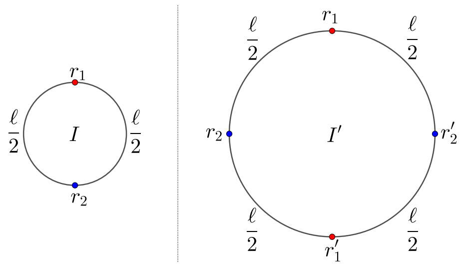

Figure 1 Left: The instance I with two robots r_1 and r_2 on a cycle of length ℓ. Right: The instance I' with four robots r_1, r_2, r'_1, and r'_2 on a cycle of length 2ℓ.

▶ **Theorem 19.** *Assume that the robots do not know ℓ nor n exactly. Then there is no Las Vegas type algorithm which solves* ELECT.

To demonstrate this we first prove the weaker statement that a Las Vegas algorithm cannot exist if the robots know nothing of n nor ℓ.

▶ **Lemma 20.** *If neither n nor ℓ is available then there is no Las Vegas type algorithm which solves* ELECT.

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 ℓ. 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 2ℓ 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 CW 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. ◀

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

▶ **Corollary 21.** *Suppose that the robots only know an upper bound N on n. Then there is no Las Vegas type algorithm which solves* ELECT.

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

$N = 5$ 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 ℓ 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 ℓ. We have established several results including optimal algorithms with respect to time and random bits when the robots know ℓ, 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 ℓ. 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

1. Hagit Attiya and Yishay Mansour. Language complexity on the synchronous anonymous ring. *Theoretical Computer Science*, 53(2-3):169–185, 1987.
2. Hagit Attiya, Marc Snir, and Manfred K. Warmuth. Computing on an anonymous ring. *J. ACM*, 35(4):845–875, 1988.
3. Rena Bakhshi, Wan Fokkink, Jun Pang, and Jaco van de Pol. Leader election in anonymous rings: Franklin goes probabilistic. In *5th IFIP International Conference On Theoretical Computer Science (TCS)*, pages 57–72, 2008.
4. Francesco Bullo, Jorge Cortes, and Sonia Martinez. *Distributed Control of Robotic Networks*. Princeton University Press, 2009.
5. Mark Cielibak, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Distributed computing by mobile robots: Gathering. *SIAM Journal on Computing*, 41(4):829–879, 2012.
6. Reuven Cohen and David Peleg. Convergence properties of the gravitational algorithm in asynchronous robot systems. *SIAM Journal on Computing*, 34(6):1516–1528, 2005.
7. Jurek Czyzowicz, Leszek Gasieniec, Adrian Kosowski, and Evangelos Kranakis. Boundary patrolling by mobile agents with distinct maximal speeds. In *19th Annual European Symposium on Algorithms, ESA*, pages 701–712, 2011.
8. Jurek Czyzowicz, Kostantinos Georgiou, and Evangelos Kranakis. *Patrolling*. In P. Flocchini, G. Prencipe, and N. Santoro, editors, *Distributed Computing by Mobile Entities*, chapter 15, pages 371–400. Springer, 2019.
9. Jurek Czyzowicz, Evangelos Kranakis, Dominik Pajak, and Najmeh Taleb. Patrolling by robots equipped with visibility. In *21st International Colloquium on Structural Information and Communication Complexity, SIROCCO*, pages 224–234, 2014.
10. Yoann Dieudonné, Franck Petit, and Vincent Franck. Leader election problem versus pattern formation problem. In *24th International Symposium on Distributed Computing (DISC)*, pages 267–281, 2010.
11. Ofer Feinerman, Amos Korman, Shay Kutten, and Yoav Rodeh. Fast rendezvous on a cycle by agents with different speeds. In *5th International Conference on Distributed Computing and Networking, ICDCN*, pages 1–13, 2014.
12. Paola Flocchini, Evangelos Kranakis, Danny Krizanc, Flaminia L. Luccio, and Nicola Santoro. Sorting and election in anonymous asynchronous rings. *Journal of Parallel and Distributed Computing*, 64(2):254–265, 2004.
13. Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Self-deployment of mobile sensors on a ring. *Theoretical Computer Science*, 402(1):67–80, 2008.
14. Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. *Distributed Computing by Oblivious Mobile Robots*. Morgan & Claypool, 2012.
15. Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. *Distributed Computing by Mobile Entities*. Springer, 2019.
16. Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Gathering of asynchronous mobile robots with limited visibility. *Theoretical Computer Science*, 337(1-3):147–168, 2006.

17 Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. *Theoretical Computer Science*, 407(1-3):412–447, 2008.

18 Greg N. Frederickson and Nicola Santoro. Breaking symmetry in synchronous networks. In *1st Aegean Workshop on Computing (now SPAA)*, pages 26–33, 1986.

19 Nao Fujinaga, Yukiko Yamauchi, Shuji Kijima, and Masafumi Yamashita. Asynchronous pattern formation by anonymous oblivious mobile robots. *SIAM Journal on Computing*, 44(3):740–785, 2015.

20 Noam Gordon, Israel A. Wagner, and Alfred M. Bruckstein. A randomized gathering algorithm for multiple robots with limited sensing capabilities. In *International Workshop on Multi-Agent Robotic Systems*, 2005.

21 Rajiv Gupta, Scott A. Smolka, and Shaji Bhaskar. On randomization in sequential and distributed algorithms. *ACM Comput. Surv.*, 26(1):7–86, 1994.

22 Evan Huus and Evangelos Kranakis. Rendezvous of many agents with different speeds in a cycle. In *14th International Conference on Ad-Hoc Networks and Wireless, ADHOC-NOW*, pages 195–209, 2015.

23 Alon Itai and Michael Rodeh. Symmetry breaking in distributed networks. *Information and Computation*, 88(1):60–87, 1990.

24 Taisuke Izumi, Tomoko Izumi, Sayaka Kamei, and Fukuhito Ooshita. Randomized gathering of mobile robots with local-multiplicity detection. In *11th International Symposium on Stabilizing, Safety, and Security of Distributed Systems, SSS*, pages 384–398, 2009.

25 Richard M. Karp. Probabilistic recurrence relations. *J. ACM*, 41(6):1136–1150, 1994.

26 Evangelos Kranakis, Danny Krizanc, and Euripides Markou. *The Mobile Agent Rendezvous Problem in the Ring.* Morgan & Claypool, 2010.

27 Ji Lin, A. Stephen Morse, and Brian D.O. Anderson. The multi-agent rendezvous problem. Parts 1 and 2. *SIAM Journal on Control and Optimization*, 46(6):2096–2147, 2007.

28 Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

29 Fukuhito Ooshita, Shinji Kawai, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. Randomized gathering of mobile agents in anonymous unidirectional ring networks. *IEEE Transactions on Parallel and Distributed Systems*, 25(5):1289–1296, 2014.

30 Yukiko Yamauchi. *Symmetry of Anonymous Robots.* In P. Flocchini, G. Prencipe, and N. Santoro, editors, *Distributed Computing by Mobile Entities*, chapter 6, pages 109–133. Springer, 2019.

A Description of Karp's theorem

Consider the stochastic recurrence relation

$$T(n) = a(n) + T(h(n)) \tag{2}$$

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 $m(n)$ is 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

$$u(n) = \sum_{k=0}^{\infty} a(m^{[k]}(n)) \tag{3}$$

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

Algorithm 1 ElectLV(n, ℓ)

Input: $n > 0$ (integer); $\ell > 0$ (real); ▷ The number of robots and the length of the cycle.
Initialize: *state*$(r) \leftarrow$ CANDIDATE; cnr$(r) \leftarrow 1$; $t \leftarrow 0$;
Begin:

```
1: repeat
2:     D ← min{ ℓ/2, (ℓ/n)(4/3)^t };
3:     ElectionRound(D); t ← t + 1;               ▷ Run one election round.
4:     if cnr(r) = n then state(r) ← LEADER;     ▷ Stack contains n robots, terminate.
5: until state(r) = FOLLOWER or LEADER
```

:End

Algorithm 2 ElectionRound(D)

Input: $D > 0$ (real);
Begin: $b(r) \leftarrow \mathit{flip}()$;
if $b(r) = \mathrm{H}$ **then** ▷ H was flipped
  Move CCW a distance D; CW a distance $2D$; CCW a distance D;
  if a robot s with $b(s) = \mathrm{T}$ is encountered while moving **then**
    $\textsc{cnr}(r) \leftarrow \textsc{cnr}(r) + \textsc{cnr}(s)$; ▷ Update $\textsc{cnr}(r)$ since s will merge with r.
else ▷ T was flipped
  Remain stationary for time $4D$:
  if a robot s with $b(s) = \mathrm{H}$ is encountered while waiting **then**
    $\mathit{state}(r) = \mathrm{FOLLOWER}$;
    Merge with robot s;
:End

C Pseudocode for algorithms of Section 3.2

Algorithm 3 ElectMC(N, L, w)

Input: $N > 0$ (integer); $L > 0$ (real); $w \geq 0$ (integer); ▷ upper bounds on n and ℓ; termination parameter w.
Initialize: $\mathit{state}(r) \leftarrow \mathrm{CANDIDATE}$; $t \leftarrow 0$; $t_\infty \leftarrow \lceil \log_{4/3}(n) \rceil + w$; ▷ t_∞ = termination round.
Begin:
repeat
  $D_t \leftarrow \min\left\{\frac{L}{2},\ \frac{L}{N}\left(\frac{4}{3}\right)^t\right\}$;
  ElectionRound(D_t); $t \leftarrow t + 1$; ▷ Run one election round.
until $\mathit{state}(r) = \mathrm{FOLLOWER}$ or $t = t_\infty$
if $\mathit{state}(r) = \mathrm{CANDIDATE}$ **then** $\mathit{state}(r) \leftarrow \mathrm{LEADER}$;
:End

D Pseudocode for algorithms of Section 4.1

Algorithm 4 CountFlips(D)

Input: $D > 0$ (real); ▷ An estimate of the length of the cycle.
Initialize: $N_{\mathrm{H}}(r) \leftarrow 0$; $N_{\mathrm{T}}(r) \leftarrow 0$; ▷ To count the robots flipping H and T.
Begin:
if $b(r) = \mathrm{H}$ **then** ▷ H was outcome of last coin flip
  Move CW a distance D;
  if a robot s with $b(s) = \mathrm{T}$ is encountered while moving **then** $N_{\mathrm{T}}(r) \leftarrow N_{\mathrm{T}}(r) + 1$;
  Move CCW a distance D;
  if $N_{\mathrm{H}}(r) = 0$ and a robot s with $b(s) = \mathrm{T}$ is encountered while moving **then**
    $N_{\mathrm{H}}(r) \leftarrow N_{\mathrm{H}}(s)$; ▷ Determine N_{H}.
else ▷ T was outcome of last coin flip
  Wait for time D;
  if a robot s with $b(s) = \mathrm{H}$ is encountered while waiting **then** $N_{\mathrm{H}}(r) \leftarrow N_{\mathrm{H}}(r) + 1$;
  Wait for time D;
  if $N_{\mathrm{T}}(r) = 0$ and a robot s with $b(s) = \mathrm{H}$ is encountered while waiting **then**
    $N_{\mathrm{T}}(r) \leftarrow N_{\mathrm{T}}(s)$; ▷ Determine N_{T}.
return $N_{\mathrm{H}}(r) + N_{\mathrm{T}}(r)$; ▷ Returns 0 if all robots flipped the same.
:End

Algorithm 5 FindMin(L, N_0)

Input: $L > 0$ (real); N_0 (real); ▷ upper bound cycle length; quantity to find the minimum of.
Initialize: $N(r) \leftarrow N_0$; ▷ Will contain the minimum of the inputs N_0.
Begin:
if $b(r) = \mathrm{H}$ **then** ▷ H was outcome of last coin-flip
  Move CW a distance L and then move CCW a distance L;
  if robot s with $b(s) = \mathrm{T}$ is encountered **then** $N(r) \leftarrow \min\{N(r), N(s)\}$;
else ▷ T was outcome of last coin-flip
  Wait for time $2L$;
  if robot s with $b(s) = \mathrm{H}$ is encountered **then** $N(r) \leftarrow \min\{N(r), N(s)\}$;
return $N(r)$;
:End

Algorithm 6 CountRobots(ℓ)

Input: $\ell > 0$ (real); ▷ The length of the cycle.
Initialize: $N(r)$; ▷ Will contain the computed value of n.
Begin:
repeat
  $b(r) \leftarrow \mathit{flip}()$; $N(r) \leftarrow$ CountFlips(ℓ);
until $N(r) > 0$
return $N(r)$;
:End

Algorithm 7 BoundCycle(n)

Input: $n > 0$ (integer); ▷ The number of robots.
Initialize: $N(r)$; $t \leftarrow -1$;
Begin:
repeat
  $t \leftarrow t + 1$;
  $L_t = n \cdot 2^{t-1}$;
  $b(r) \leftarrow \mathit{flip}()$;
  $N(r) \leftarrow$ CountFlips(L_t);
  $N(r) \leftarrow$ FindMin$(L_t, N(r))$;
until $N(r) > n$
return $\frac{2L_t}{\lfloor N(r)/n \rfloor}$;
:End

E Pseudocode for algorithms of Section 4.2

Algorithm 8 BoundRobots(L)

Input: $L > 0$, real ▷ upper bound on the length of the cycle.
Initialize: $N(r)$; ▷ Will contain the computed upper bound on n.
Begin:
repeat
  $b(r) \leftarrow \mathit{flip}()$; $N(r) \leftarrow$ CountFlips(L);
until $N(r) > 0$
$N(r) \leftarrow$ FindMin$(L,\ N(r))$;
return $N(r)$;
:End