Selecting a Leader in a Network of Finite State Machines

Yehuda Afek[1]
Tel Aviv University, Tel Aviv, Israel
afek@cs.tau.ac.il

Yuval Emek[2]
Technion - Israel Institute of Technology, Haifa, Israel
yemek@technion.ac.il

Noa Kolikant
Tel Aviv University, Tel Aviv, Israel
noakolikant@mail.tau.ac.il

Abstract

This paper studies a variant of the *leader election* problem under the *stone age* model (Emek and Wattenhofer, PODC 2013) that considers a network of n randomized finite automata with very weak communication capabilities (a multi-frequency asynchronous generalization of the *beeping* model's communication scheme). Since solving the classic leader election problem is impossible even in more powerful models, we consider a relaxed variant, referred to as *k-leader selection*, in which a leader should be selected out of at most k initial candidates. Our main contribution is an algorithm that solves k-leader selection for bounded k in the aforementioned stone age model. On (general topology) graphs of diameter D, this algorithm runs in $\tilde{O}(D)$ time and succeeds with high probability. The assumption that k is bounded turns out to be unavoidable: we prove that if $k = \omega(1)$, then no algorithm in this model can solve k-leader selection with a (positive) constant probability.

2012 ACM Subject Classification Theory of computation → Distributed computing models

Keywords and phrases stone age model, beeping communication scheme, leader election, k-leader selection, randomized finite state machines, asynchronous scheduler

Digital Object Identifier 10.4230/LIPIcs.DISC.2018.4

1 Introduction

Many distributed systems rely on the existence of one distinguishable node, often referred to as a *leader*. Indeed, the *leader election* problem is among the most extensively studied problems in distributed computing [23, 9, 29, 3]. Leader election is not confined to digital computer systems though as the dependency on a unique distinguishable node is omnipresent in *biological systems* as well [27, 34, 28]. A similar type of dependency exists also in networks of man-made micro- and even nano-scale sub-microprocessor devices [16].

The current paper investigates the task of electing a leader in networks operating under the *stone age (SA)* model [20] that provides an abstraction for distributed computing by nodes that are significantly inferior to modern computers in their computation and communication capabilities. In this model, the nodes are controlled by randomized finite automata and

[1] The work of Y. Afek was partially supported by a grant from the Blavatnik Cyber Security Council and the Blavatnik Computer Science Research Fund.
[2] The work of Y. Emek was supported in part by an Israeli Science Foundation grant number 1016/17.

can communicate with their network neighbors using a fixed message alphabet based on a weak communication scheme that can be viewed as an asynchronous extension of the *set broadcast (SB)* communication model of [25] (a formal definition of our model is provided in Section 1.1).

Since the state space of a node in the SA model is fixed and does not grow with the size of the network, SA algorithms are inherently *uniform*, namely, the nodes are anonymous and lack any knowledge of the network size. Unfortunately, classic impossibility results state that leader election is hopeless in these circumstances (even under stronger computational models): Angluin [4] proved that uniform algorithms cannot solve leader election in a network with success probability 1; Itai and Rodeh [26] extended this result to algorithms that are allowed to fail with a bounded probability.

Thus, in the distributed systems that interest us, leader election cannot be solved by the nodes themselves and some "external help" is necessary. This can be thought of as an external *symmetry breaking signal* that only one node is supposed to receive. Symmetry breaking signals are actually quite common in reality and can come in different shape and form. A prominent example for such external signaling occurs during the development process of multicellular organisms, when ligand molecules flow through a cellular network in a certain direction, hitting one cell before the others and triggering its differentiation [35].

But what if the symmetry breaking signal is *noisy* and might be received by a handful of nodes? Is it possible to detect that several nodes received this signal? Can the system recover from such an event or is it doomed to operate with multiple leaders instead of one?

In this paper, we study the *k-leader selection* problem, where at most k (and at least 1) nodes are initially marked as *candidates*, out of which exactly one should be selected. On top of the relevance of this problem to the aforementioned questions, it is also motivated by the following application. Consider scenarios where certain nodes, including the leader, may get lost during the network deployment process, e.g., a sensor network whose nodes are dropped from an airplane. In such scenarios, one may wish to produce $k > 1$ candidate leaders with the purpose of increasing the probability that at least one of them survives; a k-leader selection algorithm should then be invoked to ensure that the network has exactly one leader when it becomes operational.

The rest of the paper is organized as follows. In Section 1.1, we provide a formal definition of the distributed computing model used in the paper. Our results are summarized in Section 1.2 and some additional related literature is discussed in Section 1.3. A k-leader selection algorithm that constitutes our main technical contribution, is presented in Section 2, whereas Section 3 provides some negative results.

1.1 Model

The distributed computing model considered in this paper follows the *stone age (SA)* model of Emek and Wattenhofer [20]. Under this model, the communication network is represented by a finite connected undirected graph $G = (V, E)$ whose nodes are controlled by *randomized finite automata* with state space Q, message alphabet Σ, and transition function τ whose role is explained soon.

Each node $v \in V$ of degree d_v is associated with d_v *input ports* (or simply *ports*), one port $\psi_v(u)$ for each neighbor u of v in G, holding the last message $\sigma \in \Sigma$ received from u at v. The communication model is defined so that when node u sends a message, the same message is delivered to all its neighbors v; when (a copy of) this message reaches v, it is written into port $\psi_v(u)$, overwriting the previous message in this port. Node v's (read-only) access to its own ports $\psi_v(\cdot)$ is very limited: for each message type $\sigma \in \Sigma$, it can only distinguish between the case where σ is not written in any port $\psi_v(\cdot)$ and the case where it is written in at least one port.

The execution is event driven with an asynchronous scheduler that schedules the aforementioned message delivery events as well as node activation events.[3] When node $v \in V$ is activated, the transition function $\tau : Q \times \{0,1\}^{\Sigma} \to 2^{Q \times \Sigma}$ determines (in a probabilistic fashion) its next state $q' \in Q$ and the next message $\sigma' \in \Sigma$ to be sent based on its current state $q \in Q$ and the current content of its ports. Formally, the pair (q', σ') is chosen uniformly at random from $\tau(q, \chi_v)$, where $\chi_v \in \{0,1\}^{\Sigma}$ is defined so that $\chi_v(\sigma) = 1$ if and only if σ is written in at least one port $\psi_v(\cdot)$.

To complete the definition of the randomized finite automata, one has to specify the set $Q_{in} \subseteq Q$ of initial states that encode the node's input, the set $Q_{out} \subseteq Q$ of output states that encode the node's output, and the initial message $\sigma_0 \in \Sigma$ written in the ports when the execution begins. SA algorithms are required to have *termination detection*, namely, every node must eventually decide on its output and this decision is irrevocable.

Following the convention in message passing distributed computing (cf. [32]), the *run-time* of an asynchronous SA algorithm is measured in terms of *time units* scaled to the maximum of the time it takes to deliver any message and the time between any two consecutive activations of a node. Refer to [20] for a more detailed description of the SA model.

The crux of the SA model is that the number of states in Q and the size of the message alphabet Σ are constants independent of the size (and any parameter) of the graph G. Moreover, node v cannot distinguish between its ports and in general, its degree may be larger than $|Q|$ (and $|\Sigma|$).

Weakening the Communication Assumptions. The model defined in the current paper is a restriction of the model of [20], where the algorithm designer could choose an additional constant *bounding parameter* $b \in \mathbb{Z}_{>0}$, providing the nodes with the capability to count the number of ports holding message $\sigma \in \Sigma$ up to b. In the current paper, the bounding parameter is set to $b = 1$. This model choice can be viewed as an asynchronous multi-frequency variant of the *beeping* communication model [11, 2].

Moreover, in contrast to the existing SA literature, the communication graph $G = (V, E)$ assumed in the current paper may include *self-loops* of the form $(v, v) \in E$ which means, in accordance with the definition of the SA model, that node v admits port $\psi_v(v)$ that holds the last message received from itself. Using the terminology of the beeping model literature (see, e.g., [2]), the assumption that the communication graph is free of self-loops corresponds to a *sender collision detection*, whereas lifting this assumption means that node v may not necessarily distinguish its own transmitted message from those of its neighbors.

It turns out that self-loops have a significant effect on the power of SA algorithms. Indeed, while a SA algorithm that solves the *maximal independent set (MIS)* problem with probability 1 is presented in [20] under the assumption that the graph is free of self-loops, we prove in Section 3 that if the graph is augmented with self-loops, then no SA algorithm can solve this problem with a bounded failure probability. To distinguish between the original model of [20] and the one considered in the current paper, we hereafter denote the latter by $\text{SA}^{\circlearrowleft}$.

[3] The only assumption we make on the event scheduling is FIFO message delivery: a message sent by node u at time t is written into port $\psi_v(u)$ of its neighbor v before the message sent by u at time $t' > t$.

1.2 Results

Throughout, the number of nodes and the diameter of the graph G are denoted by n and D, respectively. We say that an event occurs *with high probability (whp)* if its probability is at least $1 - n^{-c}$ for an arbitrarily large constant c. Our main technical contribution is cast in the following two theorems.

▶ **Theorem 1.** *For any constant k, there exists a $\mathrm{SA}^{\circlearrowleft}$ algorithm that solves the k-leader selection problem in $\tilde{O}(D)$ time whp.*[4]

▶ **Theorem 2.** *If the upper bound k on the number of candidates may grow as a function of n, then there does not exist a SA algorithm (operating on graphs with no self-loops) that solves the k-leader selection problem with a failure probability bounded away from 1.*

We emphasize that the failure probability of the $\mathrm{SA}^{\circlearrowleft}$ algorithm promised in Theorem 1 (i.e., the probability that the algorithm selects multiple leaders or that it runs for more than $\tilde{O}(D)$ time) is inverse polynomial in n even though each individual node does not (and cannot) possess any notion of n – to a large extent, this, together with the termination detection requirement, capture the main challenge in designing the promised algorithm.[5] The theorem assumes that $k = O(1)$ and hides the dependency of the algorithm's parameters on k. A closer look at its proof reveals that our $\mathrm{SA}^{\circlearrowleft}$ algorithm uses local memory and messages of size $O(\log k)$ bits. Theorem 2 asserts that the dependence of these parameters on k is unavoidable. Whether this dependence can be improved beyond $O(\log k)$ remains an open question.

1.3 Additional Related Literature

As mentioned earlier, the SA model was introduced by Emek and Wattenhofer in [20] as an abstraction for distributed computing in networks of devices whose computation and communication capabilities are far weaker than those of a modern digital computer. Their main focus was on distributed problems that can be solved in sub-diameter (specifically, $\log^{O(1)} n$) time including MIS, tree coloring, coloring bounded degree graphs, and maximal matching. This remained the case also in [19], where Emek and Uitto studied SA algorithms for the MIS problem in dynamic graphs. In contrast, the current paper considers the k-leader selection problem – an inherently global problem that requires $\Omega(D)$ time.

Computational models based on networks of finite automata have been studied for many years. The best known such model is the extensively studied *cellular automata* that were introduced by Ulam and von Neumann [31] and became popular with Martin Gardner's Scientific American column on Conway's *game of life* [24] (see also [37]).

Another popular model that considers a network of finite automata is the *population protocols* model, introduced by Angluin et al. [5] (see also [6, 30]), where the network entities communicate through a sequence of atomic pairwise interactions controlled by a fair (adversarial or randomized) scheduler. This model provides an elegant abstraction for networks of mobile devices with proximity derived interactions and it also fits certain types of chemical reaction networks [18]. Some work on population protocols augments the model with

[4] The asymptotic notation $\tilde{O}(\cdot)$ may hide $\log^{O(1)} n$ factors.

[5] If we aim for a failure probability inverse polynomial in k (rather than n) and we do not insist on termination detection, then the problem is trivially solved by the algorithm that simply assigns a random ID from a set of size $k^{O(1)}$ to each candidate and then eliminates a candidate if it encounters an ID larger than its own.

a graph defined over the population's entities so that the pairwise interactions are restricted to graph neighbors, thus enabling some network topology to come into play. However, for the kinds of networks we are interested in, the fundamental assumption of sequential atomic pairwise interactions may provide the population protocol with unrealistic advantage over weaker message passing variants (including the SA model) whose communication schemes do not enable a node to interact with its individual neighbors independently. Furthermore, population protocols are typically required to *eventually converge* to a correct output and are allowed to return arbitrary (wrong) outputs beforehand, a significantly weaker requirement than the termination detection requirement considered in this paper.

The neat *amoebot model* introduced by Dolev et al. [17] also considers a network of finite automata in a (hexagonal) grid topology, but in contrast to the models discussed so far, the particles in this network are augmented with certain mobility capabilities, inspired by the amoeba contraction-expansion movement mechanism. Since its introduction, this model was successfully employed for the theoretical investigation of self-organizing particle systems [36, 15, 13, 16, 14, 10, 12], especially in the context of *programmable matter*.

Leader election is arguably the most fundamental problem in distributed systems coordination and has been extensively studied from the early days of distributed computing [23, 22]. It is synonymous in most models to the construction of a spanning tree – another fundamental problem in distributed computing – where the root is typically the leader. Leader election has many applications including deadlock detection, choosing a key/password distribution center, and implementing a distributed file system manager. It also plays a key role in tasks requiring a reliable centralized coordinating node, e.g., Paxos and Raft, where leader election is used for consensus – yet another fundamental distributed computing problem, strongly related to leader election. Notice that in our model, leader selection does not (and cannot) imply a spanning tree, but it does imply consensus.

Angluin [4] proved that uniform algorithms cannot break symmetry in a ring topology with success probability 1. Following this classic impossibility result, many symmetry breaking algorithms (with and without termination detection) that relax some of the assumptions in [4] were introduced [1, 7, 26, 33, 3]. Itai and Rodeh [26] were the first to design randomized leader election algorithms with bounded failure probability in a ring topology, assuming that the nodes know n. Schieber and Snir [33] and Afek and Matias [3] extended their work to arbitrary topology graphs.

2 $\mathrm{SA}^{\circlearrowleft}$ Algorithm for k-Leader Selection

In this section, we present our $\mathrm{SA}^{\circlearrowleft}$ algorithm and establish Theorem 1. We start with some preliminary definitions and assumptions presented in Section 2.1. Sections 2.2 and 2.3 are dedicated to the basic subroutines on which our algorithm relies. The algorithm itself is presented in Section 2.4, where we also establish its correctness. Finally, in Section 2.5, we analyze the algorithm's run-time.

2.1 Preliminaries

As explained in Section 1.1, the execution in the SA (and $\mathrm{SA}^{\circlearrowleft}$) model is controlled by an asynchronous scheduler. One of the contributions of [20] is a SA *synchronizer* implementation (cf. the α-synchronizer of Awerbuch [8]). Given a synchronous SA algorithm $\mathcal{A}$ whose execution progresses in fully synchronized *rounds* $t \in \mathbb{Z}_{>0}$ (with simultaneous wake-up), the synchronizer generates a valid (asynchronous) SA algorithm $\mathcal{A}'$ whose execution progresses in *pulses* such that the actions taken by $\mathcal{A}'$ in pulse t are identical to those taken by $\mathcal{A}$

in round t.[6] The synchronizer is designed so that the asynchronous algorithm $\mathcal{A}'$ has the same bounding parameter b ($= 1$ in the current paper) and asymptotic run-time as the synchronous algorithm $\mathcal{A}$.

Although the model considered by Emek and Wattenhofer [20] assumes that the graph has no self-loops, it is straightforward to apply their synchronizer to graphs that do include self-loops, hence it can work also in our $\mathrm{SA}^{\circlearrowleft}$ model. Consequently, in what follows, we restrict our attention to synchronous $\mathrm{SA}^{\circlearrowleft}$ algorithms. Specifically, we assume that the execution progresses in synchronous rounds $t \in \mathbb{Z}_{>0}$, where in round t, each node v

(1) receives the messages sent by its neighbors in round $t-1$;
(2) updates its state; and
(3) sends a message to its neighbors (same message to all neighbors).

Since we make no effort to optimize the size of the messages used by our algorithm, we assume hereafter that the message alphabet Σ is identical to the state space Q and that node v simply sends its current state to its neighbors at the end of every round. Nevertheless, for clarity of the exposition, we sometimes describe the algorithm in terms of sending designated messages, recalling that this simply means that the states of the nodes encode these messages.

To avoid cumbersome presentation, our algorithm's description does not get down to the resolution of the state space Q and transition function τ. It is straightforward though to implement our algorithm as a randomized finite automaton, adhering to the model presented in Section 1.1. In this regard, at the risk of stating the obvious, we remind the reader that if k is a constant, then a finite automaton supports arithmetic operations modulo $O(k)$.

In the context of the k-leader selection problem, we use the verb *withdraw* when referring to a node that ceases to be a candidate.

2.2 The Ball Growing Subroutine

We present a generic *ball growing* subroutine in graph $G = (V, E)$ with at most k candidates. The subroutine is initiated at (all) the candidates, not necessarily simultaneously, through designated signals discussed later on. During its execution, some candidates may withdraw; in the context of this subroutine, we refer to the surviving candidates as *roots*.

The ball growing subroutine assigns a *level* variable $\lambda(v) \in \{0, 1, \ldots, M-1\}$ to each node v, where $M = 2k+2$. Path $P = (v_1, \ldots, v_q)$ in G is called *incrementing* if $\lambda(v_{j+1}) = \lambda(v_j) + 1 \bmod M$ for every $1 \leq j \leq q-1$. The set of nodes reachable from a root r via an incrementing path is referred to as the *ball* of r, denoted by $B(r)$. We design this subroutine so that the following lemma holds.

▶ **Lemma 3.** *Upon termination of the ball growing subroutine,*
(1) *every incrementing path is a shortest path (between its endpoints) in G;*
(2) *every root belongs to exactly one ball (its own); and*
(3) *every non-root node belongs to at least one ball.*

Intuition spotlight: A natural attempt to design the ball growing subroutine is to grow a breadth first search tree around candidate r, layer by layer, so that node v at distance d from r is assigned with level variable $\lambda(v) = d \bmod M$. This is not necessarily

[6] We emphasize the role of the assumption that when the execution begins, the ports hold the designated initial message σ_0. Based on this assumption, a node can "sense" that some of its neighbors have not been activated yet, hence synchronization can be maintained right from the beginning.

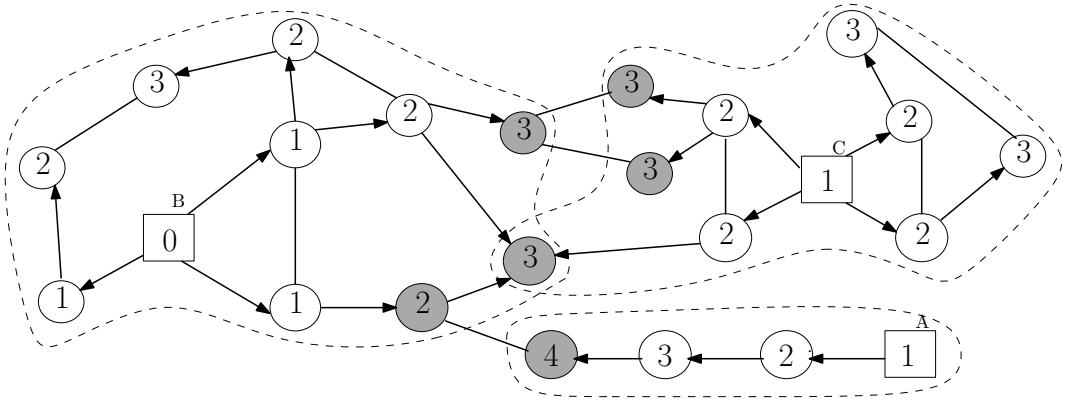

Figure 1 The result of a ball growing process invoked at candidate A in round 1, candidate B in round 2, and candidate C in round 3. The level variables $\lambda(\cdot)$ are depicted by the numbers written inside the nodes and the balls are depicted by the dashed curves. The boundary nodes appear with a gray background. The DAG $\vec{G}$ is depicted by the oriented edges.

possible though when multiple candidates exist: What happens if the ball growing processes of different candidates reach v in the same round? What happens if these ball growing processes reach several adjacent nodes in the same round? If we are not careful, these scenarios may lead to incrementing paths that are not shortest paths and even to cyclic incrementing paths. Things become even more challenging considering the weak communication capabilities of the nodes that may prevent them from distinguishing between the ball growing processes of different candidates.

The ball growing subroutine is implemented under the $\text{SA}^{\circlearrowleft}$ model by disseminating $\texttt{GrowBall}(\ell)$ messages, $\ell \in \{0, 1, \ldots, M-1\}$, throughout the graph. Consider a candidate r and let $s(r)$ be the round in which it is signaled to invoke the ball growing subroutine. If r receives a $\texttt{GrowBall}(\cdot)$ message in some round $t \leq s(r)$, then r withdraws and subsequently follows the protocol like any other non-root node; otherwise, r becomes a root in round $s(r)$. If $s(r)$ is even (resp., odd), then r assigns $\lambda(r) \leftarrow 0$ (resp., $\lambda(r) \leftarrow 1$) and sends a $\texttt{GrowBall}(\lambda(r))$ message.

Consider a non-root node v and let $g(v)$ be the first round in which it receives a $\texttt{GrowBall}(\cdot)$ message. Notice that v may receive several $\texttt{GrowBall}(\ell)$ messages with different arguments ℓ in round $g(v)$ – let L be the set of all such arguments ℓ. Node v assigns $\lambda(v) \leftarrow \ell'$ and sends a $\texttt{GrowBall}(\ell')$ message at the end of round $g(v)$, where ℓ' is chosen to be any integer in $\{0, 1, \ldots, M-1\}$ that satisfies:

- **(i)** $\ell' - 1 \bmod M \in L$; and
- **(ii)** $\ell' + 1 \bmod M \notin L$.

This completes the description of the ball growing subroutine. Refer to Figure 1 for an illustration.

Intuition spotlight: Condition (i) ensures that v joins the ball $B(r)$ of some root r. By condition (ii), nodes do not join $B(r)$ "indirectly" (this could have led to incrementing paths that are not shortest paths).

Proof of Lemma 3. Consider a (root or non-root) node $v \in V$ and let $p(v)$ be the round in which v starts its active participation in the ball growing process. More formally, if v is a root (i.e., it is a candidate signaled to invoke the ball growing subroutine strictly before

receiving any `GrowBall`$(\cdot)$ message), then $p(v) = s(v)$; otherwise, $p(v) = g(v)$. The following properties are established by (simultaneous) induction on the rounds:

- In any round $t \geq p(v)$, variable $\lambda(v)$ is even if and only if $p(v)$ is even.
- In any round $t \geq p(v)$, node v has a neighbor u with $\lambda(u) = \lambda(v) - 1 \bmod M$ if and only if v is not a root.
- In any round $t \geq p(v)$, node v belongs to ball $B(r)$ for some root r.
- In any round $t \geq p(v)$, if $v \in B(r)$ for some root r, then the incrementing path(s) that realize this relation are shortest paths in the graph.
- If $u, v \in B(r)$ for some root r and $p(u) = p(v)$, then $\lambda(u) = \lambda(v)$.
- The total number of different arguments ℓ in the `GrowBall`(ℓ) messages sent during a single round is at most k.
- Non-root node v finds a valid value to assign to $\lambda(v)$ in round $g(v) = p(v)$.

The assertion follows. ◀

▶ **Observation 4.** *If t is the earliest round in which the ball growing process is initiated at some candidate, then the process terminates by round $t + O(D)$.*

Boundary Nodes. We will see in Section 2.4 that our algorithm detects candidate multiplicity by identifying the existence of multiple balls in the graph. The key notion in this regard is the following one (see Figure 1): Node v is said to be a *boundary* node if

(1) $v \in B(r) \cap B(r')$ for roots $r \neq r'$; or

(2) $v \in B(r)$ for some root r and there exists a neighbor v' of v such that $v' \notin B(r)$.

▶ **Observation 5.** *If the graph has multiple roots, then every ball includes at least one boundary node.*

Node v is said to be a *locally observable boundary* node if it has a neighbor v' such that $\lambda(v') \notin \{\lambda(v) + \ell \bmod M \mid \ell = -1, 0, +1\}$. Notice that by Lemma 3, there cannot be a ball that includes both v and v' since then, at least one of the incrementing paths that realize these inclusions is not a shortest path. Therefore, a locally observable boundary node is in particular a boundary node.

The Directed Acyclic Graph $\vec{G}$. Given two adjacent nodes u and v, we say that v is a *child* of u and that u is a *parent* of v if $\lambda(v) = \lambda(u) + 1 \bmod M$; a childless node is referred to as a *leaf*. This induces an orientation on a subset F of the edges, say, from parents to their children (up the incrementing paths), thus introducing a directed graph $\vec{G}$ whose edge set is an oriented version of F (see Figure 1). Lemma 3 guarantees that $\vec{G}$ is acyclic (so, it is a directed acyclic graph, abbreviated *DAG*) and that it spans all nodes in V. Moreover, the sources and sinks of $\vec{G}$ are exactly the roots and leafs of the ball growing subroutine, respectively, and the source-to-sink distances in $\vec{G}$ are upper-bounded by the diameter D of G.

We emphasize that the in-degrees and out-degrees in $\vec{G}$ are unbounded. Nevertheless, the simplifying assumption that the messages sent by the nodes encode their local states, including the level variables $\lambda(\cdot)$ (see Section 2.1), ensures that node v can distinguish between messages received from its children, messages received from its parents, and messages received from nodes that are neither children nor parents of v.

2.3 Broadcast and Echo over $\vec{G}$

The assignment of level variables $\lambda(\cdot)$ by the ball growing subroutine and the child-parent relations these variables induce provide a natural infrastructure for *broadcast and echo (B&E)* over the aforementioned DAG $\vec{G}$ so that the broadcast (resp., echo) process progresses up (resp., down) the incrementing paths. These are implemented based on `Broadcast` and `Echo` messages as follows.

The broadcast subroutine is initiated at (all) the roots, not necessarily simultaneously, through designated signals discussed later on and root r becomes *broadcast ready* upon receiving such a signal. A non-root node v becomes broadcast ready in the first round in which it receives `Broadcast` messages from all its parents. A (root or non-root) node v that becomes broadcast ready in round $t_0^b = t_0^b(v)$ keeps sending `Broadcast` messages throughout the round interval $[t_0^b, t_1^b)$, where $t_1^b = t_1^b(v)$ is defined to be the first round (strictly) after t_0^b in which

(i) v receives `Broadcast` messages from all its children; and
(ii) v does not receive a `Broadcast` message from any of its parents.

(Notice that conditions (i) and (ii) are satisfied vacuously for the leaves and roots, respectively.)

The echo subroutine is implemented in a reversed manner: It is initiated at (all) the leaves, not necessarily simultaneously, after their role in the broadcast subroutine ends so that leaf v becomes *echo ready* in round $t_1^b(v)$. A non-leaf node v becomes *echo ready* in the first round in which it receives `Echo` messages from all its children. A (leaf or non-leaf) node v that becomes echo ready in round $t_0^e = t_0^e(v)$ keeps sending `Echo` messages throughout the round interval $[t_0^e, t_1^e)$, where $t_1^e = t_1^e(v)$ is defined to be the first round (strictly) after t_0^e in which

(i) v receives `Echo` messages from all its parents; and
(ii) v does not receive an `Echo` message from any of its children.

(Notice that conditions (i) and (ii) are satisfied vacuously for the roots and leaves, respectively.)

▶ **Lemma 6.** *The following properties hold for every B&E process:*

- *Rounds $t_0^b(v)$, $t_1^b(v)$, $t_0^e(v)$, and $t_1^e(v)$ exist and $t_0^b(v) < t_1^b(v) \leq t_0^e(v) < t_1^e(v)$ for every node v.*
- *If node v is reachable from node $u \neq v$ in DAG $\vec{G}$, then $t_i^b(u) < t_i^b(v)$ and $t_i^e(u) > t_i^e(v)$ for $i \in \{0, 1\}$.*
- *If t is the latest round in which the process is initiated at some root, then the process terminates by round $t + O(D)$.*

Proof. Follows since $\vec{G}$ is a DAG and all paths in $\vec{G}$ are shortest paths. ◀

Auxiliary Conditions. In the aforementioned implementation of the broadcast (resp., echo) subroutine, being broadcast (resp., echo) ready is both a necessary and sufficient condition for a node to start sending `Broadcast` (resp., `Echo`) messages. In Section 2.4, we describe variants of this subroutine in which being broadcast (resp., echo) ready is a necessary, but not necessarily sufficient, condition and the node starts sending `Broadcast` (resp., `Echo`) messages only after additional conditions, referred to later on as *auxiliary conditions*, are satisfied.

Acknowledged Ball Growing. As presented in Section 2.2, the ball growing subroutine propagates from the roots to the leaves. To ensure that root r is signaled when the construction of its ball $B(r)$ has finished (cf. termination detection), r initiates a B&E process one round after it invokes the ball growing subroutine. The valid operation of this process is guaranteed

since the ball growing process propagates at least as fast as the B&E process. We call the combined subroutine *acknowledged ball growing*.

2.4 The Main Algorithm

Our k-leader selection algorithm consists of two *phases* executed repeatedly in alternation:

- phase 0, a.k.a. the *detection* phase, that detects the existence of multiple candidates whp; and
- phase 1, a.k.a. the *elimination* phase, in which all candidates but one withdraw with probability at least $1/4$.

Starting with a detection phase, the algorithm executes the phases in alternation until the first detection phase that does not detect candidate multiplicity. Each node v maintains a *phase* variable $\phi(v) \in \{0, 1\}$ that indicates v's current phase.

The two phases follow a similar structure: The (surviving) candidates start by initiating an acknowledged ball growing process. Among its other "duties", this ball growing process is responsible for updating the phase variables $\phi(\cdot)$ of the nodes: node v with $\phi(v) = p$ that receives a `GrowBall`$(\cdot)$ message from node u with $\phi(u) = p + 1 \bmod 2$ assigns $\phi(v) \leftarrow p + 1 \bmod 2$. When updating the phase variable $\phi(v)$ to $\phi(v) = p + 1 \bmod 2$, node v ceases to participate in phase p, resetting all phase p variables. Recalling the definition of the ball growing subroutine (see Section 2.2), this means in particular that if a candidate r with $\phi(r) = p$ receives a `GrowBall`$(\cdot)$ message from node u with $\phi(u) = p + 1 \bmod 2$, then r withdraws and subsequently follows the protocol like any other non-root node.

> **Intuition spotlight:** The ball growing process of phase $p + 1 \bmod 2$ essentially "takes control" over the graph and "forcibly" terminates phase p (at nodes where it did not terminate already). We design the algorithm to ensure that at any point in time, there is at most one p value for which there is an ongoing ball growing process in the graph (otherwise, we may get to undesired situations such as all candidates withdrawing).

Upon termination of the acknowledged ball growing process, the roots run $2k$ back-to-back *B&E iterations*, initiating the broadcast process of the next B&E iteration one round after the echo process of the previous B&E iteration terminates (the choice of the parameter $2k$ will become clear soon). Each node v maintains a variable $\iota(v) \in \{0, 1, \ldots, 2k\}$ that stores v's current B&E iteration. This variable is initialized to $\iota(v) \leftarrow 0$ during the acknowledged ball growing process (considered hereafter as B&E iteration 0) and incremented subsequently from $i - 1$ to i when v becomes broadcast ready in B&E iteration i (see Section 2.3). A phase ends when the echo process of B&E iteration $2k$ terminates.

The $\iota(\cdot)$ variables may differ across the graph and to keep the B&E iterations in synchrony, we augment the B&E subroutines with the following auxiliary conditions (see Section 2.3): Node v with $\iota(v) = i$ (i.e., in B&E iteration i) does not start to send `Broadcast` (resp., `Echo`) messages as long as it has a non-child (resp., non-parent) neighbor u with $\iota(u) = i - 1$.[7] We emphasize that this includes neighbors u that are neither children nor parents of v.

For the sake of the next observation, we globally map the B&E iterations to *sequence numbers* so that B&E iterations $0, 1, \ldots, 2k$ of the first phase (which is a detection phase) are mapped to sequence numbers $1, 2, \ldots, 2k + 1$, respectively, B&E iterations $0, 1, \ldots, 2k$ of the second phase (which is an elimination phase) are mapped to sequence numbers $2k + 2, 2k + 3, \ldots, 4k + 2$, respectively, and so on. Let $\sigma(v)$ be a variable (defined only for the sake of the analysis) indicating the sequence number of node v's current B&E iteration.

[7] This can be viewed as imposing the α-synchronizer of [8] on the B&E iterations of the balls.

▶ **Observation 7.** *For every two roots r and r', we have $|\sigma(r) - \sigma(r')| \leq k - 1$.*

We say that round t is 0-*dirty* (resp., 1-*dirty*) if some node v with $\phi(v) = 0$ (resp., $\phi(v) = 1$) sends a `GrowBall`$(\cdot)$ message in round t; the round is said to be *clean* if it is neither 0-dirty nor 1-dirty. Observation 7 implies that if $\phi(r) = p$ and $\iota(r) = k$ for some root r in round t, then $\phi(r') = p$ and $1 \leq \iota(r') \leq 2k - 1$ for any other root r' in round t, hence the ball growing process of this phase has already ended and the ball growing process of the next phase has not yet started.

▶ **Corollary 8.** *Let t_0 and t_1 be some 0-dirty and 1-dirty rounds, respectively. If $t_0 \leq t_1$ (resp., $t_1 \leq t_0$), then there exists some $t_0 < t' < t_1$ (resp., $t_1 < t' < t_0$) such that round t' is clean.*

2.4.1 The Detection Phase

In the detection phase, the nodes test for candidate multiplicity in the graph. If the graph contains a single candidate r, then the algorithm terminates upon completion of this phase and r is declared to be the leader. Otherwise, certain boundary nodes (see Section 2.2) realize whp that multiple balls exist in their neighborhoods and signal the roots that they should proceed to the elimination phase (rather than terminate the algorithm) upon completion of the current detection phase. This signal is carried by `Proceed` messages delivered from the boundary nodes to the roots of their balls down the incrementing paths in conjunction with the `Echo` messages of the (subsequent) B&E iterations.

For the actual candidate multiplicity test, once all nodes in the (inclusive) neighborhood of node v participate in the detection phase, node v checks if it is a locally observable boundary node and triggers a `Proceed` message delivery if it is. As the name implies, this check can be performed (locally) under the $\text{SA}^{\circlearrowleft}$ model assuming that the messages sent by the nodes encode their local states, including the level variables.

Intuition spotlight: Although every locally observable boundary node is a boundary node, not all boundary nodes are locally observable: a node may belong to several different balls or two adjacent nodes with the same level variable may belong to different balls. For this kind of scenarios, randomness is utilized to break symmetry between the candidates and identify (some of) the boundary nodes.

Consider some root r with $\phi(r) = 0$ upon termination of the acknowledged ball growing subroutine and recall that at this stage, r runs $2k$ back-to-back B&E iterations. In each round of these $2k$ B&E iterations, r picks some *symbol* s uniformly at random (and independently of all other random choices) from a sufficiently large (yet constant size) symbol space $\mathcal{S}$ and sends a `RandSymbol`(s) message. This can be viewed as a random symbol *stream* $S_r \in \mathcal{S}^*$ that r generates, round by round, and sends to its children.

The random symbol streams S_r are disseminated throughout $B(r)$ and utilized by the nodes (the boundary nodes in particular) to test for candidate multiplicity. For clarity of the exposition, it is convenient to think of a node v that does not send a `RandSymbol`(s) message, $s \in \mathcal{S}$, as if it sends a `RandSymbol`$(\perp)$ message for the default symbol $\perp \notin \mathcal{S}$. The mechanism in charge of disseminating S_r up the incrementing paths works as follows: If non-root node v with $\phi(v) = 0$ receives `RandSymbol`(s) messages with the same argument s from all its parents at the beginning of round t, then v sends a `RandSymbol`(s) message at the end of round t; in all other cases, v sends a `RandSymbol`$(\perp)$ message.

Throughout this process, each node v verifies that

(1) all `RandSymbol`(s) messages sent by v's parents in round t carry the same argument s; and

(2) any `RandSymbol`(s) message sent by a neighbor u of v with $\lambda(u) = \lambda(v)$ in round t carries the same argument s as in the `RandSymbol`(s) message that v sends in round t (this is checked by v in round $t+1$).

If any of these two conditions does not hold, then v triggers a `Proceed` message delivery. A root that completes all $2k$ B&E iterations in the detection phase without receiving any `Proceed` message terminates the algorithm and declares itself as the leader.

> **Intuition spotlight:** Since the aforementioned random tests should detect candidate multiplicity whp (i.e., with error probability inverse polynomial in n) and since the size of the symbol space $\mathcal{S}$ from which the random symbol streams S_r are generated is bounded, it follows that the length of the random symbol streams must be $|S_r| \geq \Omega(\log n)$. How can we ensure that $|S_r| \geq \Omega(\log n)$ if the nodes cannot count beyond some constant?

To ensure that the random symbol stream S_r is sufficiently long, we augment the echo subroutine invoked during B&E iteration k of the detection phase (out of the $2k$ B&E iterations in this phase) with one additional auxiliary condition referred to as the *geometric auxiliary condition*: Consider some node v with $\phi(v) = 0$ and $\iota(v) = k$ (i.e., in the k-th B&E iteration of the detection phase) and suppose that it becomes echo ready (for B&E iteration k) in round t_0. Then, v tosses a fair coin $c(t) \in_r \{0,1\}$ in each round $t \geq t_0$ until the first round t' for which $c(t') = 1$; node v does not send `Echo` messages until round t'. This completes the description of the detection phase.

▶ **Lemma 9.** *If multiple roots start a detection phase, then all of them receive a* `Proceed` *message before completing their (respective) $2k$ B&E iterations whp.*

> **Intuition spotlight:** The proof's outline is as follows. We use the geometric auxiliary conditions to argue that there exists some root that spends $\Omega(\log n)$ rounds in B&E iteration k whp. Employing Observation 7, we conclude that the random symbol stream generated by every root r is $\Omega(\log n)$-long whp. Conditioned on that, we prove that there exists some boundary node $v \in B(r)$ that triggers a `Proceed` message delivery whp and that the corresponding `Proceed` message is delivered to r before the phase ends.

Proof of Lemma 9. Fix some detection phase. For a root r, let c_r be the number of rounds r spends in B&E iterations $1, 2, \ldots, 2k-1$, that is, the number of rounds in which $1 \leq \iota(r) \leq 2k-1$ (during this detection phase). We first argue that $c_r \geq \Omega(\log n)$ for all roots r whp. To that end, let X_v be the number of rounds in which node v is prevented from sending its `Echo` messages in B&E iteration k due to the geometric auxiliary condition ($t' - t_0$ in the aforementioned notation of the geometric auxiliary condition) and notice that this auxiliary condition is designed so that X_v is a geometric random variable with parameter $1/2$. Therefore,

$$\Pr\left(\bigwedge_{v \in V} X_v < \log(n)/2\right) = \left(1 - 2^{-\log(n)/2}\right)^n = \left(1 - 1/\sqrt{n}\right)^n \leq e^{-\sqrt{n}}.$$

Condition hereafter on the event that $X_{v^*} \geq \log(n)/2$ for some node v^*, namely, v^* is prevented from sending its `Echo` messages (in B&E iteration k) for at least $\log(n)/2 = \Omega(\log n)$ rounds. Let r^* be a root such that $v^* \in B(r^*)$. By the definition of auxiliary

conditions, B&E iteration k of r^* takes at least $\Omega(\log n)$ rounds. Observation 7 guarantees that by the time r^* starts B&E iteration k, every other root must have already started B&E iteration 1 (of this detection phase). Moreover, no root can start B&E iteration $2k$ before r^* finishes B&E iteration k. We conclude that every root r spends at least $\Omega(\log n)$ rounds in B&E iterations $1, 2, \ldots, 2k-1$, thus establishing the argument.

Let Z_r be the prefix of the random symbol stream S_r generated by root r during the first $c_r - 1$ rounds it spends in B&E iterations $1, 2, \ldots, 2k-1$, i.e., during all but the last round of these B&E iterations (the reason for this missing round is explained soon), and let $z_r = |Z_r|$. We have just showed that $z_r = c_r - 1 \geq \Omega(\log n)$ for all roots r whp.

The assertion is established by proving that if multiple roots r exist in the graph and $z_r \geq \Omega(\log n)$ for all of them, then for every root r, there exists some node $v \in B(r)$ that triggers a `Proceed` message delivery while $\iota(v) \leq 2k-1$ whp. Indeed, if the `Proceed` message delivery is triggered by v while $\iota(v) \leq 2k-1$, then a `Proceed` message is delivered to r with the `Echo` messages of B&E iteration $2k$ at the latest, thus r does not terminate the algorithm at the end of this detection phase and by the union bound, this holds simultaneously for all roots r whp.

To that end, recall that node v sends a `RandSymbol`(s) message with some symbol $s \in \mathcal{S} \cup \{\perp\}$ in every round of the detection phase. In the scope of this proof, we say that v *posts* the symbol stream $(s_1, \ldots, s_z)$ in rounds $t_1, \ldots, t_z$ if s_j is the argument of the `RandSymbol`$(\cdot)$ message sent by v in round t_j for every $1 \leq j \leq z$.

Consider some root r and let v be a boundary node in $B(r)$ that minimizes the distance to r. If v is locally observable, then it triggers a `Proceed` message delivery (deterministically) already when $\iota(v) = 0$, so assume hereafter that v is not locally observable. Let Q be an incrementing (r, v)-path and denote the length of Q by q. Taking $\hat{t}$ to be the round in which B&E iteration 1 of r begins, recall that r posts Z_r in rounds $\hat{t}, \hat{t}+1, \ldots, \hat{t}+z_r-1$. The choice of v ensures that all nodes of Q other than v are not boundary nodes, therefore if $q \geq 1$ (i.e., if $v \neq r$), then the node that precede v along Q – denote it by u – posts Z_r in rounds $\hat{t}+q-1, \hat{t}+q, \ldots, \hat{t}+q+z_r-2$. Moreover, by the definition of Z_r, specifically, by the choice of $z_r = c_r - 1$, we know that $0 \leq \iota(v) \leq 2k-1$ (and $\phi(v) = 0$) in all rounds $\hat{t} \leq t \leq \hat{t}+q+z_r$.

If v belongs to multiple balls, which necessarily means that $v \neq r$ and $q \geq 1$ (see Lemma 3), then v has another parent $u' \neq u$ such that $u' \in B(r')$ for some root $r' \neq r$. The probability that u' posts Z_r in rounds $\hat{t}+q-1, \hat{t}+q, \ldots, \hat{t}+q+z_r-2$ is at most $|\mathcal{S}|^{-z_r}$. Otherwise, if v belongs only to ball $B(r)$, then all its parents post Z_r in rounds $\hat{t}+q-1, \hat{t}+q, \ldots, \hat{t}+q+z_r-2$ (this holds vacuously if $q = 0$ and $v = r$ has no parents), thus v posts Z_r in rounds $\hat{t}+q, \hat{t}+q+1, \ldots, \hat{t}+q+z_r-1$. Since v is a non-locally observable boundary node (that belongs exclusively to ball $B(r)$), it must have a neighbor v' with $\lambda(v') = \lambda(v)$ such that $v' \notin B(r)$. The probability that v' posts Z_r in rounds $\hat{t}+q, \hat{t}+q+1, \ldots, \hat{t}+q+z_r-1$ is at most $|\mathcal{S}|^{-z_r}$ as well. Therefore, the probability that v does not trigger a `Proceed` message delivery while $\iota(v) \leq 2k-1$ is upper-bounded by $|\mathcal{S}|^{-z_r}$ which completes the proof since $z_r \geq \Omega(\log n)$ and since $|\mathcal{S}|$ is an arbitrarily large constant. ◀

2.4.2 The Elimination Phase

In the elimination phase, each candidate r picks a *priority* $\pi(r)$ uniformly at random (and independently) from a totally ordered priority space $\mathcal{P}$; a candidate whose priority is (strictly) smaller than $\pi_{\max} = \max_r \pi(r)$ is withdrawn. Taking the priority space to be $\mathcal{P} = \{1, \ldots, k\}$, it follows by standard balls-in-bins arguments that the probability that exactly one candidate picks priority k, which implies that exactly one candidate survives, is at least $1/4$ (in fact, it tends to $1/4$ as $k \to \infty$).

Intuition spotlight: The priorities of the candidates are disseminated in the graph so that candidate r withdraws if it encounters a priority $\pi > \pi(r)$. This is implemented on top of the ball growing subroutine invoked at the beginning of the elimination phase so that the ball growing process of root r "consumes" the ball of root r' if $\pi(r) > \pi(r')$, eventually reaching r' and instructing it to withdraw. The structure of the phase (specifically, the $2k$ B&E iterations that follow the ball growing process) guarantees that only roots r with $\pi(r) = \pi_{\max}$ reach the end of the phase (without being withdrawn).

We augment the ball growing subroutine invoked at the beginning of the elimination phase with the following mechanism: When candidate r is signaled to invoke the ball growing subroutine (so that it becomes a root), it appends its priority $\pi(r)$ to the $\texttt{GrowBall}(\cdot)$ message it sends. A non-root node v that joins the ball of r records r's priority in variable $\pi(v) \leftarrow \pi(r)$. A (root or non-root) node v that receives a $\texttt{GrowBall}(\cdot)$ message with priority (strictly) larger than $\pi(v)$, behaves as if this is the first $\texttt{GrowBall}(\cdot)$ message it receives in this phase. In particular, v resets all the variables of this phase and (re-)joins a ball from scratch. If v is a root, then it also withdraws.

Notice that Observation 7 still holds for the aforementioned augmented implementation of the ball growing subroutine. Therefore, when root r reaches B&E iteration k, i.e., $\iota(r) = k$, all other roots r' are in some B&E iteration $1 \leq \iota(r') \leq 2k - 1$ which means that there is no "active" ball growing processes in the graph, that is, the current round is clean (of $\texttt{GrowBall}(\cdot)$ messages). Since a candidate r with $\pi(r) < \pi_{\max}$ is certain to be withdrawn by some $\texttt{GrowBall}(\cdot)$ message appended with priority $\pi > \pi(r)$, we obtain the following observation.

▶ **Observation 10.** *If root r completes its $2k$ B&E iterations in an elimination phase, then with probability at least $1/4$, no other candidates exist in the graph.*

2.5 Run-Time

The correctness of our algorithm follows from Lemma 9 and Observation 10. To establish Theorem 1, it remains to analyze the algorithm's run-time.

The first thing to notice in this regard is that the geometric auxiliary condition does not slow down the k-th iteration of the detection phase by more than an $O(\log n)$ factor whp. Combining Observation 4 with Lemma 6, we can prove by induction on the phases that the j-th phase (for $j \leq n^{O(1)}$) ends by round $O(D(k + \log n))$ whp, which is $O(D \log n)$ assuming that k is fixed. The analysis is completed due to Observation 10 ensuring that the algorithm terminates after $O(\log n)$ elimination phases whp.

3 Negative Results

We now turn to establish some negative results that demonstrate the necessity of the assumption that $k = O(1)$. Our attention in this section is restricted to SA and $\text{SA}^{\circlearrowleft}$ algorithms operating under a fully synchronous scheduler on graph families $\{L_n\}_{n\geq 1}$ and $\{L_n^{\circlearrowleft}\}_{n\geq 1}$, where L_n is a simple path of n nodes and $L_n^{\circlearrowleft}$ is L_n augmented with self-loops.

The main lemma established in this section considers the *k-candidate binary consensus* problem, a version of the classic binary consensus problem [21]. In this problem, each node v gets a binary input $\text{in}(v) \in \{0, 1\}$ and returns a binary output $\text{out}(v) \in \{0, 1\}$ under the following two constraints: (1) all nodes return the same output; and (2) if the nodes return output $b \in \{0, 1\}$, then there exists some node v such that $\text{in}(v) = b$. In addition, at most k (and at least 1) nodes are initially marked as candidates (thus distinguished from the rest of

the nodes). We emphasize that the marked candidates do not affect the validity of the output. Since a k-leader selection algorithm clearly implies a k-candidate binary consensus algorithm, Theorem 2 is established by proving Lemma 11. Note that the proof of this lemma is based on a probabilistic indistinguishability argument, similar to those used in many distributed computing negative results, starting with the classic result of Itai and Rodeh [26].

▶ **Lemma 11.** *If the upper bound k on the number of candidates may grow as a function of n, then there does not exist a SA algorithm that solves the k-candidate binary consensus problem on the graphs in $\{L_n\}_{n\geq 1}$ with a failure probability bounded away from 1.*

Proof. Assume by contradiction that there exists such an algorithm $\mathcal{A}$ and let Σ denote its message alphabet. For $b = 0, 1$, consider the execution of $\mathcal{A}$ on an instance that consists of path L_2, where node v_1 is a candidate, node v_2 is not a candidate, and $\mathrm{in}(v_1) = \mathrm{in}(v_2) = b$. By definition, there exist constants $p_b > 0$ and ℓ_b and message sequences $S_{b,1}, S_{b,2} \in \Sigma^{\ell_b}$ such that when $\mathcal{A}$ runs on this instance, with probability at least p_b, node v_j, $j \in \{1, 2\}$, reads message $S_{b,j}(t)$ in its (single) port in round $t = 1, \ldots, \ell_b$ and outputs $\mathrm{out}(v_j) = b$ at the end of round ℓ_b.

Now, consider graph L_n for some sufficiently large n (whose value will be determined later on) and consider a subgraph of L_n, referred to as a *Q_b-gadget*, that consists of $2\ell_b + 2$ contiguous nodes $v_1, \ldots, v_{2\ell_b+2}$ of the underlying path L_n, all of which receive input $\mathrm{in}(v_i) = b$. Moreover, the nodes $v_1, \ldots, v_{2\ell_b+2}$ are marked as candidates in an alternating fashion so that if v_i is a candidate, then v_{i+1} is not a candidate, constrained by the requirement that v_{ℓ_b+1} is a candidate (and v_{ℓ_b+2} is not). The key observation is that when $\mathcal{A}$ runs on L_n, with probability at least $q_b = p_b^{2\ell_b+2}$, the nodes v_{ℓ_b+1} and v_{ℓ_b+2} of the Q_b-gadget read messages $S_{b,1}(t)$ and $S_{b,2}(t)$, respectively, in (all) their ports in round $t = 1, \ldots, \ell_b$ and output b at the end of round ℓ_b, independently of the random bits of the nodes outside the Q_b-gadget.

Fix $\ell = \ell_0 + \ell_1 + 2$ and define a *Q-gadget* to be a subgraph of L_n that consists of a Q_0-gadget appended to a Q_1-gadget, so, in total, the Q-gadget is a (sub)path that contains $2\ell_0 + 2\ell_1 + 4 = 2\ell$ nodes, ℓ of which are candidates. Following the aforementioned observation, when $\mathcal{A}$ runs on L_n, with probability at least $q = q_0 \cdot q_1$, some nodes in the Q-gadget output 0 and others output 1; we refer to this (clearly invalid) output as a *failure* event of the Q-gadget.

Since p_0, p_1, ℓ_0, and ℓ_1 are constants that depend only on $\mathcal{A}$, $\ell = \ell_0 + \ell_1 + 2$, $q_0 = p_0^{2\ell_0+2}$ and $q_1 = p_1^{2\ell_1+2}$ are also constants that depend only on $\mathcal{A}$, and thus $q = q_0 \cdot q_1$ is also a constant that depends only on $\mathcal{A}$. Take z to be an arbitrarily large constant. If n is sufficiently large, then we can embed $y = \lceil z/q \rceil$ pairwise disjoint Q-gadgets in L_n. Indeed, these Q-gadgets account to a total of $\ell \cdot y$ candidates and recalling that z, q, and ℓ are constants, this number is smaller than $k = k(n)$ for sufficiently large n. When $\mathcal{A}$ runs on L_n, each of these y Q-gadgets fails with probability at least q (independently). Therefore, the probability that all nodes return the same binary output is at most $(1 - q)^y$. The assertion follows since this expression tends to 0 as $y \to \infty$ which is obtained as $z \to \infty$. ◀

The proof of Lemma 11 essentially shows that no SA algorithm can distinguish between L_2 and L_n with a bounded failure probability. When the path is augmented with self-loops, we can use a very similar line of arguments to show that no SA$^{\circlearrowleft}$ algorithm can distinguish between $L_1^{\circlearrowleft}$ and $L_n^{\circlearrowleft}$ with a bounded failure probability. This allows us to establish the following lemma that should be contrasted with the SA MIS algorithm of [20] that works on general topology graphs (with no self-loops) and succeeds with probability 1.

▶ **Lemma 12.** *There does not exist a* SA$^{\circlearrowleft}$ *algorithm that solves the MIS problem on the graphs in $\{L_n^{\circlearrowleft}\}_{n\geq 1}$ with a failure probability bounded away from 1.*

References

1. Karl R. Abrahamson, Andrew Adler, Lisa Higham, and David G. Kirkpatrick. Probabilistic solitude verification on a ring. In *Proceedings of ACM Symposium on Principles of Distributed Computing (PODC)*, pages 161–173, 1986.
2. Yehuda Afek, Noga Alon, Ziv Bar-Joseph, Alejandro Cornejo, Bernhard Haeupler, and Fabian Kuhn. Beeping a maximal independent set. In *Proceedings of International Symposium on Distributed Computing (DISC)*, pages 32–50, 2011.
3. Yehuda Afek and Yossi Matias. Elections in anonymous networks. *Inf. Comput.*, 113(2):312–330, 1994.
4. Dana Angluin. Local and global properties in networks of processors (extended abstract). In *Proceedings of ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 82–93, 1980.
5. Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.
6. James Aspnes and Eric Ruppert. *An Introduction to Population Protocols*, pages 97–120. Springer Berlin Heidelberg, 2009.
7. Hagit Attiya, Marc Snir, and Manfred K. Warmuth. Computing on an anonymous ring. *J. ACM*, 35(4):845–875, 1988.
8. Baruch Awerbuch. Complexity of network synchronization. *J. ACM*, 32(4):804–823, 1985.
9. Baruch Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems. In *Proceedings of ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 230–240, 1987.
10. Sarah Cannon, Joshua J. Daymude, Dana Randall, and Andréa W. Richa. A Markov chain algorithm for compression in self-organizing particle systems. In *Proceedings of ACM Symposium on Principles of Distributed Computing (PODC)*, pages 279–288, 2016.
11. Alejandro Cornejo and Fabian Kuhn. Deploying wireless networks with beeps. In *Proceedings of International Symposium on Distributed Computing (DISC)*, pages 148–162, 2010.
12. Joshua J. Daymude, Zahra Derakhshandeh, Robert Gmyr, Alexandra Porter, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. On the runtime of universal coating for programmable matter. *Natural Computing*, 17(1):81–96, 2018.
13. Zahra Derakhshandeh, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. An algorithmic framework for shape formation problems in self-organizing particle systems. In *Proceedings of International Conference on Nanoscale Computing and Communication (NANOCOM)*, pages 21:1–21:2, 2015.
14. Zahra Derakhshandeh, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. Universal shape formation for programmable matter. In *Proceedings of ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 289–299, 2016.
15. Zahra Derakhshandeh, Robert Gmyr, Andréa W. Richa, Christian Scheideler, Thim Strothmann, and Shimrit Tzur-David. Infinite object coating in the Amoebot model. *CoRR*, abs/1411.2356, 2014. `arXiv:1411.2356`.
16. Zahra Derakhshandeh, Robert Gmyr, Thim Strothmann, Rida Bazzi, Andréa W. Richa, and Christian Scheideler. Leader election and shape formation with self-organizing programmable matter. In *Proceedings of International Conference on DNA Computing and Molecular Programming (DNA)*, pages 117–132, 2015.
17. Shlomi Dolev, Robert Gmyr, Andréa W. Richa, and Christian Scheideler. Ameba-inspired self-organizing particle systems. *CoRR*, abs/1307.4259, 2013. `arXiv:1307.4259`.
18. David Doty. Timing in chemical reaction networks. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 772–784, 2014.

19 Yuval Emek and Jara Uitto. Dynamic networks of finite state machines. In *Proceedings of International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 19–34, 2016.

20 Yuval Emek and Roger Wattenhofer. Stone age distributed computing. In *Proceedings of ACM Symposium on Principles of Distributed Computing (PODC)*, pages 137–146, 2013.

21 Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

22 Greg N. Frederickson and Nancy A. Lynch. Electing a leader in a synchronous ring. *J. ACM*, 34(1):98–115, 1987.

23 Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5(1):66–77, 1983.

24 M. Gardner. The fantastic combinations of John Conway's new solitaire game 'life'. *Scientific American*, 223(4):120–123, 1970.

25 Lauri Hella, Matti Järvisalo, Antti Kuusisto, Juhana Laurinharju, Tuomo Lempiäinen, Kerkko Luosto, Jukka Suomela, and Jonni Virtema. Weak models of distributed computing, with connections to modal logic. *Distributed Computing*, 28(1):31–53, 2015.

26 Alon Itai and Michael Rodeh. Symmetry breaking in distributed networks. *Inf. Comput.*, 88(1):60–87, 1990.

27 Laurent Keller and Peter Nonacs. The role of queen pheromones in social insects: queen control or queen signal? *Animal Behaviour*, 45(4):787–794, 1993.

28 Jennie J. Kuzdzal-Fick, David C. Queller, and Joan E. Strassmann. An invitation to die: initiators of sociality in a social amoeba become selfish spores. *Biology letters*, 6(6):800–802, 2010.

29 Ivan Lavallée and Christian Lavault. Spanning tree construction for nameless networks. In *Proceedings of International Workshop on Distributed Algorithms (WDAG)*, pages 41–56, 1990.

30 Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. *New Models for Population Protocols*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2011.

31 John Von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA, 1966.

32 David Peleg. *Distributed Computing: A Locality-sensitive Approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

33 Baruch Schieber and Marc Snir. Calling names on nameless networks. *Inf. Comput.*, 113(1):80–101, 1994.

34 Joanna M. Setchell, Marie Charpentier, and E. Jean Wickings. Mate guarding and paternity in mandrills: factors influencing alpha male monopoly. *Animal Behaviour*, 70(5):1105–1120, 2005.

35 Jonathan M.W. Slack. *Essential developmental biology*. John Wiley & Sons, 2009.

36 NSF workshop on self-organizing particle systems (SOPS). `http://sops2014.cs.upb.de/`, 2014.

37 Stephen Wolfram. *A New Kind of Science*. Wolfram Media Inc., Champaign, Ilinois, US, United States, 2002.