The Entropy of a Distributed Computing Schedule - the Example of Population Protocols

Joffroy Beauquier, Peva Blanchard, Janna Burman, Rachid Guerraoui

The Entropy of a Distributed Computing Schedule

The Example of Population Protocols

J. Beauquier[1], P. Blanchard[2⋆], J. Burman[1], and R. Guerraoui[2]

[1] LRI, Paris-South 11 University, Orsay, France, {joffroy.beauquier,janna.burman}@lri.fr
[2] LPD, EPFL, Lausanne, Switzerland, {peva.blanchard, rachid.guerraoui}@epfl.ch

Abstract. A distributed computing system can be viewed as the result of the interplay between a distributed algorithm specifying the effects of a local event (e.g. reception of a message), and an *adversary* choosing the interleaving (schedule) of these events in the execution. For some problems, assuming that the adversary selects the schedule according to some probability distribution greatly helps to devise (almost) correct solutions. But how much randomness is really necessary? To what extent does a problem admit implementations that are robust against a "not so random" schedule?

This paper takes a first step in addressing this question by borrowing the concept of T-*randomness*, $0 \leq T \leq 1$, from algorithmic information theory. Roughly speaking, the value T fixes the *entropy rate* of the considered schedules. For instance, the case $T = 1$ corresponds, in a specific sense, to schedules in which events are independent and sampled uniformly (perfect randomness). The holy grail question can then be precisely stated as determining the *optimal entropy rate* to solve a given problem.

We first show that perfect randomness is never required. Precisely, if a finite-state algorithm solves a problem with 1-randomness, then this algorithm still solves the same problem with T-randomness for some $T < 1$. Second, we illustrate how to compute bounds on the optimal entropy rate of a specific problem, namely the *leader election* problem.

Keywords: algorithmic randomness, entropy, leader election, distributed computing, scheduler

1 Introduction

The way that events in a distributed system are triggered depends on some external causes, often referred to as the environment. To model the environment, an abstraction, called scheduler, is introduced. The scheduler specifies which sequences of events are possible and which ones are impossible. As the correctness of a distributed algorithm depends both on the algorithm and on the scheduler, this latter is often considered as an adversary. In this context, one can think of the scheduler as trying to trigger a sequence of events that will fool the algorithm. Most of the impossibility proofs rely on exhibiting a particular schedule for which the specification of the problem is not satisfied. One way to circumvent such impossibility proofs is to assume that the adversary selects a *random* schedule. This assumption is generally not in contradiction with real environments, which endure phenomena like variations of temperature, power supply, network traffic, etc., in sort of a randomized way.

⋆ Corresponding author. Acknowledgment: This work has been supported in part by the European ERC Grant 339539 - AOC. Contact: EPFL IC IIF LPD, INR 210 (Bâtiment INR), Station 14, CH-1015 Lausanne; phone: +41 21 69 35267

However, it is not as obvious as it may seem at first sight that a given environment yields truly random schedules. Actually, truly random sequences are very hard to get. For instance, the scheduling of processes in a multi-core architecture depends on a physical process which may exhibit a partially predictable behaviour. Or, this scheduler may rely on some algorithm using a pseudo-random source, which is not truly random. In other settings, like mobile networks, the interactions between nodes may follow some regularity because, e.g., the mobility area is limited, or some paths are statistically preferred to others, etc.

This raises the following interesting question: to what extent an algorithm may exhibit some robustness against *imperfect* randomness? And, for a specific problem, what is the optimal robustness that one may hope to achieve? The main goal of this paper is to tackle these important questions.

A first step towards this goal is to lay down a definition of randomness, and a measure of robustness, which are amenable to analysis. Randomness and probability theory are obviously strongly related, but here probability theory does not help because it does not allow to qualify an individual schedule as being random. In this context, probability theory is more about measuring the number of, say, "bad" schedules, which leads to notions of solving a problem almost surely, or with high probability.

Algorithmic information theory, on the other hand, allows to define what it means for an individual schedule to be random. Intuitively, the complexity of a finite schedule is defined as the length of the shortest computer program able to produce this schedule; and the schedule is random if no program is substantially shorter than the sequence itself[3]. This intuition extends to infinite schedule by considering how the complexity of its prefixes grow. More precisely, we borrow the concept of T-randomness, $0 \leq T \leq 1$, from [22, 23], where T measures the *entropy rate* of a schedule, i.e., the complexity growth of the schedule's prefixes. Roughly speaking, the larger is T, the more random is the schedule. In particular, the case of $T = 1$ represents perfect randomness.

This notion allows to precisely quantify the robustness of an algorithm against imperfect randomness: this is simply the least entropy rate T such that any T-random schedule induces an execution that still satisfies the problem's specification. In this context, a natural issue is to determine, given a problem P, the optimal entropy rate T such that some algorithm solves P for all T-random schedules.

We illustrate this notion in a simple distributed computing model (population protocols [5]). Our contributions are twofold. First, we show that perfect randomness is never required in finite-state systems (Section 4). That is, whenever a finite-state algorithm solves a problem P over the 1-random schedules, then this algorithm solves the same problem P over the T-random schedules for some $T < 1$. That is, the optimal entropy rate of P is strictly less than 1. Moreover, our proof exhibits a general method to compute upper bounds on the optimal entropy rate.

Our second contribution focuses on a specific problem, fundamental in distributed computing, namely *leader election* (Section 5). In this problem, all processes start in the same initial state (with the same initial knowledge), and a unique process (the elected leader) eventually permanently outputs 1 while the others output 0. The method exhibited

[3] Roughly speaking, the shortest computer program just enumerates the successive events of the schedule.

in the previous part (Section 4) is applied to derive an explicit upper bound on the optimal entropy rate of leader election. Next, we compute a lower bound T_{sym}. This bound exploits the relation between leader election and symmetry breaking. Indeed, some schedulers are able to produce schedules which "maintain symmetry", in the sense that any process have the same state as some other process; thereby preventing the election of a unique leader. The bound T_{sym} quantifies exactly the maximum entropy rate above which a schedule cannot be symmetric in the previous sense.

The rest of the paper is organized as follows. Section 2 recalls basic definitions of algorithmic information theory, as well as the distributed computing model we consider. In Section 3, we introduce the notions of T-random adversary and optimal entropy rate of a distributed computing problem. Our first contribution, i.e., proving that perfect randomness is never required, is presented in Section 4. We give upper and lower bounds for the entropy rate of leader election in Section 5.

For the sake of clarity, some proofs are postponed to the appendix.

Related Work. Algorithmic information theory has started with the seminal work of Solomonoff [19, 20] and Kolmogorov [15]. One of the major achievements of this field was a precise definition of randomness. In [17], Martin-Löf defines random sequences as those which withstand all effective statistical tests. In [10], an equivalent formulation of a random sequence is given, as one whose shortest (prefix-free) program has the same length as the schedule. In [22, 23], Tadaki generalizes this formulation, and introduces the notion of partial randomness, namely T-randomness ($0 \leq T \leq 1$). The original Martin-Löf's definition of randomness then coincides with 1-randomness.

The computational model used in this paper, known as population protocol, has been introduced by Angluin *et al.* in [5] to model large wireless networks of anonymous finite-state mobile processes interacting pairwise. In much of the literature on this model [4,5,6], the scheduler is subject to a fairness condition, namely global fairness, which depends on the algorithm being run: if a configuration is reachable infinitely often, then this configuration is reached infinitely often. Actually, the proof in Section 4 (indirectly) shows that 1-randomness, which is a condition independent of the algorithm being run, implies global fairness for any finite-state algorithm. In particular, any algorithm working under global fairness is guaranteed to work under 1-randomness.

Being an important primitive in distributed computing, leader election has been extensively studied. In particular, in [3], Angluin relates the notion of graph coverings with the impossibility of leader election, and more generally, to any problem that requires "symmetry breaking". This approach has been proved to be fruitful in other contexts as well [11,18]. To circumvent this issue, many approaches [1,12,14] have proposed randomized algorithms, each process having access to some random sequence. These algorithms are probabilistic and solve the problem at hand, either almost surely, or with high probability, but not exactly. Our approach is orthogonal. We consider deterministic algorithms, the randomness being put entirely on the side of the adversary. In some sense, our approach is closer to the works [2, 7], where random scheduling helps solving the problem at hand; except that our motivation is to assess the amount of randomness required. In [8], thanks to the almost random nature of global fairness, authors propose a deterministic algorithm solving leader election over arbitrary graphs.

2 Background Definitions

Let X be a finite set, $|X|$ its cardinality, and X^* (resp. X^ω) the set of finite (resp. infinite) sequences on X. The length of a finite sequence $w \in X^*$ is denoted by $|w|$. For any $w \in X^*$, we denote by $w \upharpoonright n$ the prefix of w of length n. The concatenation of two sequences $u, v \in X^*$ is denoted by uv. We denote by u^k the concatenation of k copies of u.

2.1 Algorithmic Information Theory

Here, we briefly recall the basic definitions and properties. For further details, please refer to, e.g., [9, 16, 21, 24]. Our presentation is slightly different from the usual one as we will be dealing with several alphabets.

Consider finite alphabets X, Y. A *(partial) recursive function* $M : X^* \to Y^*$ is a function computed by some Turing machine (with input alphabet X, and output alphabet Y). The function M is *prefix-free* if its domain does not contain two elements, one of which being a prefix of the other. It is known that there exists a *universal prefix-free recursive function* $U_{XY} : X^* \to Y^*$. Intuitively, this function is universal in the sense that it can simulate any other prefix-free recursive function (of type $X^* \to Y^*$).[4] We fix such a universal function U_{XY}.

The following defines the complexity of a finite word $S \in X^*$ as the length (in bits) of the smallest word p (with the same alphabet) such that the universal function $U_{XX}(p)$ produces S. Intuitively, the word p is the *program* which computes S when executed on the machine U.

Definition 1 (Complexity). *The complexity of a sequence $S \in X^*$ is $H_X(S) = |p| \log |X|$ where $p \in X^*$ is the shortest sequence such that $U_{XX}(p) = S$.*

The factor $\log |X|$ is simply a rescaling factor, so that the complexity is expressed in bits.

We now consider infinite sequences, i.e., elements of X^ω. The notion of partial randomness from [22] allows to quantify the degree of randomness of an infinite sequence by looking at how the complexity of its prefixes grows. We adapt Tadaki's definition to the case of an arbitrary alphabet.

Definition 2 (T-randomness). *Given a real value $T \in [0, 1]$, $S \in X^\omega$ is* T-random *on X if, for all n, $H_X(S \upharpoonright n) \geq T \cdot n \cdot \log |X| - O(1)$.*

All the entropy-related computations in the paper rely solely on the following lemmas. These are adaptations of known results of algorithmic information theory [9, 16]. For the reader's convenience, we give an intuitive interpretation of Lemma 1 in the appendix (Section A).

Lemma 1. *a. Let $q : X^* \to Y^*$ be a partial recursive function. Then, for all $S \in X^*$, $H_Y(q(S)) \leq H_X(S) + O(1)$.*
b. For all $S \in X^*$, $H_X(S) \leq |S| \cdot \log |X| + 2 \cdot \log |S| + O(1)$.

[4] Formally, there exists an effective enumeration $(M_i)_{i \in X^*}$ of all the prefix-free recursive function (type $X^* \to Y^*$) such that, for all $i, p \in X^*$, $U_{XY}\langle i, p\rangle = M_i(p)$.

c. *Let $A \subseteq X^* \times \mathbb{N}$ be a recursively enumerable set such that the subset $A_n = \{w : (w, n) \in A\}$ is finite for every $n \in \mathbb{N}$. Then, for all n, for every $S \in A_n$ with $|S| = n$, $H_X(S) \leq \log |A_n| + 4 \cdot \log n + O(1)$.*
d. *An infinite sequence $S \in X^\omega$ is T-random if and only if any of its suffixes is T-random.*

Lemma 2 (Existence of a T-random sequence). *For any $0 \leq T \leq 1$, there exists an infinite sequence $S \in X^\omega$ which is T-random on X.*

2.2 Computational Model

The computational model used in this paper was introduced in [5]. The model involves two parts: a graph representing the possibilities of interactions between the processes, and a list of rules (the algorithm) describing how the states of two interacting processes are updated.

Formally, a *communication graph* is a directed graph G (without self-loops). Each node x represents a *process*. Each directed edge (x, y) represents a possible meeting event in which x is the initiator and y is the responder. We denote by $\mathcal{V}(G)$ and $\mathcal{E}(G)$ the set of processes (vertices) and meeting events (edges) of G respectively. In this work, unless stated otherwise, every graph is assumed to be weakly connected. A *schedule* on G is a sequence of edges of G, that is, a finite or infinite sequence on the alphabet $\mathcal{E}(G)$. An *assignment on G* is a map that associates with every vertex in G some value (in some given set). A *trace on G* is a sequence of assignments on G.

An *algorithm* $\mathcal{A}$ is a tuple $(Q, X \xrightarrow{I} Q, Q \xrightarrow{O} Y, Q^2 \xrightarrow{\delta} Q^2)$ where Q is a set called the state space, $X \xrightarrow{I} Q$ is the *input function*, $Q \xrightarrow{O} Y$ is the *output function*, and $Q^2 \xrightarrow{\delta} Q^2$ is the *transition function*. Note that we consider only *deterministic* algorithms (the transition function is single valued).

An *input assignment* (resp. *output assignment*) is an assignment with values in X (resp. Y). A *configuration* is an assignment with values in the state space Q. Each input assignment α yields an *initial configuration* $I \circ \alpha$. Similarly, each configuration γ yields an output assignment $O \circ \gamma$.

Given an edge $e = (x, y)$ in G and two configurations γ, γ', we write $\gamma \xrightarrow{e} \gamma'$ when $(\gamma'(x), \gamma'(y)) = \delta(\gamma(x), \gamma(y))$ and, for all $z \notin \{x, y\}$, $\gamma(z) = \gamma'(z)$. An (finite or infinite) execution of $\mathcal{A}$ on G is a sequence $\gamma_0 \xrightarrow{e_1} \gamma_1 \dots$ where γ_0 is the initial configuration. The sequence of edges of G appearing in the execution is the underlying schedule of the execution. Note that an execution is entirely determined by its underlying schedule and the initial configuration. It is assumed that every edge occurs infinitely often in the schedule. The *output trace* of an execution $\gamma_0 \xrightarrow{e_1} \gamma_1 \dots$ is the corresponding sequence $(O \circ \gamma_0)(O \circ \gamma_1) \dots$ of output assignments.

3 Entropy of Schedules

Since schedules are infinite sequences on the alphabet $\mathcal{E}(G)$, we can apply the concept of T-randomness to classify them. This motivates the following definition.

Definition 3 (Adversary $\mathbb{A}(T,G)$). *Given a real value $T \in [0,1]$, and a communication graph G, the* T-random adversary $\mathbb{A}(T,G)$ *is the set of infinite schedules $S \in \mathcal{E}(G)^\omega$ such that S is T-random on $\mathcal{E}(G)$.*

For the sake of simplicity, schedules of $\mathbb{A}(T,G)$ are simply said to be T-random on G. We denote by $H_G(\cdot)$ the quantity $H_{\mathcal{E}(G)}(\cdot)$.

A *decision problem* on G is a tuple (P,X,Y) where X is the input alphabet, Y the output alphabet, and P is a function that associates with every input assignment α (with values in X) on G, a set $P(\alpha)$ of output assignments (with values in Y) on G. The input and output alphabets are often implicitly assumed, and we refer to P as the problem directly.

An algorithm $\mathcal{A}$ is said to *solve problem P on G under adversary* $\mathbb{A}(T,G)$ if, for every schedule $S \in \mathbb{A}(T,G)$, for every input assignment α, the execution induced by S and α yields an output trace having a suffix $\beta\beta\ldots$ where $\beta \in P(\alpha)$. Intuitively, this means that the output of the algorithm eventually stabilizes to a legal output assignment, given the input assignment the execution started with.

Definition 4 (Optimal entropy rate). *The* optimal entropy rate of P on G *is* $T(P,G) = \inf\{T : \text{some algorithm solves } P \text{ on } G \text{ with adversary } \mathbb{A}(T,G)\}$. *If the problem is impossible to solve with any $\mathbb{A}(T,G)$, we set $T(P,G) = \infty$.*

Remark 1. Note that we classify the schedules according to their entropy rate only. Hence, this classification applies to a very broad spectrum of adversary schedulers, from, e.g., typical schedules of a uniform bernoullian scheduler, to, e.g., those of a possibly non-markovian one. If an algorithm is proven to solve a problem for all T-random schedules, then it does not matter how exactly the real schedules are produced: as long as they are all T-random, the algorithm will work.

4 Perfect Randomness is Never Required

Proposition 1. *Let P be a problem, and G a graph. If there exists a protocol $\mathcal{A}$ with finite state space solving problem P under adversary $\mathbb{A}(1,G)$, then there exists $0 \leq T < 1$ such that $\mathcal{A}$ solves P under $\mathbb{A}(T,G)$. In particular, the optimal entropy rate of P on G is strictly less than* 1.

The main idea consists in the analysis of the *transition graph* of the protocol, whose nodes represent the configurations, and the (directed) edges, the transitions between configurations. The transition graph can be partitioned into strongly connected components. The final components, i.e., the components with no out-going edges play a particular role. Indeed, a 1-random schedule necessarily drives the system towards a final component, and makes it visit every configuration in that component infinitely often. Thus, by the assumption on the protocol, the configurations in this component produce the same output assignment satisfying the problem's specification. In particular, it suffices to drive the system into one final component to yield an execution satisfying the problem's specification.

The proof below relies on the observation that, if an execution is stuck into a non-final component C, then its underlying schedule repeatedly avoids some pattern of events which

would drive the system out of this component. This repeated dodge imposes an upper bound $t(C) < 1$ on the underlying schedule. Taking the contrapositive, if the schedule were T-random with $t(C) < T < 1$, then the corresponding execution would escape the component C. Since the protocol is finite-state, there are finitely many non-final components, and it suffices to take $\max_C t(C) < T < 1$ for any T-random schedule to escape any non-final component. We now give the detailed proof.

Proof. (Basics). We define the transition graph $\Gamma = \Gamma(\mathcal{A})$ as the edge-labeled directed graph whose nodes are the configurations reachable from the initial configurations, and the edges denote the transition $\gamma \xrightarrow{e} \gamma'$ where e is an event (an edge of G). The underlying schedule of a finite path in Γ is the sequence of successive labels (edges of G) along the path. Since $\mathcal{A}$ has a finite state space, the graph Γ has finitely many nodes.

We can decompose Γ in strongly connected components. A *final* component is a component without any out-going transitions. Given a component C in Γ, we define for each natural number n, the set $C[n]$ of underlying schedules of paths of length n in C. Then, letting d be the number of edges in G, we define

$$T_0 = \max\{\liminf_{n\to\infty} \frac{\log |C[n]|}{n \log d}, \quad \text{non-final component } C\}$$

($T_0 < 1$). We prove that $T_0 < 1$. Consider any non-final component C. We build a finite schedule u such that applying u to any configuration in C leads outside of C. Because it is non-final, there exists configuration $\gamma_0 \in C$, and an event u_0 such that the transition $\gamma_0 \xrightarrow{u_0} \gamma_0' \notin C$. We consider the set D_0 of configurations such that applying u_0 to any of them leads outside of C. If D_0 comprise all the configurations of C, then we are done. Otherwise, pick any configuration $\gamma_1 \in C - D_0$; then $\gamma_1 \xrightarrow{u_0} \gamma_1' \in C$. Since C is strongly connected, there exists a finite schedule w_1 such that applying w_1 to γ_1' leads to some configuration in D_0. Therefore, applying the schedule $u_1 = u_0 w_1 u_0$ to any configuration in $D_0 \cup \{\gamma_1\}$ leads outside of C. We can consider the set D_1 of configurations in C such that applying the schedule u_1 to any of them leads outside of C. If D_1 comprises all the configurations of C, then we are done. Otherwise, we can repeat the same procedure. Because C has finitely many configurations, this process eventually ends: applying the constructed schedule u to any configuration of C leads outside of C.

Therefore, for all n sufficiently large, $C[n]$ is included in the set of finite schedules of length n which do not contain u as a factor. In [13], the authors describe the asymptotic behaviour of the number of finite words of length n not containing a given word. Their results imply that there exists a constant $k > 0$, and a real value $1.7 < \theta < d$ (depending on u only) such that for all n

$$|C[n]| \leq k \cdot \theta^n + O((1.7)^n)$$

In particular

$$\liminf_{n\to\infty} \frac{\log |C[n]|}{n \cdot \log d} \leq \frac{\log \theta}{\log d} < 1$$

Since there are finitely many non-final components, we have $T_0 < 1$.

(Final components reachability). Consider a T-random schedule S on G, and an input assignment α. Assume that the corresponding execution never reaches a final component

of Γ. Then a suffix of this execution remains in some non-final component C forever. In particular, the underlying schedule S' of this suffix is such that, for all n, the prefix $S' \restriction n$ belongs to $C[n]$. By Lemma 1,

$$H_G(S' \restriction n) \leq \log |C[n]| + 4 \log n + O(1)$$

But S' is also T-random, which implies that

$$T \cdot n \log d \leq \log |C[n]| + 4 \log n + O(1)$$
$$T \leq \frac{\log |C[n]|}{n \log d} + O\left(\frac{\log n}{n}\right)$$

Taking the inferior limit, we get $T \leq T_0$. Therefore, for every $T_0 < T \leq 1$, every execution whose underlying schedule is T-random reaches a final component.

(Output is constant in any final component). Let α be an input assignment, and F any final component reachable from the initial configuration γ_α corresponding to α. We claim that the output assignments yielded by the configurations in F are all equal to some $\beta \in P(\alpha)$. Let S_0 be any finite schedule leading to F when applied to γ_α, and S be any 1-random extension of S_0 (it suffices to append a 1-random schedule, which exists by Lemma 2). Let E be the execution with schedule S starting with γ_α. The execution E eventually reaches the final component F and remains in there forever. Since $\mathcal{A}$ solves P under $\mathbb{A}(1, G)$, the output assignments associated with the configurations in F which are visited infinitely often during E are all equal to some output assignment $\beta \in P(\alpha)$. Since F is strongly connected, if the execution E did not visit all the configurations in F, then, by an argument similar to the second paragraph above, the schedule S could not be 1-random. Therefore, all the configurations in F are visited infinitely often during E, and they all yield the same output assignments $\beta \in P(\alpha)$.

(Conclusion). Pick any $T_0 < T < 1$. Consider any input assignment α, and any T-random schedule S. The corresponding execution E reaches a final component F and remains in there forever. Moreover, the output assignments associated with the configurations in F are all equal to some $\beta \in P(\alpha)$. This implies that the execution E yields an output trace which is eventually constant and equal to $\beta \in P(\alpha)$. In other words, the algorithm $\mathcal{A}$ solves P under adversary $\mathbb{A}(T, G)$ with $T < 1$. □

5 Entropy Bounds for Leader Election

The previous section addressed the issue of randomness in a general setting. Now that we know that full randomness is not needed, it is natural to ask for the optimal entropy rate of a problem. We tackle this issue in this section, by considering the *leader election* problem.

Its specifications are the following. There is no input, and all the processes start in the same initial state and with the same initial knowledge. In particular, they do not have identifiers. The output of each process is 0 or 1. The goal is to eventually have a unique process permanently outputting 1 while the others permanently output 0.

We are interested in computing upper and lower bounds on the optimal entropy rate for solving leader election. Actually, the approach taken in the proof of Proposition 1 already leads to an upper bound.

Proposition 2 (LE - upper bound). *For any strongly connected graph G with $d = |\mathcal{E}(G)|$ edges*

$$T(LE, G) \leq \frac{\log \theta(G)}{\log d} < 1$$

where $\theta(G)$ is the absolute value of the largest zero of the polynomial $1+(z-d)(z^{2K(G)-1}+z^{K(G)-1})$, and $K(G)$ is the length of the shortest loop visiting each vertex at least once in G.

We postpone the details to the appendix (Section B). The proof relies on an analysis an algorithm from [8]. Roughly speaking, we exhibit a specific schedule S_0 of length $2K(G)$ which drives the system out of any non-final component of the transition graph. Then, we show that, whenever $T > \log \theta(G)/\log d$, any T-random schedule contains infinitely many occurrences of S_0.

As for a lower bound, the method presented in the proof Proposition 1 cannot be applied. In the sequel, we adopt another approach for deriving a lower bound.

5.1 Lower Bound

Our approach here exploits the fact that leader election requires symmetry breaking. If the communication graph G has local symmetries, then one can design schedules that maintain the symmetry of the system, as shown below. We prove that these schedules have an entropy rate, at most, $T_{sym}(G)$; thereby exhibiting a lower bound on $T(LE, G)$.

Formally, the local symmetries of a graph G are measured by coverings. A *covering* is a surjective graph morphism $\phi : G \to B$ such that, for every vertex $b \in \mathcal{V}(B)$, for every neighbor c of b, for every vertex $x \in \phi^{-1}(b)$ (the *fiber over* b), there exists a unique neighbor y of x in the fiber over c. This concept measures the local symmetries of G in the sense that all vertices in the same fiber have isomorphic neighborhoods. It can be shown that every fiber $\phi^{-1}(b)$ has the same cardinality Δ_ϕ, and that $|\mathcal{E}(G)| = \Delta_\phi \cdot |\mathcal{E}(B)|$. The number Δ_ϕ is the *degree of the covering*. The covering is *proper* if $\Delta_\phi \geq 2$, i.e., ϕ is not an isomorphism.

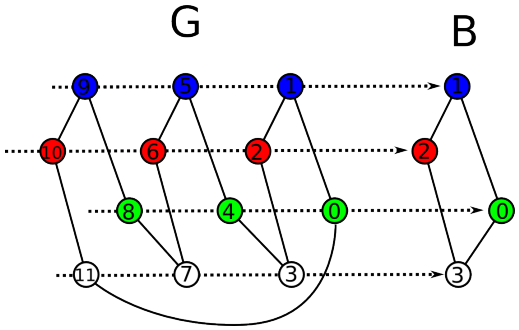

Fig. 1. Graph covering: the ring G of 12 vertices is projected onto the ring B of 4 vertices, the dashed lines indicate the fibers, the degree is $12/4 = 3$.

Proposition 3 (LE - Lower bound). *For any graph G with $d = |\mathcal{E}(G)|$ edges*

$$T_{sym}(G) \underset{def}{=} \max_{\phi} \frac{\log(d/\Delta_\phi) + \log \Delta_\phi!}{\Delta_\phi \log d} \leq T(LE, G)$$

where ϕ runs over all the proper coverings from G. If there are no proper coverings, $T_{sym}(G)$ is set to zero.

To illustrate the proof's basic idea, fix any algorithm $\mathcal{A}$. Let's say that a configuration on G is symmetric if, any two processes in the same fiber (the same color in Figure 1) have the same state. Being uniform, the initial configuration is obviously symmetric. We describe a possible strategy to obtain symmetric configurations infinitely often. The adversary first selects an edge b of B, picks any enumeration of the fiber $\phi^{-1}(b)$ (the edges in G projecting to b), triggers the events according to the enumeration order, and repeats this operation. Since the algorithm $\mathcal{A}$ is deterministic, applying such a sequence of events to a symmetric configuration yields infinitely many symmetric configurations. The successive choices of the adversary are equivalently described by a sequence $Z = (b_1, \sigma_1)(b_2, \sigma_2)\dots$ where b_i is an edge of B, and σ_i is an ordering of the fiber $\phi^{-1}(b_i)$, i.e., an element of the permutation group $\mathfrak{S}_\Delta$ on $\{1, \dots, \Delta\}$. To maximize randomness, we assume that Z is 1-random on the alphabet $\mathcal{E}(B) \times \mathfrak{S}_\Delta$. This allows to compute the entropy rate $T_{sym}(G)$ of the schedule produced by the adversary given Z. Since this schedule prevents the election of a leader, $T_{sym}(G)$ is a lower bound of the optimal entropy rate $T(LE, G)$. Now, we present the full proof.

Proof (LE - Lower bound). Pick any proper covering $\phi : G \to B$ with degree Δ. We have $d = \Delta \cdot r$ where $r = |\mathcal{B}|$. For each edge b in B, we fix a reference enumeration of the fiber over b, $\phi^{-1}(b) = \{e_1(b), \dots, e_\Delta(b)\}$. We define $X = \mathcal{E}(B) \times \mathfrak{S}_\Delta$ where $\mathfrak{S}_\Delta$ is the group of permutations on $\{1, \dots, \Delta\}$. For each element $(b, \sigma) \in X$, we define the sequence $\psi(b, \sigma) = e_{\sigma(1)}(b) \dots e_{\sigma(\Delta)}(b)$ of edges in G; $\psi(b, \sigma)$ is simply an enumeration of the fiber $\phi^{-1}(b)$ ordered according to σ. Note that the map ψ is injective.

Let $\mathcal{A}$ be any (deterministic) algorithm. Assume that γ is a symmetric configuration in the sense that for every vertex $z \in B$, for any two vertices $x, y \in \phi^{-1}(z)$, $\gamma(x) = \gamma(y)$. Then for any element $\mu \in X$, applying the schedule $\psi(\mu)$ to γ yields a configuration γ' that is also symmetric.

Let $Z = \mu_1\mu_2\dots$ be a 1-random sequence on X (which exists by Lemma 2). By definition, for all m, $H_X(Z \upharpoonright m) \geq m \cdot \log(r \cdot \Delta!) - O(1)$. We define the schedule $S = \psi(Z) = \psi(\mu_1)\psi(\mu_2)\dots$. Thanks to the previous remark, for any deterministic algorithm $\mathcal{A}$, since the initial configuration is symmetric (all the processes starts in the same state), the schedule S yields an execution in which infinitely often a symmetric configuration is reached. Hence, this prevents any algorithm to solve the leader election problem with the schedule S. Now, it remains to determine a lower bound on the entropy rate of S. For all n

$$S \upharpoonright n = \psi(\mu_1) \dots \psi(\mu_m) R$$

where $m = \lfloor n/\Delta \rfloor$. Each sequence $\psi(\mu_i)$ has length Δ, and R is strict prefix of $\psi(\mu_{m+1})$. Knowing $S \upharpoonright n$ allows to compute the sequence $Z(\upharpoonright \lfloor n/\Delta \rfloor)$. In other words, there exists a

prefix-free recursive function $q : \mathcal{E}(G)^* \to \mathcal{E}(B)^*$ such that, for all n, $q(S \upharpoonright n) = Z \upharpoonright \lfloor n/\Delta \rfloor$. By Lemma 1, and the fact that Z is 1-random on X, we have, for all n

$$\begin{aligned} H_G(S \upharpoonright n) &\geq H_X(Z \upharpoonright \lfloor n/\Delta \rfloor) - O(1) \\ &\geq \frac{n}{\Delta} \cdot \log(r \cdot \Delta!) - O(1) \\ &\geq \underbrace{\frac{\log r + \log \Delta!}{\Delta \cdot \log d}}_{T(\phi)} \cdot n \cdot \log d - O(1) \end{aligned}$$

Therefore, S is $T(\phi)$-random; whence $T(\phi) \leq T(LE, G)$. □

6 Conclusion

We have shown that, once a problem can be solved by some finite-state algorithm for perfectly random (1-random) schedules, then the problem's optimal entropy rate is strictly less than 1. Doing so, we have exhibited a general method for computing upper bounds on a problem's optimal entropy rate. Next, we focused on the leader election problem. By refining the method above, we have computed an upper bound on the optimal entropy rate of leader election. Then, we computed a lower bound T_{sym} which encodes the maximum entropy rate of schedules maintaining symmetry during the execution. Notice that this lower bound also holds for any other problem requiring symmetry breaking like, e.g., enumeration or spanning tree construction.

This work opens many interesting questions. It seems that the bound T_{sym} could be reached by some algorithm with unbounded memory. The intuition goes as follows. The processes could record the whole history of their interactions, and try to deduce the past schedule. If this schedule breaks symmetry at some point, then it means that the processes have pairwise different "views" of the past. This distinction could be used to distinguish a leader among them. Nevertheless, the required memory may be unbounded, as the scheduler may maintain the symmetry for an arbitrarily long time. This leaves open the question of determining the optimal entropy rate of leader election achievable with finite-state algorithms.

In this work, we have focused on randomness in the scheduling. But one could also study algorithms involving local random coins. Similar questions can be raised, e.g., to what extent randomized algorithms are sensitive to imperfect local coins?

From a more general point of view, we believe that the relation between randomness and hardness of problems is not yet fully understood in the context of distributed computing.

References

1. Y. Afek and Y. Matias. Elections in anonymous networks. *Information and Computation*, 113(2):312–330, 1994.
2. D. Alistarh, K. Censor-Hillel, and N. Shavit. Are lock-free concurrent algorithms practically wait-free? In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 714–723, 2014.
3. D. Angluin. Local and global properties in networks of processors (extended abstract). In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing, April 28-30, 1980, Los Angeles, California, USA*, pages 82–93, 1980.
4. D. Angluin, J. Aspnes, M. Chan, H. Jiang, M. Fischer, and R. Peralta. Stably computable properties of network graphs. In *Distributed Computing in Sensor Systems*, pages 63–74. LNCS 3560, 2005.
5. D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.
6. D. Angluin, J. Aspnes, and D. Eisenstat. Stably computable predicates are semilinear. In *PODC '06: Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 292–299, New York, NY, USA, 2006. ACM Press.
7. J. Aspnes. Fast deterministic consensus in a noisy environment. In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '00, pages 299–308, New York, NY, USA, 2000. ACM.
8. J. Beauquier, P. Blanchard, and J. Burman. Self-stabilizing leader election in population protocols over arbitrary communication graphs. In *Principles of Distributed Systems - 17th International Conference, OPODIS 2013, Nice, France, December 16-18, 2013. Proceedings*, pages 38–52, 2013.
9. C. S. Calude and M. Zimand. Algorithmically independent sequences. In *Developments in Language Theory, 12th International Conference, DLT 2008, Kyoto, Japan, September 16-19, 2008. Proceedings*, pages 183–195, 2008.
10. G. J. Chaitin. A theory of program size formally identical to information theory. *Journal of the ACM*, 22(3):329–340, July 1975.
11. J. Chalopin, Y. Métivier, and T. Morsellino. Enumeration and leader election in partially anonymous and multi-hop broadcast networks. *Fundamenta Informaticae*, 120(1):1–27, 2012.
12. Y. Emek, C. Pfister, J. Seidel, and R. Wattenhofer. Anonymous networks: randomization = 2-hop coloring. In *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, pages 96–105, 2014.
13. L. J. Guibas and A. M. Odlyzko. String overlaps, pattern matching, and nontransitive games. *Journal of Combinatorial Theory, Series A*, 30(2):183–208, 1981.
14. A. Itai and M. Rodeh. Symmetry breaking in distributive networks. In *22nd Annual Symposium on Foundations of Computer Science, Nashville, Tennessee, USA, 28-30 October 1981*, pages 150–158, 1981.
15. A. N. Kolmogorov. Three approaches to the quantitative definition of information. *Problemy Peredachi Informatsii*, 1(1):3 – 31, 1965.
16. M. Li and P. M. B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications, Third Edition*. Texts in Computer Science. Springer, 2008.
17. P. Martin-Löf. The definition of random sequences. *Information and Control*, 9(6):602 – 619, 1966.
18. A. Mazurkiewicz. Distributed enumeration. *Information Processing Letters*, 61(5):233–239, Mar. 1997.
19. R. J. Solomonoff. A formal theory of inductive inference part i. *Information and Control*, 7(1):1 – 22, March 1964.
20. R. J. Solomonoff. A formal theory of inductive inference part ii. *Information and Control*, 7(2):224 – 254, June 1964.
21. L. Staiger. The kolmogorov complexity of infinite words. *Theoretical Computer Science*, 383(2-3):187–199, 2007.
22. K. Tadaki. A generalization of Chaitin's halting probability Ω and halting self-similar sets. *Hokkaido Mathematical Journal*, 31(1):219–253, 02 2002.
23. K. Tadaki. Partial randomness and dimension of recursively enumerable reals. In *Mathematical Foundations of Computer Science*, pages 687–699, 2009.

24. A. K. Zvonkin and L. A. Levin. The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms. *Russian Mathematical Surveys*, page 11, 1970.

Appendix

A Interpretation of Lemma 1

We briefly give an intuitive interpretation of the facts mentioned in Lemma 1.

Lemma 1

a. Let $q : X^* \to Y^*$ be a partial recursive function. Then, for all $S \in X^*$, $H_Y(q(w)) \leq H_X(w) + O(1)$.
b. For all $S \in X^*$, $H_X(S) \leq |S| \cdot \log |X| + 2 \cdot \log |S| + O(1)$.
c. Let $A \subseteq X^* \times \mathbb{N}$ be a recursively enumerable set such that the subset $A_n = \{w : (w, n) \in A\}$ is finite for every $n \in \mathbb{N}$. Then, for all n, for every $S \in A_n$ with $|S| = n$, $H_X(S) \leq \log |A_n| + 4 \cdot \log n + O(1)$.
d. An infinite sequence $S \in X^\omega$ is T-random if and only if any of its suffixes is T-random.

The point *a.* states that the information content of the sequence $q(w)$ is no more than the information content of w (up to an additive constant). This simply comes from the fact that a computer program cannot produce more information than the information already present in its input.

The point *b.* comes from the fact that one can define a computer program p which simply copies its input to its output. Therefore, a program for generating the sequence S is given by the concatenation $\langle p, S\rangle$ of the program p, and the input S itself. This concatenation has length $|S|$ plus the length of p (which is independent of S), and an additional logarithmic term required to distinguish the two sequences p, S in the concatenation. The inequality follows by the definition of the complexity as the length of the shortest program producing the sequence S.

The interpretation of *c.* is slightly more involved. The set A being recursively enumerable means that there is a computer program p which can successively enumerate all the elements of A. Thanks to p, one can design a new program q which takes as input a number n (requiring $\log n$ bits), and an index $1 \leq i \leq |A_n|$ (requiring $\log |A_n|$ bits), and returns the i-th element in A_n having length equal to n. Therefore an element $S \in A_n$ with $|S| = n$ can be generated by the program $\langle q, i, n\rangle$ where i is the corresponding index of S. This explains the logarithmic terms on the right hand side.

Finally, the point *d.* simply comes from the fact that the notion of T-randomness depends only on the asymptotic behaviour of the infinite schedule.

B Leader election - upper bound

The proof is achieved by analyzing a specific algorithm, hereafter called $\mathcal{B}^3$, from a previous work [8]. The authors have shown that the algorithm $\mathcal{B}^3$ solves leader election over arbitrary strongly connected graphs using another fairness assumption, namely the global fairness [5].

We show that, actually, the algorithm $\mathcal{B}^3$ solves leader election on G under adversary $\mathbb{A}(T, G)$ for every $T > \log\theta(G)/\log d$. In the sequel, we use some combinatorial results from [13].

Definition 5 (Word Correlation [13]). *Let u, v be two sequences in X^*. The correlation $\langle u, v\rangle$ is the polynomial $a_1 z^{|u|-1} + a_2 z^{|u|-2} \cdots + a_{|u|}$ where $a_i \in \{0, 1\}$ is obtained as follows. Place v under u so that its leftmost symbol is under the i-th symbol of u. Then if all the pairs of symbols in the overlapping segment are identical, $a_i = 1$, else $a_i = 0$.*

Lemma 3 (Number of words omitting some word [13]). *Consider some sequence $u \in X^*$ of length k. Let $f(z) = \langle u, u\rangle$ be the autocorrelation polynomial of u. Then, the absolute value θ_u of the largest zero of the polynomial $1+(z-d)f(z)$ satisfies $1.7 < \theta_u < d$. Moreover, the set $A_n(u)$ of sequences of length n not containing u as a factor satisfies*

$$|A_n(u)| = \frac{\theta_u^n}{1 - (d - \theta_u)^2 f'(\theta_u)} + O((1.7)^n)$$

Now, we prove that the algorithm $\mathcal{B}^3$ presented in [8] solves leader election on G with adversary $\mathbb{A}(T, G)$ for all $T \in (\log\theta(G)/\log d, 1]$.

The pseudo-code is presented in Algorithm 1. Each process x can be leader or non-leader (variable $leader_x$) and can hold a white or black token (variable $token_x$). Initially, every process is a leader and holds a black token. The tokens move through the network by swapping between two processes during an interaction. When two black tokens meet, one of them turns white. When a white token interacts with a leader x, x becomes a non-leader and the token is destroyed. Given a configuration γ, let $b(\gamma)$ be the number of black tokens, $w(\gamma)$ the number of white tokens and $l(\gamma)$ the number of leaders in γ.

Algorithm 1: Algorithm $\mathcal{B}^3$

```
1  variables for every process x:
2      leader_x : 0 (non-leader) or 1 (leader)
3      token_x : ⊥ (no token), white or black
4  initialization: ∀x, (leader_x, token_x) = (1, black)          /* uniform */
5  algorithm (initiator x, responder y):
6      if token_x = token_y = black then
7          token_y ← white
8      if token_x = white ∧ leader_y = 1 then
9          leader_y ← 0                          /* y becomes a non-leader */
10         token_x ← ⊥                           /* the token is destroyed */
11     token_x ↔ token_y                                  /* swap the tokens */
```

Lemma 4. *For any configuration γ of Algorithm 1 reachable from the initial configuration, $b(\gamma) + w(\gamma) = l(\gamma)$ and $b(\gamma) \geq 1$.*

Proof. The initial configuration satisfies this relation. During an interaction, if no leader is turned into a non-leader, then the total number of tokens remains constant. When a leader

is turned into a non-leader (by a white token), the corresponding token is also destroyed. Moreover, destroying a black token requires that another black token collide with it, so there is always at least one black token. In any case, the first formula still holds. □

Consider the shortest loop $\pi = (a_1, a_2)(a_2, a_3)\dots(a_k, a_1)$ in G which visits every node at least once. We have $k = K(G)$. We define a finite schedule $S_0 = \pi\pi$, i.e., the path π repeated twice.

Lemma 5. *Let γ be any configuration reachable from the initial configuration such that $l(\gamma) \geq 2$. Then, applying the finite schedule S_0 to γ yields a configuration γ' such that $(1,1) \leq (l(\gamma'), b(\gamma')) < (l(\gamma), b(\gamma))$ in the lexicographical order (l first).*

Proof. We denote by l, b, w (resp. l', b', w') the value of $l(\gamma), b(\gamma), w(\gamma)$ (resp. $l'(\gamma), b'(\gamma), w'(\gamma)$). We first examine the case $b = 1$. By Lemma 4, we have $w = l - 1 \geq 1$ and, since no black token is ever created, $b' = 1$. Let a_i be some process in the path π which holds a white token in the configuration γ. When applying the schedule S_0 to the configuration γ, the white token moves (by swapping) from the process a_i through all the processes of G at least once. Thus, the white token necessarily meets with some leader in the graph, and this leader then disappears (along with the white token). Therefore, $l' < l$. Assume now that $b \geq 2$. When applying the schedule S_0 to γ, either the previous scenario occurs (and then $l' < l$), or two black tokens meet, and one of them turns white (thus $b' < b$). In either case, we have $(l', b') < (l, b)$. □

Proposition 4. *For any strongly connected graph G*

$$T(LE, G) \leq \frac{\log \theta(G)}{\log d} < 1$$

where $\theta(G)$ is the absolute value of the largest zero of the polynomial $1 + (z-d)(z^{2K(G)-1} + z^{K(G)-1})$ with $K(G)$ being the length of the shortest loop visiting every node at least once in G

Proof. Consider any $T > \log \theta(G)/\log d$, and any T-random schedule S. We prove that S_0 occurs infinitely often in S. Assume that S_0 occurs only finitely many times. Without loss of generality, we can assume that S_0 does not occur at all in S. Then, for every n, the prefix $S \restriction n$ belongs to the set $A_n(S_0)$ of finite schedule of length n which do not contain S_0 as a factor. By Lemma 1, we have, for all n,

$$H_G(S \restriction n) \leq \log |A_n(S_0)| + 4 \log n + O(1) \tag{1}$$

To estimate the value of $|A_n(S_0)|$, we compute the autocorrelation polynomial of $S_0 = \pi\pi$. Since π is a loop of shortest length $k = K(G)$ visiting every node at least once, the autocorrelation polynomial of π is $z^{K(G)-1}$. Indeed, let $\pi = (a_1, a_2)\dots(a_k, a_{k+1})$ with $a_{k+1} = a_1$. Assume that for some $1 \leq i \leq k$,

$$(a_i, a_{i+1})\dots(a_k, a_1) = (a_1, a_2)\dots(a_{k-i+1}, a_{k-i+2})$$

Then, defining $j = \max(i, k-i+1)$, the sequence $(a_1, a_2)\dots(a_j, a_{j+1})$ is a loop visiting every vertex at least once. This imposes $j = k$, and thus $i = 1$. By definition, this implies

that the autocorrelation polynomial of π is $z^{K(G)-1}$. Since $S_0 = \pi\pi$, the autocorrelation polynomial of S_0 is $z^{2K(G)-1} + z^{K(G)-1}$.

Note that $\theta = \theta(G)$ is precisely the absolute value of the largest zero of the autocorrelation polynomial of S_0. By Lemma 3, there exists a constant $k > 0$ such that, for all n, $|A_n(S_0)| = k\theta^n + O((1.7)^n)$. Plugging this into Equation 1, and using the fact that S is T-random, we have, for all n,

$$T \cdot n \log d \leq n \log \theta(G) + 4 \log n + O(1)$$

Therefore, $T \leq \log \theta(G) / \log d$; whence a contradiction. This proves that S_0 occurs infinitely often in S.

Then, by Lemma 5, the corresponding execution of $\mathcal{B}^3$ satisfies the specification of the leader election problem. In other words, $\mathcal{B}^3$ solves leader election with adversary $\mathbb{A}(T, G)$. In particular, this shows $T(LE, G) \leq T$ for every $T > \log \theta(G) / \log d$; whence $T(LE, G) \leq \log \theta(G) / \log d$. □

Remark 2. Note that this algorithm was designed in [8] to solve leader election without any knowledge about the underlying communication graph. Adding such a knowledge may yield a better upper bound on $T(LE, G)$.