Randomized Distributed Decision

Pierre Fraigniaud[*‡] Mika Göös[†] Amos Korman[*‡] Merav Parter [§¶]
David Peleg[§‖]

Abstract

The paper tackles the power of randomization in the context of local *distributed* computing by analyzing the ability to "boost" the success probability of deciding a distributed language using a Monte-Carlo algorithm. We prove that, in many cases, the ability to increase the success probability for deciding distributed languages is rather limited. This contrasts with the sequential computing setting where boosting can systematically be achieved by repeating the randomized execution.

[*]CNRS and University Paris Diderot, France . Email: {pierre.fraigniaud,amos.korman}@liafa.jussieu.fr
[†]Department of Computer Science of the University of Toronto, Canada. Email: mika.goos@mail.utoronto.ca
[‡]Supported by the ANR project DISPLEXITY, and by the INRIA project GANG.
[§]The Weizmann Institute of Science, Rehovot, Israel. Email:{merav.parter,david.peleg}@weizmann.ac.il.
[¶]Additional support from the Google Europe Fellowship in distributed computing.
[‖]Supported in part by the Israel Science Foundation (grant 894/09), the US-Israel Binational Science Foundation (grant 2008348), the Israel Ministry of Science and Technology (infrastructures grant), and the Citi Foundation.

1 Introduction

1.1 Context and objective

The impact of randomization on computation is one of the most central questions in computer science. In particular, in the context of distributed computing, the question of whether randomization helps in improving locality for construction problems has been studied extensively. While most of these studies were problem-specific, several attempts have been made for tackling this question from a more general and unified perspective. For example, Naor and Stockmeyer [26] focus on a class of problems called LCL (essentially a subclass of the class LD discussed below), and show that if there exists a randomized algorithm that constructs a solution for a LCL problem in a constant number of rounds, then there also exists a constant-time deterministic algorithm constructing a solution for that problem.

Recently, the impact of randomization has been studied [13] in the context of *local decision*, where one aims at deciding locally whether a given global input instance belongs to some specified *language* defined as a decidable collection of input instances. The efficiency of deterministic algorithms and of randomized Monte Carlo algorithms are compared in [13], in the $\mathcal{LOCAL}$ model (cf. [28]). One of the main results of [13] is that randomization does not help for locally deciding *hereditary* languages if the success probability is beyond a certain guarantee threshold. (A language is hereditary if it is closed under node deletion). More specifically, a (p, q)-*decider* for a distributed language $\mathcal{L}$ is a distributed randomized Monte Carlo algorithm that *accepts* instances in $\mathcal{L}$ with probability at least p and *rejects* instances outside of $\mathcal{L}$ with probability at least q. It was shown in [13] that every hereditary language that can be decided in t rounds by a (p, q)-decider, where $p^2+q > 1$, can actually be decided *deterministically* in $O(t)$ rounds. On the other hand, [13] showed that the aforementioned threshold is sharp, at least when hereditary languages are concerned. In particular, for every p and q where $p^2 + q \leq 1$, there exists a hereditary language that cannot be decided deterministically in $o(n)$ rounds, but can be decided in zero rounds by a (p, q)-decider.

The main motivation for this paper is to provide a deeper understanding of the use of randomization in the context of local decision. In particular, we aim at investigating the issue of boosting the success probabilities p and q of a (p, q)-decider. (Recall that in the *sequential* Monte Carlo setting, such "boosting" can easily be achieved by repeating the execution of the algorithm a large number of times.)

For constant $p, q \in (0, 1]$ and a function t of triplets $(G, \mathbf{x}, \mathrm{Id})$ where G denotes the network, $\mathbf{x}$ denotes the input vector, and Id denotes the identity assignment to the nodes, [13] defines the class $\mathrm{BPLD}(t, p, q)$ of all distributed languages that have a randomized distributed (p, q)-decider running in time at most t (i.e., can be decided in time at most t by a randomized distributed algorithm with "yes" success probability p and "no"success probability q). We are interested in studying the connections between the classes $\mathrm{BPLD}(t, p, q)$, for various t, p, and q. Note that, with this terminology, the derandomization result of [13] states that, for hereditary languages, $\mathrm{BPLD}(t, p, q)$ collapses to $\mathrm{LD}(O(t))$ for all p and q such that $p^2 + q > 1$. One crucial question addressed in this paper is whether this collapse holds in general, i.e., for all languages, and not only for hereditary languages.

Moreover, this paper questions the structure of $\mathrm{BPLD}(t, p, q)$ below the threshold $p^2 + q = 1$.

For this purpose, for every positive integer k, we consider the class

$$B_k(t) = \bigcup_{p^{1+1/k}+q>1} \text{BPLD}(t, p, q),$$

as well as the class

$$B_\infty(t) = \bigcup_{p+q>1} \text{BPLD}(t, p, q),$$

and study two concerns: the relationships between different classes B_k and $B_{k'}$, and the ability to improve (i.e., to "boost") the success probability of languages within a class B_k.

Note that, thanks to [13], $B_1(t)$ restricted to hereditary languages is contained in $\text{LD}(O(t))$, the class of languages that can be decided deterministically in $O(t)$ time. Note also that the class

$$\text{All} = \bigcup_{p+q\geq 1} \text{BPLD}(0, p, q)$$

contains *all* languages. For instance, every language can be decided using a $(1, 0)$-decider that systematically returns "yes" at every node (without any communication).

Hence, the classes B_k provide a smooth spectrum of randomized distributed complexity classes, from the class of deterministically decidable languages (at least for hereditary ones) to the class of all languages. The ability of boosting the success probabilities of a (p, q)-decider is directly related to the question of whether these classes are different, and to what extent.

1.2 Our Results

One of the main outcomes of this paper is a proof that boosting success probabilities in the context of distributed decision is quite limited. This contrasts with the sequential computing setting in which boosting can be achieved by repeating the randomized execution.

Specifically, in our first main result, we show that, in fact, $B_1(t)$ is not contained in $\text{LD}(O(t))$, and not even in $\text{LD}(o(n))$. This is achieved by providing a (non-hereditary) language with a (p, q)-decider running in constant time, where p, q are above the threshold $p^2 + q > 1$, which cannot be decided deterministically in $o(n)$ rounds. On the positive side, we show that $B_1(O(1))$ restricted to path networks is contained in $\text{LD}(O(1))$. That is, we show that any language (even non-hereditary) defined on paths having a (p, q)-decider running in time $O(1)$ with $p^2 + q > 1$ can be be decided deterministically in $O(1)$ rounds.

We then present a more refined analysis for the family of languages that can be decided randomly but not deterministically. That is, we focus on the family of languages that can be decided locally by a (p, q)-decider, where $p^2+q \leq 1$. By definition, $B_k(t) \subseteq B_{k+1}(t)$ for any k and t. We prove that these inclusions are strict. In fact, we prove a stronger separation result: there exists a language in $B_{k+1}(0)$ that is not in $B_k(t)$ for any $t = o(n)$. Moreover, $\texttt{Tree} \notin B_\infty(t)$ for any $t = o(n)$, where $\texttt{Tree} = \{(G, \epsilon) : G \text{ is a tree}\}$. Hence $B_\infty(t)$ does not contain all languages, even for $t = o(n)$. In summary, we obtain the hierarchy

$$B_1(t) \subset B_2(t) \subset \cdots \subset B_k(t) \subset \cdots \subset B_\infty(t) \subset \text{All}$$

for all $t = o(n)$. These hierarchy demonstrates that boosting the probability of success is somewhat restricted as an arbitrary (p, q)-decider with $p^{1+1/k} + q > 1$ cannot be systematically transformed into a $(\hat{p}, \hat{q})$-decider with with $\hat{p}^{1+1/(k-1)} + \hat{q} > 1$, even by allowing significantly more time.

Finally, we consider the case where the distribution of inputs can be restricted in certain ways (i.e., the case of languages with promises). For such cases, we show that the ability to boost the success probability become almost null.

All our results hold for the $\mathcal{LOCAL}$ model as well as for more restrictive models of computation, such as the $\mathcal{CONGEST}(B)$ (for $B = O(1)$).

1.3 Related Work

The notion of local decision and local verification of languages has received quite a lot of attention recently. In the $\mathcal{LOCAL}$ model, solving a decision problem requires the processors to independently inspect their local neighborhood and collectively decide whether the global instance belongs to some specified language. Inspired by classical computation complexity theory, [13] suggested that the study of decision problems may lead to new structural insights also in the more complex distributed computing setting. Indeed, following that paper, efforts were made to form a fundamental computational complexity theory for distributed decision problems in various other aspects of distributed computing [13, 14, 15, 16].

The classes LD, NLD and BPLD defined in [13] are the distributed analogues of the classes P, NP and BPP, respectively. The contribution of [13] is threefold: it establishes the impact of nondeterminism, randomization, and randomization + nondeterminism, on local computation. This is done by proving structural results, developing a notion of local reduction and establishing completeness results. One of the main results is the existence of a sharp threshold above which randomization does not help (at least for hereditary languages), and the BPLD classes were classified into two: below and above the randomization threshold. The current paper "zooms into" the spectrum of classes below the randomization threshold, and defines an infinite hierarchy of BPLD classes, each of which is separated from the class above it in the hierarchy.

The question of whether randomization helps in improving locality for construction problems has been studied extensively. Naor and Stockmeyer [26] considered a subclass of LD($O(1)$), called LCL[1], and studied the question of how to compute in $O(1)$ rounds the constructive versions of decision problems in LCL. The paper demonstrates that randomization does not help, in the sense that if a problem has a local Monte Carlo randomized algorithm, then it also has a local deterministic algorithm. There are several differences between the setting of [26] and ours that might account for this apparent dichotomy. First, [26] considers the power of randomization for *constructing* a solution, whereas we study the power of randomization for *deciding* languages[2]. Second, while [26] deals with constant time computations, our separation results apply to arbitrary time computations, potentially depending on the size of the instance (graph and input). The different settings might provide an explanation for the different impacts for randomization: while the current paper and [13]

[1] LCL is essentially LD($O(1)$) restricted to languages involving graphs of constant maximum degree and processor inputs taken from a set of constant size.

[2] There is a fundamental difference between such tasks when locality is concerned. Indeed, whereas the validity of constructing a problem in LCL is local (by definition), the validity in our setting is "global", in the sense that in an illegal instance, it is sufficient that at least one vertex in the entire network outputs "no".

show that randomization can indeed help for improving locality of decision problems, [26] shows that randomization does *not* help in constructing a solution for a problem in LCL in constant time. The question of whether randomization helps in local computations was studied for *specific* problems, such as MIS, $(\Delta+1)$-coloring, and maximal matching [2, 5, 23, 24, 25, 27, 29].

Finally, the classification of decision problems in distributed computing has been studied in several other models. For example, [6] and [18] study specific decision problems in the $\mathcal{CONGEST}$ model. In addition, decision problems have been studied in the asynchronous discipline too, specifically in the framework of *wait-free computation* [15, 16] and *mobile agent computing* [14]. In the wait-free model, the main issues are not spatial constraints but timing constraints (asynchrony and faults). The main focus of [16] is deterministic protocols aiming at studying the power of the "decoder", i.e., the interpretation of the results. While this paper essentially considers the AND-checker, (as a global "yes" corresponds to all processes saying "yes"), [16] deals with other interpretations, including more values (not only "yes" and "no"), with the objective of designing checkers that use the smallest number of values.

2 Model and preliminaries

2.1 Model

We consider the $\mathcal{LOCAL}$ model (cf. [28]), which is a standard distributed computing model capturing the essence of spatial locality. In this model, processors are woken up simultaneously, and computation proceeds in fault-free synchronous rounds during which every processor exchanges messages of unlimited size with its neighbors, and performs arbitrary computations on its data. More specifically, in the $\mathcal{LOCAL}$ model, processors perform in synchronous rounds, where in each round, every processor (1) sends messages of arbitrary size to its neighbors, (2) receives messages from its neighbors, and (3) performs arbitrary individual computations. After a number of rounds (that may depend on the network G connecting the processors, and may vary among the processors since nodes have different identities, potentially different inputs, and are typically located at non-isomorphic positions in the network), every processor v terminates and generates its output.

Clearly, all impossibility results holding for the the $\mathcal{LOCAL}$ model hold also with respect to the more restrictive $\mathcal{CONGEST}(B)$ model of computation, which allows only messages containing at most B bits (cf. [28]). Hence, all our negative results hold also for the $\mathcal{CONGEST}(B)$ model. It is important to stress that all the algorithmic constructions that we employ in our positive results use messages of constant size (some of which do not use any communication at all). Hence, all our results apply not only to the $\mathcal{LOCAL}$ model of computation but also to $\mathcal{CONGEST}(B)$, where $B = O(1)$.

A distributed algorithm $\mathcal{A}$ that runs on a graph G operates separately on each connected component, and nodes of a component C of G cannot distinguish the underlying graph G from C. Therefore, we consider connected graphs only.

We focus on *distributed decision tasks*. Such a task is characterized by a finite or infinite set Σ of symbols (e.g., $\Sigma = \{0,1\}$, or $\Sigma = \{0,1\}^*$), and by a *distributed language* $\mathcal{L}$ defined on this set of symbols. An *instance* of a distributed decision task is a pair $(G, \mathbf{x})$ where G is an n-node connected graph, and $\mathbf{x} \in \Sigma^n$. Every node $v \in V(G)$ is assigned as its *local input* a value $\mathbf{x}(v) \in \Sigma$.

(In some cases, the local input of every node is empty, i.e., $\Sigma = \{\epsilon\}$, where ϵ denotes the empty binary string.) We define a *distributed language* as a decidable collection $\mathcal{L}$ of instances[3]. Given a language $\mathcal{L}$, an instance $(G, \mathbf{x})$ is called *legal* if and only if $(G, \mathbf{x}) \in \mathcal{L}$.

In the context of distributed computing, each processor must produce a boolean output, and the decision is defined by the conjunction of the processors outputs, as follows. If the instance belongs to the language, then all processors must output "yes", and otherwise, at least one processor must output "no". Formally, for a distributed language $\mathcal{L}$, we say that a distributed algorithm $\mathcal{A}$ *decides* $\mathcal{L}$ if and only if for every instance $(G, \mathbf{x})$ and id-assignment Id, every node v of G eventually terminates and produces an output denoted $\text{out}_{\mathcal{A}}(G, \mathbf{x}, \text{Id}, v)$, which is either "yes" or "no", satisfying the following decision rules:

- If $(G, \mathbf{x}) \in \mathcal{L}$, then $\text{out}_{\mathcal{A}}(G, \mathbf{x}, \text{Id}, v) =$ "yes" for every node $v \in V(G)$;
- If $(G, \mathbf{x}) \notin \mathcal{L}$, then $\text{out}_{\mathcal{A}}(G, \mathbf{x}, \text{Id}, v) =$ "no" for at least one node $v \in V(G)$.

Decision problems provide a natural framework for tackling fault-tolerance: the processors have to collectively check if the network is fault-free, and a node detecting a fault raises an alarm. In fact, many natural problems can be phrased as decision problems, for example: "is the network planar?" or "is there a unique leader in the network?". Moreover, decision problems occur naturally when one aims at checking the validity of the output of a computational task, such as "is the produced coloring legal?", or "is the constructed subgraph an MST?".

The class of decision problems that can be solved in at most t communication rounds is denoted by $\text{LD}(t)$, for *local decision*. More precisely, let t be a function of triplets $(G, \mathbf{x}, \text{Id})$, where Id denotes the identity assignment to the nodes of G. Then $\text{LD}(t)$ is the class of all distributed languages that can be decided by a distributed algorithm that runs in at most t communication rounds. The randomized (Monte Carlo 2-sided error) version of the class $\text{LD}(t)$ is denoted $\text{BPLD}(t, p, q)$, which stands for *bounded-error probabilistic local decision*, and provides an analog of BPP for distributed computing, where p and q respectively denote the yes-error and the no-error guarantees. More precisely, a *randomized* distributed algorithm is a distributed algorithm $\mathcal{A}$ that enables every node v, at any round r during its execution, to generate a certain number of random bits. For constants $p, q \in (0, 1]$, we say that a randomized distributed algorithm $\mathcal{A}$ is a (p, q)-*decider* for $\mathcal{L}$, or, that it decides $\mathcal{L}$ with "yes" success probability p and "no" success probability q, if and only if for every instance $(G, \mathbf{x})$ and id-assignment Id, every node of G eventually terminates and outputs "yes" or "no", and the following properties are satisfied:

- If $(G, \mathbf{x}) \in \mathcal{L}$ then $\Pr[\forall v \in V(G), \text{out}_{\mathcal{A}}(G, \mathbf{x}, \text{Id}, v) = \text{"yes"}] \geq p$;
- If $(G, \mathbf{x}) \notin \mathcal{L}$ then $\Pr[\exists v \in V(G), \text{out}_{\mathcal{A}}(G, \mathbf{x}, \text{Id}, v) = \text{"no"}] \geq q$.

The probabilities in the above definition are taken over all possible coin tosses performed by the nodes. The running time of a (p, q)-decider executed on a node v depends on the triple $(G, \mathbf{x}, \text{Id})$ and on the results of the coin tosses. In the context of a randomized algorithm, $T_v(G, \mathbf{x}, \text{Id})$ denotes the maximal running time of the algorithm on v over all possible coin tosses, for the instance $(G, \mathbf{x})$

[3]Note that an undecidable collection of instances remains undecidable in the distributed setting too.

and id-assignment Id. Now, just as in the deterministic case, the running time T of the (p,q)-decider is the maximum running time over all nodes. Note that by definition of the distributed Monte-Carlo algorithm, both T_v and T are deterministic.

2.2 Preliminaries

2.2.1 The "all-yes" event

Let G be a connected graph, with node-set $V(G)$ and edge-set $E(G)$. For a set $U \subseteq V(G)$, we identify a specific probabilistic event. Given a distributed algorithm $\mathcal{A}$, we denote by

$$\mathcal{E}(G, \mathbf{x}, \mathrm{Id}, U)$$

the event that, when running $\mathcal{A}$ on $(G, \mathbf{x})$ with id-assignment Id, all nodes in U output "yes".

2.2.2 Secure subpaths

An n-node path P is denoted $P = (u_1, \ldots, u_n)$, or simply $P = (1, \ldots, n)$. (Node u_i, the i-th node of the path, does not know its position in the path.) Node 1 is the leftmost node, and node n is the rightmost node. Given an instance $(P, \mathbf{x})$ with identity assignment Id, and a given subpath $S \subseteq P$, let $\mathbf{x}_S$ (respectively Id_S) be the restriction of $\mathbf{x}$ (resp., Id) to S. We may refer to subpath $S = (i, \ldots, j) \subset P$ as $S = [i, j]$. Given a time bound t, a subpath $S = [i, j]$ is called an *internal* subpath of P if $i \geq t + 2$ and $j \leq n - t - 1$. Note that if the subpath S is internal to P, then when running a t-round algorithm, none of the nodes in S "sees" the endpoints of P. The following concept is crucial to the proofs of our separation results.

Definition 2.1. *Let S be a subpath of P. For $\delta \in (0,1)$, and a positive integer λ, a subpath S is said to be* (δ, λ)-secure *if $|S| \geq \lambda$ and $\Pr[\mathcal{E}(P, \mathbf{x}, \mathrm{Id}, V(S))] \geq 1 - \delta$.*

We typically use (δ, λ)-secure subpaths for values of $\lambda \geq 2t + 1$ where t is the running time of the (p,q)-decider $\mathcal{A}$ on $(P, \mathbf{x})$ for some fixed identity assignment Id. Indeed, it is known [13] that if $(P, \mathbf{x}) \in \mathcal{L}$, then every long enough subpath S of P contains an internal (δ, λ)-secure subpath S'. More precisely, define

$$\ell(\delta, \lambda) = (\lambda + 2t) \left\lceil \frac{\log p}{\log(1 - \delta)} \right\rceil . \tag{1}$$

Let $\mathcal{L}$ be a distributed language, and let $\mathcal{A}$ be a distributed algorithm deciding $\mathcal{L}$. The following fact is implicit in [13]; we sketch its proof for the sake of completeness.

Fact 2.2. *Let $(P, \mathbf{x}) \in \mathcal{L}$, $\delta \in (0,1)$, $\lambda \geq 1$. Then for every subpath S of P with length $\ell(\delta, \lambda)$, there is a subpath S' (internal to S) that is (δ, λ)-secure.*

Proof. Let $(P, \mathbf{x}) \in \mathcal{L}$, $\delta \in (0,1)$, $\lambda \geq 1$, and let S be a $\ell(\delta, \lambda)$-length subpath of P. Let $S_1, \ldots, S_k$ be consecutive disjoint $(\lambda + 2t)$-length segments of S where $k = \lceil \log p / \log(1 - \delta) \rceil$. The *deeply internal* vertices of S_i are all the vertices at distance at least t from both endpoints of the subpath S_i. Let E_i denote the event that all the deeply internal vertices of S_i output "yes". Note that the events E_i are mutually independent, as E_i is only a function of the information present in S_i

(e.g., the identities, inputs, and random bits). Hence, at least some E_i must occur with probability at least $1-\delta$. Indeed, otherwise, Algorithm $\mathcal{A}$ would accept $(P,\mathbf{x})$ with probability less than $(1-\delta)^k$ (by the independence of the E_i), which is smaller than p by definition of k, contradicting the fact that $\mathcal{A}$ is a (p,q)-decider. Any S_i such that E_i occurs with probability at least $1-\delta$ is (δ,λ)-secure. □

To avoid cumbersome notation, when $\lambda = 2t+1$, we may omit it and refer to $(\delta, 2t+1)$-secure subpaths as δ-secure subpaths. In addition, set

$$\ell(\delta) = \ell(\delta, 2t+1).$$

Let us next illustrate a typical use of Fact 2.2. Recall that t denotes the running time of the (p,q)-decider $\mathcal{A}$ on $(P,\mathbf{x}) \in \mathcal{L}$ with identity assignment Id. Let S be a subpath of P of length $\ell(\delta)$. Denote by L (resp., R) the subpath of P to the "left" (resp., "right") of S. Informally, if the length of S is larger than $2t+1$, then S serves as a separator between the two subpaths L and R. This follows since as algorithm $\mathcal{A}$ runs in t rounds, each node in P is affected only by its t neighborhood. As the t neighborhood of every node $u \in L$ and $v \in R$ do not intersect, the events $\mathcal{E}(P,\mathbf{x},\mathrm{Id},L)$ and $\mathcal{E}(P,\mathbf{x},\mathrm{Id},R)$ are independent.

The security property becomes useful when upper bounding the probability that at least some node in P says "no", by applying a union bound on the complements of the events

$$\mathcal{E}\left(P,\mathbf{x},\mathrm{Id},V(L)\cup V(R)\right) \text{ and } \mathcal{E}\left(P,\mathbf{x},\mathrm{Id},V\left(S\right)\right).$$

Denoting the event complementary to $\mathcal{E}(P,\mathbf{x},\mathrm{Id},V(P))$, by $\bar{\mathcal{E}}$ we have

$$\begin{aligned}
\Pr[\bar{\mathcal{E}}] &= 1-\Pr[\mathcal{E}(P,\mathbf{x},\mathrm{Id},V(P))] \\
&\leq (1-\Pr[\mathcal{E}(P,\mathbf{x},\mathrm{Id},V(L))]\cdot\Pr[\mathcal{E}(P,\mathbf{x},\mathrm{Id},V(R))]) + (1-\Pr[\mathcal{E}(P,\mathbf{x},\mathrm{Id},V(S))]) \\
&\leq 1-\Pr[\mathcal{E}(P,\mathbf{x},\mathrm{Id},V(L))]\cdot\Pr[\mathcal{E}(P,\mathbf{x},\mathrm{Id},V(R))]+\delta\ .
\end{aligned}$$

The specific choice of λ and δ depends on the context. Informally, the guiding principle is to set δ small enough so that the role of the central section S can be neglected, while dealing separately with the two extreme sections L and R become manageable for they are sufficiently far apart.

2.2.3 The language AMOS-k

Many of our results are obtained by studying the language `At-Most-k-Selected` (abbreviated in `AMOS-`k) defined as follows. Each node v has input $\mathbf{x}(v) \in \{0,1\}$. An instance is in `AMOS-`$k$ if and only if the number of nodes with input 1 (called *selected* nodes) is at most k. That is,

$$\texttt{AMOS-}k = \{(G,\mathbf{x}) \text{ s.t. } \|\mathbf{x}\|_1 \leq k\}.$$

In particular, `AMOS-1`, a.k.a. `AMOS`, consists of all instances containing at most one selected nodes (i.e., at most one node with input 1), with all other nodes unselected (having input 0).

3 On the ability to de-randomize

It is known [13] that, for *hereditary* languages, i.e., languages closed under node deletion, randomization does not help beyond the threshold $p^2 + q = 1$. In this section, we show that randomization helps in general, in the sense that there exists a (non-hereditary) language that can be decided in constant time by a (p, q)-decider for some p and q such that $p^2 + q > 1$, and yet cannot be decided deterministically, even in time $o(\sqrt{n})$. On the positive side, we show that the randomization threshold of $p^2 + q = 1$ holds for all languages, including non-hereditary languages, as long as the instances are restricted to path networks an finite input.

3.1 The de-randomization threshold does not hold in general

In this section, we show the following negative result, which proves that not only some languages decidable in constant time by a (p, q)-decider with $p^2 + q > 1$ cannot be decided in constant time by a deterministic algorithm, but even for $c \geq 2$ arbitrarily large, there are languages decidable in constant time by a (p, q)-decider with $p^c + q > 1$ that cannot be decided in constant time by a deterministic algorithm.

Theorem 3.1. *For any $c \geq 2$, there exists a (non-hereditary) language $\mathcal{L}$ with a (p, q)-decider satisfying $p^c + q > 1$ and running in one round, which cannot be decided deterministically in $o(\sqrt{n})$ rounds.*

Proof. Let a and b be two positive integers such that $1 + \frac{b}{a}$ is an integer strictly larger than c. In particular, as we are interested in $c \geq 2$, we have $b > a$. We shall soon define the language $\mathtt{mod}(a, b)$, which will be proved to be in LD(1) for every fixed a and b, where $b > a$. Towards this end, we define a family of instances $(G_{k,m}, \mathbf{x})$, for $k \geq 3$ and $m \geq 1$.

The graph $G_{k,m}$: such a graph consists of m concentric k-node cycles $C_1, C_2, \ldots, C_m$, plus an additional center vertex. The outermost cycle is C_1, the innermost cycle is C_m, and the j-th vertex of the cycle C_i is connected to the j-th vertex of the cycle C_{i-1}. Finally, the center vertex is connected to all the vertices in C_m (hence its degree is k). For an illustration see Figure 1.

The language $\mathtt{mod}(a, b)$: A legal instance of $\mathtt{mod}(a, b)$ consists in a pair $(G, \mathbf{x})$ satisfying the following. The prerequisite condition is that $G = G_{k,m}$, for some k and m. Additionally, the input $\mathbf{x}$ must have the following structure. The center node has an empty input, and, for every other node, the input consists of three fields, namely $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$, satisfying the following:

– The first field $\mathbf{x}_1(v) \in \{1, \ldots, k\}$ encodes at node v an orientation for each cycle. That is, all nodes of each cycle are consecutively labeled from 1 to k by $\mathbf{x}_1$. This orientation is called clockwise. The additional requirement is that, for every $j \in \{1, \ldots, k\}$, the node v with $\mathbf{x}_1(v) = j$ in cycle C_i, for $1 < i \leq m$, is adjacent to the node u with $\mathbf{x}_1(u) = j$ in cycle C_{i-1}. For later references, let us denote by s the node v of C_1 with $\mathbf{x}_1(v) = 1$. This node is called the *source*.

– The second input field $\mathbf{x}_2(v)$ satisfies $\mathbf{x}_2(v) \in \{0, 1\}$ for $v \in C_1$, and $\mathbf{x}_2(v) = i$ for $v \in C_i$, where $i > 1$. This input is interpreted as follows. For inner cycles, this field is simply the index of the cycle. For the outer cycle C_1, this field indicates whether the node v is a *leader* (i.e., $\mathbf{x}_2(v) = 1$)

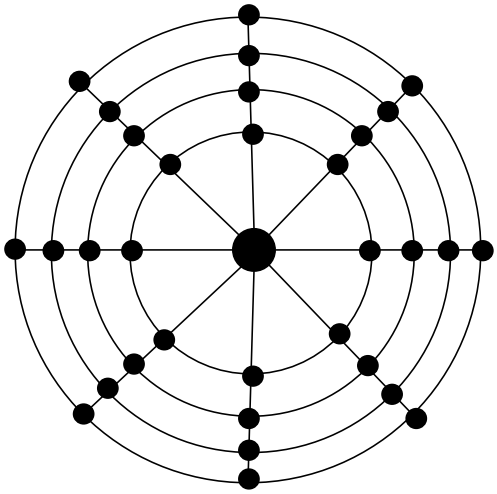

Figure 1: *Schematic illustration of the graph* $G_{k,m}$.

or not (i.e., $\mathbf{x}_2(v) = 0$). (In particular, leaders exists only on the outermost cycle C_1). Moreover, we require $\mathbf{x}_2(s) = 0$ for the source node s, i.e., the source is not a leader. Also, the predecessor of the source, i.e., the node v with $\mathbf{x}_1(v) = k$, that we call the *final*, must not be a leader. Additionally, the number L of leaders must satisfy

$$L \equiv a \pmod{b}.$$

– The third field $\mathbf{x}_3$ is empty for nodes not in C_1. For the nodes in C_1 the field must satisfy the following. First, $\mathbf{x}_3(s)$ is an arbitrary non-negative integer z. For every node in C_1, the input of the j-th node on the cycle will essentially represent z plus the number of leaders on C_1 (counted modulo b) between the source node and the $(j-1)$-st node of C_1, calculated clockwise. That is, the input $\mathbf{x}_3(v)$ of a node v with $\mathbf{x}_1(v) = j \geq 2$ is z plus the number of leaders calculated modulo b present in the subpath of C_1 consisting of all nodes u with $1 \leq \mathbf{x}_1(u) \leq j-1$.

This completes the description of the legal instances of the language $\texttt{mod}(a, b)$.

Claim 3.2. $\texttt{mod}(a, b) \in \mathrm{LD}(1)$.

To prove the claim, we need to specify a deterministic algorithm running in one round and deciding $\texttt{mod}(a, b)$. We first show that in one round, one can deterministically decide whether or not $G = G_{k,m}$, for some k and m. Indeed, in one round, a node v sends its input $\mathbf{x}(v)$ to its neighbors, and receives the inputs of all its neighbors. Then, every node first checks the local consistency of the two first fields $\mathbf{x}_1$ and $\mathbf{x}_2$. If there is local consistency at every node, then nodes are in a graph $G_{k,m}$. The remaining operations involve only the nodes of C_1. First, the source node (i.e., the node s with $\mathbf{x}_1(s) = 1$ and $\mathbf{x}_2(s) \in \{0, 1\}$ checks that it is not a leader, that is, it checks that $\mathbf{x}_2(s) = 0$. Similarly, the pre-source node v checks that $\mathbf{x}_2(s) = 0$. To establish the claim, it

remains to show that the nodes in C_1 can collectively check whether or not $L \equiv a \pmod b$. Every node v of C_1 distinct from the source checks that its predecessor u satisfies:

$$\mathbf{x}_3(v) = \begin{cases} (\mathbf{x}_3(u)+1) \bmod b & \text{if } u \text{ is a leader} \\ \mathbf{x}_3(u) & \text{if } u \text{ is a not a leader .} \end{cases}$$

Finally, the final node v (i.e., the node of C_1 with $\mathbf{x}_1(v) = k$) checks whether

$$\mathbf{x}_3(v) \equiv z + a \pmod b.$$

If an error is detected, then node v outputs "no", else it outputs "yes". If no errors are detected, then, by considering the final node, we have

$$(z + L) \bmod b = (z + a) \bmod b$$

implying $L \equiv a \pmod b$, as desired. Conversely, if $L \not\equiv a \pmod b$, then the final node will detect $\mathbf{x}_3(v) \not\equiv z + a \pmod b$ assuming that no other node outputs "no". This completes the proof of Claim 3.2. Now, let us consider the following language.

The language $\texttt{leader}(a, b)$**:** For an instance $(G, \mathbf{x})$ to be in $\texttt{leader}(a, b)$, one must have $G = G_{k,m}$ for some k and m, and the number of leaders must be exactly a. That is:

$$\texttt{leader}(a, b) = \{(G, \mathbf{x}) \in \texttt{mod}(a, b) \mid L = a\}. \tag{2}$$

For comparison, note that in $\texttt{mod}(a, b)$, it is just required that the number of leaders being equal to $a + \alpha b$ for non-negative integer α. The difference between the two languages is significant, as $\texttt{mod}(a, b) \in \mathrm{LD}(1)$ (cf. Claim 3.2) while we have the following impossibility result for $\texttt{leader}(a, b)$.

Claim 3.3. $\texttt{leader}(a, b) \notin \mathrm{LD}(o(\sqrt{n}))$.

To establish the claim, let us fix k and m, both roughly $\sqrt{n}$ for n large. Assume for contradiction that there exists a deterministic algorithm $\mathcal{D}$ enabling to decide $\texttt{leader}(a, b)$ in $o(\sqrt{n})$ rounds. We claim that there exists a legal input $\mathbf{x}$ and an illegal input $\mathbf{x}'$, both for graph $G_{k,m}$, that $\mathcal{D}$ cannot distinguish in $o(\sqrt{n})$ rounds. Essentially, $\mathbf{x}'$ is a legal instance for $\texttt{leader}(a + b, b)$. To construct these two instances, we split the outer cycle C_1 into $r = \frac{b}{a} + 1$ consecutive subpaths $P_1, \ldots, P_r$ of roughly equal length $\Omega(\sqrt{n})$. In each subpath P_i, we consider the a nodes at the center of it, forming a subpath Q_i of P_i with a nodes. In all the instances we shall consider hereafter, all nodes of C_1 outside of Q_i, $i = 1, \ldots, r$, are not leaders. Note that, in the execution of $\mathcal{D}$, every node v cannot see two nodes belonging to two different Q_i and Q_j, $j \neq i$, since such nodes are too far apart in $G_{k,m}$.

We consider $r + 1$ instances $I_0, I_1, \ldots, I_r$. These instances differ only in the number of leaders and in the value z_i assigned to the source in instance I_i. (Once the value of the source is decided, and the leaders in C_1 are decided, all the other inputs are fixed for legal instances in $\texttt{leader}(a, b)$). The instances $I_1, \ldots, I_r$ are all legal for $\texttt{leader}(a, b)$. Instance I_i consists in all nodes in Q_i being leader, and all nodes outside Q_i being non-leaders, with $z_i = (i-1)a \bmod b$. In contrast, Instance I_0 is an illegal instance for $\texttt{leader}(a, b)$. In I_0, $z_0 = 0$, and all nodes in all Q_i are leaders (all the other nodes being non-leaders). Hence, in I_0 the number of leaders is $ar = a + b$. Moreover, it induces a global legal assignment of the input $\mathbf{x}_3$ to all nodes making I_0 a legal instance in $\texttt{leader}(a + b, b)$.

Now observe that in I_0, every node in P_i executing $\mathcal{D}$ have the same view as it has in I_i. Hence, all nodes in P_i output "yes" under $\mathcal{D}$. So, in I_0, all nodes of C_1 output "yes". In fact, in I_0, every node in S_i executing $\mathcal{D}$ have the same view as in I_i, where S_i is the set of nodes closer to Q_i than to any other Q_j, $j \neq i$ (where ties are broken arbitrarily). Therefore, in I_0, all nodes are outputting "yes" under $\mathcal{D}$, contradicting the fact that $I_0 \notin \texttt{leader}(a, b)$. This completes the proof of Claim 3.3.

Claim 3.3 shows that $\texttt{leader}(a, b)$ cannot be solved fast deterministically. The claim below states that this is not the case anymore if one allows randomization.

Claim 3.4. *There exists a 1-round (p, q)-decider for $\texttt{leader}(a, b)$ satisfying $p^c + q > 1$, for any fixed $c \geq 2$.*

The desired (p, q)-decider $A_{\texttt{lead}}$ runs two algorithms. Let a and b be two positive integers such that $1 + \frac{b}{a} > c$. The first algorithm is used to decide $\texttt{mod}(a, b)$, which we know is in LD(1) by Claim 3.2. If one node outputs "no" under that algorithm, then Algorithm $A_{\texttt{lead}}$ also outputs "no" at that node. Hence, let us assume that all nodes output "yes" while deciding $\texttt{mod}(a, b)$. At this point, we just need to distinguish the case of the number of leaders L being a from being at least $a + b$. We proceed similarly to deciding AMOS in [13], that is, each non leader node output "yes" (with probability 1), and a leader node outputs "yes" with probability $p^{1/a}$, and "no" with probability $1 - p^{1/a}$. Note that this is doable in zero rounds, hence altogether our algorithm runs in just one round. On a legal instance (which has precisely $L = a$ leaders), the probability that all nodes output "yes" is

$$(p^{1/a})^L = (p^{1/a})^a = p.$$

On the other hand, on an illegal instance, we have $L \geq a + b$, and thus the probability that all nodes output "yes" is

$$(p^{1/a})^L \leq (p^{1/a})^{a+b} = p^{1+b/a} < p^c.$$

Therefore, the probability q to reject an illegal instance satisfies $q > 1 - p^c$. This completes the proof of Claim 3.4.

Claim 3.3 and Claim 3.4 together complete the proof of Theorem 3.1. □

3.2 The de-randomization threshold holds for all languages restricted to paths

In this section, we show the following positive result which states that, restricted to path networks and finite inputs, *every* language $\mathcal{L}$ for which there exists a (p, q)-decider running in constant time, with $p^2 + q > 1$, can actually be decided deterministically in constant time. This enlarges the set of languages for which de-randomization is possible. While [13] restricted the languages to be hereditary, here we restrict the languages to be defined on paths. That is, we consider only instances of the form $(G, \mathbf{x})$ where G is a path. Moreover, the alphabet Σ used to encode the inputs is restricted to be of constant size.

Theorem 3.5. *Let $\mathcal{L}$ be a distributed language restricted to paths, with a finite set of input values. If $\mathcal{L}$ can be decided with a (p, q)-decider running in constant time, with $p^2 + q > 1$, then $\mathcal{L}$ can be decided deterministically in constant time.*

Proof. Let $\mathcal{L} \in B_1(O(1))$ be a distributed language restricted to paths, and defined on the (finite) input set Σ. Consider a distributed (p,q)-decider $\mathcal{A}$ for $\mathcal{L}$ that runs in $t = O(1)$ rounds, with $p^2 + q > 1$. Fix a constant δ such that $0 < \delta < p^2 + q - 1$. Given a subpath S of a path P, let us denote by S_l (respectively, S_r) the subpath of P to the left (resp., right) of S, so that

$$P = S_l \circ S \circ S_r,$$

where $\circ$ denotes the concatenation operator. Informally, a collection of three paths P, P', and P'' (of possibly different lengths) is called a *λ-path triplet* if the inputs of those paths agree on some "middle" subpath of size at least λ, paths P and P'' coincide on their corresponding "left" parts, and paths P' and P'' coincide on their "right" parts. See Figure 2. Formally, a *λ-path triplet* is a triplet

$$\Big((P,S,\mathbf{x}),(P',S',\mathbf{x}'),(P'',S'',\mathbf{x}'')\Big)\ ,$$

such that (1) $|P|,|P'|,|P''| \geq \lambda$, (2) $\mathbf{x},\mathbf{x}',\mathbf{x}''$ are the respective inputs on these paths, and (3) $S \subset P$, $S' \subset P'$, $S'' \subset P''$ are three subpaths satisfying the following four constraints:

$$|S| = |S'| = |S''| \geq \lambda \qquad \mathbf{x}_S = \mathbf{x}'_{S'} = \mathbf{x}''_{S''} \qquad \mathbf{x}''_{S''_l} = \mathbf{x}_{S_l} \qquad \mathbf{x}''_{S''_r} = \mathbf{x}'_{S'_r}.$$

Let $\Big((P,S,\mathbf{x}),(P',S',\mathbf{x}'),(P'',S'',\mathbf{x}'')\Big)$ be a λ-path triplet. We have the following:

Claim 3.6. *If $\lambda \geq \ell(\delta)$, for ℓ as defined in Eq. (1), then*

$$(P,\mathbf{x}) \in \mathcal{L} \text{ and } (P',\mathbf{x}') \in \mathcal{L} \quad \Rightarrow \quad (P'',\mathbf{x}'') \in \mathcal{L}.$$

To establish the claim, consider an identity assignment Id'' for $(P'',\mathbf{x}'')$. Let Id and Id' be identity assignments for $(P,\mathbf{x})$, and $(P',\mathbf{x}')$, respectively, which agree with Id'' on the corresponding nodes. That is: (a) assignments Id, Id', and Id'' agree on the nodes in S, S' and S'', respectively; (b) Id and Id'' agree on the nodes in S_ℓ and S''_ℓ, respectively; and (c) Id and Id'' agree on the nodes in S'_r and S''_r, respectively. Since $(P,\mathbf{x}) \in \mathcal{L}$, and since $|S| = \lambda \geq \ell(\delta)$, it follows from Fact 2.2 that S contains an internal δ-secure subpath H. Then, let H' and H'' be the subpaths of P' and P'' corresponding to H. Since S and S'' coincide in their inputs and identity assignments, then H, H', H'' have the same t-neighborhood in P, P', P'' respectively. Hence, H'' is also a δ-secure (when running algorithm $\mathcal{A}$ in instance $(P'',\mathbf{x}'')$). Since both $(P,\mathbf{x})$ and $(P',\mathbf{x}')$ belong to $\mathcal{L}$, we have

$$\begin{aligned}
\Pr[\mathcal{E}(H''_\ell,\mathrm{Id}'',\mathbf{x}'') &= \Pr[\mathcal{E}(H_\ell,\mathrm{Id},\mathbf{x})])] \geq p, \text{ and} \\
\Pr[\mathcal{E}(H''_r,\mathrm{Id}'',\mathbf{x}'') &= \Pr[\mathcal{E}(H'_r,\mathrm{Id}',\mathbf{x}')])] \geq p.
\end{aligned}$$

Moreover, as $|H''| \geq 2t+1$, the two events $\mathcal{E}(H''_\ell,\mathrm{Id}'',\mathbf{x}'')$ and $\mathcal{E}(H''_r,\mathrm{Id}'',\mathbf{x}'')$ are independent. (Recall that these events are "all yes" events, as defined in Section 2.2.1). Hence

$$\Pr[\mathcal{E}(H''_\ell \cup H''_r,\mathrm{Id}'',\mathbf{x}'')] \geq p^2.$$

In other words, the probability that some node in $H''_\ell \cup H''_r$ says "no" is at most $1-p^2$. It follows, by union bound, that the probability that some node in H'' says "no" is at most $1 - p^2 + \delta < q$. Since $\mathcal{A}$ is a (p,q)-decider for $\mathcal{L}$, it cannot be the case that $(H'',\mathbf{x}'') \notin \mathcal{L}$. This establishes the proof of Claim 3.6.

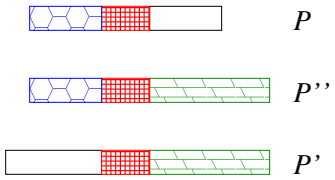

Figure 2: Example of a λ-path triplet (the central zone is of length at least λ).

We now observe that, without loss of generality, one can assume that in all instances $(P, \mathbf{x})$ of $\mathcal{L}$, the two extreme vertices of the path P have a special input symbol $\otimes$. To see why this holds, let $\otimes$ be a symbol not in Σ, and consider the following language $\mathcal{L}'$ defined over $\Sigma \cup \{\otimes\}$. Language $\mathcal{L}'$ consists of instances $(P, \mathbf{x})$ such that (1) the endpoints of P have input $\otimes$, and (2) $(P', \mathbf{x}') \in \mathcal{L}$, where P' is the path resulting from removing the endpoints of P, and where $\mathbf{x}'_v = \mathbf{x}_v$ for every node v of P'. Any (p, q) decider algorithm for $\mathcal{L}$ (resp., $\mathcal{L}'$), can be trivially transformed into a (p, q) decider algorithm for $\mathcal{L}'$ (resp., $\mathcal{L}$) with the same success guarantees and running time. Hence, in the remaining of the proof, we assume that in all instances $(P, \mathbf{x}) \in \mathcal{L}$, the two extreme vertices of the path P have input $\otimes$.

We say that a given instance $(P, \mathbf{x})$ is *extendable* if there exists an extension of it in $\mathcal{L}$, i.e., if there exists an instance $(P', \mathbf{x}') \in \mathcal{L}$ such that $P \subseteq P'$ and $\mathbf{x}'_P = \mathbf{x}$.

Claim 3.7. *There exists a (centralized) algorithm $\mathcal{X}$ that, given any configuration $(P, \mathbf{x})$ with $|P| \leq 2\ell(\delta) + 1$, decides whether $(P, \mathbf{x})$ is extendable.*

Indeed, observe that since the running time t of the (p, q)-decider $\mathcal{A}$ is constant, then $2\ell(\delta) + 1$ is also constant. Therefore, there are only finitely many configurations $(\hat{P}, \mathbf{x})$ with $|\hat{P}| \leq 2\ell(\delta) + 1$ (since Σ is finite). Call this set of configurations $\mathcal{C}$. Each of the configurations in $\mathcal{C}$ is either extendable or not. Hence, there exists a function $f : \mathcal{C} \to \{0, 1\}$ such that for every configuration $C \in \mathcal{C}$, $f(C) = 1$ if and only if C is extendable. This function f can be described in a finite manner, and hence gives rise to an algorithm as required by the claim.

We may assume, hereafter, that such an algorithm $\mathcal{X}$, as promised by Claim 3.7, is part of the language specification given to the nodes. Each node then can verify by a *local* computation if the instance restricted to its $\ell(\delta)$ neighborhood is extendable. We show that $\mathcal{L} \in \mathrm{LD}(O(t))$ by proving the existence of a deterministic algorithm $\mathcal{D}$ that recognizes $\mathcal{L}$ in $O(t)$ rounds. Given a path P, an input $\mathbf{x}$ over P, and an identity assignment Id, algorithm $\mathcal{D}$ applied at a node u of P operates as follows. If $\mathbf{x}_u = \otimes$ then u outputs "yes" if and only if u is an endpoint of P. Otherwise, i.e., if $\mathbf{x}_u \neq \otimes$, then u outputs "yes" if and only if $(B_u, \mathbf{x}_{B_u})$ is extendable (using algorithm $\mathcal{X}$), where $B_u = B(u, \ell(\delta))$ is the ball centered at u, and of radius $\ell(\delta)$ in P.

Algorithm $\mathcal{D}$ is a deterministic algorithm that runs in $\ell(\delta)$ rounds and sends constant size messages. We claim that Algorithm $\mathcal{D}$ recognizes $\mathcal{L}$. To establish that claim, consider first an instance $(P, \mathbf{x}) \in \mathcal{L}$. For every node u, $(P, \mathbf{x}) \in \mathcal{L}$ is an extension of $(B_u, \mathbf{x}_{B_u})$. Therefore, every node u outputs "yes", as desired. Now consider an instance $(P, \mathbf{x}) \notin \mathcal{L}$.

Assume, for the purpose of contradiction, that there exists an identity assignment Id such that, when applying $\mathcal{D}$ on $(P, \mathbf{x}, \mathrm{Id})$, every node u outputs "yes". We have the following:

Claim 3.8. $|P| > 2\ell(\delta) + 1$.

Indeed, assume for contradiction that $|P| \leq 2\ell(\delta) + 1$, and consider the middle node s of P. Since s outputs "yes", it follows that $(P, \mathbf{x})$ can be extended to $(P', \mathbf{x}')$ such that $(P', \mathbf{x}') \in \mathcal{L}$. However, since the extremities of P output "yes", it means that their input is $\otimes$. Therefore, as $|P'| > |P|$, we get that there is an internal node of P' which has input $\otimes$, contradicting $(P', \mathbf{x}') \in \mathcal{L}$.

We are now ready to complete the proof of Theorem 3.5. Let $S \subseteq P$ be the longest subpath of P such that there exists an extension $(P', \mathbf{x}')$ of $(S, \mathbf{x}_S)$, with $(P', \mathbf{x}') \in \mathcal{L}$. Since $|P| > 2\ell(\delta)+1$, and since the middle node of P outputs "yes", we have $|S| \geq 2\ell(\delta) + 1$. The proof carries on by distinguishing two cases for the length of S.

If $S = P$, then $(P, \mathbf{x})$ can be extended to $(P', \mathbf{x}') \in \mathcal{L}$. By the same arguments as above, since each extremity w of P has input $\otimes$, we conclude that $P = P'$, with $\mathbf{x} = \mathbf{x}'$. Contradicting the fact that $(P, \mathbf{x}) \notin \mathcal{L}$. Therefore $2\ell(\delta)+1 \leq |S| < |P|$. Let a and b be such that $S = [a, b]$. As S is shorter than P, it is impossible for both a and b to be endpoints of P. Without loss of generality, assume that a is not an endpoint of P. Since a outputs "yes", there exists an extension $(P'', \mathbf{x}'') \in \mathcal{L}$ of $(B_a, \mathbf{x}_{B_a})$. In fact, $(P'', \mathbf{x}'')$ is also an extension of $\mathbf{x}_{[a,a+\ell(\delta)]}$. Since $\mathbf{x}'$ and $\mathbf{x}''$ agree on $[a, a + \ell(\delta)]$, and since both $(P', \mathbf{x}')$, and $(P'', \mathbf{x}'')$ are in $\mathcal{L}$, we get from Lemma 3.6 that $\mathbf{x}_{[a-1,b]}$ can be extended to an input $(P''', \mathbf{x}''') \in \mathcal{L}$, which contradicts the choice of S. The theorem follows. □

4 The B_k Hierarchy is Strict

In this section we show that the classes B_k, $k \geq 1$, form an infinite hierarchy of distinct classes, thereby proving that the general ability to boost the probability of success for a randomized decision problem is quite limited. In fact, we show separation in a very strong sense: there are decision problems in $B_{k+1}(0)$, i.e., that have a (p, q)-decider running in zero rounds with $p^{1+1/(k+1)} + q > 1$, which cannot be decided by a (p, q)-decider with $p^{1+1/k} + q > 1$, even if the number of rounds of the latter is as large as $n^{1-\varepsilon}$ for every fixed $\varepsilon > 0$.

Theorem 4.1. *$B_{k+1}(0) \setminus B_k(t) \neq \emptyset$ for every $k \geq 1$ and every $t = o(n)$.*

Proof. Let k be any positive integer. We consider the following distributed language AMOS-k as defined in Section 2.2.3. In order to prove Theorem 4.1, we show that AMOS-$k \in B_{k+1}(0) \setminus B_k(t)$ for every $t = o(n)$.

We first establish that AMOS-k belongs to $B_{k+1}(0)$. The proof is similar to the proof of Claim 3.4. Specifically, we adapt algorithm $\mathcal{A}$ presented in [13] for AMOS to the case of AMOS-k. The following simple randomized algorithm runs in 0 time: every node v which is not selected, i.e., such that $\mathbf{x}(v) = 0$, says "yes" with probability 1; and every node which is selected, i.e., such that $\mathbf{x}(v) = 1$, says "yes" with probability $p^{1/k}$, and "no" with probability $1 - p^{1/k}$. If the graph has $s \leq k$ nodes selected, then all nodes say "yes" with probability $p^{s/k} \geq p$, as desired. On the other hand, if there are $s \geq k + 1$ selected nodes, then at least one node says "no" with probability

$$1 - p^{s/k} \geq 1 - p^{(k+1)/k} = 1 - p^{1+1/k}.$$

We therefore get a (p, q)-decider with $p^{1+1/k} + q \geq 1$, that is, such that $p^{1+1/(k+1)} + q > 1$. Thus AMOS-$k \in B_{k+1}(0)$.

We now consider the harder direction, and prove that AMOS-$k \notin B_k(t)$, for any $t = o(n)$. To prove this separation, it is sufficient to consider AMOS-k restricted to the family of n-node paths. Fix a function $t = o(n)$, and assume, towards contradiction, that there exists a distributed (p, q)-decider $\mathcal{A}$ for AMOS-k that runs in $O(t)$ rounds, with $p^{1+1/k} + q > 1$. Let $\varepsilon \in (0, 1)$ be such that $p^{1+1/k+\varepsilon} + q > 1$. Let P be an n-node path, and let $S \subset P$ be a subpath of P. Let $\delta \in [0, 1]$ be a constant satisfying

$$0 < \delta < p^{1+1/k} \left(1 - p^{\varepsilon}\right) / k \ . \tag{3}$$

Consider a positive instance and a negative instance of AMOS-k, respectively denoted by $I = (P, \mathbf{x})$ and $I' = (P, \mathbf{x}')$. Both instances are defined on the same n-node path P, where $n \geq k\left(\ell(\delta) + 1\right) + 1$. Recall that $\ell(\delta) = \ell(\delta, 2t + 1)$ (see Eq. (1)). We consider executions of $\mathcal{A}$ on these two instances, where nodes are given the same identities. Both instances have almost the same input. In particular, the only difference is that instance I contains k selected nodes, whereas I' has the same selected nodes as I plus one additional selected node. Therefore I is legal, while I' is illegal. In I', the path P is composed of $k + 1$ sections, each containing a unique selected node, and where each pair of consecutive sections separated by a δ-secure subpath. More precisely, let us enumerate the nodes of P from 1 to n, with node v adjacent to nodes $v - 1$ and $v + 1$, for every $1 < v < n$. Consider the k subpaths of P defined by:

$$S_i = [(i-1)\ell(\delta) + i + 1, i \cdot \ell(\delta) + i]$$

for $i = \{1, \ldots, k\}$. Let the selected nodes in I' be positioned as follows. Let $u_1 = 1$ and let $u_i = (i-1)\ell(\delta) + i$ for $i = 2, \ldots, k + 1$. Then set

$$\mathbf{x}'(v) = \begin{cases} 1, & \text{if } v = u_i \text{ for some } i \in \{1, ..., k+1\} \\ 0, & \text{otherwise.} \end{cases}$$

See Fig. 3(a) for a schematic representation of I'.

Our next goal is to define the legal instance $I = (P, \mathbf{x})$. To do so, we begin by claiming that each S_i contains a δ-secure internal subpath $S'_i = [a_i, b_i]$. Naturally, we would like to employ Fact 2.2. However, Fact 2.2 refers to subpaths of *valid* instances $(P, \mathbf{x}) \in \mathcal{L}$, and I' is illegal. So instead, let us focus on the instance $(S_i, \mathbf{x}'_{S_i})$. Since $(S_i, \mathbf{x}'_{S_i})$ contains no leaders, $\|\mathbf{x}'_{S_i}\|_1 = 0$, it follows that $(S_i, \mathbf{x}'_{S_i}) \in \mathcal{L}$, and Fact 2.2 can be applied on it. Subsequently, since $|S_i| > \ell(\delta)$ it follows that S_i contains an internal δ-secure subpath $S'_i = [a_i, b_i]$, whose t neighborhood is strictly in S_i. Therefore, when applying algorithm $\mathcal{A}$ on $(S_i, \mathbf{x}'_{S_i}, \mathrm{Id}_{S_i})$ and on $(P, \mathbf{x}', \mathrm{Id})$, the nodes in the $(2t + 1)$-length segment S'_i behave the same, thus

$$\Pr[\mathcal{E}(P, \mathbf{x}', \mathrm{Id}, V(S'_i))] = \Pr[\mathcal{E}(S_i, \mathbf{x}'_{S_i}, \mathrm{Id}_{S_i}, V(S'_i))].$$

Hence, S'_i is a δ-secure subpath in $(P, \mathbf{x}', \mathrm{Id})$ as well for every $i \in \{1, ..., k\}$, see Fig. 3(b).

The δ-secure subpaths S'_i's are now used to divide P into $2k+1$ segments. Specifically, there are $k+1$ segments T_i, $i = 1, \ldots, k+1$, each with one selected node. The δ-secure subpaths $S'_i = [a_i, b_i]$ separate T_i from T_{i+1}. More precisely, set $T_1 = [1, a_1 - 1]$, $T_i = [b_{i-1} + 1, a_i - 1]$ for $i \in 2, ..., k$, and $T_{k+1} = [b_k + 1, n]$, getting

$$P = T_1 \circ S'_1 \circ T_2 \circ S'_2 \circ \ldots \circ T_k \circ S'_k \circ T_{k+1}$$

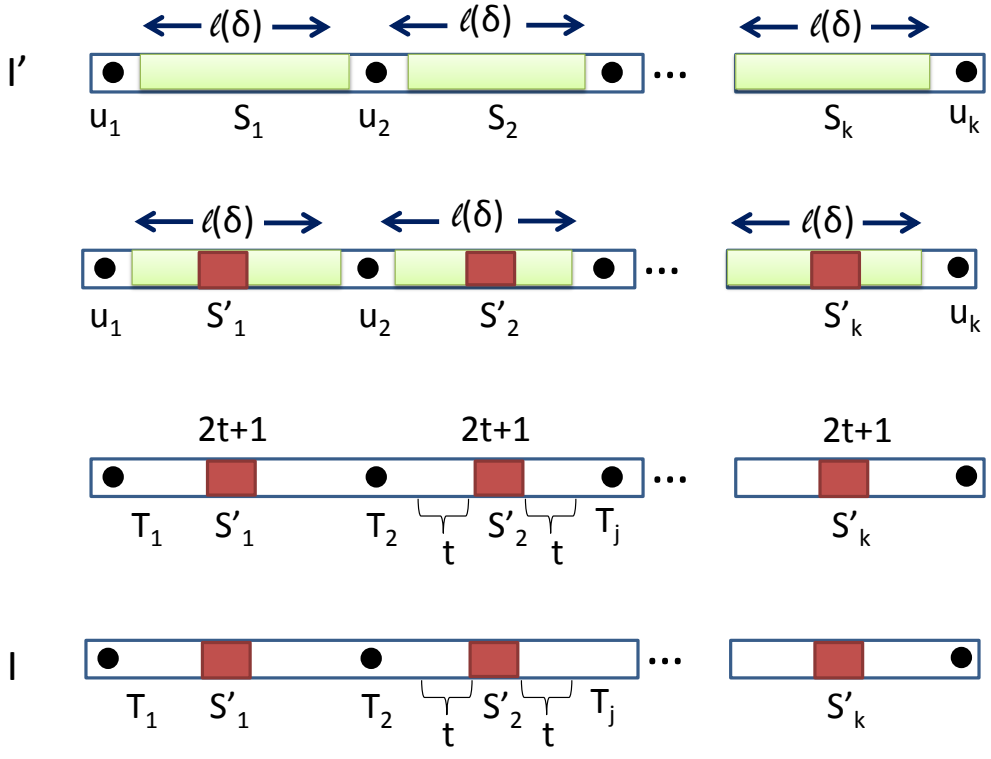

Figure 3: Illustration of the constructions for Theorem 4.1. (a) The instance $I' = (P, \mathbf{x}')$ with $k+1$ leaders (indicated by black circles) separated by $\ell(\delta)$-length segments, S_i. (b) The δ-secure subpaths S'_i in each S_i are internal to S_i. (c) The leader-segments T_i interleaving with δ-secure subpaths S'_i. (d) The legal instance $I = (P, \mathbf{x})$, the j^{th} leader of I' is discarded, resulting in a k leader instance.

where $\circ$ denotes path concatenation. Let $\mathcal{T}_i = \mathcal{E}(P, \mathbf{x}', \mathrm{Id}, V(T_i))$ be the event that all nodes in the subpath T_i say "yes" in the instance I', for $i \in \{1, ..., k+1\}$ and let $p_i = \Pr[\mathcal{T}_i]$ be its probability. Let j be such that $p_j = \max_i p_i$. We now define the valid instance $I = (P, \mathbf{x})$:

$$\mathbf{x}(v) = \begin{cases} 1, & \text{if } v = u_i \text{ for some } i \in \{1, ..., k+1\},\ i \neq j, \\ 0, & \text{otherwise.} \end{cases}$$

Note that $\|\mathbf{x}'\|_1 = k+1$ and $\|\mathbf{x}\|_1 = k$, thus $I \in$ AMOS-k while $I' \notin$ AMOS-k. See Fig. 3(c,d) for an illustration of I versus I'.

We now make the following observation.

Claim 4.2. $\forall i \neq j,\ \Pr[\mathcal{E}(P, \mathbf{x}, \mathrm{Id}, V(T_i))] = p_i$.

This follows since the distance between any two nodes u (resp., v) in distinct $T'_i s$ is greater than t, which implies that $\mathbf{x}_{L_i \circ T_i \circ R_i} = \mathbf{x}'_{L_i \circ T_i \circ R_i}$ where L_i (resp., R_i) is the subpath of length t to

the left (resp., to the right) of T_i in P, from which it follows that under $\mathcal{A}$ the nodes of T_i have the same behavior in both instances I and I'.

Now, let $\mathcal{N}$ (resp., $\mathcal{N}'$) be the event that there exists at least one node in I (resp., I') that says "no" when applying algorithm $\mathcal{A}$. Similarly, let $\mathcal{Y}$ (resp., $\mathcal{Y}'$) be the event stating that all nodes in the configuration I (resp., I') say "yes". Let

$$\mathcal{T} = \bigcup_{i=1}^{k+1} \mathcal{T}_i$$

be the event that all nodes in each subpaths T_i, for $i \in \{1, ..., k+1\}$ say "yes" in the instance I'. For every $i \in \{1, ..., k\}$, let $\mathcal{S}_i = \mathcal{E}(P, \mathbf{x}', \mathrm{Id}, V(S_i'))$ be the event that all nodes in the δ-secure subpath S_i' say "yes" in the instance I'. We have

$$\Pr(\mathcal{Y}) = \Pr[\mathcal{E}(P, \mathbf{x}, \mathrm{Id}, V(P))] \text{ and } \Pr(\mathcal{Y}') = \Pr[\mathcal{E}(P, \mathbf{x}', \mathrm{Id}, V(P)),$$

while $\Pr(\mathcal{N}) = 1 - \Pr(\mathcal{Y})$ and $\Pr(\mathcal{N}') = 1 - \Pr(\mathcal{Y}')$.

Since $\mathcal{A}$ is a (p, q)-decider, as we assume by contradiction that AMOS-k in B_k, we have $\Pr(\mathcal{N}') \geq q$, and thus $\Pr(\mathcal{N}') > 1 - p^{1+1/k+\varepsilon}$. Therefore, $\Pr(\mathcal{Y}') < p^{1+1/k+\varepsilon}$. Moreover, since $I \in$ AMOS-k, we also have that $\Pr(\mathcal{Y}) \geq p$. Therefore, the ratio $\hat{\rho} = \Pr(\mathcal{Y}')/\Pr(\mathcal{Y})$ satisfies

$$\hat{\rho} < p^{1/k+\varepsilon} \ . \tag{4}$$

On the other hand, note that by applying the union bound on the complements of the $k+1$ events $\mathcal{T}, \mathcal{S}_1, \ldots, \mathcal{S}_k$, we get

$$\Pr(\mathcal{N}') \quad \leq \quad (1 - \Pr[\mathcal{T}]) + \left(\sum_{i=1}^{k} (1 - \Pr[\mathcal{S}_i]) \right) \leq 1 - p_j \cdot \prod_{i \neq j} p_i + k \cdot \delta,$$

where the last inequality follows by the fact that each S_i' is a $(\delta, 2t+1)$-secure subpath, thus the events $\mathcal{T}_{i_1}, \mathcal{T}_{i_2}$ are independent for every $i_1, i_2 \in \{1, ..., k+1\}$ (since the distance between any two nodes $u \in T_{i_1}$ and $v \in T_{i_2}$ is at least $2t+1$). This implies that

$$\Pr(\mathcal{Y}') \geq p_j \cdot \prod_{i \neq j} p_i - k \cdot \delta \ .$$

Since $\Pr(\mathcal{Y}) \leq \prod_{i \neq j} p_i$ (by the independence of the events $\mathcal{T}_{i_1}, \mathcal{T}_{i_2}$, for every $i_1, i_2 \in \{1, ...k+1\}$), it then follows that the ratio $\hat{\rho}$ satisfies

$$\hat{\rho} \quad \geq \quad \frac{p_j \cdot \prod_{i \neq j} p_i - k \cdot \delta}{\prod_{i \neq j} p_i} \geq p_j - \frac{k \cdot \delta}{\prod_{i \neq j} p_i} \geq p_j - k \cdot \delta/p \ , \tag{5}$$

where the last inequality follows by the fact that $I \in$ AMOS-k and thus $\prod_{i \neq j} p_i \geq \Pr(\mathcal{Y}) \geq p$. Finally, note that $p_j \geq p^{1/k}$. This follows since $p_j \geq p_i$ for every $i \in \{1, ..., k+1\}$, so

$$p_j^k \geq \prod_{i \neq j} p_i \geq p.$$

By Eq. (5), we then have that

$$\hat{\rho} \geq p^{1/k} - k \cdot \delta/p.$$

Combining this with Eq. (4), we get that $p^{1/k} - k \cdot \delta/p < p^{1/k+\varepsilon}$, which is in contradiction to the definition of δ in Eq. (3). □

We conclude this section by showing that the integrality of k in the B_k hierarchy does not depend upon the fact that the inputs or the number k of allowed leaders are integers. To see this, consider the fractional version of AMOS-k, namely, the AMOS-f language, where the input $\mathbf{x}(v)$ of node v is either 0 (implying that v is not selected) or some fractional number

$$\mathbf{x}(v) \in \{1/c, 2/c, \ldots, (c-1)/c, 1\}$$

for some integer c, indicating its "weighted leadership," and $f \in \mathbb{R}_{>0}$ is a rational number representing the maximum allowed total weight of selected nodes. The language AMOS-f (for At-Most-f-Selected) is then defined by

$$\texttt{AMOS-}f = \{(G, \mathbf{x}) \text{ s.t. } \mathbf{x}(v) \in \{0, 1/c, 2/c, \ldots, (c-1)/c, 1\} \text{ and } \|\mathbf{x}\|_1 \leq f\}.$$

Let k' be the maximal integer such that $k'/c \leq f$. Then the following holds.

Lemma 4.3. *AMOS-$f \in B_{k'+1}(0) \setminus B_{k'}(t)$ for every $t = o(n)$.*

To see why, consider the language of AMOS*-k, in which k is an integral number but the inputs of the selected nodes are in $\mathbb{R}_{\geq 1}$. That is,

$$\texttt{AMOS}^*\texttt{-}k = \{(G, \mathbf{x}) \text{ s.t. } k \in \mathbb{Z}_{\geq 1},\ \mathbf{x}(v) \in \{0\} \cup \mathbb{R}_{\geq 1} \text{ and } \|\mathbf{x}\|_1 \leq k\}.$$

In the proof of Thm. 4.1 we showed that AMOS-$k \in B_{k+1}(0) \setminus B_k(t)$. It can be easily verified that it also holds that

$$\texttt{AMOS}^*\texttt{-}k \in B_{k+1}(0) \setminus B_k(t) \text{ for every } t = o(n). \tag{6}$$

Finally, note that there is a local reduction from AMOS*-k to AMOS-f and vice versa. In particular, the following equivalence relation can be shown.

Claim 4.4. *$(G, \mathbf{x}) \in$ AMOS-f if and only if $(G, c \cdot \mathbf{x}) \in$ AMOS*-k' where $k'/c \leq f$.*

For the "if" direction, let $(G, \mathbf{x}) \in$ AMOS-f. By the definition of k' and the fact that all inputs are multiplicities of $1/c$, it holds that $\sum_v \mathbf{x}(v) \leq k'/c$ and hence $\sum_v c \cdot \mathbf{x}(v) \leq k'$, concluding that $(G, c \cdot \mathbf{x}) \in$ AMOS*-k'. Conversely, let $(G, \mathbf{x}) \in$ AMOS*-k'. Then $\sum_v \mathbf{x}(v)/c \leq k'/c \leq f$, implying that $(G, \mathbf{x}/c) \in$ AMOS-f as required.

Lemma 4.3 follows by Eq. (6) and Claim 4.4.

5 Existence of languages outside the B_k hierarchy

In this section we show that the $B_k(t)$ hierarchy does not capture all languages even for $k = \infty$ and t as large as $o(n)$.

Theorem 5.1. *There is a language not in $B_\infty(t)$, for every $t = o(n)$.*

Proof. We exhibit one specific language not in $B_\infty(t)$, for every $t = o(n)$. This language consists of determining whether the underlying network is acyclic. Specifically, let

$$\texttt{Tree} = \{(G, \epsilon) \mid G \text{ is a tree}\},$$

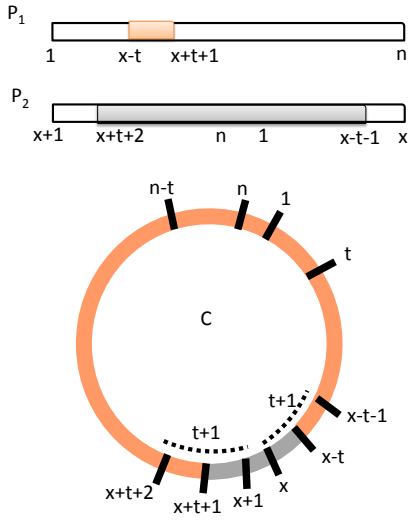

Figure 4: Illustration of the constructions for Theorem 5.1. Shown are paths P_1,P_2 and cycle C. When applying Algorithm $\mathcal{A}$ on path P_1 (respectively, P_2) and on cycle C, the nodes in the segment $[x-t, x+t+1]$ (resp., $[x+t+2, \ldots, 1, \ldots, x+t+1]$) behave the same.

where ϵ is the null input. Fix a function $t = o(n)$. Assume, towards contradiction, that there exists some finite k such that $\texttt{Tree} \in B_k(t)$. Then there is a (p,q)-decider for $\texttt{Tree}$, given by $\mathcal{A}$, running in t rounds, with $p^{1+1/k} + q - 1 > 0$. Hence, in particular, there exists some $\varepsilon > 0$ such that $p + q - 1 > \varepsilon$. Fix $\delta > 0$ such that

$$\delta = \varepsilon < p + q - 1. \tag{7}$$

We consider graphs G of size $n > \left\lceil \frac{21 \cdot \log p}{\log(1-\delta)} \right\rceil$. We will show that $\texttt{Tree} \notin B_k(t)$ for any

$$t \leq \left\lfloor \frac{\log(1-\delta)}{21 \cdot \log p} \right\rfloor \cdot n. \tag{8}$$

Consider the cycle C with n nodes labeled consecutively from 1 to n, and the path P with nodes labeled consecutively from 1 to n. This labeling defines the identity assignment Id^1. In the input configuration (P, ϵ), the probability that all nodes say "yes" when executing $\mathcal{A}$ is at least p. Let us identify a subpath $S = [x-t, \ldots, x+t+1]$ of P to be used as an internal $(\delta, 2(t+1))$-secure subpath in P. I.e.,

$$\Pr[\mathcal{E}(P, \epsilon, \mathrm{Id}^1, V(S))] \geq 1 - \delta \ . \tag{9}$$

Note that, by Eq. (1, 7, 8), it follows that $n > \ell(\delta, 2(t+1))$. Hence by Fact 2.2, since $(P, \epsilon) \in \mathcal{L}$, there exists such internal subpath $S \subset P$. Consider the event $\mathcal{E}(P, \mathbf{x}, \mathrm{Id}, V(S))$ stating that all nodes in subpath S of P with input $\mathbf{x}$ and identity-assignment Id return "yes". We have that

$$\Pr[\mathcal{E}(P, \epsilon, \mathrm{Id}^1, V(S))] = \Pr[\mathcal{E}(C, \epsilon, \mathrm{Id}^1, V(S))] \ . \tag{10}$$

Consider a subpath $\widehat{S}$ composed of the subpath S padded with a block L of t nodes before it and a block R of t nodes after it. Indeed, since S is an internal subpath of P (i.e., it is at distance at least $t+1$ from P's endpoints), the set of nodes of $\widehat{S} = L \circ S \circ R = [x-2t, \ldots, x+2t+1]$ appears consecutively in both P and in C with identity assignment Id^1, and $\widehat{S}$ have the same identities (with Id^1) and degrees in both C and P. We now consider another identity-assignment Id^2 for P, with nodes labeled consecutively from $x+1$ to n, and then from 1 to x. Consider the $(n-2(t+1))$-node subpath

$$S' = [x+t+2, \ldots, n, 1, \ldots, x-t-1] \ .$$

We have

$$\Pr[\mathcal{E}(P, \epsilon, \mathrm{Id}^2, V(S'))] = \Pr[\mathcal{E}(C, \epsilon, \mathrm{Id}^1, V(S'))] \ . \tag{11}$$

Consider a subpath $\widehat{S'}$ composed of the subpath S' padded with a block L' of t nodes before it and a block R' of t nodes after it, i.e., $\widehat{S'} = L' \circ S' \circ R'$. Indeed, the set of $\widehat{S'}$ nodes appears consecutively in both C and P with identity assignment Id^2, and $L' \circ S' \circ R'$ have the same identities (with Id^2) and degrees in both C and P, where L' (resp., R') is the subpath composed of the t nodes with identities Id^2 immediately larger than $x+t+1$ (resp., smaller than $x-t$). Formally, we have that $\mathrm{Id}^1_{\widehat{S'}} = \mathrm{Id}^2_{\widehat{S'}}$. See Fig. 4 for illustration. Let $\mathcal{S} = \mathcal{E}(C, \epsilon, \mathrm{Id}^1, V(S))$, (resp., $\mathcal{S}' = \mathcal{E}(C, \epsilon, \mathrm{Id}^1, V(S'))$) be the event that all nodes of $S \subset C$ (resp., $S' \subset C$) say "yes". We can now combine these previous results to derive a contradiction. Since $C \notin \texttt{Tree}$, by applying the union bound on the complements of the events $\mathcal{S}$ and $\mathcal{S}'$, and using Eq. (10) and (11), we get that

$$\begin{aligned} q &\leq 1 - \Pr[\mathcal{E}(C, \epsilon, \mathrm{Id}^1, V(C))] \leq \big(1 - \Pr[\mathcal{S}']\big) + (1 - \Pr[\mathcal{S}]) \\ &= \big(1 - \Pr[\mathcal{E}(P, \epsilon, \mathrm{Id}^2, V(S'))]\big) + \big(1 - \Pr[\mathcal{E}(P, \epsilon, \mathrm{Id}^1, V(S))]\big) \\ &\leq \big(1 - \Pr[\mathcal{E}(P, \epsilon, \mathrm{Id}^2, V(S'))]\big) + \delta \end{aligned}$$

where the last inequality holds by Eq. (9). Therefore we get

$$q \leq 1 - p + \delta \ ,$$

by noticing that $\Pr[\mathcal{E}(P, \epsilon, \mathrm{Id}^2, V(S'))] \geq p$ since $P \in \texttt{Tree}$. Finally, by Eq. (7), we eventually get $q < 1 - p + p + q - 1$ or $q < q$, contradiction. □

6 On the ability of boosting within B_k

Theorem 4.1 demonstrates that boosting the probability of success cannot take any language from B_k to B_{k-1} for an integral positive k. That is, boosting might still be doable, but if so, only within a class B_k: from (p, q) satisfying $p^{1+1/(k+1)} + q > 1$ to (p, q) satisfying $p^{1+1/k} + q > 1$. We believe that the separation holds for any positive *real* number r implying that the B_k hierarchy holds for a smooth spectrum of all positive reals. In the remark at the end of the current section we present a candidate language that might establish this separation.

In this section we prove a weaker separation result. Specifically, we prove that once the inputs can be restricted in certain ways, the ability to boost the success probability becomes almost null. More precisely, recall that so far we considered languages as collections of pairs $(G, \mathbf{x})$ where G is a (connected) n-node graph and $\mathbf{x} \in \Sigma^n$ is the input vector to the nodes of G, in some finite of

infinite alphabet Σ, that is, $\mathbf{x}(v) \in \Sigma$ for all $v \in V(G)$. An instance of an algorithm $\mathcal{A}$ deciding a language $\mathcal{L}$ was defined as *any* such pair $(G, \mathbf{x})$.

We now consider the case where the set of instances is restricted to some specific subset of inputs $\mathcal{I} \subset \Sigma^n$. That is, the distributed algorithm $\mathcal{A}$ has now the *promise* that in the instances $(G, \mathbf{x})$ admissible as inputs, the input vector $\mathbf{x}$ is restricted to $\mathbf{x} \in \mathcal{I} \subset \Sigma^n$.

We define the classes $C_r(t)$ in a way identical to the classes $B_k(t)$, but generalized in two ways. First, the parameter r is not required to be integral, but can be any positive real. Second, the decision problems under consideration are extended to the ones in which the set of input vectors $\mathbf{x}$ can be restricted. So, in particular, $B_k(t) \subseteq C_k(t)$, for every positive integer k, and every function t. The following theorem proves that boosting can made as limited as desired.

Theorem 6.1. *Let $r < r'$ be any two positive reals. Then, $C_{r'}(0) \setminus C_r(t) \neq \emptyset$ for every $t = o(n)$.*

Proof. Let $\widehat{r} = a/b \in [r, r')$ be a positive rational where a and b are two co-prime integers. By the density of the rational numbers, such $\widehat{r}$ is guaranteed to exist.

The promise: To establish the theorem, we consider the language AMOS-a restricted to instances in $\mathcal{I}$, where

$$\mathcal{I} = \{\mathbf{x} \in \{0,1\}^* : \ \|\mathbf{x}\|_1 \ \notin [a+1, a+b-1]\}.$$

In other words, the promise says that an input either satisfies AMOS-a, or is far from satisfying AMOS-a (very many selected nodes). Note that this promise reminds the requirements of the language leader(a, b). One important difference, however, is that Language leader(a, b) is restricted to instances having the particular structure $G_{k,m}$.

Our goal is to prove that AMOS-$a \in C_{r'}(0) \setminus C_r(t)$ for every $t = o(n)$. We begin by showing that AMOS-$a \in C_{r'}(0)$ by considering the following version of Algorithm $A_{\texttt{lead}}$ described in the proof of Claim 3.4. The algorithm is a simple randomized algorithm that runs in 0 time: every node v which is not selected, i.e., such that $\mathbf{x}(v) = 0$, says "yes" with probability 1; and every node which is selected, i.e., such that $\mathbf{x}(v) = 1$, says "yes" with probability $p^{1/a}$, and "no" with probability $1 - p^{1/a}$. If the graph has $s \leq a$ nodes selected, then all nodes say "yes" with probability $p^{s/a} \geq p$, as desired. Else, there are $s \geq a + b$ leaders, (this follows from the promise), and at least one node says "no" with probability $1 - p^{s/a} \geq 1 - p^{(a+b)/a} = 1 - p^{1+1/\widehat{r}}$. We therefore get a (p, q)-decider with $p^{1+1/\widehat{r}} + q \geq 1$, thus $p^{1+1/r'} + q > 1$ as $r' > \widehat{r}$. It therefore follows that AMOS-$a \in C_{r'}(0)$.

We now consider the harder direction, and prove that AMOS-$a \notin C_r(t)$, for any $t = o(n)$. Since $\widehat{r} \geq r$, it is sufficient to show that AMOS-$a \notin C_{\widehat{r}}(t)$. To prove this separation, consider the AMOS-a problem restricted to the family of n-node paths. Fix a function $t = o(n)$, and assume, towards contradiction, that there exists a distributed (p, q)-decider $\mathcal{A}$ for AMOS-a that runs in $O(t)$ rounds, with $p^{1+1/\widehat{r}} + q > 1$. Let $\varepsilon \in (0, 1)$ be such that $p^{1+1/\widehat{r}+\varepsilon} + q > 1$. Let P be an n-node path, and let $S \subset P$ be a subpath of P. Let $\delta \in [0, 1]$ be a constant satisfying

$$0 < \delta < p^{1+1/\widehat{r}} \left(1 - p^{\varepsilon}\right) / (a + b - 1). \tag{12}$$

Consider a positive instance and a negative instance of AMOS-a, respectively denoted by

$$I = (P, \mathbf{x}) \text{ and } I' = (P, \mathbf{x}').$$

Both instances are defined on the same n-node path P, where

$$n \geq (a+b-1)\,(\ell(\delta)+1)+1.$$

where $\ell(\delta) = \ell(\delta, 2t+1)$, as defined by Eq. (1). We consider executions of $\mathcal{A}$ on these two instances, where nodes are given the same id's. Both instances have almost the same input. In particular, the only difference is that instance I contains a selected nodes, whereas I' has the same selected nodes as I plus b additional selected nodes. Therefore I is legal, while I' is illegal. In addition, both inputs $\mathbf{x}$ and $\mathbf{x}'$ satisfy the promise.

In I', the path P is composed of $a+b$ sections, each containing a unique selected node, and where each pair of consecutive sections separated by δ-secure subpaths. More precisely, let us enumerate the nodes of P from 1 to n, with node v adjacent to nodes $v-1$ and $v+1$, for every $1 < v < n$. Consider the $a+b-1$ subpaths of P defined by:

$$S_i = [(i-1)\ell(\delta) + i + 1, i \cdot \ell(\delta) + i]$$

for $i = \{1, \ldots, a+b-1\}$. Let the selected nodes in I' be positioned as follows. Let $u_1 = 1$ and let $u_i = (i-1)\ell(\delta) + i$ for $i = 2, \ldots, a+b$. Then set

$$\mathbf{x}'(v) = \begin{cases} 1 & \text{if } v = u_i \text{ for some } i \in \{1, ..., a+b\} \\ 0 & \text{otherwise.} \end{cases}$$

Our next goal is to construct a legal input $I = (P, \mathbf{x})$ with a leaders. Towards this, we begin by showing that each S_i contains a δ-secure internal subpath $S'_i = [x_i, y_i]$ (internal to S_i). Note that Fact 2.2 refers to subpaths in *valid* instances $(P, \mathbf{x}) \in \mathcal{L}$, and since I' is illegal it cannot be directly applied. So instead, let us focus on the instance $(S_i, \mathbf{x}'_{S_i})$ with IDs Id_{S_i}. Since S_i contains no leaders, $\|\mathbf{x}'_{S_i}\|_1 = 0$, it follows that $(S_i, \mathbf{x}'_{S_i}) \in \mathcal{L}$. Now we can safely apply Fact 2.2. Indeed, since $|S_i| > \ell(\delta)$ it follows by the fact that S_i contains an internal δ-secure subpath $S'_i = [x_i, y_i]$. Therefore, when applying algorithm $\mathcal{A}$ on $(S_i, \mathbf{x}'_{S_i}, \mathrm{Id}_{S_i})$ and on $(P, \mathbf{x}', \mathrm{Id})$, the nodes of S'_i behave the same, thus $\Pr[\mathcal{E}(P, \mathbf{x}', \mathrm{Id}, V(S'_i))] = \Pr[\mathcal{E}(S_i, \mathbf{x}'_{S_i}, \mathrm{Id}_{S_i}, V(S'_i))]$. Hence, S'_i is a δ-secure subpath in I' as well, for every $i \in \{1, ..., a+b-1\}$.

The δ-secure subpaths S'_i are used to divide P into $2(a+b-1)+1$ segments. There are $a+b$ segments T_i, $i = 1, \ldots, a+b$, each with one selected nodes. The δ-secure subpaths $S'_i = [x_i, y_i]$ separate T_i from T_{i+1}. More precisely, we set

$$T_1 = [1, x_1 - 1],\ T_i = [y_{i-1} + 1, x_i - 1]$$

for $i \in 2, ..., a+b-1$, and $T_{a+b} = [y_{a+b} + 1, n]$, getting

$$P = T_1 \circ S'_1 \circ T_2 \circ S'_2 \circ \ldots \circ T_{a+b-1} \circ S'_{a+b-1} \circ T_{a+b}$$

where $\circ$ denotes path concatenation. For $i \in \{1, ..., a+b\}$, let $\mathcal{T}_i = \mathcal{E}(P, \mathbf{x}', \mathrm{Id}, V(T_i))$ be the event that all nodes in the subpath T_i say "yes" in instance I', and let $p_i = \Pr[\mathcal{T}_i]$ its probability. Let $J = \{j_1, \ldots, j_b\}$ be the set of b indices with maximal values in $\{p_1, \ldots, p_{a+b}\}$. I.e., $p_j \geq \max\{p_i \mid i \in \{1, ..., a+b\} \setminus J\}$ for every $j \in J$. We are now defining the valid instance $I = (P, \mathbf{x})$:

$$\mathbf{x}(v) = \begin{cases} 1 & \text{if } v = u_i \text{ for some } i \in \{1, ..., a+b\} \setminus J \\ 0 & \text{otherwise.} \end{cases}$$

We therefore have that $\|\mathbf{x}'\|_1 = a+b$ and $\|\mathbf{x}\|_1 = a$, thus $I \in$ `AMOS-`a while $I' \notin$ `AMOS-`a, and both I, I' satisfy the promise. We now make the following immediate observation.

Claim 6.2. $\Pr[\mathcal{E}(P, \mathbf{x}, \mathrm{Id}, V(T_i))] = p_i$, *for every* $i \notin J$.

This follows since the distance between any two nodes u (resp., v) in distinct $T_i's$ is greater than t, which implies that $\mathbf{x}_{L_i \circ T_i \circ R_i} = \mathbf{x}'_{L_i \circ T_i \circ R_i}$ where L_i (resp., R_i) is the subpath of length t to the left (resp., to the right) of T_i in P, from which it follows that under $\mathcal{A}$ the nodes of T_i have the same behavior in both instances I and I' for every $i \notin J$.

Let $\mathcal{N}$ (resp., $\mathcal{N}'$) be the event that there exists at least one node in I (resp., I') that says "no" when applying algorithm $\mathcal{A}$. Similarly, let $\mathcal{Y}$ (resp., $\mathcal{Y}'$) be the event that all nodes in the configuration I (resp., I') say "yes". Let $\mathcal{T} = \bigcup_{i=1}^{a+b} \mathcal{T}_i$ be the event that all nodes in the subpaths T_i, for $i \in \{1, ..., a+b\}$ say "yes" in the instance I'. For every $i \in \{1, ..., a+b-1\}$, let $\mathcal{S}_i = \mathcal{E}(P, \mathbf{x}', \mathrm{Id}, V(S'_i))$ be the event that all nodes in the δ-secure subpath S'_i say "yes" in the instance I'. We have

$$\Pr(\mathcal{Y}) = \Pr[\mathcal{E}(P, \mathbf{x}, \mathrm{Id}, V(P))]$$

and

$$\Pr(\mathcal{Y}') = \Pr[\mathcal{E}(P, \mathbf{x}', \mathrm{Id}, V(P))]$$

while

$$\Pr(\mathcal{N}) = 1 - \Pr(\mathcal{Y})$$

and

$$\Pr(\mathcal{N}') = 1 - \Pr(\mathcal{Y}').$$

Since $\mathcal{A}$ a (p, q)-decider, as we assume by contradiction that $\texttt{AMOS-}a \in B_k$, we have

$$\Pr(\mathcal{N}') \geq q,$$

and thus

$$\Pr(\mathcal{N}') > 1 - p^{1+1/\widehat{r}+\varepsilon}.$$

Therefore, $\Pr(\mathcal{Y}') < p^{1+1/\widehat{r}+\varepsilon}$. Moreover, since $I \in \texttt{AMOS-}a$, we also have that $\Pr(\mathcal{Y}) \geq p$. Therefore,

$$\frac{\Pr(\mathcal{Y}')}{\Pr(\mathcal{Y})} < p^{1/\widehat{r}+\varepsilon} \ . \tag{13}$$

On the other hand, by applying the union bound to the $a+b$ events $\mathcal{T}, \bigcup_{i=1}^{a+b-1} \mathcal{S}_i$, we get that

$$\begin{aligned} \Pr(\mathcal{N}') &\leq (1 - \Pr[\mathcal{T}]) + \sum_{i=1}^{a+b-1} (1 - \Pr[\mathcal{S}_i]) \\ &\leq 1 - (\prod_{i \notin J} p_i \cdot \prod_{j \in J} p_j) + (a+b-1) \cdot \delta, \end{aligned}$$

where the last inequality follows since S'_i's are δ-secure subpaths and hence the events $\mathcal{T}_i$'s are independent. We therefore get that

$$\Pr(\mathcal{Y}') \geq \left(\prod_{i \notin J} p_i \cdot \prod_{j \in J} p_j \right) - (a+b-1) \cdot \delta \ .$$

Since $\Pr(\mathcal{Y}) \leq \prod_{i \notin J} p_i$, it then follows that

$$\begin{aligned} \frac{\Pr(\mathcal{Y}')}{\Pr(\mathcal{Y})} &\geq \frac{\prod_{i \notin J} p_i \cdot \prod_{j \in J} p_j - (a+b-1) \cdot \delta}{\prod_{i \notin J} p_i} \\ &\geq \prod_{j \in J} p_j - \frac{(a+b-1) \cdot \delta}{\prod_{i \notin J} p_i}. \end{aligned}$$

Now, since $I \in \texttt{AMOS-}a$, we have $\prod_{i \notin J} p_i \geq \Pr(\mathcal{Y}) \geq p$, and thus

$$\frac{\Pr(\mathcal{Y}')}{\Pr(\mathcal{Y})} \geq \prod_{j \in J} p_j - \frac{(a+b-1) \cdot \delta}{p}. \tag{14}$$

Note that

$$\prod_{j \in J} p_j \geq p^{1/\widehat{r}}. \tag{15}$$

By the definition of J, $|J| = b$, and $\prod_{j \in J} p_j \geq p_i^b$ for every $i \notin J$. In addition, since there are a indices $i \notin J$, we get that $(\prod_{j \in J} p_j)^{a/b} \geq \prod_{i \notin J} p_i \geq \Pr(\mathcal{Y}) \geq p$. Combining with the definition of $\widehat{r}$, Eq. (15) follows. Hence, by Eq. (14), we get

$$\Pr(\mathcal{Y}')/\Pr(\mathcal{Y}) \geq p^{1/\widehat{r}} - \frac{(a+b-1) \cdot \delta}{p}.$$

Combining with Eq. (13) we get that

$$p^{1/\widehat{r}} - (a+b-1) \cdot \delta/p < p^{1/\widehat{r}+\varepsilon} \ ,$$

which is in contradiction to the definition of δ in Eq. (12). We therefore get that $\texttt{AMOS-}a \notin C_{\widehat{r}}(t)$, and since $r \leq \widehat{r}$, it also holds that $\texttt{AMOS-}a \notin C_r(t)$ as required. The theorem follows. □

Note that Theorem 6.1 demonstrates not only the (almost) inability of boosting the probability of success when the inputs to the nodes are restricted to specific kinds, but also the inability of derandomizing, even above the threshold $p^2 + q = 1$. Indeed, the following is a direct consequence of Theorem 6.1.

Corollary 6.3. *For every positive real r, there is a decision problem in $C_r(0)$ which cannot be decided* deterministically *in $o(n)$ rounds.*

Remark: As mentioned in the beginning of the section, we believe that the B_r classes are separated for every positive real number $r > 0$ even *without* a promise. That is, we believe that, given any two positive reals, $r < r'$, there exists a language that belongs to $B_{r'}(1)$ but does not belong to $B_r(t) \neq \emptyset$ for every $t = o(n)$. We present a candidate language for proving this claim. Let $\widehat{r} = a/b \in [r, r')$ be a positive rational where a and b are two co-prime integers. Our candidate language is $\texttt{leader}(a, b)$ (see Eq. (2) in Section 3.1). Since $r' > \widehat{r}$, it holds that $\texttt{leader}(a, b) \in B_{r'}(1)$. Hence, since $\widehat{r} \geq r$, it is sufficient to show that $\texttt{leader}(a, b) \notin B_{\widehat{r}}(t)$. Note that $\texttt{leader}(a, b)$ is restricted on instances of the structural form $G_{k,m}$, which can be verified in one round, and which allows to prove that

$\texttt{mod}(a, b) \in \text{LD}(1)$ (see Claim 3.2). Proving that $\texttt{leader}(a, b) \notin \text{LD}$ was done in Claim 3.3 by constructing instances $I_0, I_1, \cdots$, over the same graph $G_{k,m}$. Here, to prove the desired statement, a natural attempt would be based on imitating the proof of Theorem 6.1 by considering graphs of the form $G_{k,m}$ while viewing the outer cycle C_1 as an analogue for the path in Theorem 6.1. However, the proof of Theorem 6.1 suggests that we may have to glue parts of this cycle, requiring carefully gluing parts of $G_{k,m}$ to maintain consistency. We leave this for future work.

References

[1] Y. Afek, S. Kutten, and M. Yung. The local detection paradigm and its applications to self stabilization. *TCS*, 186:199–230, 1997.

[2] N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. Alg.*, 7:567–583, 1986.

[3] A. Amit, N. Linial, J. Matousek, and E. Rozenman. Random lifts of graphs. In *Proc. 12th SODA*, 883–894, 2001.

[4] B. Awerbuch, B. Patt-Shamir, and G. Varghese. Self-Stabilization By Local Checking and Correction. *Proc. FOCS*, 1991, 268-277.

[5] L. Barenboim and M. Elkin. Distributed $(\Delta + 1)$-coloring in linear (in delta) time. *Proc. 41st STOC*, 111–120, 2009.

[6] A. Das Sarma, S. Holzer, L. Kor, A. Korman, D. Nanongkai, G. Pandurangan, D. Peleg and R. Wattenhofer. Distributed Verification and Hardness of Distributed Approximation. *Proc. 43rd STOC*, 2011.

[7] D Dereniowski and A. Pelc. Drawing maps with advice. *JPDC*,72:132-143, 2012.

[8] E.W. Dijkstra. Self-stabilization in spite of distributed control. *Comm. ACM*, 17(11), 643–644, 1974.

[9] S. Dolev, M. Gouda, and M. Schneider. Requirements for silent stabilization. *Acta Informatica*, 36(6), 447-462, 1999.

[10] P. Fraigniaud, C. Gavoille, D. Ilcinkas and A. Pelc. Distributed Computing with Advice: Information Sensitivity of Graph Coloring. *Proc. 34th ICALP*, 231-242, 2007.

[11] P. Fraigniaud, D Ilcinkas, and A. Pelc. Communication algorithms with advice. *JCSS*, 76:222–232, 2008.

[12] P. Fraigniaud, A Korman, and E. Lebhar. Local MST computation with short advice. *Proc. 19th SPAA*, 154–160, 2007.

[13] P. Fraigniaud, A. Korman, and D. Peleg. Local Distributed Decision. *Proc. 52nd FOCS*, 708-717, 2011.

[14] P. Fraigniaud and A. Pelc. Decidability Classes for Mobile Agents Computing. Proc. 10th LATIN, 2012.

[15] P. Fraigniaud, S. Rajsbaum, and C. Travers. Locality and Checkability in Wait-free Computing. *Proc. 25th DISC*, 2011.

[16] P. Fraigniaud, S. Rajsbaum, and C. Travers. Universal Distributed Checkers and Orientation-Detection Tasks. Submitted, 2012.

[17] M. Göös and J. Suomela. Locally checkable proofs. *Proc. 30th PODC*, 2011.

[18] L. Kor, A. Korman and D. Peleg. Tight Bounds For Distributed MST Verification. *Proc. 28th STACS*, 2011.

[19] A. Korman and S. Kutten. Distributed verification of minimum spanning trees. *Distributed Computing*, 20:253–266, 2007.

[20] A. Korman, S. Kutten, and T. Masuzawa. Fast and Compact Self-Stabilizing Verification, Computation, and Fault Detection of an MST. *Proc. 30th PODC*, 2011.

[21] A. Korman, S. Kutten, and D Peleg. Proof labeling schemes. *Distributed Computing*, 22:215–233, 2010.

[22] A. Korman, J.S. Sereni, and L. Viennot. Toward More Localized Local Algorithms: Removing Assumptions Concerning Global Knowledge. *Proc. 30th PODC*, 49-58, 2011.

[23] F. Kuhn. Weak graph colorings: distributed algorithms and applications. *Proc. 21st SPAA*, 138–144, 2009.

[24] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15:1036–1053, 1986.

[25] M. Naor. A Lower Bound on Probabilistic Algorithms for Distributive Ring Coloring. *SIAM J. Discrete Math.*, 4(3): 409-412 (1991).

[26] M. Naor and L. Stockmeyer. What can be computed locally? *SIAM J. Comput.* 24(6): 1259-1277 (1995).

[27] A. Panconesi and A. Srinivasan. On the Complexity of Distributed Network Decomposition. *J. Alg.* 20: 356-374, (1996).

[28] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach.* SIAM, 2000.

[29] J. Schneider and R. Wattenhofer. A new technique for distributed symmetry breaking. In *Proc. 29th PODC*, 257-266, 2010.