Randomized Local Network Computing

Laurent Feuilloley*
Dep. of Computer Science
Aalto University
Finland

Pierre Fraigniaud†
Dep. of Computer Science
CNRS and University Paris Diderot
France

ABSTRACT

In this paper, we carry on investigating the line of research questioning the power of randomization for the design of distributed algorithms. In their seminal paper, Naor and Stockmeyer [STOC 1993] established that, in the context of network computing, in which all nodes execute the same algorithm in parallel, any *construction* task that can be solved locally by a randomized Monte-Carlo algorithm can also be solved locally by a deterministic algorithm. This result however holds in a specific context. In particular, it holds only for distributed tasks whose solutions can be locally *checked* by a deterministic algorithm. In this paper, we extend the result of Naor and Stockmeyer to a wider class of tasks. Specifically, we prove that the same derandomization result holds for every task whose solutions can be locally checked using a 2-sided error randomized Monte-Carlo algorithm. This extension finds applications to, e.g., the design of lower bounds for construction tasks which tolerate that some nodes compute incorrect values. In a nutshell, we show that randomization does not help for solving such *resilient* tasks.

1. INTRODUCTION

1.1 Context and objective

In the framework of network computing, in which all nodes execute the same algorithm in parallel, few problems (apart from notable exceptions such as *weak coloring* [28] or *fractional coloring* [18]) can be solved in constant time. That is, very few problems can be solved by having every node inspecting solely its neighborhood at distance $O(1)$. This holds even if one allows randomization. Indeed, in their seminal paper, Naor and Stockmeyer [28] proved that, under specific assumptions, any randomized Monte-Carlo algorithm solving a problem in a constant number of rounds of communication can be derandomized, i.e., turned into a deterministic algorithm also performing in a constant number of rounds.

In this paper, we question the existence of randomized algorithms solving relaxed versions of problems, in which part of the nodes are allowed to output incorrect values, i.e., values not respecting the specification of the problem. For instance, in the case of the relaxed $(\Delta+1)$-coloring problem [4], some nodes are allowed to output the same color as one or some of their neighbors. Similarly, for the relaxed constructive version of the Lovász local lemma (LLL) [6], some nodes are allowed to output assignments for which the corresponding "bad" event holds.

One type of relaxation has been extensively considered in the literature, namely *ϵ-slackness* [5, 8, 19]. Roughly, for any fixed $\epsilon \in [0,1]$, the ϵ-slack relaxation of a problem tolerates that an ϵ-fraction of the nodes outputs values that violate the specifications of the problem. In fact, randomization is a very powerful tool for solving such relaxed problems. For instance, it is known [25] that no deterministic algorithms can achieve 3-coloring of the n-node ring in less than $\Omega(\log^* n)$ rounds (the same holds for LLL [6]). Nevertheless, the trivial randomized algorithm in which every node picks independently uniformly at random a color 1, 2, or 3, enables to guarantee that, with constant probability, a fraction $1-\epsilon$ of the nodes are properly colored, i.e., do not conflict with the colors of their neighbors. That is, the ϵ-slack relaxation of a problem may be solved by a local randomized Monte-Carlo algorithm, while it cannot be solved locally by any deterministic algorithm. In other words, randomization helps for solving ϵ-slack relaxations (at least for certain tasks).

Therefore, let us consider a weaker relaxation, called *f-resilient*. Roughly, for any $f \geq 0$, the f-resilient relaxation of a problem tolerates that up to at most f nodes are "faulty", in the sense that they output values that violate the specifications of the problem. As opposed to the ϵ-slack relaxation, it seems unlikely that randomization helps for solving f-resilient relaxations. Nevertheless, establishing lower bounds for the f-resilient relaxation of a problem requires to address one major issue. *Checking* whether or not a given candidate solution to the f-resilient relaxation of a problem is a valid solution may *not* be achievable locally in a decentralized manner. In other words, f-resilient variants of locally checkable problems are not necessarily locally check-

*Part of this work was done while the first author was visiting LIAFA at university Paris Diderot, supported by ENS Cachan.
†Additional support from the ANR project DISPLEXITY, and from the INRIA project GANG.

SPAA'15, June 13–15, 2015, Portland, OR, USA.

DOI: http://dx.doi.org/10.1145/2755573.2755596.

able. For instance, checking whether a given graph coloring is proper can be done in just one round by having each node comparing its color with the colors of its neighbors. However, checking whether all but at most f nodes are properly colored is not checkable locally. This is simply because checking whether at least $n - f$ nodes have a correct output is a global property that can hardly be checked by the nodes by inspecting their local neighborhood only. This fact has strong negative consequences. In particular, it prevents us from using the results in the seminal paper [28]. In this paper, Naor and Stockmeyer established a collection of major results related to local network computing, including the following two crucial results:

1. The study of deterministic constant-time algorithms can be reduced to the study of *order-invariant* algorithms, i.e., algorithms which do not use the actual values of the node identities, but only their relative order.

2. The study of constant-time algorithms can be reduced to the study of *deterministic* algorithms, since, as we already mentioned, any randomized constant-time Monte-Carlo algorithm can be transformed into a constant-time deterministic algorithm.

These two results however hold in a specific context. In particular, they hold only for solving problems whose solutions can be *checked locally* by a deterministic algorithm. Therefore, the fact that the solutions to f-resilient relaxations of problems may not be locally checkable prevents us from using both of these results, which are corner stones for the analysis of deterministic and randomized distributed local algorithms in networks.

As a consequence of the above, our interest in the design and analysis of f-resilient algorithms led us to tackle a wider and intriguing question: to which extend the assumption on local checkability can be relaxed while preserving the integrity of both the order-invariant reduction, and the derandomization result in [28].

In fact, as far as the order-invariant reduction is concerned, the local checkability assumption is actually not much of an issue, as shown in [3], as long as the node identities are not restricted in size. Indeed, if the only requirement is that the identities given to the nodes are pairwise distinct integers, then [3] proved that any constant-time (deterministic) algorithm can be turned to an order-invariant algorithm performing in the same amount of time.

Getting rid of the local checkability assumption however appears to be much more of an issue for derandomization. Roughly, this is because, for tasks whose solutions are locally checkable, we can define the notion of legal and illegal balls: the ball $B_G(u,t)$ of radius t around a node u in a network G, including the data at the nodes, as well as their identities, is *legal* if and only if the partial solution in this ball satisfies the specification of the task. For instance, for the coloring task, a ball $B_G(u,t)$ is legal if an only if all nodes are properly colored within this ball. The crucial point is that if a ball $B = B_G(u,t)$ is legal (resp., illegal) in one network G, the same ball $B = B_H(u,t)$ remains legal (resp., illegal) in any other network H where this ball may appear.

Instead, if a task is not locally checkable, then, depending on the specification of the task, it may be the case that a ball B is legal as a part of one network G, but becomes illegal as a part of another network H. As a consequence, the classical proof technique based on glueing different networks for boosting the probability of failure of randomized algorithms becomes quite delicate in absence of the local checkability assumption. Nevertheless, despite this obstacle, we shall show that the local checkability assumption can be relaxed significantly, while still preserving the ability to derandomize constant-time Monte-Carlo algorithms in the framework of network computing.

1.2 Our results

We extend the result of Naor and Stockmeyer to a wider class of tasks. Specifically, we prove that the same derandomization result as the one in [28] holds for every distributed problem whose solutions can be checked in constant time using a *2-sided error randomized Monte-Carlo* algorithm.

More precisely, recall that BPLD, which stands for *bounded-probability local decision* (see [13]), is the class of *distributed languages* that can be probabilistically decided in constant time with constant error probability. That is, a distributed language $\mathcal{L}$ is in BPLD if and only if there exists $p > \frac{1}{2}$, and an algorithm $\mathbf{A}$, satisfying the following. After having inspected their neighborhood at constant distance t, every node outputs $true$ or $false$ such that: if the instance is in the language $\mathcal{L}$, then, with probability at least p, all nodes output $true$, and, if the instance is not in $\mathcal{L}$, then, with probability at least p, at least one node outputs $false$. We prove that, in the LOCAL model [29], for every $\mathcal{L} \in$ BPLD, if there exists a randomized Monte-Carlo construction algorithm for $\mathcal{L}$ running in $O(1)$ rounds, then there exists a deterministic construction algorithm for $\mathcal{L}$ running in $O(1)$ rounds. This generalizes the result by Naor and Stockmeyer from the class LD (which stands for *local decision*), i.e., the class of distributed languages that can be deterministically decided in constant time, to the class BPLD, i.e., the class of distributed languages for which the randomized decision algorithm may err with some probability.

This extension finds applications to the design of lower bounds for construction tasks which tolerate that some part of the nodes may compute incorrect values. That is, in particular, our result finds applications to f-resilient relaxations of classical problems such as coloring, minimal dominating set, maximal matching, etc. We prove that, while randomization helps for ϵ-slack relaxations, it does not help for f-resilient relaxations.

1.3 Related work

In the context of network computing, the issue of locality has been the source of intensive research. We refer to, e.g., the textbook [29] for an introduction to the design and analysis of local algorithms, and to [30] for a recent survey of local algorithms. In particular, the graph coloring task has been investigated in depth (see [4]), for various reasons, including its applications to, e.g., the management of radio networks [26]. It is known that the n-node cycle cannot be 3-colored in less than $\Omega(\log^* n)$ rounds, and this holds even if the nodes are aware of n, and share a common sense of direction. The $\Omega(\log^* n)$ lower bound also holds for randomized Monte-Carlo algorithms that can err with probability at most $1/2$ [27].

Several efforts have been made for understanding the different reasons why a problem can or cannot be solved locally.

Several aspects of network computing play a role, including the presence or absence of identities [12, 17], the ability to output values of unbounded size [18], and the presence or absence of a priori knowledge about the network [21]. A crucial step was made in [28] which essentially shows that randomization does not help for local computing as long as one aims at solving problems whose solutions can be checked locally. Recently, a similar result has been proved in [9] for anonymous networks provided with a special kind of coloring. Other lines of research investigate the ability to solve approximated solutions of problems that cannot be solved locally [10, 23, 24, 22], or the ability to solve locally such problems using quantum resources [2, 16].

Many of these aforementioned citations underlined strong connections between the ability to construct a solution and the ability to check whether or not a solution is correct. Local decision then became an autonomous line of research [11, 13]. Interestingly, the connection between decision and verification tasks finds applications to other aspects of network computing (see, e.g., [7, 20]), and even outside the framework of network computing (see, e.g., [14, 15]).

2. MODEL AND NOTATIONS

2.1 Computing model

2.1.1 Deterministic algorithms

We consider the usual framework for the analysis of locality in network computing, namely the LOCAL model [29]. In this model, a network is modeled as a *connected* and *simple* graph (i.e., no loops, and no multiple edges). Each node v of a network is given an *identity*, denoted by $\mathsf{id}(v)$. This identity is a positive integer, and the identities of the nodes in the same network are pairwise distinct. An algorithm $\mathbf{A}$ in the LOCAL model starts at the same time at all nodes. Then all nodes perform the same instructions, in a sequence of *synchronous rounds*. At each round, every node

1. sends messages to its neighbors,
2. receives the messages of its neighbors, and
3. performs some individual computation.

The algorithm $\mathbf{A}$ performs in *time* t if, for every instance, every node outputs after having performed at most t rounds. Note that there are no limits on the size of the messages exchanged during one round, nor on the amount of computation individually performed by every node at each round. As a consequence, lower bounds for the LOCAL model are very robust, for the algorithms in this model are only subject to one unique constraint: the maximum distance at which every node communicates in the network.

Indeed, an algorithm $\mathbf{A}$ performing in t rounds can be simulated by an algorithm $\mathbf{B}$ executing two phases: First, in a network G, every node v collects all data from nodes at distance at most t from v (i.e., their inputs and identities, as well as the structure of the connections between these nodes); Second, every node simulates the execution of $\mathbf{A}$ in $B_G(v,t)$, where $B_G(v,t)$ is the *ball* of radius t around node v in graph G, that is, $B_G(v,t)$ is the subgraph of G induced by all nodes at distance at most t from v, excluding the edges between the nodes at distance exactly t from v. (More specifically, every node simulates the r first rounds of nodes at distance $t-r$). In other words, an algorithm performing in $t = O(1)$ rounds in the LOCAL model can simply be viewed as an algorithm $\mathbf{B}$ in which every node outputs after having inspected their t-neighborhood in the network.

Note that the input $x(w)$ given to every node w in $B_G(v,t)$ may obviously impact the output $y(v)$ of a t-round algorithm at node v of network G, but so does as well the identity $\mathsf{id}(w)$ given to each of these nodes in $B_G(v,t)$.

Order-invariant algorithms. Recall that an *order-invariant* distributed algorithm is a distributed algorithm for which the output at any given node does not depend on the actual *values* of the identities of the nodes in its vicinity, but only on the *relative order* of these identities. More precisely, an algorithm $\mathbf{A}$ is order-invariant if the following holds: for any graph G, for any inputs given to the nodes, and for any two identity assignments id and id' of the nodes in G, if the ordering of the nodes in G induced by id, and the one induced by id' are identical, then the output of $\mathbf{A}$ at every node v is the same in both instances, the one with identities from id and the one with identities from id'.

2.1.2 Randomized algorithms

A randomized Monte-Carlo algorithm $\mathbf{A}$ in the LOCAL model performs the same as a deterministic algorithm, apart from the fact that every node has also access to a private source of independent random bits. These random bits may well be exchanged between nodes during the execution of the algorithm. The completion time t of $\mathbf{A}$ is deterministic, but the output $y(v)$ at every node v of a network G is random, depending on the random bits at v, as well as, potentially, the random bits of the nodes in $B_G(v,t)$.

A randomized Monte-Carlo algorithm $\mathbf{A}$ has success probability $r \in [0,1]$ if, for every instance (i.e., every connected simple graph G, every input x and every identity assignment id to the nodes), the global output y produced by the nodes satisfies the specification of the problem to be solved with probability at least r.

2.2 Decision and construction tasks

Stating our main result requires to define properly the notions of *decision* and *construction* tasks.

2.2.1 Distributed languages and tasks

Given a connected graph $G = (V, E)$, and two functions

$$x, y : V \to \{0,1\}^*,$$

the pair $(G,(x,y))$ is called an *input-output configuration*. In the configuration $(G,(x,y))$, each node $v \in V$ has input string $x(v)$, and output string $y(v)$. The pair (G,x) is called the input configuration of $(G,(x,y))$, and the pair (G,y) is called its output configuration.

A family $\mathcal{L}$ of input-output configurations such that, for every input configuration (G,x) there exists an output configuration (G,y) satisfying $(G,(x,y)) \in \mathcal{L}$, is called a *distributed language*, or simply *language* for short.

Any language defines two different kinds of tasks:

- The *construction task* for the language $\mathcal{L}$ consists in, given any input configuration (G,x), computing y such that $(G,(x,y)) \in \mathcal{L}$. That is, every node v starts with

its individual input $x(v)$, and, after having communicated long enough with its neighbors, must eventually produce an individual output $y(v)$.

Note that y does not need to be unique as there could be different y's such that $(G,(x,y)) \in \mathcal{L}$ for the same x. Which y is returned by the nodes may in particular depend on the node identities. Given an input-configuration (G,x) on a network G with identity assignment id, the triple (G,x,id) is called an *instance* of the task.

- The *decision task* for the language $\mathcal{L}$ consists in, given any input-output configuration $(G,(x,y))$, computing a boolean at each node such that: $(G,(x,y)) \in \mathcal{L}$ if and only if every node outputs *true*. If all nodes output *true*, we say that the configuration is *accepted*, otherwise it is *rejected*.

  Note that, in case $(G,(x,y)) \notin \mathcal{L}$, the node which outputs $false$ may not be the same for every instance $(G,(x,y),\mathsf{id})$ of the decision task, as, once more, the behavior of the nodes may differ as a function of their identities.

2.2.2 Local decision class

According to the terminology of [13], for any $t \geq 0$, we denote by $\mathrm{LD}(t)$ the class of languages $\mathcal{L}$ locally decidable in t rounds. That is, $\mathrm{LD}(t)$ is the class of languages $\mathcal{L}$ for which there exists a distributed algorithm performing in at most t rounds in the LOCAL model, and such that: for any $(G,(x,y)) \in \mathcal{L}$, all nodes output *true*, and, for any $(G,(x,y)) \notin \mathcal{L}$, at least one node outputs $false$. Finally, we define LD as the class of languages $\mathcal{L}$ decidable in constant number of rounds, i.e.,

$$\mathrm{LD} = \cup_{t \geq 0} \mathrm{LD}(t).$$

Note that there are languages decidable in a constant number of rounds (i.e., in LD) that are not constructible in a constant number of rounds (a typical example is `coloring` [25]). The reverse is also true, that is, there are languages constructible in a constant number of rounds, but that are not decidable in a constant number of rounds (i.e., not in LD). A typical example is `majority`, requiring that a majority of nodes output $\star$. Finally, there are languages that are both decidable and constructible in constant time (a typical example is `weak coloring` [28]), or both non decidable and non constructible in constant time (a typical example is `MST` [29]).

2.2.3 A derandomization result

In the following, we shall focus on the class of languages whose input-output configurations are defined on graphs of bounded degree, with inputs and outputs of bounded size. More specifically, let us fix a non-negative integer k. We denote by $\mathcal{F}_k$ the set of input-output configurations $(G,(x,y))$ where G has degree at most k, and, for every node v, the lengths of the strings $x(v)$ and $y(v)$ are both at most k. That is,

$$\mathcal{F}_k = \{(G,(x,y)) : \forall v \in V,\ \max\{\deg(v), |x(v)|, |y(v)|\} \leq k\}.$$

The "derandomization" theorem in [28] is seminal in the context of local computing in networks. It deals with languages in a class called LCL (for *locally checkable labelling*), which is essentially the class LD restricted to languages in $\mathcal{F}_k$ for some k. We prefer to view $\mathcal{F}_k$ as a promise, which has the advantage of clearly separating what is related to the type of algorithms deciding the languages from what is related to the type of instances the construction and decision algorithms are dealing with.

THEOREM (Naor and Stockmeyer [28]) *Let $\mathcal{L} \in$ LD, and k be a non negative integers. If there exists a randomized Monte-Carlo construction algorithm for $\mathcal{L}$ with promise $\mathcal{F}_k$, $k > 2$, running in $O(1)$ rounds, then there exists a deterministic construction algorithm for $\mathcal{L}$ with promise $\mathcal{F}_k$ running in $O(1)$ rounds.*

We will show how to extend this theorem to a class of languages wider than LD. For this purpose, we need first to define *randomized* distributed decision.

2.3 Randomized distributed decision

2.3.1 Definition

In randomized distributed decision, the decision regarding whether an input-output configuration $(G,(x,y))$ belongs to $\mathcal{L}$ for a given distributed language $\mathcal{L}$ is taken collectively as in the deterministic setting, but according to relaxed rules. More specifically, as in the deterministic setting, each node must either accept (i.e., output *true*) or reject (i.e., output $false$), but nodes are now allowed to err, up to some limited extend.

More precisely, a randomized algorithm decides $\mathcal{L}$ with guarantee $p \in (\frac{1}{2}, 1]$ if the following holds for every input-output configuration $(G,(x,y))$, and every identity assignment id to the nodes:

$$\begin{array}{l}(G,(x,y)) \in \mathcal{L} \Rightarrow \Pr[\text{all nodes accept}] \geq p \\ (G,(x,y)) \notin \mathcal{L} \Rightarrow \Pr[\text{at least one node rejects}] \geq p.\end{array} \quad (1)$$

The above definition is also equivalent to the notion of (p,q)-decider in [13] by taking $p = q > \frac{1}{2}$.

Example. The following language, also defined in [13], plays an important role in the theory of randomized decision (see also [11]).

$$\texttt{amos} = \{(G,(x,y))) :\ |\{v \in V(G), y(v) = \star\}| \leq 1\}$$

where `amos` stands for "at most one selected", and a selected node is a node marked by $\star$. On the one hand, `amos` cannot be deterministically decided in $D/2-1$ rounds in graphs of diameter D (because no nodes can decide whether or not two nodes at distance D are selected). On the other hand, `amos` can be *randomly* decided in zero rounds in all graphs, with guarantee $p = \frac{\sqrt{5}-1}{2} \simeq 0.618$ as follows. Every non selected node v accepts, and every selected node v accepts with probability p, and rejects with probability $1-p$. This algorithm can err only if there are one or more selected nodes. In case one node is selected, the algorithm accepts with probability p, as desired. In case two or more nodes are selected, the algorithm rejects with probability at least $1-p^2$, i.e., with probability at least p, as desired.

2.3.2 Bounded probability local decision class

According to the terminology of [13], for a fixed integer $t \geq 0$, we denote by $\mathrm{BPLD}(t)$ the class of languages decidable in t rounds by a randomized Monte-Carlo algorithm with guarantee p, for some constant $p \in (\frac{1}{2}, 1]$. That is, $\mathrm{BPLD}(t)$ is the class of languages for which there exists a randomized

distributed algorithm performing in at most t rounds in the LOCAL model, and such that: for any $(G,(x,y)) \in \mathcal{L}$, the probability that all nodes accept is at least p, and, for any $(G,(x,y)) \notin \mathcal{L}$, the probability that at least one node rejects is at least p.

Finally, we define BPLD as the class of languages $\mathcal{L}$ randomly decidable in constant number of rounds, i.e.,

$$\text{BPLD} = \cup_{t \geq 0} \text{BPLD}(t).$$

By definition, we have LD $\subseteq$ BPLD, and languages such as `amos` enable to show that the inclusion is strict.

Note that we do not insist on p being large: $p > \frac{1}{2}$ is sufficient. As a consequence, we get that BPLD is a fairly large class of languages, which is desirable in the perspective of generalizing Naor and Stockmeyer derandomization theorem to a large class of construction tasks.

3. MAIN RESULT

In this section, we establish our main result. That is, we prove the following extension of Naor and Stockmeyer derandomization theorem where the class LD is replaced by the larger class BPLD of languages randomly decidable in $O(1)$ rounds. Recall that $\mathcal{F}_k$ denotes the promise that the input-output configuration $(G,(x,y))$ satisfies G has degree at most k, and the input and output strings are of length at most k.

THEOREM 1. *Let $\mathcal{L}$ be a distributed language in BPLD, and let k be a non negative integer. If there exists a randomized Monte-Carlo construction algorithm for $\mathcal{L}$ with promise $\mathcal{F}_k$, $k > 2$, running in $O(1)$ rounds, then there exists a deterministic construction algorithm for $\mathcal{L}$ with promise $\mathcal{F}_k$ running in $O(1)$ rounds.*

PROOF. First, we state an intermediate result that enables to reduce the investigation of *deterministic* algorithms to the one of order-invariant algorithms. Observe that, since the language $\mathcal{L}$ in the statement of the theorem may not be in LD, this first stage of the proof requires a different approach as the one in [28], because the reduction to order-invariant algorithms in [28] requires the languages to be in LD. Nevertheless, it was recently shown (see Theorem 1 in [3]) that the LD assumption is not necessary. The result hereafter does not even need $\mathcal{L} \in$ BPLD.

CLAIM 1. [3] *Let $\mathcal{L}$ be a distributed language, and let k, t be two non negative integers. If there exists a (deterministic) construction algorithm for $\mathcal{L}$ with promise $\mathcal{F}_k$ running in t rounds, then there exists a (deterministic) order-invariant construction algorithm for $\mathcal{L}$ with promise $\mathcal{F}_k$ running in t rounds.*

For the sake of completeness, for further references, and for emphasizing the different roles of the bounded input hypothesis and of the bounded output hypothesis, a proof of Claim 1 is given in Appendix A.

Let $\mathcal{L}$ and k be as in the statement of the theorem. We always assume input-output configurations in $\mathcal{F}_k$.

Assume that there exists a randomized Monte-Carlo construction algorithm $\mathbf{C}$ for $\mathcal{L}$ running in t rounds, for some $t \geq 0$. Observe that if $\mathbf{C}$ does not err on graphs with diameter larger than D, then $\mathbf{C}$ can be transformed into a deterministic algorithm running in $\max\{D, t\}$ rounds, and the theorem holds. Therefore, from this point on, we assume that $\mathbf{C}$ errs on networks with arbitrarily large diameter.

For sake of contradiction, we make the following assumption:

($\star$) There are no t-round deterministic construction algorithms for $\mathcal{L}$.

Let us denote by r the success probability of $\mathbf{C}$. That is, for every (G,x), and every identity assignment id, the output y constructed by $\mathbf{C}$ satisfies

$$\Pr[(G,(x,y)) \in \mathcal{L}] \geq r. \tag{2}$$

An input-output configuration $(G,(x,y))$ where y is constructed by $\mathbf{C}$ on instance (G,x,id) is denoted by $\mathbf{C}(G,x,\mathsf{id})$.

The assumption ($\star$) enables to get the following result partially established in [28] — the same claim in [28] does not place any requirement on the diameter and on the identities. This claim does not use the fact that $\mathcal{L} \in$ BPLD (neither it uses the assumption $\mathcal{L} \in$ LD in [28]).

CLAIM 2. *There exists $\beta > 0$ such that, for every non-negative integers D_{min} and I_{min}, there exists a graph H with diameter $D \geq D_{min}$, an input x, and an identity-assignment id with $\mathsf{id}(v) \geq I_{min}$ for every node v of H, for which*

$$\Pr[\mathbf{C}(H,x,\mathsf{id}) \notin \mathcal{L}] \geq \beta.$$

That is, $\mathbf{C}$ fails with probability at least β on instance (H,x,id).

The proof of the claim follows the same guidelines as a proof of a similar result in [28], based on an original idea from [1], so we only sketch that part, just emphasizing the role of order-invariance, and for taking care of the additional requirements related to the diameter and the identity assignment. Since we assume input-output configurations in $\mathcal{F}_k$, we get that there is a finite number N of (deterministic) order-invariant algorithms running in t rounds. This is because there is a finite number of balls of radius t in a graph of maximum degree k, a finite number of k-bit input-output configuration for each of these balls, and a finite number of orderings for the node identities in these balls. Note that N depends only on t, k, and $\mathcal{L}$. We set

$$\beta = 1/N$$

as in [28]. Let $I_{min} \geq 0$ and $D_{min} \geq 0$ be two integers. Since, by assumption ($\star$), there are no deterministic algorithms for $\mathcal{L}$ running in T rounds, we get that each of the N order-invariant algorithms fails for an infinite family of instances. Thus, for each order-invariant algorithm $\mathbf{A}$, let us pick an instance (H,x,id) such that $\mathbf{A}$ fails on (H,x,id). We can pick H with an arbitrarily large diameter $D \geq D_{min}$ since otherwise $\mathbf{A}$ would fail for a finite family of instances only. Also, we can pick an identity assignment id with identities at least I_{min}, by the order invariant property. Let $\mathcal{H}$ be the set of such "bad" instances, one for each of the N order-invariant algorithms. By the same arguments as in [1, 28], we get that either there exists an instance $(H,x,\mathsf{id}) \in \mathcal{H}$ such that $\Pr[\mathbf{C}(H,x,\mathsf{id}) \notin \mathcal{L}] \geq \beta$, or there exists a random bits sequence σ such that, for every instance $(H,x,\mathsf{id}) \in \mathcal{H}$, $\mathbf{C}(H,x,\mathsf{id})$ is correct with the sequence σ. The latter case is impossible since, by Claim 1, this would mean that there exists an order-invariant algorithm that is correct for all instances in $\mathcal{H}$, a contradiction to the construction of $\mathcal{H}$. Therefore, there exists $(H,x,\mathsf{id}) \in \mathcal{H}$

such that $\Pr[\mathbf{C}(H,x,\mathsf{id}) \notin \mathcal{L}] \geq \beta$. This completes the proof of Claim 2. $\diamond$

From now on, the assumption $\mathcal{L} \in$ BPLD becomes crucial in the rest of the proof, which now diverges from the proof in [28].

Since $\mathcal{L} \in$ BPLD, there exists a randomized Monte-Carlo algorithm $\mathbf{D}$ deciding $\mathcal{L}$ in constant time. Let us denote by $p > \frac{1}{2}$ the success guarantee of $\mathbf{D}$. Algorithm $\mathbf{D}$ satisfies the probabilistic properties of Eq. (1). We denote by t' the number of rounds of Algorithm $\mathbf{D}$.

To provide an intuition of how the assumption $\mathcal{L} \in$ BPLD is used, let us first provide an intuition of the rest of the proof by relaxing the constraints that input-output configurations deal with connected graphs. For this purpose, let us define BPLD* as the class of language defined as BPLD but on configuration $(G,(x,y))$ where G needs not to be connected. The following result is a relaxed variant of Theorem 1 in which distributed languages are allowed to be defined on non-connected graphs (i.e., hence we are now considering input-output configurations from $\mathcal{F}_k^*$, which is defined as $\mathcal{F}_k$, but without the graph connectivity assumption).

Claim 3. *Let $\mathcal{L}^*$ be a distributed language in* BPLD**, and let k be a non negative integers. If there exists a randomized Monte-Carlo construction algorithm for $\mathcal{L}$ with promise $\mathcal{F}_k^*$, $k > 2$, running in $O(1)$ rounds, then there exists a deterministic construction algorithm for $\mathcal{L}$ with promise $\mathcal{F}_k^*$ running in $O(1)$ rounds.*

The assumption that graphs do not need to be connected has no impact on Claims 1 and 2, which remain both valid in this context. Let β as in Claim 2, and let

$$\nu = 1 + \left\lceil \frac{\ln(rp)}{\ln(1-\beta p)} \right\rceil \tag{3}$$

where we recall that r is the success probability of the construction algorithm $\mathbf{C}$, and p is the success guarantee of the decision algorithm $\mathbf{D}$. Let $I_1 = 0$, and let us fix an arbitrary diameter bound $D_{min} = 1$. Let $(H_1, x_1, \mathsf{id}_1)$ be an instance whose existence is guaranteed by Claim 2 for $I_{min} = I_1$ and D_{min}. Now let $I_2 = 1 + \max_{v \in V(H_1)} \mathsf{id}(v)$, and, by Claim 2, let $(H_2, x_2, \mathsf{id}_2)$ be an instance whose existence is guaranteed by Claim 2 for $I_{min} = I_2$ and D_{min}. We can carry on that process in the same way for constructing a sequence of instances $(H_i, x_i, \mathsf{id}_i)$, for $i = 1, 2, \ldots, \nu$.

Let (G, x, id) be the union of the instances $(H_i, x_i, \mathsf{id}_i)$, where id is the identity-assignment in G corresponding to the concatenation of the identity-assignments id_i, $i = 1, 2, \ldots, \nu$. Note that id is well defined since two different identity assignments id_i and id_j do not overlap, for any $1 \leq i < j \leq \nu$.

For a fixed i, the probability that $\mathbf{D}$ rejects $\mathbf{C}(H_i, x_i, \mathsf{id}_i)$ is at least βp since

$$\Pr[\mathbf{C}(H_i, x_i, \mathsf{id}_i) \notin \mathcal{L}] \geq \beta$$

and

$$\Pr[\mathbf{D} \text{ rejects } \mathbf{C}(H_i, x_i, \mathsf{id}_i) | \mathbf{C}(H_i, x_i, \mathsf{id}_i) \notin \mathcal{L}] \geq p.$$

Therefore,

$$\Pr[\mathbf{D} \text{ accepts } \mathbf{C}(H_i, x_i, \mathsf{id}_i)] \leq 1 - \beta p$$

and thus

$$\Pr[\mathbf{D} \text{ accepts } \mathbf{C}(G, x, \mathsf{id})] \leq (1 - \beta p)^{\nu} \tag{4}$$

since the decoder runs independently in each H_i, in which, with probability at least βp, at least one node rejects. On the other hand,

$$\Pr[\mathbf{D} \text{ accepts } \mathbf{C}(G, x, \mathsf{id})] \geq p \Pr[\mathbf{C}(G, x, \mathsf{id}) \in \mathcal{L}]. \tag{5}$$

Combining Eq. (4) with Eq. (5), we get

$$\Pr[\mathbf{C}(G, x, \mathsf{id}) \in \mathcal{L}] \leq \frac{1}{p} (1 - \beta p)^{\nu}.$$

If follows that

$$\Pr[\mathbf{C}(G, x, \mathsf{id}) \in \mathcal{L}] < r,$$

by definition of ν in Eq. (3). This is a contradiction with Eq. (2), i.e., the fact that $\mathbf{C}$ has success probability at least r. This contradiction indicates that hypothesis ($\star$) cannot be true for $\mathcal{L}^*$, that is, it cannot be true that there are no t-round deterministic algorithms for $\mathcal{L}^*$. This concludes the proof of Claim 3. $\diamond$

Implementing the main idea in the proof of Claim 3 while preserving network connectivity requires much more work. This is because, as mentioned in the introduction, boosting the error probability of the construction algorithm by running it on several hard instances in parallel requires to "glue" these instances, which is tricky for BPLD languages. Indeed, if one wants to connect the graphs H_i of the latter proof together, the following problem appears. When a graph H_i is modified somewhere, the "views" of some vertices close to the modified part of H_i change, and this might modify the outputs returned by the algorithm $\mathbf{C}$ at these nodes, which might in turn reduce the probability of failure of the construction algorithm. This issue is no big deal for languages in LD because if $\mathbf{C}$ constructs an illegal ball somewhere in H_i around some node v_i, then, as long as H_i is connected to the other graphs H_j's without changing the $(t+t')$-neighborhood of v_i, then, in the connected graphs, $\mathbf{C}$ still constructs an illegal ball around v_i, and $\mathbf{D}$ notices it. However, as we mentioned before, the notion of legal and illegal ball is irrelevant for BPLD, which deals with predicates that may be non local (e.g., at most f edges have extremities with conflicting colors). Nevertheless, we now prove that the BPLD assumption is still sufficient for allowing us to glue the graphs H_i appropriately.

Recall that $p > \frac{1}{2}$. We define

$$\mu = \left\lceil \frac{1}{2p-1} \right\rceil$$

and set

$$D = 2\mu(t + t')$$

where t and t' are the running times of $\mathbf{C}$ and $\mathbf{D}$, respectively. Using Claim 2, we construct a sequence of instances $(H_i, x_i, \mathsf{id}_i)$, $i = 1, \ldots, \nu'$, the same way we did in the proof of Claim 3, but with a diameter bound $D_{min} = D$ instead of $D_{min} = 1$, for an integer ν' to be specified later.

In order to glue the graphs H_i, $i = 1, \ldots, \nu'$, together, we need to identify in each graph H_i a node u_i around which edges will be added in order to connect H_i with other graphs H_j, $j \neq i$. Let $i \in \{1, \ldots, \nu'\}$. Consider a set S of μ vertices in H_i at distance at least $2(t+t')$ from each other. Such a set exists because the diameter of H_i is at least $D = 2\mu(t+t')$. We show how to choose the desired node u_i as one of the nodes in S. For this purpose, we do two things. One is

analyzing the execution of the construction algorithm $\mathbf{C}$ for each of the possible choices of its random coins, and the other is to restrict our attention, for each $u \in S$, to the behavior of the nodes far away from u when running the decider $\mathbf{D}$.

More specifically, any Monte-Carlo algorithm such as $\mathbf{C}$ or $\mathbf{D}$ is using a finite (potentially unbounded) random bit-string at each node. The collection of random bit-strings generated by all the nodes during one execution of the algorithm forms a multi-set of random strings indexed by node identities. This set can be viewed itself as a (large) random bit-string composed of as many sub-strings as the number of nodes, ordered by the node identities. Let $Rand(\mathbf{C})$ and $Rand(\mathbf{D})$ be the set of random bit-strings used by $\mathbf{C}$ and $\mathbf{D}$, respectively.

Recall that the event "$\mathbf{D}$ rejects $(G,(x,y))$" is a short cut for the event "$\mathbf{D}$ outputs $false$ in at least one node when executed on $(G,(x,y))$". We say that $\mathbf{D}$ rejects $(G,(x,y))$ *far from* $v \in V(G)$ if $\mathbf{D}$ outputs $false$ in at least one node at distance greater than $t+t'$ from v, when executed on $(G,(x,y))$. Similarly, we say that $\mathbf{D}$ accepts $(G,(x,y))$ far from $v \in V(G)$ if $\mathbf{D}$ outputs $true$ at all nodes at distance greater than $t+t'$ from v, when executed on $(G,(x,y))$.

Let us fix a string

$$\sigma \in Rand(\mathbf{C})$$

that makes $\mathbf{C}$ fails on instance $(H_i, x_i, \mathsf{id}_i)$. We denote by $\mathbf{C}_\sigma$ the algorithm $\mathbf{C}$ with the fixed random string σ. Notice that $\mathbf{C}_\sigma$ is deterministic.

Claim 4. *There exists $u \in S$ such that*

$$\Pr[\mathbf{D} \textit{ accepts } \mathbf{C}_\sigma(H_i, x_i, \mathsf{id}_i) \textit{ far from } u] < p.$$

To establish the claim, notice that, since $\mathbf{C}_\sigma(H_i, x_i, \mathsf{id}_i) \notin \mathcal{L}$, it follows that

$$\Pr[\mathbf{D} \text{ rejects } \mathbf{C}_\sigma(H_i, x_i, \mathsf{id}_i)] \geq p.$$

Suppose, for the purpose of contradiction, that the claim does not hold. It means that, for every node $u \in S$,

$$\Pr[\mathbf{D} \text{ accepts } \mathbf{C}_\sigma(H_i, x_i, \mathsf{id}_i) \text{ far from } u] \geq p.$$

Let $\mathcal{E}_u$ be the event $\mathbf{D}$ rejects $\mathbf{C}_\sigma(H_i, x_i, \mathsf{id}_i)$, and accepts $\mathbf{C}_\sigma(H_i, x_i, \mathsf{id}_i)$ far from u. By union bound, if follows that, for every node $u \in S$,

$$\Pr[\mathcal{E}_u] \geq 2p - 1.$$

A string in $Rand(\mathbf{D})$ for which $\mathcal{E}_u$ holds is called *critical* for node u. We show that the set of critical strings are disjoint, that is, if a string is critical for $u \in S$ then it is not critical for a different node $u' \in S$. Let

$$\sigma' \in Rand(\mathbf{D})$$

be a critical string for $u \in S$, and let us consider the set $Reject(u, \sigma')$ of vertices of H_i at which $\mathbf{D}_{\sigma'}$ rejects $\mathbf{C}_\sigma(H_i, x_i, \mathsf{id}_i)$, where $\mathbf{D}_{\sigma'}$ is the (deterministic) algorithm resulting from applying $\mathbf{D}$ with string σ'. Since $\mathbf{D}_{\sigma'}$ rejects $\mathbf{C}_\sigma(H_i, x_i, \mathsf{id}_i)$, but accepts $\mathbf{C}_\sigma(H_i, x_i, \mathsf{id}_i)$ far from u, we have:

$$Reject(u, \sigma') \subseteq B_{H_i}(u, t+t')$$

As a consequence, for any two nodes $u \neq u'$ of S, we have

$$Reject(u, \sigma') \cap Reject(u', \sigma') = \emptyset$$

because

$$B_{H_i}(u, t+t') \cap B_{H_i}(u', t+t') = \emptyset.$$

It follows that the set of critical strings are disjoint. In other words, the events $\mathcal{E}_u$, $u \in S$, are disjoint. As a consequence, if $\mathcal{E}$ denotes the event that $\mathcal{E}_u$ holds for some $u \in S$, we have

$$\Pr[\mathcal{E}] = \sum_{u \in S} \Pr[\mathcal{E}_u] \geq \mu(2p-1).$$

This is impossible since, by definition of μ, we have

$$\mu(2p-1) > 1.$$

This completes the proof of Claim 4. ◇

We are now ready to establish the existence of a convenient node u_i in H_i enabling to connect H_i to the other H_j's.

Claim 5. *There exists a node u of H_i such that*

$$\Pr[\mathbf{D} \textit{ accepts } \mathbf{C}(H_i, x_i, \mathsf{id}_i) \textit{ far from } u] < 1 - \frac{\beta(1-p)}{\mu}.$$

To establish the claim, let

$$X = \sum_{u \in S} \Pr[\mathbf{D} \text{ rejects } \mathbf{C}(H_i, x_i, \mathsf{id}_i) \text{ far from } u].$$

Note that the probabilities are taken on both the random choices of $\mathbf{C}$, and the ones of $\mathbf{D}$. Let us separate the two by defining

$$\Sigma = \{\sigma \in Rand(\mathbf{C}) : \mathbf{C}_\sigma(H_i, x_i, \mathsf{id}_i) \notin \mathcal{L}\}.$$

We have

$$X \geq \sum_{u \in S} \sum_{\sigma \in \Sigma} \Pr[\mathbf{D} \text{ rejects } \mathbf{C}_\sigma(H_i, x_i, \mathsf{id}_i) \text{ far from } u] \cdot \Pr[\sigma].$$

Now, by Claim 4 we know that, for any $\sigma \in \Sigma$, there exists $u \in S$ such that

$$\Pr[\mathbf{D} \text{ rejects } \mathbf{C}_\sigma(H_i, x_i, \mathsf{id}_i) \text{ far from } u] > 1 - p.$$

On the other hand, we also know from Claim 2 that

$$\sum_{\sigma \in \Sigma} \Pr[\sigma] \geq \beta.$$

Therefore, $X \geq \beta(1-p)$, that is,

$$\sum_{u \in S} \Pr[\mathbf{D} \text{ rejects } \mathbf{C}(H_i, x_i, \mathsf{id}_i) \text{ far from } u] \geq \beta(1-p).$$

Therefore, there exists $u \in S$ such that

$$\Pr[\mathbf{D} \text{ rejects } \mathbf{C}(H_i, x_i, \mathsf{id}_i) \text{ far from } u] \geq \beta(1-p)/\mu.$$

This concludes the proof of the Claim 5. ◇

To complete the proof of the theorem, we now glue all graphs H_i together, as follows. For every $i = 1, \ldots, \nu'$, let u_i be a node satisfying Claim 5 for H_i. Let e_i be an edge incident to u_i in H_i. We subdivide each edge e_i twice, by inserting two nodes v_i and w_i. Then we add an edge between v_i and w_{i+1}, for $i = 1, \ldots, \nu' - 1$, and an edge between $v_{\nu'}$ and w_1. In this way, we form a connected graph G with degree at most k (recall that $k > 2$). The inputs and identities given to the nodes of G not in some H_i are set arbitrarily (with the only constraint that no identities present in one H_i should be given to any node of G). This construction results in an instance (G, x, id).

Let us now compute

$$q = \Pr[\mathbf{D} \text{ accepts } \mathbf{C}(G, x, \mathsf{id})].$$

On the one hand, we have

$$q \leq \prod_{i=1}^{\nu'} \Pr[\mathbf{D} \text{ accepts } \mathbf{C}(H_i, x_i, \mathsf{id}_i) \text{ far from } u_i]$$

where "far from u_i" must be understood as "far from u_i in H_i", that is when considering only nodes in H_i at distance greater than $t+t'$ from u_i. This inequality holds because, to accept, every node must output *true*, and so, in particular, those in H_i far from u_i, for all i. These sets of nodes are at distance more than $2(t+t')'$ from each other, and each of the nodes in these sets cannot distinguish an instance on H_i from an instance on G. This implies that the decision made by $\mathbf{D}$ in each set is independent from the one taken in another set.

Now, by applying Claim 5, we get that

$$\Pr[\mathbf{D} \text{ accepts } \mathbf{C}(G, x, \mathsf{id})] \leq \left(1 - \frac{\beta(1-p)}{\mu}\right)^{\nu'}.$$

On the other hand, as for Claim 3, we have

$$\Pr[\mathbf{D} \text{ accepts } \mathbf{C}(H, x, \mathsf{id})] \geq p \Pr[\mathbf{C}(H, x, \mathsf{id}) \in \mathcal{L}].$$

Combining the two latter equations, we get

$$\Pr[\mathbf{C}(H, x, \mathsf{id}) \in \mathcal{L}] \leq \frac{1}{p}\left(1 - \frac{\beta(1-p)}{\mu}\right)^{\nu'}.$$

If follows that, by choosing

$$\nu' = 1 + \left\lceil \frac{\ln(rp)}{\ln\left(\frac{1}{p}\left(1 - \frac{\beta(1-p)}{\mu}\right)\right)} \right\rceil$$

we obtain $\Pr[\mathbf{C}(H, x, \mathsf{id}) \in \mathcal{L}] < r$, which is a contradiction with Eq. (2), i.e., the fact that $\mathbf{C}$ has success probability at least r. This contradiction yields that hypothesis ($\star$) does not hold, which implies that there must exist a t-round deterministic algorithms for $\mathcal{L}$. This concludes the proof of the theorem. □

4. RESILIENT RELAXATIONS

In this section, we derive lower bounds for relaxed construction tasks. Let $\mathcal{L}$ be a distributed language defined by the exclusion of a collection $Bad(\mathcal{L})$ of balls $B(v,t)$ for some $t = O(1)$. For instance, `coloring` is the language which exclude all balls of radius 1 such that the center of the ball has same color as one of its neighbors. According to [28], we denote by LCL the class of such languages.

DEFINITION 1. *The f-resilient relaxation of $\mathcal{L} \in$ LCL, denoted by $\mathcal{L}_f$, is the language consisting of all input-output configurations $(G,(x,y))$ containing at most f balls in $Bad(\mathcal{L})$.*

We prove that randomization does not help for the design of construction algorithm for $\mathcal{L}_f$. Note that this result does not follow from the derandomization result in [28] since $\mathcal{L}_f$ is not necessarily locally checkable (i.e., not necessarily in LD). The following is however a corollary of Theorem 1.

COROLLARY 1. *Let $\mathcal{L} \in$ LCL, $k > 2$, and $f > 0$. If there exists a randomized Monte-Carlo construction algorithm for $\mathcal{L}_f$ with promise $\mathcal{F}_k$, running in $O(1)$ rounds, then there exists a deterministic construction algorithm for $\mathcal{L}_f$ with promise $\mathcal{F}_k$ running in $O(1)$ rounds.*

PROOF. It is sufficient to show that $\mathcal{L}_f \in$ BPLD, since the result will then follows from Theorem 1. Let

$$p \in (e^{-\frac{\ln 2}{f}}, e^{-\frac{\ln 2}{f+1}}).$$

The randomized decision algorithm performs in t rounds, where t is the maximum radius of the balls excluded from $\mathcal{L}$. In instance $(G,(x,y),\mathsf{id})$, every node v collects the ball $B_G(v,t)$ of radius t in G. If $B_G(v,t) \notin Bad(\mathcal{L})$ then v accepts. Instead, if $B_G(v,t) \in Bad(\mathcal{L})$ then v accepts with probability p, and rejects with probability $1-p$. Note that this is a well defined algorithm since the set of bad balls is finite in $\mathcal{F}_k$, and therefore $B_G(v,t) \in Bad(\mathcal{L})$ is decidable (in the usual sense of sequential computing). Given $(G,(x,y),\mathsf{id})$, we define

$$F(G) = \{v \in V(G) : B_G(v,t) \in Bad(\mathcal{L})\}.$$

- Assume $(G,(x,y)) \in \mathcal{L}$. The probability that all nodes accept is $p^{|F(G)|}$. Therefore, since $|F(G)| \leq f$, we get
$$\Pr[\text{all nodes accept}] \geq p^f > \frac{1}{2}.$$
- Assume $(G,(x,y)) \notin \mathcal{L}$. The probability that at least one node rejects is $1-p^{|F(G)|}$. Therefore, since $|F(G)| \geq f+1$, we get
$$\Pr[\text{at least one node rejects}] \geq 1 - p^{f+1} > \frac{1}{2}.$$

Therefore, $\mathcal{L} \in$ BPLD, and the results follows by Theorem 1. □

As a simple illustration of Corollary 1, we get that the f-relaxations of $(\Delta+1)$-coloring and LLL cannot be solved in constant time, even if using a randomized Monte-Carlo algorithm. Indeed, both languages belong to LCL, and therefore, by Corollary 1, it is sufficient to show that there is no constant-time deterministic algorithms solving the f-relaxation of $(\Delta+1)$-coloring or the f-relaxation of LLL. By Theorem 1 in [3] (cf. Claim 1 in the proof of Theorem 1), it is actually sufficient to show that no order-invariant algorithms can solve these relaxed tasks. In the cycle C_n where adjacent nodes are given consecutive identities from 1 to n (excepted for nodes with IDs 1 and n), any order-invariant 3-coloring algorithm performing in $t = O(1)$ rounds acts identically in at least $n-(2t-1)$ nodes since all the balls centered at nodes with IDs in $[t, n-t]$ are identical as far as the ordering of the identities are concerned. Therefore, $n-(2t-1)$ nodes output the same color, and thus the algorithm cannot be f-resilient. The same result follows for LLL as well, by the reduction of LLL to coloring in [6].

Note that the example of application to $(\Delta+1)$-coloring is for the ease of presentation. Other than that, it may look like using a sledgehammer to crack a nut. Indeed, for that task, one could derive the impossibility result directly by noticing that fixing the colors of a finite number f on nodes when all the other nodes are properly colored can be done in constant time. However, this "local fixing" property does not necessarily holds for all languages in BPLD. Even for languages in LD, local fixing might be difficult. For instance, *frugal coloring* is the task requiring to proper color the nodes, with the additional constraint that no color appears more than c times in the neighborhood of any node (for

some given c). Locally fixing frugal coloring is not as easy as locally fixing unconstrained proper coloring. More generally, locally fixing a language whose specification are not fully local (as it is the case of the languages in BPLD $\setminus$ LD) is not necessarily straightforward.

5. OPEN PROBLEMS

A natural question risen by this paper is whether BPLD is the ultimate class of tasks for which a derandomization theorem à la Naor and Stockmeyer holds. There are several classes that might be good candidates for further extensions. Examples of such classes are NLD and BPNLD, that is, the classes of languages that can be locally *verified*, deterministically or randomly, respectively (see [13]). These are the classes of languages for which one can certify the membership to the class thanks to *local certificates*. They are to LD and BPLD, respectively, what NP is to P. Another candidate for further extensions is the class LD^{O}, that is, LD enhanced with an oracle O providing some information about the environment (see also [13]). Indeed, it is frequent that distributed algorithms assume that the nodes are a priori aware of global information about the network, like, e.g., its size, or an upper bound on its size.

From a technical point of view, generalizing Theorem 1 to tasks in one of the classes LD^{O}, NLD, or BPNLD, requires to overcome a serious obstacle. When constructing larger instances by glueing smaller instances, the certificates and/or the information provided to the nodes by the oracle, may change radically, and therefore it is quite hard to relate the outputs of the construction and the decision algorithms when running in smaller instances, to the outputs of these algorithms in larger instances.

Note that Theorem 1 does not extend to $\mathrm{BPLD}^{\#\mathsf{node}}$, the class of languages that can be decided by a randomized algorithm aware of the number of nodes. Indeed, the ϵ-slack relaxation of $(\Delta+1)$-coloring is in $\mathrm{BPLD}^{\#\mathsf{node}}$ (using the same algorithm as in the proof of Corollary 1 with $f = \epsilon n$). Yet, there is a constant-time (zero-round) randomized Monte-Carlo algorithm for the ϵ-slack relaxation of $(\Delta+1)$-coloring (every node picks a color in $\{1, \ldots, \Delta+1\}$ uniformly at random), but there are no constant-time deterministic algorithms for the ϵ-slack relaxation of $(\Delta+1)$-coloring.

Another issue is the extension of Theorem 1 to tasks in restricted classes of networks. Theorem 1 extends to tasks with promises such as planar graphs, or 2-connected graphs. Indeed, the construction in the proof the theorem preserve planarity and 2-connectivity. We do not know whether Theorem 1 can be generalized to every language with promise.

Finally, we have seen that randomization helps for the ϵ-slack relaxation, but does not help for the f-resilient relaxation. One intriguing question is whether randomization helps for intermediate relaxations, like allowing $O(n^c)$ nodes to output incorrect values, for $c < 1$. Note that the corresponding languages are not necessarily in BPLD (hence Theorem 1 cannot be used). They are in $\mathrm{BPLD}^{\#\mathsf{node}}$, but we have seen that this latter class is too wide for a derandomization result such as the one in [28] to hold. Nevertheless, there might be a class C, with

$$\mathrm{BPLD} \subset \mathrm{C} \subset \mathrm{BPLD}^{\#\mathsf{node}}$$

to which Theorem 1 could be extended.

6. REFERENCES

[1] L. M. Adleman: Two Theorems on Random Polynomial Time. In proc. 19th IEEE Symp. on Foundations of Computer Science, pp75-83, 1978.

[2] H. Arfaoui, P. Fraigniaud: What can be computed without communications? SIGACT News 45(3): 82-104 (2014)

[3] H. Arfaoui, P. Fraigniaud, D. Ilcinkas, F. Mathieu: Distributedly Testing Cycle-Freeness. In proc. 40th Int. Work. on Graph-Theoretic Concepts in Computer Science (WG), Springer LNCS, pp15-28, 2014.

[4] L. Barenboim, M. Elkin: Distributed Graph Coloring: Fundamentals and Recent Developments. Synthesis Lectures on Distributed Computing Theory, Morgan & Claypool Publishers, 2013.

[5] H. T.-H. Chan, M. Dinitz, A. Gupta: Spanners with Slack. In proc. 14th Annual European Symposium on Algorithms (ESA), pp196-207, 2006.

[6] K.-M. Chung, S. Pettie, H.-H. Su: Distributed algorithms for the Lovász local lemma and graph coloring. In proc. 33rd ACM Symp. on Principles of Distributed Computing (PODC), pp134-143, 2014.

[7] A. Das Sarma, S. Holzer, L. Kor, A. Korman, D. Nanongkai, G. Pandurangan, D. Peleg, R. Wattenhofer: Distributed Verification and Hardness of Distributed Approximation. SIAM J. Comput. 41(5): 1235-1265 (2012)

[8] M. Dinitz: Compact routing with slack. In proc. 26th ACM Symp. on Principles of Distributed Computing (PODC), pp81-88, 2007.

[9] Y. Emek, C. Pfister, J. Seidel, R. Wattenhofer: Anonymous networks: randomization = 2-hop coloring. In proc. 33rd ACM Symp. on Principles of Distributed Computing (PODC), pp96-105, 2014.

[10] P. Floréen, J. Kaasinen, P. Kaski, J. Suomela: An optimal local approximation algorithm for max-min linear programs. In proc. 21st ACM Symp. on Parallelism in Algorithms and Architectures (SPAA), pp260-269, 2009.

[11] P. Fraigniaud, M. Göös, A. Korman, M. Parter, D. Peleg: Randomized distributed decision. Distributed Computing 27(6): 419-434 (2014)

[12] P. Fraigniaud, M. Göös, A. Korman, J. Suomela: What can be decided locally without identifiers? In proc. 32nd ACM Symp. on Principles of Distributed Computing (PODC) 2013: 157-165

[13] P. Fraigniaud, A. Korman, D. Peleg: Towards a complexity theory for local distributed computing. J. ACM 60(5): 35 (2013)

[14] P. Fraigniaud, S. Rajsbaum, C. Travers: Locality and checkability in wait-free computing. Distributed Computing 26(4): 223-242 (2013)

[15] P. Fraigniaud, S. Rajsbaum, C. Travers: On the Number of Opinions Needed for Fault-Tolerant Run-Time Monitoring in Distributed Systems. In proc. 5th International Conference on Runtime Verification (RV), pp92-107, 2014.

[16] C. Gavoille, A. Kosowski, M. Markiewicz: What Can Be Observed Locally? In proc. 23rd Int. Symp. on Distributed Computing (DISC), Springer LNCS, pp243-257, 2009.

[17] M. Göös, J. Hirvonen, J. Suomela: Lower bounds for local approximation. J. ACM 60(5): 39 (2013)
[18] H. Hasemann, J. Hirvonen, J. Rybicki, J. Suomela: Deterministic Local Algorithms, Unique Identifiers, and Fractional Graph Colouring. In proc. 19th Colloq. on Struct. Information and Comm. Complexity (SIROCCO), Springer LNCS, pp48-60, 2012.
[19] G. Konjevod, A. W. Richa, D. Xia, H. Yu: Compact routing with slack in low doubling dimension. In proc. 26th ACM Symp. on Principles of Distributed Computing (PODC), pp71-80, 2007.
[20] A. Korman, Shay Kutten, D. Peleg: Proof labeling schemes. Distributed Computing 22(4): 215-233 (2010)
[21] A. Korman, J.-S. Sereni, L. Viennot: Toward more localized local algorithms: removing assumptions concerning global knowledge. Distributed Computing 26(5-6): 289-308 (2013)
[22] F. Kuhn, T. Moscibroda, R. Wattenhofer: What cannot be computed locally! In proc. 23rd ACM Symp. on Principles of Distributed Computing (PODC), pp300-309, 2004.
[23] C. Lenzen, Y. Anne Oswald, R. Wattenhofer: What can be approximated locally?: case study: dominating sets in planar graphs. In proc. 20th ACM Symp. on Parallelism in Algo. and Arch. (SPAA), pp46-54, 2008.
[24] C. Lenzen, R. Wattenhofer: Leveraging Linial's Locality Limit. In proc. 22nd Int. Symp. on Distributed Computing (DISC), pp394-407, 2008.
[25] N. Linial: Locality in Distributed Graph Algorithms. SIAM J. Comput. 21(1): 193-201 (1992)
[26] T. Moscibroda, R. Wattenhofer: Coloring unstructured radio networks. In proc. 17th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA), pp39-48, 2005.
[27] M. Naor: A Lower Bound on Probabilistic Algorithms for Distributive Ring Coloring. SIAM J. Discrete Math. 4(3): 409-412 (1991)
[28] M. Naor, L. J. Stockmeyer: What Can be Computed Locally? SIAM J. Comput. 24(6): 1259-1277 (1995)
[29] D. Peleg: Distributed Computing: A Locality-Sensitive Approach. SIAM, Philadelphia, PA, 2000.
[30] J. Suomela: Survey of local algorithms. ACM Comput. Surv. 45(2): 24 (2013)

APPENDIX

A. PROOF OF CLAIM 1

For any set X, and any positive integer r, we denote by $X^{(r)}$ the set of all subsets of X with size exactly r. Let X be a countably infinite set, let r and s be two positive integers, and let $c : X^{(r)} \to [s]$ be a "coloring" of each set in $X^{(r)}$ by an integer in $[s] = \{1, \ldots, s\}$. Recall that (the infinite version of) Ramsey's Theorem states that there exists an infinite set $Y \subseteq X$ such that the image by c of $Y^{(r)}$ is a singleton (that is, all sets in $Y^{(r)}$ are colored the same by c). We make use of this theorem as follows.

Let $\mathcal{L}$ be a distributed language, and let k, t be two non negative integers. Let us assume that there exists a (deterministic) construction algorithm $\mathbf{A}$ for $\mathcal{L}$ on $\mathcal{F}_k$ running in t rounds. We consider the collection $\mathcal{C}$ of all labeled balls isomorphic to any labeled ball $B(v,t)$ of radius t, centered at some node v in some graph G with maximum degree k, with inputs on at most k bits. The isomorphism should preserve not only the structure of the balls, but also the labeling of the nodes in the balls. There is a finite number ν of pairwise non-isomorphic labeled balls in $\mathcal{C}$ because the graph has bounded degree, and the inputs are of bounded size. We enumerate these labeled balls, and let n_i be the number of vertices in the ith ball, for $i = 1, \ldots, \nu$. For every i, the vertices of the ith ball can be ordered in $n_i!$ different manners, corresponding to the $n_i!$ permutations in Σ_{n_i}. We consider the $N = \sum_{i=1}^{\nu} n_i!$ ordered balls $B_{i,\sigma}$, for $i = 1, \ldots, \nu$, and $\sigma \in \Sigma_{n_i}$, and we enumerate these ordered labeled balls as $\beta_1, \ldots, \beta_N$ in an arbitrary order. Using these labeled balls, we define an infinite set $\mathcal{U}$ of identities as follows.

Let $X_0 = \mathbb{N}$, and assume that we have already secured the existence of a sequence of infinite sets $X_0 \supseteq X_1 \supseteq \cdots \supseteq X_j$, $0 \leq j < N$, such that, for every i, $1 \leq i \leq j$, the output of $\mathbf{A}$ at the center of ball β_i is the same for all possible identity assignments to the nodes in β_i with values in X_i, and respecting the ordering of the nodes in ball β_i. The fact that the outputs are on at most k bits (since the configurations are in $\mathcal{F}_k$) enables to define the coloring $c : X_j^{(r)} \to [2^k]$ where r is the number of nodes in β_{j+1}, as follows: for each r-element set $I \in X_j^{(r)}$, assign r pairwise distinct identities to the nodes of β_{j+1} using the r values in I, and respecting the order of the nodes in β_{j+1}. Then, define $c(I)$ as the output of Algorithm $\mathbf{A}$ at the center of β_{j+1} under this identity assignment to the nodes of β_{j+1}. By Ramsey's Theorem, there exists an infinite set $Y_j \subseteq X_j$ such that all r-element sets $I \in Y_j^{(r)}$ are given the same color. We set $X_{j+1} = Y_j$. We proceed that way until we exhaust all balls β_i, $i = 1, \ldots, N$, and we set $\mathcal{U} = X_N$.

By construction, the set $\mathcal{U}$ satisfies that, for every ball $B_{i,\sigma}$, for $i = 1, \ldots, \nu$, and $\sigma \in \Sigma_{n_i}$, the output of $\mathbf{A}$ at the center of $B_{i,\sigma}$ is the same for all identity assignments to the nodes of $B_{i,\sigma}$ with identities taken from $\mathcal{I}$ and assigned to the nodes in the order σ.

We now define the order-invariant algorithm $\mathbf{A}'$ as follows. Every node v inspects its radius-t ball $B_G(v,t)$ around it in the actual graph G. In particular, it collects the inputs of the nodes, as well as the identities of the nodes in that ball. Let σ be the ordering of the nodes in $B_G(v,t)$ induced by their identities. Node v simulates $\mathbf{A}$ by reassigning identities to the nodes of $B_G(v,t)$ using the $r = |B_G(v,t)|$ smallest values in $\mathcal{U}$, in the order specified by σ, and output what would have outputted $\mathbf{A}$ if nodes were given these identities.

$\mathbf{A}'$ is well defined, as nodes can be provided with the $\sum_{i=0}^{t} d^i$ smallest integers in the set $\mathcal{U}$. (They do not need to know the entire set $\mathcal{U}$, but only a finite number of values in $\mathcal{U}$). Also, by construction, $\mathbf{A}'$ is order-invariant. To establish that $\mathbf{A}'$ is correct, let us consider some input configuration (G, x) with nodes provided with pairwise distinct identities in $\mathcal{U}$, and let $y = \{y(v), v \in V(G)\}$ be the output of $\mathbf{A}$ in this context. The output y is precisely the output of $\mathbf{A}'$ in G. Indeed, every node v relabels its radius-t ball with identities in $\mathcal{U}$, respecting the order induced by the original identities in $\mathcal{U}$, and $\mathcal{U}$ is precisely defined so that the output of v will be the same in both cases. In other words, the output of $\mathbf{A}'$ is precisely the output of $\mathbf{A}$ if nodes were assigned identities restricted to be in $\mathcal{U}$. Hence, since $\mathbf{A}$ is correct, it follows that $(G, (x, y)) \in \mathcal{L}$, and therefore $\mathbf{A}'$ is correct as well. □