Randomized local network computing: Derandomization beyond locally checkable labelings

Laurent Feuilloley, Pierre Fraigniaud

Randomized local network computing*

Derandomization beyond locally checkable labelings

LAURENT FEUILLOLEY, DII, Universidad de Chile, Chile
PIERRE FRAIGNIAUD, IRIF, Université de Paris and CNRS, France

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 only for distributed tasks such that the correctness of their 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 such that the correctness of their solutions can be locally checked using a 2-sided error randomized Monte-Carlo algorithm.

CCS Concepts: • **Theory of computation** → **Distributed computing models**.

Additional Key Words and Phrases: Distributed algorithms, locality, derandomization.

ACM Reference Format:
Laurent Feuilloley and Pierre Fraigniaud. 20XX. Randomized local network computing: Derandomization beyond locally checkable labelings. *ACM Trans. Parallel Comput.* 0, 0, Article 0 (20XX), 25 pages. https://doi.org/XXX

1 INTRODUCTION

1.1 Context and objective

In the framework of network computing, in which all nodes execute the same algorithm in parallel, it is known that randomization helps, in the sense that randomized algorithms can solve tasks faster than deterministic algorithms. However, the help provided by randomized algorithms is somewhat limited. This is typically the case when considering the round complexity of algorithms running in networks, which is essentially defined as the maximum distance between nodes exchanging information during their executions. For instance, it has been shown [8] that the round-complexity of a randomized algorithm solving a task in an n-node network cannot be smaller than the round-complexity of a deterministic algorithm for the same task in networks with $\sqrt{\log n}$ nodes. Also, it has been recently proved [42] that, for n-node networks, the class of tasks with randomized round-complexity $O(\text{polylog}(n))$ is identical to the class of tasks with deterministic round-complexity

*A preliminary version of this paper appeared in the proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures (SPAA), Portland, USA, June 13-15, 2015.

Authors' addresses: Laurent Feuilloley, feuilloley@dii.uchile.cl, DII, Universidad de Chile, Santiago de Chile, Chile; Pierre Fraigniaud, pierre.fraigniaud@irif.fr, IRIF, Université de Paris and CNRS, Paris, 75205, France.

1539-9087/20XX/0-ART0 $15.00
https://doi.org/XXX

$O(\text{polylog}(n))$. In other words, randomization does not help as far as polylogarithmic round-complexity is concerned. All these results can be viewed as extensions of a seminal result by Naor and Stockmeyer [40], who proved that any randomized algorithm solving a taks 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.

The aforementioned derandomization results however apply under some specific assumptions. Specifically, they apply to *locally checkable labelings* (LCL), that is, to tasks such that the correctness of their solutions can be checked locally, i.e., by having every node inspecting solely its neighborhood at distance $O(1)$. Stated differently, these derandomization results apply only to tasks such that the (global) correctness of a solution is determined by the conjunction of *local* correctness criteria. The class of LCL tasks is very large, and includes many classical graph problems such as proper vertex or edge k-coloring, maximal matching, maximal independent set, etc. This class also contains tasks such as odd-degree weak coloring [40] or fractional coloring [28], which can be solved in a constant number of rounds. Nevertheless, other classical graph problems are outside the range of application of currently known derandomization results, as they do not belong to the class of LCL tasks. This includes various spanning tree construction problems (e.g., MST), as well as most of the standard optimization problems (e.g., vertex cover, dominating set, etc.).

This paper questions whether LCL tasks is the ultimate class of tasks for which non-trivial derandomization results such as the aforementioned ones hold. Specifically, we aim at revisiting the power of randomization w.r.t. algorithms performing in a constant number of rounds. As we mentioned before, these algorithms are of no help as far as LCL tasks are concerned [40]. We therefore consider the class of tasks such that the correctness of their solutions can be checked locally (i.e., in a constant number of rounds) by a *randomized* algorithm that is allowed to make errors by rejecting correct solutions, and/or accepting incorrect solutions.

For instance, let us consider a data structure supposed to form a linked list (similar to, e.g., Blockchain), but for which it is acceptable that some branchings occur, as long as they are resulting in up to k leaves, for some fixed parameter $k \geq 1$. If the data structure is distributed, it is not possible to check the correctness of the list locally as a branching at a node does not necessarily make the data structure incorrect. However, distributed checking can be achieved by a randomized algorithm performing in zero rounds. In the algorithm, every leaf is accepted with probability p, and rejected with probability $1-p$, where p is the unique root of $x^{k+1}+x^k-1$ in $(\frac{1}{2},1)$. Every legal instance, i.e., a linked list with $\ell \leq k$ leaves, is accepted with probability p^ℓ, that is, with probability at least p^k. In contrast, every illegal instance, i.e., a linked list with $\ell > k$ leaves, is rejected with probability $1-p^\ell$, that is, with probability at least $1-p^{k+1}=p^k$. The success probability of such a randomized distributed decision algorithm is therefore $p^k > \frac{1}{2}$, where the latter inequality holds as otherwise $p^k(p+1)-1 \leq \frac{p-1}{2} < 0$. More generally, this randomized algorithm can check whether a solution to an LCL task is correct but in at most k nodes. Specifically, every node performs a deterministic local algorithm to determine whether its neighborhood fits with the specification of the LCL task at hand, but it does not systematically reject an illegal neighborhood. Instead, it rejects it with probability $1-p$, and accepts it with probability p. Randomized decision was also used in [18] to decide whether a given graph is a lift of a smaller graph, as a tool for establishing that the class BPNLD contains all decidable classes of labeled graphs, the same way the class LCP [26] contains all decidable classes of labeled graphs — the latter class assumes distributed proofs which may depend on the actual identifiers given to the nodes, while the former class assumes distributed proofs which are oblivious to the IDs.

The question addressed in this paper is to which extent the derandomization result by Naor and Stockmeyer can be extended to the class of tasks for which the correctness of their solutions can

be checked locally, not necessarily by a deterministic algorithm but definitely by a randomized algorithm.

1.2 Our results

We extend the result of Naor and Stockmeyer by proving that the same derandomization result as the one in [40] 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 [18]), 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 a constant $p > \frac{1}{2}$, and an algorithm satisfying the following. After having inspected their neighborhood at constant distance t, every node outputs *accept* or $reject$ such that: if the instance is in the language $\mathcal{L}$, then, with probability at least p, all nodes output *accept*, and, if the instance is not in $\mathcal{L}$, then, with probability at least p, at least one node outputs $reject$. We prove that, in the LOCAL model [41], 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* [18]), 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. As for the results in [40], we consider algorithms performing in the setting of graphs with maximum degree upper bounded by a constant, and input and output labels with size upper bounded by a constant. Specifically, the class LCL considered in [40], as well as in all subsequent papers on derandomization in the LOCAL model, is merely the class LD restricted to instances involving graphs with bounded maximum degrees, and labels of bounded maximum size.

We show that our extension of derandomization to the class of languages that can be decided by randomized algorithms finds applications in the design of lower bounds, e.g., for construction tasks which tolerate that up to f nodes compute incorrect values, for some constant $f \geq 0$.

The techniques. In their paper, Naor and Stockmeyer established a collection of major results related to local network computing, including undecidability results and the following two results:

(1) The study of deterministic constant-time algorithms solving LCL tasks can be reduced to the study of *order-invariant* algorithms, i.e., algorithms which do not use the actual values of the node identifiers, but only their relative order.
(2) The study of constant-time algorithms solving LCL tasks can be reduced to the study of *deterministic* algorithms, as any randomized constant-time Monte-Carlo algorithm can be derandomized into a constant-time deterministic algorithm.

The first of these two results is actually used as a main tool for establishing the second. The question is thus to figure out to which extent the assumption on local checkability (i.e., languages in LCL) can be relaxed to randomized checkability while preserving the integrity of both the order-invariance reduction, and the derandomization reduction.

As far as the order-invariance reduction is concerned, the only techniques we are aware of, i.e., those in [3] and [40], cannot be directly used in our setting. Indeed, the reduction in [40] uses a finite version of Ramsey's theorems coupled with the assumption that the node IDs are taken from a sufficiently large set of IDs. This is sufficient as languages in LCL can be finitely described, e.g., by the list of legal local neighborhoods. It is however not adapted to languages in BPLD as such languages may not have a finite description, even in bounded-degree graphs with finite labels. The languages studied in [3] may not have a finite description. The reduction in [3] uses an infinite

version of Ramsey's theorems. However, we cannot directly use this approach either, because our proof of derandomization later requires counting arguments, which in turn requires to be applied to a finite set of instances. To handle the constraint on the languages (which may not have a finite description), we downgrade the applicability of order-invariance reduction, by merely considering a finite set of graphs, which enables to apply a finite version of Ramsey's theorems. This weak variant of order-invariance reduction is sufficient for the purpose of derandomization in BPLD.

Getting rid of the local checkability assumption for derandomization is another issue. Roughly, this is because, for tasks for which the correctness of the solutions is 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 identifiers, 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 and 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 proof technique in [40] based on gluing different networks, for boosting the probability of failure of randomized algorithms, becomes quite delicate in absence of the local checkability assumption. One reason is that it is not straightforward to connect different graphs together to create a larger connected graphs without modifying the behaviors of rejecting nodes. Nevertheless, by demonstrating the existence of nodes "far from" rejecting nodes, we show that connecting small graphs to form a large graphs can be done without modifying too much the decision of the rejecting nodes.

Overall, despite the obstacles listed above, 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.

1.3 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 [41] for an introduction to the design and analysis of local algorithms, and to [44] for a 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 [38]. It is known that the n-node cycle cannot be 3-colored in $o(\log^* n)$ rounds [37], and this holds even if the nodes are aware of n, and share a common sense of direction. This lower bound also holds for randomized Monte-Carlo algorithms that can err with probability at most $1/2$ [39].

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 identifiers [17, 25], the ability to output values of unbounded size [28], and the presence or absence of a priori knowledge about the network [32]. A crucial step was made in [40] 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 [14] for anonymous networks provided with a special kind of coloring. Other lines of research investigate the ability to compute approximate solutions of problems that cannot be solved locally [15, 34–36], or the ability to solve locally such problems using quantum resources [2, 22].

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 [16, 18]. Interestingly, the connection between decision

and verification tasks finds applications in other aspects of network computing (see, e.g., [31, 43]), and even outside the framework of network computing (see, e.g., [19, 20]).

Lots of efforts (see, e.g., [8, 23, 24]) have also been made for studying the power of randomization in the context of tasks solvable in a polylogarithmic number of rounds — this paper is concerned with tasks solvable in a constant number of rounds. The recent breakthrough [42] in this line of study is that randomization does not help in the polylogarithmic regime either. Specifically, any LCL task solvable in polylog(n) rounds by a randomized algorithm can also be solved in polylog(n) rounds by a deterministic algorithm.

Our result related to the class BPLD finds applications to 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) [11], some nodes are allowed to output assignments for which the corresponding "bad" event holds. This type of relaxation is classic in algorithmics. For example, in (sequential) clustering, one often allows that a small fraction of the points is not properly clustered [9, 10, 12, 21, 33]. This has also been a fruitful approach in the study of networks within the framework of distributed computing. In particular [6] introduces *ϵ-slackness* for distributed spanner construction, based on earlier works on embeddings [1, 7, 29]. 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. The notion of ϵ-slackness has also been proved useful for compact routing [13, 30]. In general, it is a common strategy for distributed algorithms, especially randomized ones, to first solve the problem on a large fraction of the nodes (thus solving a relaxed problem), and then to treat the parts that do not have a proper solution in a different way, or to recurse on these parts (see for example the so-called *shattering technique* introduced in distributed computing by [5]). Randomization is a very powerful tool for solving problems under the ϵ-slack relaxation. For instance, no deterministic algorithms can achieve 3-coloring of the n-node ring in less than $\Omega(\log^* n)$ rounds (the same holds for LLL [11]). Nevertheless, the trivial randomized algorithm in which every node picks independently uniformly at random a color 1, 2, or 3, guarantees that, with constant probability and for a sufficiently large ϵ, a $(1 - \epsilon)$-fraction 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. Note that this observation is not specific to 3-coloring, nor to cycles, as one can use a similar strategy for many problems, including maximal matching or maximal independent set in bounded degree graphs, given a sufficiently large ϵ.

In this paper, we apply our derandomization results to a relaxation that is more constrained than ϵ-slackness, called *f-resilient*. Roughly, for any constant $f \geq 0$, 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 an issue that does not appear for original non-relaxed problems. Namely, *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, checking whether a solution of an LCL task is correct up to f faults may not be doable in a constant number of rounds. 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 [40]. Nevertheless, every f-resilient relaxation of an LCL task belongs to BPLD, and thus our derandomization result applies.

2 MODEL AND NOTATIONS

2.1 Computing model

Deterministic algorithms. We consider the usual framework for the analysis of locality in network computing, namely the LOCAL model [41]. 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 *identifier*, denoted by $\mathsf{id}(v)$. This identifier is a positive integer, and the identifiers of the nodes in the same network are pairwise distinct.

An algorithm 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 messages from its neighbors, and
(3) performs some individual computation.

The algorithm performs in *time t* if, for every instance, every node compute an output and terminates 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, namely the maximum distance at which every node communicates in the network.

Indeed, a deterministic or randomized algorithm performing in t rounds can be simulated by an algorithm 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, identifiers and, in the case of randomized algorithms, their random bits, as well as the structure of the connections between these nodes); second, every node simulates the execution of the original algorithm 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. In other words, an algorithm performing in $t = O(1)$ rounds in the LOCAL model can simply be viewed as an algorithm 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 impact the output $y(v)$ of a t-round algorithm at node v, and so does the identifier $\mathsf{id}(w)$ given to each of these nodes in $B_G(v,t)$, as well as their random bits, if any.

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 identifiers of the nodes in its vicinity, but only on the *relative order* of these identifiers. More precisely, an algorithm is order-invariant if the following holds: for any graph G, for any inputs given to the nodes, and for any two identifier 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 the algorithm at every node v is the same in both instances, the one with identifiers from id and the one with identifiers from id'.

Randomized algorithms. A randomized Monte-Carlo algorithm 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 the algorithm 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 has success probability $r \in (0,1]$ if, for every instance (i.e., every connected simple graph G, every input x and every identifier 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.

Distributed languages and tasks. Given a connected graph $G = (V, E)$, and two functions

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

assigning labels to the nodes of G, 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 $y(v)$. Given $(G, (x, y))$, the pair (G, x) is called the input configuration, and the pair (G, y) is called the output configuration. A *distributed language*, or simply *language* for short, is a family of input-output configurations.

Any language defines two different kinds of tasks:

- The *construction task* for the language $\mathcal{L}$ consists in, given any input configuration (G, x) for which there exists $y : V(G) \longrightarrow \{0,1\}^*$ such that $(G, (x, y)) \in \mathcal{L}$, computing such a label assignment y. That is, every node v starts with its individual input $x(v)$, and, after having communicated long enough with its neighbors, it must eventually produce an individual output $y(v)$, such that the resulting $y : V \longrightarrow \{0,1\}^*$ satisfies $(G, (x, y)) \in \mathcal{L}$.
  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 identifiers. Given an input-configuration (G, x) on a network G with identifier 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 identifiers.

Local decision class. According to the terminology of [18], for any $t \geq 0$, we denote by $\mathrm{LD}(t)$ the class of languages 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 decidable in a 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 the language `coloring` [37]). 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 on graphs of bounded-degree,

where every node has odd degree [40]), or both non decidable and non constructible in constant time (a typical example is MST [41]).

A derandomization result. In the following, we shall focus on the class of languages whose input-output configurations are defined on bounded degree graphs, with inputs and outputs of bounded size. More specifically, let us fix two nonnegative integers Δ and k. We denote by $\mathcal{F}_{\Delta,k}$ the set of input-output configurations $(G,(x,y))$ where $G=(V,E)$ has maximum degree at most Δ, and, for every node v, the lengths of the strings $x(v)$ and $y(v)$ are both at most k. That is,

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

In the following, all our algorithms are executed under the *promise* that they are performed on configurations in $\mathcal{F}_{\Delta,k}$. Note that we separate the different assumptions that we use: on the one hand the language is locally checkable (the LD condition), and on the other hand the graphs considered are of bounded degrees, and of bounded inputs and outputs size (the promise that the configurations are in $\mathcal{F}_{\Delta,k}$).

The following derandomization result is seminal in the context of local computing in networks.

Theorem 2.1 (Naor and Stockmeyer (Theorem 5.2 in [40])). *Let $\mathcal{L} \in$ LD, and let $\Delta > 2$ and $k \geq 0$ be integers. If there exists a randomized Monte-Carlo construction algorithm with success probability $r \in (0,1]$ for $\mathcal{L}$ with configurations taken from $\mathcal{F}_{\Delta,k}$, running in $O(1)$ rounds, then there exists a deterministic construction algorithm for $\mathcal{L}$ with configurations taken from $\mathcal{F}_{\Delta,k}$ running in $O(1)$ rounds.*

Note also that the original formulation of Theorem 2.1 actually deals with languages in a class called LCL (for *locally checkable labeling*), which is essentially the class LD restricted to configurations in $\mathcal{F}_{\Delta,k}$ for some Δ and k. Again, we prefer to separate the issue related to the type of algorithms deciding the languages (i.e., constant-time decision, a.k.a. LD), from the issue related to the type of configurations the construction and decision algorithms are dealing with (i.e., those in $\mathcal{F}_{\Delta,k}$). 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

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 extent. More precisely, a randomized algorithm decides $\mathcal{L}$ with guarantee $p \in (0,1]$ if the following holds for every input-output configuration $(G,(x,y))$, and every identifier assignment id to the nodes:

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

The above definition is also equivalent to the notion of (p,q)-decider in [18] by taking $p=q$.

Example. The following language, also defined in [18], plays an important role in the theory of randomized decision (see also [16]).

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

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 node 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 with probability 1, 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.

Bounded probability local decision class. According to the terminology of [18], for every $p \in (0,1]$, and every integer $t \geq 0$, we denote by $\mathrm{BPLD}_p(t)$ the class of languages decidable in t rounds by a randomized Monte-Carlo algorithm with guarantee p. That is, $\mathrm{BPLD}_p(t)$ 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_p as the class of languages randomly decidable in a constant number of rounds, i.e.,

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

By definition, we have $\mathrm{LD} \subseteq \mathrm{BPLD}_p$ for very $p \in (0,1]$, and languages such as amos enable us to show that the inclusion is strict, at least for $p = \frac{\sqrt{5}-1}{2}$. For more details about the classes BPLD_p, see [16, 18].

3 MAIN THEOREM AND PROOF OUTLINE

In this section, we state our main result, and we present an outline of the proof for providing intuition. The complete proof is in the next section.

3.1 Theorem statements

We prove an extension of the derandomization theorem by Naor and Stockmeyer, where the class LD in Theorem 2.1 is replaced by the larger class BPLD_p with $p > \frac{1}{2}$. The latter is the class of languages that are *probabilistically* decidable in $O(1)$ rounds, with success probability p. More precisely, we prove two theorems. The first one establishes a derandomization result preserving the same round-complexity, but it requires to lower bound the size of the graphs. The second one does not require conditions on the size, but the derandomization result does not preserve the round-complexity. Recall that $\mathcal{F}_{\Delta,k}$ denotes the set of input-output configurations $(G,(x,y))$ satisfying that G has degree at most Δ, and the input and output strings are of length at most k.

Theorem 3.1. *Let $\mathcal{L}$ be a distributed language in BPLD_p, $p > 1/2$, and let $\Delta > 2$, $k \geq 0$, and $t \geq 0$ be integers, and let $r \in (0,1]$. There exists n_0 such that if there exists a randomized Monte-Carlo construction algorithm with success probability at least r for $\mathcal{L}$ with configurations taken from $\mathcal{F}_{\Delta,k}$, running in t rounds in graphs with at least n_0 nodes, then there exists a deterministic construction algorithm for $\mathcal{L}$ with configurations taken from $\mathcal{F}_{\Delta,k}$, also running in t rounds in graphs with at least n_0 nodes.*

The lower bound n_0 on the size of the graphs for which Theorem 3.1 applies depends on all the parameters p, Δ, k, t, and r, as well as on the smallest $t' \geq 0$ such that $\mathcal{L} \in \mathrm{BPLD}_p(t')$. This lower bound somewhat limits the applicability of the theorem. Nevertheless, it has at least two important features. First, the round-complexity is preserved, as the deterministic algorithm runs in the same number of rounds as the randomized algorithm. Second, no constraints are imposed on the set of identifiers. In particular, for n-node graphs, these identifiers may be chosen in a set $\{1,\ldots,n^{O(1)}\}$,

or even in the set $\{1, \ldots, n\}$. If one is not attached to the round-complexity of the derandomized algorithm, as long as it remains constant, then Theorem 3.1 has a simple variant.

Theorem 3.2. *Let $\mathcal{L}$ be a distributed language in* BPLD$_p$*, $p > 1/2$, and let $\Delta > 2$, and $k \geq 0$ be integers. If there exists a randomized Monte-Carlo construction algorithm with success probability $r \in (0, 1]$ for $\mathcal{L}$ with configurations taken from $\mathcal{F}_{\Delta,k}$, running in $O(1)$ rounds, then there exists a deterministic construction algorithm for $\mathcal{L}$ with configurations taken from $\mathcal{F}_{\Delta,k}$ running in $O(1)$ rounds.*

Note that, as for Theorem 2.1, Theorem 3.2 enforces no constraints on the success probability $r > 0$ of the Monte-Carlo construction algorithm, as long as r is a constant. As for Theorem 3.1, Theorem 3.2 applies even if the identifiers are supposed to be taken in $\{1, \ldots, n\}$. However, for small n, the constant hidden in the big-O notation corresponding to the round-complexity of the deterministic construction algorithm may actually exceed n. This is because all the aforementioned theorems requires identifiers taken from a sufficiently large set of integers, where "sufficiently large" refers to Ramsey's numbers.

In the next two subsections, we give a sketch of the proofs for Theorems 3.1 and 3.2, based on the guideline used for establishing Theorem 2.1. Actually Theorem 3.2 can be considered as a corollary of Theorem 3.1, as the proof of the former is very short given the latter. We cast both statements as theorems because we view them as equally useful.

3.2 Proof outline of Theorem 3.1

We first provide an outline of the proof of Theorem 3.1, and highlight the similarities and differences with the proof by Naor and Stockmeyer for Theorem 2.1, i.e., in the case of languages in LD.

The proof goes by contradiction. Let us assume that there exists a language $\mathcal{L}$ that matches the hypothesis of Theorem 3.1, for which there exists a randomized construction algorithm $\mathbf{C}$ running in t rounds, but no deterministic algorithms running in t rounds exist. We then prove that there must exist a configuration (G, x) on which the randomized algorithm $\mathbf{C}$ fails with arbitrarily large probability, thus $\mathbf{C}$ is not a correct randomized algorithm. As in [40], the configuration (G, x) is built in two steps: first we prove that the algorithm fails with positive small probability on some specific small configurations, and then we combine these small configurations into a large configuration, (G, x), for amplifying the failure probability. These two steps are detailed below.

First step: identifying configurations with probability of failure bounded away from 0. The first ingredient of the proof basically consists in identifying a family of small instances on which the algorithm $\mathbf{C}$ fails with probability bounded from below by some $\beta > 0$. This alone would follow directly from the assumption that there is no correct deterministic algorithms for $\mathcal{L}$. However, for being useful in the second step of the proof, we need such small instances to include nodes with arbitrarily large identifiers. Subsection 4.3 describes how such instances can be identified. The general idea boils down to a counting argument. However, for this approach to work, we need to argue that we can restrict attention to a *finite* number of algorithms. We do so in Subsection 4.1 using arguments from Ramsey theory. Specifically, we show that, under some conditions, we can turn any algorithm into an order-invariant algorithm (Lemma 4.1). This is sufficient for our purpose, as there is a finite number of such algorithms, as long as we are considering constant-time algorithms, for configurations taken from $\mathcal{F}_{\Delta,k}$.

The idea of focusing on order-invariant algorithms is borrowed from [40]. However, the technical details of the proof had to be modified in depth because the framework of [40] enables to use Ramsey theorem directly. This is because, under the assumption that the given language is in LD, one can describe this language by a *finite* collection of "good balls" of some constant radius.

This line of reasoning does not work for languages in BPLD_p. Nevertheless, we observe that, for reducing the study to a finite number of algorithms, we only need order-invariance applied on languages with *finite support*, where a distributed language $\mathcal{L}$ has finite support if

$$\left|\left\{G = (V, E) \mid \exists x, y : V \to \{0,1\}^*,\ (G, (x, y)) \in \mathcal{L}\right\}\right| < \infty.$$

In other words, there are finitely many graphs G for which there exists (x, y) such that $(G, (x, y)) \in \mathcal{L}$. For language with finite support, we can fortunately use Ramsey's theorem, even for languages in $\mathrm{BPLD}_p \setminus \mathrm{LD}$, and get the desired order-invariance property.

Second step: amplifying the probability of failure. Starting from a collection of small instances on which the algorithm is failing with probability at least $\beta > 0$, the objective of the second step is to construct a large configuration for which the probability of failure of the algorithm will exceed the error guarantee of the construction algorithm. If connectivity of the considered networks is not enforced, then it is easy to construct this large configuration, by merely taking the disjoint union of all the small instances. However, constructing a large *connected* configuration requires more work. The basic idea is to glue together the small configurations, resulting in a large configuration on which the failure probability is arbitrarily large. There are however several difficulties to overcome:

(1) First, the large configuration must have a proper identifier assignment to the nodes, that is, one cannot glue two small instances in which the same ID appear;
(2) Second, the large configuration must be connected, that is, taking the disjoint union of the small configurations is not sufficient.

The issue related to the identifiers can be overcome thanks to a careful control of the identifiers assigned to the small configurations selected during the first step of the proof. The issue related to connectivity requires more work. The small instances are glued together by connecting them with new edges connecting nodes in different instances. However, an immediate effect of adding edges is that the neighborhood of some nodes is modified, which may result in the construction algorithm $\mathbf{C}$ having different behaviors at these nodes in the small and large instances, which prevents us from arguing that if a node errs in a small instance, then it also errs in the large instance. This issue can be rather easily handled in the case of languages in LD. Indeed, if $\mathbf{C}$ is incorrect on some small configuration (H_i, x_i), then $\mathbf{C}$ constructs an illegal ball around some node $v_i \in V(H_i)$. Therefore, as long as H_i is connected to the other small configurations without modifying the neighborhood of v_i, $\mathbf{C}$ still constructs an illegal ball around v_i in the large configuration. However, as we mentioned before, the notion of legal/illegal ball is irrelevant for languages in BPLD, which deal with predicates that are non-local.

To tackle this problem, we argue that in each small configuration (H_i, x_i), there must exist a node v_i such that modifying its neighborhood does not decrease too much the probability that the decision algorithm witnessing the membership of the language $\mathcal{L}$ to BPLD rejects on the nodes of H_i. For establishing the existence of v_i, we consider a set of scattered nodes, i.e., each node is far enough from the others, and we show that one of these nodes has the desired property, due to the fact that the language is in BPLD. The constraint that these nodes are far from each other is required for ensuring statistical independence for the randomized algorithm deciding the language $\mathcal{L}$. The reason why we can indeed find a convenient set of scattered nodes follows from the fact that we assumed that the number of nodes, and therefore the diameter (recall that configurations in $\mathcal{F}_{\Delta,k}$ have bounded maximum degree), is large enough.

The details of these arguments are provided in Section 4.

3.3 Proof of Theorem 3.2 assuming Theorem 3.1

Let $\mathcal{L}$ be a distributed language in BPLD$_p$, let $\Delta > 2$ and $k \geq 0$ be integers, and suppose there exists a randomized Monte-Carlo construction algorithm with constant success probability $r > 0$ for $\mathcal{L}$ with configurations taken from $\mathcal{F}_{\Delta,k}$, running in constant time t. By Theorem 3.1, there exists an integer n_0, such that there exists a deterministic t-round algorithm constructing a solution for $\mathcal{L}$, assuming that the actual configuration has at least n_0 nodes. Now, consider the following algorithm. The algorithm starts by gathering at every node all the information at distance $n_0 + 1$ from that node. Every node then checks whether the network has size at least n_0 or not. Note that all the nodes will agree on the answer. Indeed, a node which has not gathered all the nodes and edges of the graph after $n_0 + 1$ rounds knows that the graph has at least n_0 nodes, and a node that has gathered all the nodes and edges of the graph after $n_0 + 1$ can tell whether the graph has at least n_0 nodes or not.

- If the network has less than n_0 nodes, then every node lets the node with smallest ID in the network computing a global solution for that instance, which is broadcasted to all the nodes in at most D additional rounds, where $D \leq n_0$ is the diameter of the network.
- Otherwise, every node executes the t-rounds algorithm given by Theorem 3.1.

This algorithm has round-complexity $O(n_0 + t)$, where both parameters are independent on n. □

4 PROOF OF THEOREM 3.1

4.1 Order-invariance

We first prove a result about order-invariant algorithms. This allows us to restrict the study of deterministic algorithms to the study of order-invariant algorithms (see Section 4.3). The following lemma is the analogue, in our setting, of Lemma 3.2 in [40]. As said earlier, we restrict our attention to languages with finite support. Recall that such languages are those such that there is a finite number of graphs G for which there exists x, y such that $(G, (x, y))$ is in the language.

Lemma 4.1. *Let $\mathcal{L}$ be a distributed language with finite support, and let $\Delta \geq 2$, $k \geq 0$, and $t \geq 0$ be three integers. There exists a constant R such that, for every set $\mathcal{S}$ of integers with $|\mathcal{S}| \geq R$, the following holds. If there exists a deterministic construction algorithm for $\mathcal{L}$ with configurations taken from $\mathcal{F}_{\Delta,k}$ running in t rounds on every ID-assignment with IDs taken from $\mathcal{S}$, then there exists an order-invariant deterministic construction algorithm for $\mathcal{L}$ with configurations taken from $\mathcal{F}_{\Delta,k}$ also running in t rounds.*

In order to establish Lemma 4.1, recall a variant on Ramsey's theorem (cf., e.g., [27]). For any set X, and for any positive integer ρ, let $X^{(\rho)}$ denote the set of all subsets of X with cardinality ρ. For any finite set X, and any two positive integers ρ and σ, a *coloring* of each set in $X^{(\rho)}$ by an integer in $[\sigma] = \{1, \ldots, \sigma\}$ is a function $c : X^{(\rho)} \to [\sigma]$.

Theorem 4.2 (Ramsey's Theorem). *For every triple of positive integers ρ, σ, and τ, there exists a constant $R = R(\rho, \sigma, \tau)$ such that if $|X| \geq R$, then, for every coloring $c : X^{(\rho)} \to [\sigma]$, there exists $Y \subseteq X$ with $|Y| = \tau$, and all sets in $Y^{(\rho)}$ are colored the same by c.*

Proof of Lemma 4.1. Let $\mathcal{L}$ be a distributed language with finite support, and Δ, k, t be nonnegative integers as in the statement of the lemma. Let $\mathcal{G}_{\mathcal{L}}$ be the (finite) family of graphs G for which there exists a configuration $(G, (x, y))$ in $\mathcal{L}$. Let $n_{\max}$ be the maximum number of vertices of the graphs in $\mathcal{G}_{\mathcal{L}}$. Let $\mathcal{S}$ be a finite set of integers with $|\mathcal{S}| \geq n_{\max}$. Let us assume that there exists a (deterministic) construction algorithm $\mathbf{A}$ for $\mathcal{L}$ with configurations taken from $\mathcal{F}_{\Delta,k}$ running in t rounds on every ID-assignment with IDs taken from $\mathcal{S}$.

Ordered labeled balls. We consider what we call *ordered labeled balls*. A ball is a graph B with one selected node, called the *center*, such that every node is at distance at most t from the center. We say that t is the *radius of the ball*. The nodes of the balls have degree at most Δ. A *labeled ball* is a ball B where every node v holds an input label $x(v)$ on at most k bits. Finally, an *ordered labeled ball* is a labeled ball (B, x) along with an ordering π of its nodes (i.e., $\pi : V(B) \to \{1, \ldots, |V(B)|\}$ is onto). The following claim directly follows from the definition.

Claim 1. *There is a finite number of ordered labeled balls* (B, x, π).

Let ν be the number of ordered labeled balls. We enumerate the ordered labeled balls as $(B_1, x_1, \pi_1), \ldots, (B_\nu, x_\nu, \pi_\nu)$ in an arbitrary order. Note that it may be the case that, for $i \neq j$, $B_i = B_j$, or even $(B_i, x_i) = (B_j, x_j)$. Let m be the maximum number of nodes in the balls, i.e., every ball B_i has at most m nodes.

Coloring. We now define a coloring of the sets in $\mathcal{S}^{(m)}$. Let $U \in \mathcal{S}^{(m)}$. Let $1 \leq q \leq \nu$, and let us consider (B_q, x_q, π_q). We set $c_q(U)$ as the output of $\mathbf{A}$ at the center of the ball B_q, where the nodes of B_q are assigned inputs thanks to x_q, and are given IDs equal to the $|V(B_q)|$ first values in U, respecting the ordering π_q. Note that $c_q(U)$ can take at most 2^k different values, as it is an output of $\mathbf{A}$ for $\mathcal{F}_{\Delta,k}$, hence on k bits. Finally the color of U is the ν-tuple

$$c(U) = (c_1(U), c_2(U), \ldots, c_\nu(U)).$$

Applying Ramsey theorem. We apply Ramsey's theorem (Theorem 4.2) on the coloring c of $\mathcal{S}^{(m)}$, as specified in Theorem 4.2, with

$$\rho = m, \ \sigma = 2^{k\nu}, \ \text{and} \ \tau = n_{\max}.$$

By picking $\mathcal{S}$ with $|\mathcal{S}| \geq R(r, s, \tau)$, we get that there exists a subset Y of $\mathcal{S}$ with size $n_{\max}$ such that, for every subset U of Y with size m, the color $c(U)$ given to U is the same. In other words, there exists a set of identifiers Y such that the algorithm $\mathbf{A}$ outputs the same at all centers of the balls for every ID-assignment with IDs taken from Y, whenever these IDs are assigned according to the ordering of the nodes specified for each ball. That is, the algorithm $\mathbf{A}$ is order-invariant when restricted to identifiers taken from Y.

New algorithm $\mathbf{A}'$. Let us now consider a new algorithm, denoted by $\mathbf{A}'$, that gathers the t-neighborhood at each node, replaces the observed identifiers by the first identifiers in Y, respecting the ordering provided by the original IDs, and outputs the same as $\mathbf{A}$ for this t-neighborhood but with IDs in Y. The algorithm $\mathbf{A}'$ is order-invariant by construction, since it maps every identifier assignment with a given order to a same ID assignment with IDs taken from Y.

Correctness of $\mathbf{A}'$. To prove that $\mathbf{A}'$ is correct, let us consider a run of $\mathbf{A}'$ on a configuration (G, x) with $G \in \mathcal{G}_{\mathcal{L}}$ and for which there exists y such that $(G, (x, y)) \in \mathcal{L}$, assuming an arbitrary ID assignment id_1. We show that $\mathbf{A}'$ outputs y such that $(G, (x, y)) \in \mathcal{L}$, which establishes the correctness of $\mathbf{A}'$. Let id_2 be an identifier assignment to the nodes of G with IDs taken from Y, respecting the global ordering of the nodes induced by id_1. Note that Y is large enough as $|Y| = n_{\max}$ where n_{max} is the maximum number of nodes in a graph in $\mathcal{G}_{\mathcal{L}}$. Let v be a node of G. While executing $\mathbf{A}'$, node v assigns IDs from Y to the nodes in its t-neighborhood, respecting the local ordering of the nodes induced by id_1. Let us denote this (local) identifier assignment by id_3. By construction, id_2 and id_3 have the same ordering in the t-neighborhood of v. Therefore, as the IDs in id_2 and id_3 are taken from Y, the output of $\mathbf{A}$ at node v is the same for these two identifier assignments. Since this holds for every node, it follows that the global output y of $\mathbf{A}'$ with id_1 is the same as the output y of $\mathbf{A}$ with id_2. Since $\mathbf{A}$ is correct, we get $(G, (x, y)) \in \mathcal{L}$, and therefore $\mathbf{A}'$ is correct, as desired. This completes the proof of Lemma 4.1. □

4.2 Proof setting

Let $\mathcal{L}$, p, Δ, k, t, and r be as in the statement of Theorem 3.1. Consider a randomized construction algorithm for $\mathcal{L}$ with configurations taken from $\mathcal{F}_{\Delta,k}$ running in t rounds, with success probability at least r. We prove that there exists an integer D_0 such that there exists a deterministic algorithm for $\mathcal{L}$ with configurations taken from $\mathcal{F}_{\Delta,k}$ running in t rounds, in all graphs of diameter at least D_0. The desired lower bound n_0 on the number of nodes can then be directly derived from D_0, as all considered graphs have maximum degree at most Δ. For the purpose of contradiction, let us assume the following.

($\star$) There exists a t-round randomized construction algorithm $\mathbf{C}$ for $\mathcal{L}$ with success probability at least r for configurations taken from $\mathcal{F}_{\Delta,k}$, but every t-round deterministic construction algorithm for $\mathcal{L}$ with configurations taken from $\mathcal{F}_{\Delta,k}$ errs on an infinite family of graphs (hence on graphs with arbitrarily large diameter).

Since r is the success probability of $\mathbf{C}$, we get that, for every input configuration (G,x) for which there exists y such that $(G,(x,y)) \in \mathcal{L}$, and for every identifier 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 $\mathbf{C}(G,x,\mathrm{id})$.

Randomized decision. Since $\mathcal{L} \in \mathrm{BPLD}_p$, there exists a randomized Monte-Carlo algorithm $\mathbf{D}$ deciding $\mathcal{L}$ in a constant number of rounds (see Section 2.3), and $p > \frac{1}{2}$ is the success guarantee of Algorithm $\mathbf{D}$, that is, $\mathbf{D}$ satisfies Eq. (1). We denote by $t' = O(1)$ the number of rounds of Algorithm $\mathbf{D}$.

4.3 Basic building block

The assumption ($\star$) allows to get the following result.

Claim 2. *For every integer $D_0 \geq 0$, there exists $\beta > 0$ such that, for every nonnegative integer $I_{\min}$, there exists a graph H with diameter $D \geq D_0$, an input-assignment $x : V(H) \to \{0,1\}^k$, and an identifier-assignment* id *to the nodes of H with* $\mathrm{id}(v) \geq I_{\min}$ *for every node v of H, for which*

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

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

Proof of Claim 2. Let D_0 be a nonnegative integer. Since we assume input-output configurations in $\mathcal{F}_{\Delta,k}$, there is a finite number of (deterministic) order-invariant algorithms running in t rounds. This is because there are finitely many balls of radius t in a graph of maximum degree k, finitely many k-bit input-output configurations for each of these balls, and finitely many orderings for the node identifiers in these balls.

Let $I_{\min} \geq 0$ be an integer. By assumption ($\star$), for each order-invariant deterministic algorithm $\mathbf{A}$, we can pick an instance (H,x,id), with diameter $D \geq D_0$, such that $\mathbf{A}$ fails on (H,x,id). As there is a finite number of order-invariant algorithms, this process generates a finite number of instances. Let $n_{\max}$ be the maximum number of nodes of H across all instances (H,x,id) obtained after having considered all order-invariant algorithms.

As Algorithm $\mathbf{A}$ is order-invariant, we can actually modify the identifier assignment id at will, as long as we keep the same global order of the identifiers, and still get an instance where $\mathbf{A}$ fails. In particular, we ask that the minimum identifier is $I_{\min}$ and that the maximum identifier is $I_{\min} + n_{\max} - 1$. Now, consider the set $\mathcal{H}$ of all the configurations on graphs with diameter at least D_0 and on at most $n_{\max}$ nodes, arbitrary inputs, and identifier assignment in $[I_{\min}, I_{\min} + n_{\max} - 1]$.

Here we will assume that D_0 is larger than the constant R of Lemma 4.1 resulting from the choice of Δ and k in the statement of Theorem 3.1, and the choice of t in ($\star$). In turn, this implies that n_{max} is larger than R.

This (finite) set contains all the graphs H occurring in the configurations (H, x, id) previously selected by considering all order-invariant algorithms. Let N be the cardinality of $\mathcal{H}$. We set

$$\beta = 1/N.$$

Note that β does not depend on $I_{\min}$, as the identifiers in $\mathcal{H}$ are taken from a set of size $n_{\max}$ for any $I_{\min}$, and $n_{\max}$ does not depend on $I_{\min}$.

At this point, there are two possibilities. The first possibility is that, for every random bit sequence, there exists an instance (H, x, id) on which C fails, and then, given that there are at most N instances (H, x, id), there exists an instance $(H, x, \mathrm{id}) \in \mathcal{H}$ such that

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

The second possibility is that there exists a random bits sequence σ such that, for every instance $(H, x, \mathrm{id}) \in \mathcal{H}$,

$$\mathbf{C}(H, x, \mathrm{id}) \text{ is correct with the sequence } \sigma.$$

We show that the later case is impossible. Indeed, if there exists σ such that, for every instance $(H, x, \mathrm{id}) \in \mathcal{H}$, $\mathbf{C}(H, x, \mathrm{id})$ is correct with the sequence σ, then there exists a deterministic algorithm for all instances in $\mathcal{H}$. We can turn this algorithm into an order-invariant algorithm by applying Lemma 4.1. Indeed, the ID space has size $n_{\max}$, thus is larger than R, and the language at hand is $\mathcal{L}$ restricted to the graph that are present in $\mathcal{H}$, which then has finite-support. Therefore, there exists an order-invariant algorithm in time t that is correct for all instances in $\mathcal{H}$,and thus in particular correct for all the instances (H, x, id), a contradiction. It follows that there exists $(H, x, \mathrm{id}) \in \mathcal{H}$ such that $\Pr[\mathbf{C}(H, x, \mathrm{id}) \notin \mathcal{L}] \geq \beta$, which completes the proof of Claim 2. □

4.4 Boosting the error probability

To provide an intuition of how the assumption $\mathcal{L} \in \mathrm{BPLD}_p$ is used, let us first provide an intuition of the rest of the proof of Theorem 3.1 by relaxing the constraint that input-output configurations deal with connected graphs. For this purpose, let us define BPLD^*_p as the class of languages defined as BPLD_p but on configurations $(G, (x, y))$ where G does not need to be connected. The following result is a relaxed variant of Theorem 3.1 in which distributed languages are allowed to be defined on non-connected graphs. For this variant, the issue regarding the diameter is irrelevant (as this constraint is precisely used to connect small graphs together to form a large *connected* configuration), hence we simply ignore it. Note that the parameter p of the BPLD_p decider does not actually need to be larger than $1/2$ for the results in this section to hold.

Claim 3. *Let $\mathcal{L}$ be a distributed language in* BPLD^**, and let $\Delta > 2$ and $k \geq 0$ be integers. If there exists a randomized Monte-Carlo construction algorithm with success probability $r \in (0, 1]$ for $\mathcal{L}$ with configurations taken from $\mathcal{F}_{\Delta,k}$, running in t rounds, then there exists a deterministic construction algorithm for $\mathcal{L}$ with configurations taken from $\mathcal{F}_{\Delta,k}$ running in t rounds.*

Proof of Claim 3. The assumption that graphs do not need to be connected has no impact on Lemma 4.1 and Claim 2, which remain both valid in this context. Let β be as in Claim 2 for $D_0 = 0$, and let ν be an integer that we will fixed later.

We use Claim 2 several times, for different values of $I_{\min}$. Let's start with $I_1 = 0$. Let $(H_1, x_1, \mathrm{id}_1)$ be an instance whose existence is guaranteed by Claim 2 for $I_{\min} = I_1$. Now let $I_2 = 1 + \max_{v \in V(H_1)} \mathrm{id}(v)$, and, let $(H_2, x_2, \mathrm{id}_2)$ be an instance whose existence is guaranteed by Claim 2 for $I_{\min} = I_2$. We can

carry on with this 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 $(H_i, x_i, \mathsf{id}_i)$. Note that id is well defined since two different identifier assignments id_i and id_j do not overlap, for any $1 \leq i < j \leq \nu$. We now study the behaviour of **C** and **D** on these instances. For a fixed i, the probability that **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, by definition of **D**,

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

Therefore

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

and thus

$$\Pr[\mathbf{D} \text{ accepts } \mathbf{C}(G, x, \mathsf{id})] \leq (1 - \beta p)^{\nu} \tag{3}$$

since the decider runs independently in each $(H_i, x_i, \mathsf{id}_i)$. On the other hand,

$$\Pr[\mathbf{D} \text{ accepts } \mathbf{C}(G, x, \mathsf{id})] \geq p \Pr[\mathbf{C}(G, x, \mathsf{id}) \in \mathcal{L}]. \tag{4}$$

Combining Eq. (3) with Eq. (4), we get

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

By taking ν large enough, it follows that

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

This is a contradiction with Eq. (2), i.e., the fact that **C** has success probability at least r. This contradiction indicates that hypothesis ($\star$) cannot be true for $\mathcal{L}$. This concludes the proof of Claim 3. □

4.5 Gluing of instances

In this subsection, we show how to amplify the probability of failure by repetition, as in the previous section, while preserving the connectivity of the considered instances. Recall that $p > \frac{1}{2}$. We define

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

and we set

$$D_0 = 2\mu(t + t' + 1),$$

where t and t' are the running times of **C** and **D**, respectively. Using Claim 2 with this value for D_0, we construct a sequence of instances $(H_i, x_i, \mathsf{id}_i)$, $i = 1, \ldots, \nu'$, for an integer ν' to be specified later, the same way we did in the proof of Claim 3. All the graphs H_i have diameter at least D_0. 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 $i \in \{1, \ldots, \nu'\}$. A crucial remark is that, since the diameter of H_i is at least $D_0 = 2\mu(t + t' + 1)$, there exists a set S of μ vertices in H_i at distance at least $2(t + t' + 1)$ from each other. We now show how to choose the desired node u_i as one of the nodes in S.

Let us analyze the execution of the construction algorithm **C** for each of the possible choices of its random coins, and let us restrict our attention to the behavior of the nodes far away from u when running the decider **D**, for each $u \in S$. More specifically, any Monte-Carlo algorithm such as **C** or **D** is using a finite (potentially unbounded) random bit-string at each node. The collection of random

bit-strings consumed by all the nodes during one execution of the algorithm forms a multi-set of random strings indexed by node identifiers. This set can be viewed itself as a (large) random bit-string composed of the concatenation of the individual strings, ordered by the node identifiers. Let Rand($\mathbf{C}$) and Rand($\mathbf{D}$) be the sets of random bit-strings used by $\mathbf{C}$ and $\mathbf{D}$, respectively.

Recall that the event "$\mathbf{D}$ rejects $(G,(x,y))$" is an abbreviation for the event "$\mathbf{D}$ outputs $false$ in at least one node when executed on $(G,(x,y))$". We define the event "$\mathbf{D}$ rejects $(G,(x,y))$ *far from* $v \in V(G)$" as the event that $\mathbf{D}$ outputs $false$ in at least one node at distance greater than $t+t'$ from v, when executed on $(G,(x,y))$. Also, 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 $(G,(x,y))$. Note that it can be the case that $\mathbf{D}$ rejects $(G,(x,y))$ but that $\mathbf{D}$ accepts $(G,(x,y))$ far from $v \in V(G)$. Let us fix a string $\sigma \in \text{Rand}(\mathbf{C})$ that makes $\mathbf{C}$ fail on instance (H_i,x_i,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 a node $u \in S$ such that* $\Pr[\mathbf{D}$ *accepts* $\mathbf{C}_\sigma(H_i,x_i,\text{id}_i)$ *far from* $u] < p$.

Proof of Claim 4. To establish the claim, notice that, since $\mathbf{C}_\sigma(H_i,x_i,\text{id}_i) \notin \mathcal{L}$, it follows that

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

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

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

Let $\mathcal{E}_u$ be the event that $\mathbf{D}$ rejects $\mathbf{C}_\sigma(H_i,x_i,\text{id}_i)$, and accepts $\mathbf{C}_\sigma(H_i,x_i,\text{id}_i)$ far from u. By a union bound, it follows from $|S|=\mu$ 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 sets 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

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

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

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

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

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

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 allowing us to connect H_i to the other H_j's.

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

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

Proof of Claim 5. To establish the claim, let

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

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

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

We have

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

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

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

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,

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

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

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

This concludes the proof of 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 arbitrarily chosen 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 Δ as the degrees of the original nodes remain unchanged, and all new nodes v_i and w_i, $i = 1, \ldots, \nu'$, have degree 3 (recall that $\Delta > 2$). The inputs and indentifiers given to the nodes of G not in some H_i are set arbitrarily (with the only constraint that no identifiers 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

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

On the one hand, we have

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

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 $2(t + t' + 1)$ 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

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

On the other hand, as in the proof of Claim 3, we have

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

Combining the two latter inequalities, we get

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

If follows that, by choosing

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

we obtain

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

which is a contradiction with Eq. (2), i.e., with 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.

4.6 Small error detection probabilities, and graphs with small maximum degree

We conclude this section by a couple of remarks.

4.6.1 On the error detection probability. Our derandomization result holds for languages in BPLD_p with success probability $p > \frac{1}{2}$. The following language $\mathcal{L}$ justifies this threshold. $\mathcal{L}$ is defined solely for configurations in $\mathcal{F}_{2,2}$, i.e., on graphs with maximum degree 2, and output labels on at most 2 bits (there are no input labels in $\mathcal{L}$). Every node u of a graph G must output a 2-bit integer $y(u) \in \{0,1,2,3\}$. If G is a cycle, then there are no contraints on these integers. However, if the graph G is a path, then the labels of its two endpoints u' and u'' must satisfy $y(u') \neq y(u'')$.

We have $\mathcal{L} \in \mathrm{BPLD}_{1/2}$ using the following decision algorithm. Every degree-2 node accepts with probability 1, and every degree-1 node accepts with probability $1/\sqrt{2}$. It follows that if G is a cycle then $\Pr[\text{all nodes accept}] = 1$, and if G is a path, then $\Pr[\text{all nodes accept}] = \frac{1}{2}$. Therefore, a legal instance is accepted with probability at least $\frac{1}{2}$, and an illegal instance is rejected with probability $\frac{1}{2}$.

On the one hand, $\mathcal{L}$ has a randomized construction algorithm, performing in zero rounds, and succeeding with probability $\frac{3}{4}$. This algorithm simply outputs an integer $y(u) \in \{0,1,2,3\}$ chosen uniformly at random at every degree-1 node u, and outputs 0 at every degree-2 node. In a path with endpoints u' and u'', the probability that $y(u') \neq y(u'')$ is $\frac{3}{4}$.

On the other hand, $\mathcal{L}$ has no deterministic construction algorithm performing in $O(1)$ rounds. To see why, let us assume, for the purpose of contradiction, the existence of a construction algorithm for $\mathcal{L}$ performing in a constant number of rounds t. In an n-node path P with $n \geq 2t+2$, the algorithm is merely a function f that maps views at distance t of the endpoints of P to $\{0,1,2,3\}$. Such a view is a word $w = x_0, \ldots, x_t$ of $t+1$ distinct IDs. For every $k \geq 0$, we define the view $w_k = k(t+1)+1, \ldots, (k+1)(t+1)$. Since f has four possible outcomes, there are two distinct indices $i, j \in \{0, \ldots, 4\}$ such that $f(w_i) = f(w_j)$. Therefore, if P has $n \geq 2t+2$ nodes, then one can assign the IDs to P in such a way that its two endpoints have respective views w_i and w_j, leading

the algorithm to output the same value at the two endpoints of P, contradicting its correctness. Note that this holds even for IDs in $\{1, \ldots, n\}$, whenever $n \geq 5(t+1)$.

It follows that there are languages in $\text{BPLD}_{1/2}$ that have a randomized construction algorithm performing in $O(1)$ rounds and outputting a correct solution with probability $\frac{3}{4}$, but that do not admit any deterministic construction algorithm performing in $O(1)$ rounds. Our derandomization result however holds for configurations taken from $\mathcal{F}_{\Delta,k}$, i.e., on the class of graphs with maximum degree at most Δ, with $\Delta > 2$. (Remember that for every for every graph we require that there exists a correct solution, thus we cannot simply rule out instances with larger degree.) For such an assumption, we do not know whether a success probability $p > \frac{1}{2}$ is required for checking the correctness of a solution, and it may well be the case that even a small (constant) probability of success $p > 0$ suffices for a derandomization result to hold. Establishing the correctness of this extension would require a more refined analysis than the one used for Claim 4.

4.6.2 On graphs with maximum degree 2. The derandomization result by Naor and Stockmeyer [40] (cf. Theorem 2.1) extends from the class of graphs with maximum degree Δ, with $\Delta > 2$, to the class of 2-regular graphs. Similarly, assuming $p > \frac{1}{2}$, our derandomization result stated for graphs with maximum degree Δ, with $\Delta > 2$, also holds the class of 2-regular graphs. Indeed, Claim 5 can also be applied in this setting for gluing small cycles together, in order to create a large cycle in which the construction algorithm fails with a probability exceeding its error threshold. Nevertheless, the specific case of the class of graphs with degree at most 2 is not covered by our theorem. We do not know whether derandomization holds for this class. Indeed, there is a specific reason why this class behaves differently from any class of graphs with maximum degree $\Delta > 2$. Specifically, the only way to glue paths while remaining in the class is to connect these paths via their endpoints. Unfortunately, Claim 5 does not say anything about the location of the nodes of H_i enabling to connect H_i with other H_js for creating a large graph in which the construction algorithm fails. In particular, if the rejecting node is too close to an endpoint, one cannot use it.

In fact, there are scenarios in which glueing paths is impossible. Let us again consider the language amos, for "at most one selected", in which either zero or one node can be marked 1, while all the other nodes are marked 0. We actually consider the variant amos′ of amos for $\mathcal{F}_{2,1}$ in which only degree-1 nodes matter (i.e., one does not count the number of marked degree-2 nodes). Note that there exists a randomized construction algorithm for amos′ that succeeds with probability $\frac{3}{4}$ under $\mathcal{F}_{2,1}$, performing as follows. Each degree-2 node outputs 0 with probability 1, and each degree-1 node outputs 0 or 1 uniformly at random. The probability that the two extremities of a path output 1 is $\frac{1}{4}$.

Now, deciding amos′ can be done in zero rounds, as follows: each degree-2 node accepts regardless of whether it is marked or not, each unmarked degree-1 node accepts with probability 1, and each marked degree-1 node accepts with probability $p = (\sqrt{5}-1)/2$, and rejects with probability $1-p$. We get amos′ $\in \text{BPLD}_p$ with success probability $p = (\sqrt{5}-1)/2 > \frac{1}{2}$. However, the rejecting nodes are systematically extremities of paths, preventing gluing to be unnoticed by these nodes. It follows that our proof does not enable to show the existence of a deterministic construction algorithm for amos′, and the fact that there exists a trivial deterministic algorithm for amos′ (every node merely outputs 0) is of no help as far as our proof is concerned.

We actually do not know whether our derandomization result for $\mathcal{F}_{\Delta,k}$, $\Delta > 2$, for languages in BPLD_p with $p > \frac{1}{2}$, extends to $\mathcal{F}_{2,k}$.

5 RESILIENT RELAXATIONS

In this section, we derive lower bounds for relaxed construction tasks. In a framework of dynamic networks, in which links can be removed or added along with time, a modification of the network

may turn a legal LCL into an illegal one. As a consequence, any application based on an LCL on dynamic networks should tolerate at least some erroneous labelings of the nodes, here and there in the network. For such applications, it might therefore be overdoing to construct perfect labelings, as slightly erroneous labeling would be sufficient. In this section, we question the ability to construct LCLs efficiently when relaxing the task by tolerating a certain number of erroneous neighborhoods. Let $Bad(\mathcal{L})$ be the set of "bad" balls for $\mathcal{L}$.

Definition 5.1. The f-resilient relaxation of $\mathcal{L} \in$ LCL for a positive integer f, 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 algorithms for $\mathcal{L}_f$. Note that this result does not follow from the derandomization result in [40] since $\mathcal{L}_f$ is not necessarily locally checkable (i.e., not necessarily in LD). The following is however a corollary of Theorem 3.2.

Corollary 5.2. *Let $\mathcal{L} \in$ LCL, $\Delta > 2$, $k \geq 0$, and $f > 0$. If there exists a randomized Monte-Carlo construction algorithm for $\mathcal{L}_f$ with configurations taken from $\mathcal{F}_{\Delta,k}$, running in $O(1)$ rounds, then there exists a deterministic construction algorithm for $\mathcal{L}_f$ with configurations taken from $\mathcal{F}_{\Delta,k}$ running in $O(1)$ rounds.*

Proof. It is sufficient to show that $\mathcal{L}_f \in \text{BPLD}_p$ for some $p > \frac{1}{2}$, and the result will then follow directly from Theorem 3.2. Let

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

The randomized decision algorithm performs in t rounds, where t is the radius of the balls excluded from $\mathcal{L}$. The decision algorithm works as follows. In instance $(G, (x, y), \text{id})$, 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 with probability 1. 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 for configurations in $\mathcal{F}_{\Delta,k}$, and therefore $B_G(v, t) \in Bad(\mathcal{L})$ is decidable (in the usual sense of sequential computing). Given $(G, (x, y), \text{id})$, 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, $\mathcal{L} \in \text{BPLD}_p$ with $p > \frac{1}{2}$, as desired. □

As a simple illustration of Corollary 5.2, we get that the f-relaxation of $(\Delta+1)$-coloring cannot be solved in constant time, even if using a randomized Monte-Carlo algorithm. Indeed, this language belongs to LCL, and therefore, by Corollary 5.2, it is sufficient to show that there is no constant-time deterministic algorithm solving the f-relaxation of $(\Delta + 1)$-coloring. By Lemma 4.1 in the proof of Theorem 3.1, 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 identifiers from 1 to n (except for nodes with IDs 1 and n), any order-invariant 3-coloring algorithm performing in $t = O(1)$ rounds acts identically in at least $n - 2t$ nodes since all the balls centered at nodes with

IDs in $[t+1, n-t]$ are identical as far as the ordering of the identifiers are concerned. Therefore, $n-2t$ nodes output the same color, and thus the algorithm cannot be f-resilient.

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 hold for all languages in BPLD_p. Even for languages in LD, local fixing might not be easy. For instance, *frugal coloring* is the task requiring to properly 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 is not fully local (as it is the case for the languages in $\mathrm{BPLD}_p \setminus \mathrm{LD}$) is not necessarily straightforward.

6 OPEN PROBLEMS

A natural question raised by this paper is whether

$$\bigcup_{p>1/2} \mathrm{BPLD}_p$$

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 non-deterministic classes of languages that can be locally *verified*, deterministically or randomly, respectively (see [18]). 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 [18]). 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 Theorems 3.1 and 3.2 to tasks in one of the classes LD^{O}, NLD, or BPNLD, requires to overcome a serious obstacle. When constructing larger instances by gluing 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 Theorems 3.1 and 3.2 do not extend to $\mathrm{BPLD}_p^{\#\mathrm{node}}$, 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 $\mathrm{BPLD}_p^{\#\mathrm{node}}$ (using the same algorithm as in the proof of Corollary 5.2 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 our results to tasks in restricted classes of networks. Theorems 3.1 and 3.2 extend to tasks for configurations restricted to graph classes such as planar graphs, or 2-connected graphs. Indeed, the construction in the proof of the theorem preserves planarity and 2-connectivity. We do not know whether our theorems can be generalized to languages restricted to arbitrary infinite class of graphs.

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_p (hence our theorems cannot be used). They are in $\mathrm{BPLD}_p^{\#\mathrm{node}}$, but we have seen that this latter class is too wide for a derandomization result such as the one in [40] to hold. Nevertheless, there might be a class C, with

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

for which Theorems 3.1 and 3.2 could be extended.

ACKNOWLEDGMENTS

Both authors were supported by the ANR project DESCARTES, and the INRIA Project GANG. The authors thank the reviewers for their very careful, detailed and helpful reviews.

REFERENCES

[1] Ittai Abraham, Yair Bartal, and Ofer Neiman. 2006. Advances in metric embedding theory. In *38th ACM Symposium on Theory of Computing (STOC)*. 271–286. https://doi.org/10.1145/1132516.1132557

[2] Heger Arfaoui and Pierre Fraigniaud. 2014. What can be computed without communications? *SIGACT News* 45, 3 (2014), 82–104. https://doi.org/10.1145/2670418.2670440

[3] Heger Arfaoui, Pierre Fraigniaud, David Ilcinkas, and Fabien Mathieu. 2014. Distributedly Testing Cycle-Freeness. In *40th Int. Workshop on Graph-Theoretic Concepts in Computer Science (WG)*. 15–28. https://doi.org/10.1007/978-3-319-12340-0_2

[4] Leonid Barenboim and Michael Elkin. 2013. *Distributed Graph Coloring: Fundamentals and Recent Developments*. Morgan & Claypool Publishers. https://doi.org/10.2200/S00520ED1V01Y201307DCT011

[5] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. 2016. The Locality of Distributed Symmetry Breaking. *J. ACM* 63, 3 (2016), 20:1–20:45. https://doi.org/10.1145/2903137

[6] Hubert T.-H. Chan, Michael Dinitz, and Anupam Gupta. 2006. Spanners with Slack. In *14th European Symposium on Algorithms (ESA)*. 196–207. https://doi.org/10.1007/11841036_20

[7] T.-H. Hubert Chan, Kedar Dhamdhere, Anupam Gupta, Jon M. Kleinberg, and Aleksandrs Slivkins. 2009. Metric Embeddings with Relaxed Guarantees. *SIAM J. Comput.* 38, 6 (2009), 2303–2329. https://doi.org/10.1137/060670511

[8] Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. 2019. An Exponential Separation between Randomized and Deterministic Complexity in the LOCAL Model. *SIAM J. Comput.* 48, 1 (2019), 122–143. https://doi.org/10.1137/17M1117537

[9] Moses Charikar, Samir Khuller, David M. Mount, and Giri Narasimhan. 2001. Algorithms for facility location problems with outliers. In *12th ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 642–651. http://dl.acm.org/citation.cfm?id=365411.365555

[10] Ke Chen. 2008. A constant factor approximation algorithm for k-median clustering with outliers. In *19th ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 826–835. http://dl.acm.org/citation.cfm?id=1347082.1347173

[11] Kai-Min Chung, Seth Pettie, and Hsin-Hao Su. 2017. Distributed algorithms for the Lovász local lemma and graph coloring. *Distributed Computing* 30, 4 (2017), 261–280. https://doi.org/10.1007/s00446-016-0287-6

[12] Marek Cygan and Tomasz Kociumaka. 2014. Constant Factor Approximation for Capacitated k-Center with Outliers. In *31st International Symposium on Theoretical Aspects of Computer Science (STACS) (LIPIcs)*, Vol. 25. 251–262. https://doi.org/10.4230/LIPIcs.STACS.2014.251

[13] Michael Dinitz. 2007. Compact routing with slack. In *26th ACM Symposium on Principles of Distributed Computing (PODC)*. 81–88. https://doi.org/10.1145/1281100.1281114

[14] Yuval Emek, Christoph Pfister, Jochen Seidel, and Roger Wattenhofer. 2014. Anonymous networks: randomization = 2-hop coloring. In *ACM Symposium on Principles of Distributed Computing (PODC)*. 96–105. https://doi.org/10.1145/2611462.2611478

[15] Patrik Floréen, Joel Kaasinen, Petteri Kaski, and Jukka Suomela. 2009. An optimal local approximation algorithm for max-min linear programs. In *21st ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 260–269. https://doi.org/10.1145/1583991.1584058

[16] Pierre Fraigniaud, Mika Göös, Amos Korman, Merav Parter, and David Peleg. 2014. Randomized distributed decision. *Distributed Computing* 27, 6 (2014), 419–434. https://doi.org/10.1007/s00446-014-0211-x

[17] Pierre Fraigniaud, Mika Göös, Amos Korman, and Jukka Suomela. 2013. What can be decided locally without identifiers?. In *ACM Symposium on Principles of Distributed Computing (PODC)*. 157–165. https://doi.org/10.1145/2484239.2484264

[18] Pierre Fraigniaud, Amos Korman, and David Peleg. 2013. Towards a complexity theory for local distributed computing. *J. ACM* 60, 5 (2013), 35:1–35:26. https://doi.org/10.1145/2499228

[19] Pierre Fraigniaud, Sergio Rajsbaum, and Corentin Travers. 2013. Locality and checkability in wait-free computing. *Distributed Computing* 26, 4 (2013), 223–242. https://doi.org/10.1007/s00446-013-0188-x

[20] Pierre Fraigniaud, Sergio Rajsbaum, and Corentin Travers. 2014. On the Number of Opinions Needed for Fault-Tolerant Run-Time Monitoring in Distributed Systems. In *5th International Conference on Runtime Verification (RV)*. 92–107. https://doi.org/10.1007/978-3-319-11164-3_9

[21] Zachary Friggstad, Kamyar Khodamoradi, Mohsen Rezapour, and Mohammad R. Salavatipour. 2019. Approximation Schemes for Clustering with Outliers. *ACM Trans. Algorithms* 15, 2 (2019), 26:1–26:26. https://doi.org/10.1145/3301446

[22] Cyril Gavoille, Adrian Kosowski, and Marcin Markiewicz. 2009. What Can Be Observed Locally?. In *23rd International Symposium on Distributed Computing (DISC)*. 243–257. https://doi.org/10.1007/978-3-642-04355-0_26

[23] Mohsen Ghaffari, David G. Harris, and Fabian Kuhn. 2018. On Derandomizing Local Distributed Algorithms. In *59th IEEE Symposium on Foundations of Computer Science (FOCS)*. 662–673. https://doi.org/10.1109/FOCS.2018.00069

[24] Mohsen Ghaffari, Fabian Kuhn, and Yannic Maus. 2017. On the complexity of local distributed graph problems. In *49th ACM Symposium on Theory of Computing (STOC)*. 784–797. https://doi.org/10.1145/3055399.3055471

[25] Mika Göös, Juho Hirvonen, and Jukka Suomela. 2013. Lower bounds for local approximation. *J. ACM* 60, 5 (2013), 39:1–39:23. https://doi.org/10.1145/2528405

[26] Mika Göös and Jukka Suomela. 2016. Locally Checkable Proofs in Distributed Computing. *Theory Comput.* 12, 1 (2016), 1–33. https://doi.org/10.4086/toc.2016.v012a019

[27] Ronald L Graham, Bruce L Rothschild, and Joel H Spencer. 1990. *Ramsey theory*. Vol. 20. John Wiley & Sons.

[28] Henning Hasemann, Juho Hirvonen, Joel Rybicki, and Jukka Suomela. 2016. Deterministic local algorithms, unique identifiers, and fractional graph colouring. *Theor. Comput. Sci.* 610 (2016), 204–217. https://doi.org/10.1016/j.tcs.2014.06.044

[29] Jon M. Kleinberg, Aleksandrs Slivkins, and Tom Wexler. 2009. Triangulation and embedding using small sets of beacons. *J. ACM* 56, 6 (2009), 32:1–32:37. https://doi.org/10.1145/1568318.1568322

[30] Goran Konjevod, Andréa W. Richa, Donglin Xia, and Hai Yu. 2007. Compact routing with slack in low doubling dimension. In *26th ACM Symposium on Principles of Distributed Computing (PODC)*. 71–80. https://doi.org/10.1145/1281100.1281113

[31] Amos Korman, Shay Kutten, and David Peleg. 2010. Proof labeling schemes. *Distributed Computing* 22, 4 (2010), 215–233. https://doi.org/10.1007/s00446-010-0095-3

[32] Amos Korman, Jean-Sébastien Sereni, and Laurent Viennot. 2013. Toward more localized local algorithms: removing assumptions concerning global knowledge. *Distributed Computing* 26, 5-6 (2013), 289–308. https://doi.org/10.1007/s00446-012-0174-8

[33] Ravishankar Krishnaswamy, Shi Li, and Sai Sandeep. 2018. Constant approximation for k-median and k-means with outliers via iterative rounding. In *50th ACM Symposium on Theory of Computing (STOC)*. 646–659. https://doi.org/10.1145/3188745.3188882

[34] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. 2004. What cannot be computed locally!. In *23rd ACM Symposium on Principles of Distributed Computing (PODC)*. 300–309. https://doi.org/10.1145/1011767.1011811

[35] Christoph Lenzen, Yvonne Anne Oswald, and Roger Wattenhofer. 2008. What can be approximated locally?: case study: dominating sets in planar graphs. In *20th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 46–54. https://doi.org/10.1145/1378533.1378540

[36] Christoph Lenzen and Roger Wattenhofer. 2008. Leveraging Linial's Locality Limit. In *22nd International Symposium on Distributed Computing (DISC)*. 394–407. https://doi.org/10.1007/978-3-540-87779-0_27

[37] Nathan Linial. 1992. Locality in Distributed Graph Algorithms. *SIAM J. Comput.* 21, 1 (1992), 193–201. https://doi.org/10.1137/0221015

[38] Thomas Moscibroda and Roger Wattenhofer. 2008. Coloring unstructured radio networks. *Distributed Computing* 21, 4 (2008), 271–284. https://doi.org/10.1007/s00446-008-0070-4

[39] Moni Naor. 1991. A Lower Bound on Probabilistic Algorithms for Distributive Ring Coloring. *SIAM J. Discrete Math.* 4, 3 (1991), 409–412. https://doi.org/10.1137/0404036

[40] Moni Naor and Larry J. Stockmeyer. 1995. What Can be Computed Locally? *SIAM J. Comput.* 24, 6 (1995), 1259–1277. https://doi.org/10.1137/S0097539793254571

[41] David Peleg. 2000. Distributed computing. *SIAM Monographs on discrete mathematics and applications* 5 (2000).

[42] Václav Rozhon and Mohsen Ghaffari. 2020. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In *52nd ACM Symposium on Theory of Computing (STOC)*. 350–363. https://doi.org/10.1145/3357713.3384298

[43] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. 2012. Distributed Verification and Hardness of Distributed Approximation. *SIAM J. Comput.* 41, 5 (2012), 1235–1265. https://doi.org/10.1137/11085178X

[44] Jukka Suomela. 2013. Survey of local algorithms. *ACM Comput. Surv.* 45, 2 (2013), 24:1–24:40. https://doi.org/10.1145/2431211.2431223