Towards a Complexity Theory for Local Distributed Computing*

Pierre Fraigniaud† Amos Korman† David Peleg‡

Abstract

A central theme in distributed network algorithms concerns understanding and coping with the issue of *locality*. Yet despite considerable progress, research efforts in this direction have not yet resulted in a solid basis in the form of a fundamental computational complexity theory for locality. Inspired by sequential complexity theory, we focus on a complexity theory for *distributed decision problems*. In the context of locality, solving a decision problem requires the processors to independently inspect their local neighborhoods and then collectively decide whether a given global input instance belongs to some specified language.

We consider the standard $\mathcal{LOCAL}$ model of computation and define $\mathrm{LD}(t)$ (for *local decision*) as the class of decision problems that can be solved in t communication rounds. We first study the intriguing question of whether randomization helps in local distributed computing, and to what extent. Specifically, we define the corresponding randomized class $\mathrm{BPLD}(t, p, q)$, containing all languages for which there exists a randomized algorithm that runs in t rounds, accepts correct instances with probability at least p, and rejects incorrect ones with probability at least q. We show that $p^2+q = 1$ is a threshold for the containment of $\mathrm{LD}(t)$ in $\mathrm{BPLD}(t, p, q)$. More precisely, we show that there exists a language that does not belong to $\mathrm{LD}(t)$ for any $t = o(n)$ but does belong to $\mathrm{BPLD}(0, p, q)$ for any $p, q \in (0, 1]$ such that $p^2 + q \leq 1$. On the other hand, we show that, restricted to hereditary languages, $\mathrm{BPLD}(t, p, q) = \mathrm{LD}(O(t))$, for any function t, and any $p, q \in (0, 1]$ such that $p^2 + q > 1$.

In addition, we investigate the impact of nondeterminism on local decision, and establish several structural results inspired by classical computational complexity theory. Specifically, we show that nondeterminism does help, but that this help is limited, as there exist languages that cannot be decided locally nondeterministically. Perhaps surprisingly, it turns out that it is the combination of randomization with nondeterminism that enables to decide *all* languages *in constant time*. Finally, we introduce the notion of local reduction, and establish a couple of completeness results.

Keywords: Local distributed algorithms, local decision, nondeterminism, randomized algorithms, oracles.

*A preliminary version of this paper has appeared in Proc. 52nd IEEE Symp. on Foundations of Computer Science (FOCS), Palm Springs, California, October 23-25, 2011 (see [24]).

†CNRS and University Paris Diderot, France. E-mail: {pierre.fraigniaud,amos.korman}@liafa.jussieu.fr. Supported by the ANR projects DISPLEXITY and PROSE, and by the INRIA project GANG.

‡The Weizmann Institute of Science, Rehovot, Israel. E-mail: david.peleg@weizmann.ac.il.

1 Introduction

1.1 Motivation and Objective

Distributed computing concerns a collection of processors that collaborate in order to achieve some global task. From a certain perspective, two main disciplines have evolved in the theory of distributed computing. One discipline deals with *timing* issues, namely, uncertainties due to asynchrony (the fact that processors run at their own speed, and possibly crash), and the other concerns *topology* issues, namely, uncertainties due to locality constraints (the lack of knowledge about far away processors)[1]. Studies carried out by the distributed computing community within these two disciplines were to a large extent problem-driven. Indeed, several major problems considered in the literature concern coping with one of the two uncertainties. For instance, in the *asynchrony discipline*, Fischer, Lynch and Paterson [17] proved that consensus cannot be achieved in the asynchronous model, even in the presence of a single fault, and in the *locality discipline*, Linial [42] proved that $(\Delta + 1)$-coloring cannot be achieved locally (i.e., in a constant number of communication rounds) in networks of maximum degree Δ, even in the ring network.

One of the significant achievements of the asynchrony discipline was its success in establishing several theories in the flavor of computational complexity theory. Some central examples of such theories are failure detectors [9, 10] and the wait-free hierarchy (including Herlihy's hierarchy) [30]. Yet despite considerable progress, we are still far from the development of a unified complexity theory for the asynchrony discipline. The situation is even worse for the locality discipline, which still suffers from the absence of any basis in the form of a fundamental computational complexity theory.[2]

Inspired by sequential complexity theory, we focus on *decision problems* [11, 25, 29, 32, 35], in which one is aiming at deciding whether a given global input instance belongs to some specified language. In the context of local computing, each processor must produce a boolean output, and the decision is defined by the conjunction of the processors' outputs, i.e., if the instance belongs to the language, then all processors must output "yes", and otherwise, at least one processor must output "no". Decision problems provide a framework for developing a complexity theory for local distributed computing. Indeed, as shown later, one can define local reductions in the framework of decision problems, thus enabling the introduction of complexity classes, and defining notions of completeness. In addition, decision problems have several other appealing features, including the following.

- First, decision problems provide a natural framework for tackling fault-tolerance: the processors have to collectively check whether the network is fault-free, and a node detecting a fault raises an alarm. In fact, this connection between decision problems and fault-tolerance is one of the pillars enabling the design of efficient *self-stabilizing* protocols (see, e.g., [34]) where: (1) the presence of some inconsistency in the system should lead some node(s) to

[1] Note that this dichotomy is not related to the relationships between shared-memory and message-passing. In fact, these two latter settings have been shown to be essentially the same computationally [4].

[2] Obviously, defining some common cost measures (e.g., time, message, memory, etc.) enables us to compare problems in terms of their relative costs. Still, from a computational complexity point of view, it is not clear how to relate the difficulty of problems in the locality discipline. Specifically, if two problems have different kinds of outputs, it is not clear how to reduce one to the other, even if they cost the same.

automatically launch a recovery procedure, but (2) the absence of inconsistencies should not allow the recovery procedure to run (so that to avoid overloading the system with useless computations).

- Decision problems are also connected to (fault-free) *construction* problems. In the former, one aims at checking whether some instance satisfy some property, while in the latter, one aims at constructing an instance satisfying that property. Similarly to the sequential setting, constructing does not necessarily reduce to deciding also in the distributed setting. While the connection between construction and decision appears to be looser in the distributed setting compared to the sequential setting, one can still identify cases in which decision helps construction in the distributed setting. This is for instance the case with various kinds of randomized iterative distributed algorithms, in the spirit of Luby's algorithm for MIS or coloring [45, 49]. The design of such an algorithm relies on the ability of each node to guess a solution (e.g., a color), and to decide *locally* whether this choice is valid. Proving that, for a given problem, the validity of a solution cannot be checked locally implies that this problem cannot be solved by a randomized algorithm that iteratively guesses solutions. In addition, some variation of decision algorithms (referred to as pruning algorithms in [36]), have been shown to be useful for generalizing construction algorithms, by removing dependancies on global knowledge assumptions [36].
- Last but not least, distributed decision is somewhat related to *property-testing* [27]. The objective of the latter is to (sequentially) decide whether a given instance of size n satisfies some property, or whether it is "far from" satisfying this property, by inspecting $o(n)$ bits of that instance. Several graph properties, including cycle-freeness [28], can be tested efficiently based on the ability to query a few random nodes, and inspect their local neighborhood. The decision is then taken by applying some centralized algorithm over the results of these queries. So property-testing and distributed decision both rely on the ability to take decisions based only on a restricted amount of information.

Returning to the setting of this paper, our purpose is to investigate the nature of local distributed decision problems.

1.2 Framework

We consider the $\mathcal{LOCAL}$ model [49], which is a standard distributed computing model capturing the essence of locality. In this model, processors are woken up simultaneously, and computation proceeds in fault-free synchronous rounds during which every processor exchanges messages of unlimited size with its neighbors, and performs arbitrary computations on its data. Informally, for a function t from the inputs to the integers, let us define $\mathrm{LD}(t)$ (for *local decision*) as the class of decision problems that can be solved in t communication rounds in the $\mathcal{LOCAL}$ model, and $\mathrm{LD}(O(t)) = \bigcup_{c>0} \mathrm{LD}(ct)$. Of special interest is the case where t is constant, namely, the class $\mathrm{LD}(O(1))$. Nevertheless, in general, we consider any function of the input, i.e., of the graph and the individual input of each node. Note that in the $\mathcal{LOCAL}$ model, every decidable problem can be solved in a number of rounds equal to the diameter of the input graph.

Some decision problems fall trivially in $\mathrm{LD}(O(1))$ (e.g., "is the given coloring legal?", "do the selected nodes form a maximal independent set (MIS)?", etc.), while some others can easily be

shown to be outside $\mathrm{LD}(t)$ for any $t = o(n)$ (e.g., "is the network planar?", "is there a unique leader?", etc). In contrast to the above examples, there are some languages whose membership in $\mathrm{LD}(t)$ is unclear, even for $t = O(1)$. To elaborate on this, consider the problem where it is required to decide whether the network belongs to some specified family $\mathcal{F}$ of graphs. If this question can be decided in a constant number of communication rounds, then this means, informally, that the family $\mathcal{F}$ can somehow be characterized by relatively simple (local) conditions. For example, a family $\mathcal{F}$ of graphs that can be characterized as consisting of all graphs having no subgraph from $\mathcal{C}$, where $\mathcal{C}$ is some specified finite set of graphs, is obviously in $\mathrm{LD}(O(1))$. However, the question of whether a family of graphs can be characterized as above is often nontrivial. For example, characterizing cographs[3] as precisely the graphs with no induced P_4 (4-node path), attributed to Seinsche [53], is not easy, and requires nontrivial usage of modular decomposition.

Moreover, a language not in $\mathrm{LD}(t)$ may or may not be decidable locally using randomization. For example, it is not clear that deciding locally whether a graph is planar can be done probabilistically in short time. To study such a question, we define, for every $p, q \in (0, 1]$, the class $\mathrm{BPLD}(t, p, q)$, as the class of all distributed languages that can be decided by a randomized distributed algorithm that runs in t communication rounds, and produces correct answers on legal (respectively, illegal) instances with probability at least p (resp., q). In particular, we study the relationships between the classes LD and BPLD.

1.3 Our Contributions

We first study the impact of randomization on local decision, and the question we focus on is whether randomization helps and to what extent. By definition, $\mathrm{BPLD}(O(t), p, q) \supseteq \mathrm{LD}(O(t))$ for every integer function t of the input. We first observe that

$$p^2 + q \leq 1 \implies \mathrm{BPLD}(O(t), p, q) \neq \mathrm{LD}(O(t))$$

for every $t = o(n)$. Indeed, for such p and q, there exists a language $\mathcal{L}^* \in \mathrm{BPLD}(0, p, q)$, such that $\mathcal{L}^* \notin \mathrm{LD}(t)$, for all $t = o(n)$. It turns out that this choice of p and q is not coincidental. Indeed, we show that, restricted to hereditary languages, if $p^2 + q > 1$, then $\mathrm{BPLD}(O(t), p, q)$ actually collapses into $\mathrm{LD}(O(t))$, for any function t. That is,

$$p^2 + q > 1 \implies \mathrm{BPLD}(O(t), p, q) = \mathrm{LD}(O(t)).$$

These results suggest that $p^2 + q = 1$ may well be a sharp threshold for distinguishing the deterministic class from the randomized one.

In the second part of the paper, we investigate the impact of nondeterminism on local decision, and establish some structural results inspired by classical computational complexity theory. We start by establishing that nondeterminism does help, but that this help is limited, as there exist languages that cannot be decided nondeterministically. Specifically, to show that nondeterminism helps local decision, we prove that the class $\mathrm{NLD}(t)$, the nondeterministic version of $\mathrm{LD}(t)$, strictly contains $\mathrm{LD}(t)$. More precisely, we prove that

$$\mathrm{LD}(O(t)) \neq \mathrm{NLD}(O(t))$$

[3] A cograph is a graph that can be generated from one node by complementation ($E \leftarrow \binom{n}{2} \setminus E$) and disjoint union ($G \leftarrow G_1 \cup G_2$). (For instance, K_2 can be obtained by taking the disjoint union of two K_1s, complemented).

for every function $t = o(n)$. This is established by exhibiting a language in $\mathrm{NLD}(O(1))$ that is not in $\mathrm{LD}(t)$ for every $t = o(n)$. On the other hand, we also show that $\mathrm{NLD}(t)$ does not capture all (decidable) languages, for $t = o(n)$. Indeed we prove that there exists a language not in $\mathrm{NLD}(t)$ for every $t = o(n)$. Specifically, this language is $\texttt{GraphSize} = \{(G, k) \text{ s.t. } |V(G)| = k\}$, which requires the nodes to decide whether the input graph has k nodes, where k is given as input to every node. Hence

$$\mathrm{NLD}(o(n)) \neq \mathrm{All},$$

where All is the set of (sequentially decidable) distributed languages.

Perhaps surprisingly, it turns out that it is the combination of randomization with nondeterminism that enables to decide *all* languages in *constant* time. To establish this result, we define the randomized version BPNLD of NLD, in the same way BPLD is defined from LD. Let $\mathrm{BPNLD}(t) = \bigcup_{p^2+q\leq 1} \mathrm{BPNLD}(t, p, q)$. We prove that $\mathrm{BPNLD}(O(1))$ contains all decidable distributed languages, i.e.,

$$\mathrm{BPNLD}(O(1)) = \mathrm{All}.$$

Alternatively, by considering oracles providing global information to the nodes, we also show that if each node can access an oracle that returns the number of nodes in the input graph, then all languages can be decided in constant time. To establish this result, we define the oracle class $\mathrm{NLD}^{\texttt{GraphSize}}(O(1))$, which is the class of decision problems that can be solved nondeterministically in a constant number of communication rounds assuming that each node has access to the oracle `GraphSize`. We prove that

$$\mathrm{NLD}^{\texttt{GraphSize}}(O(1)) = \mathrm{All}.$$

To sum up, for every $t = o(n)$, we have

$$\mathrm{LD}(O(t)) \subset \mathrm{NLD}(O(t)) \subset \mathrm{BPNLD}(O(1)) = \mathrm{NLD}^{\texttt{GraphSize}}(O(1)) = \mathrm{All}.$$

Finally, we introduce the notion of *local reduction*, and establish some completeness results. We show that there exists a problem, called `Cover`, which is, in a sense, the most difficult decision problem. That is, we show that `Cover` is $\mathrm{BPNLD}(O(1))$-complete. Interestingly, a small relaxation of `Cover`, called `Containment`, turns out to be $\mathrm{NLD}(O(1))$-complete.

1.4 Related Work

Locality issues have been thoroughly studied in the literature, via the analysis of various construction problems, including coloring and maximal independent set (MIS) [2, 7, 37, 40, 42, 45, 48], minimum-weight spanning tree (MST) [15, 39, 50], matching [31, 43, 44, 54], dominating set [38, 41], spanners [12, 16, 51], etc. For some problems (e.g., coloring [7, 37, 48]), there are still large gaps between the best known results on specific families of graphs (e.g., bounded degree graphs) and on arbitrary graphs.

The question of what can be computed in a constant number of communication rounds was posed in the seminal work of Naor and Stockmeyer [47]. In particular, that paper considers a subclass of $\mathrm{LD}(O(1))$, called LCL, which is essentially $\mathrm{LD}(O(1))$ restricted to languages involving graphs of constant maximum degree and processor inputs taken from a set of constant size, and studies the question of how to compute in $O(1)$ rounds the constructive versions of decision problems in LCL. The paper provides several general results. In particular, it shows that if there exists

a randomized algorithm that constructs a solution for a problem in LCL in $O(1)$ rounds, then there is also a deterministic algorithm constructing a solution for that problem in $O(1)$ rounds. Unfortunately, the proof of this result relies heavily on the definition of LCL. Indeed, the constant bound constraints on the degrees and input sizes enable a proof based on a clever use of Ramsey theory. It is not clear whether it is possible to extend this result to all languages in LD($O(1)$).

The question of whether randomization helps in enabling local solutions for construction problems has been the focus of numerous studies. To date, there exists evidence that, for some problems at least, randomization does not help. For instance, [46] proves this for 3-coloring on rings. In fact, for low degree graphs, the gaps between the efficiencies of the best known randomized and deterministic algorithms for problems like MIS, $(\Delta + 1)$-coloring, and maximal matching are very small. On the other hand, for graphs of arbitrarily large degrees, there seem to be indications that randomization does help, at least in some cases. For instance, $(\Delta + 1)$-coloring can be randomly computed in expected $O(\log n)$ communication rounds on n-node graphs [2, 45], whereas the best known deterministic algorithm for this problem performs in $2^{O(\sqrt{\log n})}$ rounds [48]. $(\Delta + 1)$-coloring algorithms whose performance is expressed also in terms of the maximum degree Δ illustrate this phenomenon as well. Specifically, it is known that $(\Delta + 1)$-coloring can be randomly computed in expected $O(\log \Delta + \sqrt{\log n})$ communication rounds (see [52]) recently improved to $O(\log \Delta + e^{O(\sqrt{\log \log n})})$ rounds in [8], whereas the best known deterministic algorithm performs in $O(\Delta + \log^* n)$ rounds [7, 37].

Recently, several results were established concerning decision problems in distributed computing. For example, [11] and [32] study specific decision problems in the $\mathcal{CONGEST}$ model. (In contrast to the $\mathcal{LOCAL}$ model, this model assumes that the message size is bounded by $O(\log n)$ bits, hence dealing with congestion is the main issue.) Specifically, tight bounds are established in [32] for the time and message complexities of the problem of deciding whether a given subgraph is an MST of the network, and time lower bounds for many other subgraph-decision problems (e.g., spanning tree, connectivity) are established in [11]. Decision problems have recently received attention in the asynchrony discipline too, in the framework of wait-free computing [25, 26].

The theory of *proof labeling schemes* [29, 33, 34, 35] was designed to tackle the issue of locally verifying (with the aid of a "proof", i.e., a certificate, at each node) solutions to problems that cannot be decided locally (e.g.,"is the given subgraph a spanning tree of the network?", or, "is it an MST of the network?"). Investigations in this framework mostly focus on the minimum size of the certificate necessary so that verification can be performed in a single round [29, 33, 35], or in t rounds [34]. Hence, the model of proof labeling schemes has some resemblance to our definition of the class NLD. The notion of proof labeling schemes also has interesting similarities with the notions of local detection [1], local checking [5], or silent stabilization [14], which were introduced in the context of self-stabilization [13]. The notion of NLD seems to be also related to the theory of *lifts* [3].

The use of oracles that provide information to nodes was studied intensively in the context of distributed construction tasks. In particular, it was studied in the framework of *local computation with advice*. In this framework, MST construction was studied in [22], 3-coloring of cycles in [19], and broadcast and wake up in [18]. Finally, in [36] it is shown that, in the context of local computation, access to the oracle providing the number of nodes is not required for solving efficiently several central problems (e.g., $O(\Delta)$-coloring, MIS, etc.), while previous algorithms in the literature explicitly or implicitly assumed the use of this oracle.

2 Decision Problems and Complexity Classes

2.1 Model of Computation

Let us first recall some basic notions in distributed computing. We consider the $\mathcal{LOCAL}$ model [49], which is a standard model capturing the essence of locality. In this model, processors are assumed to be nodes of a network G, provided with arbitrary distinct identities. All processors are woken up simultaneously. Initially, a processor $v \in V(G)$ is aware only of it own identity $\mathrm{Id}(v)$ and, possibly, of some local input $\mathbf{x}(v)$. Computation proceeds in fault-free synchronous rounds. In each round of an algorithm $\mathcal{A}$, every processor v exchanges messages of unrestricted size with its neighbors in G, and performs computations on its data. The model does not impose any restriction on the amount of individual computation performed at each node[4]. In other words, in each round r during the execution of a distributed algorithm $\mathcal{A}$, every processor v: (1) receives messages from its neighbors, (2) performs individual computations, and (3) sends messages to its neighbors. After a number of rounds (that may depend on the network G and may vary among the processors, simply because nodes have different identities, potentially different inputs, and are typically located at non-isomorphic positions in the network), every processor v terminates and generates its output.

Consider an algorithm $\mathcal{A}$ running in a network G with input $\mathbf{x}$ and identity assignment Id. (An identity assignment for a graph G is an assignment of distinct integers to the nodes of G.) The output of processor v in this scenario is denoted by $\mathrm{out}_{\mathcal{A}}(G, \mathbf{x}, \mathrm{Id}, v)$ (or simply $\mathrm{out}(v)$ when the parameters are clear from the context). The *running time* of the algorithm at a node v, denoted by $T_{\mathcal{A},v}$, is the number of communication rounds until v produces its output. Note that $T_{\mathcal{A},v}$ may depend on the structure of G, the global input $\mathbf{x}$, and the identity assignment Id. The *algorithm's running time*, denoted by $T_{\mathcal{A}}$, is the number of rounds until all processors terminate. Again, $T_{\mathcal{A}}$ may depend on $(G, \mathbf{x}, \mathrm{Id})$, and

$$T_{\mathcal{A}}(G, \mathbf{x}, \mathrm{Id}) \;=\; \max_{v \in V(G)} \{T_{\mathcal{A},v}(G, \mathbf{x}, \mathrm{Id})\}.$$

Let $\mathcal{C}$ be the collection of all triples $(G, \mathbf{x}, \mathrm{Id})$. A *runtime function* is a function

$$t : \mathcal{C} \to \mathbb{Z}^+ \ .$$

Let t be a runtime function. We say that an algorithm $\mathcal{A}$ has running time at most t, if $T_{\mathcal{A}}(G, \mathbf{x}, \mathrm{Id}) \leq t(G, \mathbf{x}, \mathrm{Id})$, for every $(G, \mathbf{x}, \mathrm{Id})$. We shall give special attention to the case where t represents a constant function. Note that, in general, for a given $(G, \mathbf{x}, \mathrm{Id})$, the nodes may not be aware of $t(G, \mathbf{x}, \mathrm{Id})$ because it requires the global knowledge of $(G, \mathbf{x}, \mathrm{Id})$. On the other hand, if $\gamma = t(G, \mathbf{x}, \mathrm{Id})$ happens to be known to every node, then without loss of generality one can assume that an algorithm running in time at most γ operates at each node v in two stages:

1. The node v collects all information available in its γ-neighborhood (i.e., the ball $B_G(v, \gamma)$ of radius γ around v in G), including input values, identities, and network structure;
2. The node v computes the output locally based on the information collected in Stage 1.

[4]We would like to point out that imposing restrictions on the time or the memory used by a node for local computation, may lead to interesting connections between the theory of locality and classical computational complexity theory. (See discussion in Section 5.1).

In the case of a randomized algorithm, both the running time of a node and the algorithm's running time are random variables, whose values depend on the results of mutually independent random coin flips performed individually at each node.

2.2 Local Decision (LD)

We now refine some of the above concepts. Obviously, a distributed algorithm that runs on a graph G operates separately on each connected component of G, and nodes of a component G' of G cannot distinguish the underlying graph G from G'. For this reason, we consider connected graphs only.

Definition 2.1 *An* instance *is a pair* $(G, \mathbf{x})$ *where* G *is a connected graph, and* $\mathbf{x}$ *is a mapping from* $V(G)$ *to* $\{0,1\}^*$*, i.e., every node* $v \in V(G)$ *is assigned as its* local input *a binary string* $\mathbf{x}(v) \in \{0,1\}^*$*. (In some problems, the local input of every node is empty, i.e.,* $\mathbf{x}(v) = \epsilon$ *for every* $v \in V(G)$*, where* ϵ *denotes the empty binary string.)*

Since an undecidable collection of instances remains undecidable in the distributed setting too, we consider only decidable collections of instances. Formally, we define the following.

Definition 2.2 *A* distributed language *is a decidable collection* $\mathcal{L}$ *of instances.*

In general, there are several possible ways of representing an instance of a distributed language corresponding to a standard distributed computing problem. A naturally arising type of decision languages involves getting as input an $\mathbf{x}$ claimed to be the output of some common distributed computing problem Π on one of its instances, and having to decide whether it is indeed a legal output for Π. Some examples for languages of this type are given below.

- `Consensus` $= \{(G, (\mathbf{x}_1, \mathbf{x}_2))$ s.t. $\exists u \in V(G), \forall v \in V(G), \mathbf{x}_2(v) = \mathbf{x}_1(u)\}$. This language consists of all instances in which all nodes agree on the value proposed by one of them.
- k-`Coloring` $= \{(G, \mathbf{x})$ s.t. $\forall v \in V(G), \mathbf{x}(v) \in \{1, 2, \cdots, k\}$, and $\forall w \in N(v), \mathbf{x}(v) \neq \mathbf{x}(w)\}$ where $N(v)$ denotes the (open) neighborhood of v, that is, all nodes at distance exactly 1 from v. This language consists of all instances in which $\mathbf{x}$ is a legal coloring of G with k colors.
- `MIS` $= \{(G, \mathbf{x})$ s.t. $S = \{v \in V(G) \mid \mathbf{x}(v) = 1\}$ forms a maximal independent set$\}$.
- `Tree` $= \{(G, \epsilon);$ G is a tree, i.e., it is cycle-free$\}$.
- `Planar` $= \{(G, \epsilon);$ G is a planar graph$\}$.
- `SpanningTree` $= \{(G, (\text{name}, \text{parent}))$ s.t. T is a spanning tree of $G\}$, where

  $$T = \{(v, w) \text{ s.t. } (v, w) \in E(G), \text{ and } \forall v \in V(G), \text{parent}(v) = \text{name}(w)\}.$$

  In other words, `SpanningTree` consists of all instances such that the set T of edges between every node v and its neighbor w satisfying $\text{name}(w) = \text{parent}(v)$ forms a spanning tree of G. (The language `MST`, for *minimum spanning tree*, can be defined similarly).

Let $\mathcal{L}$ be a distributed language. We say that a distributed algorithm $\mathcal{A}$ *decides* $\mathcal{L}$ if and only if, for every instance $(G, \mathbf{x})$, every node of G eventually terminates and outputs "yes" or "no", satisfying the following decision rules:

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

We are now ready to define one of our main subjects of interest, the class $\text{LD}(t)$, for *local decision*. (Recall that $\mathcal{C}$ denotes the collection of all triples $(G, \mathbf{x}, \text{Id})$).

Definition 2.3 *Let $t : \mathcal{C} \mapsto \mathbb{Z}^+$ be a runtime function.* $\text{LD}(t)$ *is the class of all distributed languages that can be decided by a distributed algorithm with running time at most t.*

For instance, observe that k-`Coloring` $\in \text{LD}(1)$ for every constant k, and similarly `MIS` $\in \text{LD}(1)$. On the other hand, it is not hard to see that languages such as `Consensus`, `Tree`, `Planar` and `SpanningTree` are not in $\text{LD}(t)$, for any $t = o(n)$.

For every runtime function t, define $\text{LD}(O(t)) = \bigcup_{c>0} \text{LD}(c \cdot t)$. Hence, for a distributed language $\mathcal{L}$ and a function t, $\mathcal{L} \in \text{LD}(O(t))$ if and only if there exists a constant c such that $\mathcal{L} \in \text{LD}(c \cdot t)$. Let $\text{LD} = \text{LD}(O(1))$.

2.3 Nondeterministic Local Decision (NLD)

A distributed *verification* algorithm is a distributed algorithm $\mathcal{A}$ that gets as input, in addition to an instance $(G, \mathbf{x})$, also a global *certificate vector* $\mathbf{y}$, i.e., every node v of a graph G gets as its input two binary strings, an input $\mathbf{x}(v) \in \{0,1\}^*$ and a certificate $\mathbf{y}(v) \in \{0,1\}^*$. A verification algorithm $\mathcal{A}$ verifies $\mathcal{L}$ if and only if, for every instance $(G, \mathbf{x})$, the following two conditions hold:

- If $(G, \mathbf{x}) \in \mathcal{L}$, then there exists a certificate $\mathbf{y}$ such that, for every identity assignment Id, the algorithm $\mathcal{A}$ returns $\text{out}_{\mathcal{A}}(G, (\mathbf{x}, \mathbf{y}), \text{Id}, v) =$"yes" for all $v \in V(G)$;
- If $(G, \mathbf{x}) \notin \mathcal{L}$, then for every certificate $\mathbf{y}$, and for every identity assignment Id, the algorithm $\mathcal{A}$ returns $\text{out}_{\mathcal{A}}(G, (\mathbf{x}, \mathbf{y}), \text{Id}, v) =$"no" for at least one node $v \in V(G)$.

One motivation for studying nondeterminism in the above sense comes from settings in which one must repeatedly perform many local verifications. In such cases, one can afford to have a relatively "wasteful" preliminary step in which a certificate is computed for each node. Using these certificates, local verifications can then be performed very fast (see [33, 35] for more details regarding such applications). Indeed, the definition of a verification algorithm bears close similarities with the notion of *proof labeling schemes* discussed therein. The two concepts differ as, in a proof labeling scheme, the construction of a "good" certificate $\mathbf{y}$ for an instance $(G, \mathbf{x}) \in \mathcal{L}$ may depend also on

the given identity assignment. Instead, in the concept of nondeterminism introduced above, the "good" certificate $\mathbf{y}$ for an instance $(G, \mathbf{x}) \in \mathcal{L}$ should be good for any identity assignment.

Since the question of whether an instance $(G, \mathbf{x})$ belongs to a language $\mathcal{L}$ is independent from the particular identity assignment , we prefer to let the "good" certificate $\mathbf{y}$ depend only on the instance. In other words, as defined above, a verification algorithm operating on an instance $(G, \mathbf{x}) \in \mathcal{L}$ and a "good" certificate $\mathbf{y}$ must lead all nodes to say "yes" regardless of the identity assignment . We now define the class NLD(t), for *nondeterministic local decision*. (Our terminology is chosen by direct analogy to the class NP in sequential computational complexity.)

Definition 2.4 *Let $t : \mathcal{C} \mapsto \mathbb{Z}^+$ be a runtime function.* NLD(t) *is the class of all distributed languages that can be verified by a distributed verification algorithm with running time at most t. Let* NLD = NLD($O(1)$).

2.4 Bounded-error Probabilistic Local Decision (BPLD)

A *randomized* distributed algorithm is a distributed algorithm $\mathcal{A}$ that enables every node v, at any point during the execution, to toss a certain number of random bits. More specifically, in this paper, randomized computation is represented by considering Monte Carlo algorithms.

Recall that, in sequential computing, a Monte Carlo algorithm is a randomized algorithm whose running time is deterministic (i.e., independent of the random choices), but whose output may be incorrect with a certain probability. We extend this concept to the distributed setting, by focusing on distributed algorithms that use randomization but whose running times are deterministic. Actually, we are more liberal, and allow the running time to depend on the values of the random bits flipped by the nodes, provided that they obey the simple restriction that the *maximum* execution time T_v of node v, over all the values of the random bits flipped by all nodes, is bounded deterministically (i.e., it depends only of the actual instance $(G, \mathbf{x})$ and identity assignment Id).

For fixed $p, q \in (0, 1]$, we say that a randomized distributed algorithm $\mathcal{A}$ is a (p, q)-*decider* for $\mathcal{L}$, or, that it decides $\mathcal{L}$ with "yes" success probability p, and "no" success probability q, if and only if for every instance $(G, \mathbf{x})$, every node of G eventually terminates and outputs "yes" or "no", and the following properties are satisfied:

- If $(G, \mathbf{x}) \in \mathcal{L}$ then, for every identity assignment Id,

$$\mathtt{Pr}[\mathrm{out}_{\mathcal{A}}(G, \mathbf{x}, \mathrm{Id}, v) = \text{"yes" for every node } v \in V(G)] \geq p.$$

- If $(G, \mathbf{x}) \notin \mathcal{L}$ then, for every identity assignment Id,

$$\mathtt{Pr}[\mathrm{out}_{\mathcal{A}}(G, \mathbf{x}, \mathrm{Id}, v) = \text{"no" for at least one node } v \in V(G)] \geq q.$$

where the probabilities in the above definition are taken over all possible coin tosses performed by the nodes.

Note that the running time of a (p, q)-decider $\mathcal{A}$ at a node v depends on the triple $(G, \mathbf{x}, \mathrm{Id})$ and on the results of the coin tosses. In the context of a randomized algorithm $\mathcal{A}$, $T_{\mathcal{A},v}(G, \mathbf{x}, \mathrm{Id})$ denotes the maximal running time of Algorithm $\mathcal{A}$ at v over all possible coin tosses, for the instance

$(G, \mathbf{x})$, and the identity assignment Id. Then, similarly to the deterministic case, the running time $T_{\mathcal{A}}$ of the (p, q)-decider $\mathcal{A}$ is the maximum running time of $\mathcal{A}$ over the nodes. Again, by definition of the distributed Monte-Carlo algorithm, both $T_{\mathcal{A},v}$ and $T_{\mathcal{A}}$ are deterministic.

We define the class BPLD(t, p, q), for *bounded-error probabilistic local decision*, as follows.

Definition 2.5 *For $p, q \in (0, 1]$ and a runtime function $t : \mathcal{C} \mapsto \mathbb{Z}^+$,* BPLD$(t, p, q)$ *is the class of all distributed languages that have a randomized distributed (p, q)-decider with running time at most t (i.e., that can be decided in time at most t by a randomized distributed algorithm with "yes" success probability p and "no" success probability q).*

3 A Sharp Threshold for Randomization

The objective of this section is to address the question of whether randomization helps local distributed computing, and if so - to what extent. Recall that [47] investigates the question of whether randomization helps for constructing, in constant time, a solution for a problem in LCL$\subset$ LD$(O(1))$. We stress that the technique used in [47] for tackling this question relies heavily on the definition of LCL, and specifically, on the fact that only graphs of constant degree and of constant input size are considered. Hence, it is not clear whether the technique of [47] can be useful for our purposes, as we impose no such assumptions on the degrees or input sizes. We also note that, although it seems at first glance that the Lovász local lemma might have been helpful here, we could not effectively apply it in our proof. Instead, we use a completely different approach. Let us start with a simple observation. Consider the following language.

Definition 3.1 `At-Most-One-Selected` (`AMOS`) $= \{(G, \mathbf{x})$ *s.t.* $\mathbf{x} : V(G) \mapsto \{0, 1\}$ *and* $\| \mathbf{x} \|_1 \leq 1\}$. *Namely,* `AMOS` *consists of all instances containing at most one "selected node" (i.e., with input 1), with all other nodes "unselected" (i.e., having input 0).*

By considering the n-node path, one can easily check that `AMOS` $\notin$ LD(t), for any $t = o(n)$. (This holds even if one assumes that the selected nodes can only be the two extremities of the path.) Yet, we claim that `AMOS` $\in$ BPLD$(0, p, q)$ for every p and q such that $p^2 + q \leq 1$. Indeed, for such p and q, we can design the following simple randomized (p, q)-decider algorithm, whose running time is zero:

- Every unselected node outputs "yes" with probability 1;
- every selected node outputs "yes" with probability p.

Clearly, if the instance has at most one selected node, then all nodes say "yes" with probability at least p. On the other hand, if there are at least $k \geq 2$ selected nodes, that is, if the instance is not in the language, then the probability that some node says "no" is at least $1 - p^k \geq 1 - p^2 \geq q$. Thus, we obtain the following:

Theorem 3.2 *For p and q such that $p^2 + q \leq 1$, there exists a language $\mathcal{L} \in$* BPLD$(0, p, q)$*, such that $\mathcal{L} \notin$* LD(t) *for any $t = o(n)$.*

In the remainder of the section, we show that, at least for a large class of languages, called *hereditary* languages, the bound $p^2 + q = 1$ is actually a sharp threshold.

For a graph G and a subset $U \subseteq V(G)$ of the nodes of G, let $G[U]$ denote the subgraph of G induced by the nodes in U. Given an instance $(G, \mathbf{x})$, let $\mathbf{x}[U]$ denote the input $\mathbf{x}$ restricted to the nodes in U. A *prefix* of an instance $(G, \mathbf{x})$ is an instance $(G[U], \mathbf{x}[U])$, where $U \subseteq V(G)$. (Note that, since an instance always deals with a connected graph, if $(G[U], \mathbf{x}[U])$ is a prefix of $(G, \mathbf{x})$, then $G[U]$ is connected.)

In what follows, when the instance $(G, \mathbf{x})$ is fixed, we sometimes refer to the prefix $(G[U], \mathbf{x}[U])$ informally as "the prefix U". Moreover, we may informally say, e.g., "$U \in \mathcal{L}$," to mean $(G[U], \mathbf{x}[U]) \in \mathcal{L}$.

Definition 3.3 *A language $\mathcal{L}$ is* hereditary *if every prefix of every instance $(G, \mathbf{x}) \in \mathcal{L}$ is also in $\mathcal{L}$.*

`Coloring` and `AMOS` are clearly hereditary languages. As another example of a hereditary language, consider a family $\mathcal{G}$ of hereditary graphs, i.e., a family closed under vertex deletion. Then the language $\{(G, \epsilon) \mid G \in \mathcal{G}\}$ is hereditary. Examples of hereditary graph families are planar graphs, interval graphs, forests, chordal graphs, cographs, perfect graphs, etc. Theorem 3.4 below asserts that, for hereditary languages, randomization does not help (up to a multiplicative constant) if one imposes the requirement that the "no" and "yes" success probabilities of the (p, q)-decider satisfy $p^2 + q > 1$.

Theorem 3.4 *Let $\mathcal{L}$ be a hereditary language and let $t : \mathcal{C} \mapsto \mathbb{Z}^+$ be a runtime function. If $\mathcal{L} \in \mathrm{BPLD}(t, p, q)$ for constants $p, q \in (0, 1]$ such that $p^2 + q > 1$, then $\mathcal{L} \in \mathrm{LD}(O(t))$.*

Informally, the proof of the theorem consists of proving the correctness of a deterministic algorithm that decides $\mathcal{L}$ in time $O(t)$. Given an instance $(G, \mathbf{x})$, the proposed deterministic algorithm $\mathcal{D}$ operating at a node v collects the topological information and inputs from the ball $B_G(v, \gamma)$ of radius $\gamma = O(t)$ around v, and outputs "yes" at v if and only if $B_G(v, \gamma) \in \mathcal{L}$. (Some care is needed here, since v might not know the time bound t, and therefore, collecting information from a ball of radius $\gamma = O(t)$ might potentially be problematic; we ignore this technicality in this informal sketch.) In proving the correctness of the deterministic algorithm $\mathcal{D}$, one direction is immediate: the fact that the given language $\mathcal{L}$ is hereditary implies that if we start with a *legal* instance, that is, $(G, \mathbf{x}) \in \mathcal{L}$, then every prefix of $(G, \mathbf{x})$ also belongs to $\mathcal{L}$, including in particular the balls $B_G(v, \gamma)$ for every v, hence in algorithm $\mathcal{D}$, each node outputs "yes". The difficult task is to show that one can choose the constant factor hidden in the $O(t)$ notation, such that for any initial illegal instance, there exists a ball $B_G(v, \gamma)$ of radius $\gamma = O(t)$ around *some* node v, such that the prefix of $(G, \mathbf{x})$ induced by this ball is not in $\mathcal{L}$. Hence, for any instance $(G, \mathbf{x}) \notin \mathcal{L}$, there would exist a node v, such that under algorithm $\mathcal{D}$, node v outputs "no". Towards achieving this task, we first establish Lemma 3.5, which informally states that the union of two legal instances is also legal provided their intersection is "sufficiently large". This crucial structural lemma uses the fact that $\mathcal{L} \in \mathrm{BPLD}(t, p, q)$ for constants $p, q \in (0, 1]$ such that $p^2 + q > 1$. Specifically, the question of how large the intersection needs to be depends on the extent to which $p^2 + q - 1$ is bounded away from zero. It is interesting to note that Lemma 3.5 does not use the fact that the given language $\mathcal{L}$ is hereditary. To complete the proof, we consider an illegal instance $(G, \mathbf{x}) \notin \mathcal{L}$ and assume,

towards contradiction, that under $\mathcal{D}$ every node outputs "yes" (or in other words, $B_G(v,\gamma) \in \mathcal{L}$ for every $v \in V(G)$). We then consider the largest prefix U of $(G,\mathbf{x})$ that is legal. This U is not empty, since by assumption $B_G(v,\gamma) \in \mathcal{L}$ for every node v. On the other hand, U is not the whole $V(G)$, since we assume $(G,\mathbf{x}) \notin \mathcal{L}$. Intuitively, if we could choose the constant factor hidden in the $O(t)$ notation to be large enough so that the intersection of U and the ball $B_G(v)$ (for a node $v \in U$) would be "sufficiently large", then we could have employed Lemma 3.5 and deduce that $U \cup B_G(v,\gamma) \in \mathcal{L}$. Instead, to deduce that $U \cup B_G(v,\gamma) \in \mathcal{L}$, we use a more refined argument that requires repeated use of Lemma 3.5 on different connected prefixes of $U \cup B_G(v,\gamma)$. Finally, to obtain the contradiction, we also make sure that v is chosen close enough to the border of U, so that $U \cup B_G(v,\gamma)$ strictly contains U, thus contradicting the maximality of U.

We now turn to the formal proof of Theorem 3.4.

Proof. Let us start with some definitions. Let $\mathcal{L}$ be a language in BPLD(t,p,q) where $p,q \in (0,1]$, $p^2+q > 1$, and let $t : \mathcal{C} \mapsto \mathbb{Z}^+$ be a runtime function. Let $\mathcal{A}$ be a randomized (p,q)-decider algorithm deciding $\mathcal{L}$, with "yes" success probability p and "no" success probability q, whose running time is at most t, namely, $T(G,\mathbf{x},\mathrm{Id}) \leq t(G,\mathbf{x},\mathrm{Id})$ for every instance $(G,\mathbf{x})$ with identity assignment Id.

For $v \in V(G)$, recall that $T_v = T_v(G,\mathbf{x},\mathrm{Id})$ denotes the maximum, over all possible coin tosses, of the running time of $\mathcal{A}$ on v, and that $T = T(G,\mathbf{x},\mathrm{Id})$ denotes the maximum of T_v over all nodes of G. Note that $T_v \leq T \leq t(G,\mathbf{x},\mathrm{Id})$, but it is not assumed that any of these values is initially known to v. For a given triple $(G,\mathbf{x},\mathrm{Id})$, the *radius* of a node v, denoted $r_v = r_v(G,\mathbf{x},\mathrm{Id})$, is the maximum value of $T_u = T_u(G,\mathbf{x},\mathrm{Id})$ over all nodes u for which v belongs to the ball $B_G(u,T_u)$ of radius T_u around u. (Informally, $B_G(u,T_u)$ stands for the collection of nodes that u can possibly "see" during the execution on $(G,\mathbf{x},\mathrm{Id})$, hence with this terminology, the radius r_v is the maximum running time of a node that can potentially "see" v.) Observe that the radius r_v of a node v satisfies $T_v \leq r_v \leq t(G,\mathbf{x},\mathrm{Id})$. The radius of a collection of nodes S is $r_S = \max_{v\in S}\{r_v\}$. In particular, $r_{V(G)} = T$.

The distance $\mathrm{dist}_G(u,v)$ between two nodes of G is the minimum number of edges in a path connecting u and v in G. The distance between two subsets $U_1, U_2 \subseteq V$ is defined as

$$\mathrm{dist}_G(U_1,U_2) \;=\; \min\{\mathrm{dist}_G(u,v) \mid u \in U_1, v \in U_2\}.$$

Fix a constant δ such that $0 < \delta < p^2+q-1$, and define

$$\lambda = 11 \cdot \lceil \log p / \log(1-\delta) \rceil .$$

A *separating partition* of $(G,\mathbf{x},\mathrm{Id})$ is a triplet (S,U_1,U_2) of pairwise disjoint subsets of nodes such that $S \cup U_1 \cup U_2 = V$, and $\mathrm{dist}_G(U_1,U_2) \geq \lambda \cdot r_S$. See Figure 1(a). (Observe that r_S may depend on the identity assignment and on the input; Therefore, being a separating partition is not a property depending only on G). Given a separating partition (S,U_1,U_2) of $(G,\mathbf{x},\mathrm{Id})$, let $G_k = G[U_k \cup S]$, and let $\mathbf{x}_k$ be the input $\mathbf{x}$ restricted to nodes in G_k, for $k = 1,2$. Note that the following structural result does not use the fact that $\mathcal{L}$ is hereditary.

Lemma 3.5 *For every instance* $(G,\mathbf{x})$ *with identity assignment* Id*, and every separating partition* (S,U_1,U_2) *of* $(G,\mathbf{x},\mathrm{Id})$*, we have*

$$((G_1,\mathbf{x}_1) \in \mathcal{L} \text{ and } (G_2,\mathbf{x}_2) \in \mathcal{L}) \;\Rightarrow\; (G,\mathbf{x}) \in \mathcal{L}.$$

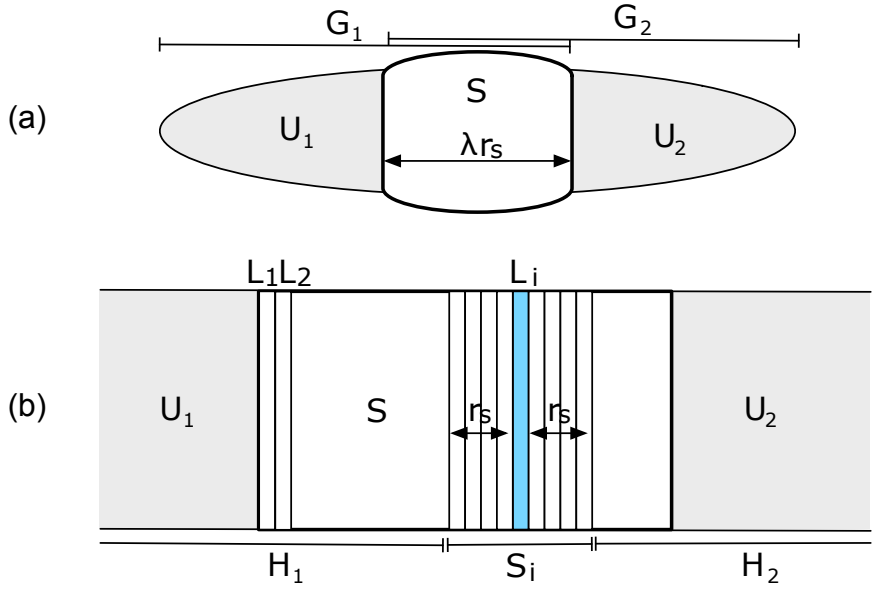

Figure 1: Separating partition

Proof. Let $(G, \mathbf{x})$ be an instance with identity assignment Id. Assume, towards contradiction, that there exists a separating partition (S, U_1, U_2) of $(G, \mathbf{x}, \mathrm{Id})$, such that $(G_1, \mathbf{x}_1) \in \mathcal{L}$ and $(G_2, \mathbf{x}_2) \in \mathcal{L}$, yet $(G, \mathbf{x}) \notin \mathcal{L}$. (Note, by the way, that the fact that $(G_1, \mathbf{x}_1) \in \mathcal{L}$ and $(G_2, \mathbf{x}_2) \in \mathcal{L}$ implies that both G_1 and G_2 are connected; However, for the claim to be true, it is not required that $G[U_1]$, $G[U_2]$ or $G[S]$ be connected.) Given a vertex $u \in S$, define the *level* of u as

$$\ell(u) = \mathrm{dist}_G(U_1, \{u\}).$$

For an integer $i \in [1, \lambda r_S]$, let L_i denote the set of nodes in S of level i (see Figure 1(b)). We now construct strips made out of $2r_S + 1$ consecutive levels (see Figure 1(b) again). Formally, for $i \in (r_S, (\lambda - 1)r_S)$, let

$$S_i = \bigcup_{j=i-r_S}^{i+r_S} L_j.$$

Finally, for a set of integers $I \subseteq (r_S, (\lambda - 1)r_S)$, let

$$S_I = \bigcup_{i \in I} S_i.$$

In what follows, we focus on the range of levels

$$\mathcal{R} = \{2r_S + 1, \ldots, (\lambda - 2)r_S - 1\}.$$

For a set $U \subseteq V(G)$, let $\mathcal{E}(G, \mathbf{x}, \mathrm{Id}, U)$ denote the event that, when running $\mathcal{A}$ on $(G, \mathbf{x})$ with identity assignment Id, all nodes in U output "yes". Define $\mathcal{I}$ as the set of levels i such that the probability that some node of S_i will say "no" is more than δ. Formally,

$$\mathcal{I} \;=\; \{i \in \mathcal{R} \mid \mathtt{Pr}[\mathcal{E}(G, \mathbf{x}, \mathrm{Id}, S_i)] \;<\; 1 - \delta\}.$$

Claim 3.6 *There exists some $i \in \mathcal{R}$ such that $i \notin \mathcal{I}$.*

Proof. Before establishing the claim, we first note that for specifying an execution of $\mathcal{A}$ on $(G, \mathbf{x}, \mathrm{Id})$ completely, it is necessary to specify a collection Γ consisting of n sequences of bits (resulting from random bit choices), one for each node of G, used in that particular execution with the random choices made by the algorithm. Denote the resulting execution, or run, of algorithm $\mathcal{A}$ by $\mathrm{Run}(G, \mathbf{x}, \mathrm{Id}, \Gamma)$.

For proving Claim 3.6, we upper bound the size of $\mathcal{I}$ by $(\lambda - 4)r_S - 2$, which is smaller than $|\mathcal{R}| = (\lambda - 4)r_S - 1$. This is done as follows. Let $\mu = 4r_S + 1$. We first cover the integers in $\mathcal{R}$ by at most μ sets, each of which is *μ-apart*, that is, the distance between every two integers in the same set is at least μ. Specifically, for $s \in [1, \mu]$ and $m(S) = \lceil (\lambda - 8)r_S/\mu \rceil$, we define the arithmetic progression

$$J_s = \{s + 2r_S + j\mu \mid j \in [0, m(S)]\}.$$

Observe that, as desired, $\mathcal{R} \subset \bigcup_{s \in [1,\mu]} J_s$, and J_s is μ-apart for each $s \in [1, \mu]$.

In what follows, fix $s \in [1, \mu]$ and let $J = J_s$. Since $(G_1, \mathbf{x}_1) \in \mathcal{L}$, it follows that

$$\mathtt{Pr}[\mathcal{E}(G_1, \mathbf{x}_1, \mathrm{Id}, S')] \ \geq \ p$$

for every vertex set S' in G_1. Note that $S_{\mathcal{I} \cap J} \subseteq S$, and therefore $S_{\mathcal{I} \cap J}$ is contained in G_1, so

$$\mathtt{Pr}[\mathcal{E}(G_1, \mathbf{x}_1, \mathrm{Id}, S_{\mathcal{I} \cap J})] \ \geq \ p \ .$$

Observe that for $i \in \mathcal{R}$ and $v \in S_i$, $T_v \leq r_v \leq r_S$, and hence the T_v-neighborhood in G of every node $v \in S_i$ is contained in S, which in turn is contained in G_1, hence $B_G(v, T_v) \subseteq G_1$. It therefore follows that for every such v, its view in the first T_v steps of $\mathrm{Run}(G_1, \mathbf{x}_1, \mathrm{Id}, \Gamma)$ of $\mathcal{A}$ is the same as in $\mathrm{Run}(G, \mathbf{x}, \mathrm{Id}, \Gamma)$ of $\mathcal{A}$, provided that the same sequences Γ of random bits were used for the random choices. Subsequently, since v halts after T_v steps of $\mathrm{Run}(G_1, \mathbf{x}_1, \mathrm{Id}, \Gamma)$, it will halt after T_v steps of $\mathrm{Run}(G, \mathbf{x}, \mathrm{Id}, \Gamma)$ too. Hence

$$\mathtt{Pr}[\mathcal{E}(G, \mathbf{x}, \mathrm{Id}, S_{\mathcal{I} \cap J})] \ = \ \mathtt{Pr}[\mathcal{E}(G_1, \mathbf{x}_1, \mathrm{Id}, S_{\mathcal{I} \cap J})] \geq p \ . \tag{1}$$

Consider two integers i and j in J. As J is μ-apart, $|i - j| \geq \mu$. Hence, the distance in G between any two nodes $u \in S_i$ and $v \in S_j$ is at least $2r_S + 1$. Thus, the events $\mathcal{E}(G, \mathbf{x}, \mathrm{Id}, S_i)$ and $\mathcal{E}(G, \mathbf{x}, \mathrm{Id}, S_j)$ are independent. It follows by the definition of $\mathcal{I}$, that

$$\mathtt{Pr}[\mathcal{E}(G, \mathbf{x}, \mathrm{Id}, S_{\mathcal{I} \cap J})] \ = \ \prod_{i \in \mathcal{I} \cap J} \mathtt{Pr}[\mathcal{E}(G, \mathbf{x}, \mathrm{Id}, S_i)] \ < \ (1 - \delta)^{|\mathcal{I} \cap J|} \ . \tag{2}$$

By (1) and (2), we have that $p < (1 - \delta)^{|\mathcal{I} \cap J|}$ and thus $|\mathcal{I} \cap J| < \log p / \log(1 - \delta)$.

Since $\mathcal{R}$ can be covered by the disjoint sets J_s, for $s = 1, \ldots, \mu$, we get that the sets $\mathcal{I} \cap J_s$, for $s = 1, \ldots, \mu$, form a partition of $\mathcal{I}$. As $|\mathcal{I} \cap J_s| < \log p / \log(1 - \delta)$ for every s, we have

$$|\mathcal{I}| \ = \ \sum_{s=1}^{\mu} |J_s \cap \mathcal{I}| \ < \ \mu(\log p / \log(1 - \delta)) \ .$$

As a consequence, we get that $(\lambda - 4)r_S - 1 > |\mathcal{I}|$. It follows by the pigeonhole principle that there exists some $i \in \mathcal{R}$ such that $i \notin \mathcal{I}$, as desired. This completes the proof of Claim 3.6. □

Applying Claim 3.6, let us fix $i \in \mathcal{R}$ such that $i \notin \mathcal{I}$, and let $\mathcal{F} = \mathcal{E}(G, \mathbf{x}, \mathrm{Id}, S_i)$. By definition,

$$\mathtt{Pr}[\overline{\mathcal{F}}] \leq \delta < p^2 + q - 1. \tag{3}$$

Let H_1 denote the subgraph of G induced by the nodes in $(\bigcup_{j=1}^{i-r_S-1} L_j) \cup U_1$. We similarly define H_2 as the subgraph of G induced by the nodes in $(\bigcup_{j>i+r_S} L_j) \cup U_2$. Note that S_i, $V(H_1)$, and $V(H_2)$ are pairwise disjoint, $S_i \cup V(H_1) \cup V(H_2) = V$, and for any two nodes $u \in V(H_1)$ and $v \in V(H_2)$ we have $d_G(u,v) > 2r_S$. It follows that, for $k = 1, 2$, the T_u-neighborhood in G of each node $u \in V(H_k)$ equals the T_u-neighborhood in G_k of u, that is, $B_G(u, t_u) \subseteq G_k$. To see why, fix $k \in \{1, 2\}$. Given $u \in V(H_k)$, it is sufficient to show that there is no $v \in V(H_{(k \bmod 2)+1})$, such that $v \in B_G(u, T_u)$. To establish the latter, we observe that if such a vertex v would exist, then $d_G(u,v) > 2r_S$, and thus $T_u > 2r_S$. Since there must exists a vertex $w \in S_i$ such that $w \in B(u, T_u)$, we would get that $r_w > 2r_S$, contradicting the fact that $w \in S$.

Since for $k = 1, 2$, $(G_k, \mathbf{x}_k) \in \mathcal{L}$, we get

$$\mathtt{Pr}[\mathcal{E}(G, \mathbf{x}, \mathrm{Id}, V(H_k))] \;=\; \mathtt{Pr}[\mathcal{E}(G_k, \mathbf{x}_k, \mathrm{Id}, V(H_k))] \;\geq\; p\ .$$

Let $\mathcal{F}' = \mathcal{E}(G, \mathbf{x}, \mathrm{Id}, V(H_1) \cup V(H_2))$. As the events $\mathcal{E}(G, \mathbf{x}, \mathrm{Id}, V(H_1))$ and $\mathcal{E}(G, \mathbf{x}, \mathrm{Id}, V(H_2))$ are independent, it follows that $\mathtt{Pr}[\mathcal{F}'] \geq p^2$, that is $\mathtt{Pr}[\overline{\mathcal{F}'}] \leq 1 - p^2$. Hence, combining this equation with Eqs. (3), and using union bound, it follows that $\mathtt{Pr}[\overline{\mathcal{F}} \vee \overline{\mathcal{F}'}] < q$. Thus,

$$\mathtt{Pr}[\mathcal{E}(G, \mathbf{x}, \mathrm{Id}, V(G))] \;=\; \mathtt{Pr}[\mathcal{E}(G, \mathbf{x}, \mathrm{Id}, S_i \cup V(H_1) \cup V(H_2))] \;=\; \mathtt{Pr}[\mathcal{F} \wedge \mathcal{F}'] > 1 - q,$$

contradicting the assumption that $(G, \mathbf{x}) \notin \mathcal{L}$. This establishes Lemma 3.5. □

Our goal now is to show that $\mathcal{L} \in \mathrm{LD}(O(t))$ by proving the existence of a deterministic local algorithm $\mathcal{D}$ that runs in time $O(t)$ and recognizes $\mathcal{L}$. (No attempt is made here to minimize the constant factor hidden in the $O(t)$ notation.) Recall that none of t, $T = T(G, \mathbf{x}, \mathrm{Id})$, or $T_v = T_v(G, \mathbf{x}, \mathrm{Id})$ may be known to v. Nevertheless, by inspecting the balls $B_G(v, 2^i)$ for increasing $i = 0, 1, 2, \ldots$, each node v can compute an upper bound on T_v, denoted T_v^*, as given by the following claim.

Claim 3.7 *Fix a constant $c > 0$, and let $(G, \mathbf{x})$ be an instance with an identity assignment* Id*. In $O(t)$ time, each node v can compute a value $T_v^* = T_v^*(c)$ such that*
(1) *$c \cdot T_v \leq T_v^* = O(t)$, and*
(2) *$T_u \leq T_v^*$ for every $u \in B_G(v, c \cdot T_v^*)$.*

Proof. To establish the claim, observe first that in $O(t)$ time, each node v can compute a value T'_v satisfying $T_v \leq T'_v \leq 2t$. Indeed, given the ball $B_G(v, 2^i)$, for some integer i, node v can simulate all its possible executions of the algorithm by considering all possible trials of the random coins at nodes in $B_G(v, r)$, for $2^{i-1} < r \leq 2^i$. (This is possible because the worst case execution time T_v over all coin flips is deterministic). The desired value satisfies $T'_v = 2^i$ where i is the smallest integer such that, for a certain $r \leq 2^i$, all executions of $\mathcal{A}$ at v up to round r conclude with an output at v.

Once T'_v is computed, node v aims at computing T_v^*. For this purpose, it starts again to inspect the balls $B_G(v, 2^i)$ for increasing $i = 0, 1, 2, \ldots$, in order to obtain T'_u from each $u \in B_G(v, 2^i)$. (For

this purpose, it may need to wait until u computes T'_u, but this delays the whole computation by at most $O(t)$ time.) Now, node v outputs $T^*_v = 2^i$ for the smallest i satisfying (1) $c \cdot T'_v \leq 2^i$ and (2) $T'_u \leq T^*_v$ for every $u \in B_G(v, c \cdot 2^i)$. For this i, we have $2^i = O(t)$, and thus $T^*_v = O(t)$. This establishes Claim 3.7. □

Given an instance $(G, \mathbf{x})$ and an identity assignment Id, the deterministic Algorithm $\mathcal{D}$, applied at a node u, first calculates T^*_u as in Claim 3.7, for $c = 6\lambda$. This can be done in $O(t)$ time. Subsequently, $\mathcal{D}$ outputs "yes" if and only if the $2\lambda T^*_u$-neighborhood of u in $(G, \mathbf{x})$ belongs to $\mathcal{L}$. That is,

$$\text{out}(u) = \text{“yes”} \iff (B_G(u, 2\lambda T^*_u), \mathbf{x}[B_G(u, 2\lambda T^*_u)]) \in \mathcal{L}.$$

Algorithm $\mathcal{D}$ is a deterministic algorithm that runs in time $O(t)$. (Recall that $\mathcal{L}$ is supposed to be (sequentially) decidable, so deciding whether $(B_G(u, 2\lambda T^*_u), \mathbf{x}[B_G(u, 2\lambda T^*_u)]) \in \mathcal{L}$ can be decided by every node u.) We claim that $\mathcal{D}$ decides $\mathcal{L}$. Indeed, since $\mathcal{L}$ is hereditary, if $(G, \mathbf{x}) \in \mathcal{L}$, then every prefix of $(G, \mathbf{x})$ is also in $\mathcal{L}$, and thus, every node u outputs out(u) ="yes". Now consider the case where $(G, \mathbf{x}) \notin \mathcal{L}$, and assume towards contradiction that by applying $\mathcal{D}$ on $(G, \mathbf{x})$ with identity assignment Id, every node u outputs out(u) ="yes". Let $U \subseteq V(G)$ be a maximal (under inclusion) set of vertices such that $G[U]$ is connected and $(G[U], \mathbf{x}[U]) \in \mathcal{L}$. Obviously, U is not empty, as $(B_G(u, 2\lambda T^*_v), \mathbf{x}[B_G(u, 2\lambda T^*_v)]) \in \mathcal{L}$ for every node u. On the other hand, we have $|U| < |V(G)|$, because $(G, \mathbf{x}) \notin \mathcal{L}$.

Let $u \in U$ be a node with maximal T_u such that $B_G(u, 2T_u)$ contains a node outside U. See Figure 2 for a graphical representation of node u, and of the sets of nodes used further in the proof. Define the subgraph of G induced by $U \cup V(B_G(u, 2T_u))$ as $G' = G[U \cup V(B_G(u, 2T_u))]$. Observe that G' is connected and that G' strictly contains U. Our goal is to show that $(G', \mathbf{x}[G']) \in \mathcal{L}$, in contradiction with the maximality of U.

Let H denote the maximal (under inclusion) graph such that H is connected and

$$V(B_G(u, 2T_u)) \subset V(H) \subseteq V(B_G(u, 2T_u)) \cup V(U \cap B_G(u, 2\lambda T^*_u)).$$

Let $W^1, W^2, \ldots, W^\ell$ be the ℓ connected components of $G[U] \setminus B_G(u, 2T_u)$, ordered arbitrarily. Let W^0 be the empty graph, and for $k = 0, 1, 2, \cdots, \ell$, define the graph

$$Z^k = H \cup W^0 \cup W^1 \cup W^2 \cup \cdots \cup W^k.$$

Observe that Z^k is connected for each $k = 0, 1, 2, \cdots, \ell$, and that $Z^\ell = G'$. We prove by induction on k that $(Z^k, \mathbf{x}[Z^k]) \in \mathcal{L}$ for every $k = 0, 1, 2, \ldots, \ell$. This will establish the contradiction since, as we mentioned before, $Z^\ell = G'$. For the basis of the induction, the case $k = 0$, we need to show that $(H, \mathbf{x}[H]) \in \mathcal{L}$. However, this is immediate by the facts that H is a connected subgraph of $B_G(u, 2\lambda t^*_u)$, the instance $(B_G(u, 2\lambda T^*_u), \mathbf{x}[B_G(u, 2\lambda T^*_u)]) \in \mathcal{L}$, and $\mathcal{L}$ is hereditary. Assume now that we have $(Z^k, \mathbf{x}[Z^k]) \in \mathcal{L}$ for $0 \leq k < \ell$, and consider the graph $Z^{k+1} = Z^k \cup W^{k+1}$. Define the sets of nodes

$$S = V(Z^k) \cap V(W^{k+1}), \quad U_1 = V(Z^k) \setminus S, \quad \text{and} \quad U_2 = V(W^{k+1}) \setminus S.$$

A crucial observation is that (S, U_1, U_2) is a separating partition of Z^{k+1}. This follows from the following arguments. Let us first show that $r_S \leq T^*_u$. By definition, we have $T_v \leq T^*_u$, for every $v \in B_G(u, 6\lambda T^*_u)$. Hence, in order to bound the radius of S (in Z^{k+1}) by T^*_u it is sufficient to prove

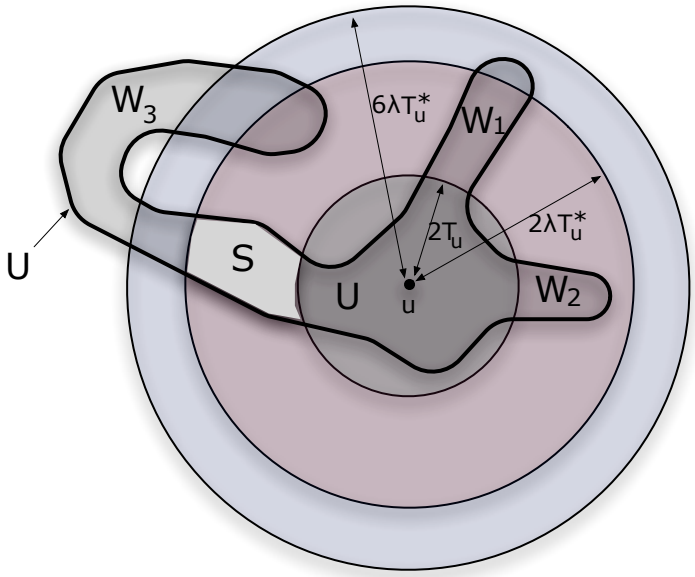

Figure 2: Illustration of the several node sets used in the proof of Theorem 3.4.

that there is no node $w \in U \setminus B_G(u, 6\lambda T_u^*)$ such that $B_G(w, T_w) \cap S \neq \emptyset$. Indeed, if such a node w exists then $T_w > 4\lambda T_u^*$ and hence $B_G(w, 2T_w)$ contains a node outside U, in contradiction to the choice of u, based on the maximality of T_u for this latter property. It follows that $r_S \leq T_u^*$.

We now claim that $\mathrm{dist}_{Z^{k+1}}(U_1, U_2) \geq \lambda T_u^*$. Consider a simple directed path P in Z^{k+1} going from a node $x \in U_1$ to a node $y \in U_2$. Since $x \notin V(W^{k+1})$ and $y \in V(W^{k+1})$, we get that P must pass through a vertex in $B_G(u, 2T_u)$. Let z be the last vertex in P such that $z \in B_G(u, 2T_u)$, and consider the directed subpath $P_{[z,y]}$ of P going from z to y. Now, let $P' = P_{[z,y]} \setminus \{z\}$. The first $d' = \min\{(2\lambda - 2)T_u^*, |P'|\}$ vertices in the directed subpath P' must belong to $V(H) \subseteq V(Z^k)$. In addition, observe that all nodes in P' must be in $V(W^{k+1})$. It follows that the first d' nodes of P' are in S. Since $y \notin S$, we get that $|P'| \geq d' = (2\lambda - 2)T_u^*$, and thus $|P| > \lambda T_u^*$. Consequently, $\mathrm{dist}_{Z^{k+1}}(U_1, U_2) \geq \lambda T_u^* \geq \lambda r_S$, as desired. This completes the proof that (S, U_1, U_2) is a separating partition of Z^{k+1}.

Now, by the induction hypothesis, we have $(G_1, \mathbf{x}[G_1]) \in \mathcal{L}$, because $G_1 = G[U_1 \cup S] = Z^k$. In addition, we have $(G_2, \mathbf{x}[G_2]) \in \mathcal{L}$, because $G_2 = G[U_2 \cup S] = W^{k+1}$, and W^{k+1} is a prefix of $G[U]$. We can now apply Lemma 3.5 and conclude that $(Z^{k+1}, \mathbf{x}[Z^{k+1}]) \in \mathcal{L}$. This concludes the induction proof. The theorem follows. □

4 Nondeterminism and Complete Problems

4.1 Separation Results

We first establish two simple separation results, whose proofs use rather standard arguments. Our first separation result indicates that nondeterminism helps for local decision. Indeed, we show that there exists a language that belongs to $\mathrm{NLD} = \mathrm{NLD}(O(1))$ but not to $\mathrm{LD}(t)$ for any $t = o(n)$. The proof is based on the fact that the language composed of trees cannot be decided locally (because locally, a cycle looks like a tree). On the other hand, the fact that the underlying graph is a tree can be verified in 1 round using a certificate at each node v containing the distance from v to a unique "root" node r. The second separation result shows that nondeterminism helps only up to a certain extent, as there exists a language which cannot be locally and nondeterministically decided. Basically, this language consists of graphs where each node has a local input that equals the precise number of nodes in the graph.

Theorem 4.1 $\mathrm{LD}(O(t)) \neq \mathrm{NLD}(O(t))$, *for any* $t = o(n)$.

Proof. To establish the theorem it is sufficient to show that there exists a language $\mathcal{L}$ such that $\mathcal{L} \notin \mathrm{LD}(o(n))$ and $\mathcal{L} \in \mathrm{NLD}(1)$. Recall that $\texttt{Tree} = \{(G, \epsilon) \mid G \text{ is a tree}\}$. We first observe that $\texttt{Tree} \notin \mathrm{LD}(o(n))$. To see why, consider the cycle C with nodes labeled consecutively from 1 to $4n$, and the two paths P_1 and P_2 with nodes labeled consecutively $1, \ldots, 4n$ and $2n+1, \ldots, 4n, 1, \ldots, 2n$ respectively, from one extremity to the other. For any algorithm $\mathcal{A}$ deciding $\texttt{Tree}$, all nodes $n+1, \ldots, 3n$ output "yes" in instance (P_1, ϵ) for any identity assignment for the nodes in P_1, while all nodes $3n+1, \ldots, 4n, 1, \ldots, n$ output "yes" in instance (P_2, ϵ) for any identity assignment for the nodes in P_2. Thus if $\mathcal{A}$ runs in $o(n)$ rounds, we get that all nodes output "yes" in instance (C, ϵ) for an n-node cycle, with n large enough. This yields a contradiction with the correctness of $\mathcal{A}$.

In contrast, we next show that $\texttt{Tree} \in \mathrm{NLD}(1)$. The (nondeterministic) local algorithm $\mathcal{A}$ verifying $\texttt{Tree}$ operates as follows. Given an instance (G, ϵ), the certificate given at node v is $\mathbf{y}(v) = \mathrm{dist}_G(v, \hat{v})$, where $\hat{v} \in V(G)$ is an arbitrary fixed node. The verification procedure is then as follows. At each node v, $\mathcal{A}$ inspects every neighbor (with its certificates), and verifies the following:

- $\mathbf{y}(v)$ is a nonnegative integer,
- if $\mathbf{y}(v) = 0$, then $\mathbf{y}(w) = 1$ for every neighbor w of v, and
- if $\mathbf{y}(v) > 0$, then there exists a neighbor w of v such that $\mathbf{y}(w) = \mathbf{y}(v) - 1$, and, for all other neighbors w' of v, we have $\mathbf{y}(w') = \mathbf{y}(v) + 1$.

If G is a tree, then applying Algorithm $\mathcal{A}$ on G with the certificate yields the answer "yes" at all nodes regardless of the given identity assignment . On the other hand, if G is not a tree, then we claim that for every certificate, and every identity assignment Id, Algorithm $\mathcal{A}$ outputs "no" at some node. Indeed, consider some certificate $\mathbf{y}$ given to the nodes of G, and let C be a simple cycle in G. Assume, for the sake of contradiction, that all nodes in C output "yes". In this case, each node in C has at least one neighbor in C with a larger certificate. This creates an infinite sequence of strictly increasing certificates, in contradiction with the finiteness of C. □

Theorem 4.2 *For any $t = o(n)$,* $\mathrm{NLD}(O(t)) \neq \mathrm{All}$.

Proof. Let $\texttt{GraphSize} = \{(G, \mathbf{x}) \mid \forall v \in V(G),\ \mathbf{x}(v) = |V(G)|\}$, i.e., $\mathbf{x}$ represents the number of nodes in the graph. We show that $\texttt{GraphSize} \notin \mathrm{NLD}(t)$, for any $t = o(n)$. Assume, for the sake of contradiction, that there exists a local nondeterministic algorithm $\mathcal{A}$ deciding $\texttt{GraphSize}$ in time $o(n)$. Let n be the smallest integer such that the running time of $\mathcal{A}$ on the instance $(C, \mathbf{x})$ is $t < \lceil \frac{n}{4} \rceil$ for every identity assignment Id, where $C = (u_0, u_1, \ldots, u_{n-1})$ is the n-node cycle, and $\mathbf{x}(u_i) = n$ for every $i = 0, \ldots, n-1$. We have $(C, \mathbf{x}) \in \texttt{GraphSize}$, thus there exists a certificate $\mathbf{y}$ such that, for any identity assignment Id of C, algorithm $\mathcal{A}$ outputs $\mathrm{out}_{\mathcal{A}}(C, \mathbf{x}, \mathbf{y}, \mathrm{Id}, u_i) =$"yes" at each node u_i of C. Now, consider the instance $(C', \mathbf{x}')$, where $C' = (v_0, v_1, \ldots, v_{2n-1})$ is the $2n$-node cycle, and for each node v_i of C', $\mathbf{x}'(v_i)$. We have $(C', \mathbf{x}') \notin \texttt{GraphSize}$. Nevertheless, it is possible to fool Algorithm $\mathcal{A}$ as follows. Define the certificate $\mathbf{y}'$ as follows:

$$\mathbf{y}'(v_i) \ = \ \mathbf{y}'(v_{i+n}) \ = \ \mathbf{y}(u_i) \ \text{ for } i = 0, 1, \ldots, n-1 \ .$$

Fix an identity assignment Id' for the nodes in $V(C')$, and let $i \in \{0, 1, \ldots, 2n-1\}$. Let us then define the partial identity assignment Id for C by:

$$\mathrm{Id}(u_{i+j \bmod n}) = \mathrm{Id}'(v_{i+j \bmod 2n}) \text{ and } \mathrm{Id}(u_{i-j \bmod n}) = \mathrm{Id}'(v_{i-j \bmod 2n}) \text{ for } j = 0, \ldots, \left\lceil \frac{n}{4} \right\rceil .$$

Since $\mathcal{A}$ runs in $t < \lceil \frac{n}{4} \rceil$ in $(C, \mathbf{x})$, we have $\mathrm{out}_{\mathcal{A}}(C', \mathbf{x}', \mathbf{y}', \mathrm{Id}', v_i) = \mathrm{out}_{\mathcal{A}}(C, \mathbf{x}, \mathbf{y}, \mathrm{Id}, u_i)$ because the t-neighborhoods of $u_{i \bmod n}$ in C and v_i in C' are identical. Therefore, $\mathrm{out}_{\mathcal{A}}(C', \mathbf{x}', \mathbf{y}', \mathrm{Id}', v_i) =$"yes". Hence, every node v_i outputs "yes" for $(C', \mathbf{x}')$ with identity assignment Id', contradicting the fact that $(C', \mathbf{x}') \notin \texttt{GraphSize}$. □

For $p, q \in (0, 1]$ and a function t, let us define $\mathrm{BPNLD}(t, p, q)$ as the class of all distributed languages that have a local randomized nondeterministic distributed (p, q)-decider running in time t. We show that such a combination of randomization with nondeterminism enables to capture all distributed languages.

Theorem 4.3 *Let $p, q \in (0, 1]$ such that $p^2 + q \leq 1$. Then,* $\mathrm{BPNLD}(1, p, q) = \mathrm{All}$.

Proof. Given an arbitrary language $\mathcal{L}$, let us describe a constant time nondeterministic (p, q)-decider for it. The certificate of a instance $(G, \mathbf{x}) \in \mathcal{L}$ is a map of G, with nodes labeled by distinct integers, with labeling $\lambda : V(G) \mapsto \{1, ..., n\}$, where $n = |V(G)|$, together with the inputs of all nodes in G. In addition, every node v receives the label $\lambda(v)$ of the corresponding vertex in the map. More formally, the certificate at node v is $\mathbf{y}(v) = (G', \mathbf{x}', i)$, where G' is an isomorphic copy of G with nodes labeled by λ from 1 to n, $\mathbf{x}'$ is an n-dimensional vector such that $\mathbf{x}'[\lambda(u)] = \mathbf{x}(u)$ for every node u, and $i = \lambda(v)$.

The verification algorithm involves checking that the instance $(G', \mathbf{x}')$ is identical to $(G, \mathbf{x})$. This is sufficient because distributed languages are sequentially decidable, hence every node can individually decide whether $(G', \mathbf{x}')$ belongs to $\mathcal{L}$ or not, once it has secured the fact that $(G', \mathbf{x}')$ is the actual instance. It remains to show that there exists a local randomized nondeterministic distributed (p, q)-decider for verifying that the instance $(G', \mathbf{x}')$ is identical to $(G, \mathbf{x})$, and running in time 1.

The nondeterministic (p, q)-decider operates as follows. First, every node v checks that it has received the input as specified by $\mathbf{x}'$, i.e., v checks whether $\mathbf{x}'[\lambda(v)] = \mathbf{x}(v)$, and outputs "no" if this does not hold. Second, each node v communicates with its neighbors to check that (1) they all got the same map G' and the same input vector $\mathbf{x}'$, and (2) they are labeled the way they should be according to the map G'. If some inconsistency is detected by a node, then this node outputs "no". Finally, consider a node v that passed the aforementioned two tests without outputting "no". If $\lambda(v) \neq 1$ then v outputs "yes" (with probability 1), and if $\lambda(v) = 1$ then v outputs "yes" with probability p.

We claim that the above implements a nondeterministic distributed (p, q)-decider for verifying that the instance $(G', \mathbf{x}')$ is identical to $(G, \mathbf{x})$. Indeed, if all nodes pass the two tests without outputting "no", then they all agree on the map G' and on the input vector $\mathbf{x}'$, and they know that their respective neighborhood fits with what is indicated on the map[5]. It follows that $(G', \mathbf{x}') = (G, \mathbf{x})$ if and only if there exists at most one node $v \in G$, whose label satisfies $\lambda(v) = 1$. This is precisely the `AMOS` problem which we already know can be decided by a (p, q)-decider whenever $p^2 + q \leq 1$ (see Theorem 3.2). This completes the proof of Theorem 4.3. □

The above theorem guarantees that the following is well-defined. Fix some $p, q \in (0, 1]$ such that $p^2 + q \leq 1$, and let BPNLD = BPNLD$(1, p, q)$. The next corollary follows from Theorems 4.1, 4.2 and 4.3.

Corollary 4.4 *For every $t = o(n)$, we have* LD$(O(t)) \subset$ NLD$(O(t)) \subset$ BPNLD = All.

It turns out that there exist some interesting connections between randomization and oracles, as far as nondeterministic computing is concerned. Motivated by the numerous examples in the literature for which the knowledge of the size of the network is required to efficiently compute solutions of distributed computing problems (cf., e.g., [36, 42, 43, 48]), we specifically focus on the oracle providing the nodes with the size of the graph. Roughly, we show that such an oracle gives the same power to nondeterministic distributed computing as randomization does. More precisely, let NLD$^{\texttt{GraphSize}}$ be the class of languages that can be locally verified by a distributed verification algorithm enhanced with an oracle for `GraphSize` (i.e., every node has access to an oracle deciding `GraphSize`).

Theorem 4.5 NLD$^{\texttt{GraphSize}}$ = BPNLD = All.

Proof. Let $\mathcal{L}$ be a language. The certificate of an instance $(G, \mathbf{x}) \in \mathcal{L}$ is identical to the one in the proof of Theorem 4.3. We show that verifying that the instance $(G', \mathbf{x}')$ provided by the certificate is identical to $(G, \mathbf{x})$ can be done locally, assuming that every node has access to an oracle deciding `GraphSize`. The verifying algorithm operates as follows. First, every node queries the oracle to get the number of nodes n of the actual instance, and a node noticing $n \neq |V(G')|$ outputs "no". Second, every node v checks that it has received the input as specified by $\mathbf{x}'$, i.e., v checks whether $\mathbf{x}'[\lambda(v)] = \mathbf{x}(v)$, and outputs "no" if this does not hold. Third, each node v communicates with its neighbors to check that (1) they all got the same map G' and the same input vector $\mathbf{x}'$, and (2) they are labeled the way they should be according to the map G'. If some inconsistency is detected

[5] $(G', \mathbf{x}')$ is actually a lift of $(G, \mathbf{x})$ [3].

by a node, then this node outputs "no". We claim that if all nodes pass these three phases without outputting "no", then $(G', \mathbf{x}')$ is identical to $(G, \mathbf{x})$. Indeed, if all nodes pass the three phases without outputting "no", then they all agree on the map G' and on the input vector $\mathbf{x}'$, they know that the map has the right size, and they know that their respective neighborhood fits with what is indicated on the map. Let L_i be the n-dimensional boolean array such that $L_i[j] = 1$ if and only if node labeled i has a neighboring node labeled j. Observe that all nodes have received pairwise distinct labels in their certificate, and that these labels are in $[1, n]$. Indeed, otherwise, by the fact that $|V(G)| = n$, there would be at least one label not assigned to any node, and this fact would be detected during the third phase of the above algorithm. So L_i is defined for every $i = 1, \ldots, n$. By construction, the $n \times n$ matrix M whose ith row is L_i is the adjacency matrix of both G and G'. Thus, G and G' are isomorphic. This completes the proof of Theorem 4.5. □

4.2 Completeness Results

Let us define a notion of reduction that fits with the class LD.

Definition 4.6 *For two languages $\mathcal{L}_1$ and $\mathcal{L}_2$, we say that $\mathcal{L}_1$ is* locally reducible *to $\mathcal{L}_2$, denoted by $\mathcal{L}_1 \preceq \mathcal{L}_2$, if there exists a constant time local algorithm $\mathcal{A}$ such that, for every instance $(G, \mathbf{x})$ and every identity assignment* Id*, $\mathcal{A}$ produces* $\mathrm{out}(v) \in \{0,1\}^*$ *as output at every node $v \in V(G)$ so that*

$$(G, \mathbf{x}) \in \mathcal{L}_1 \iff (G, \mathrm{out}) \in \mathcal{L}_2 \ .$$

By definition, $\mathrm{LD}(O(t))$ is closed under local reductions, that is, for every two languages $\mathcal{L}_1$ and $\mathcal{L}_2$ satisfying $\mathcal{L}_1 \preceq \mathcal{L}_2$, if $\mathcal{L}_2 \in \mathrm{LD}(O(t))$ then $\mathcal{L}_1 \in \mathrm{LD}(O(t))$.

It is somewhat surprising that even under this very weak notion of reduction there exists a problem which is, in some sense, the "most difficult" decision problem. The corresponding language, called `Cover` is defined as follows. Every node v of the input graph G is given as input a pair $\mathbf{x}(v) = (\mathcal{E}(v), \mathcal{S}(v))$, where $\mathcal{E}(v)$ is an element and $\mathcal{S}(v)$ is a finite collection of sets. The instance $(G, \mathbf{x})$ is in `Cover` if and only if there exists a node v such that one set in $\mathcal{S}(v)$ equals the union of all the elements given to the nodes. Formally,

$$\texttt{Cover} \ = \ \{(G, (\mathcal{E}, \mathcal{S})) \mid \exists v \in V(G),\ \exists S \in \mathcal{S}(v) \text{ s.t. } S = \{\mathcal{E}(u) \mid u \in V(G)\}\} \ .$$

Theorem 4.7 `Cover` *is* BPNLD*-complete.*

Proof. The fact that `Cover` $\in$ BPNLD follows from Theorem 4.3. To prove that `Cover` is BPNLD-hard, we consider some $\mathcal{L} \in$ BPNLD and show that $\mathcal{L} \preceq$ `Cover`. For this purpose, we describe a local distributed algorithm $\mathcal{A}$ transforming any instance for $\mathcal{L}$ into an instance for `Cover` while preserving membership. Let $(G, \mathbf{x})$ be an instance for $\mathcal{L}$ and let Id be an identity assignment. Algorithm $\mathcal{A}$ operating at a node v outputs a pair $(\mathcal{E}(v), \mathcal{S}(v))$, where $\mathcal{E}(v)$ is the "local view" at v in $(G, \mathbf{x})$, i.e., the star subgraph of G consisting of v and its neighbors, together with the inputs of these nodes and their identities, and $\mathcal{S}(v)$ is the collection of sets S defined as follows. For a binary string x, let $|x|$ denote its *length* in bits. For every vertex v, let

$$\mathrm{width}(v) = 2^{|\mathrm{Id}(v)| + |\mathbf{x}(v)|}.$$

Node v first generates all instances $(G', \mathbf{x}')$, where G' is a graph with $k \leq \text{width}(v)$ vertices and $\mathbf{x}'$ is a collection of k input strings of length at most width(v), such that $(G', \mathbf{x}') \in \mathcal{L}$. For each such instance $(G', \mathbf{x}')$, node v generates all possible identity assignments Id' to $V(G')$ such that for every node $u \in V(G')$, $|\text{Id}(u)| \leq \text{width}(v)$. Now, for each such pair of a graph $(G', \mathbf{x}')$ and an identity assignment Id', algorithm $\mathcal{A}$ associates a set $S \in \mathcal{S}(v)$ consisting of the $k = |V(G')|$ local views of the nodes of G' in $(G', \mathbf{x}')$. We show that

$$(G, \mathbf{x}) \in \mathcal{L} \iff (G, \text{out}_{\mathcal{A}}) \in \texttt{Cover}.$$

If $(G, \mathbf{x}) \in \mathcal{L}$, then by the construction of Algorithm $\mathcal{A}$, there exists a set $S \in \mathcal{S}(v)$ such that S covers the collection of local views for $(G, \mathbf{x})$, i.e., $S = \{\mathcal{E}(u) \mid u \in G\}$. Indeed, the node v maximizing width(v) satisfies

$$\text{width}(v) \geq \max\{\text{Id}(u) \mid u \in V(G)\} \geq n \qquad \text{and} \qquad \text{width}(v) \geq \max\{\mathbf{x}(u) \mid u \in V(G)\}.$$

Therefore, that specific node has constructed a set S that contains all local views of the given instance $(G, \mathbf{x})$ and Id assignment. Thus $(G, \text{out}_{\mathcal{A}}) \in \texttt{Cover}$.

Now consider the case that $(G, \text{out}_{\mathcal{A}}) \in \texttt{Cover}$. In this case, there exists a node v and a set $S \in \mathcal{S}(v)$ such that $S = \{\mathcal{E}(u) \mid u \in G\}$. Such a set S is the collection of local views of nodes of some instance $(G', \mathbf{x}') \in \mathcal{L}$ and some identity assignment Id'. On the other hand, S is also the collection of local views of nodes of the given instance $(G, \mathbf{x}) \in \mathcal{L}$ and identity assignment Id. It follows that $(G, \mathbf{x}) = (G', \mathbf{x}') \in \mathcal{L}$. □

Finally, finding an NLD-complete problem was not an easy task. Eventually, we managed to find a problem, called `Containment`, which is NLD-complete. Somewhat surprisingly, the definition of `Containment` is quite similar to the definition of `Cover`. Specifically, as in `Cover`, every node v is given as input a pair $\mathbf{x}(v) = (\mathcal{E}(v), \mathcal{S}(v))$, where $\mathcal{E}(v)$ is an element and $\mathcal{S}(v)$ is a finite collection of sets. However, in contrast to `Cover`, the union of these inputs is in the language `Containment` if there exists a node v such that some set in $\mathcal{S}(v)$ *contains* the union of all the elements given to the nodes. Formally, define

$$\texttt{Containment} = \{(G, (\mathcal{E}, \mathcal{S})) \mid \exists v \in V(G),\ \exists S \in \mathcal{S}(v) \text{ s.t. } S \supseteq \{\mathcal{E}(u) \mid u \in V(G)\}\}.$$

Theorem 4.8 `Containment` *is* NLD*-complete.*

Proof. We first prove that `Containment` is NLD-hard. Let $\mathcal{L} \in$ NLD. We show that $\mathcal{L} \preceq$ `Containment`. For this purpose, we describe a local distributed algorithm $\mathcal{A}$ transforming any instance for $\mathcal{L}$ to an instance for `Containment` while preserving membership.

Let $t \geq 0$ be some (constant) integer such that there exists a local nondeterministic algorithm $\mathcal{N}$ deciding $\mathcal{L}$ in time at most t. Let $(G, \mathbf{x})$ be an instance for $\mathcal{L}$ and let Id be an identity assignment. Algorithm $\mathcal{A}$ operating at a node v outputs a pair $(\mathcal{E}(v), \mathcal{S}(v))$, where $\mathcal{E}(v)$ is the "t-local view" at v in $(G, \mathbf{x})$, i.e., the ball of radius t around v, $B_G(v, t)$, together with the inputs of these nodes and their identities, and $\mathcal{S}(v)$ is the collection of sets S defined as follows. We set width(v) as in the proof of Theorem 4.7. Node v first generates all instances $(G', \mathbf{x}')$ where G' is a graph with $k \leq \text{width}(v)$ vertices, and $\mathbf{x}'$ is a collection of k input strings of length at most width(v), such that

$(G', \mathbf{x}') \in \mathcal{L}$. For each such instance $(G', \mathbf{x}')$, node v generates all possible identity assignments Id' to $V(G')$ such that for every node $u \in V(G')$, $|\mathrm{Id}(u)| \leq \mathrm{width}(v)$. Now, for each such pair of a graph $(G', \mathbf{x}')$ and an identity assignment Id', algorithm $\mathcal{A}$ associates a set $S \in \mathcal{S}(v)$ consisting of the $m = |V(G')|$ t-local views of the nodes of G' in $(G', \mathbf{x}')$.

We show that: $(G, \mathbf{x}) \in \mathcal{L} \iff \mathcal{A}(G, \mathbf{x}) \in \texttt{Containment}$.

If $(G, \mathbf{x}) \in \mathcal{L}$, then by the construction of Algorithm $\mathcal{A}$, there exists a set $S \in \mathcal{S}(v)$ such that S covers the collection of t-local views for $(G, \mathbf{x})$, i.e., $S = \{\mathcal{E}(u) \mid u \in G\}$. Indeed, as in the proof of Theorem 4.7, the node v maximizing $\mathrm{width}(v)$ satisfies

$$\mathrm{width}(v) \geq \max\{\mathrm{Id}(u) \mid u \in V(G)\} \geq n \qquad \text{and} \qquad \mathrm{width}(v) \geq \max\{\mathbf{x}(u) \mid u \in V(G)\}.$$

Therefore, that specific node has constructed a set S that precisely corresponds to $(G, \mathbf{x})$ and its given Id assignment; hence, S contains all corresponding t-local views. Thus, $\mathcal{A}(G, \mathbf{x}) \in \texttt{Containment}$.

Now consider the case that $\mathcal{A}(G, \mathbf{x}) \in \texttt{Containment}$. In this case, there exists a node v and a set $S \in \mathcal{S}(v)$ such that $S \supseteq \{\mathcal{E}(u) \mid u \in G\}$. Such a set S is the collection of t-local views of nodes of some instance $(G', \mathbf{x}') \in \mathcal{L}$ and some identity assignment Id'. Since $(G', \mathbf{x}') \in \mathcal{L}$, there exists a certificate $\mathbf{y}'$ for the nodes of G', such that when algorithm $\mathcal{N}$ operates on $(G', \mathbf{x}', \mathbf{y}')$, all nodes say "yes". Now, since S contains the t-local views of nodes $(G, \mathbf{x})$, with the corresponding identities, there exists a mapping $\phi : (G, \mathbf{x}, \mathrm{Id}) \to (G', \mathbf{x}', \mathrm{Id}')$ that preserves inputs and identities. Moreover, when restricted to a ball of radius t around a vertex $v \in G$, ϕ is actually an isomorphism between this ball and its image. We assign a certificate $\mathbf{y}$ to the nodes of G: for each $v \in V(G)$, $\mathbf{y}(v) = \mathbf{y}'(\phi(v))$. Now, Algorithm $\mathcal{N}$ when operating on $(G, \mathbf{x}, \mathbf{y})$ outputs "yes" at each node of G. By the correctness of $\mathcal{N}$, we obtain $(G, \mathbf{x}) \in \mathcal{L}$.

We now show that $\texttt{Containment} \in \mathrm{NLD}$. For this purpose, we design a nondeterministic local algorithm $\mathcal{D}$ that decides whether an instance $(G, \mathbf{x})$ is in $\texttt{Containment}$. Such an algorithm $\mathcal{D}$ is designed to operate on $(G, \mathbf{x}, \mathbf{y})$, where $\mathbf{y}$ is a certificate. The instance $(G, \mathbf{x})$ satisfies that $\mathbf{x}(v) = (\mathcal{E}(v), \mathcal{S}(v))$. Algorithm $\mathcal{A}$ aims at verifying whether there exists a node v^* with a set $S^* \in \mathcal{S}(v^*)$ such that $S^* \supseteq \{\mathcal{E}(v) \mid v \in V(G)\}$. Given a correct instance, i.e., an instance $(G, \mathbf{x})$, we define the certificate $\mathbf{y}$ as follows. For each node v, the certificate $\mathbf{y}(v)$ at v consists of several fields, specifically,

$$\mathbf{y}(v) = (\mathbf{y}_c(v), \mathbf{y}_s(v), \mathbf{y}_{id}(v), \mathbf{y}_l(v)).$$

The *candidate instance* field $\mathbf{y}_c(v)$ is a triplet $\mathbf{y}_c(v) = (G', \mathbf{x}', \mathrm{Id}')$, where $(G', \mathbf{x}')$ is an isomorphic copy $(G', \mathbf{x}')$ of $(G, \mathbf{x})$ and Id' is an identity assignment for the nodes of G'. The *candidate set* field $\mathbf{y}_s(v)$ is a copy of S^*, i.e., $\mathbf{y}_s(v) = S^*$. In addition, let u and u^* be the nodes in $(G', \mathbf{x}')$ corresponding to v and v^*, respectively. The *candidate identity* field $\mathbf{y}_{id}(v)$ is $\mathbf{y}_{id}(v) = \mathrm{Id}'(u)$, and the *candidate leader* field $\mathbf{y}_l(v)$ is $\mathbf{y}_l(v) = \mathrm{Id}'(u^*)$.

We describe the operation of Algorithm $\mathcal{D}$ on some triplet $(G, \mathbf{x}, \mathbf{y})$. First, each node v verifies that it agrees with its neighbors on the candidate instance and candidate set fields in their certificates. That way, if all nodes say "yes" then we know that all nodes hold the same candidate instance which is some triplet $(G', \mathbf{x}', \mathrm{Id}')$, and the same candidate set S'. Second, each node verifies that $\mathcal{E}(v) \in S'$. Also, each node checks that it agrees with its neighbors on the candidate leader field in their certificates. I.e, that there exists some integer k such that for all nodes v we have $\mathbf{y}_l(v) = k$. Each node v checks that there exists a node $u^* \in V(G')$ such that $\mathrm{Id}'(u^*) = k$, and that

there exists a node $v' \in V(G')$ such that $\mathbf{y}_{id}(v) = \mathrm{Id}'(v')$. Moreover, node v verifies that the input $\mathbf{x}'$ at v' contains a collection of sets $\mathcal{S}'(v')$ that contains S', that is, $S' \in \mathcal{S}'(v')$. Finally, each node v verifies that its immediate neighborhood $B_G(v,1)$ agrees with the corresponding neighborhood of v' in G', and that the candidate identities $\mathbf{y}_{id}(w)$ of its neighbors $w \in B_G(v,1)$ are compatible with the corresponding identities $\mathrm{Id}'(w')$ in G'. We term this verification the *neighborhood check* of v.

It is easy to see that when applying Algorithm $\mathcal{D}$ on a correct instance, together with the certificate described above, each node outputs "yes". We now show the other direction. Assume that Algorithm $\mathcal{D}$ applied on some triplet $(G, \mathbf{x}, \mathbf{y})$ outputs "yes" at each node, our goal is to show that $(G, \mathbf{x}) \in \mathcal{L}$. Since all nodes say "yes" on $(G, \mathbf{x}, \mathbf{y})$, it follows that the certificate $\mathbf{y}(v)$ at every node $v \in V(G)$ contains the same candidate instance field $(G', \mathbf{x}', \mathrm{Id}')$, the same candidate set S' and the same pointer $\mathrm{Id}'(v')$ to a vertex $v' \in G'$, such that $S' \in \mathcal{S}(v')$. Since each node $v \in V(G)$ verifies that $\mathcal{E}(v) \in S'$, it follows that $S' \supseteq \{\mathcal{E}(v) \mid v \in V(G)\}$. It remains to show that there exists a node $v^* \in V(G)$ such that $S' \in \mathcal{S}(v^*)$. This follows by the neighborhood checks of all nodes. □

5 Discussion and Future Work

This paper aims at making a first step in the direction of establishing a complexity theory for local distributed computing in networks. We conclude the paper by discussing several important issues related to our work and listing some open problems.

5.1 Connection to Sequential Complexity Theory

Our model of computation, namely, the $\mathcal{LOCAL}$ model, focuses on difficulties arising from pure locality issues, and abstracts away other complexity aspects. Naturally, it would be very interesting to come up with a rigorous complexity framework taking into account also other complexity measures. In particular, it would be interesting to investigate the connections between classical computational complexity theory and local complexity theory. A possible approach that may provide a bridge for connecting the two theories can be based on imposing constraints on the individual computational power at each node in each round, at least asymptotically, e.g., requiring the computations performed individually at a node in one round to be at most polynomial, or exponential, etc. (Note that the reduction in the proof of Theorems 4.7 and 4.8 may require exponential local computations). Also, one could restrict the memory used by a node, in addition to, or instead of, bounding its local computational power. Finally, it would be interesting to come up with a complexity framework taking also traffic congestion into account. (This can be done by, e.g., considering the $\mathcal{CONGEST}$ model).

Note that the framework studied in this paper requires each node to make its decisions based on a computation performed by an algorithm. Relaxing the framework by enabling the use of noncomputable functions may change the nature of the study. The reader is referred to [20], which discusses the power of allowing the nodes to access oracles returning the values of noncomputable functions.

5.2 Role of Identities

The $\mathcal{LOCAL}$ model assumes pairwise distinct identities given to the nodes. It is known that these identities play an important role in the context of *construction* algorithms, i.e., algorithms in which every node u must compute some output $\mathbf{x}(u)$ so that $(G, \mathbf{x}) \in \mathcal{L}$, where $\mathcal{L}$ is some distributed language specifying the task to be performed (e.g., coloring). In particular, the presence of distinct identities enables the nodes to break symmetry, which is crucial for many tasks (e.g., leader election, maximal independent set, coloring, etc.). On the other hand, identities do not seem to play such a significant role in the context of *decision* algorithms. In particular, all algorithms designed for decision in this paper, including the deterministic algorithm resulting from derandomization technique in the proof of Theorem 3.4, no longer use identities once the nodes have collected all the information available in their t-neighborhood. In fact, the role of the identities in the context distributed local decision was studied explicitly in [21], where it is proved that identities play no role for large classes of decision problems, or when nodes possess specific information about their environment (e.g., their total number). On the other hand, it has been recently proved [20] that identities do play a role in general. That is, there exist distributed languages that *can* be decided by an algorithm that makes use of identities, but not by any *identity-oblivious* algorithm (i.e., one whose outputs at every node are identical for all possible identity assignments).

The role of the identities is made more explicit in the context of *verification* (or *nondeterministic decision*) algorithms, that is, when every node is given a *certificate* (or a *guess*). In this context, the distributed algorithm is in charge of verifying that these certificates collaboratively form a proof that the given instance belongs to the considered language. The behavior of such verification algorithms differs significantly according to whether or not the certificates depend on the identities. In this paper, the certificates are not a function of the identities, and two instances $(G, \mathbf{x})$ that are the same except for the node identities associate the same certificate with each node. In contrast, in the framework of proof-labeling schemes [29, 33, 35], the certificates may vary depending on the node identities, enabling to verify all languages in a single round [35].

5.3 Role of Randomization

Many interesting questions are left open. First, an intriguing question is whether or not Theorem 3.4 holds also for non-hereditary languages. In addition, it would be interesting to investigate the connections between $\mathrm{BPLD}(t, p, q)$ for different p and q such that $p^2 + q \leq 1$. (A simple observation shows that $\mathrm{BPLD}(t, p, q) \subseteq \mathrm{BPLD}(t, p^k, 1-(1-q)^k)$, for every integer k. Indeed, given an algorithm with "yes" and "no" success probabilities p and q, one can modify the success probabilities by performing k runs and requiring each node to individually output "no" if it decided "no" on at least one of the runs. In this case, the "no" success probability increases from q to at least $1 - (1 - q)^k$, and the "yes" success probability then decreases from p to p^k.) The reader is referred to [23] for some recent advances regarding these questions. Another interesting question is whether the phenomenon we observed regarding randomization occurs also in the nondeterministic setting, that is, whether (even for a restricted family of languages) $\mathrm{BPNLD}(t, p, q)$ collapses into $\mathrm{NLD}(O(t))$, for $p^2 + q > 1$.

References

[1] Y. Afek, S. Kutten, and M. Yung. The local detection paradigm and its applications to self stabilization. *Theoretical Computer Science*, 186(1-2):199–230, 1997.

[2] N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. Algorithms*, 7(4):567–583, 1986.

[3] A. Amit, N. Linial, J. Matousek, and E. Rozenman. Random lifts of graphs. In *Proc. 12th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 883–894, 2001.

[4] H. Attiya, A. Bar-Noy, and D. Dolev. Sharing Memory Robustly in Message-Passing Systems. *J. ACM* 42(1):124-142, 1995.

[5] B. Awerbuch, B. Patt-Shamir, and G. Varghese. Self-Stabilization By Local Checking and Correction. In *Proc. IEEE Symp. on the Foundations of Computer Science (FOCS)*, 1991, 268-277.

[6] B. Awerbuch, B. Patt-Shamir, G. Varghese, and S. Dolev. Self-Stabilization by Local Checking and Global Reset. In *Proc. Workshop on Distributed Algorithms*, LNCS 857, Springer, 1994, 326-339.

[7] L. Barenboim and M. Elkin. Distributed $(\Delta + 1)$-coloring in linear (in delta) time. In *Proc. 41st ACM Symp. on Theory of computing (STOC)*, 111–120, 2009.

[8] L. Barenboim, M. Elkin, S. Pettie, and J. Schneider. The Locality of Distributed Symmetry Breaking. In *Proc 53rd IEEE Symp. on Foundations of Computer Science* (FOCS), 321–330, 2012.

[9] T. D. Chandra and S. Toueg. Unreliable Failure Detectors for Reliable Distributed Systems. *J. ACM*, 43(2):225-267, 1996.

[10] T. D. Chandra, V. Hadzilacos and S. Toueg. The Weakest Failure Detector for Solving Consensus. *J. ACM*, 43(4):685-722, 1996.

[11] A. Das Sarma, S. Holzer, L. Kor, A. Korman, D. Nanongkai, G. Pandurangan, D. Peleg and R. Wattenhofer. Distributed Verification and Hardness of Distributed Approximation. In *Proc. 43rd ACM Symp. on Theory of Computing (STOC)*, 2011.

[12] B. Derbel, C. Gavoille, D. Peleg and L. Viennot. Local computation of nearly additive spanners. In *Proc. 23rd Symp. on Distributed Computing (DISC)*, 176–190, (2009).

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

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

[15] M. Elkin. A faster distributed protocol for constructing a minimum spanning tree. *J. Comput. Syst. Sci.* 72(8): 1282-1308 (2006).

[16] M. Elkin and J. Zhang. Efficient algorithms for constructing $(1+\epsilon,\beta)$-spanners in the distributed and streaming models. *Distributed Computing*, 18(5), 375–385 (2006).

[17] M. J. Fischer, N. A. Lynch and M. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *J. ACM*, 32(2): 374–382, 1985.

[18] P. Fraigniaud, D Ilcinkas, and A. Pelc. Communication algorithms with advice. *J. Comput. Syst. Sci.*, 76(3-4):222–232, 2008.

[19] P. Fraigniaud, C. Gavoille, D. Ilcinkas and A. Pelc. Distributed Computing with Advice: Information Sensitivity of Graph Coloring. In *Proc. 34th Colloq. on Automata, Languages and Programming (ICALP)*, 231-242, 2007.

[20] P. Fraigniaud, M. Göös, A. Korman, and J. Suomela. What can be decided locally without identifiers? In *Proc. 32nd ACM Symp. on Principles of Distributed Computing*, 2013.

[21] P. Fraigniaud, M. Halldórsson, and A. Korman. On the Impact of Identifiers on Local Decision. In *Proc. 16th Int. Conference on Principles of Distributed Systems* (OPODIS), 224–238, 2012.

[22] P. Fraigniaud, A Korman, and E. Lebhar. Local MST computation with short advice. In *Proc. 19th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA)*, 154–160, 2007.

[23] P. Fraigniaud, A. Korman, M. Parter, and D. Peleg. Randomized Distributed Decision. In *Proc 26th Int. Symp. on Distributed Computing* (DISC), Spinger, LNCS 7611, 371–385, 2012.

[24] P. Fraigniaud, A. Korman, and D. Peleg. Local Distributed Decision. In *Proc. 52nd IEEE Symp. on Foundations of Computer Science (FOCS)*, 708-717, 2011.

[25] P. Fraigniaud, S. Rajsbaum, and C. Travers. Locality and Checkability in Wait-free Computing. In *Proc. 25th International Symp. on Distributed Computing (DISC)*, 2011.

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

[27] O. Goldreich (Ed.). Property Testing — Current Research and Surveys. LNCS 6390, Springer, 2010.

[28] O. Goldreich and D. Ron. Property Testing in Bounded Degree Graphs. Algorithmica 32(2): 302-343, 2002.

[29] M. Göös and J. Suomela. Locally checkable proofs. In *Proc. 30th ACM Symp. on Principles of Distributed Computing (PODC)*, 2011.

[30] M. Herlihy. Wait-Free Synchronization. *ACM Trans. Programming Languages and Systems*, 13(1):124–149, 1991.

[31] M. Hanckowiak, M. Karonski, and A. Panconesi. On the Distributed Complexity of Computing Maximal Matchings. *SIAM J. Discrete Math.* 15(1): 41-57 (2001).

[32] L. Kor, A. Korman and D. Peleg. Tight Bounds For Distributed MST Verification. In *Proc. 28th Int. Symp. on Theoretical Aspects of Computer Science (STACS)*, 2011.

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

[34] A. Korman, S. Kutten and T. Masuzawa. Fast and Compact Self-Stabilizing Verification, Computation, and Fault Detection of an MST. In *Proc. 30th ACM Symp. on Principles of Distributed Computing (PODC)*, 2011.

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

[36] A. Korman, J.S. Sereni, and L. Viennot. Toward More Localized Local Algorithms: Removing Assumptions Concerning Global Knowledge. In *Proc. 30th ACM Symp. on Principles of Distributed Computing (PODC)*, pages 49-58, 2011.

[37] F. Kuhn. Weak graph colorings: distributed algorithms and applications. In *Proc. 21st ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, 138–144, 2009.

[38] F. Kuhn, T. Moscibroda, R. Wattenhofer. What cannot be computed locally! In *Proc. ACM Symp. on Principles of Distributed Computing (PODC)*, 2004, 300-309.

[39] S. Kutten and D. Peleg. Fast distributed construction of small k-dominating sets and applications. *J. Algorithms*, 28(1):40–66, 1998.

[40] F. Kuhn, and R. Wattenhofer. On the complexity of distributed graph coloring. In *Proc. 25th ACM Symp. on Principles of Distributed Computing (PODC)*, 7–15, 2006.

[41] C. Lenzen, Y. A. Oswald and R. Wattenhofer. What can be approximated locally?: case study: dominating sets in planar graphs. In *Proc. 20th ACM Symp. on Parallelism in Alg. and Architectures (SPAA)*, 46-54, 2008.

[42] N. Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992.

[43] Z. Lotker, B. Patt-Shamir and A. Rosen. Distributed Approximate Matching. *SIAM J. Comput.* 39(2): 445-460, (2009).

[44] Z. Lotker, B. Patt-Shamir and S. Pettie. Improved distributed approximate matching. In *Proc. 20th ACM Symp. on Parallelism in Alg. and Architectures (SPAA)*, pages 129–136, 2008.

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

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

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

[48] A. Panconesi and A. Srinivasan. On the Complexity of Distributed Network Decomposition. *J. Algorithms* 20(2): 356-374, (1996).

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

[50] D. Peleg and V. Rubinovich. A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction. *SIAM J. Comput.*, 30(5):1427–1442, 2000.

[51] S. Pettie. Distributed algorithms for ultrasparse spanners and linear size skeletons. *Distributed Computing*, 22(3): 147-166 (2010).

[52] J. Schneider and R. Wattenhofer. A new technique for distributed symmetry breaking. In *Proc. 29th ACM Symp. on Principles of Distributed Computing (PODC)*, 257-266, 2010.

[53] D. Seinsche. On a property of the class of n-colorable graphs. *J. Combinatorial Theory*, Ser. B, 16, pages 191–193, 1974.

[54] M. Wattenhofer and R. Wattenhofer. Distributed Weighted Matching. In *Proc. 18th Symp. on Distributed Computing (DISC)*, 335-348, 2004.