Local Distributed Decision

Pierre Fraigniaud
CNRS and University Paris Diderot
Paris, France
pierre.fraigniaud@liafa.jussieu.fr

Amos Korman
CNRS and University Paris Diderot
Paris, France
amos.korman@liafa.jussieu.fr

David Peleg
The Weizmann Institute of Science
Rehovot, Israel
david.peleg@weizmann.ac.il

***Abstract*—A central theme in distributed network algorithms concerns understanding and coping with the issue of *locality*. 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 non-determinism on local decision, and establish some structural results inspired by classical computational complexity theory. Specifically, we show that non-determinism does help, but that this help is limited, as there exist languages that cannot be decided non-deterministically. Perhaps surprisingly, it turns out that it is the combination of randomization with non-determinism that enables to decide *all* languages *in constant time*. Finally, we introduce the notion of local reduction, and establish some completeness results.

***Index Terms*—Local distributed algorithms; local decision; nondeterminism; randomized algorithms; oracles**

I. Introduction

A. Motivation

Distributed computing concerns a collection of processors that collaborate in order to achieve some global task. With time, two main disciplines have evolved in the field. 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). 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 [13] proved that consensus cannot be achieved in the asynchronous model, even in the presence of a single fault, and in the *locality-discipline*, Linial [31] proved that $(\Delta+1)$-coloring cannot be achieved locally (i.e., in a constant number of communication rounds), even in the ring network.

One of the significant achievements of the asynchrony-discipline was its success in establishing unifying theories in the flavor of computational complexity theory. Some central examples of such theories are failure detectors [6], [7] and the wait-free hierarchy (including Herlihy's hierarchy) [19]. In contrast, despite considerable progress, the locality-discipline still suffers from the absence of a solid basis in the form of a fundamental computational complexity theory. Obviously, defining some common cost measures (e.g., time, message, memory, etc.) enables us to compare problems in terms of their relative cost. 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.

Inspired by sequential complexity theory, we focus on *decision problems*, in which one is aiming at deciding whether a given global input instance belongs to some specified language. In the context of distributed computing, each processor must produce a boolean output, and the decision is defined by the conjunction of the processors' outputs, i.e., if the instance belongs to the language, then all processors must output "yes", and otherwise, at least one processor must output "no". Observe that 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, many natural problems can be phrased as decision problems, like "is there a unique leader in the network?" or "is the network planar?". Moreover, decision problems occur naturally when one is aiming at checking the validity of the output of a computational task, such as "is the produced coloring legal?", or "is the constructed subgraph an MST?". Construction tasks such as exact or approximated solutions to problems like coloring, MST, spanner, MIS, maximum matching, etc., received enormous attention in the literature (see, e.g., [5], [28], [29], [31], [34], [32], [33], [40]), yet the corresponding decision problems have hardly been

considered.

The purpose of this paper is to investigate the nature of local decision problems. Decision problems seem to provide a promising approach to building up a distributed computational theory for the locality-discipline. Indeed, as we will show, one can define local reductions in the framework of decision problems, thus enabling the introduction of complexity classes and notions of completeness.

We consider the $\mathcal{LOCAL}$ model [38], 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, let us define LD(t) (for *local decision*) as the class of decision problems that can be solved in t communication rounds in the $\mathcal{LOCAL}$ model. Of special interest is the case where t is constant, but in general we view t as a function of the input, i.e., as a function of the input graph and the individual input of each node. Note that in the $\mathcal{LOCAL}$ model, every decidable decision problem can be solved in n communication rounds, where n denotes the number of nodes in the input graph.

Some decision problems fall trivially in LD$(O(1))$ (e.g., "is the given coloring a legal coloring?", "do the selected nodes form an MIS?", etc.), while some others can easily be shown to be outside 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 LD(t) is unclear, even for $t = O(1)$. To elaborate on this, consider the particular case 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 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 LD$(O(1))$. However, the question of whether a family of graphs can be characterized as above is often non-trivial. For example, characterizing cographs as precisely the graphs with no induced P_4, attributed to Seinsche [42], is not easy, and requires nontrivial usage of modular decomposition.

B. Our contributions

∘ **Impact of randomization:** We first study the impact of randomization on local decision, and the question we focus on is whether randomization helps and to what extent. For $p, q \in (0, 1]$, let us define 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). We first observe that LD$(t) \subset$ BPLD(t, p, q) for p and q such that $p^2 + q \leq 1$. Indeed, for such p and q, there exists a language $\mathcal{L}^* \in$ BPLD$(0, p, q)$, such that $\mathcal{L}^* \notin$ LD(t), for all $t = o(n)$. It turns out that this choice of p and q is not coincidental. Indeed, we show that $\mathcal{L}^* \notin$ BPLD(t, p, q), for every $t = o(n)$, and every p and q such that $p^2 + q > 1$. In fact, our main result is considerably more general: we prove that, restricted to hereditary languages, if $p^2 + q > 1$, then BPLD(t, p, q) actually collapses into LD$(O(t))$, for any function t, yielding BPLD$(t, p, q) =$ LD$(O(t))$. These results suggest that $p^2 + q = 1$ may well be a sharp threshold distinguishing the deterministic case from the randomized one.

∘ **Impact of non-determinism:** In the second part of the paper, we investigate the impact of non-determinism on local decision, and establish some structural results inspired by classical computational complexity theory. We start by establishing that non-determinism does help, but that this help is limited, as there exist languages that cannot be decided non-deterministically. Specifically, to show that non-determinism helps local decision, we prove that the class NLD(t) (the non-deterministic version of LD(t)) strictly contains LD(t). More precisely, we show that there exists a language in NLD$(O(1))$ which is not in LD(t) for every $t = o(n)$. On the other hand, we also show that NLD(t) does not capture all (decidable) languages, for $t = o(n)$. Indeed we prove that there exists a language not in NLD(t) for every $t = o(n)$. Specifically, this language is $\texttt{GraphSize} = \{(G, k)$ 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.

Perhaps surprisingly, it turns out that it is the combination of randomization with non-determinism 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 BPNLD$(t) = \cup_{p^2+q\leq 1}$BPNLD(t, p, q). We prove that BPNLD$(O(1))$ contains all languages. To sum up, we get for every $t = o(n)$, we have

$$\text{LD}(t) \subset \text{NLD}(t) \subset \text{BPNLD}(O(1)) = \text{All}$$

where All is the set of (sequentially decidable) distributed languages. Alternatively, by considering oracles providing global information to the nodes, we also show that NLD$^{\texttt{GraphSize}}(O(1)) =$ All where NLD$^{\texttt{GraphSize}}(O(1))$ is NLD$(O(1))$ assuming that each node can access an oracle that returns the number of nodes in the input graph.

Finally, we introduce the notion of many-one 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 BPNLD$(O(1))$-complete. Interestingly, a small relaxation of `Cover`, called `Containment`, turns out to be NLD$(O(1))$-complete.

C. Related work

Locality issues have been thoroughly studied in the literature, via the analysis of various construction problems, including coloring and MIS [2], [5], [26], [29], [31], [34], [37],

MST [28], [39], matching [20], [32], [33], [43], dominating set [27], [30], spanners [9], [12], [40], etc. For some problems (e.g., coloring [5], [26], [37]), 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 [36]. In particular, that paper considers a subclass of LD($O(1)$), called LCL, which is essentially 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 some beautiful 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 thus not clear whether it is possible to extend this result to all languages in LD($O(1)$).

The question of whether randomization helps in decreasing the locality parameter of 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, [35] proves this for 3-coloring the ring. 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], [34], whereas the best known deterministic algorithm for this problem performs in $2^{O(\sqrt{\log n})}$ rounds [37]. $(\Delta+1)$-coloring results whose performance is expressed also in terms of the maximum degree Δ illustrate this phenomenon as well. Specifically, [41] shows that $(\Delta+1)$-coloring can be randomly computed in expected $O(\log\Delta+\sqrt{\log n})$ communication rounds, whereas the best known deterministic algorithm performs in $O(\Delta+\log^* n)$ rounds [5], [26].

Recently, several results were established concerning decision problems in distributed computing. For example, [8] and [21] 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 [21] for the time and message complexities of the problem of deciding whether a given subgraph is an MST, and time lower bounds for many other subgraph-decision problems (e.g., spanning tree, connectivity) are established in [8]. Decision problems have received recent attention in the asynchrony discipline too, in the framework of wait-free computing [17].

The theory of *proof-labeling schemes* [24], [18], [22], [23] 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?"). Investigations in this framework mostly focus on the minimum size of the certificate necessary so that verification can be performed in a single round [18], [22], [24], or in t rounds [23]. 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 [4], or silent stabilization [11], which were introduced in the context of self-stabilization [10]. 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. For instance, this framework, called *local computation with advice*, was studied in [16] for MST construction, in [15] for 3-coloring a cycle, and in [14] for broadcast and wake up.

II. Decision problems and complexity classes

Model of computation: Let us first recall some basic notions in distributed computing. We consider the $\mathcal{LOCAL}$ model [38], 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, and, 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. At 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. To sum up, 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 a node v, denoted by $T_{\mathcal{A},v}$, is the number of communication rounds until v outputs. 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 t be a function of the triples $(G,\mathbf{x},\mathrm{Id})$, 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, 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 $t = t(G,\mathbf{x},\mathrm{Id})$ is happening to be known by every node, then, w.l.o.g., one can assume that an algorithm running in time at most t operates at each node v in two stages: (a) collect all information available in the t-neighborhood of v (i.e., the ball $B_G(v,t)$ of radius t around v in G), including input values, identities, and network structure; (b) compute the output locally at v based on this information. 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 at all nodes.

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 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 standard distributed computing problems. A natural type of decision problems involves getting as input an $\mathbf{x}$ claimed to be the output of some common distributed computing problem Π, and deciding whether it is indeed a legal output for Π. Some examples for problems are given below.

- $\texttt{Consensus} = \{(G,(\mathbf{x}_1,\mathbf{x}_2))$ s.t. $\exists u \in V(G), \forall v \in V(G), \mathbf{x}_2(v) = \mathbf{x}_1(u)\}$ consists of all instances such that all nodes agree on a value proposed by one of them.
- $k-\texttt{Coloring} = \{(G,\mathbf{x})$ 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.
- $\texttt{MIS} = \{(G,\mathbf{x})$ s.t. $S = \{v \in V(G) \mid \mathbf{x}(v) = 1\}$ forms a maximal independent set$\}$.
- $\texttt{Tree} = \{(G,\epsilon);\ G$ is a tree$\}$;
- $\texttt{Planar} = \{(G,\epsilon);\ G$ is a planar graph$\}$;

Let $\mathcal{L}$ be a distributed language. We say that a distributed algorithm $\mathcal{A}$ *decides* $\mathcal{L}$ iff 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, $\mathrm{out}_{\mathcal{A}}(G,\mathbf{x},\mathrm{Id},v) =$ "yes" for every node $v \in V(G)$;
- If $(G,\mathbf{x}) \notin \mathcal{L}$, then for every identity assignment Id, $\mathrm{out}_{\mathcal{A}}(G,\mathbf{x},\mathrm{Id},v) =$"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 $\mathrm{LD}(t)$, for *local decision*.

Definition 2.3: Let t be a function of triplets $(G,\mathbf{x},\mathrm{Id})$. $\mathrm{LD}(t)$ is the class of all distributed languages that can be decided by a distributed algorithm that runs in at most t communication rounds.

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

In what follows, for every function t, we define $\mathrm{LD}(O(t)) = \bigcup_{c>0} \mathrm{LD}(c\cdot t)$. Hence, for a distributed language $\mathcal{L}$ and a function t, $\mathcal{L} \in \mathrm{LD}(O(t))$ if and only if there exists a constant c such that $\mathcal{L} \in \mathrm{LD}(c\cdot t)$.

Non-deterministic local decision (NLD**):** A distributed *verification* algorithm is a distributed algorithm $\mathcal{A}$ that gets as input, in addition to an instance $(G,\mathbf{x})$, a global *certificate vector* $\mathbf{y}$, i.e., every node v of a graph G gets as 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 hold:

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

One motivation for studying the non-determinism in the above sense comes from settings in which one must perform local verifications repeatedly. 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 [22], [24] for more details regarding such applications). Indeed, the definition of a verification algorithm finds similarities with the notion of *proof-labeling schemes* discussed therein. Informally, 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 id-assignment. Since the question of whether an instance $(G,\mathbf{x})$ belongs to a language $\mathcal{L}$ is independent from the particular id-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 id-assignment. We now define the class $\mathrm{NLD}(t)$, for *nondeterministic local decision*. (Our terminology is by direct analogy to the class NP in

sequential computational complexity.)

Definition 2.4: Let t be a function of triplets $(G, \mathbf{x}, \mathrm{Id})$. NLD$(t)$ is the class of all distributed languages that can be verified in at most t communication rounds.

Bounded-error probabilistic local decision (BPLD)**:** A *randomized* distributed algorithm is a distributed algorithm $\mathcal{A}$ that enables every node v, at any round r during the execution, to toss a certain number of random bits. More specifically, in this paper, randomized computation is tackled by considering Monte Carlo algorithms. Recall that, in sequential computing, a Monte Carlo algorithm is a randomized algorithm whose running time is deterministic, but whose output may be incorrect with a certain probability. We just extend this concept to the distributed setting, by focussing on distributed algorithms that use randomization but whose running time 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, under 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 deterministic (i.e., it depends only of the actual instance $(G, \mathbf{x})$ and id-assignment Id).

For $p, q \in (0, 1]$, 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 id-assignment Id, $\mathtt{Pr}[\forall v \in V(G), \mathrm{out}_{\mathcal{A}}(G, \mathbf{x}, \mathrm{Id}, v) = \text{"yes"}\ ] \geq p$,
- If $(G, \mathbf{x}) \notin \mathcal{L}$ then, for every id-assignment Id, $\mathtt{Pr}[\exists v \in V(G), \mathrm{out}_{\mathcal{A}}(G, \mathbf{x}, \mathrm{Id}, v) = \text{"no"}] \geq q$,

where the probabilities in the above definition are taken over all possible coin tosses performed by the nodes. The running time of v performing a (p, q)-decider depends on the triple $(G, \mathbf{x}, \mathrm{Id})$ and on the results of the coin tosses. In the context of randomized algorithm, $T_v(G, \mathbf{x}, \mathrm{Id})$ denotes the maximal running time of v over all possible coin tosses, for instance $(G, \mathbf{x})$ and id-assignment Id. Then, as for non-probabilistic cases, the running time T of the (p, q)-decider is the maximum of the running time of the nodes. Again, by definition of the distributed Monte-Carlo algorithm, both T_v and T 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 function t of triplets $(G, \mathbf{x}, \mathrm{Id})$, BPLD$(t, p, q)$ is the class of all distributed languages that have a randomized distributed (p, q)-decider running in time at most t (i.e., can be decided in time at most t by a randomized distributed algorithm with "yes" success probability p and "no" success probability q).

III. A SHARP THRESHOLD FOR RANDOMIZATION

The objective of this section is to tackle the question of whether randomization helps (local) distributed computing, and to which extent. Recall that [36] 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 [36] for tackling this question relies heavily on the definition of LCL, specifically, that only graphs of constant degree and of constant input size are considered. Hence it is not clear whether the technique of [36] 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 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 by a simple observation. Consider the following language.

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

By considering the n-node path, one can easily check that this language is not in 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 algorithm that runs in time zero: every unselected node says "yes" with probability 1, and every selected node says "yes" with probability p. 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 get 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 $t = o(n)$.

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. Consider some graph G, and a subset U of the nodes of G, i.e., $U \subseteq V(G)$. 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, in particular, $G[U]$ is connected). We say that 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 an hereditary language, consider a family $\mathcal{G}$ of hereditary graphs, i.e., that is 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 3.3 below asserts that, for hereditary languages, randomization does not help if one imposes that $p^2 + q > 1$, i.e, the "no" success probability is larger than one minus the square of the "yes" success probability.

Theorem 3.3: Let $\mathcal{L}$ be an hereditary language and let t

be a function of triples $(G, \mathbf{x}, \mathrm{Id})$. If $\mathcal{L} \in \mathrm{BPLD}(t, p, q)$ for constants $p, q \in (0, 1]$ such that $p^2 + q > 1$, then $\mathcal{L} \in \mathrm{LD}(O(t))$.

Proof: Let us first give an informal sketch. The proof of the theorem consists in 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 $B_G(v)$ of radius $O(t)$ around v, and outputs "yes" at v if and only if $B_G(v) \in \mathcal{L}$. (Some care is needed here, since v might not know the time bound t, and therefore, collecting information from a ball of radius $O(t)$ 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 sub-instance of $(G, \mathbf{x})$ also belongs to $\mathcal{L}$, 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 of radius $O(t)$ around some node v, such that the sub-instance 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 this goal, we first establish Lemma 3.4, which informally states that the union of two legal instances is also legal if 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 3.4 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 by contradiction that under $\mathcal{D}$ each node outputs "yes". We then consider the largest sub-instance U of $(G, \mathbf{x})$ that is legal. This U is not empty since each node v outputs "yes" and hence we have $B_G(v) \in \mathcal{L}$. On the other hand, U is not the whole graph, 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 3.4 and deduce that $U \cup B_G(v) \in \mathcal{L}$. Instead, to deduce that $U \cup B_G(v) \in \mathcal{L}$, we use a more refined argument that requires repeated use of Lemma 3.4 on different connected sub-instances of $U \cup B_G(v)$. Finally, to obtain the contradiction we also make sure that v is chosen close enough to the border of U, so that $U \cup B_G(v)$ strictly contains U, thus contradicting the maximality of U.

We now turn to describe the formal proof. Let us start with some definitions. Let $\mathcal{L}$ be a language in $\mathrm{BPLD}(t, p, q)$ where $p, q \in (0, 1]$, $p^2 + q > 1$, and t be some function of triples $(G, \mathbf{x}, \mathrm{Id})$. Let $\mathcal{A}$ be a randomized algorithm deciding $\mathcal{L}$, with "yes" success probability p and "no" success probability q, whose running time is at most $t(G, \mathbf{x}, \mathrm{Id})$, 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 v executing $\mathcal{A}$, and that $T = T(G, \mathbf{x}, \mathrm{Id})$ denotes the maximum of T_v over all nodes of G. Note that $T_v \leq T \leq t = t(G, \mathbf{x}, \mathrm{Id})$, but we do not assume that any of these values is initially known to v. The *radius* of a node v, denoted r_v, is the maximum value of T_u 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 be possibly "see" during the execution, 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 $T_v \leq r_v \leq t$. 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 $\mathrm{dist}_G(U_1, U_2) \geq \lambda \cdot r_S$. (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 3.4: 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: $\Big((G_1, \mathbf{x}_1) \in \mathcal{L} \text{ and } (G_2, \mathbf{x}_2) \in \mathcal{L}\Big) \Rightarrow (G, \mathbf{x}) \in \mathcal{L}$.

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 $u \in S$, we define the *level* of u by $\ell(u) = \mathrm{dist}_G(U_1, \{u\})$. For an integer $i \in [1, \lambda r_S]$, let L_i denote the set of nodes in S of level i. For an integer $i \in (r_S, (\lambda - 1) r_S)$, 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 $S_I = \bigcup_{i \in I} S_i$. 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 id-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 at

least δ. Formally,

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

Claim 3.5: There exists $i \in \mathcal{R}$ 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 3.5, 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 $(G_1, \mathbf{x}_1) \in \mathcal{L}$, we know that $\Pr[\mathcal{E}(G_1, \mathbf{x}_1, \mathrm{Id}, S')] \geq p$ 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

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

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

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

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 $\Pr[\mathcal{E}(G, \mathbf{x}, \mathrm{Id}, S_{\mathcal{I}\cap J})] \;=\; \prod_{i\in\mathcal{I}\cap J} \Pr[\mathcal{E}(G, \mathbf{x}, \mathrm{Id}, S_i)] \;<\; (1-\delta)^{|\mathcal{I}\cap J|}$, where the inequality follows from the definition of $\mathcal{I}$. By (1), 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 3.5. ∎

Applying Claim 3.5, 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,

$$\Pr[\overline{\mathcal{F}}] \leq \delta < p^2 + q - 1. \quad (2)$$

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 $w \in S$.

For $k = 1, 2$, we have $(G_k, \mathbf{x}_k) \in \mathcal{L}$, and hence $\Pr[\mathcal{E}(G, \mathbf{x}, \mathrm{Id}, V(H_k))] = \Pr[\mathcal{E}(G_k, \mathbf{x}_k, \mathrm{Id}, V(H_k))] \geq p$. 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 $\Pr[\mathcal{F}'] \geq p^2$, that is $\Pr[\overline{\mathcal{F}'}] \leq 1 - p^2$. Hence, combining this equation with Eqs. (2), and using union bound, it follows that $\Pr[\overline{\mathcal{F}} \vee \overline{\mathcal{F}'}] < q$. Thus, $\Pr[\mathcal{E}(G, \mathbf{x}, \mathrm{Id}, V(G))] = \Pr[\mathcal{E}(G, \mathbf{x}, \mathrm{Id}, S_i \cup V(H_1) \cup V(H_2))] = \Pr[\mathcal{F} \wedge \mathcal{F}'] > 1-q$, contradicting the assumption that $(G, \mathbf{x}) \notin \mathcal{L}$. This establishes Lemma 3.4. ∎

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, whose proof is deferred to the full paper.

Claim 3.6: Fix a constant $c > 0$, and let $(G, \mathbf{x})$ be an instance with an id-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 $u \in B_G(v, c \cdot T_v^*)$.

Given an instance $(G, \mathbf{x})$ and an id-assignment Id, the deterministic Algorithm $\mathcal{D}$, applied at a node u, first calculates T_u^* as in Claim 3.6, 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, $\mathrm{out}(u) = \text{"yes"} \iff (B_G(u, 2\lambda T_u^*), \mathbf{x}[B_G(u, 2\lambda T_u^*)]) \in \mathcal{L}$.

Algorithm $\mathcal{D}$ is an $O(t)$-time deterministic algorithm. (Recall that $\mathcal{L}$ is (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 done 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 "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 id-assignment Id, every node u outputs "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 1 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.

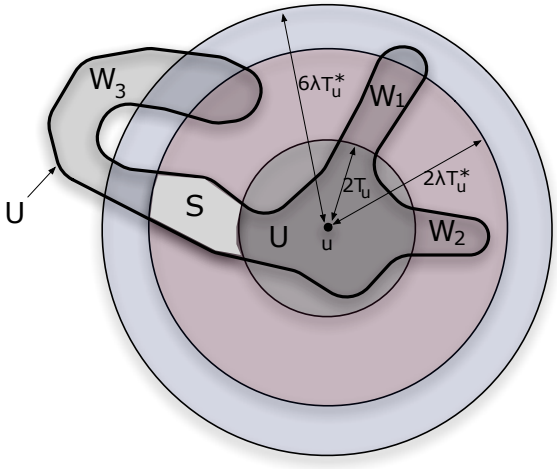

Fig. 1. Illustration of the several node sets used in the proof of Theorem 3.3.

Let H denote the maximal (under inclusion) graph such that H is connected and $B_G(u, 2T_u) \subset V(H) \subseteq B_G(u, 2T_u) \cup (U \cap B_G(u, 2\lambda T_u^*))$. 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 $S = V(Z^k) \cap V(W^{k+1})$, $U_1 = V(Z^k) \setminus S$, and $U_2 = V(W^{k+1}) \setminus S$.

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 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 $\text{dist}_{Z^{k+1}}(U_1, U_2) \geq \lambda T_u^*$. 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, $\text{dist}_{Z^{k+1}}(U_1, U_2) \geq \lambda T_u^* \geq \lambda r_S$, 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 3.4 and conclude that $(Z^{k+1}, \mathbf{x}[Z^{k+1}]) \in \mathcal{L}$. This concludes the induction proof. The theorem follows. ■

IV. Nondeterminism and complete problems

We first establish two simple separation results. (The proofs use rather standard arguments, and hence, are deferred to the full paper.) Our first separation result indicates that nondeterminism helps for local decision. Indeed, we show that there exists a language that belongs to NLD(1) but not to 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 non-deterministically 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: LD$(t) \subset$ NLD(t), for any $t = o(n)$.

Theorem 4.2: There exists a language $\mathcal{L}$ such that $\mathcal{L} \notin$ NLD(t), for any $t = o(n)$.

For $p, q \in (0, 1]$ and a function t, let us define BPNLD(t, p, q) as the class of all distributed languages that

have a local randomized non-deterministic distributed (p,q)-decider running in time t. We now claim 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 $p^2+q \leq 1$. For every language $\mathcal{L}$, we have $\mathcal{L} \in \text{BPNLD}(1,p,q)$.

Proof: Given an arbitrary language $\mathcal{L}$, let us describe a constant time non-deterministic (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 non-deterministic distributed (p,q)-decider for verifying that the instance $(G',\mathbf{x}')$ is identical to $(G,\mathbf{x})$, and running in time 1.

The non-deterministic (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 wether $\mathbf{x}'[\lambda(v)] = \mathbf{x}(v)$, 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 non-deterministic 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[1]. 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 $\lambda(v) = 1$. Consequently, if $(G',\mathbf{x}') = (G,\mathbf{x})$ then all nodes say "yes" with probability at least p. On the other hand, if $(G',\mathbf{x}') \neq (G,\mathbf{x})$ then there are at least two nodes in G whose label is "1". These two nodes say "yes" with probability p^2, hence, the probability that at least one of them says "no" is at least $1-p^2 \geq q$. 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 $\text{BPNLD} = \text{BPNLD}(1,p,q)$. The next corollary follows from Theorems 4.1, 4.2 and 4.3.

Corollary 4.4: For every $t = o(n)$, we have $\text{LD}(t) \subset \text{NLD}(t) \subset \text{BPNLD} = \text{All}$.

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., [25], [31], [32], [37]), 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 $\text{NLD}^{\texttt{GraphSize}}$ 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`). The proof appears in the full paper.

Theorem 4.5: For every language $\mathcal{L}$, we have $\mathcal{L} \in \text{NLD}^{\texttt{GraphSize}}$. I.e., $\text{NLD}^{\texttt{GraphSize}} = \text{BPNLD} = \text{All}$.

We now claim that there exists a natural problem, called `Cover`, which is in some sense the "most difficult" decision problem. Let us first define a notion of reduction that fits with the class LD. 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 id-assignment Id, $\mathcal{A}$ produces $\text{out}(v) \in \{0,1\}^*$ as output at every node $v \in V(G)$ so that $(G,\mathbf{x}) \in \mathcal{L}_1 \iff (G,\text{out}) \in \mathcal{L}_2$. By definition, $\text{LD}(O(t))$ 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 $\mathcal{L}_2 \in \text{LD}(O(t))$ then $\mathcal{L}_1 \in \text{LD}(O(t))$.

We now define the language `Cover` and claim that it is BPNLD-complete. 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, $\texttt{Cover} = \{(G,(\mathcal{E},\mathcal{S})) \mid \exists v \in V(G),\ \exists S \in \mathcal{S}(v) \text{ s.t. } S = \{\mathcal{E}(u) \mid u \in V(G)\}\}$. Due to the space limitation, we defer the proof of the following theorem to the full paper.

Theorem 4.6: `Cover` is BPNLD-complete.

Finally, finding an NLD-complete problem was not an easy task. Eventually, we managed to find a natural problem, called `Containment`, which is $\text{NLD}(O(1))$-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

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

the nodes. Formally, define $\texttt{Containment} = \{(G,(\mathcal{E},\mathcal{S})) \mid \exists v \in V(G),\ \exists S \in \mathcal{S}(v) \text{ s.t. } S \supseteq \{\mathcal{E}(u) \mid u \in V(G)\}\}$. The proof of the theorem below is deferred to the full paper.

Theorem 4.7: Containment is NLD$(O(1))$-complete.

V. Future work

This paper aims to make a first step in the direction of establishing a complexity theory for the locality discipline. Our model of computation, namely, the $\mathcal{LOCAL}$ model, focuses on difficulties arising from purely locality issues, and abstracts away other complexity measures. Naturally, it would be very interesting to come up with a rigorous complexity framework taking into account also other complexity measures. For example, it would be interesting to investigate the connections between classical computational complexity theory and the local complexity one. Bounds on the (individual) running time in each round and/or the memory used by a node may serve as bridges for connecting the two theories. 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).

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

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

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

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

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

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

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

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

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

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

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

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

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

[17] P. Fraigniaud, S. Rajsbaum, and C. Travers. Wait-free checkable tasks. Proc. 25th Int. Symp. DIStributed Computing (DISC), 2011.

[18] M. Göös and J. Suomela. Locally checkable proofs. Proc. 30th Symp. on Principles of Dist. Computing (PODC), 2011.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[39] 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.

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

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

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

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