On Polynomial Time Local Decision

Eden Aldema Tshuva
Tel Aviv University, Israel

Rotem Oshman
Tel Aviv University, Israel

Abstract

The field of distributed local decision studies the power of local network algorithms, where each network can see only its own local neighborhood, and must act based on this restricted information. Traditionally, the nodes of the network are assumed to have unbounded local computation power, and this makes the model incomparable with centralized notions of efficiency, namely, the classes P and NP. In this work we seek to bridge this gap by studying local algorithms where the nodes are required to be computationally efficient: we introduce the classes PLD and NPLD of polynomial-time local decision and non-deterministic polynomial-time local decision, respectively, and compare them to the centralized complexity classes P and NP, and to the distributed classes LD and NLD, which correspond to local deterministic and non-deterministic decision, respectively.

We show that for deterministic algorithms, requiring both computational and distributed efficiency is likely to be more restrictive than either requirement alone: if the nodes do not know the network size, then $\mathsf{PLD} \subsetneq \mathsf{LD} \cap \mathsf{P}$ holds unconditionally; if the network size is known to all nodes, then the same separation holds under a widely believed complexity assumption ($\mathsf{UP} \cap \mathsf{coUP} \neq \mathsf{P}$). However, when nondeterminism is introduced, this distinction vanishes, and $\mathsf{NPLD} = \mathsf{NLD} \cap \mathsf{NP}$. To complete the picture, we extend the classes PLD and NPLD into a hierarchy akin to the centralized polynomial hierarchy, and we characterize its connections to the centralized polynomial hierarchy and to the distributed local decision hierarchy of Balliu, D'Angelo, Fraigniaud, and Olivetti.

2012 ACM Subject Classification Theory of computation → Complexity classes; Theory of computation → Problems, reductions and completeness

Keywords and phrases Local Decision, Polynomial-Time, LD, NLD

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2023.27

Funding *Rotem Oshman*: Research funded by the Israel Science Foundation, Grant No. 2801/20, and also supported by Len Blavatnik and the Blavatnik Family foundation.

1 Introduction

The field of distributed local decision studies the computation power of distributed network algorithms, where every node can observe only its own neighborhood in the network. There is a rich body of literature characterizing the types of network properties that such algorithms can decide, including deterministic algorithms (e.g., in [23, 7, 6, 8]), randomized algorithms (e.g., in [5, 4]), nondeterministic algorithms (e.g., in [21, 12, 19, 20, 9]), and other variants. However, to our knowledge, all prior work on distributed decision allows the network nodes to have unbounded computation power – they can locally run arbitrary Turing machines (e.g., in [6, 9]), or sometimes even compute any function, even if it is undecidable (e.g., [23]). This puts the theory of efficient distributed decision on different footing from centralized notions of efficiency such as P and NP, and makes the notion of an efficient local distributed algorithm incomparable with that of an efficient centralized algorithm: some problems that are considered hard for a single machine to solve are considered "easy" for a network, simply because we do not take into consideration the computational power of the network nodes. In this paper we introduce a computationally-bounded version of local decision, and study the effects of imposing both locality restrictions and computational efficiency requirements at the same time.


© Eden Aldema Tshuva and Rotem Oshman;
licensed under Creative Commons License CC-BY 4.0

Deterministic computationally bounded local decision. The class LD (for "local decision"), introduced in [9], encompasses all network properties that can be decided by a deterministic constant-round distributed algorithm (see Section 2 for the formal definition). As explained above, this includes algorithms that are not computationally efficient, e.g., algorithms where a network node is required to perform exponential-time computations. Consequently, the class LD is incomparable with classes representing efficient *sequential* computation: it is not contained in, nor does it contain, any class $\mathsf{DTIME}(f(n))$ or $\mathsf{NTIME}(f(n))$ for any time-constructible function $f : \mathbb{N} \to \mathbb{N}$. For instance, there are problems that are known to be NP-hard even in constant-diameter graphs,[1] and these problems are in LD – that is, considered "easy" for local distributed algorithms – even though they would be considered "hard" by mainstream complexity theory.

To study the importance of local computation at the network nodes, we define the class $\mathsf{PLD} \subseteq \mathsf{LD}$, which includes all network properties that can be decided by a deterministic constant-round distributed algorithm that *uses local computation time* $\mathrm{poly}(n)$ *at every node*. The "plain" version of the model does not assume that the size n of the network is known to the nodes, but we also consider a stronger model, $\mathsf{PLD}^{[n]}$, where the nodes do know the network size. We ask:

What is the power of algorithms in PLD / $\mathsf{PLD}^{[n]}$*?*

Clearly, algorithms that are both local and computationally efficient cannot decide languages that are not in LD, nor can they decide languages that are not in P. But can they decide *every language that is both in* LD *and in* P? It turns out that the answer depends on whether or not the network size is known:

- In the "plain" version of the model, where the network size is not known to the nodes, we can prove unconditionally that the answer is *no*: $\mathsf{PLD} \subsetneq \mathsf{LD} \cap \mathsf{P}$.
- In the version where the size is known, we show that the corresponding separation, $\mathsf{PLD}^{[n]} \neq \mathsf{LD}^{[n]} \cap \mathsf{P}$, implies that $\mathsf{P} \neq \mathsf{NP}$. This makes it unlikely that we can prove such a separation unconditionally.
- However, we are able to prove a *conditional* separation: under the assumption that $\mathsf{UP} \cap \mathsf{coUP} \neq \mathsf{P}$, we can show that $\mathsf{PLD}^{[n]} \neq \mathsf{LD}^{[n]} \cap \mathsf{P}$.

The class UP includes all languages that can be decided by a nondeterministic Turing machine with at most one accepting computation path on every input, and the class coUP includes all langauges whose complements are in UP. The intersection $\mathsf{UP} \cap \mathsf{coUP}$ includes the decision version of the problem of integer factorization, which is believed not to be in P, and whose conjectured hardness serves as the basis for RSA public-key encryption.

Our conditional proof that $\mathsf{PLD}^{[n]} \neq \mathsf{LD}^{[n]} \cap \mathsf{P}$ relies on a connection to worst-case cryptography. It is well known that $\mathsf{UP} \cap \mathsf{coUP} \neq \mathsf{P}$ implies the existence of *worst-case one-way functions* [18, 13, 15, 16], that is, functions that can be computed in polynomial time, but cannot be inverted in polynomial time (in the worst case sense: there does not exist a polynomial-time algorithm that correctly computes the inverse of the function on all inputs). Our proof can be viewed as implicitly constructing such a worst-case one-way function from the hardness of $\mathsf{UP} \cap \mathsf{coUP}$, following a construction similar to the one used in [18, 13], and then applying a simple worst-case version of the Goldreich-Levin Theorem [11] to the function we constructed.

Our results for deterministic algorithms are summarized by the following theorem.

[1] For example, the Forwarding Index Problem [25], which asks whether every pair of vertices in the graph can be connected by a path, such that every vertex appears on at most k paths.

▶ **Theorem 1.** *The following holds:*
1. $\mathsf{PLD} \subsetneq \mathsf{P} \cap \mathsf{LD}$.
2. *If* $\mathsf{PLD}^{[n]} \neq \mathsf{P} \cap \mathsf{LD}^{[n]}$, *then* $\mathsf{P} \neq \mathsf{NP}$.
3. *Assuming that* $\mathsf{UP} \cap \mathsf{coUP} \neq \mathsf{P}$, *we have* $\mathsf{PLD}^{[n]} \subsetneq \mathsf{P} \cap \mathsf{LD}^{[n]}$.

Nondeterministic computationally bounded local decision and beyond. In [9], the class NLD (for "nondeterministic local decision") is introduced to capture network properties that can be decided by a nondeterministic constant-round distributed algorithm: this is an algorithm where every node is given a *certificate* (or *witness*). The class NLD can be viewed as a distributed analog of NP. It is of particular importance because of its connections to self-stabilization and fault-tolerance: certificates are used in *proof labeling schemes* [22] to help identify illegal network configurations that require attention.

Just as we did with deterministic algorithms, we introduce a computationally-bounded version of NLD, which we call NPLD, and which captures network properties that can be decided nondeterministically by a constant-round algorithm where nodes run in polynomial local time. We again ask whether the restriction to both distributed and computational efficiency is more restrictive than either restriction alone: is $\mathsf{NPLD} = \mathsf{NLD} \cap \mathsf{NP}$? Perhaps surprisingly, the answer this time is that the combination is *not* more restrictive than either restriction alone:

▶ **Theorem 2.** $\mathsf{NPLD} = \mathsf{NP} \cap \mathsf{NLD}$, *and* $\mathsf{NPLD}^{[n]} = \mathsf{NP} \cap \mathsf{NLD}^{[n]}$.

Here, $\mathsf{NPLD}^{[n]}, \mathsf{NLD}^{[n]}$ are versions of NPLD and NLD (resp.) where the size of the network is known to all nodes. We note that it was shown in [9] that $\mathsf{NLD}^{[n]}$ contains all Turing-decidable languages, which implies that $\mathsf{NP} \cap \mathsf{NLD}^{[n]} = \mathsf{NP}$.

In sequential complexity theory, P and NP are the lowest levels of the *polynomial hierarchy*, $\mathsf{PH} = \{\Sigma_k, \Pi_k\}_{k=0}^{\infty}$, which extends the notion of nondeterminism ("$x \in \mathcal{L}$ iff *there exists* a witness that causes us to accept") and co-nondeterminism ("$x \in \mathcal{L}$ iff *every* witness causes us to accept") into a hierarchy of complexity classes, with an increasing number of quantifier alternations on the witness. At the k-th level of the hierarchy, the class Σ_k allows k quantifier alternations starting with $\exists$, and the class Π_k allows k quantifier alternations starting with $\forall$. The polynomial hierarchy has been the subject of intense research, and one of the most important open problems in complexity theory is whether the inclusions between the levels of the polynomial hierarchy are strict or not. It is commonly believed that they are [10], and "the polynomial hierarchy does not collapse to level k" is a fairly standard hardness assumption, like $\mathsf{P} \neq \mathsf{NP}$.

Inspired by the polynomial hierarchy, in [2], the classes LD and NLD were extended into a full hierarchy of local decision, $\left\{\Sigma_k^{local}, \Pi_k^{local}\right\}_{k=0}^{\infty}$, where the classes in the k-th level of the hierarchy allow k quantifier alternations on the certificates, starting with $\exists$ for Σ and with $\forall$ for Π. The authors of [2] were able to fully characterize the power of each level of the hierarchy: they showed that

$$\mathsf{LD} \subsetneq \Pi_1^{local} \subsetneq \mathsf{NLD} = \Sigma_1^{local} = \Sigma_2^{local} \subsetneq \Pi_2^{local} = \mathsf{ALL}.$$

(The class ALL is defined to be all Turing-decidable languages, and all classes $\Sigma_k^{local}, \Pi_k^{local}$ are restricted to Turing-decidable languages as well.)

What happens to the local decision hierarchy when nodes are restricted to run in polynomial time? We already noted that at the first level, nondeterministic local decision (the class $\Sigma_1^{local} = \Sigma_2^{local} = \mathsf{NLD}$) is unaffected by this additional restriction. We show that the same holds for all levels above as well (Σ_k^{local} for $k \geq 3$, and Π_k^{local} for $k \geq 2$).

Our results for the local decision hierarchy (other than NLD, which was discussed above) are summarized in the following theorems. Here, $\Sigma_k^{\mathsf{P}\text{-}local}$ (resp. $\Pi_k^{\mathsf{P}\text{-}local}$) denotes the class of network properties that can be decided by a Σ_k^{local}-algorithm (resp. Π_k^{local}) where every node runs in poly-time (see Section 2 for the formal definition).

▶ **Theorem 3.** *The following holds:*
1. $\Sigma_2^{\mathsf{P}\text{-}local} = \Sigma_2^{\mathsf{P}} \cap \Sigma_2^{local} \subsetneq \Sigma_2^{\mathsf{P}}$.
2. *For every* $k \geq 2$ *we have* $\Pi_k^{\mathsf{P}\text{-}local} = \Pi_k^{\mathsf{P}}$*, and for every* $k \geq 3$ *we have* $\Sigma_k^{\mathsf{P}\text{-}local} = \Sigma_k^{\mathsf{P}}$.

The relationships between the different levels of the hierarchy $\left\{\Sigma_k^{\mathsf{P}\text{-}local}, \Pi_k^{\mathsf{P}\text{-}local}\right\}_{k=0}^{\infty}$ are as follows:

▶ **Theorem 4.** *The following holds:*
1. $\mathsf{PLD} \subsetneq \Pi_1^{\mathsf{P}\text{-}local} \cap \mathsf{NPLD}$.
2. $\mathsf{NPLD} \not\subseteq \Pi_1^{\mathsf{P}\text{-}local}$.
3. $\mathsf{NPLD} \subsetneq \Pi_2^{\mathsf{P}\text{-}local}$.
4. $\Pi_1^{\mathsf{P}\text{-}local} \subsetneq \Sigma_2^{\mathsf{P}\text{-}local} \cap \Pi_2^{\mathsf{P}\text{-}local}$.
5. *For every* $k \geq 0$*, if either* $\Sigma_{k+1}^{\mathsf{P}\text{-}local}$ *or* $\Pi_{k+1}^{\mathsf{P}\text{-}local}$ *equals* $\Pi_k^{\mathsf{P}\text{-}local}$ *or* $\Sigma_2^{\mathsf{P}\text{-}local}$*, or if* $\Sigma_{k+1}^{\mathsf{P}\text{-}local} = \Pi_{k+1}^{\mathsf{P}\text{-}local}$*, then* $\mathsf{PH} = \Pi_k^{local}$.

A partial and preliminary version of this work appeared as a brief announcement in [1]. While our model allows nodes to run in time that is polynomial in the size of the network, a follow-up work [24] considered a model where the runtime of each node must be polynomial *in the size of its neighborhood*. This yields a different model, where nodes with many neighbors are allowed to use more local computation than nodes with few neighbors.

Organization. For lack of space, many proofs are omitted here; we focus on proving Theorems 1 and 2, in Sections 3 and 4 respectively, omitting some minor technical details. In Section 5 we define the polynomial-time local hierarchy, and sketch the proof of Theorem 3. The remaining proofs will appear in the full version of the paper.

2 Preliminaries

Distributed languages and algorithms. A *distributed language* is a set of *graph configurations* (G, x), where G is an undirected graph and $x : V(G) \to \mathcal{X}$ is an input assignment which assigns an input $x(v)$ from some input domain $\mathcal{X}$ to each node $v \in V(G)$. To simplify the notation, we sometimes write $(G, (x_1, \ldots, x_k))$ or $(G, x_1, \ldots, x_k)$ for a configuration where each node $v \in V(G)$ is given a list of inputs $x_1(v), \ldots, x_k(v)$, assigned by a compound input assignment $x_1 : V(G) \to \mathcal{X}_1, \ldots, x_k : V(G) \to \mathcal{X}_k$.

We assume that nodes have unique identifiers (UIDs) from some fixed UID space $\mathcal{U}$, and that during the execution of a distributed algorithm, each node has access to its own UID and its neighbors' UIDs. We denote by (G, x, id) an *identified graph configuration*, where (G, x) is a graph configuration, and $id : V(G) \to \mathcal{U}$ assigns a UID to each node. Note that the UIDs are not part of the definition of a distributed language. However, they can be used by a distributed algorithm that decides the language, e.g., to break symmetry (see the formal definition of an algorithm below).

We make the standard assumption that in graphs of size n, the inputs and the UIDs assigned to the nodes can be encoded in $\text{poly}(n)$ bits, meaning that an identified configuration with a graph of size n can be represented in $\text{poly}(n)$ bits. (This assumption is not essential, but if the representation length of the inputs and the UIDs is not polynomially-related to the size of the graph, we would need to introduce another parameter to bound their sizes.)

Let $N^t_{G,x,id}(v)$ denotes the t-neighborhood of v in the identified configuration (G, x, id), including the UIDs and the inputs of the nodes in the t-neighborhood. In some cases, when the UID assignment or the input assignment are irrelevant, we may use the notations $N^t_{G,x}(v)$ or $N^t_G(v)$, respectively. When both G and x are clear from the context, we write simply $N^t(v)$. Let $\mathcal{B}^t$ be the set of all t-neighborhoods that appear in some identified graph configuration using inputs from $\mathcal{X}$ and UIDs from $\mathcal{U}$.

Next we formally define local distributed algorithms. In a distributed algorithm, each node observes the neighborhood around itself, and then decides whether to accept or reject:

▶ **Definition 5** (Local decision algorithms). *A* t-local decision algorithm *is a computable mapping* $A : \mathcal{B}^t \to \{0, 1\}$*, which outputs a Boolean value (accept/reject). If* $A(N^t_{G,x,id}(v)) = 1$ *at all nodes* $v \in V(G)$*, then we say that* A accepts (G, x, id)*, and write* $A(G, x, id) = 1$*. We say that* A decides *the distributed language* $\mathcal{L}$ *if for every graph configuration* (G, x) *and for every UID assignment* $id : V(G) \to \mathcal{U}$*,*

$$(G, x) \in \mathcal{L} \qquad \Leftrightarrow \qquad A(G, x, id) = 1.$$

Given a t-local decision algorithm, we refer to t as the algorithm's *locality radius*.

▶ **Definition 6** (The classes LD, PLD). *Let* $t : \mathbb{N} \to \mathbb{N}$ *be a function. A distributed language* $\mathcal{L}$ *is in the class* $\mathsf{LD}(t(n))$ *if it can be decided in graphs of size* n *by a* $t(n)$*-local decision algorithm* A*. We refer to such algorithms as "*$\mathsf{LD}(t)$*-algorithms". If in addition the algorithm* A *(i.e., the mapping from* $t(n)$*-neighborhoods to an accept/reject bit) can be computed by a Turing machine that runs in time* $\mathrm{poly}(n)$ *in graph configurations of size* n*, then* $\mathcal{L}$ *is in the class* $\mathsf{PLD}(t)$*.*

For every function $t : \mathbb{N} \to \mathbb{N}$*, define* $\mathsf{LD}(O(t)) = \bigcup_{c>0} \mathsf{LD}(c \cdot t)$*, and similarly,* $\mathsf{PLD}(O(t)) = \bigcup_{c>0} \mathsf{PLD}(c \cdot t)$*. Let* $\mathsf{LD} = \mathsf{LD}(O(1))$*, and let* $\mathsf{PLD} = \mathsf{PLD}(O(1))$*.*

Note that, as usual in the area of local decision, the local algorithm does not necessarily know the size n of the network; nevertheless, as external observers, we can study the dependence of the algorithm's locality radius and its local running time on n. We introduce a separate class, $\mathsf{PLD}^{[n]}$, for local algorithms where the nodes *do* know the size of the network:

▶ **Definition 7** (The class $\mathsf{PLD}^{[n]}$). *Let* $t : \mathbb{N} \to \mathbb{N}$ *be a function. A distributed language* $\mathcal{L}$ *is in the class* $\mathsf{PLD}^{[n]}(t(n))$ *if the following language* $\mathcal{L}'$ *is in* $\mathsf{PLD}(t(n))$*:*

$$\mathcal{L}' = \{(G, (x, 1^n)) : (G, x) \in \mathcal{L} \text{ and } n = |V(G)|\}.$$

3 Deterministic Polynomial-Time Local Decision

In this section we study deterministic algorithms that are both local and run in polynomial time, and prove Theorem 1.

3.1 Unconditional Separation of PLD from P ∩ LD

We begin by proving the first part of Theorem 1, which states that $\mathsf{PLD} \subsetneq \mathsf{P} \cap \mathsf{LD}$. The non-strict containment, $\mathsf{PLD} \subseteq \mathsf{P} \cap \mathsf{LD}$, is easy to see: every PLD-algorithm is also an LD-algorithm, so $\mathsf{PLD} \subseteq \mathsf{LD}$; moreover, if every node of the network computes its decision in $\mathrm{poly}(n)$ time, then a poly-time centralized Turing machine can simulate the distributed algorithm by computing the output of every node, and accepting iff all nodes accept. Therefore $\mathsf{PLD} \subseteq \mathsf{P}$. The main challenge is to prove that the containment is strict, that is, $\mathsf{PLD} \neq \mathsf{P} \cap \mathsf{LD}$.

High-level overview. To separate PLD from $\mathsf{P} \cap \mathsf{LD}$, we use a variation on the language ITER, which was used in [2] to separate Π_1^{local} from LD with unbounded computation power. We call our variation ITER-BOUND.

The idea is to construct a language of paths, where the center node is given a Turing machine M, two inputs $a, b \in \{0,1\}^*$, and a bound $s \in \mathbb{N}$; the goal is to decide whether M halts on both a and b within at most s computation steps, and accepts either a or b (or both). The bound s may be much larger than the length of the overall size of the configuration: it is encoded in binary. An efficient algorithm cannot afford to run M for s steps and check whether it accepts a or b, but a local algorithm with unbounded computation time can do so, and therefore ITER-BOUND $\in$ LD. The bound s serves as a "trap door" that allows unbounded-time local algorithms to decide membership in ITER-BOUND, so that ITER-BOUND $\in$ LD.

To make the task solvable for a polynomial-time centralized algorithm we would like to restrict the length n of the path to be at least as long as the number of steps required for M to halt on a and on b. This would allow a centralized algorithm to run the machine M for n steps on a and on b, and check that it halts on both inputs and accepts at least one of them. However, we cannot simply add this restriction on the length of the path, as the resulting language would no longer be in LD: a local algorithm cannot necessarily "see" the entire path and does not know its length. Fortunately, there is a way to indirectly impose this restriction in a way that is locally-checkable: we annotate the nodes of the path (using the inputs of the nodes). On the left side of the path, from the center outwards, we write the sequence of configurations that M goes through in its computation on a, until it halts; on the right side of the path we do the same for b. (In particular, the length of the path must indeed be at least the number of steps required for M to halt on a and on b.) A centralized algorithm can run now either run M for n steps on a and on b as described above, or it can simply examine the computation sequence of M, make sure it obeys the transition function of M, and verify that at either the left or the right side of the path (or both) we have an accepting configuration of M. Thus, ITER-BOUND $\in$ P. The annotations can also be checked by a local distributed algorithm which simply verifies that every two neighboring nodes have consecutive computation steps of M, and that nodes at the end of the path (that is, nodes of degree 1) have halting configurations of M. Thus, even after adding the annotations, the language is still in LD.

Finally, we prove that an algorithm that is both local and efficient cannot decide the language ITER-BOUND: intuitively, this is because it can neither afford to run M for s steps, nor can it "see" both endpoints of the path at the same time, to verify that at least one of them has an accepting configuration. The formal proof shows that if there existed a PLD-algorithm for ITER-BOUND then we could use it to decide in polynomial time a language that is not in P.

Detailed construction. Recall that the configuration of a Turing machine consists of the contents of the tape, the location of the tape head, and the current state of the machine. To avoid confusion, in the sequel we refer to configurations of Turing machines as *TM-configurations*, and to graph configurations as simply *configurations*.

Given a Turing machine M, an input $a \in \{0,1\}^*$, and a number $i \in \mathbb{N}$, let $\mathsf{config}(M, a, i)$ denote the Turing machine configuration that M reaches after taking i steps on input a; if M halts in fewer than i steps on input a, then $\mathsf{config}(M, a, i)$ is the configuration in which M halts on input a.

Let M be a Turing machine, let $a, b \in \{0,1\}^*$ be inputs on which M halts, and let $s, n_L, n_R \in \mathbb{N}$ such that $n_L \geq |a|, n_R \geq |b|$ and $s \leq c \cdot 2^{n_L+n_R+1}$, for some constant c whose value will be fixed later.[2] We define a configuration $C^{n_L,n_R}(M,a,b,s) = (G,x)$, as follows:

- G is a path of the form $u_{n_L}, \ldots, u_1, v, w_1, \ldots, w_{n_R}$, consisting of a *pivot node* $v \in V(G)$, a left sub-path $L = u_{n_L}, \ldots, u_1$, and a right sub-path $R = u_1, \ldots, u_{n_R}$.
- The input of the pivot node v is $x(v) = (0, \langle M \rangle, a, b, s)$, where $\langle M \rangle$ is the encoding of the Turing machine M.
- For each node $u_i \in L$ on the left sub-path, the input of u_i is $x(u_i) = (i, \langle M \rangle, \mathsf{config}(M,a,i))$. Similarly, for each node $w_i \in R$ on the right sub-path, the input of w_i is given by $x(w_i) = (i, \langle M \rangle, \mathsf{config}(M,b,i))$.

The language ITER-BOUND consists of all configuration $C^{n_L,n_R}(M,a,b,s)$ such that

- The TM-configurations $\mathsf{config}(M,a,n_L), \mathsf{config}(M,b,n_R)$ that are written at the ends of the two sub-paths are both halting, and
- M halts in at most s steps on a and on b, and accepts at least one of them.

Given a configuration $C^{n_L,n_R}(M,a,b,s) = (G,x)$ as defined above, we say that a node $u \in V(G)$ is *r-central* if the distance of u from the pivot is at most r.

As we explained above, it is not difficult to see that ITER-BOUND can be decided by a local algorithm, and is also in P. In particular, a local algorithm that decides membership in ITER-BOUND only needs to check the following conditions: at any node v,

- If v's input starts with the number 0, then the input is of the form $x(v) = (0, \langle M \rangle, a, b, s)$, where M halts on both a and b in at most s steps, and accepts at least one of them. Also, the degree of v is 2.
- If v's input starts with a number $i > 0$, then the input is of the form $x(v) = (i, \langle M \rangle, \mathsf{cfg})$, and v has a neighbor u whose input is $(i-1, \langle M \rangle, \mathsf{cfg}')$, where cfg' is a TM-configuration that precedes cfg according to the transition function of M. The degree of v must be either 2 or 1. If the degree is 1, then cfg must be a halting configuration of M (that is, the state of M in cfg is either the accepting or the rejecting state of M). If the degree is 2, then v must have another neighbor whose input is $(i+1, \langle M \rangle, \mathsf{cfg}'')$, where cfg'' is the TM-configuration that follows cfg according to the transition function of M.

A centralized poly-time algorithm can decide membership in ITER-BOUND by verifying the same local consistency conditions that the local algorithm checks (e.g., that each non-pivot node's input has a TM-configuration that follows the TM-configuration given to the preceding node, and so on). It can also directly check that at least one of the two endpoints of the path contains an accepting configuration. Finally, to verify that M halts in at most s steps on a (and similarly for b), the centralized algorithm can compute the lengths n_L of the left sub-path, and compare: if $s \geq n_L$, then indeed, since n_L computation steps suffice for M to half on a (which is checked using the local consistency conditions), $s \geq n_L$ steps also suffice. On the other hand, if $s < n_L$, then it is permissible for the algorithm to run M for s steps and check whether it halts – the time required to do so is polynomial in $s < n_L < n$ and in the input size $|a| \leq n_L < n$.

Next we prove that ITER-BOUND is not decidable by a polynomial-time local algorithm. In fact, the claim below generalizes to any sublinear locality radius (that is, the class $\mathsf{PLD}(t(n))$ for any $t(n) = o(n)$), but for the sake of simplicity we state it for a constant locality radius (the class $\mathsf{PLD} = \mathsf{PLD}(O(1))$.

[2] This is to ensure that the size of the graph is indeed polynomially-related to the length of the inputs (in bits). For simplicity, we assume that this constraint is imposed externally, that is, the nodes do not need to verify that their inputs have the correct number of bits. However, it is not difficult to modify the proof to verify that the inputs are not too long, though this requires some modifications to the definition of the language ITER-BOUND.

▷ Claim 8. ITER-BOUND $\notin$ PLD.

Proof. Suppose for the sake of contradiction that there is a PLD-algorithm A that decides ITER-BOUND, and let $t > 0$ be its locality radius. Let $\mathcal{L} \in \mathsf{DTIME}(2^n) \setminus \mathsf{P}$ be some language that is Turing-decidable in time $O(2^n)$ but not in polynomial time, and such that $\epsilon \notin \mathcal{L}$ (here and in the sequel, ϵ denotes the empty word). Such a language exists by the Time Hierarchy Theorem [14]. We claim that using the PLD-algorithm A that decides ITER-BOUND, we can construct a polynomial-time Turing machine that decides $\mathcal{L}$ for inputs of size n for any sufficiently large n, a contradiction to the fact that $\mathcal{L} \notin \mathsf{P}$.

Let M be a $\mathsf{DTIME}(2^n)$-time Turing machine that decides $\mathcal{L}$, and let $f \in O(2^n)$ be a function bounding the running time of M on inputs of length n. We assume that for all $n > 0$ we have $f(n) \geq f(0)$ (if not, simply define $f'(n) = \max(f(n), f(0))$ and use $f'(n)$ in place of $f(n)$).

Given input $z \in \{0,1\}^*$, let $C_z = C^{f(|z|),f(|z|)}(M, \epsilon, z, f(|z|))$ be the configuration that encodes the runs of M on ϵ on the left sub-path, and on z on the right sub-path, until M halts, using sub-paths of length $f(|z|)$ in both directions. Since $f(|z|)$ steps suffice for M to halt on ϵ and on z, but $\epsilon \notin \mathcal{L}$, we have $C_z \in$ ITER-BOUND iff $z \in \mathcal{L}$.

We define a poly-time Turing machine M' that decides $\mathcal{L}$ as follows: on input $z \in \{0,1\}^*$, M' constructs the configuration $C'_z := C^{2t,2t}(M, \epsilon, z, f(|z|))$, which is essentially the central portion of C_z, including only $2t$ nodes to the left and to the right of the pivot (a total of $4t + 1$ nodes). Next, M' simulates the local algorithm A at all the nodes of C'_z. Finally, M' accepts iff A outputs 1 at all t-central nodes of C'_z, ignoring the outputs of the other nodes.

It is not difficult to verify that the running time of M' is polynomial in $|z|$, in the description length of M (which is constant), and in t (which is also constant). To show that M' indeed decides $\mathcal{L}$, suppose first that $z \in \mathcal{L}$. Then $C_z \in$ ITER-BOUND by construction, and therefore A must output 1 at all nodes of C_z. But this means that all t-central nodes in C'_z must also accept: for each t-central node u in C'_z, the t-local view of u is the same in C_z and in C'_z, because C'_z is obtained from C_z by removing only nodes at distance greater than t from u. Since the output of u depends only on its t-local view, and we know that u accepts in C_z, it must also accept in C'_z. Therefore M' accepts z.

Now suppose that $z \notin \mathcal{L}$. In this case, $C_z \notin$ ITER-BOUND, because in C_z the two inputs encoded in the configuration are both rejected by M (as $\epsilon, z \notin \mathcal{L}$). We claim that at least one t-central node of C_z must reject; as above, this means that the same node also rejects in C'_z, causing M' to reject z.

Suppose for the sake of contradiction that all t-central nodes of C_z accept. However, since $C_z \notin$ ITER-BOUND, we know that some node of C_z rejects; let u be such a node. The distance of u from the pivot v must be greater than t, since we assumed that no t-central node rejects. Now fix some string $a \in \mathcal{L}$ (which must exist, as $\emptyset \in \mathsf{P}$ and we assumed $\mathcal{L} \notin \mathsf{P}$), let $n' = \max(f(|a|), f(|z|))$, and let $C_{a,z} = C^{n',n'}(M, a, z, f(\max(|a|, |z|)))$ be the configuration encoding the runs of M on a (on the left sub-path) and on z (on the right sub-path), using sub-paths of length n', so that M halts on both by the end of both sub-paths. Since $a \in \mathcal{L}$, we have $C_{a,z} \in$ ITER-BOUND, and thus all nodes must accept $C_{a,z}$. This includes node u. However, u is at distance greater than t from the pivot, and therefore its t-local view is the same in $C_{a,z}$ and in C_z; thus, u also accepts in C_z, a contradiction. ◁

As to the constant c that appears in the definition of the language ITER-BOUND (where we required that $s \leq c \cdot 2^{n_L+n_R+1}$), we can choose c to be any constant such that the function $f \in O(2^n)$ in the proof above satisfies $f(n) \leq c \cdot 2^n$ for sufficiently large n.

3.2 Conditional Separation for Polynomial-Time Local Algorithms with Known Network Size

In the previous section we showed that $\mathsf{P} \cap \mathsf{LD} \not\subseteq \mathsf{PLD}$, but our proof used the fact that the nodes do not know the size of the graph, and therefore their output when the graph is a short path is the same as their output on a long path, provided their local neighborhood stays the same. This is a common assumption in distributed computing, but it could be considered problematic when bounding computational resources as a function of the size of the graph. In classical computational complexity, this issue typically does not arise, as centralized algorithms are able to read their entire input and compute its length – with some notable exceptions, such as sublinear-time algorithms, but there it is usually assumed that the input's size is known to the algorithm.

In this section we ask whether the separation of PLD from $\mathsf{LD} \cap \mathsf{P}$ continues to hold if the size of the network is known: let $\mathsf{LD}^{[n]}, \mathsf{PLD}^{[n]}$ be variants of $\mathsf{LD}, \mathsf{PLD}$ (resp.), where nodes receive the size n of the graph as part of their input. Is it still true that $\mathsf{P} \cap \mathsf{LD}^{[n]} \not\subseteq \mathsf{PLD}^{[n]}$?

In what follows, we prove the second and third parts of Theorem 1. We first prove that $\mathsf{P} \cap \mathsf{LD}^{[n]} \not\subseteq \mathsf{PLD}^{[n]}$ would imply $\mathsf{P} \neq \mathsf{NP}$, which makes an unconditional proof of this separation unlikely. We then show that this separation is implied by the assumption that $\mathsf{UP} \cap \mathsf{coUP} \neq \mathsf{P}$, which is stronger than assuming $\mathsf{P} \neq \mathsf{NP}$, but still reasonable.

▷ Claim 9. If $\mathsf{P} \cap \mathsf{LD}^{[n]} \not\subseteq \mathsf{PLD}^{[n]}$, then $\mathsf{P} \neq \mathsf{NP}$.

Proof. We prove the contrapositive: assume that $\mathsf{P} = \mathsf{NP}$, and let us show that every language $\mathcal{L} \in \mathsf{P} \cap \mathsf{LD}^{[n]}$ is also in $\mathsf{PLD}^{[n]}$.

Let $\mathcal{L} \in \mathsf{P} \cap \mathsf{LD}^{[n]}$. We show that $\mathcal{L} \in \mathsf{PLD}^{[n]}$ by constructing a t-local algorithm for $\mathcal{L}$ where every node runs in polynomial time. The idea is to have each node check whether its t-neighborhood can be extended into a configuration in $\mathcal{L}$. We prove that this decides $\mathcal{L}$, relying on the fact that $\mathcal{L}$ can be decided by a t-local algorithm. Since $\mathcal{L} \in \mathsf{P}$, asking whether there exist a configuration $(G, x) \in \mathcal{L}$ that extends the current t-neighborhood of the node is an NP-question, and since we assumed that $\mathsf{P} = \mathsf{NP}$, the resulting algorithm can also be computed in *deterministic* polynomial time, yielding a $\mathsf{PLD}^{[n]}$-algorithm for $\mathcal{L}$.

More formally, let A be a t-local algorithm where each node accepts its t-neighborhood $N^t(v)$ if and only if there exists an identified configuration $(\tilde{G}, \tilde{x}, \tilde{id}) \in \mathcal{GC}$ such that

- The t-neighborhood of v appears in $(\tilde{G}, \tilde{x}, \tilde{id})$, that is, there exists some $u \in V(\tilde{G})$ such that $N^t(v) = N^t_{\tilde{G},\tilde{x},\tilde{id}}(u)$); and
- $(\tilde{G}, \tilde{x}) \in \mathcal{L}$.

We claim that A decides $\mathcal{L}$, and that A is indeed a $\mathsf{PLD}^{[n]}$-algorithm.

To see that A decides $\mathcal{L}$ we must prove that for every identified configuration (G, x, id),

- If $(G, x) \in \mathcal{L}$ then all nodes accept: indeed, for every node $v \in V(G)$, there exists an identified configuration $(\tilde{G}, \tilde{x}, \tilde{id}) = (G, x, id)$, in which the t-neighborhood of v of course appears, such that $(\tilde{G}, \tilde{x}) = (G, x) \in \mathcal{L}$.
- If all nodes accept, then $(G, x) \in \mathcal{L}$: fix some $\mathsf{LD}^{[n]}$-algorithm B for $\mathcal{L}$, which exists by our assumption that $\mathcal{L} \in \mathsf{LD}^{[n]}$. To show that $(G, x) \in \mathcal{L}$, it suffices to show that under B, all nodes accept in (G, x, id). To that end, let $v \in V(G)$. Because v accepts under A, there exists some identified configuration $(\tilde{G}, \tilde{x}, \tilde{id})$ in which v's t-neighborhood appears, such that $(\tilde{G}, \tilde{x}) \in \mathcal{L}$. But since B decides $\mathcal{L}$, all nodes must accept in $(\tilde{G}, \tilde{x}, \tilde{id})$, which means that v's t-neighborhood is accepted under B.

To see that A is an $\mathsf{PLD}^{[n]}$-algorithm, let M be a poly-time Turing machine that decides $\mathcal{L}$, which exists by our assumption that $\mathcal{L} \in \mathsf{P}$. Observe that A can be implemented by a *nondeterministic* Turing machine M' that takes as input the t-neighborhood of node v, and as witness the identified configuration $(\tilde{G}, \tilde{x}, \tilde{id})$ (verifying first that it is a legal identified

configuration); to verify that the t-neighborhood of v appears in $(\tilde{G}, \tilde{x}, \tilde{id})$, we find the unique node $u \in V(\tilde{G})$ that has $\tilde{id}(u) = id(v)$ and verify that $N^t(v) = N^t_{\tilde{G},\tilde{x},\tilde{id}}(u)$; and to verify that $(\tilde{G}, \tilde{x}) \in \mathcal{L}$, we run M. We note that since the input of node v includes the size of the network in unary (1^n), this computation requires polynomial time in the input of v. Finally, since we assumed that $\mathsf{P} = \mathsf{NP}$, the fact that A can be computed in nondeterministic polynomial time also implies that it can be computed in deterministic polynomial time, as desired. ◁

Next we prove that the separation $\mathsf{PLD} \subsetneq \mathsf{P} \cap \mathsf{LD}$ holds assuming that $\mathsf{UP} \cap \mathsf{coUP} \neq \mathsf{P}$. First, let us formally define the class UP [26], which stands for "unambiguous nondeterministic polynomial-time":

▶ **Definition 10.** *The class* UP *is the set of all languages* $\mathcal{L} \subseteq \{0,1\}^*$ *for which there exists a polynomial-time Turing machine* M *and a polynomial* p*, such that*

$$x \in \mathcal{L} \;\Leftrightarrow\; \exists!\, w \in \{0,1\}^{p(|x|)} :\; M(x,w) = 1.$$

Here, $\exists!$ *is the quantifier for unique existence.*

The class coUP is the class of all languages whose complement is in UP. The intersection $\mathsf{UP} \cap \mathsf{coUP}$ contains some natural problems, such as integer factorization and finding the winner of parity games [3, 17]. It is easy to see that $\mathsf{P} \subseteq \mathsf{UP} \cap \mathsf{coUP} \subseteq \mathsf{NP} \cap \mathsf{coNP}$, and it is widely assumed that $\mathsf{UP} \cap \mathsf{coUP} \neq \mathsf{P}$, as otherwise factoring, upon whose hardness some cryptographic systems rely, is decidable in polynomial time.

High-level overview. To prove the separation, we fix some language $\mathcal{L} \in \mathsf{UP} \cap \mathsf{coUP} \backslash \mathsf{P}$, and UP, coUP machines for it, $M_{\mathcal{L}}, M_{\bar{\mathcal{L}}}$, respectively. We construct a distributed language on paths $\mathcal{P}_{\mathcal{L}}$, where each configuration is of the form $C(z, w, j, b)$, such that:

- $z \in \{0,1\}^*$,
- w is either the unique witness for the statement "$z \in \mathcal{L}$" or the unique witness for the statement "$z \notin \mathcal{L}$", depending on which of the two statements is true (clearly both cannot be true at the same time),
- $j \in [|w|]$ is an index, and
- $b = w_j$ is the j-th bit of the witness w.

In the configuration $C(z, w, j, b)$, we give a string z and index $j \in [|w|]$ to every node of the path except for the first and the last nodes of the path. We give the bit b to the first node of the path, and we give the entire witness w to the last node of the path.

It is not hard to see that $\mathcal{P}_{\mathcal{L}} \in \mathsf{P} \cap \mathsf{LD}^{[\mathsf{n}]}$: since the configuration encodes the witness (it is the input of the last node), a centralized algorithm can decide whether a given path configuration is in the language or not in polynomial time, using the machines $M_{\mathcal{L}}$ and $M_{\bar{\mathcal{L}}}$. A local algorithm with unbounded computation power can also decide membership in $\mathcal{P}_{\mathcal{L}}$ by having each node verify that its input is correctly formatted and matches its neighbors; in addition, the first node obtains z and j from its neighbor, computes the witness w using its unbounded computation time, and verifies that $b = w_j$; the last node computes the witness w and verifies that it matches its input.

Now assume for the sake of contradiction that $\mathsf{PLD}^{[\mathsf{n}]} = \mathsf{P} \cap \mathsf{LD}^{[\mathsf{n}]}$. Then since $\mathcal{P}_{\mathcal{L}} \in \mathsf{P} \cap \mathsf{LD}^{[\mathsf{n}]}$, we also have $\mathcal{P}_{\mathcal{L}} \in \mathsf{PLD}^{[\mathsf{n}]}$, meaning there is a t-local algorithm for $\mathcal{P}_{\mathcal{L}}$ where every node runs in poly-time. We show that a centralized poly-time verifier is able to decide membership in the original language $\mathcal{L}$, by essentially "guessing" the witness bit-by-bit: given input $z \in \{0,1\}^*$, the verifier guesses the j-th bit of the witness w for z by simulating the first t nodes of the configuration $C(z, \vec{0}, j, 0)$: if the first t nodes accept, we guess that $w_j = 0$, and otherwise we guess that $w_j = 1$. Recall that in $\mathcal{P}_{\mathcal{L}}$, the witness w is only supposed to be given to the *last* node on the path. We prove that the first t nodes of the configuration must

"know" whether $w_j = 0$ or $w_j = 1$, even though they do not "see" the witness w at the end of the path, and this can be exploited to compute the j-th bit in poly-time. After computing all bits of the witness w, the verifier simply runs $M_{\mathcal{L}}(z, w)$, and outputs the same.

Detailed construction. Let $z, w \in \{0,1\}^*$ be strings, let $j \in [|w|]$ be an index, and let $b \in \{0,1\}$ be a bit. We define a configuration $C(z, w, j, b) = (G, x)$ as follows:

- G is a path $v_1, \ldots, v_n$, where $n = |z|$.
- The input assignment x is given by $x(v_1) = (1, b)$, $x(v_i) = (i, z, j)$ for every $1 < i < n$, and $x(v_n) = (n, w)$.

Let $\mathcal{L}$ be a language such that $\mathcal{L} \in \mathsf{UP} \cap \mathsf{coUP}$, and let $M_{\mathcal{L}}$ and $M_{\bar{\mathcal{L}}}$ be the respective verifying Turing machines for $\mathcal{L}$ and for $\bar{\mathcal{L}}$. We assume w.l.o.g. that $M_{\mathcal{L}}$ and $M_{\bar{\mathcal{L}}}$ take witnesses of the same polynomial length p; that is, let p be such that for every instance z of size n, there exists a unique witness w such that $|w| = p(n)$ and either $M_{\mathcal{L}}(z, w) = 1$ or $M_{\bar{\mathcal{L}}}(z, w) = 1$ (but not both). For a string $z \in \{0,1\}^n$, let $w(z) \in \{0,1\}^{p(n)}$ be that witness.

The language $\mathcal{P}_{\mathcal{L}}$ consists of all configurations $C(z, w, j, b)$ such that $w = w(z)$ and $w_j = b$.

It is not hard to see that for every $\mathcal{L} \in \mathsf{UP} \cap \mathsf{coUP}$, we have $\mathcal{P}_{\mathcal{L}} \in \mathsf{P} \cap \mathsf{LD}^{[\mathsf{n}]}$. We now show that for $\mathcal{L} \in \mathsf{UP} \cap \mathsf{coUP} \backslash \mathsf{P}$ we have $\mathcal{P}_{\mathcal{L}} \notin \mathsf{PLD}^{[\mathsf{n}]}$.

▷ Claim 11. Assume $\mathsf{UP} \cap \mathsf{coUP} \neq \mathsf{P}$, and let $\mathcal{L} \in \mathsf{UP} \cap \mathsf{coUP} \backslash \mathsf{P}$. Then $\mathcal{P}_{\mathcal{L}} \notin \mathsf{PLD}^{[\mathsf{n}]}$.

Proof. Suppose for the sake of contradiction that $\mathcal{P}_{\mathcal{L}} \in \mathsf{PLD}^{[\mathsf{n}]}$, and let A be a t-local polynomial-time algorithm for $\mathcal{P}_{\mathcal{L}}$, for some constant t. Let $\mathsf{time}(A)$ denote the running time of A in each node. Then the following centralized, polynomial-time algorithm B decides $\mathcal{L}$ for all inputs of size $n > 2t$, contradicting the fact that $\mathcal{L} \notin \mathsf{P}$. (Inputs of length $\leq 2t$ can be decided by, e.g., exhaustive tabulation – going through all of them and hard-coding the answer into the Turing machine, since there are only finitely many such inputs.)

Given input z of length $|z| = n > 2t$, for each $j \leq p(n)$, let $a_j \in \{0,1\}$ be the bit computed by the following procedure:

1. Construct the configuration $C(z, \vec{0}, j, 0)$, where $\vec{0}$ is a vector comprising $p(n)$ zeroes.
2. Simulate the execution of A on the first t nodes $v_1, \ldots, v_t$ of $C(z, \vec{0}, j, 0)$.
3. If nodes $v_1, \ldots, v_t$ all accept, set $a_j = 0$, and otherwise set $a_j = 1$.

Finally, let $a = a_1, \ldots, a_{p(n)}$. The centralized algorithm B runs $M_{\mathcal{L}}(z, a)$, and accepts iff $M_{\mathcal{L}}(z, a)$ accepts.

To prove the correctness of our algorithm B, fix $z \in \{0,1\}^*$, and let $w = w(z)$ be the witness corresponding to z. First observe that if $z \notin \mathcal{L}$, then algorithm B rejects: in this case there does not exist any string $a \in \{0,1\}^*$ such that $M_{\mathcal{L}}(z, a)$ accepts, so no matter how a is computed, B will always reject.

Now suppose that $z \in \mathcal{L}$. We claim that the string a computed by B is exactly the witness w such that $M_{\mathcal{L}}(z, w)$ accepts: for each $j \in [p(n)]$,

- If $w_j = 0$, then in the configuration $C(z, \vec{0}, j, 0)$, the first t nodes have the same view as they do in $C(z, w, j, w_j = 0)$. (Recall that we assumed $n > 2t$, and therefore the view of the first t nodes does not include v_n, the only node that is given the full witness w.) By definition, $C(z, w, j, w_j) \in \mathcal{P}_{\mathcal{L}}$, and therefore all nodes must accept; thus, the first t nodes must accept in $C(z, \vec{0}, j, 0)$. This causes us to set $a_j = w_j = 0$.
- If $w_j = 1$, then in the configuration $C(z, w, j, 0)$ some node must reject, as this configuration is not in $\mathcal{P}_{\mathcal{L}}$. Moreover, every node v_k where $k > t$ must accept, because all nodes at distance $> t$ from v_1 have the same view that they would in $C(z, w, j, 1)$, which is in $\mathcal{P}_{\mathcal{L}}$. Therefore it is one of the first t nodes that rejects, causing us to set $a_j = w_j = 1$.

This shows that $w = a$, which means that B accepts z.

Regardless of whether $z \in \mathcal{L}$ or not, the runtime of B is polynomial: simulating the execution of a $\mathsf{PLD}^{[n]}$-algorithm at $t = O(1)$ nodes requires polynomial time, and we repeat this $p(n)$ times to compute all the bits of the witness. ◁

4 Adding Nondeterminism

In this section we characterize the computation power of nondeterministic local algorithms with polynomial time local computation, and show that they can decide any language that can be decided by both a computationally unbounded nondeterministic local algorithm and poly-time nondeterministic Turing machine. This holds regardless of whether the size of the network is known or not.

We begin by formally defining the nondeterministic variants of LD, PLD and $\mathsf{PLD}^{[n]}$. Given a graph G, let $\mathcal{C}(G)$ denote the set of all assignments $c : V(G) \to \{0,1\}^*$. A nondeterministic algorithm is one where the nodes are given "nondeterministic advice", in the form of a *certificate assignment* $c \in \mathcal{C}(G)$:

▶ **Definition 12** (The classes NLD, NPLD, $\mathsf{NLD}^{[n]}$ and $\mathsf{NPLD}^{[n]}$)**.** *A distributed language $\mathcal{L}$ is in the class NLD if there exists an LD-algorithm A such that for every configuration (G,x),*

$$(G,x) \in \mathcal{L} \qquad \Leftrightarrow \qquad \exists c \in \mathcal{C}(G) \forall id \, (A \text{ accepts } (G,(x,c),id)\ ).$$

The class NPLD is defined similarly, but the algorithm A is required to run in polynomial time at every node, and the certificate $c(v)$ is required to have polynomial length at every node v.

The classes $\mathsf{NLD}^{[n]}$, $\mathsf{NPLD}^{[n]}$, are the variants of NLD, NPLD (respectively), where the size of the network is known to all nodes: each node receives 1^n in addition to its usual input.

We emphasize that, following [9], the certificates c are chosen *before* the UID assignment; in other words, the certificates may not depend on the UIDs.

Our proof of the first part of Theorem 2, which asserts that $\mathsf{NPLD} = \mathsf{NP} \cap \mathsf{NLD}$, uses a characterization of the class NLD from [7]: NLD is the class of distributed languages that are *closed under lift*. A configuration (G',x') is a *t-lift* of a configuration (G,x) if there exists a mapping $\phi : V(G') \to V(G)$ such that for every $u \in V(G')$, ϕ induces an input-preserving isomorphism between $N^t_{G,x}(\phi(u))$ and $N^t_{G',x'}(u)$, meaning that for each $v \in V(G')$ we have $x'(v) = x(\phi(v))$. A distributed language $\mathcal{L}$ is *closed under lift* if there exists some $t \geq 0$ such that for every configuration $(G,x) \in \mathcal{L}$, all t-lifts of (G,x) are also in $\mathcal{L}$.

▶ **Lemma 13** ([7])**.** NLD *is the class of all distributed languages closed under lift.*

We are now ready to prove the first part of Theorem 2.

▷ Claim 14. $\mathsf{NPLD} = \mathsf{NP} \cap \mathsf{NLD}$.

Proof. The inclusion $\mathsf{NPLD} \subseteq \mathsf{NP} \cap \mathsf{NLD}$ is easy to see, as an NPLD-algorithm is in particular an NLD-algorithm, and it can also be efficiently simulated by a polynomial-time Turing machine that is given all the nodes' certificates.

To see that $\mathsf{NP} \cap \mathsf{NLD} \subseteq \mathsf{NPLD}$, let $\mathcal{L} \in \mathsf{NP} \cap \mathsf{NLD}$, let A be a t-local algorithm for $\mathcal{L}$, and let M be an NP-verifier for $\mathcal{L}$. We construct the following NPLD-algorithm, B: given a configuration (G,x) on n nodes, we give to each node a certificate $c(v) = (i,(G',x'),w)$, where

- $i \in \{1,\ldots,n\}$ is an index,
- G' and x' represent the configuration (G,x), using $\{1,\ldots,n\}$ as the vertices,
- w is an NP-witness such that M accepts $((G',x'),w)$.

The nodes locally verify that

- Their t-neighborhood in G' is isomorphic to their true neighborhood in G, using the indices provided in the certificates as the isomorphism,
- x' correctly describes their input, again using the index,
- They received the same (G', x') as their neighbors, and finally,
- M accepts $((G', x'), w)$.

The first part of the verification passes if and only if (G, x) is a t-lift of (G', x'). If $(G, x) \in \mathcal{L}$, then the certificates specified above cause all nodes to accept. Conversely, if all nodes accept, then (G', x') is a lift of (G, x), and since $M((G', x'), w)$ accepts, we have $(G', x') \in \mathcal{L}$. This implies that $(G, x) \in \mathcal{L}$ as well, because $\mathcal{L}$ is closed under t-lifts. ◁

Finally, consider the setting where the network size is known, that is, the class $\mathsf{NLD}^{[n]}$. In [9], all Turing-decidable distributed languages are shown to be in $\mathsf{NLD}^{[n]}$: the proof is similar to the proof of Claim 14 above, except that when the size of the graph is known, the only possible lift is the graph itself. Computational efficiency was not taken into consideration in [9].

Using a very similar argument, we can modify the proof of Claim 14 to show that $\mathsf{NPLD}^{[n]} = \mathsf{NP} \cap \mathsf{NLD}^{[n]}$: in the proof of Claim 14, if nodes know the network size and check that the configuration (G', x') described in their certificate has that size, then we are guaranteed (G', x') is the true input configuration, rather than merely a lift of it. Thus, when nodes run M on (G', x') using the witness w and verify that it accepts, they are actually verifying that M accepts the true input configuration, i.e., that the input configuration is in the language $\mathcal{L}$.

5 A Polynomial-Time Local Hierarchy

Inspired by the centralized polynomial hierarchy on one hand, and by the local hierarchy of [2] on the other, we conclude by studying a hierarchy that combines both locality and computational constraints.

5.1 Defining a Polynomial Time Local Hierarchy

In what follows, we let $\mathcal{U}(G)$ denote the set of UID assignments $V(G) \to \mathcal{U}$.

▶ **Definition 15** (The classes Σ_k^{local}, Π_k^{local}, $\Sigma_k^{\mathsf{P}\text{-}local}$, $\Pi_k^{\mathsf{P}\text{-}local}$). *A distributed language $\mathcal{L}$ is in the class Σ_k^{local} (resp., $\Sigma_k^{\mathsf{P}\text{-}local}$) if it can be decided by an* LD*-algorithm (resp.,* PLD*-algorithm) A where the nodes are given certificates $c_1, \ldots, c_k$, quantified as follows:*

$$(G, x) \in \mathcal{L} \Rightarrow \exists c_1 \forall c_2 \ldots Q c_k \in \mathcal{C}(G) \forall id \in \mathcal{U}(G) : \; A(G, x, c_1, \ldots, c_k, id) = 1$$
$$(G, x) \notin \mathcal{L} \Rightarrow \forall c_1 \exists c_2 \ldots Q' c_k \in \mathcal{C}(G) \forall id \in \mathcal{U}(G) : \; A(G, x, c_1, \ldots, c_k, id) = 0,$$

where if k is even then Q is the universal quantifier ($\forall$), and if k is odd then Q is the existential quantifier ($\exists$); in both cases, Q' is the opposite quantifier to Q. In the case of the class $\Sigma_k^{\mathsf{P}\text{-}local}$, the certificates $c_1, \ldots, c_k$ are required to be of polynomial length at every node (that is, for some polynomials $p_1, \ldots, p_k$, the quantifiers range only over certificates of length $p_1(n), \ldots, p_k(n)$, respectively).

The class Π_k^{local} (resp., $\Pi_k^{\mathsf{P}\text{-}local}$) is defined similarly to Σ_k^{local} (resp., Π_k^{local}), with the first quantifier being universal instead of existential.

We remark that as in the nondeterministic case, the certificates may not depend on the UIDs (this is the original definition from [2]). This leads to some complications: for example, in the centralized polynomial hierarchy, $\Sigma_1^{\mathsf{P}} = \mathsf{NP}$ and $\Pi_1^{\mathsf{P}} = \mathsf{coNP}$, and in general, for every $k \in \mathbb{N}$ we have $\Pi_k^{\mathsf{P}} = \mathsf{co}\Sigma_k^{\mathsf{P}}$ (in other words, $\mathcal{L} \in \Pi_k^{\mathsf{P}}$ iff $\overline{\mathcal{L}} \in \Sigma_k^{\mathsf{P}}$). This is not the case with the local hierarchy and with the polynomial-time local hierarchy. For example, the complement of the condition "there exist certificates such that under all UID assignments, all nodes accept", which corresponds to Σ_1^{local}, is *not* the condition "for all certificates and for all UID assignments, all nodes accept", which corresponds to membership in Π_1^{local}. Thus, it is not necessarily the case that $\mathcal{L} \in \Pi_1^{local}$ iff $\overline{\mathcal{L}} \in \Sigma_1^{local}$.

Nevertheless, some characteristics of the centralized polynomial hierarchy do carry over to the local hierarchy, and similarly, to the polynomial local hierarchy: for example, for every $k \geq 0$ we have $\Sigma_k^{local} \subseteq \Pi_{k+1}^{local}$ and $\Pi_k^{local} \subseteq \Sigma_{k+1}^{local}$, as we can simply have the algorithm ignore the $(k+1)$-th level of certificates if desired.

5.2 The Upper Levels of The Hierarchy

We show that on the higher levels of the polynomial local hierarchy, the local computation constraint dominates the locality constraint: each higher-level class of the polynomial local hierarchy is *equal* to the corresponding class of the centralized polynomial hierarchy, with no loss in expressiveness caused by the locality restriction.

▷ Claim 16 (Theorem 3, part 2). For every $k \geq 2$ we have $\Pi_k^{\mathsf{P}\text{-}local} = \Pi_k^{\mathsf{P}}$, and for every $k \geq 3$ we have $\Sigma_k^{\mathsf{P}\text{-}local} = \Sigma_k^{\mathsf{P}}$.

The proof follows almost immediately from an argument used in [2] to prove that Π_2^{local} contains all Turing-decidable languages: we can use the same argument to prove Claim 16, taking care to analyze the running times involved.

In [2], to prove that all decidable languages $\mathcal{L}$ are in Π_2^{local}, the authors construct a local distributed algorithm D which, informally speaking, verifies the following statement: "for every configuration (G', x'), either (G', x') is in $\mathcal{L}$, or (G', x') is not the current configuration". (This statement is logically equivalent to asserting that the current configuration is in $\mathcal{L}$.) To that end, the algorithm D is given two certificates: c_1, which is universally quantified, is a description of a configuration (G', x'), along with indices embedding each node into (G', x'), as in the proof of Claim 14; c_2, which is existentially quantified, can take one of two forms:

- If $(G', x') \in \mathcal{L}$, then $c_2 = \perp$ at all nodes.[3] The nodes can verify by themselves that $(G', x') \in \mathcal{L}$, as they have no computational restrictions, and this is what they do in this case.
- If $(G', x') \notin \mathcal{L}$, then c_2 points out some inconsistency that convinces the nodes that (G', x') is not the current configuration. For example, c_2 may prove that c_1 assigns two nodes the same index, or that there is some node whose true neighborhood does not match G', or that there exist two nodes that received different descriptions of (G', x').

Together, both certificates assert that "every input configuration that is not in $\mathcal{L}$ is not the current input configuration", which implies that the current input configuration *is* in $\mathcal{L}$. Although this is not stated explicitly in [2], the algorithm that verifies the certificates c_1, c_2 runs in time polynomial in the size of the graph and the input to the nodes, *except* when $c_2 = \perp$, in which case the nodes need to run some Turing machine whose running time can be unbounded. This is the part we will replace.

[3] This differs from the description of the algorithm in [2], but it is convenient for the way we will use their algorithm later on.

To obtain Claim 16 from the argument of [2] described above, consider a language $\mathcal{L} \in \Pi_k^{\mathsf{P}}$ where $k \geq 2$ is odd (the proof for even k, and for Σ_k^{P} with $k \geq 3$, is similar). Let M be a Π_k^{P}-Turing machine for $\mathcal{L}$, such that $(G, x) \in \mathcal{L}$ iff $\forall w_1 \exists w_2 \ldots \forall w_k M(G, x, w_1, \ldots, w_k)$ accepts. Let D be the algorithm of [2]. We construct a $\Pi_k^{\mathsf{P}\text{-}local}$-algorithm D', which interprets the certificates $c'_1, \ldots, c'_k$ that it receives as follows:

- $c'_1(v) = (c_1(v), w_1)$, where $c_1(v)$ is the universally-quantified certificate from the algorithm D of [2], and w_1 is the universally-quantified first witness of M.
- $c'_2(v) = (c_2(v), w_2)$, where $c_2(v)$ is the existentially-quantified certificate from the algorithm D of [2], and w_2 is the existentially-quantified second witness of M.
- For all $j > 2$ we interpret $c'_j(v) = w_j$, the corresponding witness of M.

The algorithm D' first verifies that all nodes receive the same witnesses $w_1, \ldots, w_k$, and that if $c_2 = \perp$ at some node, then $c_2 = \perp$ at all nodes. If $c_2 = \perp$ at all nodes, then each node runs M on the configuration (G', x') that it extracts from c_1, using the witnesses $w_1, \ldots, w_k$, and outputs the output of M. On the other hand, if $c_2 \neq \perp$, then the nodes collectively run D on the certificates c_1, c_2, and output the output of D. Correctness is similar to that of the original algorithm D: together, the certificates assert that either c_1 describes a configuration in $\mathcal{L}$, or c_1 describes a configuration that differs from the current configuration. This is equivalent to asserting that the current configuration is in $\mathcal{L}$.

The local running time of D' is equal to that of D when $c_2 \neq \perp$, plus the running time of M. Both are polynomial.

The proof of Claim 16 that we sketched above works for any class in the hierarchy that includes an alternation of the form $\forall\exists$, as this is what we require to use the algorithm from [2] (recall that it was originally intended to prove membership in the class Π_2^{local}, which has exactly this quantifier alternation). Thus, the proof applies to $\Pi_k^{\mathsf{P}\text{-}local}$ for $k \geq 2$, and to $\Sigma_k^{\mathsf{P}\text{-}local}$ for $k \geq 3$. What about $\Sigma_2^{\mathsf{P}\text{-}local}$, which has the quantifier alternation $\exists\forall$? It turns out that this class is indeed different: in the claim below, we prove that $\Sigma_2^{\mathsf{P}\text{-}local} = \Sigma_2^{local} \cap \Sigma_2^{\mathsf{P}} \subsetneq \Sigma_2^{\mathsf{P}}$. This shows that for Σ_2, there *is* a loss in expressive power compared to centralized computation: $\Sigma_2^{\mathsf{P}\text{-}local} \subsetneq \Sigma_2^{\mathsf{P}}$, unlike the classes to which Claim 16 applies.

▷ Claim 17 (Theorem 3, part 1). We have $\Sigma_2^{\mathsf{P}\text{-}local} = \Sigma_2^{\mathsf{P}} \cap \Sigma_2^{local} \subsetneq \Sigma_2^{\mathsf{P}}$.

Proof sketch. The containment $\Sigma_2^{\mathsf{P}\text{-}local} = \Sigma_2^{\mathsf{P}} \cap \Sigma_2^{local}$ is, as usual, easy to see. The other direction follows from the fact, proven in [2], that Σ_2^{local} contains only languages that are closed under lift (in fact, [2] proves that $\Sigma_2^{local} = \mathsf{NLD}$). This means we can apply a proof very similar to that of Claim 14: given a language $\mathcal{L} \in \Sigma_2^{\mathsf{P}} \cap \Sigma_2^{local}$, we can construct a $\Sigma_2^{\mathsf{P}\text{-}local}$-algorithm for $\mathcal{L}$ in the same way that we did in Claim 14, except that the algorithm is now also given a second, universally-quantified witness w', which the nodes feed to the Turing machine M along with the first witness w.

As for the claim that $\Sigma_2^{\mathsf{P}} \cap \Sigma_2^{local} \subsetneq \Sigma_2^{\mathsf{P}}$, it suffices to show that there is some language $\mathcal{L} \in \Sigma_2^{\mathsf{P}} \setminus \Sigma_2^{local}$. One such language is defined in [2]: the language EXTS, "exactly two selected", which includes all graph configurations where every node is given a Boolean input and exactly two nodes have the input 1, is not in Σ_2^{local} [2], but it is easy to see that this language is in P, and hence it is in Σ_2^{P}. ◁

References

1 Eden Aldema Tshuva and Rotem Oshman. Brief announcement: On polynomial-time local decision. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, pages 48–50, 2022. doi:10.1145/3519270.3538463.

2 Alkida Balliu, Gianlorenzo D'Angelo, Pierre Fraigniaud, and Dennis Olivetti. What can be verified locally? *Journal of Computer and System Sciences*, 97:106–120, 2018. doi:10.1016/J.JCSS.2018.05.004.

3 Michael R Fellows and Neal Koblitz. Self-witnessing polynomial-time complexity and prime factorization. *Designs, Codes and Cryptography*, 2(3):231–235, 1992. doi:10.1007/BF00141967.

4 Laurent Feuilloley and Pierre Fraigniaud. Randomized local network computing. In *Proceedings of the 27th ACM symposium on Parallelism in Algorithms and Architectures*, pages 340–349, 2015. doi:10.1145/2755573.2755596.

5 Pierre Fraigniaud, Mika Göös, Amos Korman, Merav Parter, and David Peleg. Randomized distributed decision. *Distributed Computing*, 27(6):419–434, 2014. doi:10.1007/S00446-014-0211-X.

6 Pierre Fraigniaud, Mika Göös, Amos Korman, and Jukka Suomela. What can be decided locally without identifiers? In *Proceedings of the 2013 ACM symposium on Principles of distributed computing*, pages 157–165, New York, NY, USA, 2013. ACM. doi:10.1145/2484239.2484264.

7 Pierre Fraigniaud, Magnús M Halldórsson, and Amos Korman. On the impact of identifiers on local decision. In *International Conference On Principles Of Distributed Systems*, pages 224–238, Berlin, Heidelberg, 2012. Springer. doi:10.1007/978-3-642-35476-2_16.

8 Pierre Fraigniaud, Juho Hirvonen, and Jukka Suomela. Node labels in local decision. *Theoretical Computer Science*, 751:61–73, 2018. doi:10.1016/J.TCS.2017.01.011.

9 Pierre Fraigniaud, Amos Korman, and David Peleg. Towards a complexity theory for local distributed computing. *Journal of the ACM (JACM)*, 60(5):1–26, 2013. doi:10.1145/2499228.

10 William I Gasarch. Guest column: The third P=? NP poll. *ACM SIGACT News*, 50(1):38–59, 2019. doi:10.1145/3319627.3319636.

11 O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, STOC '89, pages 25–32, 1989. doi:10.1145/73007.73010.

12 Mika Göös and Jukka Suomela. Locally checkable proofs in distributed computing. *Theory Comput.*, 12(1):1–33, 2016. doi:10.4086/TOC.2016.V012A019.

13 Joachim Grollmann and Alan L Selman. Complexity measures for public-key cryptosystems. *SIAM Journal on Computing*, 17(2):309–335, 1988. doi:10.1137/0217018.

14 Juris Hartmanis and Richard E Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

15 Lane A Hemaspaandra and Jörg Rothe. Characterizing the existence of one-way permutations. *Theoretical Computer Science*, 244(1-2):257–261, 2000. doi:10.1016/S0304-3975(00)00014-1.

16 Christopher M Homan and Mayur Thakur. One-way permutations and self-witnessing languages. *Journal of Computer and System Sciences*, 67(3):608–622, 2003. doi:10.1016/S0022-0000(03)00068-0.

17 Marcin Jurdziński. Deciding the winner in parity games is in UP∩coUP. *Information Processing Letters*, 68(3):119–124, 1998. doi:10.1016/S0020-0190(98)00150-1.

18 Ker-I Ko. On some natural complete operators. *Theoretical Computer Science*, 37:1–30, 1985. doi:10.1016/0304-3975(85)90085-4.

19 Amos Korman and Shay Kutten. Distributed verification of minimum spanning trees. *Distributed Computing*, 20(4), 2007. doi:10.1007/S00446-007-0025-1.

20 Amos Korman, Shay Kutten, and Toshimitsu Masuzawa. Fast and compact self stabilizing verification, computation, and fault detection of an mst. In *Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 311–320, 2011. doi:10.1145/1993806.1993866.

21 Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. In *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, pages 9–18, 2005. doi:10.1145/1073814.1073817.

22 Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. In *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, pages 9–18, 2005. doi:10.1145/1073814.1073817.

23 Moni Naor and Larry Stockmeyer. What can be computed locally? *SIAM Journal on Computing*, 24(6):1259–1277, 1995. doi:10.1137/S0097539793254571.

24 Fabian Reiter. A local perspective on the polynomial hierarchy. *arXiv preprint*, 2023. arXiv:2305.09538.

25 Rachid Saad. Complexity of the forwarding index problem. *SIAM Journal on Discrete Mathematics*, 6(3):418–427, 1993. doi:10.1137/0406033.

26 Leslie G Valiant. Relative complexity of checking and evaluating. *Information processing letters*, 5(1):20–23, 1976. doi:10.1016/0020-0190(76)90097-1.