Feuilloley, Laurent; Fraigniaud, Pierre; Hirvonen, Juho

A Hierarchy of Local Decision

Published in:
43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)

DOI:
10.4230/LIPIcs.ICALP.2016.118

Published: 01/01/2016

Document Version
Publisher's PDF, also known as Version of record

A Hierarchy of Local Decision*

Laurent Feuilloley[1], Pierre Fraigniaud[2], and Juho Hirvonen[3]

1 **Institut de Recherche en Informatique Fondamentale (IRIF), CNRS and University Paris Diderot, Paris, France**
laurent.feuilloley@liafa.univ-paris-diderot.fr

2 **Institut de Recherche en Informatique Fondamentale (IRIF), CNRS and University Paris Diderot, Paris, France**
pierre.fraigniaud@liafa.univ-paris-diderot.fr

3 **Helsinki Institute for Information Technology (HIIT), Department of Computer Science, Aalto University, Aalto, Finland**
juho.hirvonen@aalto.fi

Abstract

We extend the notion of *distributed decision* in the framework of distributed network computing, inspired by recent results on so-called *distributed graph automata*. We show that, by using distributed decision mechanisms based on the interaction between a *prover* and a *disprover*, the size of the certificates distributed to the nodes for certifying a given network property can be drastically reduced. For instance, we prove that minimum spanning tree can be certified with $O(\log n)$-bit certificates in n-node graphs, with just one interaction between the prover and the disprover, while it is known that certifying MST requires $\Omega(\log^2 n)$-bit certificates if only the prover can act. The improvement can even be exponential for some simple graph properties. For instance, it is known that certifying the existence of a nontrivial automorphism requires $\Omega(n^2)$ bits if only the prover can act. We show that there is a protocol with two interactions between the prover and the disprover enabling to certify nontrivial automorphism with $O(\log n)$-bit certificates. These results are achieved by defining and analysing a *local hierarchy* of decision which generalizes the classical notions of *proof-labelling schemes* and *locally checkable proofs*.

1998 ACM Subject Classification D.1.3 Concurrent Programming (Distributed programming), F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Distributed Network Computing, Distributed Algorithm, Distributed Decision, Locality

Digital Object Identifier 10.4230/LIPIcs.ICALP.2016.118

1 Introduction

This paper is tackling the long-standing issue of characterizing the power of local computation in the framework of distributed network computing [27]. Our concern is the ability to design *local* algorithms, defined as distributed algorithms in which every node of a network (i.e., every computing entity in the system) can compute its output after having consulted only nodes in its vicinity. That is, communications proceed along the links of the network, and, in a local algorithm, every node must output after having exchanged information with nodes at constant distance only. A *construction* task consists, for the nodes of a network $G = (V, E)$ where each node u is given an input $x(u)$, to collectively and concurrently compute a collection

* The first and second authors received additional supports from ANR project DISPLEXITY, and from INRIA project GANG.

43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016).
Editors: Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi;
Article No. 118; pp. 118:1–118:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$y(u)$, $u \in V$, of individual outputs, such that (G, x, y) satisfies some property characterizing the task to be solved. For instance, the minimum-weight spanning tree (MST) task consists, given the weights $x(u)$ of all the incident edges of every node u, in computing a subset $y(u)$ of edges incident to u such that the set $\{y(u), u \in V\}$ forms a MST in G. Similarly, the maximal independent set (MIS) task consists of computing $y(u) \in \{0, 1\}$, $u \in V$, such that the set $\{u \in V : y(u) = 1\}$ forms an MIS. It is an easy observation that the MST task cannot be solved locally as the weights of far-away edges may impact the output of a node. In a seminal result Linial showed that the same is true for MIS [24]: there is no local algorithm for constructing an MIS, even on an n-node ring. Nevertheless, there are many construction tasks that can be solved locally, such as approximate solutions of NP-hard graph problems (see, e.g., [8, 21, 22, 23]). In general it is Turing-undecidable whether or not a construction task can be solved locally [26].

Interestingly, the Turing-undecidability result of Naor and Stockmeyer [26] concerning the locality of construction tasks holds even if one restricts the question to properties that can be locally decided. A distributed *decision* task [1, 13] consists, given an input $x(u)$ to every node in a network G, in deciding whether (G, x) satisfies some given property. An instance is accepted by a distributed algorithm if and only if every node individually accepts (i.e., every node u outputs $y(u) = \text{true}$). For instance, proper colouring can easily be decided locally by having each node merely comparing its colour with the ones of its neighbours. On the contrary, deciding whether a collection of edges defined by $\{x(u), u \in V\}$ forms a MST is not possible locally (in fact, even separating paths from cycles is not possible locally). Similarly to the sequential computing setting, there are strong connections between the construction variant of a task and the ability to locally decide the legality of a given candidate solution for the same task, as illustrated by, e.g., the derandomization results in [6, 26], and the approximation algorithms in [29]. These connections have motivated work focusing on the basic question: what can be decided locally? This paper is aiming at pushing further our current knowledge on this question.

Two specific lines of work have motivated our approach of local decision in this paper. The first line of research is related to the notion of *proof-labelling schemes* introduced by Korman et al. [20], who showed that while not all graph properties can be decided locally, they can all be *verified* locally, with the help of local *certificates* provided by a prover. Unfortunately, there are natural graph properties (e.g., the existence of a non-trivial automorphism) which require $\Omega(n^2)$-bit certificates to be verified by any local distributed algorithm [16]. Göös and Suomela introduced the more practical class LogLCP of all graph properties that can be verified using certificates of size $O(\log n)$ bits [16], i.e., merely the size required to store the identities of the nodes. The class LogLCP contains non locally decidable properties such as hamiltonicity and non-bipartiteness. LogLCP even contains graph properties that are not in NP. Also, all existential-MSO graph properties are shown to be in LogLCP.

The second line of research which motivated our approach is the study of *distributed graph automata*. In particular, [28] recently proved that an analogue of the polynomial hierarchy, where sequential polynomial-time computation is replaced by distributed local computation, turns out to coincide with MSO. However, while this result is important for our understanding of the computational power of finite automata, the model does not quite fit with the standard model of distributed computing aiming at capturing the power of large-scale computer networks (see, e.g., [27]). Indeed, on the one hand, the model in [28] is somewhat weaker than desired, by assuming a finite-state automaton at each node instead of a Turing machine, and by assuming anonymous computation instead of the presence of unique node identities. On the other hand, the very same model is also stronger than the standard

model, by assuming a decision-making mechanism based on an arbitrary mapping from the collection of all node states to $\{\text{true}, \text{false}\}$. Instead, the classical distributed decision mechanism is based on the logical conjunction of the individual decisions. This is crucial as this latter decision mechanism provides the ability for every node rejecting the current instance to raise an alarm, and/or to launch a recovery procedure, without having to collect all of the individual decisions.

In this paper, our objective is to push further the study initiated in [16] on the LogLCP class, by adopting the approach of [28]. Indeed, LogLCP can be seen as the first level Σ_1 of a *local hierarchy* $(\Sigma_k, \Pi_k)_{k\geq 0}$, where $\Sigma_0 = \Pi_0 = \mathsf{LD}$, the class of properties that can be locally decided [13], and, for $k \geq 1$, Σ_k is the class of graph properties for which there exists a local algorithm A such that, for every instance (G, x),

$$(G, x) \text{ is legal } \iff \exists \ell_1 \forall \ell_2 \exists \ell_3 \ldots Q\ell_k : A(G, x, \ell_1, \ell_2, \ldots, \ell_k) \text{ accepts}$$

with k alternations of quantifiers, and where Q is the universal quantifier if k is even, and the existential quantifier otherwise. (Π_k is defined similarly as Σ_k, but starting with a universal quantifier). The ℓ_i's are called *labelling functions*, assigning a label $\ell_i(v) \in \{0,1\}^*$ to every node v, such that, for every node v, $|\ell_i(v)| = O(\log n)$ in n-node networks. Our aim is to analyze the local hierarchy in the general context of distributed network computing [27], where each node has an identity which is unique in the network, every node has the computational power of a Turing machine, and where the acceptance of an instance by an algorithm is defined as the logical conjunction of the individual decisions of the nodes.

1.1 Our Results

We study a hierarchy $(\Sigma_k, \Pi_k)_{k\geq 0}$ of local decision which represents a natural extension of proof-labelling scheme, as well as of locally checkable proof, with succinct certificates (i.e., of size $O(\log n)$ bits). In addition to its conceptual interest, this hierarchy might have some practical impact. Indeed, any level k of the hierarchy can be viewed as a game between a *prover* and a *disprover*, who play in turn by alternating k times. Roughly, on legal instances, the prover aims at assigning distributed certificates responding to any attempt of the disprover to demonstrate that the instance is illegal, and vice-versa on illegal instances. The referee judging the correctness of the collection of certificates produced by the players is a local distributed algorithm. For instance, the extensively studied class Σ_1 includes problems whose solutions are such that their legality can be certified by a prover using distributed certificates. Instead, the class Π_2 includes problems whose solutions are such that their legality can be certified by a prover against any other candidate solution provided by a disprover, both using distributed certificates.

We show that many problems have succinct proofs in the hierarchy. Actually, climbing up the hierarchy enables to reduce drastically the size of the certificates. For instance, we show a quadratic improvement for MST, which requires locally checkable proofs of $\Omega(\log^2 n)$ bits, while MST stands at the second level of our hierarchy. That is, there is a Π_2-protocol for MST using distributed certificates of $O(\log n)$ bits. For graph properties such as nontrivial automorphism, the improvement can even be exponential in term of certificate size, by relaxing the verification from locally checkable proofs with $\Omega(n^2)$ bits proofs to Σ_3 (with $O(\log n)$ bits proofs). More generally, many natural optimization problems are on the second level of our hierarchy. On the other hand, we also show that there are simple (Turing-computable) languages outside the local hierarchy. This latter property illustrates the impact of insisting on compact $O(\log n)$-bits certificates: there are graph properties that cannot be

locally certified via a finite number of interactions between a prover and a disprover using succinct certificates.

In addition, we prove several results regarding the structure of the hierarchy. In particular, we show that if the hierarchy collapses partially at any level, then it collapses all the way down to that level. On the other hand, we prove that the hierarchy does not collapse to the first level (i.e., the first and second levels are distinct). Distributed decision is naturally asymmetric, that is, reversing the individual decision of the algorithm at each node does not correctly reverse the global decision of the algorithm. As a consequence, it is not necessarily the case that co-$\Sigma_k = \Pi_k$, and vice-versa. However, we show that one additional level of quantifiers is always sufficient to reverse a decision (i.e., to decide the complement of a language). Finally, we show that, for every graph property at the intersection of a level-k class and the complement of this class, there is a protocol deciding that property at level k with *unanimous* decision, for both legal and illegal instances.

All our positive results hold in the classical CONGEST model (in which every edge can transmit at most $O(\log n)$ bits at each round), while all our negative results hold in the more liberal LOCAL model (in which there are no constraints on the amount of bits that can be sent through an edge at each round).

All proofs missing from this extended abstract can be found in [7].

1.2 Related Work

Several forms of "local hierarchies" have been investigated in the literature, with the objective of understanding the power of local computation, or for the purpose of designing verification mechanisms for fault-tolerant computing. In particular, as we already mentioned, [28] has investigated the case of *distributed graph automata*, where nodes are anonymous finite automata, and where the decision function is a global interpretation of all the individual outputs of the nodes. In this context, it was proved that the local hierarchy is exactly captured by the MSO formulas on graphs.

The picture is radically different in the framework in which the computing entities are Turing machines with pairwise distinct identities, and where the decision function is the logical conjunction of all the individual boolean outputs. In [13], the authors investigated the local hierarchy in which the certificates must not depend on the identity-assignment to the nodes. Under such *identity-oblivious* certificates, there are distributed languages outside Σ_1. However, all languages are in the probabilistic version of Σ_1, that is, in Σ_1 where the correctness of the verification is only stochastically guaranteed with constant probability. In [11], it is proved that Σ_1 is exactly captured by the set of distributed languages that are closed under lift. (A configuration (G', x') is a t-lift of a configuration (G, x) if there is an input-preserving mapping from $V(G')$ to $V(G)$ which preserves the t-neighbourhood of the nodes in these graphs). Interestingly, in the same framework as [13] but where the decision function is a global interpretation of the all the individual outputs, instead of the logical conjunction of individual boolean outputs, [3, 4] proved that the local hierarchy collapses to Σ_1. Also, in the same framework as [13], but where the certificates may depend on the identity assignment, all distributed languages are in Σ_1 (see [20]).

In [16], the authors proved that, to be placed in the first level Σ_1 of the local hierarchy, there are distributed languages on graphs (e.g., the existence of a nontrivial automorphism) which require to exchange certificates of size $\Omega(n^2)$ bits among neighbours, which is enough to trivially decide any problem. Similarly, [18, 20] has proved that certifying Minimum-weight Spanning Tree (MST) requires to exchange certificates on $\Theta(\log^2 n)$ bits, which can be costly in networks with limited bandwidth, i.e., under the CONGEST model [27]. In [19], it is

proved that the size of the certificates for MST can be decreased to $O(\log n)$ bits, but to the expense of $O(\log n)$ rounds of communication. Recently, [25] has proved that the amount of communication between nodes (but not necessarily the size of the certificates) for verifying general network configurations can be exponentially decreased if using randomization, and [9] analyzed in depth the certificate size for s-t connectivity and acyclicity.

It is also worth mentioning the role of the node identities in distributed decision. For instance, after noticing that the identities are leaking information to the nodes about the size of the network (e.g., at least one node has an ID at least $n-1$ in n-node network), it was recently proved that restricting the algorithms to be identity-oblivious reduces the ability to decide languages locally in Σ_0 (see [10]), while this is not the case for Σ_1 (see [11]). Recently, [12] characterized the "power of the IDs" in local distributed decision. In [5], the authors discussed what can be computed in an anonymous networks, and showed that the answer to this question varies a lot depending on the commitment of the nodes to their first computed output value, i.e., whether it is revocable or not. In the context of local decision, the output is assumed to be irrevocable.

In general, we refer to [32] for a recent survey on local distributed computing, and we refer to [14, 15] for distributed decision in the context of asynchronous crash-prone systems with applications to runtime verification, and to [2] for distributed decision in contexts where nodes have the ability to share non classical resources (e.g., intricate quantum bits).

2 Local Decision

Let $G = (V, E)$ denote an undirected graph, where V is the set of nodes, and E is the set of edges. The subgraph induced by nodes at distance (i.e., number of hops) at most t from a node v is denoted by $B_G(v, t)$. All graphs considered in this paper are assumed to be connected (for non connected graphs, our results apply separately to each connected components). The number of nodes in the graph is denoted by n. In every graph $G = (V, E)$, each node $v \in V$ is assumed to have a name from the set $\{1, \ldots, N\}$, denoted by $\mathrm{id}(v)$, where N is polynomial in n. In other words, all identities are stored on $O(\log n)$ bits. In a fixed graph, all names are supposed to be pairwise distinct.

Distributed languages. A *distributed language* L is a set of pairs (G, x), where G is a graph and x is a function that assigns some local input $x(v)$ to each node v. We assume that all inputs $x(v)$ are polynomial in n, and thus can be stored locally on $O(\log n)$ bits. The following are typical examples of distributed languages:

- 3-COLOURING: (G, x) such that x encodes a proper 3-colouring of G;
- 3-COLOURABILITY: graphs that can be properly 3-coloured;
- NTA: graphs with a nontrivial automorphism;
- PLANARITY: planar graphs.

The *complement* $\bar{L}$ of a distributed language L is defined as the set $\bar{L} = \{(G, x) : (G, x) \notin L\}$. For instance, the complement of 3-COLOURING is NON-3-COLOURING, consisting of all pairs (G, x) such that x is not a proper 3-colouring of G.

Labellings. A *labelling* ℓ is a function $\ell \colon V(G) \to \{0,1\}^*$, assigning a bit string to each node. If, for every graph G and every node $v \in V(G)$, $\ell(v) \in \{0,1\}^k$, we say that the labelling ℓ is of size k. In this paper, we are mostly interested in labellings of logarithmic size in the number of nodes in the input graph.

Local algorithms. We use the standard LOCAL model of distributed computing [27, 24]. In this model each node $v \in V(G)$ is a computational entity that has direct communication links to other nodes, represented by the edges of G. Every node runs the same algorithm. In this paper, all algorithms are deterministic. Nodes communicate with each other in synchronous communication rounds. During each round, every node is allowed to (1) send a message to each of its neighbours, (2) receive a message from each of its neighbours, and (3) perform individual computation. At some point every node has to halt and produce a local output. The number of rounds until all nodes have halted is the *running time* of an algorithm.

A *local* algorithm is a distributed algorithm A for which there exists a constant t such that, for every instance (G, x), the running time of A in (G, x) is at most t. Since the most a node can do in t communication rounds is to gather all the information available in its local neighbourhood $B_G(v, t)$, a local algorithm A can be defined as a (computable) function from all possible labelled local neighbourhoods to some output set. Given an ordered set $\bar{\ell} = (\ell_1, \ell_2, \ldots, \ell_k)$ of labellings, for some $k \geq 0$, and given an instance (G, x), we denote by $A(G, v, x, \bar{\ell})$ the output of v in algorithm A running in G with input x and labelling $\bar{\ell}$.

Local decision. In *distributed decision*, the output of each node v corresponds to its own individual decision. That is, each node either *accepts* or *rejects*. Globally, the instance (G, x) is accepted if and only if every node accepts individually. In other words, the global acceptance is subject to the logical conjunction of all the individual acceptances. For the sake of simplifying the presentation, $A(G, v, x, \bar{\ell}) = 1$ (resp., $A(G, v, x, \bar{\ell}) = 0$) denotes the fact that v accepts (resp., rejects) in an execution of algorithm A on (G, x) labelled with $\bar{\ell}$. We say that A accepts if $A(G, v, x, \bar{\ell}) = 1$ for every node $v \in V(G)$, and rejects otherwise. We will use the shorthand $A(G, x, \bar{\ell}) = 1$ to denote that $\forall v \in V(G), A(G, v, x, \bar{\ell}) = 1$, and $A(G, x, \bar{\ell}) = 0$ to denote that $\exists v \in V(G), A(G, v, x, \bar{\ell}) = 0$.

The first class in the local hierarchy considered in this paper is *local decision*, denoted by LD. A language L is in LD if there exists a local algorithm A, such that for all graphs G, and all possible inputs x on G, we have that $(G, x) \in L \iff A(G, x)$ accepts.

As an example, deciding whether x is a 3-colouring of G is in LD, but deciding whether G is 3-colourable is not. Note that LD does not refer to any labellings. The algorithm A runs solely on graphs G with possible inputs to the nodes.

Example: certifying spanning trees. In a graph G, a *spanning tree* can be encoded as a distributed data-structure x such that, for every $v \in V(G)$, $x(v)$ encodes the identity of one of v's neighbours (its parent in the tree), but one node r for which $x(r) = \perp$ (this node is the root of the tree). Deciding whether x is a spanning tree of G is not in LD. However, a spanning tree can be *certified* locally as follows (see [1, 17]). Given a spanning tree x of G rooted at node r, a *prover* assigns label $\ell(v) = (\mathrm{id}(r), d(v))$ to each node v, where $d(v)$ is the distance of v to the root r in the spanning tree x. Such a label is on $O(\log n)$ bits. The verification algorithm A at node v checks that v agrees on $\mathrm{id}(r)$ with all its neighbours, and that $d(x(v)) = d(v) - 1$. If both tests are passed, then v accepts, otherwise it rejects. It follows that Algorithm A accepts if and only if x is a spanning tree of G. Now we can also accumulate counters, such as the number of nodes in the graph, toward the root. This ability to certify spanning trees and to use them to carry information is a simple but powerful tool that will be used throughout the paper.

3 The Local Hierarchy

We generalize the various classes of distributed decision from previous work into a hierarchy of distributed decision classes, in a way analogous to the polynomial hierarchy (in particular, our class Σ_1^{LD} is equal to[1] the class logLCP introduced by Göös and Suomela [16]).

3.1 Definition

We first define an infinite hierarchy $\{(\Sigma_k^{\mathsf{LD}})_{k\geq 0}, (\Pi_k^{\mathsf{LD}})_{k\geq 0}\}$ of classes. For the sake of simplifying the notation, each of these classes is now abbreviated in Σ_k or Π_k. Informally, each class can be defined by a game between two players, called the *prover* and the *disprover*, who can assign labels to the nodes. The nodes take these labels as additional inputs when running their local algorithm A. Both players are given the language L, the instance (G, x), and the algorithm A. The goal of the prover is to make the nodes accept the instance, whereas the disprover wants it to be rejected. In Σ_k (resp., Π_k), with $k > 0$, the prover (resp., disprover) goes first, and assigns an $O(\log n)$-bit label to each node. Then, the players alternate, assigning $O(\log n)$-bit labels to each node in turn, until k labels $\ell_1, \ell_2, \ldots, \ell_k$ are assigned. A language L is in the corresponding class if there exists a local algorithm A such that, for all instances (G, x), the prover has a winning strategy if and only if $(G, x) \in L$. In other words, the algorithm is such that if $(G, x) \in L$, no matter how the disprover assigns its own labels, the prover can make A accept. Conversely, if $(G, x) \notin L$, then the disprover has a winning strategy and thus it can force A to reject. Such a combination local algorithm A and prover-disprover pair is called a *decision protocol* for L in the corresponding class. Equivalently, we define $\mathsf{LD} = \Sigma_0 = \Pi_0$, and, for $k > 0$, Σ_k is defined as the set of languages L for which there exists $c \geq 0$, and a local algorithm A such that

$$(G, x) \in L \iff \exists \ell_1 \, \forall \ell_2 \ldots \mathrm{Q}\, \ell_k, A(G, x, \ell_1, \ell_2, \ldots, \ell_k) = 1,$$

where Q is the existential (resp., universal) quantifier if k is odd (resp., even), and every label ℓ_i is of size at most $c \log n$. The class Π_k is defined similarly, except that the acceptance condition is: $(G, x) \in L \iff \forall \ell_1 \, \exists \ell_2 \ldots \mathrm{Q}\, \ell_k, A(G, x, \ell_1, \ell_2, \ldots, \ell_k) = 1$.

▶ Remark. For both Σ_k and Π_k, the equivalence should hold *for every identity-assignment* to the nodes with identities in $[1, N]$, where N is a fixed function polynomial in n. Indeed, the membership of an instance (G, x) to a language is independent of the identities given to the nodes. On the other hand, the labels given by the prover and the disprover may well depend on the actual identities of the nodes in the graph where the decision algorithm A is run. This is for instance the case of the protocol for certifying spanning trees described in the previous section, establishing that SPANNING-TREE $\in \Sigma_1$.

3.2 The odd-even collapsing and the Λ_k-hierarchy

Interestingly, the ending universal quantifier in both Σ_{2k} and Π_{2k+1} does not help. The class Π_1 turns out to be just slightly stronger than LD. Specifically, we prove the following result.

▶ **Theorem 1.** *For every $k \geq 1$, $\Sigma_{2k} = \Sigma_{2k-1}$ and $\Pi_{2k+1} = \Pi_{2k}$. Moreover, $\mathsf{LD} \subseteq \Pi_1 \subseteq \mathsf{LD}^{\#\mathrm{node}}$, that is, local decision with access to an oracle providing each node with the number of nodes in the graph.*

[1] If fact, Σ_1^{LD} is equal to LogLCP as defined by Göös and Suomela [16] when one restricts computation to be performed by Turing Machines ([16] makes no assumption on the computational power of the nodes).

Proof. The result follows from the fact that an existential quantification on labels of size $O(\log n)$ bit is sufficient to provide the nodes with the exact size of the graph. This can be certified by accumulating subtree counters along a spanning tree (see Section 2 [16]).

▶ **Claim 1.** *Let $L = \{(G, x) : \text{for every } v \in V(G),\ x(v) = |V(G)|\}$. We have that $L \in \Sigma_1$.*

We show how to use this mechanism in the case of Σ_{2k}, for $k > 0$. Let $L \in \Sigma_{2k}$, and let A be a t-round local algorithm such that:

$$(G, x) \in L \iff \exists \ell_1\, \forall \ell_2\, \ldots \exists \ell_{2k-1}\, \forall \ell_{2k}, A(G, x, \ell_1, \ell_2, \ldots, \ell_{2k}) = 1.$$

Recall that all labellings ℓ_i, $i = 1, \ldots, 2k$, are of size at most $c \log n$ for some $c \geq 0$. We construct an algorithm A' that simulates A for a protocol that does not need the last universal quantifier on ℓ_{2k}. The first labelling ℓ'_1 consists of some correct ℓ_1 for A, with the aforementioned additional label that encodes a spanning tree x' (rooted at an arbitrary node) and the value of the number of nodes in G. Regarding the remaining labellings, for each ℓ_{2i-1} assigned by the disprover, the prover assigns ℓ_{2i} as in the protocol for A, ignoring the bits padded to ℓ_1 for creating ℓ'_1. After the labellings have been assigned, each node v gathers its radius-t neighbourhood. Then, it virtually assigns every possible combination of $(c \log n)$-bit labellings $\ell_{2k}(u)$ to each node $u \in B_G(v, t)$, and simulates A at v to check whether it accepts or rejects with this labelling. If every simulation accepts, then A' accepts at v, else it rejects. Since every node generates all possible ℓ_{2k} labellings in its neighbourhood, we get that

$$(G, x) \in L \iff \exists \ell'_1\, \forall \ell_2\, \ldots \exists \ell_{2k-1}, A'(G, \ell_1, \ell_2, \ldots, \ell_{2k-1}) \text{ accepts},$$

which places L in Σ_{2k-1}. The proof of $\Pi_{2k+1} = \Pi_{2k}$ is similar by using the first existential quantifier (which appears in second position) to certify the number of nodes in the graph. For the case of Π_1, the nodes use the number of nodes provided by the oracle. ◀

A consequence of Theorem 1 is that only of the classes Σ_k for odd k, and Π_k for even k, are worth investigating.

▶ **Definition 2.** We define the classes $(\Lambda_k)_{k \geq 0}$ as follows: $\Lambda_k = \begin{cases} \Sigma_k & \text{if } k \text{ is odd;} \\ \Pi_k & \text{otherwise.} \end{cases}$

In particular, $\Lambda_0 = \Pi_0 = \mathsf{LD}$. By definition, we get $\Lambda_k \subseteq \Lambda_{k+1}$ for every $k \geq 0$, as the distributed algorithm can simply ignore the first label.

▶ **Definition 3.** The local hierarchy is defined as $\mathsf{LH} = \cup_{k \geq 0} \Lambda_k$.

3.3 Complementary classes

We define the complement classes co-Λ_k, for $k \geq 0$, as co-$\Lambda_k = \{L : \bar{L} \in \Lambda_k\}$. Note that, due to the asymmetric nature of distributed decision (unanimous acceptance, but not rejection), simply reversing the individual decision of an algorithm deciding L is generally not appropriate to decide $\bar{L}$. Nevertheless, we show that an additional existential quantifier is sufficient to reverse any decision, implying the following theorem.

▶ **Theorem 4.** *For every $k \geq 0$,* co-$\Lambda_k \subseteq \Lambda_{k+1}$.

Proof. The proof uses a spanning tree certificate to reverse the decision, in a way similar to the proof that the complement of LD is contained in logLCP (i.e., according to our terminology, co-$\Lambda_0 \subseteq \Lambda_1$) due to Göös and Suomela [16]. Let $L \in \Lambda_k$, and let A be a t-round local algorithm deciding $L \in \Lambda_k$ using labels on at most $c \log n$ bits. We construct

an algorithm A' which simulates A, but uses an additional label ℓ_{k+1} to reverse the decisions made by A. Let us assume that k is even (as it will appear clear later, the proof is essentially the same for k odd). We have that

$$(G,x) \in L \iff \forall \ell_1 \, \exists \ell_2 \, \ldots \exists \ell_k, A(G,x,\ell_1,\ell_2,\ldots,\ell_k) = 1,$$

with all labels ℓ_i's of size at most $c \log n$ for some constant $c \geq 0$. In Algorithm A', the prover and the disprover essentially switch their roles. From the above, we have

$$(G,x) \notin L \iff \exists \ell_1 \, \forall \ell_2 \, \ldots \forall \ell_k \, \exists v \in G, A(G,v,x,\ell_1,\ell_2,\ldots,\ell_k) = 0.$$

The prover for A' always follows the disprover for A, and can always pick labellings $\ell_1, \ell_3, \ldots, \ell_{k-1}$ such that there is a rejecting node if and only if $(G,x) \notin L$. In the protocol for A', the prover sets ℓ_{k+1} to be a spanning tree rooted at one such rejecting node v. Every other node $u \neq v$ simply checks that ℓ_{k+1} constitutes a proper encoding of a spanning tree, and rejects if not. If all nodes $u \neq v$ accept, then ℓ_{k+1} is indeed a proper spanning tree, and it only remains to check that v rejects in A. To this end, the node v designated as the root of the spanning tree encoded by ℓ_{k+1} gathers all labellings in its radius-t neighbourhood, and computes $A(G,x,v,\ell_1,\ell_2,\ldots,\ell_k)$. If A rejects at v, we set A' to accept at v, and, otherwise, we set A' to reject at v.

As discussed in Section 2, the spanning tree can be encoded using $O(\log n)$ bits. All labellings $\ell_1, \ell_2, \ldots, \ell_k$ have size at most $c \log n$, therefore all labels of A' are of size at most $c' \log n$ for some $c' \geq c$. The protocol is correct, as a rejecting node exists in A if and only if $(G,x) \notin L$, and A' correctly accepts in this case. If $(G,x) \in L$, then we have that, for every choice the prover can make, the disprover can always choose its labellings so that A accepts. Thus, if the spanning tree ℓ_{k+1} is correct, the root of that tree will indeed detect that it is an accepting node in A, and so reject in A'. ◀

▶ **Corollary 5.** *For every $k \geq 0$, co-$\Lambda_k \subseteq$ co-Λ_{k+1}, and $\Lambda_k \subseteq$ co-Λ_{k+1}.*

Proof. If $L \in$ co-Λ_k, then, by definition, $\bar{L} \in \Lambda_k$, and thus also $\bar{L} \in \Lambda_{k+1}$, which implies that $L \in$ co-Λ_{k+1}. If $L \in \Lambda_k$, then $\bar{L} \in$ co-Λ_k, and thus, by Theorem 4, we get that $\bar{L} \in \Lambda_{k+1}$, which implies that $L \in$ co-Λ_{k+1}. ◀

The following theorem shows that, for every $k \geq 0$, and every language L in $\Lambda_k \cap$ co-Λ_k, there is an algorithm deciding L such that an instance $(G,x) \in L$ is accepted at all nodes, and an $(G,x) \notin L$ is rejected at all nodes.

▶ **Theorem 6.** *Let $k \geq 1$, and let $L \in \Lambda_k \cap$ co-Λ_k. Then there exists a local algorithm A such that, for every instance (G,x), and for every $v \in V(G)$,*

$$(G,x) \in L \iff \begin{cases} \forall \ell_1 \, \exists \ell_2 \, \forall \ell_3 \, \ldots \exists \ell_k, A(G,v,x,\ell_1,\ldots,\ell_k) = 1 & \text{if } k \text{ is even} \\ \exists \ell_1 \, \forall \ell_2 \, \ldots \exists \ell_k, A(G,v,x,\ell_1,\ldots,\ell_k) = 1 & \text{otherwise} \end{cases}$$

In Theorem 9 in the next section, we shall see several example of languages in $\Lambda_1 \cap$ co-Λ_1, in relation with classical optimization problems on graphs. By Theorem 6, all of these languages can be decided unanimously.

3.4 Separation results

From the previous results in this section, we get that the local hierarchy $\mathsf{LH} = \cup_{k \geq 0} \Lambda_k$ has a typical "crossing ladder" as depicted on Figure 1.

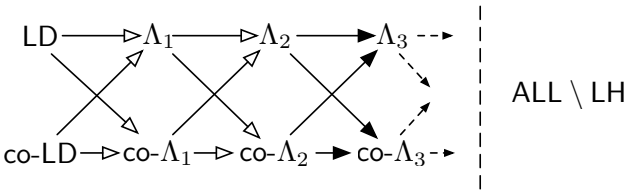

Figure 1 Structure of the local hierarchy. Arrows indicate inclusions, while hollow-headed arrows indicate strict inclusions.

In addition, we can show that some of the inclusions are strict. Indeed, it is known for long that LD is strictly included in Λ_1 (for instance, 2-COLOURABILITY $\in \Lambda_1 \setminus$ LD). Also, $\Lambda_0 \cup$ co-Λ_0 is strictly included in co-Λ_1. Indeed, for instance, NON-3-COLOURABILITY $\in$ co-$\Lambda_1 \setminus(\Lambda_0 \cup$ co-$\Lambda_0)$. Therefore, all inclusions between LD and co-LD and the classes at the first level are strict. Moreover, it is known [16] that NON-3-COLOURABILITY $\notin \Lambda_1$, implying that 3-COLOURABILITY $\notin$ co-Λ_1. On the other hand, both languages are in Λ_2, by application of Theorem 4. As a consequence, both are also in co-Λ_2. Therefore, all inclusions between the classes at the first and second levels are strict.

For $k \geq 2$, separating the classes at the kth level from the classes at the next level appears to be not straightforward. In particular, all classical counting arguments used to separate the three first levels (i.e., levels 0, 1, and 2) fail. On the other hand, we show that if $\Lambda_k = \Lambda_{k+1}$ for some k, then LH collapses to the kth level.

▶ **Theorem 7.** *If there exists $k \geq 0$ such that $\Lambda_k = \Lambda_{k+1}$, then $\Lambda_i = \Lambda_k$ for all $i > k$, that is,* LH *collapses at the kth level.*

Finally, we show that there are languages outside LH. In fact, this result holds, even if we restrict ourselves to languages with inputs 0 or 1 on oriented paths, i.e., with identity-assignment where nodes are given consecutive ID from 1 to n. The result follows from the fact that there are "only" $2^{2^{O(\log n)}}$ different local algorithms for such n-node instances at any fixed level of LH, while there are 2^{2^n} different languages on such instances.

▶ **Theorem 8.** *There exists a Turing-computable language on 0/1-labelled oriented paths that is outside* LH.

4 Positive results

In this section, we precisely identify the position of some relevant problems for distributed computing in the local hierarchy.

Optimization problems. Given an optimization problem π on graphs (e.g., finding a minimum dominating set), one defines two distinct distributed languages: the language OPT$_\pi$ (resp., ADM$_\pi$) is composed of all configurations (G, x) such that x encodes an optimal (resp., admissible) solution for π in graph G. Informally, in the context of optimization problems, the disprover aims at demonstrating that the current solution is not optimal or not admissible. Typically, the disprover does so by exhibiting a better solution or a proof of non-admissibility.

The minimum-weight spanning tree (MST) problem, which is one of the most studied problem in the context of network computing [18, 19, 20], is a typical example of optimization problems that we aim at considering in this section, but many other problems such as maximum independent set, max-cut, etc., are also of our interest. We show that, for any

optimization problem π, if deciding whether a candidate solution for π is admissible is "easy", and if the objective function for π has an additive form, then $\text{OPT}_\pi \in \text{co-}\Lambda_1$, and $\text{OPT}_\pi \in \Lambda_2$.

▶ **Theorem 9.** *Let π be an optimization problem on graphs. If the following two properties are satisfied:* (a) $\text{ADM}_\pi \in \Lambda_1 \cap \text{co-}\Lambda_1$, *and* (b) *the value to the objective function for π is the sum, over all nodes, of an individual value at each node which can be computed locally and encoded on $O(\log n)$ bits, then $\text{OPT}_\pi \in \text{co-}\Lambda_1$.*

Let us give concrete examples of problems satisfying hypotheses (a) and (b). In fact, most classical optimization problems are satisfying these hypotheses, and all the ones typically investigated in the framework of local computing (cf. the survey [32]) do satisfy (a) and (b).

▶ **Corollary 10.** *Let π be one of the following optimization problems: maximum independent set, minimum dominating set, maximum matching, max-cut, or min-cut. Then $\text{OPT}_\pi \in \text{co-}\Lambda_1$.*

The following corollary of Theorem 9 deals with two specific optimization problems, namely travelling salesman and MST. The former illustrates a significant difference between the local hierarchy defined from distributed graph automata in [28], and the one in this paper. Indeed, we show that travelling salesman is at the second level of our hierarchy, while it does not even belong to the graph automata hierarchy (as Hamiltonian cycle is not in MSO). Let TRAVELLING SALESMAN be the distributed language formed of all configurations (G, x) where G is a weighted graph, and x is an Hamiltonian cycle C in G of minimum weight (i.e., at node u, $x(u)$ is the pair of edges incident to u in C). Similarly, let MST be the distributed language formed of all configurations (G, x) where G is a weighted graph, and x is a MST T in G (i.e., at node u, $x(u)$ is the parent of u in T). Note that the case of MST is also particularly interesting. Indeed, MST is known to be in $\text{LCP}(\log^2(n))$ [18], but not in $\Lambda_1 = \text{LCP}(\log(n))$ [19]. Note also that, for MST, it is possible to trade locality for the size of the certificates, as it was established in [19] that one can use logarithmic certificates to certify MST in a logarithmic number of rounds. A consequence of Theorem 9 is the following.

▶ **Corollary 11.** $\text{MST} \in \text{co-}\Lambda_1$ *and* $\text{TRAVELLING SALESMAN} \in \text{co-}\Lambda_1$ *for weighted graphs with weights bounded by a polynomial in n.*

Non-trivial automorphism. The graph automorphism problem is the problem of testing whether a given graph has a nontrivial automorphism (i.e., an automorphism[2] different from the identity). Let NONTRIVIAL AUTOMORPHISM be the distributed language composed of the (connected) graphs that admit such an automorphism. It is known that this language is maximally hard for locally checkable proofs, in the sense that it requires proofs with size $\Omega(n^2)$ bits [16]. Nevertheless, we prove that this language is low in the local hierarchy.

▶ **Theorem 12.** $\text{NONTRIVIAL AUTOMORPHISM} \in \Lambda_3$.

Proof. The first label ℓ_1 at node u is an integer that is supposed to be the identity of the image of u by a nontrivial automorphism. Let us denote by $\phi : V(G) \to V(G)$ the mapping induced by ℓ_1. We are left with proving that deciding whether a given ϕ is a nontrivial automorphism of G is in Λ_2. Thanks to Theorem 4, it is sufficient to prove that this decision can be made in co-Λ_1. Thus let us prove that checking that (G, ϕ) is *not* a nontrivial automorphism is in Λ_1. If ϕ is the identity, then the certificate can just encode this

[2] Recall that $\phi : V(G) \to V(G)$ is an automorphism of G if and only if ϕ is a bijection, and, for every two nodes u and v, we have: $\{u, v\} \in E(G) \iff \{\phi(u), \phi(v)\} \in E(G)$.

information, and each node u checks that $\phi(u)$ is equal to its own ID. So assume now that ϕ is distinct from the identity, but is not an automorphism. To certify this, the prover assigns to each node a set of at most four spanning tree certificates, that "broadcast" to all nodes the identity of at most four nodes witnessing that ϕ is not an automorphism. Specifically, if $\phi(u) = \phi(v)$ with $u \neq v$, then the certificates are for three spanning trees, respectively rooted at u, v, and $\phi(u)$, and if $\{u, v\} \in E(G)$ is mapped to $\{\phi(u), \phi(v)\} \notin E(G)$, or $\{u, v\} \notin E(G)$ is mapped to $\{\phi(u), \phi(v)\} \in E(G)$, then the certificates are for four spanning trees, respectively rooted at $u, v, \phi(u)$, and $\phi(v)$. Checking such certificates can be done locally, and thus checking that (G, ϕ) is *not* a nontrivial automorphism is in Λ_1, and the claim follows. ◀

Problems from the polynomial hierarchy. As the local hierarchy LH is inspired by the polynomial hierarchy, it is natural to ask how their respective levels are connected. In this section, we show that some connections can indeed be established, for central problems in the polynomial hierarchy. For instance, let $k \geq 0$, and let us consider all (connected) graphs $G = (V, E)$ such that there exists $X \subseteq V$, $|X| \geq k$, such that, for every $S \subseteq X$, there is a cycle C in G containing all vertices in S, but none in $X \setminus S$. Such graphs have Cycle-VC-dimension, $\mathrm{VC}_{cycle}(G)$, at least k. Deciding whether, given G and k, we have $\mathrm{VC}_{cycle}(G) \geq k$ is Σ_3^P-complete [30, 31]. Let CYCLE-VC-DIMENSION be the distributed language composed of all configurations (G, k) such that all nodes of G have the same input k, and $\mathrm{VC}_{cycle}(G) \geq k$.

▶ **Theorem 13.** CYCLE-VC-DIMENSION $\in \Lambda_3$.

Proof. The existence of the set X can be certified setting a flag at each node in X, together with a tree T_X spanning X for proving that $|X| \geq k$. Given $S \subseteq X$, the cycle C can be certified in the same way as the Hamiltonian cycle in the proof of Corollary 11. ◀

We also show that the natural graph version QBF-SAT$_k$ of QBF-k-SAT is in Λ_k. Also, as a direct consequence of the fact that any language at level k of the distributed graph automata hierarchy [28] is at level at most $k + 1$ of LH, we get:

▶ **Theorem 14.** *All graph properties expressible in MSO are in* LH.

5 Conclusion

In this paper, we have defined and analyzed a local hierarchy LH of decision generalizing proof-labelling schemes and locally checkable proofs. Using this hierarchy, we have defined interactive local decision protocols enabling to decrease the size of the distributed certificates. We have defined the hierarchy for $O(\log n)$-bit size labels, mostly because this extends the class LogLCP in [16], and because this is consistent with the classical CONGEST model for distributed computation [27]. However, most of our results can be extended to labels on $O(B(n))$ bits, for $B(n)$ larger than $\log n$. In particular, it is worth noticing that the existence of a language L outside LH holds as long as $B = o(n)$.

The main open problem is whether LH has infinitely many levels, or whether it collapses at some level Λ_k. We know that the latter can only happen for $k \geq 2$, and thus it would be quite interesting to know whether $\Lambda_3 \neq \Lambda_2$. In particular, all the typical counting arguments used to separate Λ_2 from Λ_1, or, more generally, to give lower bounds on the label size in proof-labelling schemes or locally checkable proofs appear to be too weak for separating Λ_3 from Λ_2. A separation result for $\Lambda_3 \neq \Lambda_2$ would thus probably provide new tools and concepts for the design of space lower bounds in the framework of distributed computing.

Acknowledgements. We thank Jukka Suomela for pointing out that a counting argument can be used to find languages outside the local hierarchy, and Fabian Reiter for fruitful discussions about distributed graph automata.

References

1 Yehuda Afek, Shay Kutten, and Moti Yung. The local detection paradigm and its application to self-stabilization. *Theor. Comput. Sci.*, 186(1-2):199–229, 1997. doi:10.1016/S0304-3975(96)00286-1.

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

3 Heger Arfaoui, Pierre Fraigniaud, David Ilcinkas, and Fabien Mathieu. Distributedly testing cycle-freeness. In *Graph-Theoretic Concepts in Computer Science – 40th International Workshop, WG 2014, Nouan-le-Fuzelier, France, June 25-27, 2014. Revised Selected Papers*, pages 15–28, 2014. doi:10.1007/978-3-319-12340-0_2.

4 Heger Arfaoui, Pierre Fraigniaud, and Andrzej Pelc. Local decision and verification with bounded-size outputs. In *Stabilization, Safety, and Security of Distributed Systems – 15th International Symposium, SSS 2013, Osaka, Japan, November 13-16, 2013. Proceedings*, pages 133–147, 2013. doi:10.1007/978-3-319-03089-0_10.

5 Yuval Emek, Jochen Seidel, and Roger Wattenhofer. Computability in anonymous networks: Revocable vs. irrecovable outputs. In *Automata, Languages, and Programming – 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, pages 183–195, 2014. doi:10.1007/978-3-662-43951-7_16.

6 Laurent Feuilloley and Pierre Fraigniaud. Randomized local network computing. In *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2015, Portland, OR, USA, June 13-15, 2015*, pages 340–349, 2015. doi:10.1145/2755573.2755596.

7 Laurent Feuilloley, Pierre Fraigniaud, and Juho Hirvonen. A hierarchy of local decision. *CoRR*, abs/1602.08925, 2016.

8 Patrik Floréen, Marja Hassinen, Joel Kaasinen, Petteri Kaski, Topi Musto, and Jukka Suomela. Local approximability of max-min and min-max linear programs. *Theory Comput. Syst.*, 49(4):672–697, 2011. doi:10.1007/s00224-010-9303-6.

9 Klaus-Tycho Förster, Thomas Luedi, Jochen Seidel, and Roger Wattenhofer. Local checkability, no strings attached. In *Proceedings of the 17th International Conference on Distributed Computing and Networking, Singapore, January 4-7, 2016*, page 21, 2016. doi:10.1145/2833312.2833315.

10 Pierre Fraigniaud, Mika Göös, Amos Korman, and Jukka Suomela. What can be decided locally without identifiers? In *ACM Symposium on Principles of Distributed Computing, PODC'13, Montreal, QC, Canada, July 22-24, 2013*, pages 157–165, 2013. doi:10.1145/2484239.2484264.

11 Pierre Fraigniaud, Magnús M. Halldórsson, and Amos Korman. On the impact of identifiers on local decision. In *Principles of Distributed Systems, 16th International Conference, OPODIS 2012, Rome, Italy, December 18-20, 2012. Proceedings*, pages 224–238, 2012. doi:10.1007/978-3-642-35476-2_16.

12 Pierre Fraigniaud, Juho Hirvonen, and Jukka Suomela. Node labels in local decision. In *Structural Information and Communication Complexity – 22nd International Colloquium, SIROCCO 2015, Montserrat, Spain, July 14-16, 2015, Post-Proceedings*, pages 31–45, 2015. doi:10.1007/978-3-319-25258-2_3.

13 Pierre Fraigniaud, Amos Korman, and David Peleg. Towards a complexity theory for local distributed computing. *J. ACM*, 60(5):35, 2013. doi:10.1145/2499228.

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

15 Pierre Fraigniaud, Sergio Rajsbaum, and Corentin Travers. On the number of opinions needed for fault-tolerant run-time monitoring in distributed systems. In *Runtime Verification – 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings*, pages 92–107, 2014. doi:10.1007/978-3-319-11164-3_9.

16 Mika Göös and Jukka Suomela. Locally checkable proofs. In *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011, San Jose, CA, USA, June 6-8, 2011*, pages 159–168, 2011. doi:10.1145/1993806.1993829.

17 Gene Itkis and Leonid A. Levin. Fast and lean self-stabilizing asynchronous protocols. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 226–239, 1994. doi:10.1109/SFCS.1994.365691.

18 Amos Korman and Shay Kutten. Distributed verification of minimum spanning trees. *Distributed Computing*, 20(4):253–266, 2007.

19 Amos Korman, Shay Kutten, and Toshimitsu Masuzawa. Fast and compact self-stabilizing verification, computation, and fault detection of an MST. *Distributed Computing*, 28(4):253–295, 2015. doi:10.1007/s00446-015-0242-y.

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

21 Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. What cannot be computed locally! In *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing, PODC 2004, St. John's, Newfoundland, Canada, July 25-28, 2004*, pages 300–309, 2004. doi:10.1145/1011767.1011811.

22 Christoph Lenzen, Yvonne Anne Oswald, and Roger Wattenhofer. What can be approximated locally?: case study: dominating sets in planar graphs. In *SPAA 2008: Proceedings of the 20th Annual ACM Symposium on Parallelism in Algorithms and Architectures, Munich, Germany, June 14-16, 2008*, pages 46–54, 2008. doi:10.1145/1378533.1378540.

23 Christoph Lenzen and Roger Wattenhofer. Leveraging Linial's locality limit. In *Distributed Computing, 22nd International Symposium, DISC 2008, Arcachon, France, September 22-24, 2008. Proceedings*, pages 394–407, 2008. doi:10.1007/978-3-540-87779-0_27.

24 Nathan Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992. doi:10.1137/0221015.

25 Baruch Mor, Pierre Fraigniaud, and Boaz Patt-Shamir. Randomized proof-labeling schemes. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21-23, 2015*, pages 315–324, 2015. doi:10.1145/2767386.2767421.

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

27 David Peleg. *Distributed Computing: A Locality-Sensitive Approach*, volume 5. SIAM, 2000.

28 Fabian Reiter. Distributed graph automata. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 192–201, 2015. doi:10.1109/LICS.2015.27.

29 Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM J. Comput.*, 41(5):1235–1265, 2012. doi:10.1137/11085178X.

30 Marcus Schaefer. Deciding the Vapnik-Cervonenkis dimension in Σ_3^p-complete. *J. Comput. Syst. Sci.*, 58(1):177–182, 1999. doi:10.1006/jcss.1998.1602.

31 Marcus Schaefer and Christopher Umans. Completeness in the polynomial-time hierarchy: A compendium. *SIGACT news*, 33(3):32–49, 2002.

32 Jukka Suomela. Survey of local algorithms. *ACM Comput. Surv.*, 45(2):24, 2013. doi:10.1145/2431211.2431223.