Interactive Distributed Proofs

Gillat Kol
Princeton University
Princeton, NJ, USA
gillat.kol@gmail.coml

Rotem Oshman
Tel-Aviv University
Tel-Aviv, Israel
roshman@tau.ac.il

Raghuvansh R. Saxena
Princeton University
Princeton, NJ, USA
rrsaxena@princeton.edu

ABSTRACT

Interactive proof systems allow a resource-bounded *verifier* to decide an intractable language (or compute a hard function) by communicating with a powerful but untrusted *prover*. Such systems guarantee that the prover can only convince the verifier of true statements. In the context of centralized computation, a celebrated result shows that interactive proofs are extremely powerful, allowing polynomial-time verifiers to decide any language in PSPACE.

In this work we initiate the study of *interactive distributed proofs*: a network of nodes interacts with a single untrusted prover, who sees the entire network graph, to decide whether the graph satisfies some property. We focus on the communication cost of the protocol — the number of bits the nodes must exchange with the prover and each other. Our model can also be viewed as a generalization of the various models of "distributed NP" (proof labeling schemes, etc.) which received significant attention recently: while these models only allow the prover to present each network node with a string of advice, our model allows for back-and-forth interaction. We prove both upper and lower bounds for the new model. We show that for some problems, interaction can exponentially decrease the communication cost compared to a non-interactive prover, but on the other hand, some problems retain non-trivial cost even with interaction.

CCS CONCEPTS

• **Theory of computation → Interactive computation**; **Interactive proof systems**; **Distributed algorithms**;

ACM Reference Format:
Gillat Kol, Rotem Oshman, and Raghuvansh R. Saxena. 2018. Interactive Distributed Proofs. In *PODC '18: ACM Symposium on Principles of Distributed Computing, July 23–27, 2018, Egham, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3212734.3212772

1 INTRODUCTION

Prover-assisted computation has received significant attention from the distributed computing community [4, 10–14, 17, 20–23, 25, 26]. In this "distributed NP" setting, a powerful prover attempts to convince a set of nodes communicating over a network graph G that the graph and their inputs satisfy some property, by giving each node a short proof (usually called "advice" or "label") of this fact. The nodes then communicate briefly and decide whether to accept or reject the proof given to them; the proof is accepted iff *all* nodes accept it.

Several notions of "distributed NP" have been considered in the literature. All assume that the prover knows the entire graph topology and input, while each node sees only local information and a small part of the proof. Some problems, such as checking bipartiteness, admit very short proofs [23], but some problems essentially require the prover to give each node the entire network graph [17, 23].

One motivation for "distributed NP" is certifying the correctness of a distributed algorithm; for example, a distributed spanning tree algorithm might store, at each node, some additional information beyond the tree pointers, so that the nodes can efficiently verify that the output of the algorithm is indeed a spanning tree. In this scenario we do not have a physical prover that is a separate entity from the network — the advice is computed by the nodes themselves when they run the distributed algorithm. However, another reason to consider distributed NP is *actual* prover-assisted computation, with a powerful prover that has *more* resources than the nodes.

The setting involving a "live" powerful prover is more relevant today than ever, due to the rise of the internet, cloud-based computation, and data-intensive applications. Cloud computing allows computationally limited devices to *delegate* their costly computation to a cloud with tremendous computational power. Interactive proof systems can thus allow devices that do not trust their cloud service (as it may be malicious, motivated by self-interest, or simply buggy) to verify the correctness of the computation performed. In the same manner, interactive distributed proofs can allow central entities holding massive amounts of data (Facebook knowing the topology of a social network, 23andMe and other genetic companies having large graphs describing gene propagation, etc.) to convince their clients of some truth about the network. (For example, if the graph has good expansion, this means that small communities do not tend to segregate.)

With this motivation in mind, in this work we initiate the study of *interactive distributed proofs*, a distributed analogue of the interactive proof systems that are well-studied in complexity theory. Instead of merely having the prover give the network nodes a proof and "disappear", we allow the nodes to engage in a brief *interaction* with the prover, spanning multiple (but typically constant) rounds. We ask whether such interaction can help reduce the proof size for problems that are intractable in "distributed NP".

We follow the usual complexity-theoretic nomenclature, where the prover is called *Merlin* (because of its unbounded power) and the verifier is called *Arthur* (the mortal, with bounded resources). For example, a *distributed Arthur-Merlin protocol*, or a dAM-protocol for short, proceeds as follows: first, each network node v sends

PODC '18, July 23–27, 2018, Egham, United Kingdom

ACM ISBN 978-1-4503-5795-1/18/07…$15.00
https://doi.org/10.1145/3212734.3212772

the prover some independent random bits, R_v (a "challenge").[1] The prover responds by giving each node v a response M_v. The node then examines the responses received by itself and its immediate neighbors, $\{M_u \mid u \in N(v)\}$, together with its input and randomness R_v, and decides whether to accept or reject. The protocol accepts if and only if all nodes accept. Similarly, a dMAM protocol (*distributed Merlin-Arthur-Merlin protocol*) will start with the prover giving each node a message. This is followed by each node responding with some random bits (challenge), and the prover replying to the challenge with a second message to each node.

An interactive protocol is *correct* if for any YES-instance (i.e., any graph and input satisfying the property we are interested in), there exists a prover that has probability at least 2/3 of causing all nodes to accept; and on the other hand, for any NO-instance, no prover has probability more than 1/3 of causing all nodes to accept. The complexity measure we are interested in is the total amount of communication between any individual node and the prover. For upper bounds, we include the random bits sent to the prover in the cost; for our lower bound we do not charge for them. Unlike the complexity-theoretic setting, but in keeping with typical distributed models, the network nodes and the prover are computationally unbounded.

We note that interaction lends tremendous power to verifiers in the computational complexity setting [2, 24, 27]: adding interaction allows the verifier to decide all languages in PSPACE [24, 27], which is widely believed to be *much* larger than NP. Interactive proofs were also considered in the communication complexity setting, where it is assumed that the two communicating parties may consult a prover that knows both of their inputs [1, 18, 19]. Unfortunately, proving lower bounds in the interactive communication-complexity setting is very difficult, and for Arthur-Merlin protocols only certain restricted lower bounds are known [16].

1.1 Our Contributions

In this paper we make some first steps towards understanding the role of interaction in distributed proofs. We define interactive distributed proof systems and give positive and negative results regarding their power.

1.1.1 The Graph Symmetry Problem (Section 3). In the Graph Symmetry Problem, Sym, studied in [17], the network must decide whether the network graph has a non-trivial automorphism (see Section 2 for a formal definition). It is shown in [17] that Sym requires proofs of length $\Theta(n^2)$ at each node in the NP setting; we show that interaction is very useful for this problem.

dMAM *protocol (Section 3.1).* We give a dMAM interactive protocol in which the communication between each network node and the prover is $O(\log n)$ bits. In our protocol, the prover first "commits" to a function ρ (that is supposedly an automorphism of the graph) by giving each node its image under ρ. The next two rounds are devoted to verifying that ρ is a permutation and that ρ "respects" the edges of the graph. We show that this verification can be done with low communication using distributed linear hashing. We mention that our approach crucially utilizes the interaction, as the seed for the hash *must* be selected by the verifiers after the prover is already committed to ρ.

Theorem 1.1. $\mathsf{Sym} \in \mathsf{dMAM}[O(\log n)]$.

Exponential separation of distributed NP and distributed AM (Section 3.3). The Symmetry problem can be modified to yield an exponential separation between distributed NP and AM: if we restrict the automorphism to a *fixed* one instead of asking whether there *exists* an automorphism, we can eliminate the first round, where the prover commits to the automorphism. We call the restricted problem DSym, and observe that it is still subject to the quadratic lower bound from [17], therefore yielding:

Theorem 1.2 (informal statement). $\mathsf{DSym} \in \mathsf{dAM}[O(\log n)]$, *but any "distributed NP" scheme requires advice of length* $\Omega(n^2)$.

An interesting open problem is whether this is the largest possible gap between distributed NP and AM.

dMAM *protocol (Section 3.2).* While for restricted versions of Symmetry our dMAM protocol implies a dAM protocol with the same cost, it is unclear if this is achievable in the general case. However, we are able to show a dAM protocol for Symmetry with nearly linear cost.

Theorem 1.3. $\mathsf{Sym} \in \mathsf{dAM}[O(n \log n)]$.

We mention that in the computational complexity setting, it is known that $AM[k] = AM[2]$ for any constant k, meaning that round reduction is always possible. The same is not known for our model, and an intriguing open problem is devising general round reduction theorems.

Lower bound (Section 3.4). In [17], an $\Omega(n^2)$ lower bound for Symmetry is proven by essentially reducing from the nondeterministic two-party communication complexity of the Equality function.[2] A lower bound of $\Omega(n^2/\log(n))$ for Non-3-Colorability is proven by reduction from Set Disjointness. Unfortunately, the strategy of reducing from a hard 2-party communication complexity problem is not available to us here: proving any explicit non-trivial lower bound for two-party Arthur-Merlin communication complexity is a long-standing open problem (see for example [16]).

Instead, we leverage the distributed nature of the problem: the fact that each node can only see a small part of the input and the prover's response. We show a lower bound of $\Omega(\log \log n)$ on Arthur-Merlin interactive protocols for Symmetry, even if the nodes share an unbounded amount of randomness with the prover and with each other. (Sharing unbounded randomness with the prover amounts to allowing the nodes to send the prover messages of unbounded length "for free".) We note that such bounds cannot be obtained by simple counting arguments, as the prover's message to each of the nodes may depend on the random challenges communicated by *all* nodes.

Theorem 1.4. *If* $\mathsf{Sym} \in \mathsf{dAM}[f(n)]$, *then* $f(n) = \Omega(\log \log n)$.

We develop a framework for proving lower bounds like the one in Theorem 1.4, using a packing argument: we show how to associate

[1] We note that the restriction to challenges that are purely random is not significant if the verifiers do not have randomness hidden from the prover: the prover can see each node's input, so we can *simulate* more complicated challenges by asking the prover to compute the challenge using the node's randomness and input.

[2] The proof in [17] is not stated as a reduction, and it re-proves the hardness of Equality, but it can easily be transformed into a formal reduction from Equality.

the execution of a protocol of length L on a graph G with an $f(L)$-dimensional vector (in our case, $f(L) = 2^{2^{O(L)}}$). We prove that if the protocol is correct, then it must be able to differentiate between the different graphs, in the sense that the vectors associated with different graphs must be "far apart". Since it is not possible to pack too many "far apart" vectors into $[0, 1]^{f(L)}$, but the set of possible graphs is large, a lower bound on L is obtained. We mention that part of our argument is general and also holds for properties other than Sym. Adapting the full proof to other graph properties, and closing the gap between our dAM upper and lower bounds for Symmetry, remain interesting challenges.

1.1.2 The Graph Non-Isomorphism Problem (Section 4). Finally, we turn our attention to the famous Graph Non-Isomorphism (GNI) problem, which is in some sense the poster child for classical interactive and zero-knowledge proofs. It seems that the "standard" protocol for GNI cannot be adapted to the distributed setting (see section 4 for discussion). Instead, we develop a distributed version of the beautiful Goldwasser-Sipser protocol for GNI. This protocol was developed in [15] in the context of proving that interactive proofs do not require the verifier to have private randomness which is hidden from the prover. We show that with a few rounds of interaction (Arthur-Merlin-Arthur-Merlin), the Goldwasser-Sipser protocol can be made distributed, and obtain an $O(n \log n)$ protocol for GNI.

Theorem 1.5. GNI $\in$ dAMAM$[O(n \log n)]$.

Our main technical contribution here is a type of distributed hash function that we call *distributed almost pairwise-independent hash*. The Goldwasser-Sipser protocol crucially relies on the use of a pairwise-independent hash function with a short seed length, but the seed length is too long for us ($\Theta(n^2)$ bits), and it is not possible to "break" the seed into small parts and give each node one part without ruining the linearity of the hash[3]. Instead, we show that the requirement of pairwise-independence can be relaxed into *almost* pairwise-independence, and for the weaker requirement we construct a distributed hash function where the seed *can* be distributed, with each node contributing a small part.

We mention that it can be shown that the distributed GNI problem requires $\Omega(n^2)$ bits of advice without interaction, using an argument similar to the one used in [17] to prove the $\Omega(n^2)$ lower bound for Symmetry.

1.2 Related Work

There is a large body of work on various notions for "distributed NP" and their relative expressive power, e.g., [6, 8, 9, 20–23, 25, 26]; we refer to the excellent surveys [7, 28] for a comprehensive overview. For lack of space, we mention here only the most directly relevant work.

Our protocols rely on the spanning-tree construction from [23], which uses advice of length $\Theta(\log n)$.

Our lower bound construction for Symmetry is inspired by the dumbbell construction from [17].

In [4], a notion of *randomized proof-labeling schemes* (RPLS) is introduced, where the prover can give advice of unbounded length to each node, and the nodes then send each other randomized messages and decide whether to accept or reject. It is shown in [4] that any problem has an RPLS of exponentially-smaller length compared to the deterministic length required. However, this result is not applicable to our setting, because we do charge the prover for its communication with the nodes; the scheme from [4] increases the advice length by a factor corresponding to the maximum degree, i.e., up to linear.

In [3], the authors equip the verifiers with *alternating* Turing Machines, and classify decision tasks based on the levels of alternation (exists/forall) required. Alternation can be viewed as a type of interaction with two provers: one prover trying to convince the verifiers of the "exists" statement, and the other of its negation, a "for all" statement. However, in [3] there is no randomization, and the provers *compete* with each other. In contrast, here we have a single prover, and the interaction is randomized.

[3]Recall that our protocols for Symmetry exploit the linearity of the hash to compute the hash value in a distributed manner.

2 PRELIMINARIES

2.1 Notation

We let $N_G(u)$ denote the neighborhood of node u in graph G, including node u itself. We omit the subscript G when the graph is clear from the context. We let $\mathcal{G}(V)$ denote the set of all graphs on vertices V. We sometimes give each graph node an input from some domain $\mathcal{I}$; let $\mathcal{I}(V)$ denote the set of all assignments of inputs from $\mathcal{I}$ to the nodes in V.

When we work with a graph $G = (V, E)$ on n vertices, we use $\{0, 1\}^{m \times V}$ to denote the set of n-vectors where each coordinate is in $\{0, 1\}^m$, and the coordinates are indexed by V. We let $x_v \in \{0, 1\}^m$ denote the coordinate corresponding to v in $x \in \{0, 1\}^{m \times V}$; and for a subset $S = \{v_1, \ldots, v_k\} \subseteq V$, we let x_S denote the vector $x_{v_1} \ldots x_{v_k}$ corresponding to nodes in S. For example, if a prover's response to all nodes in the graph is $M \in \{0, 1\}^{m \times V}$, then a specific node v's response is written M_v, and the responses given to v's neighbors, including v, are $M_{N_G(v)}$.

We often conflate a set $S \subseteq X$ with its *characteristic vector* $\chi : X \to \{0, 1\}$, where $\chi_x = 1$ iff $x \in S$. We abuse notation by using the same letter to denote both the set and its characteristic vector.

2.2 Interactive Distributed Proofs

We define here one-round Arthur-Merlin proofs, the class dAM, somewhat informally.

Definition 1 (1-Round Arthur-Merlin Protocol, Informal)**.** *In a 1-round Arthur-Merlin protocol of length ℓ, each vertex v sends the prover a uniformly random challenge $r_v \in \{0, 1\}^{\ell}$. The prover P sends back a response $M_v \in \{0, 1\}^{\ell}$, which is a function of the graph G, the input I, and the random challenges $\{r_v \mid v \in V\}$.*

The protocol specifies a decision function out_v *for each $v \in V$, which takes node v's neighborhood and input, the random challenges of node v and its neighbors, and the response issued to node v and its neighbors, and outputs a Boolean value. We denote:*

- $\mathrm{out}_v^{\Pi}(G, I, r, M) = \mathrm{out}_v^{\Pi}(N_G(v), I(v), r_{N_G(v)}, M_{N_G(v)})$, *the decision value at node v when the graph is G, the input is I, the random challenge is r and the prover's response is M;*
- $\Pi(P, G, I, v)$, *the random variable representing v's decision when it interacts with a prover P.*

We remark that our upper bounds do not need the nodes to see the randomness of their neighbors, $r_{N_G(v)}$, only their own randomness r_v. Our lower bound does give $r_{N_G(v)}$ to each node v.

Definition 2 (The class $\mathsf{dAM}[\ell]$)**.** *A language $\mathcal{L} \subseteq \mathcal{G}(V) \times \mathcal{I}(V)$ is in the class $\mathsf{dAM}[\ell]$, for $\ell \in \mathbb{N}$, if there is a protocol Π of length ℓ satisfying:*

$$(G, I) \in \mathcal{L} \implies \exists P, \Pr(\textit{all nodes accept}) > 2/3$$
$$(G, I) \notin \mathcal{L} \implies \forall P, \Pr(\textit{all nodes accept}) < 1/3.$$

Sometimes we are interested in "pure graph properties", where there is no input other than the network graph. In this case we omit the input I from all our notation.

In Section 3.1 we give a *Merlin-Arthur-Merlin (*dMAM*)* protocol. Here, the prover goes first, and gives each node v a response M_v^1. The nodes then send a random challenge r_v, and the prover sends back a second response M_v^2. Finally, each node outputs a decision, depending on its input, and the randomness and responses of itself and its neighbors. Our correctness requirement is the same as for dAM.

We assume for simplicity that we have a fixed set of nodes V, known in advance to all the participants. However, our upper bounds generalize in a straightforward manner to the case where we have some polynomially-large namespace N, and we draw n nodes from N.

Describing interactive protocols. When we specify an interactive protocol, we specify two types of interaction: messages sent by the prover to the nodes, and messages sent by the nodes to the prover. Messages from nodes to the prover are indicated by $\boldsymbol{v \to P}$, followed by the contents of the message. Here, v is the identifier of the sending node.

For convenience, we use two types of prover responses:

- *Broadcast* messages, where the prover is required to provide all nodes with the same value, and the nodes verify that indeed they all got the same value. This type of message is indicated by $\boldsymbol{P \to v}$ **(Broadcast)**, followed by the contents of the message. We assume implicitly that each node compares the response it received to the responses its neighbors received, and rejects if some neighbor received a different response.
- *Unicast* messages, where the prover can give a different value to each node. This type of message is indicated by $\boldsymbol{P \to v}$ **(Unicast)**, followed again by the contents.

2.3 Problem Definitions

We consider the following problems:

Definition 3 (Graph Symmetry)**.** *The language* Sym *is the set of all symmetric graphs, that is, graphs that have a non-trivial automorphism.*

(A permutation $\rho : V \to V$ is an *automorphism* of $G = (V, E)$ if for every $u, v \in V$ we have $\{u, v\} \in E$ iff $\{\rho(u), \rho(v)\} \in E$. A *non-trivial* automorphism is an automorphism that is not the identity function.)

We formulate a distributed version of Graph Non-Isomorphism, as follows:

Definition 4 (Graph Non-Isomorphism)**.** *The language* GNI *is the set of all pairs of graphs (G_0, G_1) such that G_0 is not isomorphic to G_1.*

Here, the network graph is the graph G_0, and the second graph G_1 is given as inputs to the nodes, with each node v receiving its neighborhood $N_{G_1}(v)$. An alternative definition, which our algorithm also solves, is the following: we have only one graph, the network graph G. Each node in the graph is marked with an input from $\{0, 1, \perp\}$, and the goal is to determine whether the subgraph induced by the nodes marked 0 is not isomorphic to the subgraph induced by the nodes marked 1. The version we formulated in Definition 4 is a little stronger than this second version, because it does not allow nodes to communicate over the edges of the second graph G_1.

3 INTERACTIVE DISTRIBUTED PROOFS FOR THE SYMMETRY PROBLEM

3.1 An $O(\log n)$-Bit dMAM Protocol

Recall that Sym is the class of all graphs that have a non-trivial automorphism. To verify that $G \in \mathsf{Sym}$, we ask the prover to provide a non-trivial automorphism $\rho : V \to V$ of the graph, by giving each node $v \in V$ its image $\rho(v)$ under the automorphism. We then *verify* that ρ is indeed a non-trivial automorphism, that is, ρ is not the identity function, and ρ is an automorphism of G.

Verifying that ρ is not the identity function is easy: we ask the prover to broadcast (i.e., tell all the nodes) a node $v \in V$ such that $\rho(v) \neq v$, and node v then checks that indeed $\rho(v) \neq v$.

To check that ρ is an automorphism of G we use the *adjacency matrix* of G, a matrix A_G whose rows and columns are indexed by nodes $v \in V$, and where $A_G(u, v) = 1$ iff $\{u, v\} \in E$. Intuitively, ρ is an automorphism of G iff when we permute the rows and columns of A_G according to ρ, we get the same matrix we started with: $(u, v) \in E$ iff $(\rho(u), \rho(v)) \in E$. Let $\rho(A_G)$ denote the ρ-permuted version of A_G, where the v-th row is replaced by the $\rho(v)$-th row for each $v \in V$, and similarly for the columns.

No single node of G knows the entire adjacency matrix A_G, but each node v knows its row in A_G. Moreover, since v sees all the images $\rho(u)$ of its neighbors $u \in N_G(v)$, it can compute row $\rho(v)$ of the ρ-permuted matrix, $\rho(A_G)$. We exploit this to check whether $A_G = \rho(A_G)$.

3.1.1 The Permuted Adjacency Matrix. For a vector $r \in \{0, 1\}^n$, we denote by $[i, r]$ the $n \times n$ boolean matrix where the i-th row is r, and the rest of the matrix is 0. This is a convenient representation: for any $n \times n$ boolean matrix A, if the rows of A are $r_1, \ldots, r_n \in \{0, 1\}^n$, then $A = \sum_{i=1}^n [i, r_i]$. In particular, for a graph G, the adjacency matrix of G can be written as $A_G = \sum_{v \in V} [v, N(v)]$, where here we think of the neighborhood $N(v)$ as represented by a vector $N(v) \in \{0, 1\}^V$ (with self-loops for all vertices).

Given a function $\rho : V \to V$ and a subset $S \subseteq V$, let $\rho(S) = \{\rho(s) \mid s \in S\}$. Equivalently, when S is represented as the characteristic vector $S \in \{0, 1\}^V$, then $\rho(S)$ is the characteristic vector where $\rho(S)_v = 1$ iff there exists u such that $\rho(u) = v$ and $S_u = 1$.

Our protocol requires the prover to commit to an automorphism of G. To verify that the prover-provided mapping is an automorphism, we rely on the following observation:

Observation 1. A permutation $\rho : V \to V$ is an automorphism of G iff for all $v \in V$ we have $N(\rho(v)) = \rho(N(v))$.

Proof. By definition, ρ is an automorphism iff for every $u, v \in V$, it holds that $u \in N(v)$ iff $\rho(u) \in N(\rho(v))$. Since ρ is a permutation, this is equivalent to saying that for every $u, v \in V$, $\rho^{-1}(u) \in N(v)$ iff $u \in N(\rho(v))$, that is, $\rho(N(v)) = N(\rho(v))$. □

As a corollary, we can show:

Lemma 3.1. *Let $\rho : V \to V$ be a function such that $\sum_{v \in V}[v, N(v)] = \sum_{v \in V}[\rho(v), \rho(N(v))]$. Then ρ is an automorphism of G.*

Proof. First, we prove that ρ is a permutation on V. Assume for contradiction that ρ is not a permutation. In this case, there exists a $v^* \in V$ such that v^* is not in the range of ρ, *i.e.*, $\rho(v) \neq v^*$ for any $v \in V$. Thus, in the sum $\sum_{v \in V}[v, N(v)]$, the row corresponding to v^* is non-zero, as $N(v^*)_{v^*} = 1$. On the other hand, the row corresponding to v^* in the sum $\sum_{v \in V}[\rho(v), \rho(N(v))]$ is the all zero row, a contradiction.

Having established that ρ is a permutation on V, it is easy to see that ρ is an automorphism of G: for each $u \in V$, the row corresponding to $\rho(u)$ in the sum $\sum_{v \in V}[v, N(v)]$ is $N(\rho(u))$, and in the sum $\sum_{v \in V}[\rho(v), \rho(N(u))]$ the row corresponding to $\rho(u)$ is $\rho(N(u))$. Therefore these two rows must be equal for every $u \in V$. By Observation 1, this means that ρ is an automorphism of G. □

We see that to check if ρ is an automorphism, it suffices to verify that $\sum_{v \in V}[v, N(v)] = \sum_{v \in V}[\rho(v), \rho(N(v))]$. However, we can not afford to have nodes send their entire rows, because this would require $O(n)$ bits per node in the worst case. Instead, we use *linear hash functions* to hash the rows of A_G and $\rho(A_G)$, sum up the hashes for the rows of each matrix, and verify that the sums are equal.

3.1.2 *Hashing the Adjacency Matrix.*

A linear hash function allows us to hash a composite object, $x_1 + x_2$, by hashing its building blocks, x_1 and x_2; in our case, we can hash the entire adjacency matrix by having each node hash its own row.

A small linear hash function family is known to exist:

Theorem 3.2. *Let $m, p \in \mathbb{N}$, where p is prime. There exists a family $\mathcal{H}$ of functions $h : \{0,1\}^m \to \{0, \dots, p-1\}$ of size $|\mathcal{H}| = p$ such that for $x, x' \in \{0,1\}^m$, we have the following properties:*

(1) ***Linearity:*** *$h(x + x') = h(x) + h(x')$.*
(2) ***Small collision probability:*** *If $x \neq x'$ then*

$$\Pr_{h \in \mathcal{H}}\left(h(x) = h(x')\right) \leq \frac{m}{p}.$$

Note that in (1) above, the sum on the right-hand is modulo p, and the sum on the left-hand side is defined as $x + x' = (x_1 + x'_1 \ (mod\ p), \dots, x_m + x'_m \ (mod\ p))$.

We always use Theorem 3.2 to hash $n \times n$ boolean matrices, so we always take $m = n^2$. Note that since $|\mathcal{H}| = p$, picking a random hash function from $\mathcal{H}$ only requires $O(\log p)$ random bits.

Unless stated otherwise, let $\mathcal{H}$ be the hash function family given by Theorem 3.2, for some prime $p \in [10n^3, 100n^3]$. (Such a prime exists by Bertrand's Postulate / Chebyshev's Theorem.) Picking a random hash function from $\mathcal{H}$ only requires $O(\log n)$ random bits ("the seed").

3.1.3 *The Protocol.*

Now we are ready to describe our dMAM protocol for Symmetry; see the formal description in Protocol 1. As a building-block, we use the spanning tree proof labeling scheme from [23], and the prover helps the nodes "sum their hash values up the tree".

In the first round, the prover presents each node $v \in V$ with: the value $\rho_v \in V$, the parent $t_v \in V$ of v in the spanning tree, the root $r \in V$ of the tree, and v's distance $d_v \in [n]$ from the root, in (what the prover claims is) a spanning tree rooted at a node r such that $\rho_r \neq r$.

The nodes verify that indeed they received a spanning tree, in the standard way [23], and that they agree on the root of the tree. The root r verifies that $\rho_r \neq r$, to ensure that the claimed automorphism is not trivial.

Let $\rho : V \to V$ be defined by $\rho(v) = \rho_v$ (where ρ_v is the value the prover gave to v). Next, we want to verify that $\sum_{v \in V}[v, N(v)] = \sum_{v \in V}[\rho(v), \rho(N(v))]$, by having the nodes agree on a hash function and apply it to both sides. However, the hash function needs to be random, and all nodes need to know it; since the nodes do not have shared randomness, we use the prover to "simulate" it. The root of the tree selects a random hash function $h_i \in \mathcal{H}$ and sends its index to the prover; the prover then gives h_i to all the nodes, and they verify that they all received the same h_i. The prover also assists the nodes in applying the linear hash function "up the tree": to each node v, if T_v is the subtree rooted at v (in the spanning tree the prover provided), the prover gives v two values $a_v, b_v \in [p]$, which it claims are the two hash values:

$$a_v = h_i(\sum_{u \in T_v}[u, N(u)]), \qquad b_v = h_i(\sum_{u \in T_v}[\rho(u), \rho(N(u))]). \quad (1)$$

To verify that the prover computed the hash correctly, each node sums its children's hash values and adds the hash of its neighborhood, and verifies that the sum matches the value it got from the prover.

In Protocol 1, each of the nodes exchanges a total of $O(\log n)$ bits with the prover. The rest of this subsection is devoted to proving the correctness of the protocol.

Lemma 3.3. *Suppose that all vertices in the graph decided to accept. Then, $a_r = h_i\left(\sum_{v \in V}[v, N(v)]\right)$ and $b_r = h_i\left(\sum_{v \in V}[\rho(v), \rho(N(v))]\right)$.*

Proof. Due to Line 1, there is an edge between v and t_v for all $v \neq r$. Let T be the subgraph formed by these edges. Then T is a spanning tree rooted at r; this part is standard, so we omit the proof.

We show by induction on the height of node v in T that $a_v = \sum_{u \in T_v} h_i([u, N(u)])$, and $b_v = \sum_{u \in T_v} h_i([\rho(u), \rho(N(u))])$.

The base case is immediate: leaves have no children, so their verification step (Line 3) verifies the claim. As for the step, suppose that the claim holds for all children of v: for each $u \in C(v)$, $a_u = \sum_{w \in T_u} h_i([w, N(w)])$, and similarly for b_u. Then in v's verification

Protocol 1 A dMAM protocol for Sym

$P \to v$ **(Broadcast)**: The prover sends a vertex r (claimed spanning tree root).

$P \to v$ **(Unicast)**: The prover sends:
the identifier ρ_v (claimed image under the automorphism);
the identifier t_v (claimed parent);
and a number $d_v \in [n]$ (claimed distance from the root).

$v \to P$: Each node sends a random index $i_v \in [|\mathcal{H}|]$.

$P \to v$ **(Broadcast)**: The prover sends an index $i \in [|\mathcal{H}|]$ (claimed hash index sent by the root).

$P \to v$ **(Unicast)**: The prover sends hash values $a_v, b_v \in [p]$.

The verification procedure:

1: If $v \neq r$, verify that $t_v \in N(v)$ and $d_{t_v} = d_v - 1$.
If $v = r$, verify that $d_v = 0$.

2: Let $C(v) = \{u \in N(v) \mid t_u = v\}$.

3: v verifies that

$$a_v = h_i([v, N(v)]) + \sum_{u \in C(v)} a_u,$$
$$b_v = h_i([\rho_v, N_\rho(v)]) + \sum_{u \in C(v)} b_u.$$

Here, $N_\rho(v)$ is the vector whose v^{th} coordinate is 1 iff there exists $c \in C(v)$ such that $\rho_c = v$.

4: If $v = r$, verify that $a_r = b_r$, $\rho_r \neq r$, and $i = i_r$.

step, it verifies that

$$\begin{aligned} a_v &= h_i([v, N(v)]) + \sum_{u \in C(v)} a_u \\ &= h_i([v, N(v)]) + \sum_{u \in C(v)} \sum_{w \in T_u} h_i([w, N(w)]) \\ &= \sum_{w \in T_v} h_i([w, N(w)]) = h_i\left(\sum_{w \in T_v} [w, N(w)]\right), \end{aligned}$$

where the last step is by linearity of h_i. The verification for b_v is similar. □

Theorem 3.4. *Algorithm 1 satisfies the followings:*

$$\begin{aligned} G \text{ is symmetric} &\implies \exists P, \Pr(\text{all nodes accept}) > 2/3 \\ G \text{ is not symmetric} &\implies \forall P, \Pr(\text{all nodes accept}) < 1/3. \end{aligned}$$

Proof. Consider first the case where G is symmetric, and let ρ be a non-trivial automorphism of G. Let $r \in V$ be some vertex such that $\rho(r) \neq r$. An "honest" prover can convince all nodes to accept, by specifying:

- Any spanning tree T rooted at r. The prover gives each vertex v its image under ρ, its parent in T, its distance from r, and the name of r.
- After receiving the random indices from each vertex, the prover responds with the index $i = i_r$ sent by the root. It then gives each vertex v the values a_v, b_v computed according to (1).

It is easy to verify that these responses pass the verification procedure.

Now suppose that G is *not* symmetric, and fix a prover P. We show that the probability that P convinces all nodes to accept is less than $1/3$. Specifically, we show that for any choice of root r_0, if the prover specifies r_0 as the root, then the probability it can convince all nodes to accept is less than $1/3$;

To that end, consider a fixed root $r_0 \in V$. By Lemma 3.3, if all vertices decided to accept we have

$$a_{r_0} = h_i\left(\sum_{v \in V} [v, N(v)]\right) \quad \text{and} \quad b_{r_0} = h_i\left(\sum_{v \in V} [\rho(v), \rho(N(v))]\right).$$

If G is not symmetric, then in particular, the mapping ρ provided by the prover in the first round is not a non-trivial automorphism. We know that ρ is not the identity mapping, because $\rho(r_0) \neq r_0$ (otherwise the root rejects). Therefore, it is not an automorphism, and by Theorem 3.1, we must have $\sum_{v \in V} [v, N(v)] \neq \sum_{v \in V} [\rho(v), \rho(N(v))]$. However, the root only accepts if $a_{r_0} = b_{r_0}$. Due to Theorem 3.2 and by the selection of the parameters of $\mathcal{H}$, it holds that if $\sum_{v \in V} [v, N(v)] \neq \sum_{v \in V} [\rho(v), \rho(N(v))]$, then the probability of the event $a_{r_0} = b_{r_0}$ is at most $1/3$. The reason is that the hash indices i_v were sent to the prover after the prover already committed to r and ρ_v. Therefore, $i = i_r$ is a random hash index that is only known to the prover after the prover is already committed to the values $\sum_{v \in V} [v, N(v)]$ and $\sum_{v \in V} [\rho(v), \rho(N(v))]$.

□

3.2 An $O(n \log n)$-Bit dAM Protocol

Let us now leverage the ideas from the previous section to obtain a dAM protocol for Symmetry. The prover in dAM has more power than it does in dMAM: in dAM, we cannot force the prover to *commit* to the permutation ρ before the random challenge is issued. To deal with the extra difficulty, our dAM protocol is longer, and uses $O(n \log n)$ bits. We use the extra bits first, we ask the prover to give each vertex v the full automorphism $\rho : V \to V$, not just its own value $\rho(v)$. Second, we use a much longer hash, to make the probability of collision so small that we can union-bound over all possible mappings $V \to V$.

Other than these two changes, the protocol is similar to the dMAM protocol, but the random challenge is now issued before the prover provides the spanning tree and the permutation (see formal description in Protocol 2). The protocol uses the hash function family given by Theorem 3.2 for some prime $p \in [10n^{n+2}, 100n^{n+2}]$. Picking a random hash function from $\mathcal{H}$ requires $O(n \log n)$ random bits.

In Protocol 2, each of the nodes exchanges $O(n \log n)$ bits with the prover. The rest of this subsection is devoted to proving the correctness of the protocol.

Observe that Lemma 3.3 continues to hold: its correctness does not depend on when the random challenge i_r is issued, whether it is before or after the prover specifies the spanning tree and the permutation.

Theorem 3.5. *Algorithm 2 satisfies the followings:*

$$\begin{aligned} G \text{ is symmetric} &\implies \exists P, \Pr(\text{all nodes accept}) > 2/3 \\ G \text{ is not symmetric} &\implies \forall P, \Pr(\text{all nodes accept}) < 1/3. \end{aligned}$$

Proof. If G is symmetric, an honest prover can convince all nodes to accept, much the same as it did in the dMAM protocol.

Protocol 2 A dAM protocol for Sym

$\boldsymbol{v} \to \boldsymbol{P}$: Each node sends a random index $i_v \in [|\mathcal{H}|]$.

$\boldsymbol{P} \to \boldsymbol{v}$ **(Broadcast)**: a permutation $\rho : V \to V$, an index $i \in [|\mathcal{H}|]$ and a vertex $r \in V$.

$\boldsymbol{P} \to \boldsymbol{v}$ **(Unicast)**: an identifier $t_v \in V$, a number $d_v \in [n]$, and two hash values $a_v, b_v \in [p]$.

The verification procedure:

1: If $v \neq r$, verify that $t_v \in N(v)$ and $d_{t_v} = d_v - 1$.
If $v = r$, verify that $d_v = 0$.

2: Let $C(v) = \{u \in N(v) \mid t_u = v\}$.

3: v verifies that

$$a_v = h_i([v, N(v)]) + \sum_{u \in C(v)} a_u,$$

$$b_v = h_i([\rho(v), \rho(N(v))]) + \sum_{u \in C(v)} b_u.$$

4: If $v = r$, verify that $a_r = b_r$ and $\rho(r) \neq r$ and $i = i_r$.

Suppose that G is not symmetric, and fix a root r_0. Since G is not symmetric, by Lemma 3.1, for any mapping $\sigma : V \to V$ that is not the identity, we have $\sum_{v \in V}[v, N(v)] \neq \sum_{v \in V}[\sigma(v), \sigma(N(v))]$. However, by Lemma 3.3, the root only accepts if

$$h_i\left(\sum_{v \in V}[v, N(v)]\right) = h_i\left(\sum_{v \in V}[\rho(v), \rho(N(v))]\right).$$

We show that for a random $h_i \in \mathcal{H}$, the probability that there *exists* a mapping $\sigma : V \to V$ that is not the identity and will make the verifiers accept is small:

$$\Pr_{h_i \in \mathcal{H}}\left(\exists \sigma : \; h_i\left(\sum_{v \in V}[v, N(v)]\right) = h_i\left(\sum_{v \in V}[\sigma(v), \sigma(N(v))]\right)\right) < \frac{1}{3}.$$

Since by Line 4, $\rho(r) \neq r$, thus ρ is not the identity, it follows that the prover can only convince all nodes to accept with probability less than 1/3.

Consider a specific mapping $\sigma : V \to V$ that is not the identity. By universality,

$$\Pr_{h_i \in \mathcal{H}}\left(h_i\left(\sum_{v \in V}[v, N(v)]\right) = h_i\left(\sum_{v \in V}[\sigma(v), \sigma(N(v))]\right)\right) \leq \frac{1}{3n^n}.$$

A union bound across all n^n mappings yields the claim. □

3.3 An Exponential Separation between Distributed NP and Distributed AM

In the previous sections, we gave a dMAM protocol for Symmetry using $O(\log n)$ bits, and an dAM protocol using $O(n \log n)$ bits. Symmetry is known to require $\Omega(n^2)$ bits with no interaction [17] even in the relatively powerful Locally Checkable Proof model. We now define a simple variant, which we call *Dumbbell Symmetry*, for which we can give an $O(\log n)$-bit AM protocol. The variant is still subject to the $\Omega(n^2)$ lower bound from [17], so this shows an exponential separation between one-round distributed interactive proofs and LCPs. In fact, the definition of Dumbbell Symmetry is directly inspired by the lower bound of [17]: it is the Symmetry problem, but restricted to symmetric dumbbell graphs with a fixed isomorphism mapping one side of the dumbbell to the other.

For the sake of concreteness, let us restrict attention to the case of LCPs with local horizon 1, that is, each node sees only the advice of itself and its immediate neighborhood. (Extending to any constant local horizon, even one which is not known in advance, is easy.)

Theorem 3.6. *There is a language* DSym *of $O(n)$-vertex graphs, which is in* dAM$[O(\log n)]$*, but does not have a Locally Checkable Proof with local horizon 1 and length $\Omega(n^2)$.*

Definition 5 (Dumbbell Symmetry). *The language* DSym *is the set of $(2n+2r+1)$-vertex graphs $G = (\{0, \ldots, 2n+2r\}, E)$ which have the following structure:*

- *Let F_0, F_1 be the subgraphs induced by G on vertices $\{0, \ldots, n-1\}$ and $\{n, \ldots, 2n-1\}$, respectively. Then the mapping $\sigma'(x) = x + n$ is an isomorphism from F_0 to F_1.*
- *The subgraphs F_0, F_1 are connected to each other by a path using the remaining vertices,*

$$0 - (2n) - (2n+1) - \ldots - \ldots (2n+2r) - n.$$

- *G contains no edges other than the internal edges of F_0, F_1 and the path.*

In [17] it is shown that DSym requires $\Omega(n^2)$ bits for Locally Checkable Proofs, even if each node can see the advice of its entire r-neighborhood.

We next show that DSym can be decided by a dAM protocol using $O(\log n)$ bits. By definition of DSym, we have $G \in$ DSym iff all the following conditions hold:

(1) The following mapping is an automorphism of G:

$$\sigma(x) = \begin{cases} x + n & \text{if } x \in \{0, \ldots, n-1\}, \\ x - n & \text{if } x \in \{n, \ldots, 2n-1\}, \\ 2n + 2r - (x - 2n) & \text{if } x \in \{2n, \ldots, 2n+r\}, \\ 2n + (2n + 2r - x) & \text{if } x \in \{2n+r+1, \ldots, 2n+2r\}. \end{cases}$$

(the bottom two cases map $2n \to 2n+2r$, $2n+1 \to 2n+2r-1, \ldots$)

(2) The path $0 - (2n) - (2n+1) - \ldots - \ldots (2n+2r) - n$ is present in G.

(3) G contains no edges other than the internal edges in $\{0, \ldots, n-1\}$ and in $\{n, \ldots, 2n-1\}$, and the path edges.

The second and third conditions can be verified locally without the prover's assistance: each path node verifies that it indeed has edges to both its neighbors on the path, and all nodes verify that they have no edges that are not path edges or internal to $\{0, \ldots, n-1\}$ or to $\{n, \ldots, 2n-1\}$. To verify the first condition, we use Protocol 1 from Section 3.1, but now the prover does not need to commit upfront to the mapping ρ, because we need only consider one specific automorphism, σ. This eliminates the first Merlin round, and yields an Arthur-Merlin protocol for DSym.

3.4 Lower Bound

In this section we show a lower bound of $\Omega(\log \log n)$ on the communication cost of every dAM protocol for Sym. For our lower bound, we require a large family $\mathcal{F}$ of graphs on vertices $\{1, \ldots, n\}$, with the property that all graphs in $\mathcal{F}$ are asymmetric (i.e., they have no non-trivial automorphisms), and no two graphs in $\mathcal{F}$ are isomorphic to each other. It is known that for sufficiently large n, there is such a class $\mathcal{F}$ of size $\Omega(2^{n^2}/n!) = \Omega(2^{n^2 - n\log n}) = \Omega(2^{n^2})$.

Our lower bound for the Symmetry problem uses a graph family $\mathcal{G}$ which is similar to the family used for the Symmetry lower bound in [17]. Each graph in $G \in \mathcal{G}$ is defined by $G = G(F_A, F_B)$, where $F_A, F_B \in \mathcal{F}$ are not necessarily distinct. We fix three disjoint sets: $V_A = \{u_1^A, \ldots, u_n^A\}$, $V_B = \{u_1^B, \ldots, u_n^B\}$, and a *bridge*, consisting of two nodes $x_A, x_B \notin V_A \cup V_B$. We also fix two nodes $v_A \in V_A, v_B \in V_B$. The nodes x_A, x_B are referred to as *bridge nodes*.

Each graph $G = G(F_A, F_B) \in \mathcal{G}$ is defined as follows: on the nodes in V_A and V_B respectively, we construct copies of F_A, F_B (resp.) by replacing node i of V_A (resp. V_B) with u_i^A (resp. u_i^B). We also add edges $\{v_A, x_A\}, \{x_A, x_B\}, \{x_B, v_B\}$. This yields a "dumbbell graph", with copies of F_A and F_B on either side, connected by a short "bridge". Crucially, $G(F_A, F_B)$ has a non-trivial automorphism iff $F_A = F_B$.

Fix a dAM protocol Π, and let L be the maximum length of the prover's response to any node. We prove the lower bound in two steps. First we show that any dAM protocol can be transformed into what we call a *simple* protocol, where the two bridge nodes always expect to get the same message, and their decision whether to accept or reject is based only on this (shared) message and the randomness. This makes it easier to focus on the view of these bridge nodes. Then we show that the distribution of the message the bridge nodes receive must "encapsulate" all the information about the two sides F_A and F_B of the graph. This distribution, which can be represented as a $d = 2^{2^{O(L)}}$-dimensional vector, must be "far apart" for different graphs $G(F_A, F_B)$. Since it is not possible to pack too many "far apart" vectors into $[0,1]^{2^{2^{O(L)}}}$, but the class $\mathcal{G}$ does contain many different graphs ($|\mathcal{G}| = 2^{\Omega(n^2)}$), we get that L must be at least $\Omega(\log\log n)$.

We begin by defining *simple protocols*, where we restrict the output function for the bridge nodes, so that they only accept if they both receive the same response from the prover. In this definition and the rest of this section, we implicitly assume that all graphs are from the family $\mathcal{G}$.

Definition 6 (Simple protocols)**.** *A* dAM *protocol* Π *is said to be* simple *if there exist* $f_{x_A}, f_{x_B} : \{0,1\}^L \times \{0,1\}^L \to \{0,1\}$ *such that*

(1) For all graphs G, challenges R and responses M,

$$\mathrm{out}_{x_A}(G, R, M) = 1 \Leftrightarrow \left[f_{x_A}(R_{N(x_A)}, M_{x_A}) = 1 \text{ and } M_{x_A} = M_{x_B} \right].$$

(2) Similarly for x_B.

We use the short-hand notation $f_{x_A}(R, M)$ for $f_{x_A}(R_{N(x_A)}, M_{x_A})$. It is not hard to see that any dAM[1] protocol for the class $\mathcal{G}$ can be transformed into a simple protocol at little extra cost:

Lemma 3.7. *If a graph property $\mathcal{P}$ is decided by a 1-round distributed AM protocol Π with length L, then there exists a simple 1-round distributed AM protocol Π' that decides $\mathcal{P}$ for $\mathcal{G}$ and has length $4L$.*

In Π', we "ask" the prover to give each bridge node $4L$ bits, comprising the four responses it would have given nodes v_A, x_A, x_B, v_B under Π. Nodes v_A, x_A, x_B, v_B verify that the prover gave them the same response, extract their part of the response (i.e., the L bits they expected to receive under Π), and apply their decision function from Π.

From now on we restrict attention to simple protocols, and we focus on the view of the two bridge nodes, x_A and x_B. We claim that for graphs of the form $G(F, F)$, the distribution of messages these nodes receive must capture everything there is to know about F: intuitively, if there exist $F_1 \neq F_2 \in \mathcal{F}$ such that x_A and x_B receive similar responses on $G(F_1, F_1)$ and on $G(F_2, F_2)$, then the adversary can convince all nodes to accept the graph $G(F_1, F_2)$, which is not symmetric. Let us now formalize this intuition.

Define $\mathcal{M}_A(F, r)$ to be the set of responses m to bridge node x_A, such that when the challenge is r and the graph is $G = G(F, F)$, the response m can be extended into a response M for all nodes of V_A and x_A that makes them accept. That is, $m \in \mathcal{M}_A(F, r)$ iff there exists $M \in \{0,1\}^{L \times V_A \cup \{x_A\}}$ with $M_{x_A} = m$, such that (a) for each $v \in V_A$ we have $\mathrm{out}_v(G, r, M) = 1$, and (b) $f_{x_A}(r, M) = 1$. Similarly, let $\mathcal{M}_B(F, r)$ be the set of responses m to x_B, such that on challenge r and the graph is $G(F, F)$, the response m can be extended into a response M that causes all nodes in $V_B \cup \{x_B\}$ to accept.

In a graph $G(F_A, F_B)$, for a given challenge r and response M, the decisions of nodes in $V_A \cup \{x_A\}$ whether to accept or reject does not depend on F_B, and similarly for $V_B \cup \{x_B\}$ and F_A. Therefore we get:

Lemma 3.8. *For any $F, F' \in \mathcal{F}$ and random challenge r, we have: $m \in \mathcal{M}_A(F, r) \cap \mathcal{M}_B(F', r)$ iff there is a prover response M with $M_{x_A} = M_{x_B} = m$, such that $\mathrm{out}_v(G(F, F'), r, M) = 1$ for all $v \in V$.*

Because of this behavior, the prover's probability to convince all nodes to accept is exactly characterized by the probability that the challenge it received has $\mathcal{M}_A(F_A, r) \cap \mathcal{M}_B(F_B, r) \neq \emptyset$:

Lemma 3.9. *For any $G(F_A, F_B) \in \mathcal{G}$,*

$$\max_P \left(\Pr_r(\forall v : \Pi(G, P, v) = 1) \right) = \Pr_r \left(\mathcal{M}_A(F_A, r) \cap \mathcal{M}_B(F_B, r) \right).$$

From the correctness of Π, together with Lemma 3.9 above, we get:

Corollary 3.10. *For any $F_A, F_B \in \mathcal{F}$, if $F_A = F_B$, then*

$$\Pr_r \left(\mathcal{M}_A(F_A, r) \cap \mathcal{M}_B(F_B, r) \right) \geq 2/3.$$

On the other hand, if $F_A \neq F_B$, then

$$\Pr_r \left(\mathcal{M}_A(F_A, r) \cap \mathcal{M}_B(F_B, r) \right) \leq 1/3.$$

Now let $\mu_A(F_A)$ be the distribution of $\mathcal{M}_A(F_A, r)$, where the challenge r is uniformly random. From the previous corollary, for any $F_1, F_2 \in \mathcal{F}$, there is an event — having a non-empty intersection with $\mathcal{M}_B(F_1, r)$ — that has large probability under $\mu_A(F_1)$, but small probability under $\mu_A(F_2)$. Therefore the distributions are *far apart*:

Lemma 3.11. *For any two $F_1, F_2 \in \mathcal{F}$, if $F_1 \neq F_2$, then $\|\mu_A(F_1) - \mu_A(F_2)\|_1 \geq 2/3$.*

Here, $\|\cdot\|_1$ is the L_1 norm, $\|x\|_1 = \sum_{i=1}^N |x_i|$, where N is the dimension of x. It is known that if η_1, η_2 are two distributions on some domain Ω, and there exists an event $Q \subseteq \Omega$ such that $|\eta_1(Q) - \eta_2(Q)| \geq p$, then $\|\eta_1 - \eta_2\|_1 \geq 2p$.

Finally, the following standard argument shows that it is not possible to pack too many distributions that are far apart in L_1-distance into a domain of small size:

Lemma 3.12. *Fix $d \in \mathbb{N}$, and let $\mathcal{U}$ be a set of distributions on the domain $[d]$ such that for all $\mu \neq \eta \in \mathcal{U}$, we have $\|\mu - \eta\| > 1/2$. Then $|\mathcal{U}| < 5^d$.*

Proof. We represent each distribution in $\mathcal{U}$ as a vector in $[0, 1]^d$. For a distribution $\mu \in \mathcal{U} \subseteq [0, 1]^d$ and a radius $r \in [0, 1]$, define an open ball

$$B(\mu, r) = \left\{x \in [0, 1]^d \mid \|\mu - \eta\|_1 < r\right\}.$$

Then we must have $B(\mu, 1/4) \cap B(\eta, 1/4) = \emptyset$ for any $\mu \neq \eta \in \mathcal{U}$: suppose not, and let $x \in B(\mu, 1/4) \cap B(\eta, 1/4)$. Then by the triangle inequality,

$$\|\mu - \eta\|_1 \leq \|\mu - x\|_1 + \|x - \eta\|_1 \leq 1/4 + 1/4 = 1/2.$$

This contradicts our assumption about $\mathcal{U}$.

Now let $B(\vec{0}, 5/4)$ be the open ball

$$B(\vec{0}, 5/4) = \left\{x \in [0, 1]^d \mid \|x\|_1 < 5/4\right\}.$$

(Here, $\vec{0}$ is the d-dimensional zero vector.)

Since each $\mu \in \mathcal{U}$ is a distribution, it has $\|\mu\|_1 = 1$, and therefore $B(\mu, 1/4) \subseteq B(\vec{0}, 5/4)$: if $x \in B(\mu, 1/4)$, then $\|\mu - x\|_1 < 1/4$, and hence $\|x\|_1 \leq \|\mu\| + 1/4 < 5/4$.

So, we have shown that the set $\mathcal{U}$ induces $|\mathcal{U}|$ mutually-disjoint balls of radius 1/4, and all are contained within the ball $B(\vec{0}, 5/4)$. The d-dimensional volume (for natural $d \geq 1$) of the ball $B(x, r)$ with respect to L_1 is given by

$$\mathrm{vol}\,(B(x, r)) = \frac{(4r)^d}{(d+1)!}.$$

Therefore,

$$|\mathcal{U}| \leq \frac{\mathrm{vol}\left(B(\vec{0}, 5/4)\right)}{\mathrm{vol}\left(B(\vec{0}, 1/4)\right)} = \frac{\frac{(4\cdot\frac{5}{4})^d}{(d+1)!}}{\frac{(4\cdot\frac{1}{4})^d}{(d+1)!}} = 5^d.$$

□

Now we can put all the ingredients together to prove the main theorem:

Proof of Theorem 1.4. Fix a simple protocol Π of length L, and let $\mathcal{U} = \{\mathcal{M}_A(F) \mid F \in \mathcal{F}\}$ be the set of distributions that Π induces for the different graphs in $\mathcal{F}$. Each $\mu \in \mathcal{U}$ is a distribution on *sets* of prover's responses to node x_A, so the domain of the distribution in $\mathcal{U}$ has size at most 2^{2^L}.

By Lemma 3.11, for any two $F \neq F' \in \mathcal{F}$ we must have $\|\mathcal{M}_A(F) - \mathcal{M}_A(F')\|_1 \geq 2/3$. In particular, no two distributions in $\mathcal{U}$ are the same, so $|\mathcal{U}| = |\mathcal{F}|$. By Lemma 3.12, we have $|\mathcal{U}| < 5^{2^{2^L}}$. On the other hand, the number of graphs in $\mathcal{F}$ is $2^{\Omega(n^2)}$, and therefore we must have $L = \Omega(\log \log n)$. □

4 INTERACTIVE DISTRIBUTED PROOFS FOR GRAPH NON-ISOMORPHISM

The "standard" GNI protocol. We first attempt to explain why the standard "pepsi vs. cola" protocol for the Graph Non-Isomorphism problem in the centralized setting cannot be adapted to the distributed setting. Recall that in the "pepsi vs. cola" protocol for GNI, the verifier's input is a pair of graphs (G_0, G_1) ("pepsi and cola") on the same vertex set V. The prover claims that G_0 is not isomorphic to G_1. The verifier tests the prover by subjecting it to a "blind taste test": it chooses a random bit $a \in \{0, 1\}$, scrambles the graph G_a by applying a random permutation σ to it, and sends $\sigma(G_a)$ to the prover. If pepsi and cola are not "the same drink with a different name" — that is, if G_0 and G_1 are not isomorphic — then the prover can tell which scrambled graph was sent to it, $\sigma(G_0)$ or $\sigma(G_1)$, because no permutation will turn one graph into the other. It responds to the verifier with a bit b, and the prover accepts iff $a = b$. If G_0 *is* isomorphic to G_1, however, the prover has only probability 1/2 of guessing which graph was presented to it.

Now consider, for example, the following distributed version of GNI (described in subsection 2.3): in addition to getting its neighbor set $N \subseteq V$ in network graph G over which the nodes communicate, each node is given two additional subsets of N as inputs, denoted N_0 and N_1. These sets naturally define two subgraphs G_0 and G_1 of G. We ask the nodes whether G_0 and G_1 are not isomorphic.

Unfortunately, developing a distributed version of the "pepsi vs. cola" GNI protocol seems like a non-starter: in a distributed interactive proof, each network node only sees a small part of the graph, and the nodes have no shared randomness. Even if the nodes *did* have shared randomness, it seems impossible for the nodes to apply a permutation to the entire graph without enlisting the prover's assistance; but asking the prover for help would reveal which graph the nodes chose to scramble, thus "removing the blindfold" and allowing the prover to always succeed.

The Goldwasser-Sipser GNI protocol. Finally, we describe our dAMAM protocol for the GNI problem. This protocol is a distributed version of the Goldwasser-Sipser protocol for GNI [15]. The protocol from [15] is based on the following fact: fix graphs G_0 and G_1, and consider the set S of graphs which are isomorphic to *either* G_0 *or* G_1 (or both). If G_0, G_1 are not isomorphic, then applying any permutation σ to G_0 yields a different graph than we would get if we apply σ to G_1. Intuitively (but incorrectly — see below), since there are $n!$ permutations, then:

(1) if G_0 and G_1 are isomorphic, we should have $|S| = n!$; but
(2) if G_0 and G_1 are not isomorphic, then we should have $|S| = 2n!$, as each permutation of G_0 yields a graph that is not a permutation of G_1.

Thus, to tell whether $(G_0, G_1) \in$ GNI, the verifier *estimates* the size of S: it challenges the prover with a pairwise-independent hash function $h : \{0, 1\}^{n^2} \to \{0, 1\}^L$ (for large enough L), and a random value $r \in \{0, 1\}^L$. The prover is supposed to respond with a graph $G \in S$ which is in the pre-image of r under h, that is, $h(G) = r$. The probability that there *exists* such a graph G is proportional to the size of S, so there is a gap between the acceptance probability when $(G_0, G_1) \in$ GNI compared to when $(G_0, G_1) \in$ GNI.

Specifying a graph on n vertices requires $\Theta(n^2)$ bits. However, the prover is supposed to respond with a graph G that is isomorphic to either G_0 or G_1. Thus, it specifies G by giving a permutation σ and a bit $b \in \{0, 1\}$, with the implicit understanding that $G = \sigma(G_b)$. The verifier then checks that $h(\sigma(G_b)) = r$.

As we mentioned, it is not really true that if $(G_0, G_1) \in$ GNI then $|S| = 2n!$, and if $(G_0, G_1) \notin$ GNI then $|S| = n!$: if one of the graphs is *symmetric*, applying different permutations to it could yield the same result, reducing the size of S. This is fixed cleverly in [15] by changing the definition of S: in addition to the graph G_b, the prover is asked to provide an automorphism of G_b, to "compensate" for symmetries (if there are any). With this new definition of S, it is indeed true that if $(G_0, G_1) \in \mathsf{GNI}$ then $|S| = 2n!$. To simplify the presentation here, we avoid this issue altogether, and restrict attention to asymmetric graphs. To solve the unrestricted GNI problem, we utilize the dAM protocol for Symmetry constructed in Section 3.2.

A distributed GNI protocol. To develop a distributed version of Goldwasser-Sipser, after the prover specifies σ and b, the nodes enlist the prover's help in verifying that $h(\sigma(G_b)) = r$. This requires an additional Arthur-Merlin exchange.

The main difficulty is that in [15], the hash function h is pairwise-independent (PI); this is needed to ensure that the size of the image $h(S)$ is proportional to the true size of S. Unfortunately, PI hash functions require a long random seed [29]. (The family from Theorem 3.2, which we used for Symmetry, is not PI.) We could try to "break up" the seed and have each node send a small part of it to the prover, but we could not find a way to do this that also allowed the nodes to later *verify* that the prover computed the hash correctly.

Instead, we relax the pairwise-independence requirement, and use *ε-almost pairwise-independence* (ε-API) [5, 29]. if the hash is from n bits to m bits, we only require that for each $x_1 \neq x_2 \in \{0,1\}^n$ and $y_1, y_2 \in \{0,1\}^m$, we have

(1) $\Pr_{h \in \mathcal{H}}(h(x_1) = y_1 \wedge h(x_2) = y_2) \leq (1+\epsilon)/2^{2m}$.
(2) $\Pr_{h \in \mathcal{H}}(h(x_1) = y_1) = 1/2^m$.

When we plug in an ε-API hash function h in the GNI protocol, the size of the image $h(S)$ is distorted, but not by too much. We show that we can still recover good success probability for the verifiers.

We give a construction for a distributed ε-API hash, which can be computed "up a spanning tree" in a recursive manner. Each node of the tree has a small part of the input to be hashed, and a short private seed that is not known to the other nodes. The hash for a subtree T_v of node v with children $u_1, \ldots, u_c$ is computed by taking the "partial hashes" $h(T_{u_1}), \ldots, h(T_{u_c})$ sent up by the children of v, and applying a local operation, $h(T_v) = f(h(T_{u_1}), \ldots, h(T_{u_c}), I(v))$, where $I(v)$ is the input of v. Our construction has the useful property that a claimed hash value can be efficiently verified by the nodes with the assistance of the prover.

We use this hash in our GNI protocol: each node v sends the prover its seed, and the prover must respond with the hash value $h(\sigma(G_b)|_{T_v})$. Here, $\sigma(G_b)|_{T_v}$ denotes the "partial adjacency matrix" $\sum_{u \in T_v} [u, \sigma(N(u))]$, which includes only rows corresponding to nodes in v's subtree. Node v verifies that the hash was computed correctly by checking that indeed $h(T_v) = f(h(T_{u_1}), \ldots, h(T_{u_c}), I(v))$. At the root r, we have $\sigma(G_b)|_{T_r} = \sigma(G_b)$, so the hash is complete; the root then verifies that indeed $h(\sigma(G_b)) = r$.

REFERENCES

[1] Scott Aaronson and Avi Wigderson. Algebrization: A new barrier in complexity theory. *ACM Transactions on Computation Theory (TOCT)*, 1(1):2, 2009.
[2] László Babai and Shlomo Moran. Arthur-merlin games: a randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36(2):254–276, 1988.
[3] Alkida Balliu, Gianlorenzo D'Angelo, Pierre Fraigniaud, and Dennis Olivetti. What can be verified locally? In *STACS*, volume 66, pages 8:1–8:13, 2017.
[4] Mor Baruch, Pierre Fraigniaud, and Boaz Patt-Shamir. Randomized proof-labeling schemes. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 315–324. ACM, 2015.
[5] Jürgen Bierbrauer, Thomas Johansson, Gregory Kabatianskii, and Ben Smeets. On families of hash functions via geometric codes and concatenation. In *Annual International Cryptology Conference*, pages 331–342. Springer, 1993.
[6] Keren Censor-Hillel, Ami Paz, and Mor Perry. Approximate proof-labeling schemes. In *International Colloquium on Structural Information and Communication Complexity*, pages 71–89. Springer, 2017.
[7] Laurent Feuilloley and Pierre Fraigniaud. Survey of distributed decision. *Bulletin of the (EATCS)*, 119, 2016.
[8] Laurent Feuilloley and Pierre Fraigniaud. Error-sensitive proof-labeling schemes. In *Symposium on Distributed Computing (DISC)*, pages 16:1–16:15, 2017.
[9] Laurent Feuilloley, Pierre Fraigniaud, and Juho Hirvonen. A hierarchy of local decision. In *43rd International Colloquium on Automata, Languages, and Programming, (ICALP)*, pages 118:1–118:15, 2016.
[10] Pierre Fraigniaud, Mika Göös, Amos Korman, Merav Parter, and David Peleg. Randomized distributed decision. *Distributed Computing*, 27(6):419–434, 2014.
[11] 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)*, pages 157–165. ACM, 2013.
[12] 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. Springer, 2012.
[13] Pierre Fraigniaud, Amos Korman, and David Peleg. Local distributed decision. In *Foundations of Computer Science (FOCS)*, pages 708–717. IEEE, 2011.
[14] Pierre Fraigniaud, Amos Korman, and David Peleg. Towards a complexity theory for local distributed computing. *Journal of the ACM (JACM)*, 60(5):35, 2013.
[15] Shafi Goldwasser and Michael Sipser. Private coins versus public coins in interactive proof systems. *Advances in Computing Research*, 5:73–90, 1989.
[16] Mika Göös, Toniann Pitassi, and Thomas Watson. Zero-information protocols and unambiguity in arthur–merlin communication. *Algorithmica*, 76(3):684–719, 2016.
[17] Mika Göös and Jukka Suomela. Locally checkable proofs in distributed computing. *Theory of Computing*, 12(1):1–33, 2016.
[18] Hartmut Klauck. Rectangle size bounds and threshold covers in communication complexity. In *Conference on Computational Complexity (CCC)*, pages 118–134. IEEE, 2003.
[19] Hartmut Klauck. On arthur merlin games in communication complexity. In *Conference on Computational Complexity (CCC)*, pages 189–199. IEEE, 2011.
[20] Liah Kor, Amos Korman, and David Peleg. Tight bounds for distributed minimum-weight spanning tree verification. *Theory of Computing Systems*, 53(2):318–340, 2013.
[21] Janne H Korhonen and Jukka Suomela. Towards a complexity theory for the congested clique. In *Symposium on Distributed Computing (DISC)*, pages 55:1–55:3, 2017.
[22] Amos Korman and Shay Kutten. Distributed verification of minimum spanning trees. *Distributed Computing*, 20(4):253–266, 2007.
[23] Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. *Distributed Computing*, 22:215–233, 2010.
[24] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, October 1992.
[25] Boaz Patt-Shamir and Mor Perry. Proof-labeling schemes: broadcast, unicast and in between. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 1–17. Springer, 2017.
[26] 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 Journal on Computing*, 41(5):1235–1265, 2012.
[27] Adi Shamir. IP= PSPACE. *Journal of the ACM (JACM)*, 39(4):869–877, 1992.
[28] Jukka Suomela. Survey of local algorithms. *ACM Computing Surveys (CSUR)*, 45(2):24, 2013.
[29] Salil P Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, 7(1–3):1–336, 2012.