Anonymous Networks: Randomization = 2-Hop Coloring

Yuval Emek
Faculty of Industrial Engineering and Management
Technion, Haifa
Israel
yemek@ie.technion.ac.il

Christoph Pfister
Distributed Computing
ETH Zurich
Switzerland
pfistchr@ethz.ch

Jochen Seidel
Distributed Computing
ETH Zurich
Switzerland
seidelj@ethz.ch

Roger Wattenhofer
Distributed Computing
ETH Zurich
Switzerland
wattenhofer@ethz.ch

ABSTRACT

This paper considers the computational power of *anonymous* message passing algorithms (henceforth, anonymous algorithms), i.e., distributed algorithms operating in a network of unidentified nodes. We prove that every problem that can be solved (and verified) by a *randomized* anonymous algorithm can also be solved by a *deterministic* anonymous algorithm provided that the latter is equipped with a 2-*hop coloring* of the input graph. Since the problem of 2-hop coloring a given graph (i.e., ensuring that two nodes with distance at most 2 have different colors) can by itself be solved by a randomized anonymous algorithm, it follows that with the exception of a few mock cases, the execution of every randomized anonymous algorithm can be decoupled into a generic preprocessing randomized stage that computes a 2-hop coloring, followed by a problem-specific deterministic stage. The main ingredient of our proof is a novel simulation method that relies on some surprising connections between 2-hop colorings and an extensively used graph lifting technique.

Categories and Subject Descriptors

F.1.2 [**Theory of Computation**]: Modes of Computation—*Relations among modes*

General Terms

Theory

Keywords

Anonymous Networks; Derandomization; 2-Hop Coloring

PODC'14, July 15–18, 2014, Paris, France.

http://dx.doi.org/10.1145/2611462.2611478 .

1. INTRODUCTION

Computability in networks (a.k.a. distributed computability), where the processors are represented as nodes in a graph and the task is to produce an output at every node, is Turing machine-equivalent if the nodes are equipped with unique IDs. This fact remains intact even when the attention is restricted to *deterministic* distributed algorithms. In contrast, the distributed problems that can be solved deterministically in *anonymous* networks (where the nodes are not uniquely identified) are of a rather limited nature [38]. The question of distributed computability becomes interesting however once the nodes in an anonymous network gain access to random bits. For example, the extensively studied *maximal independent set (MIS)* problem [34, 3] is solvable in an anonymous network only if random bits are available.

Our goal in this paper is to investigate the role that (Las-Vegas type) randomness plays in the computational power of anonymous message passing algorithms (referred to hereafter as *anonymous algorithms*), regardless of round and message complexity considerations. However, before we can do so, care must be taken to rule out distributed problems in which unique IDs are (perhaps implicitly) encoded in the input instances as those mock cases obviously do not faithfully represent the properties of distributed computability in anonymous networks. To that end, we restrict our focus to the class GRAN (standing for Genuinely solvable by Randomized algorithms in Anonymous Networks) of distributed problems Π that satisfy (1) there exists a randomized anonymous algorithm that solves Π; and (2) there exists a randomized anonymous algorithm that decides whether a given graph is a legal input instance of Π; we refer to Section 1.1 for a formal definition. Notice that with the exception of some artificial cases generated for the purpose of investigating the leader election problem, essentially all interesting distributed problems studied in the existing literature in the context of anonymous networks belong, in fact, to GRAN.

What exactly characterizes the computational power of a randomized anonymous algorithm as opposed to a deterministic one? Surprisingly, randomization is only required to establish a 2-*hop coloring* of the network: Once a 2-hop coloring is known, every problem in GRAN can be solved by a deterministic anonymous algorithm.

1.1 Model

Labeled Graphs.

We denote the node- and edge-set of a graph G by V and E, respectively. For a node $v \in V$, we denote the set containing all neighbors of v by $\Gamma(v)$. To make it explicit, we only consider finite connected simple[1] graphs G. A *labeling function* for V is a function $\ell : V \to L$ that assigns a *label* to every node in V. For the sake of simplicity, unless stated otherwise, we assume hereafter that all labels are finite bitstrings. Tuples $G = (V, E, \ell)$, where (V, E) is a graph and ℓ is a labeling function for V, are called *labeled graphs*. We often label vertices by more than one labeling function $\ell_1, \ldots, \ell_k$; in that case, we treat $G = (V, E, \ell_1, \ldots, \ell_k)$ as being labeled by a single labeling function ℓ that assigns $\ell(v) = \langle \ell_1(v), \ldots, \ell_k(v) \rangle$ to each node $v \in V$.

Distributed Problems.

A distributed problem Π is specified by a set of *input instances* and for every input instance I of Π, a set $\Pi(I)$ of *valid outputs* for I. The input instances of Π are labeled graphs $I = (V, E, i)$ and the labeling function i, called the *input labeling* of I, assigns an *input label* to every node in V. For the input instance $I = (V, E, i)$, the set of valid outputs $\Pi(I)$ for I consists of labeling functions o for V called *valid output labelings* for I. At the risk of abusing the notation, we use Π to denote the set of input instances as well as the problem itself. For the sake of simplicity we assume that in every input instance $I = (V, E, i)$, the input label $i(v)$ of every node v includes v's degree. A typical example for a distributed problem is *graph coloring*, where the input is an arbitrary graph and the output must obey the rule $o(u) \neq o(v)$ if (u, v) is an edge in the input instance.

2-*Hop Colorings.*

For a graph $G = (V, E)$, the labeling ℓ is said to be a *k-hop coloring* if $\ell(u) \neq \ell(v)$ for every $u, v \in V$, $u \neq v$, that are at most k hops away, i.e., G admits a path between u and v that consists of at most k edges. A labeling function that plays a central role in the present paper is the 2-hop coloring, i.e., a coloring that assigns a different label to each node in $\{u\} \cup \Gamma(u)$ for every $u \in V$. We say that a labeled graph $G = (V, E, \ell)$ is 2-*hop colored* if ℓ is a 2-hop coloring. Note that in the context of the present paper, we do not pay attention to the number of distinct colors used by the nodes under ℓ.

Consider some distributed problem Π. The 2-*hop colored variant* Π^c of Π is the problem defined as follows: the input instance set Π^c is

$$\Pi^c = \{(V, E, i, c) : (V, E, i) \in \Pi, \text{ and } c \text{ is a 2-hop coloring of } (V, E)\}$$

and given an input instance $I = (V, E, i) \in \Pi$, the valid output labeling set for every corresponding input instance $I^c = (V, E, i, c) \in \Pi^c$ is $\Pi^c(I^c) = \Pi(I)$.

Randomized Anonymous Algorithms.

Our model for randomized anonymous algorithms corresponds to the anonymous variant of the message passing model as defined in [40] (c.f. the port numbering model of [5]) with irrevocable outputs. In a labeled graph $I = (V, E, i)$, all nodes v execute the same message passing algorithm $\mathcal{A}$ with input $i(v)$. The input to a node v is fully specified by $i(v)$ — in particular, nodes are not equipped with a *(unique) identifier* nor do they possess an apriori knowledge of any global network parameter (unless specified as part of $i(\cdot)$). The execution of $\mathcal{A}$ on I is performed in synchronous rounds. In every round, each node v sends/receives messages of finite size to/from each of its neighbors, where v distinguishes between the ports corresponding to its incident edges. We consider randomized algorithms, where in every round, node v has access to one random bit. (Note that this is equivalent to accessing finitely many random bits per round as multiple rounds can be grouped together.)

Algorithm $\mathcal{A}$ is said to *solve* the distributed problem Π if the following two requirements hold for every $I \in \Pi$: (1) if $\mathcal{A}$ is executed on I, then $\mathcal{A}$ produces an irrevocable local output $\mathcal{A}(v)$ for every node v within finite time with probability 1; and (2) each output labeling o obtained with positive probability by setting $o(v) = \mathcal{A}(v)$ for every $v \in V$ satisfies $o \in \Pi(I)$ (i.e., we only consider Las-Vegas algorithms). Algorithm $\mathcal{A}$ is *deterministic* if it does not access any random bits. Note that the round and message complexities of $\mathcal{A}$ are not taken into consideration (as long as they are finite).

Genuine Solvability.

Let Y be a set of labeled graphs called *yes-instances*. The *distributed decision problem* Δ_Y obtained from Y (see, e.g., [21]) is the problem whose input instances I are all labeled graphs, and whose valid output labelings $o \in \Delta_Y(I)$ are such that all nodes output "YES" if $I \in Y$ and at least one node outputs "NO" if $I \notin Y$. We say that problem Π is *genuinely* solvable by randomized algorithms in anonymous networks if (1) there exists a randomized anonymous algorithm that solves Π; and (2) there exists a randomized anonymous algorithm that solves the distributed decision problem Δ_Π, namely, the problem of deciding whether a given labeled graph is an input instance of Π. Denote the class of such problems Π by GRAN (standing for Genuinely solvable by Randomized algorithms in Anonymous Networks).

Local Views.

Given a node v in the labeled graph $G = (V, E, \ell)$, we denote by $L_d(v, G)$ a rooted tree called the *depth-d local view* of v in G. (When G is clear from the context we may write $L_d(v)$ instead.) To avoid confusion, we distinguish between nodes and labels in G and *vertices* and *marks* in $L_d(v)$. The local view of node v is defined inductively as follows: $L_1(v)$ consists of a single vertex x marked with $\ell(v)$; $L_{d+1}(v)$ is the tree obtained by connecting the root of $L_d(u)$ as a child of $L_1(v)$'s root for every $u \in \Gamma(v)$. Refer to Figure 1 for an illustration. Notice that the the local view $L_d(v)$ in G essentially captures all information that a deterministic algorithm $\mathcal{A}$ executed by node v in G could possibly gather in d rounds of execution. The *depth-infinity local view* of a node v is the infinite tree $L_\infty(v)$ obtained from the inductive construction of $L_d(v)$ by taking d to infinity.

1.2 Our Contribution

Classic distributed symmetry breaking problems such as maximal independent set and graph (1-hop) coloring are known to be in GRAN. Common to these two problems

[1] A graph is simple if it is undirected and does not contain any loops or parallel edges.

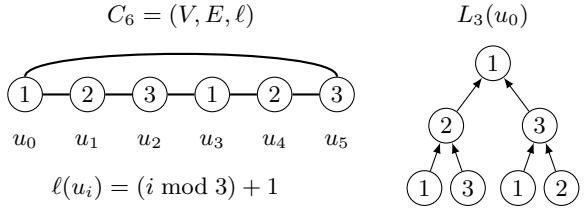

Figure 1: Depth-3 local view of node u_0 in the labeled graph C_6.

is the local nature of their symmetry breaking challenges.[2] While the 2-hop variant of graph coloring is still solvable by randomized anonymous algorithms and thus, belongs to GRAN, it is not difficult to show that this no longer holds for its k-hop variant for any $k > 2$. (In fact, the same can be said for maximal independent set under a natural extension to k-hop variants that are not discussed in the present paper, cf. [37].) Is this a coincidence? Does GRAN contain problems that require (systematic) symmetry breaking between nodes which are more than two hops apart? Our main result provides a negative answer to these questions.

THEOREM 1. *If $\Pi \in$ GRAN, then Π^c is solvable by a deterministic anonymous algorithm.*

1.3 Related Work

The seminal work by Angluin [5] established the connection between computation in networks and *factors/products* of graphs (see Section 2 for a definition) and marks the beginning of history for distributed computability theory. Her work employs a lifting technique from a graph to its products to establish impossibility of leader election (and equivalent problems, e.g., assigning IDs), even under the assumption of Las-Vegas algorithms. As stated in [23] graph products also characterize recognizable cases for graph rewriting systems, a localized model for distributed computation. Fibrations, i.e, a related generalization for directed graphs (see [13] for an extensive overview), were found to characterize problems solvable by self stabilization (a possibly incorrect output stabilizes to a correct one) in [14]. The lifting technique also plays a key role in our proof of Theorem 1.

Product graphs are also studied in their own right (see, e.g., [8, 31]), also products obtained from a random process [4]. For a graph G the *universal cover* $U(G)$ (cf. [5]) is a (possibly infinite) product of G closely related to the depth-infinity local view. (The un-rooted tree $U(G)$ can be obtained from $L_\infty(v)$ of any node v in G by (1) for every vertex x in $L_\infty(v)$ pruning x's child corresponding to x's parent; and (2) making every edge in the resulting tree undirected.) In Section 3 we apply the result of Norris [39] that isomorphism in $U(G)$ up to depth $|V| - 1$ implies isomorphism to all depths to obtain a finite representation of a specific factor of G.

Another line of research investigates (mock-anonymous) problems where the input instances permit leader election. For example, electing a leader in a ring network is possible if the size n of the ring is known [26, 27]. Later it was found that a $(2 - \varepsilon)$-approximation of n is enough [1], even in general networks [42]. The impact of prior knowledge (e.g., the network size) on the solvability of various problems was studied in [43, 11]. We restrict ourselves to problems in GRAN, which rules out cases that permit leader election.

Electing a leader with a Monte-Carlo algorithm (a randomized algorithm that is allowed to fail) was studied in rings [26, 27] and in general graphs [2, 41]. Recently, the problem was found to be solvable with high probability (i.e., with probability $1 - n^{-c}$ for any $c \geq 1$, w.h.p. for short) in [36]. Since electing a leader and assigning IDs are equivalent, any distributed problem solvable with IDs is w.h.p. solvable in an anonymous network. On the other hand it is known that some symmetry-breaking problems, e.g., MIS [34, 3] or coloring [33], are in GRAN. It is thus a natural question to ask what exactly distinguishes Las-Vegas algorithms from deterministic ones. We find that a 2-hop coloring suffices to completely characterize the capabilities of a Las-Vegas algorithm solving a GRAN problem.

Anonymous networks and electing a leader therein also plays a major role in self-stabilization research, e.g. [17]. Self-stabilizing leader election is possible with population protocols (nodes are controlled by asynchronous finite state machines, cf. [7]) if the network is a ring [10]. In the presence of an oracle the problem becomes solvable also in general networks, however, the required oracle is impossible to implement as a population protocol [9]. We hope that our contribution may also play a role towards a better understanding of randomization in self-stabilization (cf. [32, 18]).

Naor and Stockmeyer [38] introduced the notion of locally checkable labelings (LCL), a labeling that can be checked by a deterministic constant-time algorithm in a network with IDs. With IDs a randomized constant-time algorithm cannot solve more LCLs than a deterministic one. This is in contrast to the anonymous model, where the run-time is unbounded but finite and a deterministic algorithm requires a 2-hop coloring to replace randomization. The impact of having/not having identifiers on verifying a proof (solution) to a decision problem was studied in [28], and the authors observe that the existence of a uniquely determined leader cannot be verified without identifiers. A hierarchy of decision problems in terms of bit complexity required for a proof is established in [25]. In [21, 20] Monte-Carlo algorithms for decision problems are studied in networks with unidentified but distinguishable nodes and a strict hierarchy depending on the success probability is found. We utilize the notion of decision problems to characterize the problem class GRAN.

The notion of a 2-hop coloring (also referred to as distance-2 coloring) has been used to assign frequencies in radio networks [29], to solve optimization problems in parallel on shared memory computers [22], and to emulate Turing machines in population protocols [6]. The related k-local election problem, where a local leader needs to be unique only up to distance k, was studied in an asynchronous model in [37]. A solution to the 2-hop coloring problem can already be found in the weak model of [19], where nodes are controlled by finite state machines. Minimizing the number of colors in a k-hop coloring is, however, $\mathcal{NP}$-complete for any $k \geq 1$ due to a result in [35]. We would like to note that port numbers are not necessary under the assumption of randomized algorithms. Since each node v knows its degree a 2-hop coloring can be found even without port numbers and by including the sender's color in every message missing port

[2] Despite the inherent locality of the notions of maximal independent set and coloring, the results of [33, 30] show that the corresponding distributed computational tasks cannot be solved in constant time.

numbers can be emulated. We show that a 2-hop coloring uniquely determines a graph's prime factor.

2. THE CASE FOR INFINITY

Before presenting the proof of Theorem 1, we state and prove a slightly easier variant of it that captures some of its main ideas. To that end, for the moment, assume the following rather strong *infinity model* for anonymous computation. Fix some problem Π, let Π^c be its 2-hop colored variant, and let $I^c = (V, E, i, c) \in \Pi^c$ be some input instance. An algorithm under the infinity model is fully specified by a function $\mathcal{A}_\infty$ from the set of depth-infinity local views to the possible output labels. The algorithm sets the output of node $v \in V$ to be $o(v) = \mathcal{A}_\infty(L_\infty(v))$, namely, it applies the function $\mathcal{A}_\infty$ to $L_\infty(v)$ and uses the returned image as the output of v. We shall use $\mathcal{A}_\infty$ to denote the algorithm under the infinity model as well as the function that lies at its heart. Disregarding computability issues of $\mathcal{A}_\infty$ for the moment, we say that $\mathcal{A}_\infty$ *solves* Π^c if the labeling o satisfies $o \in \Pi^c(I^c)$. Note that the infinity model involves neither communication nor randomization. In other words, node v's output is completely determined by $L_\infty(v)$ in which the vertices are only marked with input labels and a 2-hop coloring. The remainder of this section is devoted to proving the following theorem.

THEOREM 2. *If $\Pi \in$ GRAN, then Π^c is solvable in the infinity model.*

A key ingredient of our proof for Theorem 2 is the notion of an *infinite view graph* G_∞ of a 2-hop colored graph G.

Definition 1. Let $G = (V, E, \ell)$ be a 2-hop colored graph. We define the *infinite view graph* $G_\infty = (V_\infty, E_\infty, \ell_\infty)$ of G by identifying $L_\infty(v)$ with $\tilde{v}$ and setting

$$V_\infty := \{\tilde{v} : v \in V\} \qquad \text{(the different local views in } G\text{)},$$
$$E_\infty := \{(\tilde{u}, \tilde{v}) : (u, v) \in E\},$$
$$\ell_\infty(\tilde{v}) := \ell(v)$$

Note that $|V_\infty| \leq |V|$, where the inequality is strict when different nodes in G have the same depth-infinity local view For example in the graph C_6 from Figure 1 the local views of nodes with the same color are equal.

For the remainder of this section, fix some problem $\Pi \in$ GRAN, let $\mathcal{A}_\mathrm{R}$ be a randomized anonymous algorithm solving Π, and let $I^c = (V, E, i, c)$ be some input instance of Π^c. We would like to construct an algorithm $\mathcal{A}_\infty$ that solves Π^c under the infinity model. The idea behind $\mathcal{A}_\infty$ is to perform the following three steps for each node $v \in V$:

i. construct the infinite view graph $I^c_\infty = (V_\infty, E_\infty, i_\infty, c_\infty)$ from $L_\infty(v)$;
ii. simulate a specific terminating execution of $\mathcal{A}_\mathrm{R}$ on $J = (V_\infty, E_\infty, i_\infty)$; and
iii. use the output of node $\tilde{v}$ in that simulation as output for node v.

We now turn to explaining these three steps in detail.

2.1 Constructing I^c_∞

Consider $L_\infty(v, I^c)$ for some node $v \in V$. For every node $u \in V$, the local view $L_\infty(u)$ appears as a sub-tree in $L_\infty(v)$. Conversely, every depth-infinity sub-tree of $L_\infty(v)$ is the depth-infinity local view of some node (or nodes) in V. Therefore, the set of all depth-infinity sub-trees of $L_\infty(v)$ is exactly the node set of I^c's infinite view graph I^c_∞. Moreover, (u, u') is an edge in I^c if and only if $L_\infty(u')$ appears as a sub-tree of $L_\infty(u)$ rooted at a child of $L_\infty(u)$'s root. In other words, I^c_∞ can be uniquely constructed from $L_\infty(v)$.

Algorithm $\mathcal{A}_\infty$ and its analysis relies on a canonical representation of the depth-infinity trees $L_\infty(u)$, namely, fixing the order of the vertices in each depth-level. For that purpose, it suffices to fix a total order among the children of each vertex. Such a total order follows immediately by noticing that since I^c is 2-hop colored, every two siblings must have distinct marks. Using these canonical representations, two depth-infinity local views can now be compared level by level, thus implying a total order on V_∞; let $\tilde{u}_1, \ldots, \tilde{u}_k$ be the nodes in V_∞ indexed according to this total order.

2.2 Simulating $\mathcal{A}_\mathrm{R}$

The second step in $\mathcal{A}_\infty$ is to simulate an execution of algorithm $\mathcal{A}_\mathrm{R}$, the randomized anonymous algorithm solving Π. More precisely, multiple executions of $\mathcal{A}_\mathrm{R}$ will be simulated on the input $J = (V_\infty, E_\infty, i_\infty)$. A t-round simulation σ of $\mathcal{A}_\mathrm{R}$ on J (corresponding to executing $\mathcal{A}_\mathrm{R}$ on J for t rounds) is fully determined by an assignment $b : V_\infty \to \{0,1\}^t$ of t random bits to every node in V_∞. We refer to this simulation σ as the simulation *induced by* b. The simulation σ is said to be *successful* if every node $\tilde{v} \in V_\infty$ produces an output under σ.

It will be essential for $\mathcal{A}_\infty$'s correctness that all nodes in I^c choose the same simulation of $\mathcal{A}_\mathrm{R}$ on J — i.e., the same assignment b — to determine their output. This will be accomplished by using the total order on V_∞ to fix a total order among the possible assignments $b : V_\infty \to \{0,1\}^t$. To that end, given two assignments $b_1, b_2 : V_\infty \to \{0,1\}^t$, $b_1 \neq b_2$, the relation $b_1 < b_2$ holds if and only if $(b_1(\tilde{u}_1), \ldots, b_1(\tilde{u}_k)) < (b_2(\tilde{u}_1), \ldots, b_2(\tilde{u}_k))$, where the latter comparison is done lexicographically. For convenience, we extend the total order on the assignments b so that it also covers assignments $b_1 : V_\infty \to \{0,1\}^{t_1}$ and $b_2 : V_\infty \to \{0,1\}^{t_2}$, $t_1 \neq t_2$, by defining that $b_1 < b_2$ holds if and only if $t_1 < t_2$.

Assuming that there exists a successful simulation of $\mathcal{A}_\mathrm{R}$ on J, algorithm $\mathcal{A}_\infty$ selects the successful simulation induced by the smallest assignment $b : V_\infty \to \{0,1\}^t$. We denote this simulation by σ_∞ and summarize in the following lemma.

LEMMA 1. *If algorithm $\mathcal{A}_R$ returns an output when executed on J, then in $\mathcal{A}_\infty$, all nodes select the same successful simulation σ_∞.*

2.3 The Output of $\mathcal{A}_\infty$

The output value $o(v) = \mathcal{A}_\infty(L_\infty(v))$ of node $v \in V$ is set to the output produced by node $\tilde{v}$ in simulation σ_∞ of $\mathcal{A}_\mathrm{R}$ on J. For that to be well defined, there must exist an execution of $\mathcal{A}_\mathrm{R}$ on J in which every node $\tilde{v}$ produces an output (leading to a successful simulation). We establish the existence of such an execution by using the well-known lifting lemma [5, 12] to assert that $J \in \Pi$ and thus, guarantee that a terminating execution of $\mathcal{A}_\mathrm{R}$ on J exists. This requires developing a better understanding of the infinite view graph's fundamental properties based on the notion of factor graphs.

2.3.1 Factor Graphs and 2-Hop Colorings

The central concept of our analysis is that of *factor* and *product graphs*.[3] For two labeled graphs $G = (V, E, \ell)$ and

[3] Our notion of product graphs should not be confused with

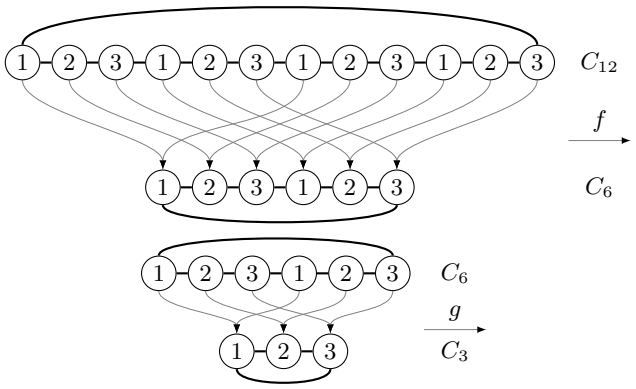

Figure 2: The labeled graph C_6 is a factor of C_{12} induced by the factorizing map f (and C_{12} is a product of C_6). Similarly, the labeled graph C_3 is a factor of C_6 induced by the factorizing map g.

$G' = (V', E', \ell')$, we say that G' is a *factor* of G and that G is a *product* of G' if there exists a function $f : V \to V'$, referred to as a *factorizing map*, that satisfies:

1. the mapping f is surjective (onto);
2. f respects the labeling functions, that is, $\ell(v) = \ell'(f(v))$ for every node $v \in V$; and
3. f is a *local isomorphism*, that is, for every node $v \in V$, the restriction $f|_{\Gamma(v)}$ is a bijection onto $\Gamma(f(v))$.

We shall use the notations $G' \preceq_f G$ (and $G \succeq_f G'$) to denote that G' is a factor of G (and G is a product of G'). The role of the factorizing map f is sometimes emphasized by saying that the factor/product is *induced* by f. Refer to Figure 2 for an illustration.

It is known that $|V| = m \cdot |V'|$ for some positive integer m (see, e.g., [24]). If $m = 1$, then the factorizing map is bijective and both $G' \preceq_f G$ and $G \preceq_{f^{-1}} G'$ hold. In that case, we refer to the two labeled graphs G and G' as being *isomorphic* (since f is a graph isomorphism that respects the node labels) and write $G \cong_f G'$ or $G \cong G'$ if the specific bijection is not relevant. Moreover, it is well-established that if $G \succeq_f G'$, then the graphs G and G' are indistinguishable from the perspective of a node v in G or G' (see, e.g., [39]) as cast in the following fact.

Fact 1. *Let G, G' be two labeled graphs. If $G \succeq_f G'$, then $L_\infty(v) = L_\infty(f(v))$ for every node v in G.*

It follows from Fact 1 that the factor of a 2-hop colored graph is also 2-hop colored. Let $G = (V, E, \ell)$ be a 2-hop colored graph and let $G_\infty = (V_\infty, E_\infty, \ell_\infty)$ be its infinite view graph. Lemmas 2 to 4 establish some important properties of G and G_∞. These three lemmas can be derived from the results on graph fibrations presented in [13] using an intricate construction. (we briefly sketch this connection in Section 4); for completeness, we also present stand-alone proofs. Our first Lemma 2 establishes that G_∞ is a factor of G induced by the factorizing map $f_\infty : V \to V_\infty$ that maps v to $\tilde{v}$; we subsequently refer to f_∞ as the *infinite view (factorizing) map* of G.

Lemma 2. *Let $G = (V, E, \ell)$ be a 2-hop colored graph, $G_\infty = (V_\infty, E_\infty, \ell_\infty)$ its infinite view graph, and $f_\infty : V \to V_\infty$ the infinite view map. Then, G_∞ is a factor of G induced by f_∞, i.e., $G_\infty \preceq_{f_\infty} G$.*

Proof. We show that f_∞ is a factorizing map inducing the factor G_∞ by verifying the properties required in the definition of factor graphs. (1) The function $f_\infty : V \to V_\infty$ is surjective by the definition of V_∞. (2) For every $v \in V$, the labeling functions satisfy $\ell(v) = \ell_\infty(f_\infty(v))$ by the definition of ℓ_∞. (3) Bijectivity of $f_\infty|_{\Gamma(v)}$ for every $v \in V$ is established by showing that $f_\infty|_{\Gamma(v)}$ is injective and surjective separately.

To see that $f_\infty|_{\Gamma(v)}$ is injective, observe that two nodes $u_1, u_2 \in \Gamma(v)$, $u_1 \neq u_2$, have different labels $\ell(u_1) \neq \ell(u_2)$ since ℓ is a 2-hop coloring. Property (2) thus ensures that $f_\infty(u_1) \neq f_\infty(u_2)$, i.e., $f_\infty|_{\Gamma(v)}$ is injective. Denote by $\tilde{v}$ the node $f_\infty(v)$ and let $\tilde{u}$ be some neighbor of $\tilde{v}$ in G_∞. It follows from the definitions of E_∞ and local views that the tree $\tilde{u}$ is a sub-tree rooted at a child of $\tilde{v}$'s root vertex. Thus, G admits some node $u \in \Gamma(v)$ with $L_\infty(u) = \tilde{u}$ and the function f_∞ satisfies $f_\infty|_{\Gamma(v)}(u) = \tilde{u}$. Hence, $f_\infty|_{\Gamma(v)}$ is also surjective. □

A labeled graph G is called *prime* (cf. [13]) if all factors of G are isomorphic to it. For example, the labeled 3-cycle C_3 in Figure 2 is prime, whereas C_{12} and C_6 are not. Both C_{12} and C_6 have C_3 as a prime factor. Lemma 3 states that the only prime factor of a 2-hop colored graph G is its infinite view graph (indeed, C_3 is isomorphic to the infinite view graph of C_{12} and C_6).

Lemma 3. *If G is a 2-hop colored graph, then the infinite view graph G_∞ is the unique prime factor of G (up to isomorphism).*

Proof. Let G and G' be 2-hop colored graphs with $G' \preceq_f G$. To establish the statement we show that G' is either isomorphic to G_∞ or not prime. The key to our proof is to show that G and G' have the same infinite view graph; this establishes the assertion due to Lemma 2.

To that end, let $G_\infty = (V_\infty, E_\infty, \ell_\infty)$ and $G'_\infty = (V'_\infty, E'_\infty, \ell'_\infty)$ be the infinite view graphs of G and G', respectively, i.e., $G_\infty \preceq_{f_\infty} G$ and $G'_\infty \preceq_{f'_\infty} G'$. Fact 1 implies that $V'_\infty = V_\infty$. The construction of E_∞ guarantees that an edge $(\tilde{u}, \tilde{v})$ is in E_∞ if and only if $\tilde{v}$ is a sub-tree rooted at one of the children of $\tilde{u}$'s root. Since $V'_\infty = V_\infty$, the same edge is also in E'_∞, and vice versa, every edge in E'_∞ is also in E_∞. Therefore, it also holds that $E'_\infty = E_\infty$. Finally, observe that $\ell_\infty(\tilde{w})$ and $\ell'_\infty(\tilde{w}')$ for any nodes $\tilde{w} \in V_\infty$ and $\tilde{w}' \in V'_\infty$, respectively, are completely determined by the marks attributed to the corresponding root vertices of $\tilde{w}$ and $\tilde{w}'$. We conclude that $(V_\infty, E_\infty, \ell_\infty) = (V'_\infty, E'_\infty, \ell'_\infty)$, i.e., $G_\infty = G'_\infty$. □

Note that the previous Lemma 3 does not hold for arbitrary graphs G, since, e.g., the uncolored 12-cycle has two distinct prime factors, namely, the 3-cycle and the 4-cycle. In prime 2-hop colored graphs however, based on the following key lemma, we can use $L_\infty(v)$ of node v in a prime 2-hop colored graph G as the *alias* of v in G.

Lemma 4. *Let $G = (V, E, \ell)$ be a prime 2-hop colored graph and consider some $u, v \in V$. Then, $u = v$ if and only if $L_\infty(u) = L_\infty(v)$.*

binary operations on two graphs referred to as *graph products*. The unlabeled counterpart of the concept of product graphs is often called graph lifts, or covering graphs in the existing literature. We extend the definition presented in [24] to incorporate node labels. Note that changes are required when considering non-simple or undirected graphs.

PROOF. Let $G = (V, E, \ell)$ be a prime 2-hop colored graph and let $u, v \in V$ be two nodes in G. Since the "only-if" direction is true by the assumption $u = v$, we only need to show that $L_\infty(u) = L_\infty(v)$ implies $u = v$. To that end, consider the infinite view graph $G_\infty = (V_\infty, E_\infty, \ell_\infty)$ of G and assume for the sake of contradiction that $L_\infty(u) = L_\infty(v)$ in G but $u \neq v$. By the definition of G_∞, this implies that $|V_\infty| < |V|$ and thus, Lemma 2 guarantees that G admits a non-trivial factor, in contradiction to the assumption that G is prime. □

2.3.2 Establishing the Output's Validity

Getting back to $\mathcal{A}_\infty$, we have shown that the graph I^c_∞ constructed by $\mathcal{A}_\infty$ (independently, at every node v) satisfies $I^c_\infty \preceq_{f_\infty} I^c$. (Recall that $I^c = (V, E, i, c)$ is an input instance of Π's 2-hop colored variant Π^c and that $I = (V, E, i)$ is the corresponding input instance of Π.) Algorithm $\mathcal{A}_\infty$ simulates algorithm $\mathcal{A}_R$ on the input $J = (V_\infty, E_\infty, i_\infty)$. Since $I^c_\infty \preceq_{f_\infty} I^c$, the input instance I satisfies $J \preceq_{f_\infty} I$ with the same factorizing map f_∞.

We argue that J is an input instance of Π. Indeed, as Π is genuinely solvable, there exists a randomized anonymous algorithm $\mathcal{B}$ that decides whether a given labeled graph is an input instance of Π. By the lifting lemma, $\mathcal{B}$ cannot distinguish J from I (cf. [5, 12]), hence the fact that $I \in \Pi$ implies that $J \in \Pi$, as required.

It follows that $\mathcal{A}_R$ returns a correct output when executed on J, thus Lemma 1 ensures that the same successful simulation σ_∞ is selected by every node. Employing the lifting lemma once more, we conclude that the simulation obtained by lifting σ_∞ from nodes in V_∞ to nodes in V corresponds to a possible execution η of $\mathcal{A}_R$ on I (cf. [5, 12]). The output labeling that $\mathcal{A}_\infty$ produces for the input instance I^c is exactly the output labeling produced by η in I and since the latter is valid, the former must also be valid by the definition of problem Π^c, thus establishing Theorem 2.

Since the description of $\mathcal{A}_\infty$ involves trees of infinite depth, one may wonder whether it can be replaced by a "real" algorithm. This issue is addressed in the next section, where we also establish Theorem 1.

3. DEALING WITH (IN)FINITY

Recall that under the infinity model introduced in Section 2, each node v in the graph essentially receives $L_\infty(v)$. This luxury, of course, cannot be realized in a standard anonymous algorithm, where node v can only obtain $L_d(v)$ for finite values of d. Nevertheless, in this section we show how the algorithm presented in Section 2 can be adapted to finite depth local views. A key ingredient in this adaptation is the following theorem established by Norris [39].[4]

THEOREM 3 ([39]). *Let G be a labeled graph with n nodes. The local view $L_n(v)$ fully determines $L_\infty(v)$ for every node $v \in V$.*

Employing Lemma 4, we obtain the following corollary that facilitates the usage of depth n local views as aliases for the nodes in an n-node prime 2-hop colored graph (instead of the depth-infinity local views that were used in Section 2).

[4] The result in [39] is described in terms of the depth $n-1$ sub-trees of a graph's universal cover. To prove the corresponding statement for depth-n local views the same refinement argument can be made.

COROLLARY 1. *Let $G = (V, E, \ell)$ be an n-node prime 2-hop colored graph and consider some $u, v \in V$. Then, $u = v$ if and only if $L_n(u) = L_n(v)$.*

Consider a 2-hop colored graph $G = (V, E, \ell)$ and denote by $n = |V_\infty|$ the number of different depth-infinity local views in G. For a node $v \in V$, denote by $\overset{*}{v} = L_n(v)$ the depth-n local view in G. The graph $G_* = (V_*, E_*, \ell_*)$, where $V_* = \{\overset{*}{v} : v \in V\}$, $E_* = \{(\overset{*}{u}, \overset{*}{v}) : (u, v) \in E\}$, and $\ell_*(\overset{*}{v}) = \ell(v)$ is called the *finite view graph* of G. The following corollary is established due to Theorem 3, where f_n is the *depth-n truncating function* that truncates every depth-k local view, $k \geq n$, to depth n, i.e., $f_n(\overset{\infty}{u}) = \overset{*}{v}$.

COROLLARY 2. *For a 2-hop colored graph G, it holds that $G_* \cong_{f_n} G_\infty$.*

The graph G_* can thus serve as a canonical representative for its equivalence class under the equivalence relation $\cong$. This is crucial because in contrast to G_∞, the graph G_* has a finite bitstring representation. Furthermore, each node v in a 2-hop colored graph G can identify its corresponding node in G_*, within $n = |V_*|$ rounds of the execution.

3.1 Algorithm $\mathcal{A}_*$

Fix some problem $\Pi \in$ GRAN and randomized anonymous algorithm $\mathcal{A}_R$ that solves Π. Let Π^c be the 2-hop colored variant of Π and let $I^c = (V, E, i, c) \in \Pi^c$ be an arbitrary input instance of Π^c. To establish Theorem 1, we present a deterministic algorithm $\mathcal{A}_*$ that solves Π^c. Algorithm $\mathcal{A}_*$ resembles algorithm $\mathcal{A}_\infty$ presented in Section 2, however, the graph I^c_∞ is replaced by its finite representation I^c_* utilizing Corollary 2.

Algorithm $\mathcal{A}_*$, described from the perspective of an arbitrary node $v \in V$, proceeds in phases indexed by the positive integers. Throughout the execution of $\mathcal{A}_*$, node v keeps track of an initially empty bitstring $b(v)$, where the value of $b(v)$ for phase $p+1$ is determined during phase p. It will be convenient to denote by b^p the labeling function derived from the values $b(v)$ in phase p by setting $b^p(v) = b(v)$ for phase p. Correspondingly, we denote by $I^p = (V, E, i, c, b^p)$ the graph obtained by augmenting I^c with the labeling b^p. In each phase p, every node v invokes the three sub-procedures `Update-Graph`, `Update-Output`, and `Update-Bits` in this order. We now describe each sub-procedure individually. For convenience, we also include a pseudo-code style description of $\mathcal{A}_*$ in Figure 3.

`Update-Graph`.

We say that a labeled graph $\hat{G} = (\hat{V}, \hat{E}, \hat{i}, \hat{c}, \hat{b})$ is a *candidate* for phase p if it satisfies the following three conditions:

C1. $|\hat{V}| \leq p$;

C2. there exists a node $\hat{v} \in \hat{V}$ such that $L_p(\hat{v}, \hat{G}) = L_p(v, I^p)$; and

C3. $(\hat{V}, \hat{E}, \hat{i}, \hat{c})$ is an input instance of Π^c.

Denote by $\mathcal{F}$ the set containing the finite view graphs of all candidates for phase p. In phase p, node v computes $L_p(v, I^p)$ (requires p rounds) and based on that, constructs the set $\mathcal{F}$.

Note that the set $\mathcal{F}$ can be totally ordered in a predetermined way. To see this, observe that for any finite view graph $G_* = (V_*, E_*, \ell_*)$, the set V_* can be totally ordered in a predetermined way similarly to the order used in Sec-

Algorithm: $\mathcal{A}_*$, a deterministic algorithm solving Π^c at node v

▷ *Initialization of variables for node v:*
$(\hat{V}_*, \hat{E}_*, \hat{\imath}_*, \hat{c}_*, \hat{b}_*) \leftarrow$ the empty labeled graph.
$\overset{*}{v} \leftarrow$ NULL
$b(v) \leftarrow$ the empty bitstring
for *phase* $p \leftarrow 1, \ldots$ **do**
  `Update-Graph`(p)
  `Update-Output`(p)
  `Update-Bits`(p)

Procedure `Update-Graph`(p)**:**
$L \leftarrow L_p(v, I^p)$ ▷ $b(v)$ *is treated as a labeling function*
construct the set $\mathcal{F}$ of candidates for phase p
if $\mathcal{F} = \emptyset$ **then**
  skip phase p
else
  $\hat{G}_* = (\hat{V}_*, \hat{E}_*, \hat{\imath}_*, \hat{c}_*, \hat{b}_*) \leftarrow$ the smallest graph in $\mathcal{F}$
  $\hat{G} = (\hat{V}, \hat{E}, \hat{\imath}, \hat{c}, \hat{b}) \leftarrow$ a candidate that corresponds to $\hat{G}_*$
  $\hat{v} \leftarrow$ the node $\hat{v} \in \hat{V}$ such that $L_p(\hat{v}, \hat{G}) = L_p(v, I^p)$ as assured by C2
  $q \leftarrow |\hat{V}_*|$
  $\overset{*}{v} \leftarrow$ the node $\overset{*}{v} = L_q(\hat{v}, \hat{G}) \in \hat{V}_*$ with $L_p(\hat{v}, \hat{G}) = L$

Procedure `Update-Output` (p)**:**
$\sigma \leftarrow$ the simulation of $\mathcal{A}_{\mathrm{R}}$ on $(\hat{V}_*, \hat{E}_*, \hat{\imath}_*)$ induced by $\hat{b}_*$
if σ *is successful* **then set** $o(v)$ to be the output of $\overset{*}{v}$ in σ

Procedure `Update-Bits` (p)**:**
$\mathcal{B} \leftarrow \{b : b$ is a p-extension of $\hat{b}_*$, and the simulation of $\mathcal{A}_{\mathrm{R}}$ on $(\hat{V}_*, \hat{E}_*, \hat{\imath}_*)$ induced by b is successful$\}$
if $\mathcal{B} \neq \emptyset$ **then**
  $b_{\min} \leftarrow$ the smallest $b \in \mathcal{B}$
  $b(v) \leftarrow b_{\min}(\overset{*}{v})$

Figure 3: The deterministic algorithm $\mathcal{A}_*$ that solves Π^c.

tion 2.1. This total order on V_* fully determines a representation of G_* as a finite bitstring $s = s(G_*)$ (encoding the ordinal number and label of every node as well as every edge in G_*). Given two finite view graphs $G_* = (V_*, E_*, \ell_*)$ and $G'_* = (V'_*, E'_*, \ell'_*)$, we write $G_* < G'_*$ if either $|V_*| < |V'_*|$ or $|V_*| = |V'_*|$ and $s(G_*) < s(G'_*)$ lexicographically.

If the set $\mathcal{F}$ is empty in phase p, then node v skips the remainder of this phase. Otherwise, `Update-Graph` selects the smallest finite view graph $\hat{G}_* = (\hat{V}_*, \hat{E}_*, \hat{\imath}_*, \hat{c}_*, \hat{b}_*) \in \mathcal{F}$. Let $\hat{G}$ be a candidate that corresponds to $\hat{G}_*$ and recall that condition C2 guarantees the existence of a node $\hat{v} \in \hat{V}$ such that $L_p(\hat{v}, \hat{G}) = L_p(v, I^p)$. Let $\overset{*}{v} = L_q(\hat{v}, \hat{G})$, where $q = |\hat{V}_*|$, be the node in $\hat{V}_*$ that corresponds to v.

`Update-Output`.

Node v simulates $\mathcal{A}_{\mathrm{R}}$ on the instance $J = (\hat{V}_*, \hat{E}_*, \hat{\imath}_*)$ using the bitstrings provided by $\hat{b}_*$ as a replacement for $\mathcal{A}_{\mathrm{R}}$'s random bits. Recall that $\hat{b}_*$ corresponds to the bitstring assignment $\hat{b}$ in some candidate $\hat{G}$ and as such, reflects the bitstring assignment b^p of I^p. Since b^p may assign bitstrings of varying lengths to the nodes in V, $\hat{b}_*$ may also assign bitstrings of varying lengths to the nodes in $\hat{V}_*$. Therefore, the simulation of $\mathcal{A}_{\mathrm{R}}$ on J, denoted by σ, lasts for l rounds, where $l = \min\{\mathrm{length}(\hat{b}_*(\overset{*}{u})) : \overset{*}{u} \in \hat{V}_*\}$ is the length of the shortest bitstring assigned under $\hat{b}_*$ to the nodes in $\hat{V}_*$. If the simulation σ is successful (recall the definition of a successful simulation in Section 2.2), then `Update-Output` sets v's output to the value returned by node $\overset{*}{v}$ in σ.

`Update-Bits`.

The task of `Update-Bits` is to update the value of $b(v)$, extending it to a bitstring of length p. An assignment $\hat{b}'_* : \hat{V}_* \to \{0,1\}^p$ is said to be a *p-extension* of $\hat{b}_*$ if the bitstring $\hat{b}_*(\overset{*}{u})$ is a prefix of $\hat{b}'_*(\overset{*}{u})$ for every node $\overset{*}{u} \in \hat{V}_*$. Let $\mathcal{B}$ be the set of p-extensions of $\hat{b}_*$ that induce successful simulations of $\mathcal{A}_{\mathrm{R}}$ on $J = (\hat{V}_*, \hat{E}_*, \hat{\imath}_*)$.

If $\mathcal{B}$ is empty, then $b(v)$ remains unchanged. Otherwise, the predetermined total order on $\hat{V}_*$ implies a predetermined total order on $\mathcal{B}$ — let $b_{\min}$ be the smallest bitstring assignment in $\mathcal{B}$ according to this total order and update the bitsring $b(v)$ so that $b(v) \leftarrow b_{\min}(\overset{*}{v})$.

3.2 Analysis

In our effort to prove Theorem 1, we need to show two things: (1) algorithm $\mathcal{A}_*$ terminates; and (2) the output of $\mathcal{A}_*$ is valid. We use the same notation as in Section 3.1 to denote the various graphs involved with $\mathcal{A}_*$. Recall that the description of $\mathcal{A}_*$ in Section 3.1 is provided from the perspective of node v in phase p and when necessary, we explicitly mention p and/or v by adding them as a superscript.

Specifically, let

- $I^p_* = (V^p_*, E^p_*, i^p_*, c^p_*, b^p_*)$ be the finite view graph of I^p;
- $\mathcal{F}^{p,v}$ be the set of finite view graphs from `Update-Graph`;
- $\hat{G}^{p,v}_* = (\hat{V}^{p,v}_*, \hat{E}^{p,v}_*, \hat{i}^{p,v}_*, \hat{c}^{p,v}_*, \hat{b}^{p,v}_*) \in \mathcal{F}^{p,v}$ be the finite view graph selected by `Update-Graph`;
- $\hat{v}^p_*$ be the node in $\hat{V}^{p,v}_*$ corresponding to v;
- $J^{p,v} = (\hat{V}^{p,v}_*, \hat{E}^{p,v}_*, \hat{i}^{p,v}_*)$ be the graph used to simulate $\mathcal{A}_{\mathrm{R}}$ in `Update-Output`; and
- $\sigma^{p,v}$ be the corresponding simulation of $\mathcal{A}_{\mathrm{R}}$ on $J^{p,v}$ induced by $\hat{b}^{p,v}_*$.

We further denote by $I^c_* = (V_*, E_*, i_*, c_*)$ the finite view graph of I^c and set $n = |V_*|$.

Termination.

Recall the node $\hat{v}$ in a candidate $\hat{G}$ promised by condition C2; we henceforth also refer to the node $\hat{v}^*$ in the finite view graph of $\hat{G}$ that corresponds to $\hat{v}$ as being *promised* by condition C2. Our analysis of $\mathcal{A}_*$ begins with the following insight regarding the graphs in $\mathcal{F}^{p,v}$, which follows from Corollary 1 as $|\hat{V}^{p,v}_*| \leq p$.

Corollary 3. *Consider some $\hat{G}'_* = (\hat{V}'_*, \hat{E}'_*, \hat{i}'_*, \hat{c}'_*, \hat{b}'_*) \in \mathcal{F}^{p,v}$ and let $\hat{v}^* \in \hat{V}'_*$ be the node promised by condition C2. Then, $L_p(\hat{v}^*, \hat{G}_*) = L_p(v, I^p)$.*

Denote by $\hat{H}^{p,v}_* = (\hat{V}^{p,v}_*, \hat{E}^{p,v}_*, \hat{i}^{p,v}_*, \hat{c}^{p,v}_*)$ the graph obtained from $\hat{G}^{p,v}_*$ by ignoring the labeling $\hat{b}^{p,v}_*$. To establish that $\mathcal{A}_*$ terminates, we show that the graph $\hat{H}^{p,v}_*$ "converges" towards I^c_* as cast in the following lemma.

Lemma 5. *There exists some q such that for every phase $p \geq q$, the graph $\hat{H}^{p,v}_*$ satisfies $\hat{H}^{p,v}_* \cong I^c_*$ for all nodes $v \in V$.*

The difficulty in proving Lemma 5 is that the finite view graphs $\hat{G}^{p,v}_*$ are constructed based on the local views in (V, E, i, c, b^p) rather than (V, E, i, c) — in particular, the labels $b^p(v)$ are constantly changing. Observe however that in $\mathcal{A}_*$, the value $b^{p+1}(v)$ depends solely on $L_p(v, I^p)$. This means that for every two nodes $u, v \in V$ and phase p, if $L_p(u, I^p) = L_p(v, I^p)$, then $b^{p+1}(u) = b^{p+1}(v)$. By induction on p, we conclude that $L_\infty(u, I^p) = L_\infty(v, I^p)$ if and only if $L_\infty(u, I^c) = L_\infty(v, I^c)$. In other words, nodes indistinguishable without the labeling b^p are also indistinguishable when the labeling b^p is included. This immediately implies that in every phase p, the graph obtained from I^p_∞ by ignoring the labeling b^p is isomorphic to I^c_∞, which derives the following observation due to Corollary 2.

Observation 1. *For every phase p under algorithm $\mathcal{A}_*$, it holds that $(V^p_*, E^p_*, i^p_*, c^p_*) \cong I^c_*$.*

Utilizing Observation 1, Lemma 5 can be established by showing that $\hat{G}^{p,v}_* = I^p_*$. The following Lemma 6 assures that I^p_* is among the candidates in all phases $p \geq n$.

Lemma 6. *If $p \geq n$, then the set $\mathcal{F}^{p,v}$ contains I^p_* for every node $v \in V$.*

Proof. We establish the assertion by showing that I^p_* (or a graph isomorphic to it) is a candidate for phase p, noticing that $(I^p_*)_* = I^p_*$. By Observation 1, we conclude that $|V^p_*| = |V^c_*| = n \leq p$, thus condition C1 holds. Since I^p_* is a factor of I^p, Fact 1 guarantees that node $\hat{v}^* = L_n(v, I^p) \in V^p_*$ satisfies $L_p(\hat{v}^*, I^p_*) = L_p(v, I^p)$, thus condition C2 holds as well. As already argued in Section 2.3.2, the lifting lemma guarantees that I^c_∞ is an instance of Π^c, therefore by Corollary 2, so is I^p_*, implying that condition C3 holds which completes the proof. □

Lemma 6 confirms that I^p_* *may* be selected by `Update-Graph` if $p \geq n$. It remains to show that there is a phase p in which I^p_* *will* be selected by `Update-Graph`. The following Lemma 7 confirms that the latter occurs if $p \geq 2n$, thus establishing Lemma 5.

Lemma 7. *If $p \geq 2n$, then $\hat{G}^{p,v}_* = I^p_*$ for all nodes $v \in V$.*

Proof. Lemma 6 guarantees that $I^p_* \in \mathcal{F}^{p,v}$ for every node $v \in V$. The assertion is established by showing that I^p_* is the smallest graph in $\mathcal{F}^{p,v}$ according to the total order used in `Update-Graph`.

Let $\hat{G}'_* = (\hat{V}'_*, \hat{E}'_*, \hat{i}'_*, \hat{c}'_*, \hat{b}'_*) \in \mathcal{F}^{p,v}$ be the finite view graph of some candidate $\hat{G}'$ and set $n' = |\hat{V}_*|$. Assume for the sake of contradiction that $\hat{G}_* < I^p_*$. This implies that either (1) $n' < n$; or (2) $n' = n$ and $s(\hat{G}'_*) < s(I^p_*)$. We will show that neither (1) nor (2) hold and thus, contradict $\hat{G}'_* < I^p_*$.

To that end, let $\hat{v}'^* \in \hat{V}'_*$ and $\hat{v}^* \in V^p_*$ be the nodes in $\hat{G}'_*$ and I^p_*, respectively, promised by property C2. Since $|V^p_*| = n$ and $|V'_*| = n' \leq n$, it follows that the diameter of both I^p_* and $\hat{G}'_*$ is at most $n - 1$. Since $p \geq 2n$, the local view $L_p(\hat{v}'^*, \hat{G}'_*)$ contains the sub-tree $\hat{u}'^*$ for every $u' \in \hat{V}'_*$ and each distinct depth-n' sub-tree of $L_p(\hat{v}'^*, \hat{G}'_*)$ corresponds to a different node $\hat{u}'^* \in \hat{V}'_*$. Similarly, the local view $L_p(\hat{v}^*, I^p_*)$ contains the sub-tree $\hat{u}^*$ for every $u \in V^p_*$ and each distinct depth-n sub-tree of $L_p(\hat{v}^*, I^p_*)$ corresponds to a node $\hat{u}^* \in V^p_*$.

Corollary 3 guarantees that $L_p(\hat{v}'^*, \hat{G}'_*) = L_p(v, I^p) = L_p(\hat{v}^*, I^p_*)$. Therefore, the depth-$n'$ truncating function $f_{n'}$ maps every node $\hat{u}^* \in V^p_*$ to a node in $\hat{V}'_*$. It follows by the definition of local view that $\hat{G}'_* \preceq_{f_{n'}} I^p_*$.

If $n' = n$, then $f_{n'}$ is the identity function, hence $\hat{G}'_* = I^p_*$, in contradiction to the assumption that $\hat{G}'_* < I^p_*$. If on the other hand $n' < n$, then $\hat{G}'_*$ is a non-trivial factor of I^p_*, contradicting the fact that the finite view graph I^p_* is prime. The assertion follows. □

Consider some node $v \in V$ and phase $p \geq 2n$. Lemma 7 guarantees that the graph $J^{p,v}$ on top of which the simulation $\sigma^{p,v}$ is carried out is in fact (V_*, E_*, i_*) — denote this graph by $\widetilde{J}$. The design of `Update-Graph` ensures that $\widetilde{J} \in I$ and the reasoning from Section 2.2 can be applied to show that all nodes $u \in V$ perform the same simulation $\sigma^{p,u}$ on $\widetilde{J}$ — denote this simulation by σ^p and let $b^p : V_* \to \{0,1\}^*$ be the bitstring assignment that induces σ^p.

Let z be the smallest integer $z \geq 2n$ so that there exists a z-extension of b^{2n} that induces a successful simulation on $\widetilde{J}$ and let b' be the smallest such z-extension according to the predetermined total order on the bitstring assignments. Notice that the integer z is well defined since $\mathcal{A}_{\mathrm{R}}$ is guaranteed to produce a correct output with probability 1. The design of `Update-Bits` ensures that in phase z, every node $u \in V$ updates $b(u) \leftarrow b'(\hat{u}^*)$, where $\hat{u}^*$ is the node in V_* that corresponds to u. The design of `Update-Output` then ensures that in phase $z + 1$, all nodes set their outputs according to the successful simulation σ^{z+1}, thus establishing the termination of $\mathcal{A}_*$ as cast in the following lemma.

LEMMA 8. *In phase $z+1$, all nodes $v \in V$ set their outputs $\mathcal{A}_*(v)$.*

Correctness.

It remains to show that the output produced by $\mathcal{A}_*$ is correct. Denote by $o^p(v)$ the output of node $v \in V$ in phase p of $\mathcal{A}_*$, where we use the designated symbol ε to indicate that v does not return any output in phase p, writing $o^p(v) = \varepsilon$. Intuitively, we establish the correctness of $\mathcal{A}_*$ by arguing that there exists an execution η of $\mathcal{A}_{\mathrm{R}}$ on $I = (V, E, i)$ such that for every node $v \in V$ and integer $1 \leq p \leq z+1$, if $o^p(v) \neq \varepsilon$, then the output of node v in *round* p under η, denoted by $o^p_\eta(v)$, is $o^p_\eta(v) = o^p(v)$. (In fact, the execution η is obtained by lifting σ^{z+1} from V_* to V.) The correctness of $\mathcal{A}_*$ then follows from the correctness of $\mathcal{A}_{\mathrm{R}}$.

LEMMA 9. *For any phase p, if node v returns an output $o^p(v) \neq \varepsilon$ in phase p, then $o^p(v) = o^p_\eta(v)$.*

PROOF. Let p be a phase, and suppose that node $v \in V$ sets its output in phase p to $o^p(v) \neq \varepsilon$. Since v sets $o^p(v)$ in phase p, the graph $\hat{G}^{p,v}_*$ is defined and the simulation $\sigma^{p,v}$ of the randomized algorithm $\mathcal{A}_{\mathrm{R}}$ is successful. With that in mind, let t be the length of $\sigma^{p,v}$, and let η be the execution of $\mathcal{A}_{\mathrm{R}}$ on $I = (V, E, i)$ obtained by lifting σ^{z+1} from V_* to V. The goal now is to show that $o^p(v) = o^p_\eta(v)$.

Note that for any $k > 0$, the first k rounds in η are fully determined by the first k random bits of each node $u \in V$ and their respective input values. More specifically, the first k rounds in η for a *single* node v are fully determined by $L_k(v, I)$ and the first $k - i$ bits replacing the random bits of each node $u \in \mathcal{H}^i(v)$, $0 \leq i \leq k$, where $\mathcal{H}^i(v)$ the set of all nodes at most i hops away from v, i.e, $\mathcal{H}^0(v) = \{v\}$ and $\mathcal{H}^{i+1}(v) = \mathcal{H}^i(v) \cup \Gamma(\mathcal{H}^i(v))$ for every $i \geq 0$.

By the construction of $J^{p,v}$ the equality $L_p(\mathring{v}^p, J^{p,v}) = L_p(v, I)$ holds. Moreover, for every $u \in \mathcal{H}^p(v)$, there exists a node $\mathring{u} \in \hat{V}^{p,v}_*$ such that the bitstring assigned to $\mathring{u}$ satisfies $\hat{b}^{p,v}_*(\mathring{u}) = b^p(u)$. Since v sets $o^p(v)$ in `Update-Output`, the simulation $\sigma^{p,v}$ is successful and thus, the length of $\hat{b}^{p,v}_*(\mathring{u})$ is at least t. The design of `Update-Bits` ensures that $b^p(u)$ is a prefix of $b^{p+1}(u)$ for every phase p and node $u \in V$. Therefore, the first $t - i$ bits assigned to node $u \in \mathcal{H}^i(v)$, $0 \leq i \leq t$ are the same $t - i$ bits that are used in the first t rounds of η. We conclude that in η node v returns the output $o^t_\eta(v) = o^p(v)$ at node v in round p. □

Lemma 9 is sufficient to establish the correctness of $\mathcal{A}_*$ as well: Using the same line of arguments as in Section 2.3.2, once more invoking the lifting lemma, one can show that the output obtained by lifting the output of simulation σ^{z+1} is valid for I (and I^c). By combining Lemmas 8 and 9, Theorem 1 now follows.

4. FIBRATIONS AND 2-HOP COLORINGS

Boldi and Vigna [13] extensively study the notion of fibrations, roughly speaking a generalization of factorizing maps to edge-colored directed graphs (refer to [13] for an exact definition). A special case the two authors study are *deterministic (edge) colorings* which require that for every node, all out-edges must be colored differently. In this section we wish to highlight a connection between our observations regarding 2-hop colored graphs in Section 2 and deterministically edge colored directed graphs. In the following, we write *undirected* as well as *directed* edges as tuples (u, v). It will be clear from the context whether we are referring to a directed or an undirected edge.

Let $G = (V, E, c)$ be a 2-hop colored undirected graph, and consider the edge colored directed graph $H = (V', E', c')$ obtained by (1) choosing $V' = V$; (2) adding two directed edges (u, v) and (v, u) to E' for every undirected edge $(u, v) \in E$; and (3) setting $c'(e) = \langle c(u), c(v) \rangle$ for every directed edge $e = (u, v) \in E'$. In the terminology of [13], the graph H is symmetric, since for every edge (u, v) a symmetric edge (v, u) is present. Moreover the edge coloring c' is deterministic, and c' respects the edge symmetries since for every edge $e = (u, v)$, colored $\langle c_1, c_2 \rangle$, the symmetric edge $\bar{e} = (v, u)$ is colored $\langle c_2, c_1 \rangle$. We call the graph H obtained from G in this manner as being G's *directed (edge colored) representation*. Note that reversing the construction, in hope to obtain a 2-hop colored graph, is not possible for general deterministically edge-colored symmetric directed graphs.

Observe that a fibration $\varphi : H \to H'$, where H and H' are directed representations of two graphs G and G', translates to a factorizing map $f : G \to G'$ as defined in Section 2, and vice versa. One can use this connection to derive the statements from Section 2 from the results presented in [13].

5. ACKNOWLEDGEMENTS

We would like to thank our anonymous reviewers for their invaluable comments.

6. REFERENCES

[1] K. R. Abrahamson, A. Adler, L. Higham, and D. G. Kirkpatrick. Probabilistic solitude verification on a ring. In J. Y. Halpern, editor, *PODC*, pages 161–173. ACM, 1986.

[2] Y. Afek and Y. Matias. Elections in anonymous networks. *Inf. Comput.*, 113(2):312–330, 1994.

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

[4] A. Amit, N. Linial, J. Matousek, and E. Rozenman. Random lifts of graphs. In S. R. Kosaraju, editor, *SODA*, pages 883–894. ACM/SIAM, 2001.

[5] D. Angluin. Local and global properties in networks of processors (extended abstract). In R. E. Miller, S. Ginsburg, W. A. Burkhard, and R. J. Lipton, editors, *STOC*, pages 82–93. ACM, 1980.

[6] D. Angluin, J. Aspnes, M. Chan, M. J. Fischer, H. Jiang, and R. Peralta. Stably computable properties of network graphs. In V. K. Prasanna, S. S. Iyengar, P. G. Spirakis, and M. Welsh, editors, *DCOSS*, volume 3560 of *Lecture Notes in Computer Science*, pages 63–74. Springer, 2005.

[7] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. In Chaudhuri and Kutten [15], pages 290–299.

[8] D. Angluin and A. Gardiner. Finite common coverings of pairs of regular graphs. *J. Comb. Theory, Ser. B*, 30(2):184–187, 1981.

[9] J. Beauquier, P. Blanchard, and J. Burman. Self-stabilizing leader election in population protocols over arbitrary communication graphs. In R. Baldoni,

N. Nisse, and M. van Steen, editors, *OPODIS*, volume 8304 of *Lecture Notes in Computer Science*, pages 38–52. Springer, 2013.

[10] J. Beauquier, M. Gradinariu, and C. Johnen. Memory space requirements for self-stabilizing leader election protocols. In Coan and Welch [16], pages 199–207.

[11] P. Boldi and S. Vigna. Computing anonymously with arbitrary knowledge. In Coan and Welch [16], pages 181–188.

[12] P. Boldi and S. Vigna. An effective characterization of computability in anonymous networks. In J. L. Welch, editor, *DISC*, volume 2180 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 2001.

[13] P. Boldi and S. Vigna. Fibrations of graphs. *Discrete Mathematics*, 243(1-3):21–66, 2002. Using the author's version from `http://vigna.dsi.unimi.it/ftp/papers/FibrationsOfGraphs.pdf`.

[14] P. Boldi and S. Vigna. Universal dynamic synchronous self-stabilization. *Distributed Computing*, 15(3):137–153, 2002.

[15] S. Chaudhuri and S. Kutten, editors. *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing, PODC 2004, St. John's, Newfoundland, Canada, July 25-28, 2004*. ACM, 2004.

[16] B. A. Coan and J. L. Welch, editors. *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing, PODC, '99Atlanta, Georgia, USA, May 3-6, 1999*. ACM, 1999.

[17] S. Dolev, A. Israeli, and S. Moran. Uniform dynamic self-stabilizing leader election. *IEEE Trans. Parallel Distrib. Syst.*, 8(4):424–440, 1997.

[18] S. Dolev, E. Schiller, and J. L. Welch. Random walk for self-stabilizing group communication in ad-hoc networks. In *SRDS*, pages 70–79. IEEE Computer Society, 2002.

[19] Y. Emek and R. Wattenhofer. Stone age distributed computing. In P. Fatourou and G. Taubenfeld, editors, *PODC*, pages 137–146. ACM, 2013.

[20] P. Fraigniaud, A. Korman, M. Parter, and D. Peleg. Randomized distributed decision. In M. K. Aguilera, editor, *DISC*, volume 7611 of *Lecture Notes in Computer Science*, pages 371–385. Springer, 2012.

[21] P. Fraigniaud, A. Korman, and D. Peleg. Local distributed decision. In R. Ostrovsky, editor, *FOCS*, pages 708–717. IEEE, 2011.

[22] A. H. Gebremedhin, F. Manne, and A. Pothen. Parallel distance-k coloring algorithms for numerical optimization. In B. Monien and R. Feldmann, editors, *Euro-Par*, volume 2400 of *Lecture Notes in Computer Science*, pages 912–921. Springer, 2002.

[23] E. Godard, Y. Métivier, and A. Muscholl. Characterizations of classes of graphs recognizable by local computations. *Theory Comput. Syst.*, 37(2):249–293, 2004.

[24] C. D. Godsil and G. Royle. *Algebraic Graph Theory*. Graduate Texts in Mathematics. Springer, 2001.

[25] M. Göös and J. Suomela. Locally checkable proofs. In C. Gavoille and P. Fraigniaud, editors, *PODC*, pages 159–168. ACM, 2011.

[26] A. Itai and M. Rodeh. Symmetry breaking in distributive networks. In *FOCS*, pages 150–158. IEEE Computer Society, 1981.

[27] A. Itai and M. Rodeh. Symmetry breaking in distributed networks. *Inf. Comput.*, 88(1):60–87, 1990.

[28] A. Korman, S. Kutten, and D. Peleg. Proof labeling schemes. In M. K. Aguilera and J. Aspnes, editors, *PODC*, pages 9–18. ACM, 2005.

[29] S. O. Krumke, M. V. Marathe, and S. S. Ravi. Models and approximation algorithms for channel assignment in radio networks. *Wireless Networks*, 7(6):575–584, 2001.

[30] F. Kuhn, T. Moscibroda, and R. Wattenhofer. What cannot be computed locally! In Chaudhuri and Kutten [15], pages 300–309.

[31] F. T. Leighton. Finite common coverings of graphs. *J. Comb. Theory, Ser. B*, 33(3):231–238, 1982.

[32] C. Lenzen, J. Suomela, and R. Wattenhofer. Local algorithms: Self-stabilization on speed. In R. Guerraoui and F. Petit, editors, *SSS*, volume 5873 of *Lecture Notes in Computer Science*, pages 17–34. Springer, 2009.

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

[34] M. Luby. A simple parallel algorithm for the maximal independent set problem. In R. Sedgewick, editor, *STOC*, pages 1–10. ACM, 1985.

[35] S. T. McCormick. Optimal approximation of sparse hessians and its equivalence to a graph coloring problem. *Mathematical Programming*, 26(2):153–171, 1983.

[36] Y. Métivier, J. M. Robson, and A. Zemmari. Analysis of fully distributed splitting and naming probabilistic procedures and applications - (extended abstract). In T. Moscibroda and A. A. Rescigno, editors, *SIROCCO*, volume 8179 of *Lecture Notes in Computer Science*, pages 153–164. Springer, 2013.

[37] Y. Métivier, N. Saheb, and A. Zemmari. Randomized local elections. *Inf. Process. Lett.*, 82(6):313–320, 2002.

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

[39] N. Norris. Universal covers of graphs: Isomorphism to depth $n - 1$ implies isomorphism to all depths. *Discrete Applied Mathematics*, 56(1):61–74, 1995.

[40] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, 2000.

[41] M. K. Ramanathan, R. A. Ferreira, S. Jagannathan, A. Grama, and W. Szpankowski. Randomized leader election. *Distributed Computing*, 19(5-6):403–418, 2007.

[42] B. Schieber. Calling names in nameless networks. In P. Rudnicki, editor, *PODC*, pages 319–328. ACM, 1989.

[43] M. Yamashita and T. Kameda. Computing on anonymous networks: Part i-characterizing the solvable cases. *IEEE Trans. Parallel Distrib. Syst.*, 7(1):69–89, 1996.