Locally Restricted Proof Labeling Schemes

Yuval Emek
Technion – Israel Institute of Technology, Haifa, Israel

Yuval Gil
Technion – Israel Institute of Technology, Haifa, Israel

Shay Kutten
Technion – Israel Institute of Technology, Haifa, Israel

Abstract

Introduced by Korman, Kutten, and Peleg (PODC 2005), a *proof labeling scheme (PLS)* is a distributed verification system dedicated to evaluating if a given *configured graph* satisfies a certain property. It involves a centralized *prover*, whose role is to provide proof that a given configured graph is a yes-instance by means of assigning *labels* to the nodes, and a distributed verifier, whose role is to verify the validity of the given proof via local access to the assigned labels. In this paper, we introduce the notion of a *locally restricted* PLS in which the prover's power is restricted to that of a LOCAL algorithm with a polylogarithmic number of rounds. To circumvent inherent impossibilities of PLSs in the locally restricted setting, we turn to models that relax the correctness requirements by allowing the verifier to accept some no-instances as long as they are not "too far" from satisfying the property in question. To this end, we evaluate (1) *distributed graph optimization problems (OptDGPs)* based on the notion of an *approximate proof labeling scheme (APLS)* (analogous to the type of relaxation used in sequential approximation algorithms); and (2) *configured graph families (CGFs)* based on the notion of a *testing proof labeling schemes (TPLS)* (analogous to the type of relaxation used in property testing algorithms). The main contribution of the paper comes in the form of two generic compilers, one for OptDGPs and one for CGFs: given a black-box access to an APLS (resp., PLS) for a large class of OptDGPs (resp., CGFs), the compiler produces a locally restricted APLS (resp., TPLS) for the same problem, while losing at most a $(1+\epsilon)$ factor in the scheme's relaxation guarantee. An appealing feature of the two compilers is that they only require a logarithmic additive label size overhead.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Theory of computation → Approximation algorithms analysis

Keywords and phrases proof labeling schemes, generic compilers, SLOCAL algorithms

Digital Object Identifier 10.4230/LIPIcs.DISC.2022.20

Related Version *Full Version*: `https://arxiv.org/abs/2208.08718` [8]

Funding This work was supported in part by the Technion Hiroshi Fujiwara Cyber Security Research Center and the Israel National Cyber Directorate. In addition, the work of Shay Kutten was also supported in part by ISF grant 1346/22.

1 Introduction

A *proof system* is a tool designed to verify the correctness of a certain claim. It is composed of two entities: a *prover*, whose role is to provide proof for the claim in question; and a computationally bounded *verifier* that seeks to verify the validity of the given proof. The crux of a proof system is that the proof given by the prover cannot be blindly trusted. That is, for a proof system to be correct, the verifier must be able to distinguish between an honest prover, providing a correct proof, and a malicious prover who tries to convince the verifier of a false claim.

In the realm of *distributed computing*, the study of proof systems, also known as *distributed proof systems*, has attracted considerable attention. The goal of a distributed proof system is to decide if a given *configured graph* satisfies a certain property. This is typically done by means of a centralized prover, that has a global view of the entire configured graph, and a distributed verifier, that operates at all nodes concurrently and is subject to locality restrictions. Various models for distributed proof systems have been introduced in the literature, including *proof labeling schemes (PLSs)* [19], *locally checkable proofs* [16], *nondeterministic local decisions* [12], and *distributed interactive proofs* [17].

The current paper focuses on the PLS model, introduced by Korman, Kutten, and Peleg [19] (see Sec. 2.1 for the formal definition). In a PLS, the prover generates its proof by means of assigning a *label* to each node. The verification process performed by the verifier at each node v has access to v's label and to the labels of v's neighbors, but it cannot access the labels assigned to nodes outside its local neighborhood. The correctness requirements state that if the given configured graph is a yes-instance, then all nodes must accept; and if the given configured graph is a no-instance, then at least one node must reject. The standard performance measure of a PLS is its *proof size*, defined to be the size of the largest label assigned by the honest prover.

Recently, there is a growing interest from (sequential) computational complexity researchers in *doubly efficient* proof systems [15, 26]. These proof systems are characterized by restricting the (honest) prover to "efficient computations" – i.e., polynomial time algorithms – on top of the restrictions imposed on the computational power of the (still weaker) verifier. For example, Goldwasser et al. [15] consider polynomial time provers vs. logarithmic space verifiers, whereas Reingold et al. [26] consider polynomial time provers vs. linear time and near-linear time verifiers.

Motivated by the success story of doubly efficient proof systems in sequential computational complexity, in this paper, we initiate the study of such proof systems in the distributed computing realm. To do so, we adjust the notion of "efficient computations" from sequential algorithms running in polynomial time to LOCAL algorithms running in a polylogarithmic number of rounds [25]. This introduces a new type of PLSs, called *locally restricted* PLSs, where the label assigned to a node v is computed by the (honest) prover based on the subgraph induced by the nodes within polylogarithmic distance from v, rather than the whole graph (refer to Sec. 2.2 for a formal definition).

Beyond the theoretical interest that lies in this new type of distributed proof systems, we advocate for their investigation also from a more practical point of view: A natural application of PLSs is local checking for self-stabilizing algorithms [2] which involves a detection module and a correction module. In this mechanism, the verifier's role is played by the detection module and the prover's role is played by a dedicated sub-module of the correction module responsible for the label assignment to the nodes [19] (the correction module typically includes another sub-module, responsible for constructing the actual solution, which is abstracted away by the PLS). Since both modules operate as distributed algorithms, any attempt to implement them in practice should take efficiency considerations into account. While classic PLSs consider this efficiency requirement (only) from the verifier's point of view, in locally restricted PLSs, we impose efficiency demands on both the verifier and the prover.

It turns out that locally restricted PLSs are impossible for many interesting properties, regardless of proof size (as shown in the simple observation presented in Appendix B). This leads us to slightly relax the correctness requirements of a PLS so that the verifier may also accept no-instances as long as they are not "too far" from satisfying the property in question. Specifically, we consider locally restricted schemes in the context of two relaxed models called *approximate proof labeling schemes (APLS)* [6, 7] and *testing proof labeling schemes (TPLS)*.

Table 1 Locally restricted APLS (left) and TPLS (right) results, where ℓ stands for the proof size and α stands for the approximation ratio.

OptDGP	graph family	$\boldsymbol{\alpha}$	$\boldsymbol{\ell}$
min. weight vertex cover	any	$2(1+\epsilon)$	$O(\log n)$
min. vertex cover	odd-girth $= \omega(\log n)$	$1+\epsilon$	$O(\log n)$
max. ind. set	any	$\Delta(1+\epsilon)$	$O(\log n)$
	odd-girth $= \omega(\log n)$	$1+\epsilon$	$O(\log n)$
min. weight dom. set	any	$O(\log n)$	$O(\log n)$
any canonical OptDGP	any	$1+\epsilon$	$O(n^2)$

CGF	ℓ
planarity	$O(\log n)$
bounded arboricity	$O(\log n)$
k-colorability	$O(\log n)$
forest	$O(\log n)$
DAG	$O(\log n)$

The APLS model was introduced by Censor-Hillel et al. [6] and studied further by Emek and Gil [7]. For an approximation parameter $\alpha \geq 1$, the goal of an α-APLS for a *distributed graph optimization problem (OptDGP)* is to distinguish between optimal instances and instances that are α-far from being optimal (refer to Sec. 2 for the definitions). Interestingly, for some classic edge-based *covering/packing* OptDGPs (e.g., maximum matching and minimum edge cover), locally restricted APLSs are already established in previous works [16, 6, 7]. In contrast, the existing APLSs for node-based covering/packing OptDGPs require that the prover has a global view of the given configured graph (see, e.g., the APLS for minimum weight vertex cover presented in [7]). In Sec. 4, we develop a generic compiler that gets a (not necessarily locally restricted) α-APLS for an OptDGP Ψ, belonging to a large class of node-based covering/packing OptDGPs, and generates a locally restricted $((1+\epsilon)\alpha)$-APLS for Ψ, where ϵ is a constant performance parameter. The proof size of the locally restricted $((1+\epsilon)\alpha)$-APLS generated by our compiler is $\ell_{\Psi,\alpha} + O(\log n)$, where $\ell_{\Psi,\alpha}$ is the proof size of the α-APLS provided to the compiler. Refer to Section 4.3 for a high-level overview of this construction.

The TPLS model is developed in the current paper based on the notion of *property testing* [14, 1]. For a parameter $\delta > 0$, the goal of a δ-TPLS for a *configured graph family (CGF)* Φ is to distinguish between configured graphs belonging to Φ and configured graphs that are δ-far from belonging to Φ, where the distance here is measured in terms of the graph topology. In Sec. 5, we develop a generic compiler that gets a (not necessarily locally restricted) PLS for a CGF Φ, that is closed under node-induced subgraphs and disjoint union, and generates a locally restricted δ-TPLS for Φ. The proof size of the locally restricted δ-TPLS generated by our compiler is $\ell_\Phi + O(\log n)$, where ℓ_Φ is the proof size of the PLS provided to the compiler. Refer to Section 5.2 for a high-level overview of this construction.

The applicability of our compilers is demonstrated in Appendix A, where we show how the two compilers can be used to obtain APLSs and TPLSs for various well-known OptDGPs and CGFs, respectively; refer to Table 1 for a summary of these results. We conclude with additional related work presented in Appendix C.

1.1 Paper's Organization

In Section 2, we present the model. Preliminaries are presented in Section 3. Following that, in Sections 4 and 5, we present our compiler for OptDGPs and CGFs, respectively. Within these sections, a high-level overview of the compilers appears in Subsections 4.3 and 5.2.

2 Model

We consider *distributed verification systems* in which evaluated instances are called *configured graphs*. A configured graph $G_s = \langle G, s \rangle$ is a pair consisting of an undirected graph $G = (V, E)$ and a *configuration function* $s : V \to \{0,1\}^*$ that assigns a bit string $s(v)$, referred to as v's *local configuration*, to each node $v \in V$. Throughout this paper, we use the notation $n = |V|$ and $m = |E|$.

For a node $v \in V$, we stick to the convention that $N_G(v) = \{u \mid (u, v) \in E\}$ denotes the set of v's *neighbors* in G and that $\deg_G(v) = |N_G(v)|$ denotes v's *degree* in G. When G is clear from the context, we may omit it from our notations and use $N(v)$ and $\deg(v)$ instead of $N_G(v)$ and $\deg_G(v)$, respectively.

We assume that all configured graphs considered in the context of this paper are *identified*, i.e., the configuration function $s : V \to \{0,1\}^*$ assigns a unique id of size $O(\log n)$, denoted by $id(v)$, to each node $v \in V$. Moreover, we assume that the local configuration $s(v)$ distinguishes between node v's incident edges by means of a set $\mathcal{A}(v)$ of abstract *port names*, and a bijection $\rho_v^s : N(v) \to \mathcal{A}(v)$, referred to as the *internal port name assignment* of v, that assigns a (locally unique) port name $\rho_v^s(u)$ to each node $u \in N(v)$. More concretely, assume that the local configuration $s(v)$ includes a designated field for each neighbor $u \in N(v)$ and that this field is indexed by $\rho_v^s(u)$. Unless stated otherwise, when we refer to an ordered list $u_1, \ldots, u_{\deg(v)}$ of v's neighbors, it is assumed that the list is ordered by v's internal port name assignment.

Given a configured graph G_s consisting of graph $G = (V, E)$ and configuration function s, we say that a configured graph $G'_{s'}$ consisting of graph $G' = (V', E')$ and configuration function s', is a *configured subgraph* of G_s if (1) G' is a subgraph of G, i.e., $V' \subseteq V$ and $E' \subseteq E$; and (2) the configuration function s' is the projection of s on G', where for each node $v \in V'$, the fields corresponding to nodes $u \in N_G(v) \setminus N_{G'}(v)$ are omitted from the local configuration $s'(v)$ and the internal port name assignment $\rho_v^{s'}$ associated with s' is defined so that $\rho_v^{s'}(u) = \rho_v^s(u)$ for each $u \in N_{G'}(v)$. For a node subset $U \subseteq V$, let $G(U)$ denote the subgraph induced on G by U and let $G_s(U)$ be the configured subgraph of G_s defined over the subgraph $G(U)$.

We define a *configured graph family (CGF)* as a collection of configured graphs. A CGF type that plays a central role in this paper is that of a *distributed graph problem (DGP)* Π, where for each configured graph $G_s \in \Pi$, the configuration function s is composed of an *input assignment* $\mathsf{i} : V \to \{0,1\}^*$ and an *output assignment* $\mathsf{o} : V \to \{0,1\}^*$. We refer to such a configured graph as an *input-output (IO) graph* and often denote it by $G_{\mathsf{i},\mathsf{o}}$. The input assignment i assigns to each node $v \in V$, a bit string $\mathsf{i}(v)$, referred to as v's *local input*, that encodes attributes associated with v and its incident edges (e.g., node ids, edge orientations, edge weights, and node weights); whereas the output assignment o assigns a *local output* $\mathsf{o}(v)$ to each node $v \in V$. For an input assignment i, we refer to the configured graph $G_\mathsf{i} = \langle G, \mathsf{i} \rangle$ as an *input graph*.

Consider a DGP Π. An input graph G_i is said to be *legal* (and the graph G and input assignment i are said to be *co-legal*) if there exists an output assignment o such that $G_{\mathsf{i},\mathsf{o}} \in \Pi$, in which case we say that o is a *feasible solution* for G_i (or simply for G and i). For a DGP Π, we denote the set of legal input graphs by $\mathcal{LEG}(\Pi) = \{G_\mathsf{i} \mid \exists \mathsf{o} : G_{\mathsf{i},\mathsf{o}} \in \Pi\}$.

A *distributed graph minimization problem (MinDGP)* (resp., *distributed graph maximization problem (MaxDGP)*) Ψ is a pair $\langle \Pi, f \rangle$, where Π is a DGP and $f : \Pi \to \mathbb{Z}$ is a function, referred to as the *objective function* of Ψ, that maps each IO graph $G_{\mathsf{i},\mathsf{o}} \in \Pi$ to an integer value $f(G_{\mathsf{i},\mathsf{o}})$.[1] Given a co-legal graph G and input assignment i, define

[1] We assume for simplicity that the images of the objective functions used in the context of this paper are

$$OPT_\Psi(G,\mathsf{i}) = \inf_{\mathsf{o}:G_{\mathsf{i},\mathsf{o}}\in\Pi}\{f(G_{\mathsf{i},\mathsf{o}})\}$$

if Ψ is a MinDGP; and

$$OPT_\Psi(G,\mathsf{i}) = \sup_{\mathsf{o}:G_{\mathsf{i},\mathsf{o}}\in\Pi}\{f(G_{\mathsf{i},\mathsf{o}}\}$$

if Ψ is a MaxDGP. We often use the general term *distributed graph optimization problem (OptDGP)* to refer to MinDGPs as well as MaxDGPs. Given an OptDGP $\Psi = \langle \Pi, f\rangle$ and co-legal graph G and input assignment i, the output assignment o is said to be an *optimal solution* for G_i (or simply for G and i) if o is a feasible solution for G_i and $f(G_{\mathsf{i},\mathsf{o}}) = OPT_\Psi(G,\mathsf{i})$.

2.1 Proof Labeling Schemes

In this section we present the notion of proof labeling schemes as well as its approximation variants. To that end, we first present the notion of gap proof labeling schemes, as defined in [7].

Fix some universe $\mathcal{U}$ of configured graphs. A *gap proof labeling scheme (GPLS)* is a mechanism designed to distinguish the configured graphs in a *yes-family* $\mathcal{F}_Y \subset \mathcal{U}$ from the configured graphs in a *no-family* $\mathcal{F}_N \subset \mathcal{U}$, where $\mathcal{F}_Y \cap \mathcal{F}_N = \emptyset$. This is done by means of a (centralized) *prover* and a (distributed) *verifier* that play the following roles: Given a configured graph $G_s \in \mathcal{U}$, if $G_s \in \mathcal{F}_Y$, then the prover assigns a bit string $L(v)$, called the *label* of v, to each node $v \in V$. Let $L^N(v) = \langle L(u_1), \ldots, L(u_{\deg(v)})\rangle$ be the vector of labels assigned to v's neighbors. The verifier at node $v \in V$ is provided with the 3-tuple $\langle s(v), L(v), L^N(v)\rangle$ and returns a Boolean value $\varphi(v)$.

We say that the verifier *accepts* G_s if $\varphi(v) = \texttt{True}$ for all nodes $v \in V$; and that the verifier *rejects* G_s if $\varphi(v) = \texttt{False}$ for at least one node $v \in V$. The GPLS is said to be *correct* if the following requirements hold for every configured graph $G_s \in \mathcal{U}$:

▶ **R1.** *If $G_s \in \mathcal{F}_Y$, then the prover produces a label assignment $L : V \to \{0,1\}^*$ such that the verifier accepts G_s.*

▶ **R2.** *If $G_s \in \mathcal{F}_N$, then for any label assignment $L : V \to \{0,1\}^*$, the verifier rejects G_s.*

We emphasize that no requirements are made for configured graphs $G_s \in \mathcal{U} \setminus (\mathcal{F}_Y \cup \mathcal{F}_N)$; in particular, the verifier may either accept or reject these configured graphs (the same holds for configured graphs that do not belong to the universe $\mathcal{U}$). The performance of a GPLS is measured by means of its *proof size* defined to be the maximum length of a label $L(v)$ assigned by the prover to the nodes $v \in V$ assuming that $G_s \in \mathcal{F}_Y$.

Proof Labeling Schemes for CGFs. Consider some CGF Φ and let $\mathcal{U}$ be the universe of all configured graphs. A *proof labeling scheme (PLS)* for Φ is the GPLS over $\mathcal{U}$ defined by setting the yes-family to be $\mathcal{F}_Y = \Phi$; and the no-family to be $\mathcal{F}_N = \mathcal{U} \setminus \mathcal{F}_Y$. In other words, a PLS for Φ determines whether a given configured graph G_s belongs to Φ.

integral. Lifting this assumption and allowing for real numerical values would complicate some of the arguments, but it does not affect the validity of our results.

In this paper, we also define a relaxed model of PLSs for a CGF Φ in which we allow the verifier to accept configured graphs that are not "too far" from belonging to Φ. To that end, we use the following distance measure which is widely used in the realm of property testing (see e.g., [1]).

let G_s and $G'_{s'}$ be two configured graphs. Given a parameter $\delta > 0$, we say that G_s and $G'_{s'}$ are *δ-close* if $G'_{s'}$ is a configured subgraph of G_s and G' can be obtained from G by removing at most δm edges (or vice versa).

Consider a CGF Φ. We say that a configured graph G_s is *δ-far* from belonging to Φ if $G'_{s'} \notin \Phi$ for any configured graph $G'_{s'}$ which is δ-close to G_s. We define a *δ-testing proof labeling scheme (δ-TPLS)* in the same way as a PLS for Φ with the sole difference that the no-family is defined by setting $\mathcal{F}_N = \{G_s \mid G_s \text{ is } \delta\text{-far from belonging to } \Phi\}$.

Proof Labeling Schemes for OptDGPs. Consider some OptDGP $\Psi = \langle \Pi, f \rangle$ and let $\mathcal{U} = \{G_{\mathsf{i},\mathsf{o}} \mid G_{\mathsf{i}} \in \mathcal{LEG}(\Pi)\}$. A *proof labeling scheme (PLS)* for Ψ is defined as a GPLS over $\mathcal{U}$ by setting the yes-family to be

$$\mathcal{F}_Y = \{G_{\mathsf{i},\mathsf{o}} \in \Pi \mid f(G_{\mathsf{i},\mathsf{o}}) = OPT_\Psi(G, \mathsf{i})\}$$

and the no-family to be $\mathcal{F}_N = \mathcal{U} \setminus \mathcal{F}_Y$. In other words, a PLS for Ψ determines for a given IO graph $G_{\mathsf{i},\mathsf{o}} \in \mathcal{U}$ whether the output assignment $\mathsf{o} : V \to \{0,1\}^*$ is an optimal solution (which means in particular that it is a feasible solution) for the co-legal graph $G = (V, E)$ and input assignment $\mathsf{i} : V \to \{0,1\}^*$.

In the realm of OptDGPs, a relaxed model called approximate proof labeling scheme has been considered in [6, 7]. In this model, the correctness requirement of a PLS are relaxed so that it may also accept feasible solutions that only approximate the optimal ones. Specifically, given an approximation parameter $\alpha \geq 1$, an *α-approximate proof labeling scheme (α-APLS)* for an OptDGP $\Psi = \langle \Pi, f \rangle$ is defined in the same way as a PLS for Ψ with the sole difference that the no-family is defined by setting

$$\mathcal{F}_N = \begin{cases} \mathcal{U} \setminus \{G_{\mathsf{i},\mathsf{o}} \in \Pi \mid f(G_{\mathsf{i},\mathsf{o}}) \leq \alpha \cdot OPT_\Psi(G, \mathsf{i})\} , & \text{if } \Psi \text{ is a MinDGP} \\ \mathcal{U} \setminus \{G_{\mathsf{i},\mathsf{o}} \in \Pi \mid f(G_{\mathsf{i},\mathsf{o}}) \geq OPT_\Psi(G, \mathsf{i})/\alpha\} , & \text{if } \Psi \text{ is a MaxDGP} \end{cases} .$$

2.2 Locally Restricted Proof Labeling Schemes

In this paper, we focus on provers whose power is limited as follows. We say that a GPLS is *locally restricted* if there exists a constant c such that for every configured graph $G_s = \langle G = (V, E), s \rangle \in \mathcal{F}_Y$ and for every node $v \in V$, the label $L(v)$ is computed by the prover as a function of $G_s(B^r(v))$, where $r = \log^c n$ and $B^r(v)$ denotes the set of nodes at (hop) distance at most r from v in G. Equivalently, the prover is restricted to a distributed algorithm operating under the LOCAL model [22, 25] with polylogarithmic rounds. We emphasize that if $G_s \in \mathcal{F}_N$, then the verifier is required to reject G_s for any label assignment, including label assignments that were not produced in a locally restricted fashion.

3 Preliminaries

Sequentially Local Algorithms. In the *sequentially local (SLOCAL)* model, introduced in [13], each node $v \in V$ maintains two (initially empty) bit strings denoted by info(v) and decision(v). Nodes are processed sequentially in an arbitrary order $p = v_1, \ldots, v_n$ (i.e., irrespective of node ids). We refer to the time that node v_i is processed as the i-th iteration

of the algorithm. In the i-th iteration, v_i has a read/write access to info(u) for all nodes $u \in B^r(v_i)$, where $r \in \mathbb{Z}_{\geq 0}$ is a parameter referred to as the *locality* of the algorithm. Following that, v_i writes an irrevocable value into decision(v_i) based strictly on $G_s(B^r(v_i))$ and the bit strings info(u) of all $u \in B^r(v_i)$.

A consequence of the seminal work of Ghafari et al. [13, 27] is that any SLOCAL algorithm with $\log^{O(1)} n$ locality can be simulated by a LOCAL algorithm with $\log^{O(1)} n$ rounds. Therefore, in the context of a locally restricted GPLS, by allowing the prover to compute the label $L(v)$ of each node $v \in V$ using an SLOCAL algorithm with locality $r = \log^{O(1)} n$ (rather than a LOCAL algorithm with $\log^{O(1)} n$ rounds), we do not increase the scheme's power.

Comparison Schemes. Let $\mathcal{U}$ be a universe of configured graphs $G_{s_{a,b}}$, such that $G = (V, E)$ is a connected undirected graph and the configuration function $s_{a,b} : V \to \{0,1\}^*$ assigns two values $a(v), b(v) \in \mathbb{R}$ to each node $v \in V$. A *comparison scheme* is a mechanism whose goal is to decide if $\sum_{v \in V} a(v) \geq \sum_{v \in V} b(v)$ for a given configured graph $G_{s_{a,b}} \in \mathcal{U}$. Formally, a comparison scheme is defined as a GPLS over $\mathcal{U}$ by setting the yes-family to be $\mathcal{F}_Y = \{G_{s_{a,b}} \in \mathcal{U} \mid \sum_{v \in V} a(v) \geq \sum_{v \in V} b(v)\}$; and the no family to be $\mathcal{F}_N = \mathcal{U} \setminus \mathcal{F}_Y$.

In [19, Lemma 4.4], Korman et al. present a generic design for comparison schemes as follows. Consider a configured graph $G_{s_{a,b}} \in \mathcal{U}$. The label assignment $L : V \to \{0,1\}^*$ constructed by the prover encodes a spanning tree of G rooted at some (arbitrary) node $r \in V$ (see [19, Lemma 2.2] for details on spanning tree construction). In addition, the prover encodes the sum of $a(\cdot)$ and $b(\cdot)$ values in the sub-tree rooted at node v for each $v \in V$. This allows the verifier to check that the sums assigned at each node $v \in V$ are correct (using the sums assigned to v's children); and the verifier at the root r can evaluate if $G_{s_{a,b}} \in \mathcal{F}_Y$. The proof size of this scheme is $O(\log n + M_{a,b})$, where $M_{a,b}$ is the maximum length (in bits) of values $\sum_{v \in U} a(v)$ and $\sum_{v \in U} b(v)$ over all node-subsets $U \subseteq V$. This comparison scheme construction is used as an auxiliary tool in the compilers presented in Sec. 4 and 5.

4 Compiler for OptDGPs

In this section, we present our generic compiler for OptDGPs. It is divided into five subsections as follows. First, in Sec. 4.1 we characterize the OptDGPs that are suited for our compiler, referred to as canonical OptDGPs, based on the notions of locally checkable labelings and covering/packing OptDGPs (these terms are formally defined in Sec. 4.1). In Sec. 4.2, we establish an important property of optimal solutions for covering/packing OptDGPs that serve the compiler construction. Sec. 4.4 and 4.5 are dedicated to the compiler construction. More formally, these sections constructively prove the following theorem.

▶ **Theorem 4.1.** *Let Ψ be a canonical OptDGP that admits an α-APLS with a proof size of $\ell_{\Psi,\alpha}$. For any constant $\epsilon > 0$, there exists a locally restricted $(\alpha(1+\epsilon))$-APLS for Ψ with a proof size of $\ell_{\Psi,\alpha} + O(\log n)$.*

For convenience, the compiler construction is divided between Sec. 4.4 and 4.5 as follows. In Sec. 4.4, we present an SLOCAL algorithm with logarithmic locality that partitions the nodes into disjoint clusters, such that the subgraph induced by each cluster is of logarithmic diameter. The goal of this partition is to enable the prover to construct the label of a node as a function of the subgraph induced by its cluster (and possibly some nodes adjacent to its cluster) without information on nodes that are farther away. This partition facilitates the label assignment and verification process described in Sec. 4.5. In Sec. 4.3, we provide a high-level overview of the SLOCAL algorithm and how it is used in the label assignment and verification process.

4.1 Canonical OptDGPs

Locally Checkable Labelings. A DGP Π is said to be a *locally checkable labeling (LCL)* (cf. [23]) if there exists a Boolean predicate family $\mathcal{L}^\Pi = \{p^\Pi_{d,\ell} : (\{0,1\}^*)^{d+1} \to \{\texttt{True},\texttt{False}\}\}_{d\in\mathbb{Z}_{\geq 0},\ell\in\{0,1\}^*}$ such that for every legal input graph $G_\mathsf{i} \in \mathcal{LEG}(\Pi)$, an output assignment $\mathsf{o} : V \to \{0,1\}^*$ is a feasible solution for G and i if and only if $p^\Pi_{\deg(v),\mathsf{i}(v)}(\mathsf{o}(v),\mathsf{o}(u_1),\ldots,\mathsf{o}(u_{\deg(v)})) = \texttt{True}$ for every node $v \in V$ with neighbors $u_1,\ldots,u_{\deg(v)}$.

For convenience, we assume that the local input $\mathsf{i}(v)$ of a node $v \in V$ is partitioned into two fields, denoted by $\texttt{prd}(\mathsf{i}(v))$ and $\texttt{data}(\mathsf{i}(v))$, where the former (fully) determines the predicate $p^\Pi_{\deg(v),i(v)}$ associated with $\deg(v)$ and $\mathsf{i}(v)$ and the latter encodes all other pieces of information included in $\mathsf{i}(v)$. This allows us to slightly abuse the notation and write $p^\Pi_{\texttt{prd}(\mathsf{i}(v))}$ instead of $p^\Pi_{\deg(v),\mathsf{i}(v)}$. We further assume that the Boolean predicate family $\mathcal{L}^\Pi$ includes the trivial tautology predicate $taut_d : (\{0,1\}^*)^{d+1} \to \{\texttt{True},\texttt{False}\}$, $d \in \mathbb{Z}_{\geq 0}$, that satisfies $taut_d(x) = \texttt{True}$ for every $x \in (\{0,1\}^*)^{d+1}$ and that this predicate is encoded by writing the designated bit string $taut_d$ in the $\texttt{prd}(\cdot)$ field of the local input.

We say that an LCL Π is *self-induced* if the following two conditions are satisfied for every legal input graph $G_\mathsf{i} \in \mathcal{LEG}(\Pi)$ and node subset $U \subseteq V$: (1) $G_\mathsf{i}(U) \in \mathcal{LEG}(\Pi)$; and (2) if $\mathsf{i}' : V \to \{0,1\}^*$ is the input assignment derived from i by setting $\texttt{data}(\mathsf{i}'(v)) = \texttt{data}(\mathsf{i}(v))$ and $\texttt{prd}(\mathsf{i}'(v)) = taut_{\deg(v)}$ for every $v \in U$, then $G_{\mathsf{i}'} \in \mathcal{LEG}(\Pi)$.

Let Π be an LCL and let $G_\mathsf{i} \in \mathcal{LEG}(\Pi)$. For a subset $U \subseteq V$ of nodes, we denote by $\texttt{inner}(U) = \{u \in U \mid N_G(u) \subseteq U\}$ the set of nodes in U for which every neighbor is in U and define $\texttt{inner}^2(U) = \texttt{inner}(\texttt{inner}(U))$ and $\texttt{rim}(U) = U \setminus \texttt{inner}^2(U)$. We say that a function $g : U \to \{0,1\}^*$ *respects* Π if $p^\Pi_{\texttt{prd}(\mathsf{i}(v))}(g(v),g(u_1),\ldots,g(u_{\deg(v)})) = \texttt{True}$ for each $v \in \texttt{inner}(U)$ with neighbors $u_1,\ldots,u_{\deg(v)}$.

Canonical OptDGPs. A MinDGP (resp., MaxDGP) $\Psi = \langle \Pi, f\rangle$ is said to be a *covering* (resp., *packing*) OptDGP if the following conditions hold: (1) Π is an LCL; (2) for each n-node IO graph $G_{\mathsf{i},\mathsf{o}} \in \Pi$, there exists a positive integer $k = k(\Pi,n) = n^{O(1)}$ such that the output assignment o assigns a nonnegative integer $\mathsf{o}(v) \in \{0,\ldots,k\}$, referred to as v's *multiplicity*, to each node $v \in V$; (3) for each predicate $p^\Pi_{d,\ell} \in \mathcal{L}^\Pi$, if $p^\Pi_{d,\ell}(x_0,x_1,\ldots,x_d) = \texttt{True}$ for nonnegative integers $x_0,x_1,\ldots,x_d \in \{0,\ldots,k\}$, then $p^\Pi_{d,\ell}(x'_0,x'_1,\ldots,x'_d) = \texttt{True}$ for any nonnegative integers $x'_0,x'_1,\ldots,x'_d \in \{0,\ldots,k\}$ that satisfy $x'_j \geq x_j$ (resp., $x'_j \leq x_j$) for all $0 \leq j \leq d$; (4) for every legal input graph $G_\mathsf{i} \in \mathcal{LEG}(\Pi)$, there exists a node-weight function $w : V \to \{1,\ldots n^{O(1)}\}$ such that the weight $w(v)$ of node v is encoded in v's local input field $\texttt{data}(\mathsf{i}(v))$; and (5) $f(G_{\mathsf{i},\mathsf{o}}) = \sum_{v\in V} w(v)\cdot\mathsf{o}(v)$ for every $G_{\mathsf{i},\mathsf{o}} \in \Pi$. The OptDGP $\Psi = \langle\Pi,f\rangle$ is said to be *canonical* if it is covering/packing and Π is self-induced.

Consider a covering/packing OptDGP $\Psi = \langle\Pi,f\rangle$. Let $G_\mathsf{i} \in \mathcal{LEG}(\Pi)$ be a legal input graph with the underlying node-weight function $w : V \to \{1,\ldots,n^{O(1)}\}$ and let $U \subseteq V$. Given a function $g : U \to \{0,\ldots,k(\Pi,n)\}$ that assigns a multiplicity value $g(u)$ to each node $u \in U$, we define $w(U,g) = \sum_{u\in U} w(u)\cdot g(u)$.

For a covering MinDGP Ψ, let $w_{\min}(U)$ denote the minimum possible value of $w(U,g)$ obtained by a function $g : U \to \{0,\ldots,k(\Pi,n)\}$ that respects Π. Let $N^2(U)$ be the set of nodes in $V \setminus U$ at distance at most 2 from a node in U. For a packing MaxDGP, let $w_{\max}(U)$ denote the maximum possible value of $w(U,g)$ obtained by a function $g : U \cup N^2(U) \to \{0,\ldots,k(\Pi,n)\}$ that satisfies (1) $g(v) = 0$ for each node $v \in N^2(U)$; and (2) g respects Π.

4.2 Properties of Optimal Solutions for Covering/Packing OptDGPs

In the following lemmas, we establish important properties regarding optimal solutions of covering and packing OptDGPs. Consider a covering (resp., packing) OptDGP $\Psi = \langle \Pi, f \rangle$. Let $G_{\mathsf{i},\mathsf{o}} \in \Pi$ be an IO graph such that $\mathsf{o} : V \to \{0, \ldots, k(\Pi, n)\}$ is an optimal solution for G and i.

▶ **Lemma 4.2.** *If Ψ is a covering MinDGP, then $w(\mathtt{inner}^2(U), \mathsf{o}) \leq w_{\min}(U)$ for any $U \subseteq V$.*

Proof. Assume by contradiction that $w(\mathtt{inner}^2(U), \mathsf{o}) > w_{\min}(U)$ for some $U \subseteq V$. This means that there exists an assignment $\mathsf{o}' : U \to \{0, \ldots, k(\Pi, n)\}$ that respects Π, such that $w(\mathtt{inner}^2(U), \mathsf{o}) > w(U, \mathsf{o}')$. Let $\tilde{\mathsf{o}}$ be the output assignment defined as follows: $\tilde{\mathsf{o}}(v) = \mathsf{o}(v)$ for all $v \in V \setminus U$; $\tilde{\mathsf{o}}(v) = \mathsf{o}'(v)$ for all $v \in \mathtt{inner}^2(U)$; and $\tilde{\mathsf{o}}(v) = \max\{\mathsf{o}(v), \mathsf{o}'(v)\}$ for all $v \in \mathtt{rim}(U)$. Recall that o is a feasible solution for G and i and that o' respects Π. Since Ψ is a covering OptDGP, we get that $p^{\Pi}_{\mathtt{prd}(\mathsf{i}(v))}(\tilde{\mathsf{o}}(v), \tilde{\mathsf{o}}(u_1), \ldots, \tilde{\mathsf{o}}(u_{\deg_G(v)})) = \mathtt{True}$ for each node $v \in V$ (where $u_1, \ldots, u_{\deg_G(v)}$ denote v's neighbors in G). It follows that $\tilde{\mathsf{o}}$ is a feasible solution with objective value $f(G_{\mathsf{i},\tilde{\mathsf{o}}}) = w(V, \tilde{\mathsf{o}}) \leq w(V \setminus U, \mathsf{o}) + w(\mathtt{inner}^2(U), \mathsf{o}') + w(\mathtt{rim}(U), \mathsf{o}) + w(\mathtt{rim}(U), \mathsf{o}') = w(V, \mathsf{o}) - w(\mathtt{inner}^2(U), \mathsf{o}) + w(U, \mathsf{o}') < w(V, \mathsf{o}) = f(G_{\mathsf{i},\mathsf{o}})$ which contradicts the optimality of o. ◀

▶ **Lemma 4.3.** *If Ψ is a packing MaxDGP, then $w(U, \mathsf{o}) \geq w_{\max}(\mathtt{inner}^2(U))$ for any $U \subseteq V$.*

The proof of Lemma 4.3 is similar to the proof for Lemma 4.2. We defer it to the full version of this paper [8].

4.3 Overview

In Sec. 4.4, we present an SLOCAL algorithm called `Part_OPT` that partitions the nodes of a given IO graph $G_{\mathsf{i},\mathsf{o}}$ into clusters. Before formally describing the algorithm in Sec. 4.4, let us provide some intuition by presenting the high-level idea of the partition and how it is used in the label and verification process (as described in Sec. 4.5) for the case of a canonical MinDGP Ψ (the high-level idea for MaxDGPs is similar and the differences are mostly technical).

Given an IO graph $G_{\mathsf{i},\mathsf{o}}$, where o is an optimal solution, we use a ball growing argument to obtain a partition of the nodes into clusters such that: (1) the subgraph induced by each cluster is of logarithmic diameter; and (2) the total weight of nodes in the rim of clusters (i.e., nodes with distance at most 2 from a different cluster) is an ϵ-fraction of the total weight of inner nodes of clusters (i.e., nodes with distance at least 3 from a different cluster).

The goal of the partition obtained by `Part_OPT` is to allow the prover to compute the label assigned to each node based on its cluster. Essentially, the prover seeks to provide the verifier with a proof that the partition satisfies 2 main properties: (1) for each cluster V_j, the weight of the given solution induced on the inner nodes is at most the weight of an approximately optimal (global) solution induced on the cluster; and (2) the total weight of nodes in the rim of clusters is at most an ϵ-fraction of the total weight of inner nodes.

While providing proof for the first property is rather straightforward using the labels of an α-APLS for Ψ in a black box manner, providing proof for the second property is somewhat more challenging. The reason is that the property as presented above is rather global – not every cluster is guaranteed to have at most an ϵ-fraction of its weight assigned to the rim nodes. Constructing a label that sums the total weights of rim and inner nodes of all clusters is a global task and can not be accomplished in a locally restricted fashion. To that end, during `Part_OPT`, we may assign some nodes with a *secondary affiliation* to an

adjacent cluster. The idea is that for each cluster V_j, the sum between weights of nodes with secondary affiliation to V_j and nodes in $\texttt{rim}(V_j)$ that do not have a secondary affiliation to any cluster is bounded by an ϵ-fraction of V_j's inner nodes weight.

Throughout Part_OPT, each node v maintains a *color* whose role is to keep track of the changeability status of v's secondary affiliation. The color white indicates that the secondary affiliation may still change; whereas black indicates that the secondary affiliation is final.

4.4 Partition Algorithm

Algorithm's Description. We now provide a formal description of the Part_OPT algorithm. Consider a canonical OptDGP $\Psi = \langle \Pi, f \rangle$. Let $G_{\mathsf{i},\mathsf{o}} \in \Pi$ be an IO graph such that o is an optimal solution for G and i. The algorithm partitions the nodes of G into (possibly empty) clusters $V = V_1 \dot\cup \ldots \dot\cup V_n$. As usual in the SLOCAL model, the nodes are processed sequentially in n iterations based on an arbitrary order $v_1, \ldots, v_n$ on the nodes, where node v_j is processed in the j-th iteration.

Throughout the execution of Part_OPT, each node $v \in V$ maintains three fields referred to as cluster(v), sec(v), and color(v). The field cluster(v) is initially empty and its role is to identify v's cluster, where each cluster V_j is identified by the id of node v_j which is processed in the j-th iteration, i.e., $V_j = \{v \mid \text{cluster}(v) = id(v_j)\}$. The field sec$(v)$ is initially empty and its role is to identify v's *secondary affiliation* to a cluster if such affiliation exists (otherwise it remains empty throughout the algorithm). The field color$(v) \in \{\text{black}, \text{white}\}$, initially set to white, maintains v's *color*.

We describe the j-th iteration of Part_OPT as follows. Let G_j be the subgraph induced on G by $V \setminus (V_1 \cup \cdots \cup V_{j-1})$. If $v_j \in V_1 \cup \cdots \cup V_{j-1}$, then we define $V_j = \emptyset$ and finish the iteration; so, assume that v_j is a node in G_j. For an integer $r \in \mathbb{Z}_{\geq 0}$, let D_j^r be the set of nodes at distance exactly r from v_j in G_j and let $B_j^r = \bigcup_{r'=0}^{r} D_j^{r'}$. Let $white_j$ be the set of nodes in G_j that are colored white in the beginning of the j-th iteration.

Suppose that Ψ is a MinDGP. We define $r(j)$ to be the smallest integer that satisfies $w(\texttt{inner}^2(B_j^{r(j)+6}), \mathsf{o}) \leq (1+\epsilon) \cdot w(\texttt{inner}^2(B_j^{r(j)+2}), \mathsf{o})$. Notice that $\texttt{inner}^2(\cdot)$ is taken with respect to nodes in G (and not G_j), i.e., $\texttt{inner}^2(B_j^{r(j)+6})$ (resp., $\texttt{inner}^2(B_j^{r(j)+2})$) is the set of nodes in $B_j^{r(j)+6}$ (resp., $B_j^{r(j)+2}$) for which every node within distance 2 in G is in $B_j^{r(j)+6}$ (resp., $B_j^{r(j)+2}$). In the case that Ψ is a MaxDGP, define $r(j)$ to be the smallest integer that satisfies $w(B_j^{r(j)+6}, \mathsf{o}) \leq (1+\epsilon) \cdot w(B_j^{r(j)+2}, \mathsf{o})$.

Following the computation of $r(j)$, we define the cluster V_j and modify the color and secondary affiliation of some nodes as follows (this process is the same for MinDGPs and MaxDGPs). Let X_j be the set of white nodes in $\texttt{inner}^2(B_j^{r(j)+6})$ at distance exactly $r(j)+3$ from v_j, and let Y_j be the set of white nodes in $\texttt{inner}^2(B_j^{r(j)+6})$ at distance exactly $r(j)+4$ from v_j that have a neighbor in X_j. We complete the j-th iteration by setting cluster$(v) = id(v_j)$ for each node $v \in B_j^{r(j)+2}$ (i.e., setting $V_j = B_j^{r(j)+2}$); sec$(v) = id(v_j)$ for each node $v \in X_j \cup Y_j$; and color$(v) =$ black for each node $v \in X_j$.

Algorithm's Properties. We go on to analyze some properties of Part_OPT. Consider a cluster V_j. Let $\texttt{sec}(V_j) = \{v \mid \text{sec}(v) = id(v_j)\}$ be the set of nodes whose secondary affiliation is to V_j by the end of the algorithm and let $\texttt{ext}(V_j) = V_j \cup \texttt{sec}(V_j)$.

▶ **Lemma 4.4.** *The subgraphs $G(V_j)$ and $G(\texttt{ext}(V_j))$ induced on G by V_j and $\texttt{ext}(V_j)$, respectively, are connected and have diameter $O(\log n)$ for each $j \in [n]$.*

Proof. Suppose that $V_j \neq \emptyset$ (as the lemma is trivial otherwise). First, observe that by definition, all nodes $v \in V_j$ are reachable from v_j in $G(V_j)$, thus $G(V_j)$ is connected.

To see that $G(\texttt{ext}(V_j))$ is connected, we first observe that the subgraph $G(V_j \cup X_j \cup Y_j)$ is connected. By the time cluster V_j is determined, we color the nodes of X_j black. Thus, their secondary affiliation remains to V_j throughout the algorithm. At termination, it follows that $\texttt{ext}(V_j) = V_j \cup X_j \cup Y$ for some $Y \subseteq Y_j$. Since the nodes of Y all have a neighbor in X_j, we get that $G(\texttt{ext}(V_j)) = G(V_j \cup X_j \cup Y)$ is connected.

To show that the diameters of $G(V_j)$ and $G(\texttt{ext}(V_j))$ are $O(\log n)$ it is sufficient to show that $r(j) = O(\log n)$. We use a ball growing argument. By definition, for every $r' < r(j) + 6$, it holds that

$$w(\texttt{inner}^2(B_j^{r(j)+6}), \mathsf{o}) \geq w(\texttt{inner}^2(B_j^{r'}), \mathsf{o}) > (1+\epsilon) \cdot w(\texttt{inner}^2(B_j^{r'-4}), \mathsf{o}) > (1+\epsilon)^2 \cdot w(\texttt{inner}^2(B_j^{r'-8}), \mathsf{o}) > \dots$$

if Ψ is a MinDGP; and

$$w(B_j^{r(j)+6}, \mathsf{o}) \geq w(B_j^{r'}, \mathsf{o}) > (1+\epsilon) \cdot w(B_j^{r'-4}, \mathsf{o}) > (1+\epsilon)^2 \cdot w(B_j^{r'-8}, \mathsf{o}) > \dots$$

if Ψ is a MaxDGP. Since the terms $w(\texttt{inner}^2(B_j^{r(j)+6}), \mathsf{o})$ and $w(B_j^{r(j)+6}, \mathsf{o})$ are both bounded by a polynomial of n, it follows that $r(j) = O((1/\epsilon)\log n) = O(\log n)$ in both cases. ◀

A simple observation derived from Lemma 4.4 is that `Part_OPT` has locality $O(\log n)$. This observation combined with the results of [13, 27] lead to the following corollary.

▶ **Corollary 4.5.** *The algorithm* `Part_OPT` *can be simulated by a LOCAL algorithm with polylogarithmic round-complexity.*

For each $j \in [n]$, define $S_j = \texttt{sec}(V_j) \cup \{v \in \texttt{rim}(V_j) \mid \text{sec}(v) \text{ is empty}\}$ as the set of nodes composed of nodes outside of V_j whose secondary affiliation is to V_j and nodes in $\texttt{rim}(V_j)$ that do not have a secondary affiliation. The following observation establishes an important property regarding the sets $\texttt{rim}(V_j)$ and S_j.

▶ **Observation 4.6.** $\bigcup_{j\in[n]} \texttt{rim}(V_j) \subseteq \bigcup_{j\in[n]} S_j$

Proof. Consider a node $v \in \texttt{rim}(V_j)$ for some $j \in [n]$. By definition, if $\text{sec}(v)$ is empty, then $v \in S_j$. If $\text{sec}(v)$ is not empty, then there exists some $j' \in [n]$ such that $v \in \texttt{sec}(V_{j'})$, and therefore $v \in S_{j'}$. Overall, we get that $v \in \bigcup_{\ell\in[n]} S_\ell$. ◀

4.5 Labels and Verification

In this section, we describe the label assignment and verification process of our compiler for the case of MinDGPs. The description of the changes required to establish the same for MaxDGPs is deferred to the full version of this paper [8]. In both cases, we establish the proof size and correctness of our construction, thus proving Theorem 4.1.

Consider a canonical MinDGP $\Psi = \langle \Pi, f \rangle$ and an IO graph $G_{\mathsf{i},\mathsf{o}} \in \Pi$, where o is an optimal solution for G and i. The prover uses the SLOCAL algorithm `Part_OPT` presented in Sec. 4.4 to compute the values $r(j)$ and subsets V_j, $\texttt{sec}(V_j)$, $\texttt{ext}(V_j) = V_j \cup \texttt{sec}(V_j)$, and $S_j = \texttt{sec}(V_j) \cup \{v \in \texttt{rim}(V_j) \mid \text{sec}(v) \text{ is empty}\}$ for all $j \in [n]$.

The goal of the prover is to provide proof of four properties satisfied by the given solution o and the outcome of `Part_OPT`. We refer to those properties as *feasibility*, *rim*, *growth*, and *optimality*. The four properties are defined as follows. The feasibility property states that o

is a feasible solution for G and i, i.e., $G_{\mathsf{i},\mathsf{o}} \in \Pi$; the rim property states that for each $j \in [n]$ and node $v \in \mathtt{rim}(V_j)$, there exists $j' \in [n]$, such that $v \in S_{j'}$; the growth property states that $w(S_j, \mathsf{o}) \leq \epsilon \cdot w(\mathtt{inner}^2(V_j), \mathsf{o})$ for each $j \in [n]$; and the optimality property states that $w(\mathtt{inner}^2(V_j), \mathsf{o}) \leq \alpha \cdot w_{\min}(V_j)$ for each $j \in [n]$.

The prover provides its proof by means of a label assignment $L : V \to \{0,1\}^*$ that assigns each node v with a label $L(v) = \langle L_{\text{feas}}(v), L_{\text{rim}}(v), L_{\text{grw}}(v), L_{\text{opt}}(v)\rangle$. The label $L(v)$ is composed of the fields $L_{\text{feas}}(v)$, $L_{\text{rim}}(v)$, $L_{\text{grw}}(v)$, and $L_{\text{opt}}(v)$ that provide proof for the feasibility, rim, growth, and optimality properties, respectively.

The field $L_{\text{feas}}(\cdot)$ provides a proof for the feasibility property by setting $L_{\text{feas}}(v) = \mathsf{o}(v)$ for each node $v \in V$. Notice that since Π is an LCL, verifying o's feasibility is done by checking that $L_{\text{feas}}(v) = \mathsf{o}(v)$, and $p^{\Pi}_{\mathtt{prd}(\mathsf{i}(v))}(L_{\text{feas}}(v), L_{\text{feas}}(u_1), \ldots L_{\text{feas}}(u_{\deg_G(v)})) = \mathtt{True}$ at each node v with neighbors $u_1, \ldots, u_{\deg_G(v)}$.

The field $L_{\text{rim}}(\cdot)$ provides a proof for the rim property as follows. First, the sets V_j and S_j are encoded for all $1 \leq j \leq n$, where each of the sets is identified by $id(v_j)$. In addition, each node $v \in \mathtt{rim}(V_j)$ is assigned the minimal distance to a node $u \notin V_j$ (notice that by definition, these values are either 1 or 2). This allows the verifier to check that for each node $v \in \mathtt{rim}(V_j)$, there exists $j' \in [n]$ such that $v \in S_{j'}$, i.e., verify that the rim property is satisfied.

The field $L_{\text{grw}}(\cdot)$ provides a proof for the growth property simply by using a comparison scheme (as defined in Sec. 3) that compares between $w(S_j, \mathsf{o})$ and $\epsilon \cdot w(\mathtt{inner}^2(V_j), \mathsf{o})$. This comparison scheme is used concurrently for each $\mathtt{ext}(V_j) \neq \emptyset$, based on a shortest paths tree of $G(\mathtt{ext}(V_j))$ rooted at node v_j. Observe that by Lemma 4.4, this tree spans the nodes of $\mathtt{ext}(V_j)$ and has diameter $O(\log n)$.

The field $L_{\text{opt}}(\cdot)$ provides a proof for the optimality property as follows. First, for each $V_j \neq \emptyset$, the prover computes an assignment $g_j : V_j \to \{0, \ldots, k(\Pi, n)\}$, such that g_j respects Π and $w(V_j, g_j) = w_{\min}(V_j)$. The prover assigns each node $v \in V_j$ with the multiplicity $g_j(v)$ and proves that $w(\mathtt{inner}^2(V_j), \mathsf{o}) \leq w(V_j, g_j)$ by means of a comparison scheme based on a shortest paths (spanning) tree of $G(V_j)$ rooted at node v_j. Finally, the prover proves that $w(V_j, g_j) \leq \alpha \cdot w_{\min}(V_j)$ by means of an α-APLS for Ψ on the configured subgraph $G_{\mathsf{i},\mathsf{o}}(V_j)$. Notice that an α-APLS for Ψ is well-defined over the instance $G_{\mathsf{i},\mathsf{o}}(V_j)$ since Π is self-induced, and thus $G_{\mathsf{i}}(V_j) \in \mathcal{LEG}(\Pi)$.

Proof Size and Correctness. We observe that the label assignment produced by the prover can be computed by means of an SLOCAL algorithm with locality $O(\log n)$ and thus it can be simulated by a locally restricted prover. Moreover, for each node $v \in V$, the sub-labels $L_{\text{feas}}(v)$, $L_{\text{rim}}(v)$, and $L_{\text{grw}}(v)$ are of size $O(\log n)$; whereas $L_{\text{opt}}(v)$ is of size $\ell_{\Psi,\alpha} + O(\log n)$, where $\ell_{\Psi,\alpha}$ is the proof size of an α-APLS for Ψ. Overall, the proof size of this scheme is $\ell_{\Psi,\alpha} + O(\log n)$.

Regarding the correctness requirements, we start by showing the completeness requirement, i.e., we show that if o is an optimal solution for G and i, then the verifier accepts $G_{\mathsf{i},\mathsf{o}}$. To that end, it is sufficient to show that all four aforementioned properties are satisfied. The feasibility property holds since by definition, o is a feasible solution for G and i; the rim property follows directly from Observation 4.6; the growth property holds by the construction of the clusters V_j; and the optimality property follows from Lemma 4.2. We note that as established in Lemma 4.2, the optimality property is satisfied by o with parameter $\alpha = 1$. However, providing proof for this stronger property might be costly in terms of proof size. Thus, to obtain a small proof size, we settle for an approximated version.

As for the soundness requirement, consider an IO graph $G_{\mathsf{i},\mathsf{o}}$ such that the verifier accepts $G_{\mathsf{i},\mathsf{o}}$. This means that all four properties hold for $G_{\mathsf{i},\mathsf{o}}$. First, observe that by the feasibility property, it holds that $G_{\mathsf{i},\mathsf{o}} \in \Pi$. Let $V_1^L, \ldots V_k^L$ and $S_1^L, \ldots S_k^L$ be the subsets V_j and S_j encoded by the prover in the field $L_{\text{rim}}(\cdot)$. By the rim property, it holds that $\bigcup_{j\in[k]} \texttt{rim}(V_j^L) \subseteq \bigcup_{j\in[k]} S_j^L$. From the growth property it follows that $\epsilon \cdot w(\bigcup_{j\in[k]} \texttt{inner}^2(V_j^L), \mathsf{o}) \geq w(\bigcup_{j\in[k]} S_j^L, \mathsf{o}) \geq w(\bigcup_{j\in[k]} \texttt{rim}(V_j^L), \mathsf{o})$. Let $\mathsf{o}^* : V \to \{0, \ldots, k(\Pi, n)\}$ be an optimal solution for G and i. We observe that for any $U \subseteq V$, the assignment of o^* on the nodes of U must respect Π, and therefore $w(U, \mathsf{o}^*) \geq w_{\min}(U)$. The optimality property combined with the last observation implies that $w(\bigcup_{j\in[k]} \texttt{inner}^2(V_j^L), \mathsf{o}) \leq \alpha(w_{\min}(V_1^L) + \cdots + w_{\min}(V_k^L)) \leq \alpha(w(V_1^L, \mathsf{o}^*) + \cdots + w(V_k^L, \mathsf{o}^*)) = \alpha \cdot w(V, \mathsf{o}^*) = \alpha \cdot f(G_{\mathsf{i},\mathsf{o}^*})$. Combining this inequality with the rim and growth properties implies that $f(G_{\mathsf{i},\mathsf{o}}) = w(V, \mathsf{o}) = w(\bigcup_{j\in[k]} \texttt{inner}^2(V_j^L), \mathsf{o}) + w(\bigcup_{j\in[k]} \texttt{rim}(V_j^L), \mathsf{o}) \leq (1+\epsilon) \cdot w(\bigcup_{j\in[k]} \texttt{inner}^2(V_j^L), \mathsf{o}) \leq \alpha \cdot (1+\epsilon) \cdot f(G_{\mathsf{i},\mathsf{o}^*})$, thus establishing the soundness requirement.

5 Compiler for CGFs

In this section, we present our generic compiler for CGFs. It is divided into three subsections as follows. First, in Sec. 5.1 we characterize the CGFs that are suited for our compiler, namely SU-closed CGFs. Following that, Sec. 5.3 and 5.4 are dedicated to the compiler construction. More formally, these sections constructively prove the following theorem.

▶ **Theorem 5.1.** *Let Φ be an SU-closed CGF that admits a PLS with a proof size of ℓ_Φ. For any constant $\delta > 0$, there exists a locally restricted δ-TPLS for Φ with a proof size of $\ell_\Phi + O(\log n)$.*

For convenience, the compiler construction is divided between Sec. 5.3, in which we present an SLOCAL partition algorithm (that plays a similar role to the one presented in the OptDGP compiler), and Sec. 5.4, in which we describe the label assignment and verification process. In Sec. 5.2, we provide a high-level overview of the SLOCAL algorithm and how it is used in the label assignment and verification process.

5.1 SU-Closed CGFs

A CGF Φ is said to be *closed under node-induced subgraphs* if for every configured graph $G_s \in \Phi$ and node subset $U \subseteq V$, it holds that $G_s(U) \in \Phi$. We say that two configured graphs $G_s = \langle G = (V, E), s\rangle$ and $G'_{s'} = \langle G' = (V', E'), s'\rangle$ are *disjoint* if $V \cap V' = \emptyset$. We define the *disjoint union* between two disjoint configured graphs $G_s = \langle G = (V, E), s\rangle$ and $G'_{s'} = \langle G' = (V', E'), s'\rangle$ as the configured graph $\tilde{G}_{\tilde{s}} = \langle \tilde{G}, \tilde{s}\rangle$ consisting of the graph $\tilde{G} = (V \dot\cup V', E \dot\cup E')$ and the configuration function $\tilde{s} : V \dot\cup V' \to \{0,1\}^*$ that assigns the local configuration $\tilde{s}(v) = s(v)$ to any node $v \in V$; and $\tilde{s}(v) = s'(v)$ to any node $v \in V'$. We say that a CGF Φ is *closed under disjoint union* if for any two disjoint configured graphs $G_s, G'_{s'} \in \Phi$ with disjoint union $\tilde{G}_{\tilde{s}}$, it holds that $\tilde{G}_{\tilde{s}} \in \Phi$. We refer to a CGF Φ as *SU-closed* if it is closed under node-induced subgraphs and under disjoint union.

5.2 Overview

In Sec. 5.3, we present an SLOCAL algorithm called `Part_CGF` that partitions the nodes of a given configured graph G_s into clusters. Before formally describing the algorithm in Sec. 5.3, let us provide some intuition by presenting the high-level idea of the partition and how it is used to design the label assignment and verification process for an SU-closed CGF Φ.

Given a configured graph $G_s \in \Phi$, we use a ball growing argument to obtain a partition of the nodes into clusters such that: (1) the subgraph induced by each cluster is of logarithmic diameter; and (2) the number of crossing edges between clusters is a δ-fraction of the number of edges in the clusters.

In order to allow the prover to provide a proof for the second property by local means, during `Part_CGF`, some nodes may be assigned a *secondary affiliation* to an adjacent cluster. The idea is that for each cluster V_j, the number of crossing edges to nodes with secondary affiliation to V_j is a δ-fraction of the number of edges within V_j.

5.3 Partition Algorithm

Algorithm's Description. We now provide a formal description of the `Part_CGF` algorithm. Consider an SU-closed CGF Φ and a configured graph $G_s \in \Phi$. The algorithm partitions the nodes of G into (possibly empty) clusters $V = V_1 \dot\cup \ldots \dot\cup V_n$. As usual in the SLOCAL model, the nodes are processed sequentially in n iterations based on an arbitrary order $v_1, \ldots, v_n$ on the nodes, where node v_j is processed in the j-th iteration.

Throughout the execution of `Part_CGF`, each node $v \in V$ maintains two fields referred to as cluster(v) and sec(v). The field cluster(v) is initially empty and its role is to identify v's cluster, where each cluster V_j is identified by the id of node v_j which is processed in the j-th iteration, i.e., $V_j = \{v \mid \text{cluster}(v) = id(v_j)\}$. The field sec$(v)$ is initially empty and its role is to identify v's *secondary affiliation* to a cluster if such affiliation exists (otherwise it remains empty throughout the algorithm).

The j-th iteration of `Part_CGF` is executed as follows. Let G_j to be the subgraph induced on G by $V \setminus (V_1 \cup \cdots \cup V_{j-1})$. If $v_j \in V_1 \cup \cdots \cup V_{j-1}$, then we define $V_j = \emptyset$ and finish the iteration; so, assume that v_j is a node in G_j. For an integer $r \in \mathbb{Z}_{\geq 0}$, let D_j^r be the set of nodes at distance exactly r from v_j in G_j and let $B_j^r = \bigcup_{r'=0}^{r} D_j^{r'}$. Let E_j^r be the set of edges in the subgraph $G(B_j^r)$ and let C_j^r be the set of edges $(u, v) \in E$, such that $u \in B_j^r$ and $v \notin B_j^r$. Define $r(j)$ to be the smallest integer that satisfies $|C_j^{r(j)}| \leq \delta \cdot |E_j^{r(j)}|$. The j-th iteration is completed by setting cluster$(v) = id(v_j)$ for each node $v \in B_j^{r(j)}$; and sec$(v) = id(v_j)$ for each node $v \in D_j^{r(j)+1}$.

Algorithm's Properties. The following lemma establishes an upper bound on the diameter of each subgraph $G(V_j)$.

▶ **Lemma 5.2.** *The diameter of subgraph $G(V_j)$ is $O(\log n)$ for each $j \in [n]$.*

Proof. Suppose that $V_j \neq \emptyset$ (as the lemma is trivial otherwise). To show that the diameter of $G(V_j)$ is $O(\log n)$ it is sufficient to show that $r(j) = O(\log n)$. We use a ball growing argument. Note that for any integer r, it holds that $|E_j^r| \geq |E_j^{r-1}| + |C_j^{r-1}|$. Thus, for every $r' < r(j) + 1$, we have

$$|E_j^{r(j)+1}| \geq |E_j^{r'}| > (1+\delta) \cdot |E_j^{r'-1}| > (1+\delta)^2 \cdot |E_j^{r'-2}| > \ldots$$

and since $n^2 > m \geq |E_j^{r(j)+1}|$, we get that $r(j) = O((1/\delta) \log n) = O(\log n)$. ◀

A simple observation derived from Lemma 5.2 is that `Part_CGF` has locality $O(\log n)$. This observation combined with the results of [13, 27] lead to the following corollary.

▶ **Corollary 5.3.** *The algorithm `Part_CGF` can be simulated by a LOCAL algorithm with polylogarithmic round-complexity.*

5.4 Labels and Verification

Consider an SU-closed CGF Φ and a configured graph $G_s \in \Phi$. The prover uses the SLOCAL algorithm `Part_CGF` presented in Sec. 5.3 to compute the values $r(j)$ for all $1 \leq j \leq n$, and the fields cluster(v), sec(v). The goal of the prover is to provide proof of two properties satisfied by the given configured graph G_s and the outcome of `Part_CGF`. We refer to those properties as *secondary clusters*, *crossing edges*, and *inclusion*. To that end, the prover produces a label assignment $L : V \to \{0,1\}^*$ that assigns each node v with a label $L(v) = \langle L_{\text{sec}}(v), L_{\text{cross}}(v), L_{\text{inc}}(v)\rangle$. The label $L(v)$ is composed of the fields $L_{\text{sec}}(v)$, $L_{\text{cross}}(v)$, and $L_{\text{inc}}(v)$, that provide proof for the secondary clusters, crossing edges and inclusion properties, respectively.

The secondary clusters property states that sec(v) is not empty for every node v that has a neighbor belonging to a different cluster. To that end, the sub-label $L_{\text{sec}}(v)$ assigns the values cluster(v) and sec(v) to each node $v \in V$. Observe that this information is sufficient for the verifier to verify the secondary clusters property.

For all $j \in [n]$, let $\texttt{sec}(V_j) = \{v \mid \text{sec}(v) = id(v_j)\}$ be the set of nodes whose secondary affiliation is to V_j by the end of the `Part_CGF` algorithm, and let $F_j = \{(u,v) \in E \mid u \in V_j, v \in \texttt{sec}(V_j)\}$ denote the set of edges with one endpoint in V_j and the other endpoint in $\texttt{sec}(V_j)$. The crossing edges property states that each cluster V_j satisfies $|F_j| \leq \delta \cdot |E_j^{r(j)}|$. The field $L_{\text{cross}}(\cdot)$ serves the crossing edges property by means of a comparison scheme between $|F_j|$ and $\delta \cdot |E_j^{r(j)}|$. This comparison scheme is based on a shortest paths (spanning) tree rooted at node v_j for each cluster $V_j \neq \emptyset$. Notice that each node $v \in V_j$ knows its incident edges from $|F_j|$ based on the $L_{\text{sec}}(\cdot)$ field of its neighbors.

The inclusion property states that $G_s(V_j) \in \Phi$ for all $V_j \neq \emptyset$. To that end, the prover uses the field $L_{\text{inc}}(\cdot)$ to encode a PLS for Φ concurrently on all subgraphs $G(V_j)$.

Proof Size and Correctness. We observe that the label assignment produced by the prover can be computed by means of an SLOCAL algorithm with locality $O(\log n)$ and thus it can be simulated by a locally restricted prover. Moreover, the sub-labels $L_{\text{sec}}(v)$ and $L_{\text{cross}}(v)$ are of size $O(\log n)$; and $L_{\text{inc}}(v)$ is of size ℓ_Φ, where ℓ_Φ is the proof size of a PLS for Φ. Overall, the proof size of this scheme is $\ell_\Phi + O(\log n)$.

We now show that the correctness requirements are satisfied. We start with completeness, i.e., showing that if $G_s \in \Phi$, then the verifier accepts G_s. Observe that the secondary clusters and crossing edges properties are satisfied by construction of `Part_CGF`. In addition, the inclusion property follows from the fact that Φ is closed under node-induced subgraphs.

As for the soundness requirement, consider a configured graph G_s such that the verifier accepts G_s. Let $V_1^L, \ldots V_k^L$ be the clusters encoded in the field $L_{\text{sec}}(\cdot)$. For every $j \in [k]$, let E_j^L denote the edge set of subgraph $G(V_j^L)$, let $\texttt{sec}_j^L$ be the set of nodes for which the field $L_{\text{sec}}(\cdot)$ encodes a secondary affiliation to V_j^L, and let F_j^L be the set of edges with one endpoint in V_j^L and one in $\texttt{sec}_j^L$. The inclusion property guarantees that $G_s(V_j^L) \in \Phi$ for each $j \in [k]$. Let $G'_{s'}$ be the disjoint union of $G_s(V_1^L), \ldots, G_s(V_k^L)$. Since Φ is closed under disjoint union, we get that $G'_{s'} \in \Phi$. By the secondary clusters property, $G'_{s'}$ is the configured subgraph obtained from G_s by removing the set $F_1^L \cup \cdots \cup F_k^L$ of edges. The crossing edges property implies that $|F_1^L \cup \cdots \cup F_k^L| = \sum_{j\in[k]} |F_j^L| \leq \delta \cdot \sum_{j\in[k]} |E_j^L| \leq \delta m$. Thus, G_s is not δ-far from belonging to Φ, i.e., $G_s \notin \mathcal{F}_N$.

In conclusion, this scheme describes a correct locally restricted δ-TPLS for SU-closed CGFs with a proof size of $\ell_\Phi + O(\log n)$, thus proving Theorem 5.1.

References

1. Noga Alon, Tali Kaufman, Michael Krivelevich, and Dana Ron. Testing triangle-freeness in general graphs. *SIAM J. Discret. Math.*, 2008.
2. B. Awerbuch, B. Patt-Shamir, and G. Varghese. Self-stabilization by local checking and correction. In *Proceedings 32nd Annual Symposium of Foundations of Computer Science*, pages 268–277, 1991.
3. Baruch Awerbuch, Andrew V. Goldberg, Michael Luby, and Serge A. Plotkin. Network decomposition and locality in distributed computation. In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October – 1 November 1989*, pages 364–369. IEEE Computer Society, 1989.
4. Nir Bacrach, Keren Censor-Hillel, Michal Dory, Yuval Efron, Dean Leitersdorf, and Ami Paz. Hardness of distributed optimization. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019.
5. Lélia Blin, Pierre Fraigniaud, and Boaz Patt-Shamir. On proof-labeling schemes versus silent self-stabilizing algorithms. In *Stabilization, Safety, and Security of Distributed Systems*, pages 18–32, 2014.
6. Keren Censor-Hillel, Ami Paz, and Mor Perry. Approximate proof-labeling schemes. *Theor. Comput. Sci.*, 811:112–124, 2020.
7. Yuval Emek and Yuval Gil. Twenty-two new approximate proof labeling schemes. In *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*, 2020.
8. Yuval Emek, Yuval Gil, and Shay Kutten. Locally restricted proof labeling schemes (full version), 2022. `arXiv:2208.08718`.
9. Laurent Feuilloley and Pierre Fraigniaud. Error-sensitive proof-labeling schemes. In *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, volume 91 of *LIPIcs*, pages 16:1–16:15, 2017.
10. Laurent Feuilloley, Pierre Fraigniaud, Juho Hirvonen, Ami Paz, and Mor Perry. Redundancy in distributed proofs. In *32nd International Symposium on Distributed Computing, DISC*, volume 121 of *LIPIcs*, pages 24:1–24:18, 2018.
11. Laurent Feuilloley, Pierre Fraigniaud, Pedro Montealegre, Ivan Rapaport, Éric Rémila, and Ioan Todinca. Compact distributed certification of planar graphs. *Algorithmica*, 83(7):2215–2244, 2021.
12. Pierre Fraigniaud, Amos Korman, and David Peleg. Local distributed decision. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS*, pages 708–717, 2011.
13. Mohsen Ghaffari, Fabian Kuhn, and Yannic Maus. On the complexity of local distributed graph problems. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 784–797. ACM, 2017.
14. Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998.
15. Shafi Goldwasser, Guy N. Rothblum, and Yael Tauman Kalai. Delegating computation: Interactive proofs for muggles. *Electron. Colloquium Comput. Complex.*, page 108, 2017.
16. Mika Göös and Jukka Suomela. Locally checkable proofs in distributed computing. *THEORY OF COMPUTING*, 12:1–33, 2016.
17. Gillat Kol, Rotem Oshman, and Raghuvansh R. Saxena. Interactive distributed proofs. In *PODC 2018 - Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, pages 255–264, July 2018.
18. Amos Korman and Shay Kutten. Distributed verification of minimum spanning trees. *Distributed Comput.*, 20(4):253–266, 2007.
19. Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. *Distributed Comput.*, 22(4):215–233, 2010.
20. E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 2006.

21 Nathan Linial. Distributive graph algorithms-global solutions from local data. In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 331–335. IEEE Computer Society, 1987.

22 Nathan Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992.

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

24 Boaz Patt-Shamir and Mor Perry. Proof-labeling schemes: Broadcast, unicast and in between. In *Stabilization, Safety, and Security of Distributed Systems - 19th International Symposium, SSS*, volume 10616, pages 1–17. Springer, 2017.

25 David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, USA, 2000.

26 Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. *SIAM J. Comput.*, 50(3), 2021.

27 Václav Rozhon and Mohsen Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 350–363. ACM, 2020.

A Bounds for Concrete OptDGPs and CGFs

A.1 OptDGPs

In this section, we show how the compiler presented in Section 4 can be used in the design of locally restricted APLSs for some classical OptDGPs that fit the canonical structure. In Sections A.1.1, A.1.2, and A.1.3, we present locally restricted APLSs with a logarithmic proof size for the problems of minimum weight vertex cover, maximum independent set, and minimum weight dominating set, respectively. Then, in Section A.1.4, we present a locally restricted $(1+\epsilon)$-APLS that applies to any canonical OptDGP.

A.1.1 Minimum Weight Vertex Cover

Consider a graph $G = (V, E)$ associated with a node-weight function $w : V \to \{1, \ldots, n^{O(1)}\}$ and let $C \subseteq E$ be a set of *constrained* edges. A *vertex cover* of C is a subset $U \subseteq V$ of nodes such that every edge $e \in C$ has at least one endpoint in U. A *minimum weight vertex cover (MWVC)* of C is a vertex cover U of C that minimizes $w(U) = \sum_{u \in U} w(u)$.

Observe that MWVC is a covering OptDGP. Moreover, vertex cover is self-induced (notice that this is in contrast to the common case of vertex cover where all edges are constrained). Thus, MWVC is canonical. We aim to use our compiler to construct a locally restricted $2(1+\epsilon)$-APLS for MWVC. To that end, we establish the following lemma.

▶ **Lemma A.1.** *There exists a 2-APLS for MWVC with a proof size of $O(\log n)$.*

Proof. As presented in [7], there exists a 2-APLS for the instance of MWVC where all edges are constrained and the graph is connected. We can obtain a 2-APLS for MWVC in the more general case where a subset $C \subseteq E$ of edges are constrained simply by applying the 2-APLS from [7] on the connected subgraphs induced by the constrained edge set C. The proof size of this scheme is $O(\log n + \log W)$, where W is an upper bound on the node-weights. Since in our case $W = n^{O(1)}$, it follows that the proof size is $O(\log n)$. ◀

Plugging the 2-APLS obtained in Lemma A.1 into our compiler leads to the following corollary.

▶ **Corollary A.2.** *For any constant $\epsilon > 0$, there exists a locally restricted $(2(1+\epsilon))$-APLS for MWVC with a proof size of $O(\log n)$.*

We now consider the unweighted version, simply referred to as *minimum vertex cover (MVC)*, on graphs with large odd-girth (where the odd-girth of a graph is defined to be the shortest odd cycle). As the following theorem shows, this case allows for an improved approximation ratio.

▶ **Theorem A.3.** *For any constant $\epsilon > 0$, there exists a locally restricted $(1+\epsilon)$-APLS for MVC on graphs of odd-girth $\omega(\log n)$ with a proof size of $O(\log n)$.*

Proof. Recall that our compiler first partitions the nodes into clusters of diameter $O(\log n)$, and then proceeds to apply an α-APLS concurrently on the subgraph induced by each cluster. We observe that each of these subgraphs created by the partition is bipartite since the odd-girth of the graph is $\omega(\log n)$. Thus, it is sufficient to show that there exists a PLS for MVC on bipartite graphs with a proof size of $O(\log n)$.

The well known König's theorem states that in bipartite graphs the size of minimum vertex cover is equal to the size of maximum matching. This allows for a PLS for MVC in bipartite graphs with a proof size of $O(\log n)$ constructed as follows. The prover simply encodes a maximum matching on the graph along with a proof that the size of this matching is equal to the size of the given vertex cover (e.g., by means of a comparison scheme). ◀

A.1.2 Maximum Independent Set

Consider a graph $G = (V, E)$ and let $C \subseteq E$ be a set of *constrained* edges. An *independent set* of C is a subset $I \subseteq V$ of nodes, such that each node $v \in I$ is incident on an edge in C and every edge $e \in C$ has at most one endpoint in I. A *maximum independent set (MaxIS)* of C is an independent set I of C that maximizes $|I|$.

Observe that MaxIS is a packing OptDGP. Moreover, independent set is self-induced (notice that this is in contrast to the common case of independent set where all edges are constrained). Thus, MaxIS is canonical.

Let us denote by $\Delta = \max_{v \in V}\{\deg(v)\}$ the largest degree in graph $G = (V, E)$. In the following lemma, we present a simple Δ-APLS for the MaxIS problem.

▶ **Lemma A.4.** *There exists a Δ-APLS for MaxIS with a proof size of $O(\log n)$.*

Proof. We use the fact that the ratio between the size of a maximum independent set and the size of a maximal independent set (i.e., an independent set that is not a subset of any other independent set) is at most Δ. The Δ-APLS construction is simple. The prover encodes a maximal independent set along with the value of Δ and a proof that its size is at most a multiplicative factor of Δ away from the given independent set. ◀

Plugging the Δ-APLS from Lemma A.4 into our compiler leads to the following corollary.

▶ **Corollary A.5.** *For any constant $\epsilon > 0$, there exists a locally restricted $(\Delta(1+\epsilon))$-APLS for MaxIS with a proof size of $O(\log n)$.*

Similarly to the MVC problem, restricting MaxIS to families of graphs with odd-girth $\omega(\log n)$ allows for a better approximation ratio, as established by the following theorem.

▶ **Theorem A.6.** *For any constant $\epsilon > 0$, there exists a locally restricted $(1+\epsilon)$-APLS for MaxIS on graphs of odd-girth $\omega(\log n)$ with a proof size of $O(\log n)$.*

Proof. Similarly to the proof of Theorem A.3, it is sufficient to show that there exists a PLS for MaxIS on bipartite graphs with a proof size of $O(\log n)$. To that end, we can use the fact that in bipartite graphs, the size of MaxIS is equal to the size of a minimum edge cover. We can now construct a PLS where the prover encodes a minimum edge cover of the graph, along with a proof that it is equal in size to the given independent set. ◀

A.1.3 Minimum Weight Dominating Set

Consider a graph $G = (V, E)$ associated with a node-weight function $w : V \to \{1, \ldots n^{O(1)}\}$ and let $C \subseteq V$ be a subset of *constrained* nodes. A *dominating set* of C is a subset $D \subseteq V$ of nodes, such that $D \cap (v \cup N(v)) \neq \emptyset$ for each constrained node $v \in C$. A *minimum weight dominating set (MWDS)* of C is a dominating set D of C that minimizes $\sum_{u \in D} w(u)$.

Observe that MWDS is a covering OptDGP. Moreover, dominating set is self-induced (notice that this is in contrast to the common case of dominating set where all nodes are constrained). Thus, we get that MWDS is canonical. We aim to use our compiler to construct a locally restricted $O(\log n)$-APLS for MWDS. To that end, we establish the following lemma.

▶ **Lemma A.7.** *There exists an $O(\log n)$-APLS for MWDS with a proof size of $O(\log n)$.*

Proof. An $O(\log n)$-APLS for the instance of MWDS where all nodes are constrained and is presented in [7]. The idea behind that $O(\log n)$-APLS is that the prover provides a feasible solution to the dual LP, such that the objective value of this dual solution is at most a multiplicative factor of $O(\log n)$ from the given dominating set. We argue that this technique can be applied to obtain $O(\log n)$-APLS for MWDS (in its more generalized version described above). This follows from the fact that the gap between an optimal MWDS solution and an optimal dual solution remains $O(\log n)$ (since MWDS is an instance of set cover). ◀

Plugging the $O(\log n)$-APLS obtained in Lemma A.7 into our compiler leads to the following corollary.

▶ **Corollary A.8.** *There exists a locally restricted $O(\log n)$-APLS for MWDS with a proof size of $O(\log n)$.*

A.1.4 Generic Locally Restricted $(1 + \epsilon)$-APLS for Canonical OptDGPs

We establish a generic upper bound that applies to any canonical OptDGP.

▶ **Theorem A.9.** *Consider a canonical OptDGP Ψ. For any constant $\epsilon > 0$, there exists a locally restricted $(1 + \epsilon)$-APLS for Ψ with a proof size of $O(n^2)$.*

Proof. As stated in [19, Theorem 3.2], any decidable property admits a PLS. The idea behind this universal PLS is that the prover can assign each node v with a label $L(v)$ that encodes the entire configured graph G_s. In response, the verifier at node v verifies that v's neighbors agree on the structure of the configured graph encoded in $L(v)$, and that v's local neighborhood is consistent with the one encoded in $L(v)$. Following that, the verifier can evaluate whether G_s is a yes-instance or not.

Observe that applying the universal PLS described above for a canonical OptDGP requires a proof size of $O(n^2)$. We can now plug this PLS construction into our compiler to obtain the desired locally restricted $(1 + \epsilon)$-APLS for Ψ. ◀

A.2 CGFs

In this section, we show how the compiler presented in Section 5 can be combined with known PLS constructions to obtain locally restricted δ-TPLSs for various well-known SU-closed CGFs.

A.2.1 Planarity

A graph $G = (V, E)$ is called *planar* if it can be embedded in the plane. The following lemma has been established by Feuilloley et al. [11].

▶ **Lemma A.10.** *There exists a PLS for planarity with a proof size of $O(\log n)$.*

Observe that planar graphs are SU-closed. Thus, plugging the PLS for planarity into our compiler implies the following corollary.

▶ **Corollary A.11.** *For any constant $\delta > 0$, there exists a locally restricted δ-TPLS for planarity with a proof size of $O(\log n)$.*

A.2.2 Bounded Arboricity

The *arboricity* of a graph $G = (V, E)$ is the minimum number k for which there exists an edge partition $E = E_1 \dot\cup \ldots \dot\cup E_k$ such that $G_i = (V, E_i)$ is a forest for each $i \in [k]$. Let $arb(G)$ denote the arboricity of graph G. We say that graph G is of bounded arboricity if $arb(G) = O(1)$.

▶ **Lemma A.12.** *There exists a PLS for bounded arboricity with a proof size of $O(\log n)$.*

Proof. As established in [19], there exists a PLS for forests with a proof size of $O(\log n)$. A PLS for bounded can be implemented by using the PLS construction for forests concurrently on $arb(G)$ edge-induced subgraphs of G. ◀

Observe that bounded arboricity is SU-closed. Thus, plugging the PLS for bounded arboricity into our compiler implies the following corollary.

▶ **Corollary A.13.** *For any constant $\delta > 0$, there exists a locally restricted δ-TPLS for bounded arboricity with a proof size of $O(\log n)$.*

A.2.3 k-Colorability

For a positive integer k, we say that a graph $G = (V, E)$ is *k-colorable* if there exists a proper k-coloring of its nodes. Observe that k-colorability admits a (simple) PLS with proof size $O(\log k)$ and that it is SU-closed. Thus, combined with our compiler, we get the following theorem.

▶ **Theorem A.14.** *For any constant $\delta > 0$, there exists a locally restricted δ-TPLS for k-colorability with a proof size of $O(\log n)$.*

A.2.4 Forests and DAGs

The following two lemmas have been established by Korman et al. [19].

▶ **Lemma A.15.** *There exists a PLS for forests with a proof size of $O(\log n)$.*

▶ **Lemma A.16.** *There exists a PLS for directed acyclic graphs (DAGs) with a proof size of* $O(\log n)$.

Observe that both forests and DAGs are SU-closed. Thus, plugging the PLSs for forests and DAGs into our compiler implies the following corollaries.

▶ **Corollary A.17.** *For any constant* $\delta > 0$*, there exists a locally restricted* δ*-TPLS for forests with a proof size of* $O(\log n)$.

▶ **Corollary A.18.** *For any constant* $\delta > 0$*, there exists a locally restricted* δ*-TPLS for directed acyclic graphs with a proof size of* $O(\log n)$.

B Impossibilities of Locally Restricted GPLS

In this section, we establish some inherent limitations of locally restricted GPLSs based on the following observation.

▶ **Observation B.1.** *If there exists a locally restricted GPLS over* $\mathcal{U}$ *with yes-family* $\mathcal{F}_Y$ *and no-family* $\mathcal{F}_N$*, then there exists a LOCAL algorithm with a* $\log^{O(1)}(n)$ *round-complexity that given a configured graph* $G_s \in \mathcal{U}$*, decides if* $G_s \in \mathcal{F}_Y$ *(in which case all nodes return* `True`*); or* $G_s \in \mathcal{F}_N$ *(in which case at least one node returns* `False`*).*

Proof. Given a configured graph $G_s \in \mathcal{U}$, we obtain a LOCAL algorithm by first simulating the locally restricted prover on G_s (using a polylogarithmic number of rounds), and then simulating the verifier (using 1 round). By the correctness requirements of a GPLS, the outcome of this algorithm is that all nodes return `True` if $G_s \in \mathcal{F}_Y$; whereas at least one node returns `False` if $G_s \in \mathcal{F}_N$. ◀

The observation above implies that it is impossible to construct a locally restricted GPLS for verification tasks that require $\omega(poly \log n)$ rounds in the LOCAL model. Notice that this impossibility applies to a large class of verification tasks associated with OptDGPs and CGFs. For example, using a simple indistinguishability argument, one can show that there is no locally restricted PLS for forests (i.e., a PLS deciding if a given graph is a forest). Similar arguments can be applied to exclude a locally restricted PLS for most of the OptDGPs and CGFs considered in Section A.

C Additional Related Work

The PLS model was introduced by Korman, Kutten, and Peleg in [19] and studied extensively since then, see, e.g., [11, 18, 5, 9, 24, 10]. Research in this field include [18], where a PLS for minimum spanning tree is shown to have a proof size of $O(\log n \log W)$, where W is the largest edge-weight, and [11], where a PLS for planarity is shown to have a proof size of $O(\log n)$.

In parallel, several researchers explored the limitations of the PLS model, often relying on known lower bounds from nondeterministic communication complexity [20]. Lower bounds of $\Omega(n^2)$ and $\Omega(n^2/\log n)$ are established in [16] with regards to the proof size of any PLS for graph symmetry and non 3-colorability, respectively. A similar technique was used by the authors of [4] to show that many classic optimization problems require a proof size of $\tilde{\Omega}(n^2)$.

The lower bounds on the proof size of PLSs for some optimization problems have motivated the authors of [6] to introduce the APLS notion, further studied recently in [7]. Optimization problems considered in the context of APLS include maximum weight matching, which was

shown in [6] to admit a 2-APLS with a proof size of $O(\log W)$, and minimum weight vertex cover, which was shown in [7] to admit a 2-APLS with a proof size of $O(\log n + \log W)$, where in both cases W refers to the largest weight value.

In the current paper, we also introduce the TPLS model which is suited for properties that are not formulated as optimization problems. This model is based on the notion of property testing [14]. More specifically, the TPLS model is formulated using the distance measure between graphs defined in [1].

Our focus in this paper is on locally restricted APLSs and TPLSs, restricting the prover to a LOCAL algorithm with a polylogarithmic number of rounds. Interest in the power of deterministic LOCAL algorithms with polylogarithmic round-complexity was initiated by Linial's seminal work [21, 22]. One particular problem that raised a lot of interest in this context is the network decomposition problem introduced by Awerbuch et al. in [3]. In a recent breakthrough [27], Ghaffari and Rozhon presented a deterministic algorithm with polylogarithmic round-complexity for the network decomposition problem. As established in [13], a consequence of this result is that any SLOCAL algorithm with $\log^{O(1)} n$ locality can be simulated by a LOCAL algorithm with $\log^{O(1)} n$ rounds. This simulation technique is used in the construction of our compilers in Sections 4 and 5.