Explicit Space-Time Tradeoffs for Proof Labeling Schemes in Graphs with Small Separators

Orr Fischer
Tel-Aviv University, Israel

Rotem Oshman
Tel-Aviv University, Israel

Dana Shamir
Tel-Aviv University, Israel

Abstract

In distributed verification, our goal is to verify that the network configuration satisfies some desired property, using pre-computed information stored at each network node. This is formally modeled as a *proof labeling scheme* (PLS): a prover assigns to each node a *certificate*, and then the nodes exchange their certificates with their neighbors and decide whether to accept or reject the configuration. Subsequent work has shown that in some specific cases, allowing more rounds of communication – so that nodes can communicate further across the network – can yield shorter certificates, trading off the *space* required to store the certificate against the *time* required for verification. Such tradeoffs were previously known for trees, cycles, and grids, or for proof labeling schemes where all nodes receive the same certificate.

In this work we show that in large classes of graphs, every one-round PLS can be transformed into a multi-round PLS with shorter certificates. We give two constructions: given a 1-round PLS with certificates of ℓ bits, in graphs families with balanced edge separators of size $s(n)$, we construct a t-round PLS with certificates of size $\tilde{O}(\ell \cdot s(n)/t)$, and in graph families with an excluded minor and maximum degree Δ, we construct a t-round PLS with certificates of size $\tilde{O}(\ell \cdot \Delta/\sqrt{t})$. Our constructions are explicit, and we use erasure codes to exploit the larger neighborhood viewed by each node in a t-round PLS.

2012 ACM Subject Classification Networks; Theory of computation → Distributed algorithms

Keywords and phrases proof-labeling schemes, space-time tradeoffs, families with excluded minor

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2021.21

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

1 Introduction

Distributed proofs are a mechanism that allows a distributed network to verify that its current configuration is legal: each node is equipped with a pre-computed *certificate*, which serves as part of a global proof that the network configuration is legal. To verify the proof, nodes examine the certificates in their neighborhoods and decide whether to *accept* or *reject* the proof.

Most of the literature on distributed proofs assumes that in the verification phase, each node can only view the certificates and the network structure in some constant-sized neighborhood around itself: for example, in *proof labeling schemes* [16], nodes only see the certificates of their immediate neighbors, and in *locally-checkable proofs* [10], nodes can see their r-neighborhood for some constant r. However, under this restriction it is known that some graph predicates require very large certificates, up to $\Omega(n^2)$ bits per node [10]. This gives rise to the following question, proposed by [18, 6]: can we trade *time* for *space*? In other words, if we allow the verification phase of the proof to run for more rounds and collect information about a larger neighborhood, can we decrease the certificate size?

A distributed proof where the verifier runs in t rounds is called a *t-proof-labeling scheme*, or *t-PLS* for short [18]. Our goal when proving space-time tradeoffs is to show that for some problem or class of problems, as t grows, we can construct a t-PLS with smaller certificates. Several such tradeoffs are shown in [18, 6]: for example, in [18] it is shown that every graph property on n-vertex graphs can be verified using certificates of size $O(n^2/t)$, and in [6] it is shown that in trees, cycles and grids, we can save a factor of $O(t)$ in the certificate size of *any* proof-labeling scheme. However, the general case remains open [6]: is it true that for any network predicate we might wish to verify, the size of the certificate scales down as t grows? In the current paper we take a step in this direction, and show that for a large class of graphs the answer is positive.

Our main contributions are the following:

- We show that *erasure codes* are a useful building block for constructing t-PLS, yielding simple, explicit constructions with improved parameters.
- We show that in two large classes of graphs – namely, any graph family that excludes a fixed minor, and any graph family that admits a small balanced edge separator[1] – we can transform a 1-PLS into a t-PLS with reduced certificate size, depending on the exact parameters of the graph. These families include, for example, planar graphs (and more generally bounded-genus graphs), minor-closed graph famlies, bounded-treewidth graphs, and others.

  Our proof for graph families with an excluded minor generalizes to any graph family with *polynomial expansion*, but for lack of space, we defer the more general case to the full version of the paper.

Using erasure codes to construct t-proof-labeling schemes. Suppose we are given a 1-PLS Π, and wish to construct from it a t-PLS Π' with shorter certificates. A natural approach, used in [18, 6], is to take the certificate a_v computed under Π for each node v, "chop it" into pieces $a_v^1, \ldots, a_v^k$ of length $|a_v|/k$ each (where k is a parameter of the construction), and distribute the pieces to k nodes in v's t-neighborhood. In the t-round verifier of Π', node v collects the pieces, reconstructs its certificate a_v, and uses the original verifier of Π to decide whether to accept or reject.

The main difficulty when designing a t-PLS of this type is to decide, for each node v, which nodes in v's vicinity should receive which pieces of v's certificate, so that no single node u is given too many certificate pieces to store. Several such construction are given in [18, 6], each tailored for a specific problem or class of graphs, and one construction in [6] is non-explicit (i.e., the probabilistic method is used). Erasure codes offer a simple mechanism for simplifying this process: an erasure code encodes a string w into pieces $c_1, \ldots, c_m$, such that from any sufficiently large subset $c_{i_1}, \ldots, c_{i_d}$ of pieces, we can reconstruct the original string w. This allows us not to worry about *which* nodes receive *which* pieces of a given certificate: any subset of sufficiently many pieces allows us to reconstruct the certificate in full. In Section 4 we show that using erasure codes, we can simplify, unify and slightly improve two constructions from [18, 6], which transform a 1-PLS where all nodes receive the same certificate (a *uniform PLS* [6]) into a t-PLS with shorter certificates.

Using codes has other advantages as well: for example, if we use error-correcting codes (of which erasure codes are a special case) to encode the certificates of the nodes, we can gain some degree of resilience against *benign data corruption*, a scenario where the network configuration is still legal but the certificates of some nodes have become corrupted. In

[1] We formally define the notion of balanced edge separators later in the paper, but informally, a graph is said to have small balanced edge separators if there is a small set of edges, whose removal partitions the graph into connected components that are not too large.

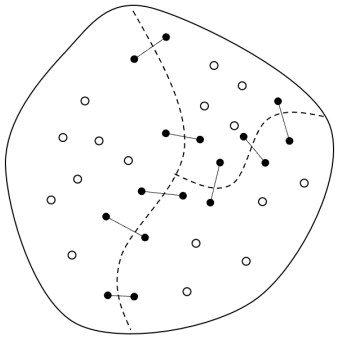

Figure 1 A partition of the graph, with border nodes indicated in black.

Section 4 we define the notion of *resilient t-proof-labeling schemes*, and show that by compiling a 1-PLS into a t-PLS using an error-correcting code, we can withstand up to $O(b(t))$ corrupted certificates in each t-neighborhood, assuming the t-neighborhood is of size at least $b(t)$.

Partition-based t-proof-labeling schemes. In [18, 6] it is shown that in trees, cycles and grids, we can transform any 1-PLS with ℓ-bit certificates into a t-PLS with certificates of size $O(\ell/t)$. The idea underlying the three constructions (for trees, cycles and grids) is similar, and we refer to it as a *partition-based t-PLS*: we partition the graph into units $U_1, \ldots, U_k$, such that each unit

- Has diameter $O(t)$,
- Contains at least r nodes, where r is a parameter of the construction, and
- Contains few *border nodes* – nodes that have neighbors outside the unit (see Fig. 1).

In the t-PLS, we take the 1-PLS certificate a_v of each border node v, and split a_v into smaller pieces $a_v^1, \ldots, a_v^s$, where $s = \Theta(t)$.[2] We distribute the pieces across the nodes of v's unit,[3] in such a way that each node receives $O(1)$ certificate pieces in total (from all the border nodes). The certificates of non-border nodes are simply ignored. In the resulting t-PLS, the certificate size is reduced by a factor of $\Theta(t)$.

To verify the proof, each border node v first reconstructs its original certificate a_v by collecting the s certificate-parts from its unit. Next, in each unit U_i, we check whether there is an assignment A_i of certificates to the non-border nodes of U_i which would cause all nodes of the unit to accept (including the border nodes, using the certificates they reconstructed for themselves); if so, the nodes of the unit accept, and otherwise they reject. The soundness of this scheme follows from the fact that since the certificates of the border nodes are fixed, we can "stitch together" the assignments $A_1, \ldots, A_k$ that we found for the units $U_1, \ldots, U_k$ into a global certificate assignment that would be accepted by the original 1-PLS.

In Section 5 we define a general scheme for transforming a 1-PLS into a t-PLS, which codifies the strategy outlined above, and which we then instantiate for graph families that have small separators.

[2] We describe here the parameters used in the constructions of [6]; our constructions use different values.
[3] In the case of trees, the construction from [6] may distribute the pieces across the nodes of an adjacent unit, not necessarily v's unit.

Constructing t-proof-labeling schemes for graphs with small separators. The constructions given in [6] for trees, cycles and grids exploit the fact that graphs of these classes can be partitioned into vertex-disjoint units such that if a unit U_i contains b border nodes, then the unit has at least $\Omega(b \cdot t)$ nodes in total, allowing us to distribute pieces of the certificates of the border nodes across the unit efficiently. What other classes of graphs can be partitioned into units with large "area" and a small "perimeter"? It is natural to consider graphs with *bounded expansion*, such as planar graphs, or graphs with bounded treewidth. We show that indeed, there are two such classes of graphs for which we can construct a t-PLS with shorter certificates compared to a 1-PLS. In Section 7, we handle graph families that admit a *small balanced edge separator* (see Section 7 for the definition), and prove:

▶ **Theorem 1.** *Fix a subgraph-closed family $\mathcal{G}$ and a function $s : \mathbb{N} \to \mathbb{N}$, such that any graph $G \in \mathcal{G}$ has a balanced edge separator of size $s(|V(G)|)$. Let Π be a 1-PLS recognizing some predicate $\mathcal{P}$. Then for every $t \geq 2$, there exists a t-PLS Π' for $\mathcal{P}$, such that for every configuration $(G, I) \in \mathcal{P}$ of diameter at least t,*

$$\mathrm{cost}(\Pi', (G, I)) = \tilde{O}\left(\frac{\mathrm{cost}(\Pi, (G, I)) \cdot s(n)}{t}\right).$$

Here, $\mathrm{cost}(\Pi, (G, I))$ denotes the length of the certificate assigned by the PLS Π to the nodes of the graph G, when the input to each node is given by $I : V(G) \to \mathcal{I}$ (see Section 3 for the formal definitions).

Graphs that have small balanced edge separators include low-degree graphs with small treewidth; for example, outerplanar graphs have treewidth 2, and are known to admit a balanced edge separator of size at most $O(\Delta)$, where Δ is the maximum degree.

In Section 8 we handle graph families that excludes some constant-sized minor, and show:

▶ **Theorem 2.** *Fix a family $\mathcal{G}$ of graphs that exclude some fixed minor H. Let Π be a 1-PLS recognizing some predicate $\mathcal{P}$. Then for every $t \geq 2$, there exists a t-PLS Π' recognizing $\mathcal{P}$, such that for every configuration $(G, I) \in \mathcal{P}$ of diameter at least t and maximum degree Δ,*

$$\mathrm{cost}(\Pi', (G, I)) = \tilde{O}\left(\frac{\mathrm{cost}(\Pi, (G, I)) \cdot \Delta}{\sqrt{t}}\right).$$

Many interesting graph families can be described by a set of forbidden (excluded) minors; perhaps the most famous example is *planar graphs*, and more generally, graphs of fixed genus.

Strictly speaking, the two constructions above could be implemented without using erasure codes, but using erasure codes simplifies them.

We conjecture that the connection between the existence of an efficient t-PLS and bounded expansion goes both ways, namely, in graphs that are *good* expanders,[4] a t-PLS cannot always have certificate size significantly smaller than a 1-PLS for the same predicate.

2 Related Work

Distributed proofs, proof-labeling schemes have been extensively studied under various modeling assumptions (e.g. [2, 4, 5, 12, 13, 15, 14, 16, 19, 21, 7, 11, 17, 10]). In the current section we discuss only prior work that is directly relevant to the current paper; we refer to the surveys of Feuilloley and Fraigniaud [3] and of Suomela [22] for a more complete overview of the field.

[4] An *expander* is a graph where every sufficiently-small set S of vertices has a *large boundary* – e.g., S has many outgoing edges, or many adjacent nodes, depending on the precise definition used.

While almost all work on proof-labeling schemes assumes that the verifier runs for a single round, Göös et al. [10] considered a model where the verifier may run for r rounds, where r is a constant. Among other results, they showed a *universal* 1-*PLS* which uses $O(\min(n^2, m \log n))$-bit certificates by giving each node an encoding of the entire graph; this can be used to locally check any property $\mathcal{P}$. On the other hand, they showed that there are properties that require a certificate size of $\Omega(n^2)$ bits to be checked locally, such as the existence of a non-trivial automorphism.

Ostrovsky et al. [18] introduced the notion of a t-PLS, in which the verification procedure is allowed to use t rounds of communication instead of only one. They analyze the tradeoffs between time (number of rounds), certificate size, and total communication, and show that the certificate size of a universal t-PLS is $O(\min(n^2, m \log n)/t)$. In the universal t-PLS of [18], the graph is divided into blocks, and the representation of the graph is distributed the between the nodes in each block, such that each node can collect the entire graph representation from its t-neighborhood. They also construct an optimal t-PLS that determines whether a graph is acyclic using $O(\log n/t)$-bit certificates, and prove a matching lower bound.

Feuilloley et al. [6] gives additional general tradeoffs for t-proof-labeling schemes. First, they show near-linear scaling of the certificate size for any proof labeling scheme in trees, cycles and grids (as outlined in Section 1). In the current paper we use a similar but more general approach in larger classes of graphs. Feuilloley et al. [6] also show that every *uniform* 1-PLS (a PLS in which the prover gives the same certificate to all the nodes) can be transformed into a t-PLS with certificates smaller by a factor of $O(\log(n)/b(t))$; here, $b(t)$ is the minimum number of nodes in a t-neighborhood of a node in the graph. Their construction uses the probabilistic method: they show that if each node stores each bit of the original certificate with probability roughly $\log(n)/b(t)$, then there is non-zero probability that each node will have all bits of the certificate in its t-neighborhood, which allows all nodes to recover the original certificate. In the current paper we use erasure codes to simplify the scheme, eliminate the multiplicative factor of $\log(n)$, and obtain an explicit construction. Finally, [6, 18] also construct t-PLS for specific problems (e.g., shortest paths and cycle-freeness).

3 Preliminaries

Graph notation. Given a graph G, we let $V(G)$ and $E(G)$ denote the vertex set and edge set of G, respectively. We let $H \subset G$ denote the fact that H is a subgraph of G (that is, $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$). We sometimes abuse notation by writing $v \in G$ instead of $v \in V(G)$.

The neighborhood of a node $v \in V(G)$ is denoted $N(v)$. The t-neighborhood of a node $v \in V(G)$, i.e., the set of all the nodes with distance at most t from v, is denoted $B_t(v)$ ("the ball of size t around v"). We say that a graph G has growth $b : \mathbb{N} \to \mathbb{N}$ if for every $v \in V$ and $t \geq 1$, $|B_t(v)| \geq b(t)$.

An $\mathcal{I}$-*configuration* (or *configuration* for short) is a pair (G, I), where $G = (V, E)$ is a graph, and $I : V \to \mathcal{I}$ is an assignment of inputs to the nodes of G. Given a family $\mathcal{G}$ of graphs, we denote by $\mathcal{G}_\mathcal{I}$ the set of all $\mathcal{I}$-configurations (G, I) where $G \in \mathcal{G}$ and $I : V \to \mathcal{I}$.

Proof labeling schemes. Let $\mathcal{G}_\mathcal{I}$ be a family of configurations, let $\mathcal{P} \subseteq \mathcal{G}_\mathcal{I}$ be a predicate, and let $t > 1$. A *t-proof labeling scheme (t-PLS) for $\mathcal{P}$* is a pair $\Pi = (\mathbf{Prv}, \mathbf{Ver})$, where

- $\mathbf{Prv}$, the *prover*, is a mapping that takes a configuration $(G, I) \in \mathcal{P}$ and produces a certificate assignment $\mathbf{Prv}(G, I) = \{a_v\}_{v \in V(G)}$ for the nodes of G, where $a_v \in \{0, 1\}^*$ for each $v \in V(G)$.

- **Ver**, the *verifier*, is a t-round deterministic[5] distributed algorithm that takes as input the certificate $a_v \in \{0,1\}^*$ at each node v, and outputs a Boolean value. We let $\mathbf{Ver}(G, I, a, v)$ denote the output of the algorithm at node $v \in V(G)$, when executed in configuration (G, I), with certificates $a = \{a_v\}_{v \in V(G)}$.

We require:

- Soundness: for every configuration $(G, I) \in \mathcal{G}_{\mathcal{I}}$ and certificate assignment $a = \{a_v\}_{v \in V(G)}$, if $\mathbf{Ver}(G, I, a, v) = 1$ for all $v \in V(G)$, then $(G, I) \in \mathcal{P}$.
- Completeness: for every configuration $(G, I) \in \mathcal{P}$ and for every node $v \in V(G)$ we have $\mathbf{Ver}(G, I, \mathbf{Prv}(G, I), v) = 1$.

We note that the verifier may send arbitrarily large messages (i.e., the verifier is a t-round LOCAL algorithm).

A *1-PLS* is a restricted type of proof labeling scheme $\Pi = (\mathbf{Prv}, \mathbf{Ver})$, following the original definition from [16]: the prover is the same as in the case of t-PLS for $t > 1$ above, but the verifier is restricted to a single round, where each node v sends its certificate a_v to all of its neighbors. In the case of a 1-PLS, we let $\mathbf{Ver}(v, I(v), N(v), a_v, \{a_u\}_{u \in N(v)})$ denote the output of the verifier at node v, when node v's input is $I(v)$, its neighborhood is $N(v)$, its certificate is a_v, and its neighbors' certificates are $\{a_u\}_{u \in N(v)}$.

We say a t-PLS is *uniform* if it gives the same certificates to all nodes in the graph, i.e., for every $(G, I) \in \mathcal{G}_{\mathcal{I}}$ and $v, u \in V(G)$, $\mathbf{Prv}(G, I)(v) = \mathbf{Prv}(G, I)(u)$.

Given a t-PLS $\Pi = (\mathbf{Prv}, \mathbf{Ver})$, the *cost* of Π in a configuration (G, I), denoted $\mathrm{cost}(\Pi, (G, I))$, is the length of the longest certificate in $\mathbf{Prv}(G, I)$. If Π is a t-PLS for a predicate $\mathcal{P}$, we define the cost of Π in a family of configurations $\mathcal{G}_{\mathcal{I}}$ as follows:

$$\mathrm{cost}(\Pi, \mathcal{P}, n) = \max\{\mathrm{cost}(\Pi, (G, I)) : (G, I) \in \mathcal{P}, |V(G)| = n\}$$

Note that the cost is based only on configurations that satisfy $\mathcal{P}$, as the prover has no particular obligation otherwise.

Initial knowledge. Our techniques require the nodes to have unique identifiers, but generally the nodes do not need to know in advance the size of the graph or the value of any other graph parameter; accordingly, the t-PLS we construct in Section 4 does not assume that nodes know the size of the graph. However, to simplify the presentation of the remainder of our results, following Section 4 we do assume that the size n of the graph is known to all the nodes. This is not essential, but it avoids some technical details.

Erasure codes. An erasure code is a type of code that allows us to recover from the erasure of some symbols in the codeword:

▶ **Definition 3.** *Given a prime number $q \in \mathbb{N}$ and parameters $n, k, d \in \mathbb{N}$, an* $(n, k, d)_q$-erasure code *is a pair (Enc, Dec), where*

- $Enc : \mathbb{F}_q^k \to \mathbb{F}_q^n$ *is an* encoder*,*
- $Dec : (\mathbb{F}_q \cup \{?\})^n \to \mathbb{F}_q^k$ *is a* decoder*,*
- *For each $w \in \mathbb{F}_q^k$, if $c \in (\mathbb{F}_q \cup \{?\})^n$ is obtained from $Enc(w)$ by replacing fewer than d symbols by '?', then $Dec(c) = w$.*

[5] It is interesting to explore the case where the verifier is *randomized*, as this is known to help in some contexts [8], but in the current paper the verifiers that we construct are deterministic.

Note that n is used for both the size of the graphs we are working with and the length of the code, and this suits our purposes, because whenever we apply an erasure code, the length of the code will be the number of nodes in the graph: for convenience, we always produce one piece for each node in the graph, even though not all the pieces are used in every construction.

The codes that we use in the current paper are $(n, k, n-k+1)_q$-erasure codes, which are optimal in the number of erasures that they can tolerate. For such a code to exist, it suffices to have $k \leq n \leq q$ [20]. The celebrated *Reed-Solomon code* is an example of such a code.

Definition 3 assumes that the decoder is given a string of length n where at most d symbols have been *erased*, i.e., replaced by '?'. For our purposes here, it is more convenient to think of the decoder as a function that takes a set of at most n *pieces*, which are numbered symbols of the form $(i, c) \in [n] \times \mathbb{F}_q$, and reconstructs a word $w \in \mathbb{F}_q^k$. Formally, given a set $S = \{(i_1, c_1), \ldots, (i_{n'}, c_{n'})\}$ such that $n' \leq n$ and $i_j \neq i_{j'}$ for every $j \neq j'$, we abuse notation by writing $Dec(S)$ to represent the output of the following procedure: let $c' \in (\mathbb{F}_q \cup \{?\})^n$ be the string where

$$c'_j = \begin{cases} c_j & \text{if } (j, c_j) \in S, \\ ? & \text{otherwise.} \end{cases}$$

Then we return $Dec(c')$. (Note that this is well-defined, because we assumed that there is at most one value c_j such that $(j, c_j) \in S$ for each j.)

From binary certificates to codewords over $\mathbb{F}_q$. In the current paper we use erasure codes to encode certificates that are represented as binary strings. To do so, we view the certificate as a string over some finite field $\mathbb{F}_q$, where q is a sufficiently large prime number. The size q of the field we must take depends on the total number n of pieces, the length ℓ of the binary certificate, and the number $m \leq n$ of pieces that suffice to reconstruct the certificate: given $n, \ell, m \in \mathbb{N}$, let

$$q(n, \ell, m) = \max\left(n, 2^{\lceil \ell/m \rceil}\right). \tag{1}$$

Let $\ell' = \lceil \ell / \log q_{n,\ell,m} \rceil$ be the number of $\mathbb{F}_{q_{n,\ell,m}}$-elements required to represent an ℓ-bit string. By choice of $q_{n,\ell,m}$ we have $\ell' \leq m \leq n \leq q_{n,\ell,m}$.

Throughout the paper, we fix a family $\{C_{n,\ell,m}\}_{n,\ell,m \in \mathbb{N}}$ of erasure codes, where each $C_{n,\ell,m} = (Enc_{n,\ell,m}, Dec_{n,\ell,m})$ is an $(n, \ell', n-\ell'+1)_{q_{n,\ell,m}}$-erasure code. (Such a code exists, because $\ell' \leq m \leq n \leq q_{n,\ell,m}$.) To encode a certificate $a \in \{0,1\}^\ell$, we view a as an ℓ'-symbol string over $\mathbb{F}_{q_{n,\ell,m}}$, and apply the encoder $Enc_{n,\ell,m}$. To reconstruct the certificate from m or more pieces, we apply the decoder $Dec_{n,\ell,m}$ to obtain an ℓ'-symbol string over $\mathbb{F}_{q_{n,\ell,m}}$, which we then view as an ℓ-bit binary string.

When using the code $C_{n,\ell,m}$, the length (in bits) of each piece that we produce is

$$\log q(n, \ell, m) = O((\ell/m) + \log n), \tag{2}$$

matching the intuition that m pieces of size roughly ℓ/m are necessary and sufficient to reconstruct an ℓ-bit string.

4 Space-Time Tradeoff for Uniform Proof Labeling Schemes

Recall that a *uniform* proof labeling scheme is one where the prover assigns the same label to all nodes. To illustrate the usefulness of erasure codes, in this we use them to transform a uniform 1-PLS Π for a graph family with growth $b : \mathbb{N} \to \mathbb{N}$, into a t-PLS for the same

predicate using certificates that are smaller by a factor of roughly $b(t-1)$. This is a simpler, tighter and explicit proof for a similar claim from [6],[6] and it also generalizes the universal t-PLS from [18].

In this section we do not assume that the nodes of the graph know the size n of the graph or the growth function b. (Since the size of the certificates does depend on these parameters, we assume that certificates are encoded in some predetermined variable-length encoding.) Instead, we ask the prover to provide n and b to each node; the prover may lie, but we can show that this does not affect the soundness of our construction.

▶ **Theorem 4.** *Let $\mathcal{G}_\mathcal{I}$ be a family of configurations, $\mathcal{P} \subseteq \mathcal{G}_\mathcal{I}$ be a predicate and Π be a uniform 1-PLS for $\mathcal{P}$. Then for every $t > 1$, there exists a t-PLS Π' for $\mathcal{P}$, such that for every configuration $(G, I) \in \mathcal{P}$ with n nodes and growth $b : \mathbb{N} \to \mathbb{N}$,*

$$\mathrm{cost}(\Pi', (G, I)) = O\left(\frac{\mathrm{cost}(\Pi, \mathcal{P}, n)}{b(t-1)} + \log n\right).$$

Proof. Let $\Pi = (\mathbf{Prv}, \mathbf{Ver})$ be the uniform 1-PLS and fix $t > 1$. Let $k(n) = \mathrm{cost}(\Pi, \mathcal{P}, n)$. We define the t-PLS $\Pi' = (\mathbf{Prv}', \mathbf{Ver}')$ as follows.

Given an n-node configuration $(G, I) \in \mathcal{P}$ with growth factor $b : \mathbb{N} \to \mathbb{N}$, let $a \in \{0,1\}^{k(n)}$ be the certificate assigned by $\mathbf{Prv}$ to the nodes $\{v_1, \ldots, v_n\}$ of G. For convenience we denote $k = k(n), b = b(t-1)$ and $q = q(n, k, b)$. We use the erasure code $C_{n,k,b} = (Enc_{n,k,b}, Dec_{n,k,b})$ to generate the encoding $Enc_{n,k,b}(a) = (c_1, \ldots, c_n) \in \mathbb{F}_q^n$.

The new prover $\mathbf{Prv}'$ assigns each node v of G the certificate $a'_v = (i, c_i, b, n)$. The length of the certificate is $O(k/b + \log n)$: encoding the index $i \in [n]$, the ball size $b = b(t-1) \leq n$, and the graph size n requires $O(\log n)$ bits. The piece $c_i \in \mathbb{F}_q$ is of length $O(k/b + \log n)$, by (2).

Next we describe the verifier $\mathbf{Ver}'$. Each node v uses t rounds of communication to learn the entire ball $B_t(v)$ of radius t around itself, including all certificates of the nodes in $B_t(v)$. Node v verifies that:

- All certificates in the ball are well-formed 4-tuples.
- All certificates agree on the last two values (the claimed values of b and n), and
- For any two certificates $(i_1, c_{i_1}, b_1, n_1), (i_2, c_{i_2}, b_2, n_2)$ collected, we have $i_1 \neq i_2$.

In the sequel, we denote by $s_x = (i_x, c_{i_x}, \tilde{b}, \tilde{n})$ the certificate of node $x \in B_t(v)$. As we said above, the values $\tilde{b}, \tilde{n}$ are the same across all certificates in $B_t(v)$, otherwise v rejects. Also let $\tilde{k} = k(\tilde{n})$ and $\tilde{q} = q(\tilde{n}, \tilde{k}, \tilde{b})$.

For each $u \in N(v) \cup \{v\}$, let $S_u = \{(i_x, c_{i_x}) : x \in B_{t-1}(u)\}$ denote the pieces in the $(t-1)$-ball around node u. Node v verifies that $|S_u| \geq \tilde{b}$, $i_x \in [\tilde{n}]$, and c_{i_x} is an element of $\mathbb{F}_{\tilde{q}}$. Next, for each $u \in N(v) \cup \{v\}$, node v uses the decoder $Dec_{\tilde{n},\tilde{k},\tilde{b}}$ to reconstruct from S_u a certificate $a_u = Dec_{\tilde{n},\tilde{k},\tilde{b}}(S_u) \in \{0,1\}^{\tilde{k}}$.

Finally, node v verifies that: $a_u = a_v$ for every $u \in N(v)$, and that the original verifier $\mathbf{Ver}(v, I(v), N(v), a_v, \{a_u\}_{u \in N(v)})$ accepts. If so, node v accepts, and otherwise it rejects.

For lack of space, we omit the detailed proof that Π' is sound and complete here. Completeness is straightforward; soundness is proven by showing that if all nodes accept, then all have reconstructed the same certificate $a \in \{0,1\}^k$ for themselves and their neighbors, and the original verifier $\mathbf{Ver}$ accepts this certificate. ◀

[6] In [6] it is claimed that the factor saved is $b(t)$, but we believe it should be $b(t-1)$, as the verifier requires one extra round to make sure that all nodes have reconstructed the same certificate.

Bounding the message size. In the universal t-PLS construction from [18], the verifier uses messages whose size is bounded by the size of the certificate, whereas our construction above (and the corresponding one from [6]) requires nodes to learn their entire t-neighborhood, which cannot be done using small messages. However, our construction is easily modified to work with messages whose size is bounded by the certificate size: instead of learning the entire neighborhood, we use pipelining to have each node collect t pieces from its t-neighborhood, allowing us to save a factor of $\Theta(t)$ in the certificate size (instead of $b(t-1)$). A bit of additional effort is required to ensure that all nodes reconstruct the same certificate; the details are deferred to the full version of the paper.

Handling benign data corruption. If we replace erasure codes by their more powerful cousins, *error-correcting codes*, we gain the ability to withstand some bounded number of *corrupted certificates* in every neighborhood. Since proof-labeling schemes are intended to enable fault-tolerance and self-stabilization, this can be useful. We introduce the notion of a *resilient t-PLS*:

▶ **Definition 5.** *Let $f : \mathbb{N} \to \mathbb{N}$, $\mathcal{G}_\mathcal{I}$ a family of graph configurations and $\mathcal{P}$ a Boolean predicate over $\mathcal{G}_\mathcal{I}$. An* f-resilient t-PLS for $\mathcal{P}$ *is a t-PLS $\Pi = (\mathbf{Prv}, \mathbf{Ver})$ satisfying:*

- *Soundness, defined as in Section 3.*
- *f-resilient completeness: for every configuration $(G, I) \in \mathcal{P}$ and node $v \in V(G)$, if a' is a certificate assignment that agrees with $\mathbf{Prv}(G, I)$ on all but at most $f(r)$ certificates in $B_r(v)$ for every $r \leq t$, then $\mathbf{Ver}(G, I, a', v) = 1$.*

In the full version of the paper, we show that using error-correcting codes, we can withstand $f(t)$ potential corruptions in every t-neighborhood, and still save a factor of roughly $b(t-1) - 2f(t-1)$ in the certificate size.

5 Partition-Based t-Proof-Labeling Schemes

Next, we introduce our constructions for t-PLS in graphs with small separators. We begin by defining a class of t-PLS, called *partition-based schemes*, which codify the idea underlying our constructions and several constructions from [18, 6], and in the following sections we give the details of our two concrete constructions.

In a partition-based scheme, we partition the graph into vertex-disjoint connected components $C_1, \ldots, C_m$, each with a small boundary $Y_i \subseteq C_i$ separating it from the rest of the graph, and distribute the certificates of the boundary nodes across the nodes in their component. The diameter of each component must be less than t, so that during the verification phase, each node in the component can learn the entire component, and the boundary nodes can recover the pieces of their certificate. Formally, we require the following properties for each component C_j in the partition:

(P-1) The induced subgraph $G[C_j]$ is connected and has diameter at most r, where $r < t-1$ is a parameter of the construction.

(P-2) Only boundary nodes may have neighbors outside their own component: if $u \in C_j$ has a neighbor $v \in N(u)$ such that $v \notin C_j$, then $u, v \in \bigcup_{i \in [m]} Y_i$.

Given a 1-PLS $\Pi = (\mathbf{Prv}, \mathbf{Ver})$ with certificate length k, and a partition satisfying the conditions above, we construct a t-PLS along the following outline.

Partition-based provers. Let $k = k(n)$ and $q = q(n, k, d)$, where n is the size of the input graph (which we now assume is known to the nodes), $k(n)$ is the certificate length of Π in graphs of size n, and d is a parameter of our construction.

Our new prover $\mathbf{Prv}'$ begins by computing, for each boundary node $x \in \bigcup_j Y_j$, the certificate a_x that the original prover $\mathbf{Prv}$ would give to node x. The prover then uses $C_{n,k,d}$ to encode a_x across some nodes $U_x \subseteq B_t(x)$ in x's vicinity, giving each node $v \in U_x$ a piece of the encoding of a_x. We refer to a_x as *the* $\mathbf{Prv}$*-certificate of* x, to distinguish it from the certificate assigned by the new prover $\mathbf{Prv}'$. The $\mathbf{Prv}$-certificates of non-boundary nodes are not used; the verifier will guess them.

In addition to the $\mathbf{Prv}$-certificates of the boundary nodes, $\mathbf{Prv}'$ specifies the partition into components, by telling each node the index of the component to which it belongs, and whether or not it is a boundary node. Thus, the certificate that $\mathbf{Prv}'$ assigns to each node $v \in V$ is of the form (cid_v, s_v, P_v), where

- $cid_v \in \mathbb{N}$ is the index of the component to which v belongs,
- $s_v \in \{0, 1\}$ indicates whether or not v is a boundary node, and
- $P_v \subseteq V \times \mathbb{N} \times \mathbb{F}_q$ is a collection of pieces of $\mathbf{Prv}$-certificates for some boundary nodes in v's vicinity. Each piece is of the form (x, i, c_i), where x is the ID of a boundary node, i is the index of the piece, and c_i is the i-th piece in the encoding $Enc_{n,k,d}(a_x) = (c_1, \ldots, c_n)$.

The actual construction of the prover $\mathbf{Prv}'$ depends on the graph family we want to handle: specifically, $\mathbf{Prv}'$ must be able to compute a "good" partition for graphs from the family, and it must be able to assign pieces of the original certificates of the boundary nodes, in such a way that no node receives too many pieces (to keep the certificates of $\mathbf{Prv}'$ short). In the current paper we construct two partition-based provers:

- In Section 7 we construct a prover for graph families with a small edge separator, and
- In Section 8 we construct a prover for graph families that exclude some fixed minor.

The partition-based verifier. We define *a single verifier* $\mathbf{Ver}^t_{r,d}$, parameterized by:

- The number of rounds t that the verifier may use,
- An upper bound $r < t - 1$ on the diameter of each component C_i in the honest prover's partition,
- The parameter d of the erasure codes $C_{n,k(n),d}$ used by the honest prover.

The verifier $\mathbf{Ver}^t_{r,d}$ is sound whenever the original 1-PLS verifier $\mathbf{Ver}$ on which it is based is sound. To yield a sound and complete t-PLS, it can be combined with any partition-based prover that matches the parameters r, d, t and distributes certificate pieces in a way that each node can reconstruct what it needs.

We formally define $\mathbf{Ver}^t_{r,d}$ in the next section, but on a high level, it operates as follows: each node v learns its $\Theta(t)$-neighborhood and the certificates in it, and verifies that the component containing v is well-formed and has diameter at most r. Node v uses the certificate-pieces in its $\Theta(t)$-neighborhood to reconstruct an "original certificate" a_x, purportedly computed by the original prover $\mathbf{Prv}$, for each boundary node x in v's component. It then checks if there exists an assignment of "original certificates" to the non-boundary nodes in v's component, that would cause all nodes in the component to accept; if so, v accepts, and otherwise v rejects.

The soundness of this construction stems from the fact that we can "stitch together" the original certificates reconstructed or guessed for the various components, into a certificate assignment that is accepted by the original verifier $\mathbf{Ver}$.

6 The Partition-Based Verifier

In this section we define the partition-based t-round verifier $\mathbf{Ver}^t_{r,d}$, assuming we are given a 1-PLS verifier $\mathbf{Ver}$.

As explained in Section 5, we assume that every node $v \in V$ is given a certificate of the form (cid_v, s_v, P_v), where $cid_v \in \mathbb{N}, s_v \in \{0,1\}$, and $P_v \subseteq \mathcal{V} \times [n] \times \mathbb{F}_q$, where $\mathcal{V}$ is the domain from which UIDs are drawn. We say that node v is a *boundary node* if $s_v = 1$, and otherwise node v is an *inner node*.

Each node v spends the first $t-1$ rounds learning its entire $(t-1)$-neighborhood $B_{t-1}(v)$, including the graph structure, the certificates and the inputs of the nodes. Define the following subsets of $B_{t-1}(v)$ (see Fig. 2):

- $\tilde{C}(v) = \{u \in B_{t-1}(v) : cid_u = cid_v\}$. These are the nodes that the prover has indicated belong to the same component as v.
- $Adj(v) = \big(B_{t-1}(v) \cap N(\tilde{C}(v)\big) \setminus \tilde{C}(v)$. These are the nodes adjacent to $\tilde{C}(v)$ which node v can see in its $(t-1)$-neighborhood.
- $Inner(v) = \big\{u \in \tilde{C}(v) : s_u = 0\big\}$,
- $Boundary(v) = \big(\tilde{C}(v) \setminus Inner(v)\big) \cup Adj(v)$,
- $All(v) = Inner(v) \cup Boundary(v)$.

Intuitively, the nodes in $Inner(v)$ are the nodes for which v will "guess" certificates, and the nodes in $Boundary(v)$ are the nodes for which v will reconstruct certificates from pieces given by the prover.

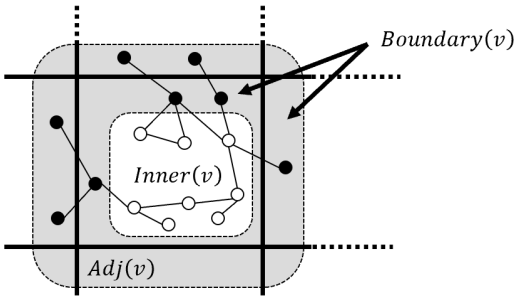

Figure 2 The sets $Inner(v)$, $Boundary(v)$ and $Adj(v)$ computed by the verifier at node v. The bold lines indicate the partition into components, $\big\{\tilde{C}(u)\big\}_{u \in V}$.

Node v verifies the following conditions:

(V-1) The subgraph induced by $\tilde{C}(v)$ on v's $(t-1)$-neighborhood is connected and has diameter at most r.

(V-2) All neighbors of v that are inner nodes are in $\tilde{C}(v)$, that is, for every $u \in N(v)$, if $s_u = 0$, then $cid_u = cid_v$.

The requirements (V-1) and (V-2) mirror our requirements (P-1) and (P-2) from the partition-based prover (see Section 5): if the prover constructed a partition satisfying (P-1) and (P-2), then (V-1) and (V-2) will be satisfied at all nodes. We point out that (V-2), when verified at all nodes, implies that every node in $Boundary(v)$ is a boundary node: if $x \in \big(\tilde{C}(v) \setminus Inner(v)\big)$ then this is immediate, and if $x \in Adj(v)$, then $x \notin \tilde{C}(v)$ but x has a neighbor $y \in \tilde{C}(v)$. If x were not a boundary node, then y would reject when verifying (V-2).

The next step is for node v to reconstruct original certificates for the nodes in $Boundary(v)$, as follows: for each $x \in Boundary(v)$, let $A_v(x) = \{(x, i_x^u, c_x^u) \in P_u : u \in B_{t-1}(v)\}$ be the pieces associated with node x that node v can see in its $(t-1)$-neighborhood. Node v verifies that $|A_v(x)| \geq d$, and that for every $u \neq u'$ in $B_{t-1}(v)$ we have $i_x^u \neq i_x^{u'}$. Then, node v applies the decoder $Dec_{n,k,d}$ to $A_v(x)$ to reconstruct a string $\tilde{a}_v(x) \in \{0,1\}^k$.

After reconstructing a certificate $\tilde{a}_v(x)$ for each $x \in Boundary(v)$, node v sends the list $\{(x, \tilde{a}_v(x)) : x \in Boundary(v)\}$ to all of its neighbors, and receives from each neighbor $u \in N(v)$ a list $\{(x, \tilde{a}_u(x)) : x \in Boundary(u)\}$. Node v verifies that for every $x \in Boundary(v) \cap Boundary(u)$ the same certificate was reconstructed by v and u, that is, $\tilde{a}_v(x) = \tilde{a}_u(x)$.

Following the previous step, node v has an assignment $\tilde{a}_v : \mathit{Boundary}(v) \to \{0,1\}^k$ of certificates to the boundary nodes. It now checks if there exists an extension of $\tilde{a}_v$ to all nodes in $\mathit{All}(v)$, such that the original verifier **Ver** accepts $\tilde{a}_v$ at every node in $\tilde{C}(v)$. Formally, we require that $\mathbf{Ver}(u, I(u), N(u), \tilde{a}_v(u), \{\tilde{a_v}(w)\}_{w\in N(u)}) = 1$ for every $u \in \tilde{C}(v)$. If such an extension exists, then node v accepts, and otherwise it rejects.

In Appendix A, we prove that our verifier is sound, by showing that if $\mathbf{Ver}^t_{r,d}$ accepts at all nodes, then there is a **Prv**-certificate assignment that causes the original verifier **Ver** to accept at all nodes.

7 A Partition-Based t-PLS for Graphs with Small Edge Separators

In this section we construct a partition-based prover $\mathbf{Prv}^t_{sep}$ for graph families that admit *small balanced edge separators*:

▶ **Definition 6.** *Fix $\alpha \in (1/2, 1)$. A set S of edges $S \subseteq E$ is an* α-edge separator *for a graph $G = (V, E)$ if the graph $G' = (V, E \setminus S)$ that remains after the edges in S are removed has no connected components of size larger than $\alpha|V|$.*

We note that in our construction we always choose a *minimal* balanced edge separator, that is, an edge separator S such that the removal of any edge $e \in S$ would yield a set $S \setminus \{e\}$ that is not a balanced edge separator. This will be important for correctness.

The cost of an edge separator. We define the *cost* of an edge separator $S \subseteq E$ to be the number of nodes incident to edges in S: $\mathrm{cost}(S) = |\{u \in V : \exists v (u, v) \in S\}|$.

We note that this differs from the standard definition – usually, one tries to find a set $S \subseteq E$ of minimum cardinality (i.e., the smallest number of edges). However, in our construction, it is the *number of nodes incident to* S that matters, not the size of S itself, because the prover will need to specify a certificate for every node incident to S. Our notion of cost is never greater than the standard notion (the number of edges), but it can be smaller.

Given a graph family $\mathcal{G}$ that is closed under taking subgraphs, we say that $\mathcal{G}$ *admits a balanced edge separator of cost* $s : \mathbb{N} \to \mathbb{N}$ (where s is a non-decreasing function) if every graph $G \in \mathcal{G}$ has a minimal $(2/3)$-balanced edge separator of cost at most $s(|V(G)|)$.

For a graph family that admits balanced edge separators of cost s, we construct a partition-based prover $\mathbf{Prv}^t_{sep}$, as follows.

The decomposition tree. Let $d = \lfloor t/2 \rfloor - 1$. We recursively decompose the input graph G into smaller subgraphs using balanced edge separators, until only components of size smaller than d remain; these components are the partition that will be used by our prover.

The recursive decomposition is represented by a labeled tree T, where

- Each vertex of T is a connected subgraph of G, and the root of T is G itself.
- Each non-leaf vertex $C \subseteq G$ is labeled by a minimal $(2/3)$-balanced edge separator $S_C \subseteq E(C)$ of cost $\leq s(|V(C)|)$.
- If a vertex C of T has size $|V(C)| \geq d$, then C is an inner vertex, and its children are the maximal connected components of the graph $(V(C), E(C) \setminus S_C)$.
- If a vertex C of T has size $|V(C)| < d$, then C is a leaf of T. For convenience, in this case we let $S_C = \emptyset$.

For each tree vertex C, let $X_C \subseteq V(C)$ denote the nodes incident to an edge in S_C. The nodes in X_C will be boundary nodes in our partition.

It is easy to see that the depth of T is $O(\log n)$: at each inner vertex C, the edge set S_C is a $(2/3)$-balanced edge separator, so the children of C have size at most $(2/3)|V(C)|$. We state several additional properties of the decomposition, which will be needed for our t-PLS; the proofs of these properties appear in Appendix B.

In the sequel, we say that C is an *ancestor* of C' if C is on the path from C' to the root of T, including the case $C = C'$.

▶ **Lemma 7.** *Each tree vertex C is an induced subgraph of G, and the leafs $L_1, \ldots, L_k$ of T induce a partition $V(L_1), \ldots, V(L_k)$ of the vertices $V(G)$.*

For a node $v \in V$, let $C(v)$ denote the leaf C of T such that $v \in V(C)$.

▶ **Lemma 8.** *For each $v \in V$, $C(v)$ is a connected subgraph of G of size at most $d - 1$.*

▶ **Corollary 9.** *The construction satisfies (P-1), with $r = d - 1$: for every node $v \in V$, the induced subgraph $G[C(v)]$ is connected and has diameter at most $d - 1 < t - 1$.*

▶ **Lemma 10.** *For every two vertices C, C' of T,*

- *If C is an ancestor of C' in T, then $C' \subseteq C$;*
- *If C is not an ancestor of C' and C' is also not an ancestor of C, then C and C' are vertex-disjoint.*

Let $S = \bigcup_{C \in T} S_C, X = \bigcup_{C \in T} X_C$. The nodes in X serve as the *boundary nodes* from Section 5.

▶ **Lemma 11.** *The construction satisfies condition (P-2): for every node $u \in V$ and neighbor $v \in N(u)$, if $C(u) \neq C(v)$, then $u, v \in X$.*

Next, for a node $v \in V$, let

$$Up(v) = \bigcup_{\text{ancestors } C' \text{ of } C(v)} X_{C'}, \text{ and } \qquad Down(x) = \{v \in V : x \in Up(v)\}.$$

In our t-PLS, every node v receives a piece of the certificate for each node in $Up(v)$. Because T has logarithmic depth, and the separator taken out at each vertex has size at most $s(n)$, we have:

▶ **Lemma 12.** *For each node $v \in V$ we have $|Up(v)| \leq s(n) \log n$.*

On the other hand, it is also important that each node $x \in X$ have at least $\Omega(t)$ nodes to which we can give pieces of x's certificate. If $x \in X$, then there is some inner vertex C of T such that $x \in X_C$. By definition, $V(C) \subseteq Down(x)$. Since inner vertices are connected subgraphs of size at least d, we have:

▶ **Lemma 13.** *For each $x \in X$ we have $|Down(x) \cap B_d(x)| \geq d$.*

The prover. We now define the prover $\mathbf{Prv}^t_{sep}$, given a 1-PLS prover $\mathbf{Prv}$ for some predicate $\mathcal{P}$. Fix a configuration $(G, I) \in \mathcal{P}$, and let a_v be the certificate assigned by $\mathbf{Prv}$ to $v \in V$. Let $n = |V|$, $k = k(n)$ be the length of the certificates produced by $\mathbf{Prv}$, and let $q = q(n, k, d)$ (from Section 3). Finally, let us fix an erasure code $C_{n,k,d} = (Enc, Dec)$ over $\mathbb{F}_q$.

Using the decomposition tree T of G, our new prover assigns certificates as follows: first, the prover assigns a unique identifier $ID(C) \in [n]$ to each leaf C of T. Then, the prover encodes the certificate of each node $x \in X$ across the nodes in $Down(x)$: the prover computes the encoding $Enc(a_x) = (c_x^1, \ldots, c_x^n)$ of x's certificate a_x, and assigns a unique

index $\ell_x(u) \in [|Down(x)|]$ to each node $u \in Down(x)$. We refer to a triplet (x, ℓ, c_x^ℓ) as *the ℓ-th piece of x's certificate*. Each node $v \in Down(x)$ receives the piece $(x, \ell_x(v), c_x^{\ell_x(v)})$ (among other information).[7]

The certificate of node $v \in V$ is $(cid_v, s_v, P_v) \in [n] \times \{0,1\} \times 2^{V \times [n] \times \mathbb{F}_q}$, where

- $cid_v = ID(C(v))$, and
- $s_v = 1$ iff $v \in X$,
- P_v is a collection of certificate pieces: $P_v = \left\{(x, \ell_x(v), c_x^{\ell_x(v)}) : x \in Up(v)\right\}$.

Since $|Up(v)| \leq s(n) \log n$ and $\log q = O(k(n)/d + \log n)$, the size of each certificate is bounded by $O(\log n + ((k(n)/d) + \log n)s(n) \log n) = \tilde{O}(k(n)s(n)/t)$.

In Appendix B, we prove that the t-PLS $(\mathbf{Prv}^t_{sep}, \mathbf{Ver}^t_{r,d})$ is complete. Since we have already shown that $\mathbf{Ver}^t_{r,d}$ preserves the soundness of the original verifier $\mathbf{Ver}$, together this proves Theorem 1 from Section 1.

8 A Partition-Based t-PLS for Graph Families with an Excluded Minor

In this section we show that graph families with an excluded constant-sized minor admit a partition into components $C_0, \ldots, C_m$ of size $r = O(t)$, such that the boundary $Y_i \subseteq C_i$ of each component C_i satisfies $|Y_i|/|C_i| = O(\Delta/\sqrt{r})$. (Here, Δ is the maximum degree in the graph.) This allows the prover to split the certificate of each boundary node $y \in Y_i$ into roughly $\sqrt{r}/\Delta$ pieces, and assign them to the nodes of C_i, so that each node receives only a single certificate piece in total.

Our construction is based on the *r-region decomposition* of Frederickson [9], which takes a planar graph $G = (V, E)$ and produces $O(n/r)$ sets of nodes, $R_1, \ldots, R_m \subseteq V$, called *regions*. The regions are not necessarily vertex-disjoint; nodes that appear in more than one region are called *border nodes*.[8] The regions $R_1, \ldots, R_m$ satisfy:

- For each edge $\{u, v\} \in E$, there is some region R_i such that $u, v \in R_i$,
- $|R_i| \leq r$ for each i, and
- Each region contains at most $O(\sqrt{r})$ border nodes: $|R_i \cap \bigcup_{j \neq i} R_j| = O(\sqrt{r})$ for each i.

We note that G need not be a connected graph for Frederickson's construction, and the regions $R_1, \ldots, R_m$ need not induce connected subgraphs.

Although Frederickson originally stated his results only for planar graphs, the r-region decomposition from [9] relies only on the existence of a *balanced weighted vertex separator of size $O(\sqrt{n})$* (we omit the definition of this object here, as it is not needed directly for our construction). Later, [1] showed that any graph family with a constant-size excluded minor H admits a balanced weighted separator of size $O(c_H\sqrt{n})$, where c_H is a constant depending on the minor H. Thus, Frederickson's r-region decomposition applies to any graph family with an excluded minor of constant size.[9]

The r-region decomposition is an excellent starting point for constructing a partition for our prover, but there are two issues we need to solve:

---

[7] Note that while the encoding $Enc(a_x)$ of x's certificate comprises n pieces, only the first $|Down(x)|$ pieces will actually be used, and the rest are not given to any graph node. This is fine: because $|Down(x)| \geq d$ (Lemma 13), we can still recover a_x from the pieces that were actually used.

[8] In [9] these nodes are called *boundary nodes*, but here we refer to them as *border nodes*, as we use *boundary nodes* to mean something different in the context of Sections 5, 6.

[9] In fact, Frederickson's decomposition can be extended to any graph family with *polynomial expansion*, as it is known that such graph families have sublinear vertex separators. This allows us to extend the construction given here to any graph family with polynomial expansion, yielding a t-PLS with that saves a factor of $O(\Delta/t^\delta)$ for some $\delta \in (1/2, 1)$. The details are deferred to the full version of the paper.

- The subgraph $G[R_i]$ induced by a region R_i is not necessarily connected, and
- Regions can be arbitrarily small, so even though each region has size at most r and contains $O(\sqrt{r})$ border nodes, the ratio between the size of the region and the number of border nodes in it is unbounded. This means there may not be enough nodes inside the region to allow us to partition the certificates of the border nodes across them.

To address these issues, we start from an r-region decomposition of the graph, and show that we can carve out at least one component C of size $O(r)$, which is connected and contains at most $O(|C|/\sqrt{r})$ border nodes. We recurse on the connected components that remain after C is removed, until we have covered the original graph in its entirety.

▶ **Lemma 14.** *Let $\mathcal{G}$ be a family of graphs that exclude a fixed minor H. Then there exists a constant $\alpha \in \mathbb{R}^+$ such that for each $r \leq n$, every graph $G \in \mathcal{G}$ admits a partition $C_0, \ldots, C_m$ of $V(G)$, along with border sets $X_0 \subseteq C_0, \ldots, X_m \subseteq C_m$, such that*

(R-1) *Condition (P-1) is satisfied: for each i, $G[C_i]$ is a connected subgraph and $|C_i| \leq r$.*

(R-2) *For each $i \in [m]$, $u \in C_i$ and $v \in V \setminus C_i$ such that $\{u, v\} \in E$, either $u \in \bigcup_{j<i} X_j$ or $v \in \bigcup_{j<i} X_j$ (or both).*[10]

(R-3) *For each $i \in [m]$ we have $|C_i|/|X_i| \geq \alpha\sqrt{r}$ (or $|X_i| = 0$).*

Proof sketch. We describe how the partition is constructed; the proof that the conditions of the lemma are satisfied is deferred to Appendix C.

Let $c_1, c_2 \geq 1$ be the constants from Frederickson's construction, so that for every graph over at least r vertices, there is an r-region decomposition comprising at most $c_1 n/r$ regions, each of size at most r, and each containing at most $c_2\sqrt{r}$ border nodes.

Let $\alpha = 1/(c_1 c_2)$. We build our partition recursively, by carving out a sequence of components $C_0, \ldots, C_m$, each with a border set $X_i \subseteq C_i$, satisfying the conditions of the lemma. Suppose we have already found and removed the first $i \geq 0$ components, and let $G_i = G[V_i]$ be the remaining graph, where $V_i = V \setminus \bigcup_{j<i} C_j$.

If $|V_i| < r$, we cannot apply the region decomposition to G_i, as it is too small. In this case we let C_i be some maximal connected component of G_i, and let $X_i = \emptyset$. It is easy to see that the conditions of the lemma are satisfied.

If $|V_i| \geq r$, then let $R_1, \ldots, R_m$ be an r-region decomposition of G_i, comprising at most $c_1|V_i|/r$ regions, each of size at most r and containing at most $c_2\sqrt{r}$ border nodes (here, "border nodes" means nodes that appear in more than one region $R_1, \ldots, R_m$; we do not consider regions removed in previous steps of the construction). Since we have at most $c_1|V_i|/r$ regions that together cover all nodes of V_i, and since each region contains at most r nodes, there is some region R^* such that $r/c_1 \leq |R^*| \leq r$. Let $R^- = \bigcup_{R' \neq R^*} R'$ be the union of all the regions except R^*, and let $W_1, \ldots, W_s$ be the maximal connected components of the induced subgraph $G_i[R^*]$. We claim that there is some maximal component W_j that has

$$\frac{|W_j|}{|W_j \cap R^-|} \geq \frac{\sqrt{r}}{c_1 c_2}, \tag{3}$$

otherwise the total number of border nodes in R^* would be too large:

$$|R^* \cap R^-| = \sum_{j=1}^{s} |W_j \cap R^-| > \sum_{j=1}^{s} |W_j| \cdot \frac{c_1 c_2}{\sqrt{r}} = \frac{c_1 c_2}{\sqrt{r}} |R^*| \geq \frac{c_1 c_2}{\sqrt{r}} \frac{r}{c_1} = c_2\sqrt{r},$$

a contradiction to the fact that every region contains at most $c_2\sqrt{r}$ border nodes.

[10] Note that this condition is weaker than (P-2).

We define the component C_i to be some maximal connected component W_j satisfying (3), and we let $X_i = W_j \cap R^-$, the border nodes in W_j. The proof that the conditions of the lemma are satisfied appears in Appendix C. ◀

The prover. We are now ready to define the prover $\mathbf{Prv}^t_{minor}$, given a 1-PLS $(\mathbf{Prv}, \mathbf{Ver})$ for some predicate $\mathcal{P}$.

Let $\mathcal{G}$ be a family of graphs with an excluded minor H, and let α be the constant from Lemma 14. Let $r = \lfloor t/2 \rfloor - 1$ and let $d = \alpha\sqrt{r}/(\Delta+1)$. Let $C_{n,k,d} = (Enc_{n,k,d}, Dec_{n,k,d})$ where $k = k(n)$ is the certificate size of the original PLS and let $q = q(n,k,d)$ be the prime from Section 3.

Given a configuration $(G, I) \in \mathcal{G}_\mathcal{I}$ over n nodes with maximum degree Δ, the prover $\mathbf{Prv}^t_{minor}$ computes the certificates $\{a_v\}_{v\in V}$ assigned by $\mathbf{Prv}$ to (G, I), and the partition $C_0, \ldots, C_m, X_0, \ldots, X_m$ from Lemma 14.

For each component C_i with border $X_i \subseteq C_i$, let $Y_i = X_i \cup N(X_i)$. The prover partitions C_i into $|Y_i|$ sets, $\{D_y\}_{y\in Y_i}$, each of size at least $d = \alpha\sqrt{r}/(\Delta+1)$. This is possible, because

$$|Y_i| \leq (\Delta+1)|X_i| \leq (\Delta+1)\,|C_i|/(\alpha\sqrt{r}).$$

For each node $y \in Y_i$, the prover computes the encoding $Enc_{n,k,d}(a_y) = (c^1_y, \ldots, c^n_y)$ of a_y into n pieces. The prover also assigns a unique index $\ell_y(v) \in [d]$ to each node $v \in D_y$, and node $v \in D_y$ will then receive the piece $(y, \ell_y(v), c^{\ell_y(v)}_y)$ (among other information). The certificate of node $v \in C_i$ is given by (i, s_v, P_v), where

- $s_v = 1$ iff $v \in Y_i$,
- $P_v = \left\{(y, \ell_y(v), c^{\ell_y(v)}_y) : v \in D_i\right\}$. This consists of a single certificate piece, because the sets $D_y, D_{y'}$ are disjoint for $y \neq y'$.

The certificate size is bounded by $O(\log n + \log q) = O((k/d) + \log n) = \tilde{O}(k\Delta/\sqrt{t})$.

This completes the definition of $\mathbf{Prv}^t_{minor}$. The proof that the t-PLS $(\mathbf{Prv}^t_{minor}, \mathbf{Ver}^t_{r,d,q})$ is complete appears in Appendix C. Together, this proves Theorem 2 from Section 1.

References

1. Noga Alon, Paul D. Seymour, and Robin Thomas. A separator theorem for graphs with an excluded minor and its applications. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 293–299, 1990.
2. Keren Censor-Hillel, Ami Paz, and Mor Perry. Approximate proof-labeling schemes. In *Structural Information and Communication Complexity - 24th International Colloquium (SIROCCO)*, volume 10641, pages 71–89, 2017.
3. Laurent Feuilloley and Pierre Fraigniaud. Survey of distributed decision. *Bull. EATCS*, 119, 2016.
4. Laurent Feuilloley and Pierre Fraigniaud. Error-sensitive proof-labeling schemes. In *31st International Symposium on Distributed Computing*, volume 91, pages 16:1–16:15, 2017.
5. Laurent Feuilloley, Pierre Fraigniaud, and Juho Hirvonen. A hierarchy of local decision. *Theor. Comput. Sci.*, 856:51–67, 2021.
6. Laurent Feuilloley, Pierre Fraigniaud, Juho Hirvonen, Ami Paz, and Mor Perry. Redundancy in distributed proofs. *Distributed Computing*, 34(2):113–132, 2021.
7. 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.
8. Pierre Fraigniaud, Boaz Patt-Shamir, and Mor Perry. Randomized proof-labeling schemes. *Distributed Computing*, 32, 2019.

9 Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16(6):1004–1022, 1987.

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

11 Gillat Kol, Rotem Oshman, and Raghuvansh R. Saxena. Interactive distributed proofs. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 255–264, 2018.

12 Liah Kor, Amos Korman, and David Peleg. Tight bounds for distributed minimum-weight spanning tree verification. *Theory Comput. Syst.*, 53(2):318–340, 2013.

13 Janne H. Korhonen and Jukka Suomela. Towards a complexity theory for the congested clique. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 163–172, 2018.

14 Amos Korman and Shay Kutten. Distributed verification of minimum spanning trees. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 26–34, 2006.

15 Amos Korman and Shay Kutten. On distributed verification. In *Distributed Computing and Networking, 8th International Conference, ICDCN*, volume 4308, pages 100–114, 2006.

16 Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. *Distributed Comput.*, 22(4):215–233, 2010.

17 Moni Naor, Merav Parter, and Eylon Yogev. The power of distributed verifiers in interactive proofs. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1096–115, 2020.

18 Rafail Ostrovsky, Mor Perry, and Will Rosenbaum. Space-time tradeoffs for distributed verification. In *International Colloquium on Structural Information and Communication Complexity*, pages 53–70. Springer, 2017.

19 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, 2017.

20 Ron M. Roth. *Introduction to coding theory.* Cambridge University Press, 2006.

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

22 Jukka Suomela. Survey of local algorithms. *ACM Comput. Surv.*, 45(2):24:1–24:40, 2013.

A Soundness of the Partition-Based Verifier

Fix a certificate assignment $\{(cid_v, s_v, P_v) : v \in V\}$ that is accepted by all nodes under $\mathbf{Ver}^t_{r,d}$, and let us construct a certificate assignment $\{a_v : v \in V\} \subseteq \{0,1\}^k$ that is accepted by all nodes under the original verifier **Ver**: for each node $v \in V$, let $z_v \in \tilde{C}(v)$ be the node that has the smallest ID in $\tilde{C}(v)$. Then we define $a_v = \tilde{a}_{z_v}(v)$, where $\tilde{a}_{z_v}$ is the certificate assignment found by z_v during the verification phase (i.e., the certificate assignment that makes **Ver** accept at all nodes in $\tilde{C}(z_v)$). We now prove that our stitched-together certificate assignment $\{a_v\}_{V \in V}$ is indeed accepted by **Ver** at all nodes.

First, it is not hard to see that if u is in the component $\tilde{C}(v)$, then u and v agree on the structure of the component:

▶ **Lemma 15.** *If $u \in \tilde{C}(v)$ then $\tilde{C}(u) = \tilde{C}(v)$, $Boundary(u) = Boundary(v)$, and $Adj(u) = Adj(v)$.*

Proof. Node v verifies that $\tilde{C}(v)$ induces a connected component of size at most r inside $B_{t-1}(v)$. Since $u \in \tilde{C}(v)$, for every $w \in \tilde{C}(v)$ there is a path of length at most r between u and w; therefore $\tilde{C}(v) \subseteq B_r(u) \subseteq B_{t-1}(u)$, and since all nodes $w \in \tilde{C}(v)$ have $cid_w = cid_v = cid_u$, it follows that $\tilde{C}(v) \subseteq \tilde{C}(u)$. In particular, $v \in \tilde{C}(u)$, and the same reasoning now shows that $\tilde{C}(u) \subseteq \tilde{C}(v)$ as well. Together we have $\tilde{C}(v) = \tilde{C}(u)$.

Now consider $Adj(v)$. By definition, $w \in Adj(v)$ iff $w \in (B_{t-1}(v) \cap N(\tilde{C}(v))) \setminus \tilde{C}(v)$, that is, $w \notin \tilde{C}(v)$ and there is a node $x \in \tilde{C}(v)$ such that $w \in N(x)$. Node v has verified that $\tilde{C}(v)$ induces a connected component of size at most r inside $B_{t-1}(v)$, so there is a path of length at most $r < t-1$ between node x and node u, implying that $w \in N(x) \subseteq B_{t-1}(u)$. Together with the fact that $\tilde{C}(u) = \tilde{C}(v)$, which we showed above, we see that $w \in (B_{t-1}(u) \cap N(\tilde{C}(u))) \setminus \tilde{C}(u) = Adj(u)$. Containment in the other direction is similar, with the roles of u and v exchanged.

Finally, what we showed above implies that $Boundary(u) = Boundary(v)$: by definition, $Boundary(u) = (\tilde{C}(u) \setminus Inner(u)) \cup Adj(u)$. We already showed that $\tilde{C}(u) = \tilde{C}(v)$ and $Adj(u) = Adj(v)$. It is immediate that $Inner(u) = Inner(v)$, as $Inner(u)$ comprises all the inner nodes in $\tilde{C}(u)$, and similarly for v. The claim follows. ◀

Next, to show that we stitched together the certificates of the various components in a consistent manner, it is crucial to prove that all nodes agree on the certificates they reconstructed for all boundary nodes they have in common:

▶ **Lemma 16.** *Let $u, v \in V$, and let $x \in Boundary(u) \cap Boundary(v)$. Then $\tilde{a}_u(x) = \tilde{a}_v(x)$.*

Proof. We show that there is a path π_u between u and x, and a path π_v between v and x, such that all nodes w on both paths have $x \in Boundary(w)$. Since every node w verifies that it agrees with its neighbors on certificates they reconstructed for the nodes in $Boundary(w)$, this implies the claim.

We prove the existence of the path π_u between u and x; π_v is similar. Consider first the case where $x \in \tilde{C}(u)$. Node u has verified that $\tilde{C}(u)$ is connected and that $|\tilde{C}(u)| \leq r$, so there is a path $\pi_u = u_0, \ldots, u_\ell$ from u to x, such that $\ell \leq r$, $u_0 = u, u_\ell = x$, and $u_0, \ldots, u_\ell \in \tilde{C}(u)$. By Lemma 15, each path node u_i has $Boundary(u_i) = Boundary(u)$; and since $x \in Boundary(u)$, this shows that every node u_i in the path has $x \in Boundary(u_i)$.

Now suppose that $x \notin \tilde{C}(u)$. Then we must have $x \in Adj(u)$, as $x \in Boundary(u) = \big(\tilde{C}(u) \setminus Inner(u)\big) \cup Adj(u)$. By definition of $Adj(u)$, node x has a neighbor $y \in N(x) \cap \tilde{C}(u)$. Since $\tilde{C(u)}$ is connected and $|\tilde{C}(u)| \leq r$, there is a path $\pi_u = u_0, \ldots, u_\ell, u_{\ell+1}$ from u to x, such that $\ell \leq r$, $u_0 = 0$, $u_\ell = y$, $u_{\ell+1} = x$, and $u_0, \ldots, u_\ell \in \tilde{C}(u)$. By Lemma 15, each path node u_i where $i \leq \ell$ has $Adj(u_i) = Adj(u)$ and hence $x \in Adj(u_i) \subseteq Boundary(u_i)$. And of course the last node, $u_{\ell+1} = x$, also has $x \in Boundary(x)$, as $x \in \tilde{C}(x) \setminus Inner(x)$. ◀

Finally, we show that the "representative" z_v, whose certificate assignment $\tilde{a}_z$ we used to define a_v, in fact has v in its component $\tilde{C}(z_v)$:

▶ **Lemma 17.** *For each $v \in V$ we have $v \in \tilde{C}(z_v)$.*

Proof. Let $z = z_v$. By definition, $z \in \tilde{C}(v)$, so by Lemma 15, $\tilde{C}(v) = \tilde{C}(z)$. Since $v \in \tilde{C}(v)$, we also have $v \in \tilde{C}(z)$. ◀

We can now conclude that under the original verifier **Ver**, all nodes accept the certificate assignment a that we defined:

▶ **Lemma 18.** *For each node $v \in V$ we have $\mathbf{Ver}(v, I(v), N(v), a_v, \{a_u\}_{u \in N(v)}) = 1$.*

Proof. Let $v \in V$, and let $z = z_v \in \tilde{C}(v)$. Since $v \in \tilde{C}(z)$ (Lemma 17), node z verifies that v accepts $\tilde{a}_z$: formally, node z verifies that $\mathbf{Ver}(v, I(v), N(v), \tilde{a}_z(v), \{\tilde{a}_z(u)\}_{u \in N(v)}) = 1$. We now prove that the certificate assignments $\tilde{a}_z$ and a agree on the certificates of v and all of its neighbors: that is, $\tilde{a}_z(u) = a_u$ for each node $u \in \{v\} \cup N(v)$.

- For v itself, we defined $a_v = \tilde{a}_z(v)$.
- For a neighbor $u \in N(v) \cap \tilde{C}(v)$ inside v's component, Lemma 15 shows that $\tilde{C}(u) = \tilde{C}(v)$. In particular, $z_u = z_v = z$ (the node with the smallest ID in the component), and therefore $a_u = \tilde{a}_{z_u}(u) = \tilde{a}_z(u)$.
- For a neighbor $u \in N(v) \setminus \tilde{C}(v)$ outside v's component, node u must be a boundary node ($s_u = 1$), otherwise condition (V-2) would cause u to reject. Also, $u \in \mathit{Adj}(v)$, by definition of $\mathit{Adj}(v)$. Lemma 15 shows that $\mathit{Adj}(z) = \mathit{Adj}(v)$. and hence $u \in \mathit{Adj}(z) \subseteq \mathit{Boundary}(z)$. Now let $z' = z_u$ be the node "responsible" for u; we defined $a_u = \tilde{a}_{z'}(u)$. By Lemma 17 we have $u \in \tilde{C}(z')$, and since u is a boundary node, $u \in \mathit{Boundary}(z')$. We now have that $u \in \mathit{Boundary}(z) \cap \mathit{Boundary}(z')$, and therefore, by Lemma 16, $a_u = \tilde{a}_{z'}(u) = \tilde{a}_z(u)$.

Since node z has verified that $\mathbf{Ver}(v, I(v), N(v), \tilde{a}_z(v), \{\tilde{a}_z(u)\}_{u \in N(v)}) = 1$, and we have shown that $a_u = \tilde{a}_z(u)$ for every $u \in \{v\} \cup N(v)$, we immediately see that the verifier at node v accepts the assignment a, i.e., $\mathbf{Ver}(v, I(v), N(v), a_v, \{a_u\}_{u \in N(v)}) = 1$. ◀

B Missing Proofs from Section 7

Proof of Lemma 7. It is easy to see that at each level of the tree, the vertices $C_1, \ldots, C_k$ at that level induce a partition $V(C_1), \ldots, V(C_k)$ of $V(G)$, because the children of each vertex are the connected components that remain after removing some edges (but no vertices are removed). Thus, in particular, the leafs of T induce a partition of $V(G)$.

Next we show that each vertex C is an induced subgraph of G, by induction on the distance from the root to C. The base case is immediate, because the root of the tree is G itself. For the induction step, suppose that C is an induced subgraph of G, and let D be a child of C. Suppose for the sake of contradiction that there is an edge $\{u, v\} \in E(G)$, such that $u, v \in V(D)$, but $\{u, v\} \notin E(D)$. Recall that by definition of the tree, D is a maximal connected component of the graph $(V(C), E(C) \setminus S_C)$. Since $D \subseteq C$, we have $u, v \in V(C)$, and since C *is* an induced subgraph of G, we must have $\{u, v\} \in E(C)$; therefore $\{u, v\}$ must have been removed in the recursive step at C as part of the edge separator, i.e., $\{u, v\} \in S_C$. But since u, v are in the same connected component C of the graph $(V(C), E(C) \setminus S_C)$, there is some path $\pi_{u,v}$ between u and v that does not include any edge from S_C. We argue that this contradicts the minimality of S_C, as taking out only the edges $S_C \setminus \{\{u, v\}\}$ yields exactly the same maximal connected components as taking out all of S_C.

To see this, let us abuse notation slightly, and denote by $D \setminus F$ the graph $(V(D), E(D) \setminus F)$. We claim that for any two nodes $x, y \in D$, node x is reachable from y in $D \setminus S_C$ iff it is reachable from y in $D \setminus (S_C \setminus \{\{u, v\}\})$: clearly, if x is reachable from y in $D \setminus S_C$, then it is also reachable from y in $D \setminus (S_C \setminus \{\{u, v\}\})$. For the other direction, if x is reachable from y in $D \setminus (S_C \setminus \{\{u, v\}\})$, then we can also reach x from y by a path that does not include any edge of S_C: given a path that avoids all edges in $S_C \setminus \{\{u, v\}\}$ but does include $\{u, v\}$, we simply replace $\{u, v\}$ by the path $\pi_{u,v}$ whose existence we showed above, which does not include any edges of S_C. ◀

Proof of Lemma 8. Since $C(v)$ is a leaf of the decomposition tree, $|C(v) \leq d - 1$. Also, every vertex of the decomposition tree is a connected subgraph of G. ◀

Proof of Lemma 10. If C is an ancestor of C', an easy induction on the distance between C and C' in T shows that C' is a subgraph of C: the children of every inner vertex D of T are subgraphs of D. If C, C' are not ancestors of one another, then let D be the lowest common ancestor of C and C'. We have $D \neq C, D \neq C'$. Let D' be the child of D that is an ancestor of C, and let D'' be the child of D that is an ancestor of C'. Then D', D'' are

vertex-disjoint, as they are both maximal connected components of the graph obtained from D by removing the edges in S_D. Since we have already shown that $C \subseteq D'$ and $C' \subseteq D''$, it follows that C and C' are also vertex-disjoint. ◀

Proof of Lemma 11. Let D be the lowest common ancestor of $C(u)$ and $C(v)$ in T, and let D_u, D_v be the children of D that are ancestors of $C(u), C(v)$, respectively. Of course, $D_u \neq D_v$. By Lemma 10, we have $C(u) \subseteq D_u \subseteq D$ and $C(v) \subseteq D_v \subseteq D$, and hence $u \in D_u$, $v \in D_v$, and $u, v \in D$. And since D is an induced subgraph of G (Lemma 7), and we know that $\{u, v\} \in E$ (as $v \in N(u)$), it must be that $\{u, v\} \in E(D)$. But D_u and D_v are maximal connected components of $(V(D), E(D) \setminus S_D)$, and so the edge $\{u, v\}$ must be in S_D, otherwise u, v would be in the same child of D. It follows that $u, v \in X_D \subseteq X$. ◀

Proof of Lemma 12. T has depth $O(\log n)$, and at every ancestor C' of $C(v)$ we have $|X_{C'}| \leq s(|C'|) \leq s(n)$. The claim follows. ◀

Proof of Lemma 13. Since $x \in X$, there is some inner vertex C such that $x \in X_C$, and by definition, for every $v \in C$ we have $x \in \mathit{Up}(v)$. Therefore $V(C) \subseteq \mathit{Down}(x)$. In particular, since C is a connected subgraph of size at least d, and since $x \in X_C \subseteq V(C)$, we have $|V(C) \cap B_d(x)| \geq d$. Therefore $|\mathit{Down}(x) \cap B_d(x)| \geq |V(C) \cap B_d(x)| \geq d$. ◀

Completeness of the prover $\mathbf{Prv}^t_{sep}$. To show that the honest prover $\mathbf{Prv}^t_{sep}$ causes all nodes to accept, we relate the values that nodes compute during their verification to the decomposition tree computed by the prover. We continue to refer to nodes as 'inner' or 'boundary' nodes, except that now, since we are working with the honest prover, we know that v is a boundary node (i.e., $s_v = 1$) iff $v \in X$.

▶ **Lemma 19.** *For each inner node v we have $\tilde{C}(v) = C(v)$.*

Proof. Recall that $\tilde{C}(v)$ is the set of nodes u in $B_{t-1}(v)$ that have $\mathit{cid}_u = \mathit{cid}_v$. By definition, the prover assigns $\mathit{cid}_u = \mathit{ID}(C(u))$ to each node u. Therefore, $\mathit{cid}_u = \mathit{cid}_v$ iff $C(u) = C(v)$, that is, iff $u \in C(v)$. It follows that $\tilde{C}(v) = C(v) \cap B_{t-1}(v)$. But by Lemma 8 we have $C(v) \subseteq B_d(v)$, implying that $C(v) \cap B_{t-1}(v) = C(v)$, and therefore, $\tilde{C}(v) = C(v)$. ◀

▶ **Lemma 20.** *Conditions **(V-1)** and **(V-2)** are satisfied at every node $v \in V$.*

Proof. Fix $v \in V$, and let us verify that the conditions hold.

V-1: By Lemma 19 we have $\tilde{C}(v) = C(v)$, and since $C(v)$ is a leaf of the decomposition tree, it is a connected component of size $|C(v)| \leq r = d - 1$. Thus, $\tilde{C}(v)$ induces a connected component of size at most r in $B_{t-1}(v)$.

V-2: Let $u \in N(v)$ be an inner node ($s_u = 0$). Then $u \notin X$, and by Lemma 11, every neighbor of u must be in the same component, $C(u) = C(v)$. This implies that $\mathit{cid}_u = \mathit{cid}_v$. ◀

Next we show that the checks associated with the reconstruction of the boundary nodes' certificates succeed:

▶ **Lemma 21.** *Let $v \in V$. For each $x \in \mathit{Boundary}(v)$ we have $|A_v(x)| \geq d$, and all piece indices that appear in $A_v(x)$ are distinct. Moreover, the reconstructed certificate $\tilde{a}_v(x)$ is the true certificate a_x.*

Proof. First, note that x is a boundary node ($s_x = 1$): if $x \in \tilde{C}(v) \setminus \mathit{Inner}(v)$ then this is immediate by definition of $\mathit{Inner}(v)$. Otherwise, we have $x \in \mathit{Adj}(v)$, meaning that $x \notin \tilde{C}(v)$ and there is some neighbor $y \in \tilde{C}(v)$ such that $x \in N(y)$. Inner nodes cannot have neighbors

from a different component (by (V-2), which as we proved in Lemma 20 is satisfied). Therefore x is a boundary node, and the prover disperses pieces of a_x across all the nodes in $Down(x)$.

By Lemma 13, there are at least d nodes in $Down(x) \cap B_d(x)$. All these nodes are in $B_{t-1}(v)$: because $|\tilde{C}(v)| \leq r = d-1$ (Lemma 8) and $x \in N(\tilde{C})$, we have $x \in B_d(v)$, and therefore $B_d(x) \subseteq B_{2d}(v) \subseteq B_{t-1}(v)$ (as $d = \lfloor t/2 \rfloor - 1$). Consequently, $A_v(x)$ includes at least d distinct pieces of a_x, and the decoder D reconstructs a_x successfully. ◀

▶ **Lemma 22.** *The verifier* $\mathbf{Ver}^t_{r,d}$ *accepts at all nodes* $v \in V$.

Proof. We have already shown that conditions (V-1) and (V-2) are satisfied at all nodes, and that the reconstruction step succeeds. It remains to prove that for every node $v \in V$, there is some certificate assignment $\tilde{a}_v$ that agrees with the certificates that v reconstructed for each boundary node $x \in Boundary(v)$, and which causes **Ver** to accept at all nodes in $C(v)$. Naturally, we define $\tilde{a}_v(u) = a_u$ for each $u \in All(v)$. By Lemma 21, this indeed agrees with the certificates that v reconstructed for its boundary nodes.

Since we know that $\mathbf{Ver}(u, I(u), N(u), a_u, \{a_w\}_{w \in N(u)}) = 1$ for all $u \in V$, we get that for every $u \in C(v)$, we have $\mathbf{Ver}(u, I(u), N(u), \tilde{a}_v(u), \{\tilde{a}_v(w)\}_{w \in N(u)}) = 1$. ◀

C Missing Proofs from Section 8

Conditions (R-1)–(R-3) in the case where $|V_i| < r$. We prove that C_i, X_i satisfy the requirements:

(R-1) (P-1): clearly, $|C_i| \leq |V_i| < r$, and C_i is a maximal connected component of G_i, which is an induced subgraph of G. Therefore $G[C_i]$ is a a connected subgraph of size at most r,as required.

(R-2) Let $u \in C_i$ and $v \in V \setminus C_i$ be such that $\{u, v\} \in E$. It cannot be that $v \in V_i$, because V_i is an induced subgraph of G, so this would imply that $\{u, v\} \in E[G_i]$, contradicting the fact that C_i is a maximal connected component of G_i. Thus, $v \notin V_i$, meaning that $v \in C_j$ for some $j < i$. In addition, since $C_j \cap V_i = \emptyset$ and $u \in C_i \subseteq V_i$, we have $u \notin C_j$. Because C_j satisfies the requirements of the lemma, this implies that either $u \in \bigcup_{j' < j} X_{j'}$ or $v \in \bigcup_{j' < j} X_{j'}$. And since $j < i$, the requirement is satisfied for C_i as well.

(R-3) We have $|X_i| = 0$.

Conditions (R-1)–(R-3) in the case where $|V_i| \geq r$. Let us prove that the requirements are satisfied:

(R-1) We have $|C_i| \leq |R^*| \leq r$, and since C_i is a maximal connected component of $G_i[R^*]$, and G_i is itself an induced subgraph of G, we see that $G[C_i]$ is a connected subgraph of size at most r, as required.

(R-2) Let $u \in C_i$ and $v \in V \setminus C_i$ such that $\{u, v\} \in E$. There are two cases:

- $v \in G_i$: note that since C_i is a maximal connected component of $G_i[R^*]$ and G_i is itself an induced subgraph of G, we must have $v \notin R^*$, otherwise the presence of the edge $\{u, v\} \in E$ would mean that u, v are both in the same maximal connected component. Thus, there is some region $R' \neq R$ that covers the edge $\{u, v\}$, that is, $u, v \in R'$. But this implies that $u \in R^* \cap R'$, so u is a border node, and $u \in X_i$.
- $v \notin G_i$: then since G_i is the graph induced by $V_i = V \setminus \bigcup_{i' < i} C_{i'}$, there is some component $C_{i'}$, $i' < i$, such that $v \in C_{i'}$. Since $C_{i'}$ satisfies (R-2), we have either $u \in \bigcup_{i'' < i'} X_{i''}$ or $v \in \bigcup_{i'' < i'} X_{i''}$, and since $i' < i$, this implies that (R-2) is satisfied for C_i as well.

(R-3) Holds by choice of C_i as a maximal connected component W_j that satisfies (3).

Completeness of the prover $\mathbf{Prv}^t_{minor}$. Suppose that $(G, I) \in \mathcal{P}$, and let us show that all nodes accept the honest prover's certificates. As in Section 7, we do so by relating the values that nodes compute during their verification to the partition the prover computed.

▶ **Lemma 23.** *For each node $v \in C_i$ we have $\tilde{C}(v) = C_i$.*

Proof. By definition, the prover assigns the same index $cid_u = i$ to all nodes $u \in C_i$. Therefore $\tilde{C}(v) = C(v) \cap B_{t-1}(v)$. In addition, we know that $|C_i| \leq r < t-1$ and that $G[C_i]$ is connected, so $C_i \subseteq B_{t-1}$, implying that $C_i \cap B_{t-1}(v) = C_i$, and therefore, $\tilde{C}(v) = C_i$. ◀

▶ **Corollary 24.** *Condition (V-1) is satisfied at all nodes.*

Proof. For each node v, Lemma 23 implies that $\tilde{C}(v) = C_i$ for some part C_i in the partition. By the conditions of Lemma 14, $|C_i| \leq r$ and $G[C_i]$ is connected, as required. ◀

▶ **Lemma 25.** *Condition (V-2) is satisfied at all nodes.*

Proof. Fix $v \in V$, and let $u \in N(v)$ be such that $cid_u \neq cid_v$. By Lemma 23, there are two distinct parts $C_i \neq C_j$ such that $u \in C_i$ and $v \in C_j$. Thus, from condition (R-2) of Lemma 14, either $u \in X_i$ or $v \in X_j$ (or both). Since the prover marks all border nodes and also their neighbors, we therefore have $s_u = s_v = 1$. ◀

Finally, we show that each boundary node has enough certificate-pieces in its vicinity:

Lemma 21, for the current construction. As in the proof of Lemma 21 for $\mathbf{Prv}^t_{sep}$, for any $v \in V$ and $x \in \mathit{Boundary}(v)$, we must have $s_x = 1$. (This part of the proof depends only on the verifier, which is the same in both proof systems.) Thus, there is some i such that $x \in Y_i$, and the prover distributes pieces of x's proof across the nodes in D_i. Recall that $|D_i| = d$. Moreover, $D_i \subseteq C_i \subseteq B_r(x)$, and $x \in B_{r+1}(v)$ (because $x \in \mathit{Boundary}(v) \subseteq \tilde{C}(v) \cup N(\tilde{C}(v))$ and $\tilde{C}(v)$ is a connected subgraph comprising at most r nodes). Thus, $D_i \subseteq B_{2r+1}(x) \subseteq B_{t-1}(x)$, by choice of $r = \lfloor t/2 \rfloor - 1$. This proves that node v indeed sees at least d distinct pieces of x's proof in $A_v(x)$, and is able to successfully reconstruct $\tilde{a}_v(x) = a_x$. ◀

This suffices to prove completeness of $\mathbf{Prv}^t_{minor}$, similar to that of $\mathbf{Prv}^t_{sep}$ in Appendix B.