An Alternate Proof of $\oplus$L-hardness of Graph Isomorphism

Shiva Kintali*

June 27th 2026

Abstract

Torán proved that graph isomorphism is hard for several small-space counting classes, including $\mathrm{Mod}_k\,\mathrm{L}$ for every $k \geq 2$, and hence for parity logspace. We give an alternate proof of the parity case:

$$\oplus\mathrm{L} \leq_m^{\mathrm{L}} \mathrm{GI}.$$

Our reduction starts from the parity of the number of paths in a layered directed acyclic graph, encodes this parity question as solvability of a linear system over $\mathbb{F}_2$, and realizes the linear system as a pair of CFI-type XOR graphs. A color-preserving isomorphism of the resulting colored graphs exists exactly when the linear system has a solution. A rigid color-removal construction then yields simple undirected graphs. The proof isolates the elementary mod-2 linear algebra behind the reduction and gives explicit size and uniformity arguments. The reduction is many-one: it outputs one graph pair and makes no graph-isomorphism oracle calls.

Keywords. graph isomorphism, parity logspace, logspace reductions, CFI graphs, linear equations over finite fields.

1 Introduction

The graph isomorphism problem (GI) asks whether two graphs are the same up to a relabeling of vertices. In this paper, unless a variant is explicitly named, graph isomorphism means isomorphism of finite simple undirected graphs. It is a central problem in complexity theory: it is in NP, not known to be in polynomial time, and not known to be NP-complete; see the monograph of Köbler, Schöning, and Torán [22] and the surveys of Grohe [11] and Grohe and Neuen [12]. The problem is important both as a basic test of structural equivalence and as a benchmark for the boundary between efficient computation and NP-completeness. It appears in canonical labeling, chemical database search, graph matching, finite model theory, and the study of symmetry in combinatorial structures.

For several important special graph classes, polynomial-time and even small-space algorithms are known. Tree isomorphism admits deterministic logspace canonization by Lindell [23]. Planar graph isomorphism has a classical linear-time algorithm due to Hopcroft and Wong [16], and later work placed planar graph isomorphism in logspace [8]. Luks's group-theoretic method gives polynomial-time algorithms for graphs of bounded valence [25]; more broadly, it is one of the foundations of the best general-purpose worst-case upper bounds for GI. Babai's breakthrough quasipolynomial-time algorithm places general graph isomorphism in $\exp((\log n)^{O(1)})$ time [3], improving the previous subexponential group-theoretic bounds of Babai and Luks [4] and reshaping the known upper-bound

*Email: shiva.kintali@gmail.com Homepage: shivakintali.github.io

picture. Babai's theorem also treats closely related permutation-group problems, including string isomorphism under group action and coset intersection [3].

There is also a large algorithmic literature for special graph classes. Bounded-treewidth graph isomorphism is polynomial-time solvable, with later restricted-space and fixed-parameter canonization algorithms [5, 9, 24]. For bounded-genus graphs, Kawarabayashi gave a linear-time isomorphism algorithm for each fixed Euler genus [18]. Grohe and Schweitzer proved polynomial-time isomorphism testing for graphs of bounded rank width, and hence for classes of bounded clique width [13]. These results show that many structural restrictions make GI significantly easier even though the unrestricted problem remains open.

Colored, directed, hypergraph, multigraph, and finite-relational-structure isomorphism over fixed finite signatures are standard GI-complete variants under polynomial-time many-one reductions, because the extra structure can be encoded by graph gadgets [34, 22]. GI remains complete for many restricted graph families, including connected, bipartite, regular, line, split, and chordal graphs [34, 22]. In contrast, nearby problems that ask for more than ordinary isomorphism can become NP-complete: subgraph isomorphism is NP-complete [15], and Lubiw proved NP-completeness for several constrained isomorphism and automorphism variants [26]. Separately, the CFI construction gives lower bounds for Weisfeiler–Leman refinement and finite-variable counting logics [7], and CFI-type instances also yield sum-of-squares/Lasserre lower bounds for isomorphism relaxations [28, 31].

Graph isomorphism is also part of a wider family of algebraic isomorphism and orbit problems. Finite group isomorphism, when groups are given by their multiplication tables, can be viewed as isomorphism of finite structures with one ternary multiplication relation and therefore reduces to graph isomorphism by the standard encoding of finite structures as graphs; this is a one-way comparison and is not a known equivalence [22]. General finite-group isomorphism has a classical generator-enumeration bound of $n^{O(\log n)}$ for groups of order n, and Rosenbaum gave improved algorithms for solvable groups and p-groups [29]. Other algebraic isomorphism problems, including isomorphism of tensors, certain p-groups of class two and exponent p, cubic forms, and finite-dimensional algebras, are tied together by polynomial-time reductions in the Tensor Isomorphism framework of Grochow and Qiao [14]. These related problems provide useful context for our proof: our reduction in this paper is algebraic in spirit, but its target is still undirected graph isomorphism.

GI also satisfies strong structural upper bounds in counting and interactive-proof complexity. Graph non-isomorphism has an Arthur–Merlin protocol, and hence GI lies in coAM [22]; this is one reason GI is considered unlikely to be NP-complete under ordinary reductions unless the polynomial hierarchy collapses. On the counting side, GI is low for PP [21] and is contained in SPP by the theorem of Arvind and Kurur [1]. These upper-bound and lowness results show that GI does not behave like an arbitrary NP problem.

Lower bounds for GI are necessarily subtler. Since GI is not known to be NP-hard and is believed not to be NP-complete, one seeks lower bounds inside small complexity classes. A familiar small-space reference point is tree isomorphism, which is decidable in deterministic logspace and is a special case of graph isomorphism. Under ordinary logspace many-one reductions, however, lower bounds from L itself are not very informative. Indeed, if a target language has at least one fixed yes-instance and one fixed no-instance, then every language in L reduces to that target by first deciding the source language in logspace and then outputting the appropriate fixed instance.

The strongest small-space hardness results for ordinary GI in this direction are due to Torán [33]. Torán proved that GI is hard under logarithmic-space many-one reductions for SL, for $\mathrm{Mod}_k\,\mathrm{L}$ for every $k \geq 2$, for NL, for $\mathrm{C}*=\mathrm{L}$, for PL, and for DET. He also showed that bits of #L functions reduce to GI, and derived randomized logspace reductions from perfect matching to GI. In particular, the case $k = 2$ of his modular theorem already gives $\oplus\mathrm{L} \leq_m^{\mathrm{L}} \mathrm{GI}$. Thus the present paper is not the

first proof that GI is parity-logspace hard. Its purpose is to give a direct, self-contained alternate proof of this parity case: $\oplus\mathrm{L} \leq_m^{\mathrm{L}} \mathrm{GI}$.

The notation $\leq_m^{\mathrm{L}}$ means deterministic logspace many-one reducibility. For each language L in $\oplus$L, there is a deterministic machine using only $O(\log n)$ work space that maps an input x to a pair of graphs (G_x, H_x) such that

$$x \in L \quad \iff \quad G_x \cong H_x.$$

The output graphs may have polynomial size, but the reduction is not allowed to store them while constructing them. Instead, it must be logspace-uniform: vertex names, equations, gadgets, and adjacencies are given by local rules that can be recomputed with logarithmic counters. Equivalently, to output an adjacency matrix, the transducer loops over pairs of output vertices and decides each adjacency bit using only logarithmic working memory. This is meaningful because $\oplus$L captures the parity of the number of accepting paths of nondeterministic logspace computations, and it is not known whether $\oplus\mathrm{L} = \mathrm{L}$. Our proof identifies a particularly transparent algebraic source of hardness for GI at the parity-logspace level.

The proof is constructive. Starting with a language in $\oplus$L, we take a nondeterministic logspace machine whose accepting paths encode membership modulo 2. Its computation is represented as a layered directed acyclic graph. The parity of paths from the initial configuration to the accepting configuration is then written as a triangular linear system over $\mathbb{F}_2$. A generalized CFI-style gadget turns the solvability of this system into a colored graph isomorphism question. The last step replaces colors by rigid uncolored graph structure, producing simple undirected graphs. The CFI construction of Cai, Fürer, and Immerman [7] is a standard source of parity gadgets in graph isomorphism and finite model theory. Our reductions isolate the elementary XOR mechanism needed here. Related completeness results for linear congruences modulo a constant are discussed by de Beaudrap [10]; the present proof instead gives the needed $\mathbb{F}_2$-linear system directly from a layered parity-path instance.

Our proof differs from Torán's approach in both scope and mechanism. Torán works uniformly over all moduli $k \geq 2$ by building graph gadgets that simulate addition gates over $\mathbb{Z}_k$ and then reducing modular circuit value problems to restricted automorphism questions, which are finally reduced to GI. Our proof specializes to the mod-2 case and replaces modular arithmetic circuits by the triangular path-parity equations that arise directly from a layered configuration graph. The isomorphism gadgets are then described as colored XOR graphs for bounded-width linear systems over $\mathbb{F}_2$, followed by an explicit rigid color-removal construction. The gain is not a stronger class lower bound than Torán's; the gain is a shorter, linear-algebraic route to the parity case, with all intermediate matrices and graphs made explicit and locally uniform. The organizing feature of the present proof is the separation of the parity lower bound into four elementary layers. First, path parity is encoded by a triangular dynamic-programming system, so the linear system has a unique intended solution up to the final consistency equation. Second, the width-three expansion is a transparent XOR-chain construction with explicit polynomial-size bounds. Third, the graph gadget is stated as a general XOR-isomorphism lemma for arbitrary bounded-width binary systems, rather than as a gate-by-gate simulation of modular circuits. Fourth, colors are removed by a rigid anchor construction whose correctness is independent of the algebraic gadget. This modular organization makes the parity mechanism visible and gives a local-output proof in which the polynomial-size and logspace-uniformity requirements can be checked at each step.

Novelty and intended contribution. The intended contribution is structural rather than a stronger complexity-theoretic lower bound. Torán's theorem already gives the parity-logspace lower

bound as part of a broader collection of small-space hardness results. The point of the present proof is to give a direct parity-only derivation whose intermediate objects are elementary: layered path-counting instances, triangular linear systems over $\mathbb{F}_2$, bounded-width XOR systems, colored XOR graphs, and a rigid uncolored color-removal gadget. This makes the proof independently checkable without invoking the full modular-circuit and restricted-automorphism framework used in Torán's construction. Its main value is to isolate the reusable XOR-isomorphism mechanism behind the parity lower bound in the simplest possible setting.

Why this proof may be useful. Our reduction isolates the exact mod-2 linear algebra that graph isomorphism must encode for the parity lower bound. This may make the argument easier to teach, easier to audit, and easier to reuse as a gadget template: the XOR-isomorphism lemma applies to arbitrary bounded-width binary linear systems, and the color-removal lemma is independent of the algebraic origin of those systems. Thus the proof also serves as a modular worked example of how local linear constraints can be represented by graph isomorphism instances.

2 Preliminaries

Definition 2.1 (Parity logspace)**.** *A language L is in $\oplus$L if there is a polynomially time-bounded nondeterministic logspace Turing machine M such that, for every input x,*

$$x \in L \quad \Longleftrightarrow \quad \#\,\mathrm{acc}_M(x) \equiv 1 \pmod 2,$$

where $\#\,\mathrm{acc}_M(x)$ is the number of accepting computation branches of M on x.

Informally, $\oplus$L is the mod-2 analogue of nondeterministic logspace. A nondeterministic machine may have many computation branches. Instead of asking whether at least one branch accepts, as in ordinary nondeterminism, $\oplus$L asks whether the number of accepting branches is odd. Since only the parity matters, two accepting branches can cancel each other modulo 2. The polynomial time bound in the definition ensures that the number of computation paths is finite and at most $c^{p(n)}$ for some constant branching bound c and polynomial time bound p, so its parity is well-defined. The constant c comes from the usual finite-control Turing-machine model: there are only finitely many transition rules, so each configuration has only constantly many possible next moves. A computation path is a sequence of nondeterministic transition choices, counted with multiplicity: if two different choices lead through the same sequence of configurations, they are still two different paths. Only accepting branches are counted in $\#\,\mathrm{acc}_M(x)$; rejecting branches and branches that enter rejecting final behavior contribute 0 to this number. The normalization in Section 3 will make this convention explicit by forcing every branch to reach a unique accepting or rejecting final configuration at a common time.

Here and throughout the paper, L denotes deterministic logspace.

Definition 2.2 (Logspace many-one reduction)**.** *For languages A and B, we write $A \le_m^{\mathrm{L}} B$ if there is a deterministic logspace transducer computing a function f such that $x \in A \iff f(x) \in B$. The output tape is write-only, is not counted toward the workspace bound, and is not available as scratch memory.*

In this paper, the output of every reduction is an explicit polynomial-size instance of the target problem. For a reduction to GI, this means a single encoded pair of undirected graphs. The word *many-one* is important: the reduction produces one target instance whose yes/no answer is the

desired answer. It does not make queries to a graph-isomorphism oracle and it does not adaptively inspect answers to intermediate instances.

The output of a logspace transducer may be much longer than its work tape. The restriction is that, while writing the output from left to right, the transducer may store only $O(\log n)$ bits of temporary information, where n is the input length. In reductions to graph isomorphism, this usually means that the transducer loops over pairs of output vertices and recomputes whether the corresponding edge should be present. For a reduction to GI, the string $f(x)$ is an encoding of a pair of graphs. This is the standard logspace transducer model used for many-one reductions; see, for example, [32, 2, 22].

We will use this model in the following concrete form. Each constructed matrix or graph comes with a canonical polynomially bounded index set of output positions: row and column indices for matrices, and vertex or vertex-pair indices for graphs. A transducer is allowed to enumerate this index set using logarithmic counters. When it reaches one output position, it recomputes the corresponding local datum from the original input, writes the bit or symbol for that position, and then discards the temporary information. The transducer is not allowed to build the whole configuration graph, linear system, colored graph, or final graph in its work tape before writing the output. Thus the uniformity arguments below always prove two facts: the number of output positions is polynomial, and each requested local output symbol is computable with logarithmic workspace.

Definition 2.3 (Graph isomorphism)**.** *The graph isomorphism problem* GI *has as input two simple undirected graphs G and H. It asks whether there exists a bijection*

$$\varphi : V(G) \to V(H)$$

such that, for all distinct vertices $u, v \in V(G)$,

$$\{u, v\} \in E(G) \quad \iff \quad \{\varphi(u), \varphi(v)\} \in E(H).$$

Here *simple'* means that there are no loops and no parallel edges, so an edge is an unordered pair $\{u, v\}$ with $u \neq v$.

Definition 2.4 (Colored graph isomorphism)**.** *A colored graph is a pair (G, γ), where G is a simple undirected graph and $\gamma : V(G) \to \mathcal{C}$ is a color map to a finite palette of color names. An isomorphism from (G, γ) to (H, η), with both color maps using the same palette $\mathcal{C}$, is a graph isomorphism $\varphi : V(G) \to V(H)$ such that*

$$\eta(\varphi(v)) = \gamma(v) \qquad \text{for every } v \in V(G).$$

We write

$$(G, \gamma) \cong *\mathrm{col}(H, \eta)$$

when the two colored graphs are color-preservingly isomorphic. When the color maps are fixed by context, we also write $G \cong *\mathrm{col}H$.

Colors are a temporary bookkeeping device. The color names are part of the intermediate encoded instance, and they are preserved literally; the isomorphism is not allowed to permute the palette. Equivalently, the color name attached to a vertex is part of the data that the isomorphism must preserve. In the reduction below, colors first keep the variables and equations from being renamed. Section 8 then replaces these colors by rigid uncolored graph structure, so the final

problem is still graph isomorphism. Whenever an ordered palette is mentioned later, the order is part of this fixed naming convention; it is not additional freedom for an isomorphism.

All linear algebra below is over the field $\mathbb{F}_2 = \{0, 1\}$, where addition is XOR. Also, subtraction is the same operation as addition over $\mathbb{F}_2$.

Definition 2.5 (Support and width of a binary equation)**.** *Let $a_1 z_1 + \cdots + a_n z_n = \beta$ be a linear equation over $\mathbb{F}_2$, where each $a_j, \beta \in \mathbb{F}_2$. The support of the equation is $\operatorname{supp}(a) = \{j : a_j = 1\}$. The width of the equation is $|\operatorname{supp}(a)|$, the number of variables that actually occur in it. A row of a binary matrix has the same support and width as the corresponding equation. If the support is empty, the left side is the empty sum, which is 0 in $\mathbb{F}_2$.*

Definition 2.6 (Satisfying assignment)**.** *For an $m \times n$ matrix A over $\mathbb{F}_2$ and a vector $c \in \mathbb{F}_2^m$, an assignment is a vector $z = (z_1, \ldots, z_n) \in \mathbb{F}_2^n$. It satisfies $Az = c$ if, for every row r,*

$$\sum_{j : A_{rj} = 1} z_j = c_r$$

in $\mathbb{F}_2$, with the empty sum interpreted as 0. Thus each row asks whether the XOR of a specified set of variables is the prescribed bit c_r.

Encoding convention. Ordinary graphs are encoded by adjacency matrices unless explicitly stated otherwise; the number of vertices is written as part of the encoding. A pair of graphs is encoded by writing the two graph encodings in a fixed order. When colored graphs appear as intermediate objects, the adjacency matrix is accompanied by a list of color names, or equivalently color indices from a canonical finite palette, one for each vertex. Section 7 spells out the exact local encoding used for the constructed colored graphs. For the polynomial-size graph families constructed in this paper, the standard explicit encodings are logspace interconvertible, and the adjacency-matrix convention makes the local nature of the reductions transparent. A logspace transducer is allowed a read-only input tape, $O(\log n)$ work space, and a write-only output tape. Thus it may produce polynomial-size output by recomputing local predicates while looping over polynomially many output positions.

It is useful to distinguish two notions of size. The output graph, or graph pair, may have polynomially many vertices, and therefore its adjacency-matrix encoding may contain polynomially many bits. The work tape, however, may store only logarithmically many bits at any one time. This is why all later constructions are described by local adjacency tests: to output the graph, the transducer loops through vertex pairs, recomputes whether that pair should be adjacent, writes one bit, and forgets the local calculation.

Notation table. The reduction uses several layers of indices. The following table records the main symbols and where they first become relevant.

Symbol	Meaning
$D = (V_0, \dots, V_T, E)$	layered directed acyclic graph with start s and target t
$x_{i,v}$	path-parity variable for vertex $v \in V_i$
$A_0 x = c_0$	dynamic-programming linear system for path parity
r	original row of $A_0 x = c_0$ before width reduction
$y_{r,h}$	auxiliary partial-XOR variable in the chain replacing row r
R	final row identifier after width reduction, either (U, r) or (C, r, h)
$Az = c$	final width-three system, including all auxiliary variables
S_R	support of final row R in A
$\mathrm{Var}_j, \mathrm{Eq}_R$	colors naming variable j and final row R
$p_{j,b}$	variable vertex representing bit b of variable j
$q_{R,a}$	equation vertex for a locally satisfying assignment a of row R
$X(A, \rho)$	colored XOR graph for right-hand side ρ
$R(Y)$	ordinary uncolored graph obtained from a colored graph Y by rigid color removal

The letters r and R are deliberately different: r refers to a row before width reduction, while R refers to a row of the final width-three system. This distinction matters because one original row may give rise to several final chain rows. The notation $R(Y)$ is separate: it denotes the color-removal construction applied to a colored graph Y, not a row identifier.

3 A Complete Problem for $\oplus$L

The first step is to replace machines by graphs. A configuration of a space-bounded machine records everything needed to continue the computation: the current state, the positions of the tape heads, and the contents of the work tape. If we draw an edge from a configuration to each possible next configuration, then a computation branch becomes a path in this configuration graph. To avoid cycles and to make path counting finite, we put a clock on the machine and place configurations at successive time layers.

Definition 3.1 (Layered parity s-t path)**.** *The problem* LAYERED-PARITY-ST-PATH *is defined as follows. The input is a simple layered directed acyclic graph $D = (V_0 \cup V_1 \cup \cdots \cup V_T, E)$, a source vertex $s \in V_0$, and a target vertex $t \in V_T$. The layer partition $V_0, \dots, V_T$ is part of the input. These layers are pairwise disjoint, and every edge goes from V_i to V_{i+1} for some $i < T$. The question is whether the number of directed paths from s to t is odd.*

We use any standard explicit encoding of such an instance: the layer partition, the distinguished vertices s, t, and the adjacency relation are written as part of the input. The reductions below use only local queries to this encoding, and standard explicit encodings of layered graphs are interconvertible in deterministic logspace.

Here "simple" means that for any ordered pair of vertices there is at most one directed edge. This convention loses no generality for the reductions in this paper. If a construction ever produces parallel directed edges between two consecutive layers, subdivide every edge between those two layers by inserting one new intermediate vertex per edge-copy in a fresh layer. A path through the new layer chooses exactly one original edge-copy, so the number of s-t paths with multiplicity is preserved. This edge-subdivision is computable in logspace. The choice vertices used in the configuration-graph construction below are exactly this multiplicity-preserving idea.

A path in such a graph is a sequence $s = v_0 \to v_1 \to \cdots \to v_T = t$ with $v_i \in V_i$. Because edges only go from one layer to the next, there are no directed cycles and every s-t path has the same length.

Theorem 3.2. LAYERED-PARITY-ST-PATH *is* $\oplus$L*-complete under deterministic logspace many-one reductions.*

The proof has the standard two directions. For membership, a nondeterministic logspace machine guesses a path one layer at a time. For hardness, the accepting branches of an arbitrary nondeterministic logspace computation are encoded as paths in a layered computation graph.

4 Encoding Path Parity as Linear Equations

Fix an instance (D, s, t) of layered parity path, with layers $V_0, \ldots, V_T$. All arithmetic in this section is over $\mathbb{F}_2$. Thus addition is XOR, subtraction is the same as addition, and an empty sum is 0.

The next construction is dynamic programming, written as linear algebra over $\mathbb{F}_2$. For each vertex v in layer i, we introduce a variable whose intended value is the parity of the number of paths from s to v. At layer 0, this parity is known: there is one empty path from s to itself and no path from s to any other vertex in V_0. At later layers, every path to a vertex w must first reach one of the predecessors of w, so the parity of paths to w is the XOR of the parities of paths to its predecessors.

This is the same recurrence one would use to count paths in an acyclic graph, except that all arithmetic is done modulo 2. If two predecessors contribute odd numbers of paths, their contributions add to 0 in $\mathbb{F}_2$ because odd plus odd is even. More generally, the result is 1 exactly when an odd number of predecessors contribute odd path counts. Thus the linear system does not need to know the actual path counts, which may be exponentially large; it only stores one bit of information for each vertex.

For every vertex $v \in V_i$, introduce a variable $x_{i,v}$. The intended value is $x_{i,v} = \#\,\mathrm{Paths}_D(s, v) \bmod 2$. We impose the following equations over $\mathbb{F}_2$.

Initial equations. $x_{0,s} = 1$, and for every $v \in V_0$ with $v \neq s$, $x_{0,v} = 0$.

Transition equations. For every $i < T$ and every $w \in V_{i+1}$,

$$x_{i+1,w} + \sum_{u \to w} x_{i,u} = 0.$$

The sum ranges over predecessors $u \in V_i$ with an edge $u \to w$. If w has no predecessors, the sum is empty and the equation is simply $x_{i+1,w} = 0$, as it should be. Because the input graph is simple, each predecessor contributes at most one edge to w, so no predecessor variable is repeated in this sum. Every transition equation has nonempty support because it contains the new-layer variable $x_{i+1,w}$, even when w has no predecessors.

For example, if $w \in V_{i+1}$ has exactly three predecessors $u_1, u_2, u_3 \in V_i$, then the transition equation is

$$x_{i+1,w} + x_{i,u_1} + x_{i,u_2} + x_{i,u_3} = 0.$$

Equivalently,

$$x_{i+1,w} = x_{i,u_1} + x_{i,u_2} + x_{i,u_3}$$

over $\mathbb{F}_2$. Thus an even number of odd predecessor parities gives $x_{i+1,w} = 0$, while an odd number gives $x_{i+1,w} = 1$.

Target equation. $x_{T,t} = 1$.

Let the resulting system be $A_0 x = c_0$. The key point is that these equations do not merely constrain the path parities; they determine them uniquely. The variables in layer 0 are fixed by the initial equations. Then the variables in layer 1 are forced by the transition equations from layer 0 to layer 1, the variables in layer 2 are forced by the next transition equations, and so on. This is why the final equation $x_{T,t} = 1$ is the only possible place where inconsistency can occur.

Lemma 4.1. *The system $A_0 x = c_0$ is satisfiable if and only if the number of directed paths from s to t in D is odd.*

Proof. The system is triangular by layers: if the variables are ordered by their layer number, every transition equation for layer $i+1$ contains exactly one variable from layer $i+1$ and only variables from the previous layer i besides it. The initial equations uniquely force all variables in layer 0. Once all variables in layer i are fixed, every variable in layer $i+1$ is uniquely forced by

$$x_{i+1,w} = \sum_{u \to w} x_{i,u}.$$

By induction on i, the unique values forced by the initial and transition equations are exactly

$$x_{i,v} = \#\,\mathrm{Paths}_D(s, v) \bmod 2.$$

The induction is a dynamic-programming argument. At $i = 0$, there is one length-zero path from s to itself and no length-zero path from s to any other vertex. For the induction step, every path ending at $w \in V_{i+1}$ is obtained uniquely by taking a path to some predecessor $u \in V_i$ and then using the edge $u \to w$. Therefore the number of paths to w, modulo 2, is the XOR of the parities at all predecessors. Thus the only possible inconsistency is the final equation $x_{T,t} = 1$. It is satisfied exactly when the path parity from s to t is odd.

This statement also covers the degenerate case $T = 0$. Then the transition equations are absent. If $s = t$, the initial equation and the target equation both require $x_{0,s} = 1$, matching the single empty path from s to itself. If $s \neq t$, the initial equations force $x_{0,t} = 0$ while the target equation requires $x_{0,t} = 1$, matching the fact that there is no length-zero path from s to t.

This also shows why the system is not merely a necessary condition but an exact encoding. There is no freedom to choose the variables in a way that avoids the path-counting interpretation: the equations determine them layer by layer. □

4.1 Bounded-Width Expansion

The graph gadget in the next section is polynomial-size when every row has bounded support. If an equation has large support, replace it by a chain of equations of width at most 3.

The bounded-support condition is important because an equation of positive width d has 2^{d-1} locally satisfying assignments. The XOR graph below creates one equation vertex for each such local assignment. If d were allowed to be large, this could create exponentially many vertices from a single row. By reducing every equation to width at most 3, we ensure that each nonempty row contributes at most four equation vertices.

This bounded-width conversion is a standard trick. A long XOR $t_1 + t_2 + \cdots + t_\ell$ is evaluated by remembering partial sums. The auxiliary variable y_h represents the partial XOR $t_1 + \cdots + t_h$.

Before applying the conversion, fix a deterministic order of the variables appearing in the row. For an initial or target row there is only one variable. For a transition row, we put $x_{i+1,w}$ first and then list the predecessor variables $x_{i,u}$ in the order in which the vertices of V_i are encoded. The particular order does not affect satisfiability; it only makes the transformation canonical.

Consider an equation $t_1 + t_2 + \cdots + t_\ell = \beta$, where each t_i is a variable and $\beta \in \mathbb{F}_2$. Repeated variables, if present, are first cancelled in pairs over $\mathbb{F}_2$; the path-parity system above already has distinct variables in each equation. If cancellation ever produced an empty equation, then $0 = 0$ would simply be deleted and $0 = 1$ would make the whole system unsatisfiable. This exceptional case does not arise in the path-parity systems constructed here.

If $1 \le \ell \le 3$, keep the equation unchanged. This is the convention used throughout the rest of the paper. If $\ell \ge 4$, introduce auxiliary variables

$$y_2, y_3, \ldots, y_{\ell-1}$$

and replace the equation by

$$y_2 + t_1 + t_2 = 0,$$

$$y_h + y_{h-1} + t_h = 0 \qquad (3 \le h \le \ell - 1),$$

and

$$y_{\ell-1} + t_\ell = \beta.$$

There are $\ell - 2$ new auxiliary variables and $\ell - 1$ new equations. Every new equation has width 2 or 3. Thus, after the conversion, every row has support size at most 3.

Auxiliary variables introduced for different original rows are distinct. The notation y_h above describes the chain for one row only. In the full system, the same auxiliary variable is named by the pair consisting of the original row identifier and the index h, often written $y_{r,h}$. This convention is useful for uniformity: a logspace transducer can output such a name using only the current row index and the current chain position.

Lemma 4.2. *The bounded-width expansion preserves satisfiability.*

Proof. Rows that were not replaced are unchanged, so there is nothing to prove for them. Consider a row that was replaced by a chain. Given values satisfying the original equation, set

$$y_h = t_1 + \cdots + t_h$$

for each auxiliary variable y_h with $2 \le h \le \ell - 1$. Then all chain equations hold. For example,

$$y_2 + t_1 + t_2 = (t_1 + t_2) + t_1 + t_2 = 0,$$

and, for $3 \le h \le \ell - 1$,

$$y_h + y_{h-1} + t_h(t_1 + \cdots + t_h) + (t_1 + \cdots + t_{h-1}) + t_h = 0.$$

The last chain equation holds because

$$y_{\ell-1} + t_\ell t_1 + \cdots + t_\ell \beta.$$

Conversely, summing all chain equations cancels every auxiliary variable twice, hence yields

$$t_1 + \cdots + t_\ell = \beta.$$

Thus any solution of the chain restricts to a solution of the original equation.

This proves equivalence for one replaced row while keeping all other variables fixed. Since different rows receive disjoint auxiliary variables, the same argument applies independently to every

replaced row in the system. Therefore replacing all long rows preserves satisfiability of the entire system.

For instance, the two occurrences of y_2 in the first two chain equations add to 0 over $\mathbb{F}_2$, and the same cancellation happens for every other auxiliary variable. This is why summing the chain recovers exactly the original long equation.

There is no hidden choice in the auxiliary variables once the original variables are fixed. The first chain equation forces y_2, the second forces y_3, and so on. Conversely, if the chain is satisfied, the last equation says that the final partial sum has the required value β. Hence the chain is exactly a width-three implementation of the original XOR equation. □

Let $Az = c$ denote the final width-3 system. We have proved: $Az = c$ is satisfiable $\iff$ $\#\,\mathrm{Paths}_D(s,t) \equiv 1 \pmod 2$.

Proposition 4.3 (From path parity to bounded-width linear equations)**.** *There is a deterministic logspace transformation that maps a layered parity-path instance (D, s, t) to a binary linear system $Az = c$ such that every row of A has nonempty support of size at most 3 and*

$$Az = c \text{ is satisfiable} \quad \iff \quad \#\,\mathrm{Paths}_D(s,t) \equiv 1 \pmod 2.$$

Proof. The transformation first writes the dynamic-programming system $A_0 x = c_0$ with one variable $x_{i,v}$ for each layered vertex. By Lemma 4.1, that system is satisfiable exactly in the odd-path case. It then replaces every row of support larger than 3 by the XOR chain described above. Lemma 4.2 shows that this replacement preserves satisfiability. The resulting system is $Az = c$, and all of its rows have nonempty support of size at most 3: initial and target rows have one variable, every unchanged transition row has the variable $x_{i+1,w}$, and every chain row has two or three variables.

The construction is logspace-computable because the rows are local. To write an initial or target equation, the transducer only needs the index of the relevant vertex. To write a transition equation for $w \in V_{i+1}$, it scans the input and enumerates the row support: first the variable $x_{i+1,w}$ and then the predecessors $x_{i,u}$ in fixed vertex order. If that support list is long, the transducer writes the corresponding XOR chain using logarithmic counters for the row identifier and the position in the support list. For example, to write the chain equation involving the hth support term, the transducer can rescan the vertices of the previous layer in the fixed order, count predecessor hits up to h, and recover that support term using only logarithmic counters. It never needs to store the whole support list at once.

The auxiliary variables can be named by pairs (row, h), or, if a standard contiguous indexing of variables is desired, by the rank of this pair in the lexicographic order of all original variables and all auxiliary pairs. That rank can be computed by repeated scans and logarithmic counters because the number of rows, vertices, and support positions is polynomial in the input size. Thus the output is a deterministic logspace many-one reduction in the usual matrix encoding. At no point does the transducer need to store the entire matrix.

The output size is polynomial. The dynamic-programming system has one variable for each layered vertex, one initial row for each vertex of V_0, one transition row for each vertex outside V_0, and one target row. Thus the number of original rows is $|V(D)| + 1$. Initial and target rows have width 1, so only transition rows can need expansion. A transition row for a vertex w has support length $1 + \mathrm{indeg}(w)$. Expanding it therefore adds only linearly many auxiliary variables and rows in $1 + \mathrm{indeg}(w)$. Summed over all transition rows, the indegrees are exactly the number of directed edges of D. Hence the final width-three system has polynomially many variables, rows, and nonzero matrix entries. □

5 XOR Graphs for Bounded-Width Linear Systems

Let A be an $m \times n$ binary matrix whose rows have nonempty support of size at most 3. The systems produced in Section 4 have this property: every original dynamic-programming row contains at least one variable, and every row introduced by the bounded-width expansion contains two or three variables. For row r, define

$$S_r = \{j : A_{rj} = 1\}.$$

Let $\rho \in \mathbb{F}_2^m$ be a right-hand side. All graphs in this section are finite simple undirected colored graphs. When we compare $X(A, \rho)$ with $X(A, \rho')$, we write primed symbols such as $p'_{j,b}$ and $q'_{r,a}$ for the corresponding vertices in the second graph.

We now build a colored graph that remembers the system $Az = \rho$. The idea is simple. A variable z_j is represented by a pair of vertices $p_{j,0}, p_{j,1}$. An isomorphism may either preserve this pair or swap it. We interpret "swap the pair for z_j" as adding 1 to the variable z_j. Thus an isomorphism determines a vector

$$y = (y_1, \ldots, y_n) \in \mathbb{F}_2^n,$$

where $y_j = 1$ means the jth variable pair is swapped.

This is the key mechanism of the reduction. The graph does not contain a vertex labeled by a global assignment $z \in \mathbb{F}_2^n$. Instead, a global assignment is encoded dynamically by the behavior of an isomorphism on the variable pairs. If the isomorphism swaps exactly the pairs with indices in a set S, then the corresponding vector has value 1 on S and value 0 elsewhere.

The equation gadgets enforce consistency. For a row r, we create one equation vertex for each assignment to the variables in that row that satisfies the row. The adjacency of that equation vertex records which bit was chosen for each variable in the row. If the variable pairs are swapped according to y, then a locally satisfying assignment for right-hand side ρ_r is carried to a locally satisfying assignment for right-hand side ρ'_r precisely when

$$\sum_{j \in S_r} y_j = \rho_r + \rho'_r.$$

This is the rth equation of $Ay = \rho + \rho'$.

Here is the dictionary to keep in mind:

linear-algebra object	graph-theoretic object
variable z_j	pair $\{p_{j,0}, p_{j,1}\}$
value $y_j = 0$	do not swap the pair
value $y_j = 1$	swap $p_{j,0}$ and $p_{j,1}$
row r	color class $\mathrm{Eq} * r$
local satisfying assignment a	equation vertex $q * r, a$

The colors are essential at this intermediate stage: the color Var_j forces the jth variable pair to map to the jth variable pair, and the color Eq_r forces the rth equation gadget to map to the rth equation gadget.

Without colors, an isomorphism might rename variables or equations in a way that has nothing to do with solving the original linear system. The colored graph should therefore be viewed as a controlled algebraic object: variables and rows are named, while the only intended freedom is whether each two-vertex variable pair is flipped.

Definition 5.1 (The colored XOR graph $X(A, \rho)$)**.** *The colored graph $X(A, \rho)$ has two kinds of vertices. For every variable z_j, create two vertices $p_{j,0}, p_{j,1}$, both of color Var_j. For every equation*

row r and every local assignment $a : S_r \to \mathbb{F}_2$ satisfying $\sum_{j \in S_r} a_j = \rho_r$, create one vertex $q_{r,a}$ of color Eq_r. Add edges $q_{r,a}p_{j,a_j}$ for every r, every satisfying local assignment a, and every $j \in S_r$. These are undirected edges, and no other edges are added. This rule creates no loops, since variable vertices and equation vertices are distinct, and no parallel edges, since a local assignment chooses a unique bit for each variable in its support.

Thus $X(A, \rho)$ is a bipartite colored graph: one side consists of the variable vertices $p_{j,b}$, and the other side consists of the equation vertices $q_{r,a}$. There are no edges between two variable vertices and no edges between two equation vertices. An equation vertex $q_{r,a}$ has exactly one neighbor in the pair $\{p_{j,0}, p_{j,1}\}$ for each $j \in S_r$, namely p_{j,a_j}, and it has no neighbors in variable pairs whose indices are outside S_r.

Since $1 \leq |S_r| \leq 3$, each row contributes at most $2^{|S_r|-1} \leq 4$ equation vertices. Hence $X(A, \rho)$ has polynomial size. For any two right-hand sides ρ and ρ', the color classes in $X(A, \rho)$ and $X(A, \rho')$ have the same sizes: each variable color has two vertices, and each equation color Eq_r has $2^{|S_r|-1}$ vertices. If a column of A is zero, then the corresponding variable pair is isolated except for its color; this is harmless, because the associated swap bit does not occur in any equation of $Ay = \rho + \rho'$.

The number $2^{|S_r|-1}$ comes from a basic parity fact. If a row contains d variables, then among the 2^d possible bit assignments, exactly half have XOR 0 and exactly half have XOR 1. Thus, for either right-hand side, there are 2^{d-1} locally satisfying assignments. Because $d \leq 3$, this number is at most four. For $d = 1$ there is one satisfying assignment, for $d = 2$ there are two, and for $d = 3$ there are four. One quick way to see the half-and-half statement is to toggle the first bit of an assignment: this gives a bijection between assignments of XOR 0 and assignments of XOR 1. Formally, a local assignment is a function $a : S_r \to \mathbb{F}_2$. When such an assignment is written below as a bit string, the bits are listed in the canonical increasing order of the support S_r.

Lemma 5.2 (XOR-isomorphism lemma)**.** *For any $\rho, \rho' \in \mathbb{F}_2^m$,*

$$X(A, \rho) \cong *\mathrm{col}X(A, \rho') \quad \Longleftrightarrow \quad \exists y \in \mathbb{F}_2^n \text{ such that } Ay = \rho + \rho'.$$

Proof. First suppose $Ay = \rho + \rho'$. Define a map

$$\varphi : X(A, \rho) \to X(A, \rho')$$

on variable vertices by

$$\varphi(p_{j,b}) = p' * j, b + y_j.$$

For an equation vertex $q_{r,a}$, define

$$\varphi(q * r, a) = q' * r, a + y| * S_r,$$

where

$$(a + y| * S_r)j = a_j + y_j.$$

All additions in subscripts and local assignments are in $\mathbb{F}_2$. This target equation vertex exists because

$$\sum j \in S_r(a_j + y_j) \sum *j \in S_r a_j + \sum_{j \in S_r} y_j \rho_r + (Ay) * r\rho_r + (\rho_r + \rho' r)\rho' r.$$

For each fixed row r, adding $y|_{S_r}$ is therefore a bijection from the local assignments satisfying right-hand side ρ_r to the local assignments satisfying right-hand side ρ'_r. Edges are preserved because

$$qr, apj, a_j \quad \mapsto \quad q'r, a + y|S_r p' * j, a_j + y_j.$$

The right-hand side is an edge of $X(A, \rho')$ by the definition of the XOR graph: the image equation vertex records the local assignment $a + y|_{S_r}$, and its neighbor in the jth variable pair is precisely p'_{j,a_j+y_j}. The map is bijective because adding the fixed vector y is its own inverse over $\mathbb{F}_2$: applying the same swaps twice returns every bit to its original value. The same formula, now viewed as a map from $X(A, \rho')$ back to $X(A, \rho)$, preserves edges by the same calculation because $\rho + \rho' = \rho' + \rho$. Hence adjacency is preserved in both directions. Since the graph has no edges except the incidence edges between equation vertices and their selected variable vertices, this proves preservation of the entire edge relation. The map preserves variable colors Var_j and equation colors Eq_r by construction, so it is a color-preserving isomorphism.

In words, the vector y tells us exactly which variable pairs to flip. After these flips, each equation vertex still records a satisfying local assignment, but now for the new right-hand side ρ'.

Conversely, suppose $\varphi : X(A, \rho) \to X(A, \rho')$ is a color-preserving isomorphism. Since the color class Var_j consists of exactly two vertices, φ either preserves or swaps that pair. Define $y_j \in \mathbb{F}_2$ by $\varphi(p_{j,0}) = p' * j, y_j$. Then necessarily $\varphi(p * j, b) = p'_{j,b+y_j}$ for both $b \in \mathbb{F}_2$.

Thus the behavior of φ on all variable vertices is completely summarized by the vector y. What remains to prove is that adjacency forces this same vector y to satisfy every row equation.

At this point the colors have already done half the work. They prevent φ from sending a vertex in the jth variable pair to a different variable pair, and they prevent an equation vertex for row r from moving to a different row. The only remaining question is whether the allowed flips of the variable pairs are compatible with the equation vertices. The adjacencies answer exactly that question.

Take any row r and choose an equation vertex $q_{r,a}$. Such a vertex exists for every row, because S_r is nonempty and exactly half of the local assignments satisfy any prescribed right-hand side. Since colors are preserved,

$$\varphi(q_{r,a}) = q' * r, b$$

for some local assignment $b : S_r \to \mathbb{F}_2$ satisfying

$$\sum * j \in S_r b_j = \rho' * r.$$

For each $j \in S_r$, the vertex $q_{r,a}$ is adjacent to p_{j,a_j}. Therefore $q'_{r,b}$ is adjacent to $p' * j, a_j + y_j$. But $q'_{r,b}$ has exactly one neighbor in the variable pair $\{p'_{j,0}, p'_{j,1}\}$, namely p'_{j,b_j}. Hence $b_j = a_j + y_j$ for all $j \in S_r$. Thus

$$\rho' * r \sum * j \in S_r b_j \sum_{j \in S_r} (a_j + y_j)\rho_r + (Ay)_r.$$

Therefore $(Ay)_r = \rho_r + \rho'_r$ for every row r, so $Ay = \rho + \rho'$.

This proves that no other kind of color-preserving isomorphism is possible. Once the behavior on the variable pairs is known, adjacency forces the behavior on every equation vertex. □

Taking $\rho = 0$ and $\rho' = c$ gives:

$$X(A, 0) \cong * \mathrm{col} X(A, c) \quad \Longleftrightarrow \quad Az = c \text{ is satisfiable.}$$

Indeed, the vector called y in the lemma is exactly a solution vector for the system $Az = c$.

Remark 5.3 (Relation to classical CFI)**.** *Classical CFI graphs arise when the variables are edges of a base graph and the equations are vertex-incidence parity equations. Then A is the vertex-edge incidence matrix over $\mathbb{F}_2$. The construction above is the same XOR principle with an arbitrary bounded-width binary matrix A, so it captures arbitrary bounded-width linear systems rather than only incidence systems.*

6 Reduction to Colored Graph Isomorphism

Given the layered path instance (D, s, t), construct the width-3 system $Az = c$ from Section 4. This is the final bounded-width system, after the possible introduction of auxiliary variables and XOR-chain rows. Let m be the number of rows and n the number of variables in this system. Every row of this system has nonempty support of size at most 3, so the XOR-graph construction of Section 5 applies. Then build

$$G_D^{\mathrm{col}} = X(A, 0), \qquad H_D^{\mathrm{col}} = X(A, c).$$

Here 0 means the zero vector in $\mathbb{F}_2^m$. The two graphs use the same named colors $\mathrm{Var}_1, \ldots, \mathrm{Var}_n$ and $\mathrm{Eq}_1, \ldots, \mathrm{Eq}_m$. The first graph uses the homogeneous right-hand side; the second uses the right-hand side c of the final bounded-width path-parity system. Thus a color-preserving isomorphism from G_D^{col} to H_D^{col} is forced to choose swaps of the variable pairs that transform the zero right-hand side into c. By Lemma 5.2, that is exactly the same as solving $Az = c$.

This is already a single colored graph-isomorphism instance: the output is the pair

$$\left(G_D^{\mathrm{col}}, H_D^{\mathrm{col}}\right),$$

not a list of separate row gadgets to be queried independently. The row gadgets all share the same variable-pair vertices, so a putative isomorphism must choose one global swap vector for all rows at once. This point is important for the many-one reduction: the transducer outputs one pair of colored graphs, and the decision problem asks one colored-isomorphism question about that pair. Colors are still part of the target instance at this stage. They will be removed in Section 8; the present section proves only the reduction to color-preserving graph isomorphism.

The use of $X(A, 0)$ on one side and $X(A, c)$ on the other is deliberate. Both graphs have the same variables, rows, and color palette. They differ only in which local assignments are allowed inside each equation gadget. A color-preserving isomorphism between them must therefore explain the change in right-hand side by flipping variable pairs. The XOR-isomorphism lemma says precisely that such flips exist exactly when the linear system has a solution.

The two colored graphs also have the same number of vertices in each color class. Each variable color Var_j contains two vertices on both sides. If row r has support size d, then the row color Eq_r contains 2^{d-1} vertices for either right-hand side. Thus the colored-isomorphism question is comparing two graphs with the same named color classes; the issue is not whether the color classes match, but whether their adjacencies can be matched by a consistent choice of variable-pair swaps.

Proposition 6.1 (From bounded-width linear systems to colored GI)**.** *For every layered parity-path instance (D, s, t), the colored graphs G_D^{col} and H_D^{col} are computable in deterministic logspace and satisfy*

$$G_D^{\mathrm{col}} \cong {*\mathrm{col}} H_D^{\mathrm{col}} \quad \Longleftrightarrow \quad \#\,\mathrm{Paths}_D(s, t) \equiv 1 \pmod 2.$$

Proof. By Lemma 5.2, applied with $\rho = 0$ and $\rho' = c$,

$$X(A, 0) \cong {*\mathrm{col}} X(A, c) \quad \Longleftrightarrow \quad \exists y \in \mathbb{F}_2^n \text{ such that } Ay = 0 + c = c.$$

Over $\mathbb{F}_2$, $0 + c = c$, so the right-hand condition is exactly satisfiability of the linear system $Az = c$; the name of the solution vector is immaterial. By Proposition 4.3, that satisfiability condition is equivalent to odd s-t path parity in D. Since $G_D^{\mathrm{col}} = X(A, 0)$ and $H_D^{\mathrm{col}} = X(A, c)$, the stated equivalence follows.

The colored graphs are logspace-computable because the system $Az = c$ is logspace-computable and every row has support at most 3. For each row, there are at most four satisfying local assignments,

so the transducer can enumerate the equation vertices and their incident variable vertices using logarithmic counters and constant-size local assignment strings. It never needs to store the whole system or either output graph. Concretely, when the transducer needs information about a row of $Az = c$, it recomputes that row from the layered input using the procedure of Proposition 4.3; it then tests the constant-size local assignments for right-hand side 0 or for the corresponding bit of c.

Equivalently, fix the canonical output order that lists the variable vertices (j, b) and then, for each final row R, the locally satisfying equation vertices (R, a). Given an output position, the transducer can decode whether it refers to a color entry or an adjacency entry. For a color entry, the vertex type and its index determine the color immediately. For an adjacency entry, the transducer recomputes the support S_R of the relevant final row, checks the constant-size assignment a, and writes 1 exactly when one endpoint is $q_{R,a}$, the other is p_{j,a_j}, and $j \in S_R$. All counters range over polynomially many rows, variables, and vertices, so they require only logarithmically many bits. This proves the logspace-computability part of the proposition. The next section spells out this indexing and adjacency predicate in more detail. □

7 Logspace Uniformity

We now explain why the map $(D, s, t) \longmapsto \big(X(A, 0), X(A, c)\big)$ is computable by a deterministic logspace transducer. The transducer never needs to store A, c, or the output graphs. It only needs to compute their local incidence information on demand. This is the main reason for using canonical indices for all variables, equations, and graph vertices. Given an index, the transducer can decode what object it refers to, compute the few neighboring objects it depends on, and write the appropriate adjacency bit.

There is one notational convention in this section. A row of the final width-three system may either be an unchanged row from the dynamic-programming system or one row of an XOR chain introduced during width reduction. Thus a final row identifier is itself a small tuple, for example (U, r) or (C, r, h) below. When the XOR graph construction uses a color Eq_R, the symbol R denotes such a final row identifier. This keeps it separate from the original row index r used before width reduction.

Concretely, think of the output as a standard colored-graph encoding: a vertex list with color labels, followed by an adjacency matrix. The color of a vertex is determined by its index: (P, j, b) has color Var_j, and (Q, R, α) has color Eq_R. To write the adjacency-matrix entry in row u and column v, the transducer performs the following local calculation.

(1) Decode the vertex indices u and v as either variable vertices or equation vertices.

(2) If they are of the same kind, write 0, since the XOR graph has no variable-variable or equation-equation edges.

(3) If one is a variable vertex (P, j, b) and the other is an equation vertex (Q, R, α), compute the at most three variables in the final row R and the right-hand side bit used by the graph currently being output.

(4) Interpret α as the first $|S_R| - 1$ bits of the unique local assignment whose parity is that right-hand side, and write 1 exactly when j is one of the variables in S_R and that local assignment gives the bit b to j.

Every counter used in these steps ranges over vertices, rows, or positions in a predecessor list, all of which have polynomial size. Hence each counter has only logarithmically many bits.

This is the standard way to think about logspace output. The final graph may contain many vertices, but a vertex name is only a tuple of small indices. To decide whether two named vertices are adjacent, the transducer recomputes the relevant row of the linear system from the original layered graph. Recomputing information is allowed; storing the entire intermediate matrix is not.

7.1 Indexing variables

Original path variables have the form $x_{i,v}$. They are indexed by the pair (i, v). Auxiliary variables introduced during width reduction have the form $y_{r,h}$, where r indexes an original equation and h indexes a position in its XOR chain. All such indices have $O(\log |D|)$ bits.

For example, the variable $x_{i,v}$ can be represented by the binary strings for i and for the rank of v within layer V_i. An auxiliary variable $y_{r,h}$ is represented by the rank of the original row r and the position h in its chain. Since there are only polynomially many rows and variables, all these ranks have logarithmic length.

We use these symbolic names to define a single canonical order of the final variables. First list the original variables $x_{i,v}$ in increasing layer order and, within each layer, in the fixed encoding order of the vertices. Then list the auxiliary variables $y_{r,h}$ in increasing order of the original row identifier r and, for each such row of support length $\ell \geq 4$, increasing $h \in \{2, \ldots, \ell - 1\}$. The column index j used in the XOR graph is the rank of the variable in this order. A logspace transducer can compute or compare these ranks by repeated scans with logarithmic counters.

7.2 Indexing equations

Original equations have the following forms:

$$x_{0,s} = 1,$$

$$x_{0,v} = 0 \quad (v \neq s),$$

$$x_{i+1,w} + \sum_{u \to w} x_{i,u} = 0,$$

and

$$x_{T,t} = 1.$$

After width reduction, a final output row identifier R is either an unchanged row (U, r) of support size at most 3, or a chain row (C, r, h) produced from an original row r whose support length is $\ell \geq 4$. The index h records which equation in the XOR chain is being output. A logspace machine can recover the at most three variables appearing in either kind of output row by scanning the input and counting terms in the canonical order. All components of R have logarithmic length: the original row index r ranges over polynomially many dynamic-programming rows, and a chain position h is at most the support length of that original row, which is also polynomially bounded.

The final row order is fixed similarly. Scan the original rows in their canonical order. If an original row r has support length at most 3, list the single final row (U, r). If it has support length $\ell \geq 4$, list the final rows $(C, r, 2), (C, r, 3), \ldots, (C, r, \ell)$ in increasing order of h. Thus a row identifier R is either this symbolic tuple or its rank in the canonical final-row list; these two representations are logspace interconvertible.

For a transition equation associated with a vertex $w \in V_{i+1}$, the terms are $x_{i+1,w}$ together with the variables $x_{i,u}$ for predecessors u of w. To recover the kth predecessor, the transducer scans the vertices of V_i in canonical order. For each candidate vertex u, it scans the input encoding, or equivalently evaluates the encoded adjacency relation, to test whether $u \to w$ is present. It counts

only those candidates that are actual predecessors of w. When the counter reaches k, the current vertex is the kth predecessor in the canonical order. The counter k, the candidate vertex index, and the current input-scan position require only logarithmic space.

More explicitly, suppose an original row has ordered terms $t_1, \ldots, t_\ell$ and right-hand side β. If $\ell \leq 3$, the unchanged output row (U, r) has those terms and right-hand side β. If $\ell \geq 4$, the chain rows are:

$$(C, r, 2): \quad y_{r,2} + t_1 + t_2 = 0,$$
$$(C, r, h): \quad y_{r,h} + y_{r,h-1} + t_h = 0 \qquad (3 \leq h \leq \ell - 1),$$

and the last row

$$(C, r, \ell): \quad y_{r,\ell-1} + t_\ell = \beta.$$

Thus h ranges over $2, \ldots, \ell$ for this chain; the terminal value $h = \ell$ denotes the final two-variable row, even though the last auxiliary variable is $y_{r,\ell-1}$. The index (C, r, h) therefore determines both the support and the right-hand side of the output row using only the row index r, the position h, and repeated scans to recover the needed terms. The right-hand side bit c_R is computed at the same time: for an unchanged row (U, r) it is the original right-hand side β of row r; for a chain row (C, r, h) it is 0 unless $h = \ell$, and for the last chain row (C, r, ℓ) it is β.

7.3 Indexing graph vertices

A vertex of $X(A, \rho)$ is encoded as one of: (P, j, b) for $p_{j,b}$, or (Q, R, α) for $q_{R,a}$. Here R is a final row identifier, S_R is its support in the final width-three system, and α is a bit string of length $|S_R| - 1$. For each fixed R, the strings α are listed in lexicographic order. Thus the canonical graph-vertex order is: first all variable vertices (P, j, b), ordered by j and then by b, and then all equation vertices (Q, R, α), ordered by R and then by α. The final bit of the local assignment is forced, over $\mathbb{F}_2$, by the row parity ρ_R:

$$a_d = \rho_R + \alpha_1 + \cdots + \alpha_{d-1},$$

where $S_R = \{j_1, \ldots, j_d\}$ is in canonical order. For unchanged initial or target rows, this order consists of the single variable in the row. For unchanged transition rows, it is the ordered support described above: the distinguished output variable first, followed by predecessor variables in canonical vertex order. For chain rows, it is the order in which the variables are displayed in the chain equations. The choice of order has no mathematical effect, but fixing it makes the output encoding unambiguous.

Since $d \leq 3$, the assignment component α has constant length. When $d = 1$, the string α has length 0 and there is exactly one equation vertex for that row; its single assignment bit is $a_1 = \rho_R$.

This convention avoids listing all satisfying assignments explicitly. For a row of width d, the first $d - 1$ bits of the assignment are stored in α, and the last bit is chosen so that the row parity is correct. Thus every possible α gives exactly one locally satisfying assignment. In particular, the same index set (Q, R, α) is used for $X(A, 0)$ and for $X(A, c)$; only the forced last bit, and hence some adjacencies, may differ. Consequently the two colored graphs have identical vertex and color lists before their adjacency matrices are written.

7.4 Adjacency predicate

There are no variable-variable edges and no equation-equation edges. A variable vertex (P, j, b) is adjacent to an equation vertex (Q, R, α) if and only if $j \in S_R$ and the local assignment encoded by α gives value b to variable j.

The row support S_R, the right-hand side bit ρ_R, and the local assignment bit can all be computed using $O(\log |D|)$ workspace. The value of ρ_R is 0 when the transducer is outputting $X(A, 0)$, and it

is the bit c_R described above when the transducer is outputting $X(A, c)$. The transducer first scans the at most three variables of S_R to decide whether j occurs and, if it does, to find its position q in the canonical order. If $j \notin S_R$, the vertices are not adjacent. If j is the qth variable in the canonical order of S_R and $q < |S_R|$, the assignment bit is the qth bit of α. If $q = |S_R|$, the bit is the forced value $\rho_R + \alpha_1 + \cdots + \alpha_{|S_R|-1}$. Since $|S_R| \leq 3$, this calculation uses only constant scratch space once the row support is recovered. Thus the adjacency predicate for both $X(A, 0)$ and $X(A, c)$ is logspace-computable.

This is the only adjacency case because the XOR graphs are bipartite between variable vertices and equation vertices. Therefore, once a candidate pair of output vertices is decoded, the transducer either immediately writes 0 or performs the constant-size row-support test above.

7.5 Output

For definiteness, let the transducer output a standard encoding of the pair of colored graphs: first the encoding of $X(A, 0)$ and then the encoding of $X(A, c)$. Each graph encoding lists the number of vertices, then the color label of each vertex in the canonical order, and then the adjacency matrix. The transducer loops over all vertex indices and evaluates the adjacency predicate for each pair. The two graphs have the same vertex list and color list; the only difference is that the predicate uses $\rho = 0$ for the first graph and $\rho = c$ for the second. The number of vertices in each graph is

$$2n + \sum_R 2^{|S_R|-1},$$

where the sum ranges over final rows. The transducer can compute this number by scanning the final variable and row indices and adding only the constants $1, 2$, or 4 for each row contribution. The running total is polynomially bounded, so its binary accumulator uses only $O(\log |D|)$ bits. The loop counters require logarithmic space because the output graphs have polynomial size. Hence the colored graph pair is logspace-uniform.

Size bounds. Let N_D be the number of vertices of the layered graph and let E_D be the number of directed edges. The original path-parity system has one variable for each layered vertex and one transition equation for each layered vertex outside V_0, one initial equation for each vertex of V_0, and one target equation. The support of a transition equation is one plus the indegree of the corresponding vertex. Replacing a long equation by an XOR chain introduces only linearly many auxiliary variables and rows in the length of that equation. Summed over all transition equations, this is $O(N_D + E_D)$. Thus the final system $Az = c$ has polynomially many variables and rows. Since each row of A has support at most 3, each row contributes at most four equation vertices to $X(A, \rho)$. The colored XOR graphs are therefore polynomial in $|D|$.

8 Rigid Removal of Colors

We now convert colored graph isomorphism to graph isomorphism without allowing accidental color permutations.

The colored construction deliberately gives a separate color to every variable pair and every equation row. This prevents an isomorphism from renaming variables or equations. graph isomorphism has no colors, so we must encode each color by graph structure. The standard method is to attach a rigid marker gadget to each color. Here we use a large clique of anchor vertices. A vertex of color C_i is connected to the common anchor a_0 and to the color anchor a_i. Thus its color is recoverable from its two anchor-neighbors.

The pendant paths attached to the anchors serve a second purpose: they make the anchors distinguishable from one another. Since the path lengths are all different, no isomorphism can permute the anchors.

The uncolored graph $R(Y)$ has three visible kinds of vertices. The original vertices of Y are called core vertices. The new clique vertices a_i are anchors. The remaining new vertices lie on pendant paths. The proof below shows that these three kinds of vertices can be recognized purely from the uncolored graph structure. Once that is known, any graph isomorphism of the graphs $R(Y)$ and $R(Y')$ must induce a color-preserving isomorphism of the original colored graphs.

The construction has three safeguards. First, the anchor clique is made so large that it is the unique largest clique in the graph; this forces any isomorphism to map anchors to anchors. Second, every anchor has a private pendant path of a different length; this forces the ith anchor to map to the ith anchor. Third, an original vertex of color C_i is adjacent to exactly the two anchors a_0 and a_i; once anchors are fixed, this recovers the old color from uncolored adjacency alone. Figure 1 shows these ingredients in a small schematic instance.

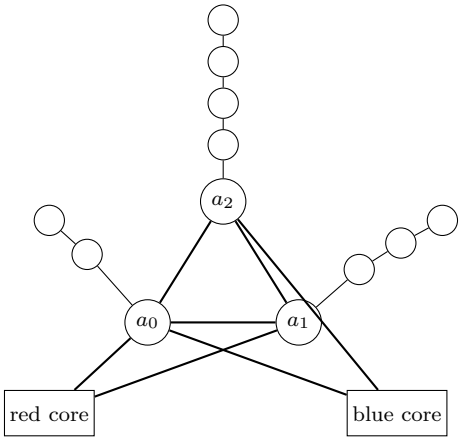

Figure 1: A schematic color-removal gadget with two old colors. The anchors form a clique and have private pendant paths of different lengths. A red core vertex is adjacent to a_0 and a_1, while a blue core vertex is adjacent to a_0 and a_2. The actual construction may contain many additional padding anchors; they are omitted from the picture.

Let Y be one of the colored graphs $X(A, 0)$ or $X(A, c)$. Let $C_1, \ldots, C_K$ be its ordered color palette, consisting of the colors Var_j and Eq_R in the fixed canonical order used by the reduction. Let $N = |V(Y)|$.

Definition 8.1 (Rigid color-removal graph $R(Y)$)**.** *Construct an uncolored graph $R(Y)$ as follows.*

First, keep all vertices and edges of the underlying uncolored graph of Y; the color labels themselves are not kept as labels.

Add anchor vertices $a_0, a_1, \ldots, a_{B-1}$, where $B = N + K + 4$, and make them a clique. Only the anchors $a_1, \ldots, a_K$ are used to encode the old colors. The additional anchors $a_{K+1}, \ldots, a_{B-1}$ are harmless padding: they make the anchor clique larger than any clique that can involve core vertices. Since $B = N + K + 4$, there are $N + 3$ padding anchors beyond the common anchor and the K color anchors.

For every original vertex v of color C_i, add edges va_0 and va_i. Thus all original vertices are adjacent to the common anchor a_0, and the second anchor-neighbor records the original color. No edges are added between two original vertices beyond those already present in Y, and no edges are added between original vertices and pendant-path vertices.

Finally, for every anchor a_i, attach a private pendant path of unique length $L_i = B + 10 + i$. Here the length is the number of edges from the anchor to the endpoint; equivalently, the component left after deleting the anchor clique has L_i path vertices. Add vertices $r_{i,1}, \ldots, r_{i,L_i}$ and edges $a_i r_{i,1},\ r_{i,1} r_{i,2},\ \ldots,\ r_{i,L_i-1} r_{i,L_i}$.

Example 8.2 (Encoding one color)**.** *Suppose the original colored graph has a color C_i. Every original vertex of that color is adjacent to exactly two anchors: the common anchor a_0 and the color anchor a_i. A vertex of a different color C_j is adjacent to a_0 and a_j instead. Therefore, once the anchors themselves are fixed, the unordered pair of anchor-neighbors recovers the original color.*

Example 8.3 (Two colors and their anchor-neighborhoods)**.** *Suppose the colored graph has two colors, red and blue, ordered as $C_1 =$* red *and $C_2 =$* blue*. In $R(Y)$, every red core vertex is adjacent to a_0 and a_1, while every blue core vertex is adjacent to a_0 and a_2. If an graph isomorphism has already been forced to fix the anchors a_0, a_1, a_2 by their pendant-path lengths, then it cannot map a red core vertex to a blue core vertex: the anchor-neighborhood $\{a_0, a_1\}$ would become $\{a_0, a_2\}$. Thus the old colors are recovered from uncolored adjacency.*

Lemma 8.4 (Color removal)**.** *Let Y and Y' be finite simple undirected colored graphs whose vertices are colored by the same ordered palette $C_1, \ldots, C_K$. Assume that every vertex has exactly one palette color and that $|V(Y)| = |V(Y')| = N$. Then*

$$R(Y) \cong R(Y') \quad \Longleftrightarrow \quad Y \cong *\mathrm{col} Y'.$$

The two graphs need not be assumed to have the same number of vertices of each color; if $R(Y) \cong R(Y')$, the proof below forces equality of the color-class sizes automatically.

Proof. If $\psi : Y \to Y'$ is a color-preserving isomorphism, extend it to $R(Y)$ by sending each anchor a_i to the corresponding anchor a'_i and each pendant-path vertex $r_{i,h}$ to $r'_{i,h}$. Since colors are preserved, anchor-neighborhoods are preserved, so this extension is an graph isomorphism $R(Y) \cong R(Y')$.

Indeed, each kind of edge is preserved: old core-core edges are preserved by ψ, anchor-anchor edges are preserved because anchors are mapped index-by-index, pendant-path edges are preserved by the map $r_{i,h} \mapsto r'_{i,h}$, and a core-anchor edge records either the common anchor a_0 or the color of the core vertex. There are no other kinds of edges in the construction, and the same description applied to ψ^{-1} preserves edges in the reverse direction. Thus the extension is a bijection preserving both adjacency and non-adjacency.

This proves the right-to-left implication: any isomorphism that already respects colors also respects the uncolored encoding of those colors.

Conversely, let $\varphi : R(Y) \to R(Y')$ be an graph isomorphism. The hypothesis $|V(Y)| = |V(Y')| = N$ ensures that both constructions use the same value $B = N + K + 4$ and the same list of pendant-path lengths.

The anchors form a clique of size $B = N + K + 4$. Consider any clique that contains a non-anchor vertex. If it contains an original core vertex v of color C_i, then every anchor in that clique must be adjacent to v, so the clique can contain only the two anchors a_0 and a_i. It can contain at most all N core vertices, and it cannot contain pendant-path vertices because no core vertex is adjacent to a pendant-path vertex. Hence such a clique has size at most $N + 2$. If the clique contains a pendant-path vertex and no core vertex, then it has size at most 2: the first path vertex is adjacent to one anchor and one path vertex, no anchor is adjacent to the second path vertex, and the two path-neighbors of an internal path vertex are not adjacent to each other. Hence any clique containing a non-anchor vertex has size at most $N + 2 < B$, whereas the anchor clique has size B.

Therefore the anchor clique is the unique largest clique, and the same argument applies to $R(Y')$. Thus φ maps the anchor clique of $R(Y)$ onto the anchor clique of $R(Y')$.

This is the reason for choosing $B = N + K + 4$ rather than merely $K + 1$. The anchor clique is so large that no clique involving original vertices can have the same size.

This step is what prevents the uncolored graph isomorphism from confusing an anchor with an original vertex. Once the anchor clique has been recognized as the unique largest clique, every isomorphism must map the set of anchors to the set of anchors before it has any freedom to act on the original graph.

We next show that the anchors are matched by their indices. Write $a'_0, \dots, a'_{B-1}$ and $r'_{i,h}$ for the corresponding vertices in $R(Y')$. Once the anchor clique is mapped to the anchor clique, the number of anchor-neighbors of a non-anchor vertex is invariant. For each anchor a_i, the vertex $r_{i,1}$ is the unique neighbor of a_i outside the anchor clique that has exactly one anchor-neighbor. Among the other neighbors of a_i outside the anchor clique, core vertices have two anchor-neighbors. Pendant-path vertices farther along the private path are not adjacent to a_i at all, and vertices on other private paths are not adjacent to a_i. Thus if $\varphi(a_i) = a'_j$, then $\varphi(r_{i,1}) = r'_{j,1}$. The component reached from $r_{i,1}$ after removing the anchor clique is a path on L_i vertices, while the corresponding component reached from $r'_{j,1}$ is a path on L_j vertices. Since the lengths L_i are pairwise distinct, and since φ maps the graph obtained by deleting the anchor clique in $R(Y)$ isomorphically to the corresponding graph for $R(Y')$, we must have $L_i = L_j$ and hence $i = j$. Therefore $\varphi(a_i) = a'_i$ for every i. This includes the common anchor, the color anchors, and all padding anchors.

After this point, the anchor set behaves like a set of named constants: an isomorphism of the uncolored graphs must send the ith anchor in $R(Y)$ to the ith anchor in $R(Y')$.

Among non-anchor vertices, the original vertices of Y are exactly those with two neighbors in the anchor clique. Pendant-path vertices have at most one anchor neighbor: $r_{i,1}$ has the single anchor neighbor a_i, while $r_{i,h}$ with $h \geq 2$ has none. Therefore φ maps original vertices to original vertices.

This recognition is purely graph-theoretic. We are not using the old labels 'core,"' anchor," or "pendant" after the construction is made; an isomorphism only sees the uncolored graph. The point of the degree and clique-size choices is that these three types can be identified from adjacency patterns alone.

An original vertex has color C_i exactly when its anchor-neighborhood is $\{a_0, a_i\}$. Since all anchors are matched by their indices, this property is preserved. Thus the restriction of φ to the original vertices of Y is a color-preserving isomorphism $Y \cong *\mathrm{col}Y'$. Indeed, the original vertices form an invariant set, and the induced subgraph on that set is exactly the old colored graph with the colors forgotten. No extra edges were added between pairs of original vertices. Therefore the restriction of φ preserves and reflects the original adjacencies, while the anchor-neighborhood argument above shows that it preserves the old colors.

This proves that the color-removal gadget has introduced no new isomorphisms. It has only translated color information into ordinary adjacency information. □

Apply this to

$$G_D^{\mathrm{col}} = X(A, 0), \qquad H_D^{\mathrm{col}} = X(A, c).$$

These two colored graphs have the same ordered color palette. They also have the same number of vertices in every color class: each variable color contains two vertices, and a row of support size d contributes 2^{d-1} satisfying local assignments for either right-hand side. Therefore Lemma 8.4 applies. Define the final ordinary graphs

$$G_D = R(G_D^{\mathrm{col}}), \qquad H_D = R(H_D^{\mathrm{col}}).$$

Proposition 8.5 (From colored GI to ordinary GI)**.** *The graphs G_D and H_D are ordinary finite simple undirected graphs, are computable in deterministic logspace from (D, s, t), and satisfy*

$$G_D \cong H_D \quad \Longleftrightarrow \quad G_D^{\mathrm{col}} \cong *\mathrm{col} H_D^{\mathrm{col}}.$$

Proof. The graphs are ordinary finite simple undirected graphs by construction: we only add undirected edges between distinct vertices, and the original colored graph edges are also ordinary graph edges once colors are ignored. The equivalence follows directly from Lemma 8.4, because the two colored XOR graphs have the same color palette and the same color-class sizes, as noted above.

For logspace computability, the transducer uses the same vertex-indexing principle as in Section 7. It first decides whether a requested output vertex is a core vertex, an anchor, or a pendant-path vertex. It then applies the appropriate local adjacency rule: core-core adjacency is the old XOR-graph adjacency, anchor-anchor adjacency means membership in the anchor clique, core-anchor adjacency records the core vertex's color, and pendant-path adjacency means consecutive positions on the same private path. All indices have logarithmic length because the graph has polynomial size.

More explicitly, write output vertices as (core, v), (anchor, i) with $0 \le i < B$, and (path, i, h) with $0 \le i < B$ and $1 \le h \le L_i$. Two distinct anchors are adjacent. A core vertex of old color C_i is adjacent exactly to $(\mathrm{anchor}, 0)$ and (anchor, i). The path vertex $(\mathrm{path}, i, 1)$ is adjacent to (anchor, i), and two path vertices are adjacent exactly when they have the same path index i and consecutive positions. All other mixed-type adjacencies are absent, except for core-core adjacencies inherited from Y.

The construction of $R(Y)$ is logspace-uniform. The quantities N, K, $B = N + K + 4$, and the lengths $L_i = B + 10 + i$ are polynomially bounded and computable with logarithmic counters. In the XOR-graph application, if the width-three system has n variables and m rows, then $K = n + m$ and

$$N = 2n + \sum_{R} 2^{|S_R|-1},$$

where the sum ranges over the final rows of the width-three system. The transducer can compute these quantities by enumerating the variables, rows, and constant-size local assignment sets. Vertices are indexed as core vertices, anchors, or pendant-path vertices. Adjacency is decidable by comparing logarithmic-size indices: one checks core-core adjacency using the predicate from Section 7, anchor-anchor adjacency by membership in the anchor clique, core-anchor adjacency by the encoded color, and path adjacency by successive path indices. The counters and accumulators used for N, K, B, and L_i range over polynomially bounded values, so each uses only $O(\log |D|)$ bits.

The size remains polynomial. The construction adds $B = O(N + K)$ anchors and

$$\sum_{i=0}^{B-1} L_i = \sum_{i=0}^{B-1} (B + 10 + i) \quad = B(B + 10) + \frac{B(B-1)}{2} = O(B^2).$$

pendant-path vertices. Since N and K are already polynomial in the input layered graph, $R(Y)$ is polynomial-size. This matters for logspace reductions: logarithmic counters are enough to range over all vertices of $R(Y)$ because the number of such vertices is polynomial.

The quadratic number of pendant-path vertices is harmless. Logspace reductions may output polynomial-size graphs, and $O(B^2)$ is still polynomial. The benefit of these paths is that they give each anchor a unique, easily recognizable fingerprint without requiring colors. □

Acknowledgements: The main reductions were written by the author and then verified and expanded by using AI. The diagram and examples were created by AI. The entire *Logspace Uniformity* section was written by AI and verified by the author.

References

[1] V. Arvind and P. P. Kurur. Graph isomorphism is in SPP. *Information and Computation*, 204(5):835–852, 2006.

[2] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[3] L. Babai. Graph isomorphism in quasipolynomial time. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC)*, pages 684–697. ACM, 2016.

[4] L. Babai and E. M. Luks. Canonical labeling of graphs. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC)*, pages 171–183. ACM, 1983.

[5] H. L. Bodlaender. Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *Journal of Algorithms*, 11(4):631–643, 1990.

[6] M. Bonamy, N. Bousquet, K. K. Dabrowski, M. Johnson, D. Paulusma, and T. Pierron. Graph isomorphism for (H_1, H_2)-free graphs: An almost complete dichotomy. arXiv preprint arXiv:1811.12252, 2018.

[7] J.-Y. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.

[8] S. Datta, N. Limaye, P. Nimbhorkar, T. Thierauf, and F. Wagner. Planar graph isomorphism is in log-space. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity (CCC)*, pages 203–214. IEEE, 2009.

[9] B. Das, J. Torán, and F. Wagner. Restricted space algorithms for isomorphism on bounded treewidth graphs. arXiv preprint arXiv:1001.0383, 2010.

[10] N. de Beaudrap. On the complexity of solving linear congruences and computing nullspaces modulo a constant. arXiv preprint arXiv:1202.3949, 2012.

[11] M. Grohe. The graph isomorphism problem. *Communications of the ACM*, 63(11):128–134, 2020.

[12] M. Grohe and D. Neuen. Recent advances on the graph isomorphism problem. arXiv preprint arXiv:2011.01366, 2020.

[13] M. Grohe and P. Schweitzer. Isomorphism testing for graphs of bounded rank width. arXiv preprint arXiv:1505.03737, 2015.

[14] J. A. Grochow and Y. Qiao. Isomorphism problems for tensors, groups, and cubic forms: completeness and reductions. arXiv preprint arXiv:1907.00309, 2019.

[15] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[16] J. E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs. In *Proceedings of the 6th Annual ACM Symposium on Theory of Computing (STOC)*, pages 172–184. ACM, 1974.

[17] V. Kaibel and A. Schwartz. On the complexity of polytope isomorphism problems. *Graphs and Combinatorics*, 19(2):215–230, 2003.

[18] K.-i. Kawarabayashi. Graph isomorphism for bounded genus graphs in linear time. arXiv preprint arXiv:1511.02460, 2015.

[19] D. Kozen. A clique problem equivalent to graph isomorphism. *ACM SIGACT News*, 10(2):50–52, 1978.

[20] S. Kratsch and P. Schweitzer. Graph isomorphism for graph classes characterized by two forbidden induced subgraphs. arXiv preprint arXiv:1208.0142, 2012.

[21] J. Köbler, U. Schöning, and J. Torán. The graph isomorphism problem is low for PP. *Computational Complexity*, 2(4):301–330, 1992.

[22] J. Köbler, U. Schöning, and J. Torán. *The Graph Isomorphism Problem: Its Structural Complexity*. Birkhäuser, 1993.

[23] S. Lindell. A logspace algorithm for tree canonization. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC)*, pages 400–404. ACM, 1992.

[24] D. Lokshtanov, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. Fixed-parameter tractable canonization and isomorphism test for graphs of bounded treewidth. arXiv preprint arXiv:1404.0818, 2014.

[25] E. M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences*, 25(1):42–65, 1982.

[26] A. Lubiw. Some NP-complete problems similar to graph isomorphism. *SIAM Journal on Computing*, 10(1):11–21, 1981.

[27] R. Mathon. A note on the graph isomorphism counting problem. *Information Processing Letters*, 8(3):131–132, 1979.

[28] R. O'Donnell, J. Wright, C. Wu, and Y. Zhou. Hardness of robust graph isomorphism, Lasserre gaps, and asymmetry of random graphs. arXiv preprint arXiv:1401.2436, 2014.

[29] D. Rosenbaum. Breaking the $n^{\log n}$ barrier for solvable-group isomorphism. arXiv preprint arXiv:1205.0642, 2012.

[30] P. Schweitzer. Towards an isomorphism dichotomy for hereditary graph classes. arXiv preprint arXiv:1411.1977, 2014.

[31] A. Snook, G. Schoenebeck, and P. Codenotti. Graph isomorphism and the Lasserre hierarchy. arXiv preprint arXiv:1401.0758, 2014.

[32] M. Sipser. *Introduction to the Theory of Computation*. 3rd edition, Cengage Learning, 2013.

[33] J. Torán. On the hardness of graph isomorphism. *SIAM Journal on Computing*, 33(5):1093–1108, 2004.

[34] V. N. Zemlyachenko, N. M. Korneenko, and R. I. Tyshkevich. Graph isomorphism problem. *Journal of Soviet Mathematics*, 29(4):1426–1481, 1985.