On the Expressive Power of GNNs for Boolean Satisfiability

Saku Peltonen (✉) and Roger Wattenhofer

ETH Zürich, Switzerland speltonen@ethz.ch

Abstract. We analyze the expressive power of Graph Neural Networks for SAT solving through the lens of the Weisfeiler-Leman (WL) test. We prove that certain pairs of 3-SAT instances with $O(n)$ variables are indistinguishable under the n-WL test, despite one being satisfiable and the other unsatisfiable. In contrast, planar SAT—a well-known NP-complete SAT variant—is fully distinguishable by 4-WL. We argue that random SAT instances are largely distinguishable, which we prove for a particular generation method. We validate this hypothesis through experiments on random instances from the G4SAT benchmark.

Keywords: SAT, GNN, Weisfeiler-Leman, expressivity

1 Introduction

Machine learning has made incredible progress in "thinking fast" tasks – such as image recognition, speech synthesis and generative modeling. The next frontier lies in "thinking slow," where problems demand reasoning, logical deduction and combinatorial search. While there has already been remarkable progress in areas like abstract reasoning [27] and theorem proving [33], some of the hardest areas remain open. One of the ultimate tests for AI reasoning is Boolean Satisfiability (SAT). As an NP-complete problem, general SAT solving is among the hardest (yet solvable) computational challenges. The question is whether machine learning methods can learn to solve SAT problems, and if there are clearly definable areas where learning is impossible.

Graph Neural Networks (GNNs) have emerged as the main approach to learning-based SAT solving. Current approaches include end-to-end SAT solvers, such as NeuroSAT [31] and QuerySAT [28], as well as hybrid approaches augmenting components of classic solvers [36,14]. GNNs are particularly well-suited for SAT solving, because formulas can be naturally represented as graphs, such as in the Literal Clause Graph (LCG), connecting literals to clauses in a bipartite graph (see Figure 2 for an example).

However, SAT is inherently a structural problem. The ability to distinguish different graph structures plays a crucial role in being able to solve it. This raises a fundamental question: *Are GNNs expressive enough to distinguish satisfiable instances from unsatisfiable instances?*

We study this question through the lens of the Weisfeiler-Leman (WL) test [37,40], and the extended k-WL hierarchy [18]. Our main result proves that even the full WL hierarchy cannot distinguish satisfiable from unsatisfiable instances in general (Theorem 4). The seminal work of Cai, Fürer and Immerman [7] showed that n-WL cannot distinguish certain pairs of non-isomorphic graphs. Our result can be viewed as an intuitive interpretation of this construction in the context of boolean formulas.

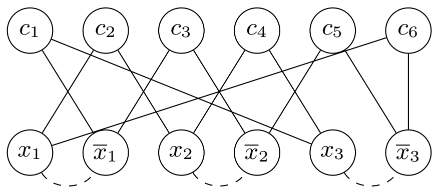
We show that indistinguishability of instances arises also in a more practical setting. Specifically, we construct a natural family of *regular* SAT formulas and show that it is NP-complete (Theorem 2). Due to the regular structure, all instances of the same size are indistinguishable, making GNNs essentially useless for this family.

By contrast, there are cases where GNNs can provably distinguish all instances. We show that PlanarSAT, a highly studied NP-complete SAT variant, is fully identified by the 4-WL test (Theorem 6). Similarly, we argue that random SAT instances are largely distinguishable by the 1-WL test, making them easier from the expressivity perspective. This mirrors behavior observed in graph isomorphism, where random graphs are typically distinguished by WL in just a few iterations [4]. We prove this formally for formulas generated using the method in [39] (Lemma 7).

So far, learning-based SAT solvers are often trained on random instances, primarily because they are easy to generate [22]. However, our results raise the question of whether random SAT instances are representative of the general SAT problem from the perspective of GNNs. It is well known that the structure of random instances is different from those of *industrial* and *crafted* instances (see [2] for a survey). Industrial instances arise from real-world problems and are often large and complex, while crafted instances are intentionally designed to be hard for SAT solvers. In contrast, random SAT instances exhibit phase-transition behavior, where hard instances only appear at a critical clause-to-variable ratio [9].

In our experiments, we assess how expressivity needs differ between random and industrial instances. We evaluate how well the WL test can distinguish literals in a formula, providing an upper bound on the expressive power of WL-powerful GNNs such as GIN [40]. Given a satisfiable formula, we test whether WL can separate literals that must be assigned different values. To do this, we construct a new formula where WL-equivalent literals are constrained to the same value. If the constrained formula is unsatisfiable, more expressivity is necessary to predict a satisfying assignment.

We conduct this experiment on random instances from the G4SAT benchmark [22] as well as industrial and crafted instances from the 2024 SAT competition [16]. In general, literals in random instances are quickly distinguished from each other. In contrast, competition instances often require significantly more iterations—and in some cases WL-powerful GNNs are not expressive enough to predict satisfying assignments. This indicates that industrial and crafted instances pose a greater challenge for GNN-based SAT solvers. More broadly, our results suggest that random formulas may not adequately capture the challenges in SAT solving for GNNs.



$$f = (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3)$$

Fig. 1: Literal-clause graph with negation connections (LCN) of a formula f. Removing the literal-literal edges represented by dashed lines gives the literal-clause graph (LCG).

2 Preliminaries

Boolean satisfiability. Let $x_1, \dots, x_n$ denote variables in a propositional logic. A literal ℓ is a variable x or its negation $\neg x$. A clause $c = \{\ell_1, \dots, \ell_s\}$ is a set of literals, representing the disjunction $\ell_1 \vee \cdots \vee \ell_s$. A formula f in Conjunctive Normal Form (CNF) is a set of m clauses $\{c_1, \dots, c_m\}$, representing the conjunction $c_1 \wedge \cdots \wedge c_m$. We write f as a logical formula $\bigwedge_{c \in f} \bigvee_{\ell \in c} \ell$ or as sets, depending on the context. Let $L(f) = \cup_{c \in f} \cup_{\ell \in c} \ell$ be the set of all literals in a formula. The Boolean Satisfiability Problem (SAT) is defined as follows: given a formula f, check whether f is satisfiable. 3-SAT is the SAT problem where each formula is in CNF and each clause consists of at most 3 literals.

Graph Notation. A graph G is a tuple (V, E). If the graph is not clear from context, we write $V(G)$ and $E(G)$. All graphs are undirected unless otherwise specified. $N(v)$ denotes the set of neighbors of a node v and $d(v) = |N(v)|$ is the degree of v. On a directed graph d^{out} denotes the outdegree of a node. A node coloring is a function $\lambda_V : V \to \mathcal{C}$ where $\mathcal{C}$ is a set of colors. Similarly, an edge coloring is a function $\lambda_E : E \to \mathcal{C}$. On edge-colored graphs we write $N_c(v) = \{w \in V(G) : \exists \{v, w\} \in E(G) \text{ s.t. } \lambda_E(\{v, w\}) = c\}$ for the neighbors of v through edges of color c.

Isomorphism. Two graphs G and H are isomorphic if there exists a bijection $\sigma : V(G) \to V(H)$ such that $\{v, w\} \in E(G)$ iff $\{\sigma(v), \sigma(w)\} \in E(H)$. On graphs with a node-coloring (c_G, c_H for G, H, respectively) we also require that $c_G(v) = c_H(\sigma(v))$ for all $v \in V(G)$. The definition is extended for edge-colored graphs in the natural way. Note the distinction between colors/features and labels: unlike colors or features, labels are unique identifiers, for example $1, \dots, n$, which do not need to be preserved under isomorphism unless otherwise specified.

Two CNF formulas f, g are isomorphic if there are bijections $\sigma_L : L(f) \to L(g)$ and $\sigma_C : f \to g$ such that (1.) $\sigma_L(\neg \ell) = \neg \sigma_L(\ell)$ for all $\ell \in L(f)$, i.e. σ_L preserves the relationship between a literal and its negation, and (2.) $\sigma_C(c) = \{\sigma_L(\ell) : \ell \in c\}$ for all $c \in f$.

Graph Neural Networks. We focus on Message Passing Neural Networks (MPNNs) which encapsulate the majority of GNN architectures. Nodes have some initial features $s_v^0 \in \mathbb{R}^d$. An MPNN operates in synchronous rounds, which are typically structured as follows. In each round $1 \leq \ell \leq L$, every node aggregates the states of its neighbors, $a_v^\ell = \mathsf{agg}(\{\{s_w^{\ell-1} : w \in N(v)\}\})$, where $\{\{.\}\}$ denotes a multiset. The nodes update their state using their previous state and the aggregated messages: $s_v^\ell = \text{upd}(s_v^{\ell-1}, a_v^\ell)$. The functions agg and upd are differentiable functions typically parameterized by neural networks. The final representations s_v^L can be used for node-level prediction tasks, or they can be aggregated into a graph-level representation.

Weisfeiler-Leman Test. The expressive power of MPNNs is bounded by the Weisfeiler-Leman (WL) algorithm, also known as color refinement:

Definition 1 (Weisfeiler-Leman algorithm). *Let $\lambda_V : V(G) \to \mathcal{C}$ be a vertex coloring. The WL algorithm computes new colorings of the graph iteratively. The initial coloring is given by $\chi^0 := \lambda_V$. For $\ell \in \mathbb{N}$, a new coloring χ^ℓ is defined as $\chi^\ell(v) := (\chi^{\ell-1}(v), \{\{\chi^{\ell-1}(v) : w \in N(v)\}\})$. The color refinement is continued until the partition of nodes given by χ^ℓ equals the partition given by $\chi^{\ell+1}$. The output of the WL algorithm is the stable coloring χ^ℓ.*

The WL algorithm can be generalized to also use an edge-coloring $\lambda_E : E(G) \to \mathcal{C}_E$. The update considers each color class of edges separately: $\chi^\ell(v) := (\chi^{\ell-1}(v), \{\{\chi_{N_c} : c \in \mathcal{C}_E\}\})$, where $\chi_{N_c} = \{\{\chi^{\ell-1}(v) : w \in N_c(v)\}\}$ are the colors of neighbors through edges of color c.

In the *Weisfeiler-Leman test*, the WL algorithm is applied to the disjoint union of G and G'. The WL test *distinguishes* G and G' if there is a color c such that the sets $\{v : v \in V(G), \chi(v) = c\}, \{v : v \in V(G'), \chi(v) = c\}$ have different cardinalities. We say that a graph G is *identified* by WL if it is distinguished from every other non-isomorphic graph H. A class of graphs $\mathcal{K}$ is identified if every graph in $\mathcal{K}$ is distinguished from every other non-isomorphic graph H (possibly $H \notin \mathcal{K}$).

k-Weisfeiler Leman Test. Let $k \geq 2$ be an integer. The *atomic type* of a tuple $\overline{v} \in V(G)^k$ encodes all facts about edge connections and colors within the tuple. Two tuples $\overline{v} \in V(G)^k, \overline{u} \in V(G')^k$ have the same atomic type if and only if the mapping $v_i \mapsto u_i$ is an isomorphism of the induced colored subgraph $G[\{v_1, \ldots, v_k\}]$ to $G[\{u_1, \ldots, u_k\}]$.

Definition 2 (k-dimensional WL algorithm). *The k-Weisfeiler-Leman (k-WL) algorithm initializes $\chi^0(\overline{v})$ as the atomic type of $\overline{v}$ for each $\overline{v} \in V(G)^k$. For $\ell \in \mathbb{N}$, a new coloring χ^ℓ is defined as $\chi^\ell(\overline{v}) := (\chi^{\ell-1}(\overline{v}), \chi_1^{\ell-1}(\overline{v}), \chi_2^{\ell-1}(\overline{v}), .., \chi_k^{\ell-1}(\overline{v}))$, where $\chi_i^{\ell-1}(\overline{v}) = \{\{\chi^{\ell-1}(\overline{v}_1, \ldots, \overline{v}_{i-1}, u, \overline{v}_{i+1}, \ldots, \overline{v}_k) : u \in V(G)\}\}$. The color refinement is continued until the partition of tuples given by χ^ℓ equals the partition given by $\chi^{\ell+1}$. The output of the WL algorithm is the stable coloring χ^ℓ.*

The k-dimensional WL test is defined analogously to the WL test. We say that G and G' are distinguished if there exists a color c such that the sets $\{\overline{v} : \overline{v} \in V(G)^k, \chi(\overline{v}) = c\}$ and $\{\overline{v} : \overline{v} \in V(G'), \chi(\overline{v}) = c\}$ have different cardinalities.

Remark 1. There are two algorithms and naming conventions in the literature. This version of the k-WL algorithm is most common in machine learning literature. Through connections to counting logic, it can be shown [12] to be equivalent to $(k-1)$-WL, as defined in for example [7,21]. See [17] for an overview.

3 Graph Representations of SAT formulas

One of the most common way to represent SAT formulas as graphs is the *literal-clause graph* (LCG). The LCG is a bipartite graph with literals on one side and clauses on the other. Edges connect literals to clauses where they appear. See Figure 1 for an example.

In GNNs, node labels of a graph are omitted (i.e. not given as input features) to preserve permutation invariance. To prevent information loss due to the removed labels, it is crucial to include edges between literals and their negations, as formalized in the next definition:[1]

Definition 3 (LCG with Negation Connections). *An* LCG *with additional edges $\{x, \neg x\}$ connecting each variable to its negation is called a Literal-Clause graph with Negation connections (*LCN*). The edges are categorized into two types: those connecting variables to their negation (literal-literal edge) and those connecting literals to clauses (literal-clause edge), with each category assigned a distinct color.*

The LCN of a CNF formula f is denoted LCN(f). Adding literal-literal edges is necessary to preserve information once labels are removed:

Lemma 1. *There are* 3-SAT *formulas f, f' such that the literal-clause graphs G_f and $G_{f'}$ are isomorphic but f is satisfiable and f' is not.*

The proof follows from a relatively simple example, which we present in Appendix A. The LCN representation is necessary, and sufficient to preserve all information:

Observation 1 *An* LCN *without node labels uniquely determines the corresponding* SAT *formula up to isomorphism.*

Given an LCN, we construct the corresponding formula by first identifying variables as pairs of nodes connected by a literal-literal edge. These pairs are labeled $x_1, \ldots, x_n$ in an arbitrary order, with one node arbitrarily designated as

[1] Literal-literal edges are already used in practice in most GNN SAT solvers [31,22], but their importance is not always stated explicitly.

the positive literal and the other as the negative. The clauses are formed based on the literal-clause edges.

Another popular representation is the *variable-clause graph* (VCG), with n variable nodes on one side and the clauses on the other. Clauses are connected to their respective variables, with edges colored depending on the sign of the variable. The VCG clearly preserves all information about the formula, making it at least as expressive as the LCN representation. However, the VCG does not respect our notion of isomorphism between formulas, as there are non-isomorphic VCG representations for isomorphic formulas. For example, flipping the sign of all occurrences of a variable x in $f = (x \vee y) \wedge (\neg x)$ produces an isomorphic formula $f' = (\neg x \vee \neg y) \wedge (x)$, but the VCGs are non-isomorphic. On the other hand, the LCNs of f and f' are the same.

Other known graph representations include the *literal-incidence graph* (LIG) and the *clause-incidence graph* (CIG), where literals (clauses) are connected to other literals (clauses) if they co-occur in a clause (share a variable). These representations are less commonly used, as they are inherently lossy.

4 Related Work

Machine Learning for SAT. One of the earliest works in this area is NeuroSAT [31] – an end-to-end SAT solver framework with GNNs. Their algorithm is based on predicting satisfiability, and hence works with single-bit supervision. QuerySAT [28] uses an unsupervised loss function computed from continuous variable values, and a query mechanism to update the variable values. Some of the recent architectures and losses can be tried with the G4SAT benchmark [22].

The end-to-end SAT solvers are mostly of methodological interest, and currently not practical for large instances. Another line of research augments classic SAT solvers with machine learning components, such as learned heuristics or branching strategies. The work of [30] adapts the NeuroSAT architecture to predict unsatisfiability cores, which is used to select branching variables. Other SAT solving components with potential for ML solutions include variable initialization [38], clause deletion [35] and restart policy [23]. See [14] for a comprehensive survey on machine learning methods in SAT solving.

Dataset generation is another promising application area. To mimic industrial SAT instances, [39] use a learning-based graph representation and design a method to generate SAT instances from their implicit model. Another line of work frames SAT generation as a bipartite graph generation problem [41].

Expressivity and the Weisfeiler-Leman test. It is well known that the Weisfeiler-Leman test bounds the expressive power of MPNNs [40]. This limitation has motivated a large number of more expressive GNN architectures, with expressivity corresponding to k-WL for some $k > 2$ [26,25,20]. A comparison of the expressivity of different GNN extensions is given by [29].

The k-WL is a powerful tool, but it is not able to solve graph isomorphism in general. The seminal work of [7] shows that there are pairs of non-isomorphic

$O(n)$-node graphs that are indistinguishable by the n-WL test. There are positive results for special graph classes. Namely, [21] show that planar graphs are identified by 4-WL. Random graphs are mostly identifiable by the WL test in two iterations [4].

Beyond structural expressivity, [13] analyzes the power of GNNs in terms of circuit complexity, showing that GNNs can decide problems in TC^0 (constant-depth circuits with polynomial size).[2]

Complexity Theory. Boolean satisfiability was the first problem proven to be NP-complete, by Stephen Cook [8]. Later, several variants of SAT have been proven to be equally hard, such as PlanarSAT [24]. SAT solving remains an active area of research, with SAT competitions being held annually [16].

The computational complexity of different equivalence relations between boolean functions was studied by [6]. In their work, two formulas f, f' on the same set of variables are said to be isomorphic if there is a bijection σ of the variables such that f, f' agree on all assignments up to the mapping σ. Under this definition, for instance, a tautology is isomorphic to the empty formula. Our notion of isomorphism is stricter, as it requires clauses to be preserved under the mapping. We find that this notion is better suited for the setting with graph representations.

Proof Complexity. The complexity of proving the unsatisfiability of propositional formulas is a central topic in proof complexity. One of the most studied systems is resolution, where the proof consists of clauses derived from the original formula using a simple inference rule. Hard examples for resolution include the Pigeonhole principle [15] and the Tseitin formulas [34]. Resolution proof length can be related to the width of the proof, where the width of a resolution proof is the maximum number of literals in any clause of the proof [5]. The complexity of resolution has also been characterized in terms of pebbling games [3,10]. In this setting, pebbling games are played on a single graph—unlike the two-graph pebbling games that correspond to k-WL indistinguishability [7].

5 Indistinguishable Families of SAT Instances

In this section, we construct explicit families of SAT formulas that are provably indistinguishable by the WL-test and the WL hierarchy. As our main technical contribution, we show that there are 3-SAT formulas that are indistinguishable by the n-WL test, despite one being satisfiable and the other not (Section 5.2). In general, distinguishing SAT instances (regardless of satisfiability) is as hard as graph isomorphism (Section 5.3). We also identify a practically relevant

[2] This implies (under common complexity theoretic assumptions) that SAT cannot be decided by GNNs. However, note that this does not imply that there must exist n-WL indistinguishable satisfiable and unsatisfiable formulas (which is what we show in Theorem 4). Indeed, as shown in Theorem 6, all PlanarSAT instances are distinguishable by 4-WL, even though PlanarSAT is NP-complete.

family of *regular* SAT formulas that are indistinguishable by 1-WL, yet remain NP-complete.

5.1 3-regular SAT

To motivate our first contribution, we start the section with a simple example of a pair of WL-indistinguishable formulas. Consider a CNF formula f on three variables x_1, x_2, x_3:

$$\begin{aligned} f = (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee x_3) \ldots & \qquad x_1 = x_3 \\ \wedge (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \ldots & \qquad x_1 \otimes x_2 \\ \wedge (x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3) & \qquad x_2 \otimes x_3 \end{aligned}$$

where $\otimes$ denotes the xor. See Figure 1 for the graph representation. The formula is satisfied by $x = (1, 0, 1)$ or $x = (0, 1, 0)$. We can make a similar but unsatisfiable formula f' by replacing the clauses encoding $x_2 \otimes x_3$ (the third line) with clauses encoding $x_2 = x_3$. Note that this change keeps all literal degrees the same. Since each literal appears in exactly two clauses in both f and f', the LCGs of f and f' are WL-indistinguishable. However, f' is clearly unsatisfiable. We generalize this example by constructing a class of regular formulas.

Definition 4. *A* SAT *instance is k-regular if each literal appears in exactly k clauses and each clause contains exactly k literals*

Despite this strong regularity, the class remains computationally hard:

Theorem 2. *3-regular* SAT *is* NP-complete*.*

Although related NP-complete variants have appeared in the literature (3-SAT [19], 3-SAT with each *variable* appearing at most 4 times [32]), we are not aware of a formal proof of this specific result, so we provide one in Appendix D for completeness.

If formulas are given in the 3-regular SAT format, a WL-powerful GNN is essentially useless in solving them:

Observation 3 *The WL test does not distinguish between any two 3-regular* SAT *formulas with the same number of variables.*

5.2 k-WL Indistinguishable SAT Instances

Given that 1-WL cannot distinguish some formulas, one might wonder whether higher-order WL tests suffice. In this section, we answer the question in the negative, showing that there are 3-SAT formulas that are indistinguishable by the WL hierarchy, despite one being satisfiable and the other not.

Theorem 4. *There are* 3-SAT *formulas $f, \tilde{f}$ with $O(n)$ variables and $O(n)$ clauses such that f is satisfiable and $\tilde{f}$ is not, but the* LCN*s of f and $\tilde{f}$ are indistinguishable by the n-WL test.*

Our result uses the seminal work of Cai, Fürer and Immerman [7], giving a pair of non-isomorphic graphs H and $\tilde{H}$ which are indistinguishable by n-WL. We construct a pair of formulas $f, \tilde{f}$ with LCNs isomorphic to H and $\tilde{H}$, respectively. On a high level, our formula f_G encodes the existence of an *even orientation* for a graph G. This is an orientation of the edges such that each node has an even number of outedges. We show that such an orientation exists if and only if the number of edges is even. Then, we construct a *twisted* formula $\tilde{f}_G$, encoding the existence of an even orientation when one of the edges is bidirectional. Exactly one of f_G and $\tilde{f}_G$ are satisfiable, depending on the parity of m. The proof of Theorem 4 is given in Appendix B.

Interestingly, the construction in Theorem 4 is similar to Tseitin formulas, which are known as hard instances for resolution refutation [34]. Tseitin formulas encode a set of linear inequalities over nodes of a graph. See Definition 6 for a formal definition. Resolution proofs—and likewise the WL test—rely on exploiting local patterns in the formula (or graph), and in both settings, the hardest instances include a global inconsistency that cannot be detected through purely local reasoning. To the best of our knowledge, this connection between Tseitin formulas and the construction of [7] has not been previously observed.

5.3 Graph Isomorphism Completeness of Distinguishing Literal-Clause Graphs

The graph isomorphism problem (GI) asks whether there is an edge-preserving bijection of the nodes. To complement the indistinguishability results, we show that, in general, distinguishing LCNs of CNF formulas is as hard as graph isomorphism. Theorem 5 is proved in Appendix C.

Theorem 5. *The graph isomorphism problem on* LCN*s of* 3-SAT *formulas is equally hard as graph isomorphism on general graphs.*

6 Positive Results for Distinguishability

The previous section showed various expressivity limitations of GNNs for SAT solving. Here, we shift focus to positive results, showing that there are classes of SAT instances that can be reliably distinguished. First, we show that PlanarSAT instances are distinguished by 4-WL. Second, we prove that under a natural random generation model, most random SAT instances are identified by 1-WL with high probability.

6.1 Planar SAT

PlanarSAT is a variant of SAT where the clauses are represented as a planar graph. The PlanarSAT language is NP-complete [24]. The following result is a consequence of [21], who showed that the 4-WL test distinguishes all planar graphs.

Theorem 6. *For any* SAT *formula f, there is an equisatisfiable* PlanarSAT *formula f' with polynomially many variables and clauses, such that the 4-WL test distinguishes f' from any other formula.*

Proof. PlanarSAT is NP-complete [24] (this also works in the LCN representation, see Lemma 1 in [24]). The 4-WL test distinguishes between all planar graphs [21].

This result shows that, despite the general limitations of the WL-hierarchy in distinguishing formulas (Section 5.2), there exist natural and computationally hard subsets of SAT, such as PlanarSAT, where already 4-WL is fully expressive.

The reduction to PlanarSAT is done by replacing edge crossings in the LCN with gadgets that ensure planarity. Unfortunately, the reduction is not efficient in practice, because each gadget adds 9 variables and 20 clauses and there may be up to $O(n^2)$ edge crossings.

Note: One might ask whether combining the reduction from GI to GI_{SAT} (Theorem 5) with Theorem 6 gives a fast graph isomorphism algorithm. After all, Theorem 6 allows us to distinguish any two planar formulas efficiently. However, the problem is that the reduction to PlanarSAT [24] is not permutation invariant, which is necessary for the isomorphism between formulas to be preserved.

6.2 Random SAT Instances

Random SAT instances can be defined in various ways, often depending on the desired structure or difficulty. In this section, we consider instances generated from randomly sampled literal-incidence graphs (LIGs), where each literal is a node and edges connect literals that co-occur in a clause. While the LIG representation loses some logical information, [39] gives a principled procedure for extracting a CNF formula from it:

Lemma 2 ([39]). *Given a literal-incidence graph G, a corresponding* CNF *formula can be extracted by computing a minimal clique edge cover of G.*[3] *The clauses of the formula correspond to the cliques in the edge cover. The generated formula does not contain duplicate clauses, subsumed clauses or unit clauses.*

We show that a CNF formula extracted from a random literal-incidence graph is likely identified by WL. The proof can be found in Appendix E.

Theorem 7. *A* CNF *formula extracted from a uniformly random literal-incidence graph with n literals is identified by the WL test with probability at least $1-(n)^{-1/7}$, over the choice of a LIG, for a large-enough n.*

[3] A clique edge cover is a set of cliques in G, such that all edges belong to at least one clique.

7 Experiments

We aim to evaluate whether 1-WL powerful architectures (such as GIN [40]) are, in principle, capable of predicting a satisfying assignment to SAT formulas. Our experiments are based on the fact that in a node-level prediction task, nodes that are equivalent under WL must have the same output. In the context of SAT, this means that if two literals are indistinguishable by WL, they must have the same value in a satisfying assignment. For datasets, we use the 2024 SAT competition instances [16] and random instances generated with the G4SAT benchmark [22].

7.1 Setup

Given a satisfiable formula, we add equality constraints between all WL-equivalent literals, and check whether the augmented formula remains satisfiable. This is a necessary (but not a sufficient) condition for a WL-powerful GNN to predict a satisfying assignment.

Formally, let f be a satisfiable formula. Running WL for $r \geq 1$ rounds on $\mathrm{LCN}(f)$ gives a partition of the literals $L_1, \ldots, L_s$. We construct an augmented formula f_r that restricts literals in each partition to the same value. For each equivalence class $L_j = \{\ell_1^j, \ldots, \ell_{n_j}^j\}$, we add the clauses $g_j := (\neg \ell_{n_j}^j \vee \ell_1^j) \wedge \bigwedge_{i=1}^{n_j-1} (\neg \ell_i^j \vee \ell_{i+1}^j)$. The formula g_j encodes an equality constraint between the literals in L_j. Given a satisfiable formula f, a WL-powerful architecture can predict a satisfying assignment within r rounds *only if* $f_r = f \wedge \bigwedge_{i=1}^{s} g_j$ is satisfiable. We solve f_r for different values of r, from $r = 1$ to $r = r_{\text{converged}}$, where $r_{\text{converged}}$ is the number of rounds for WL to converge on the LCN of the original formula f. We let r_{crit} be the smallest r such that f_r is satisfiable, if such a round exists. Conversely, if f_r is unsatisfiable for all r, we conclude that 1-WL is not sufficiently powerful to predict satisfying assignments for f.

7.2 Datasets

Random Instances. We use the G4SAT benchmark to generate random instances from various families. The families include random 3-SAT [9], the *CA* family mimicking community structures in industrial instances [11], and the *SR* family designed for NeuroSAT [31]. Additional families include *k-clique*, *k-domset*, and *k-vercov*, which encode combinatorial problems on Erdős-Rényi random graphs. See [22] for a complete list of descriptions of the families.

Competition Instances. We use instances from the 2024 SAT competition [16]. The instances are selected as hard examples for various applications, such as scheduling, cryptography, and hardware equivalence checking. Some families are purely synthetic and hand-crafted to be difficult for SAT solvers, such as formulas encoding the Pigeonhole principle. Detailed descriptions of the families can be found in the competition proceedings [16].[4]. The size of the instance varies from

[4] A list of instances is available at `https://benchmark-database.de/?track=main_2024&result=sat`

Table 1: Results on random instances, grouped by family. r_{crit} is the smallest number of rounds for which the augmented formula f_r is satisfiable. $r_{\text{converged}}$ is the number of rounds for WL to converge. WL-expressible is the percentage of instances where the augmented formula is satisfiable. All values are reported as mean $\pm$ standard deviation.

family	r_{crit}	$r_{\text{converged}}$	WL-expressible
3-sat	2.99 ± 0.13	3.81 ± 0.39	100%
k-clique	4.11 ± 0.62	6.28 ± 0.88	98%
k-domset	4.23 ± 0.66	5.55 ± 0.82	100%
k-vercov	4.86 ± 1.07	6.01 ± 1.23	100%
sr	2.01 ± 0.10	3.00 ± 0.04	100%

a few hundred variables to 50 million variables. Due to limited computational resources, we limit instances to those under 10 MB in size and test a few instances from each family.

7.3 Results

See Table 1 for the results. The number of rounds needed for WL to distinguish literals sufficiently is very low, usually 3 or 4. We observed that in many cases (about 40% of all formulas), WL actually gives all literals unique identifiers. In this case, the constrained formula f_r is trivially satisfiable because it is equal to f. The only family with some formulas that could not be solved by WL is the k-clique. This is likely due to symmetries in the underlying graph that WL cannot resolve.[5] See Table 4 (Appendix F) for the full results on random instances.

For random 3-SAT instances, regardless of the size of the formula, literals are almost always sufficiently identified after 3 iterations, and WL converges in 4 rounds. This pattern is due to the constant degree of the clauses. In the first iteration, each literal sees its degree d_ℓ. However, the second iteration does not refine the literal partition because every neighbor is a clause with degree 3—only on the third iteration, the literals observe the degrees of other literals in shared clauses.

SAT Competition Instances. An overview of the results is shown in Table 2 and a more detailed breakdown is given in Table 3 (Appendix F). WL takes considerably more rounds to converge, which is partly explained by the larger size of the instances. Across 28 instance families, 15 contained instances where WL is not expressive enough, and 15 families contained instances where WL is expressive enough to predict a satisfying assignment. In general, all formulas in a family tend to be either sufficiently distinguished by WL or not, with the

[5] For example, the instance may contain two nodes that are each fully connected to the same clique but not to each other, making them indistinguishable under WL yet mutually exclusive in the satisfying assignment.

Table 2: Results on a selection of the 2024 SAT competition instances. Each row is a separate instance. For the formulas where WL is expressive enough, r_{crit} is the smallest number of rounds for which the augmented formula f_r is satisfiable.

Family	r_{crit}	$r_{\text{converged}}$	n_{vars}
circuit-multiplier	-	7	1013
heule-folkman	-	5	15215
heule-nol	-	9	1419
minimum-disagreement-parity	-	8	1021
random-circuits	-	6	3248
argumentation	2	3	300
binary-tree-parity	22	22	511
cryptography-simon	21	22	4128
hamiltonian	4	5	600
maxsat-optimum	28	29	21894

exception of the scheduling and cryptography families. An example of a family with indistinguishable structure is *heule-nol*, which encodes a type of grid coloring problem [16]. The regular structure of the instances makes it difficult for WL to distinguish literals.

8 Conclusions and Future Work

Our theoretical results establish fundamental limitations on the expressive power of GNNs for SAT solving. We show that even the full WL hierarchy cannot distinguish between satisfiable and unsatisfiable formulas, while also revealing connections to resolution complexity and offering a new perspective on the classic construction of Cai, Fürer and Immerman [7]. Additionally, we identify an NP-complete but WL-indistinguishable class of SAT instances, as well as provide positive guarantees for distinguishing random and planar SAT instances.

Experimentally, we show that 1-WL powerful architectures are, in principle, expressive enough to predict satisfying assignments for random SAT instances, but struggle with industrial and crafted benchmarks. Our test setup allows us to see if a WL-powerful GNN has the *necessary* expressive power for predicting satisfying assignments, though it does not capture whether that expressivity is *sufficient* for generalizable learning. Even for random SAT formulas, improved generalization may require higher-order GNNs, symmetry breaking, or other architectural improvements.

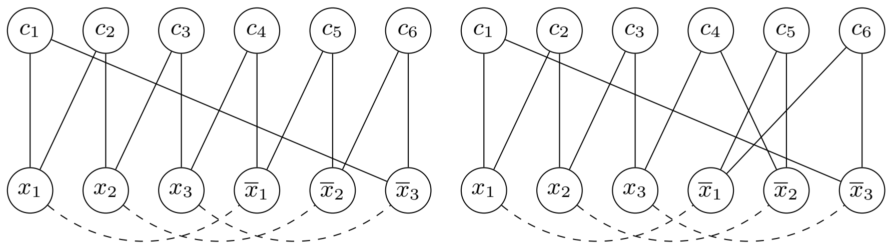
We hope to see GNNs applied to industrial SAT instances in the future. While this remains challenging—due to the lack of scalable generators and the large size of many industrial instances—these instances provide a structurally richer and potentially more demanding testbed. Progress in this direction could offer new insights into generalization that remain hidden when only using random instance distributions.

References

1. Ajtai, M.: Recursive construction for 3-regular expanders. Combinatorica **14**(4), 379–416 (1994). https://doi.org/10.1007/BF01302963, https://doi.org/10.1007/BF01302963
2. Alyahya, T.N., Menai, M.E.B., Mathkour, H.: On the structure of the boolean satisfiability problem: A survey. ACM Comput. Surv. **55**(3) (Mar 2022). https://doi.org/10.1145/3491210, https://doi.org/10.1145/3491210
3. Atserias, A., Dalmau, V.: A combinatorial characterization of resolution width. Journal of Computer and System Sciences **74**(3), 323–334 (2008). https://doi.org/https://doi.org/10.1016/j.jcss.2007.06.025, https://www.sciencedirect.com/science/article/pii/S0022000007000876, computational Complexity 2003
4. Babai, L., Erdös, P., Selkow, S.M.: Random graph isomorphism. SIAM Journal on Computing **9**(3), 628–635 (1980). https://doi.org/10.1137/0209047, https://doi.org/10.1137/0209047
5. Ben-Sasson, E., Wigderson, A.: Short proofs are narrow—resolution made simple. J. ACM **48**(2), 149–169 (Mar 2001). https://doi.org/10.1145/375827.375835, https://doi.org/10.1145/375827.375835
6. Borchert, B., Ranjan, D., Stephan, F.: On the computational complexity of some classical equivalence relations on boolean functions. Theory of Computing Systems **31**(6), 679–693 (1998). https://doi.org/10.1007/s002240000109, https://doi.org/10.1007/s002240000109
7. Cai, J.Y., Fürer, M., Immerman, N.: An optimal lower bound on the number of variables for graph identification. Combinatorica **12**(4), 389–410 (1992). https://doi.org/10.1007/BF01305232, https://doi.org/10.1007/BF01305232
8. Cook, S.A.: The complexity of theorem-proving procedures. In: Proceedings of the Third Annual ACM Symposium on Theory of Computing. p. 151–158. STOC '71, Association for Computing Machinery, New York, NY, USA (1971). https://doi.org/10.1145/800157.805047
9. Crawford, J.M., Auton, L.D.: Experimental results on the crossover point in random 3-sat. Artificial Intelligence **81**(1), 31–57 (1996). https://doi.org/https://doi.org/10.1016/0004-3702(95)00046-1, https://www.sciencedirect.com/science/article/pii/0004370295000461, frontiers in Problem Solving: Phase Transitions and Complexity
10. Galesi, N., Thapen, N.: Resolution and pebbling games. In: Bacchus, F., Walsh, T. (eds.) Theory and Applications of Satisfiability Testing. pp. 76–90. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
11. Giráldez-Cru, J., Levy, J.: A modularity-based random sat instances generator. In: Proceedings of the 24th International Conference on Artificial Intelligence. p. 1952–1958. IJCAI'15, AAAI Press (2015)
12. Grohe, M.: Descriptive Complexity, Canonisation, and Definable Graph Structure Theory. Lecture Notes in Logic, Cambridge University Press (2017)
13. Grohe, M.: The descriptive complexity of graph neural networks. In: 2023 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS). pp. 1–14 (2023). https://doi.org/10.1109/LICS56636.2023.10175735
14. Guo, W., Zhen, H.L., Li, X., Luo, W., Yuan, M., Jin, Y., Yan, J.: Machine learning methods in solving the boolean satisfiability problem. Machine Intelligence Research **20**(5), 640–655 (2023). https://doi.org/10.1007/s11633-022-1396-2, https://doi.org/10.1007/s11633-022-1396-2

15. Haken, A.: The intractability of resolution. Theoretical Computer Science **39**, 297–308 (1985). https://doi.org/https://doi.org/10.1016/0304-3975(85)90144-6, https://www.sciencedirect.com/science/article/pii/0304397585901446, third Conference on Foundations of Software Technology and Theoretical Computer Science
16. Heule, M.J., Iser, M., Järvisalo, M., Suda, M.: Proceedings of SAT competition 2024: Solver, benchmark and proof checker descriptions (2024)
17. Huang, N., Villar, S.: A short tutorial on the Weisfeiler-Lehman test and its variants. ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) pp. 8533–8537 (2021)
18. Immerman, N., Lander, E.: Describing Graphs: A First-Order Approach to Graph Canonization, pp. 59–81. Springer New York, New York, NY (1990). https://doi.org/10.1007/978-1-4612-4478-3_5, https://doi.org/10.1007/978-1-4612-4478-3_5
19. Karp, R.M.: Reducibility among Combinatorial Problems, pp. 85–103. Springer US, Boston, MA (1972). https://doi.org/10.1007/978-1-4684-2001-2_9, https://doi.org/10.1007/978-1-4684-2001-2_9
20. Keriven, N., Peyré, G.: Universal invariant and equivariant graph neural networks. Curran Associates Inc., Red Hook, NY, USA (2019)
21. Kiefer, S., Ponomarenko, I., Schweitzer, P.: The Weisfeiler–Leman dimension of planar graphs is at most 3. J. ACM **66**(6) (Nov 2019). https://doi.org/10.1145/3333003, https://doi.org/10.1145/3333003
22. Li, Z., Guo, J., Si, X.: G4SATBench: Benchmarking and advancing SAT solving with graph neural networks. Transactions on Machine Learning Research (2024)
23. Liang, J.H., Oh, C., Mathew, M., Thomas, C., Li, C., Ganesh, V.: Machine learning-based restart policy for CDCL SAT solvers. In: Beyersdorff, O., Wintersteiger, C.M. (eds.) Theory and Applications of Satisfiability Testing – SAT 2018. pp. 94–110. Springer International Publishing, Cham (2018)
24. Lichtenstein, D.: Planar formulae and their uses. SIAM Journal on Computing **11**(2), 329–343 (1982). https://doi.org/10.1137/0211025, https://doi.org/10.1137/0211025
25. Maron, H., Ben-Hamu, H., Serviansky, H., Lipman, Y.: Provably Powerful Graph Networks. Curran Associates Inc., Red Hook, NY, USA (2019)
26. Morris, C., Ritzert, M., Fey, M., Hamilton, W.L., Lenssen, J.E., Rattan, G., Grohe, M.: Weisfeiler and Leman go neural: higher-order graph neural networks. In: Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence. AAAI'19/IAAI'19/EAAI'19, AAAI Press (2019). https://doi.org/10.1609/aaai.v33i01.33014602
27. OpenAI: Openai o3 breakthrough high score on arc-agi-pub (2024), https://arcprize.org/blog/oai-o3-pub-breakthrough?utm_source=chatgpt.com, accessed: January 2024
28. Ozolins, E., Freivalds, K., Draguns, A., Gaile, E., Zakovskis, R., Kozlovics, S.: Goal-aware neural SAT solver. In: 2022 International Joint Conference on Neural Networks (IJCNN). p. 1–8. IEEE (Jul 2022). https://doi.org/10.1109/ijcnn55064.2022.9892733, http://dx.doi.org/10.1109/IJCNN55064.2022.9892733
29. Papp, P.A., Wattenhofer, R.: A theoretical comparison of graph neural network extensions. In: Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., Sabato, S. (eds.) Proceedings of the 39th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 162, pp. 17323–17345. PMLR (17–23 Jul 2022)

30. Selsam, D., Bjørner, N.S.: Guiding high-performance SAT solvers with unsat-core predictions. In: International Conference on Theory and Applications of Satisfiability Testing (2019)
31. Selsam, D., Lamm, M., Bünz, B., Liang, P., de Moura, L., Dill, D.L.: Learning a SAT solver from single-bit supervision (2019), https://arxiv.org/abs/1802.03685
32. Tovey, C.A.: A simplified NP-complete satisfiability problem. Discrete Applied Mathematics **8**(1), 85–89 (1984). https://doi.org/https://doi.org/10.1016/0166-218X(84)90081-7
33. Trinh, T.H., Wu, Y., Le, Q.V., He, H., Luong, T.: Solving olympiad geometry without human demonstrations. Nature **625**(7995), 476–482 (2024). https://doi.org/10.1038/s41586-023-06747-5, https://doi.org/10.1038/s41586-023-06747-5
34. Urquhart, A.: Hard examples for resolution. J. ACM **34**(1), 209–219 (Jan 1987). https://doi.org/10.1145/7531.8928, https://doi.org/10.1145/7531.8928
35. Vaezipoor, P., Lederman, G., Wu, Y., Grosse, R.B., Bacchus, F.: Learning clause deletion heuristics with reinforcement learning (2020), https://api.semanticscholar.org/CorpusID:221089812
36. Wang, W., Hu, Y., Tiwari, M., Khurshid, S., McMillan, K., Miikkulainen, R.: Neuroback: Improving CDCL SAT solving using graph neural networks. In: The Twelfth International Conference on Learning Representations (2024)
37. Weisfeiler, B., Lehman, A.A.: A reduction of a graph to a canonical form and an algebra arising during this reduction. Nauchno-Technicheskaya Informatsia p. 2(9):12–16 (1968)
38. Wu, H.: Improving SAT-solving with machine learning. In: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education. p. 787–788. SIGCSE '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3017680.3022464, https://doi.org/10.1145/3017680.3022464
39. Wu, H., Ramanujan, R.: Learning to generate industrial SAT instances. Proceedings of the International Symposium on Combinatorial Search **10**, 206–207 (09 2021). https://doi.org/10.1609/socs.v10i1.18493
40. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: International Conference on Learning Representations (2019)
41. You, J., Wu, H., Barrett, C., Ramanujan, R., Leskovec, J.: G2SAT: Learning to generate SAT formulas. NeurIPS (2019)



Left: $f = (x_1 \vee \neg x_3) \wedge (x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (x_3 \vee \neg x_1) \wedge (\neg x_1 \vee \neg x_2) \wedge (\neg x_2 \vee \neg x_3)$
Right: $f' = (x_1 \vee \neg x_3) \wedge (x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (x_3 \vee \neg \mathbf{x_2}) \wedge (\neg x_2 \vee \neg x_1) \wedge (\neg \mathbf{x_1} \vee \neg x_3)$

Fig. 2: LCNs of f and f'. The difference between the formulas is highlighted in bold. Removing the literal-literal edges represented by the dashed lines gives the literal-clause graphs.

A Graph Representations of SAT formulas

The following lemma highlights the importance of adding the literal-negation edges.

Lemma 1. *There are* 3-SAT *formulas f, f' such that the literal-clause graphs G_f and $G_{f'}$ are isomorphic but f is satisfiable and f' is not.*

Proof. Consider the formulas shown in Figure 2 The two LCGs (dashed lines excluded) are isomorphic. f has a solution $x_1 = 1, x_2 = 0, x_3 = 1$. However, f' is not satisfiable: $x_1 = 1$ implies $x_2 = 0$ because of c_5, and $x_3 = 0$ because of c_6. Now c_3 is false. Conversely, $x_1 = 0$ implies $x_2 = 1$ because of c_2, and $x_3 = 0$ because of c_1. This makes c_4 false.

B k-WL Indistinguishable Instances (Proof of Theorem 4)

In this section, we prove the following theorem:

Theorem 4. *There are* 3-SAT *formulas $f, \tilde{f}$ with $O(n)$ variables and $O(n)$ clauses such that f is satisfiable and $\tilde{f}$ is not, but the* LCN*s of f and $\tilde{f}$ are indistinguishable by the n-WL test.*

The construction is based on the seminal work of Cai, Fürer, Immerman [7] (CFI), giving a pair of non-isomorphic graphs H and $\tilde{H}$ which are indisintuishable by n-WL. We construct a pair of formulas $f, \tilde{f}$ with LCNs isomorphic to H and $\tilde{H}$, respectively. On a high level, our formula f_G encodes the existence of an *even orientation* for a graph G. This is an orientation of the edges such that each node has an even number of outedges. We show that such an orientation exists if and only if the number of edges is even. Then, we construct a *twisted* formula $\tilde{f}_G$, encoding the existence of an even orientation when one of the edges is bidirectional. Exactly one of f_G and $\tilde{f}_G$ are satisfiable, depending on the parity of m.

B.1 The construction

The CFI construction [7] takes in a low degree graph G with only linear sized separators and produces non-isomorphic graphs $X(G)$ and $\tilde{X}(G)$ that are indistinguishable by WL. Next, we go over the steps to construct $X(G)$, or equivalently a formula f_G with an LCN isomorphic to $X(G)$. For each vertex $v \in V(G)$, we define the following subformula X_k, where $k = d(v)$:

$$\begin{aligned}
&\text{Literals: } A_k \cup B_k, \text{ where}\\
&A_k = \{a_i \mid 1 \le i \le k\},\\
&B_k = \{b_i \mid 1 \le i \le k\}\\
&\text{Clauses: } C_k \cup D_k, \text{ where}\\
&C_k = \{c_S = \left(\vee_{i \in S}\, a_i\right) \vee \left(\vee_{i \notin S}\, b_i\right) \mid S \subseteq [k], |S| \text{ is even}\}\\
&D_k = \{a_i \vee b_i \mid i \in \{1, \ldots, k\}\}
\end{aligned}$$

Here $[k]$ denotes the set $\{1, \ldots, k\}$. See Figure 3 for a diagram of the LCN of X_k with $k = 3$. This graph corresponds exactly to the graph X_k in the CFI construction[6]. Note that the negations of the literals are not contained in X_k – they will be defined later.

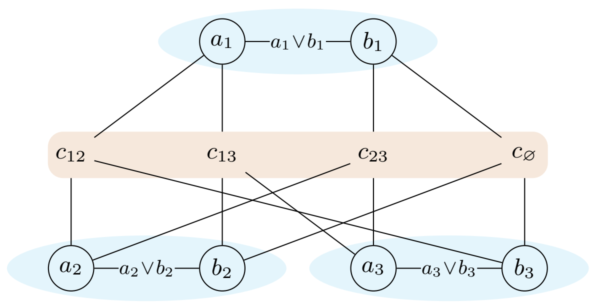
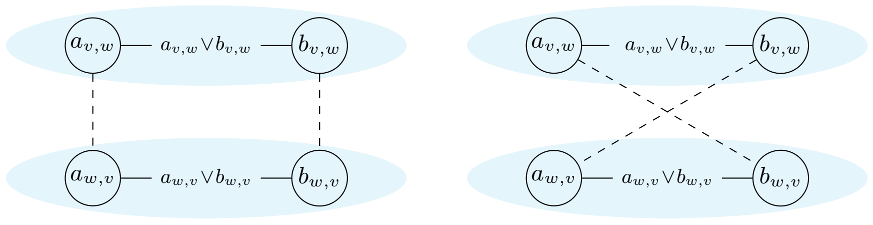


Fig. 3: An LCN of the formula X_3, corresponding to the graph X_3 in the CFI construction. Literal nodes are circled. The clauses are connected to the literals by solid lines.

For a given graph G, the full formula f_G is constructed as follows. For each vertex $v \in V(G)$, add the subformula $X_{d(v)}$. Each edge $\{v, w\}$ of v is associated with one of the literal pairs (a_i, b_i), where we call the literals $a_{v,w}, b_{v,w}$. The node w on the other side of this edge uses the negations of the literals, that is, $a_{w,v} = \neg a_{v,w}$ and $b_{w,v} = \neg b_{v,w}$. In the LCN, the literal $a_{v,w}$ is connected to its negation $a_{w,v}$, and $b_{v,w}$ to $b_{w,v}$. See Figure 4a for an example.

[6] The nodes corresponding to the D_k clauses are not present in the standard construction in [7], but they mention that nodes connecting each a_i to b_i can be added.



(a) A normal edge. (b) A twisted edge.

Fig. 4: Constructions in the formula f_G and $\tilde{f}_G$ for an edge $\{v, w\} \in E(G)$. The edges are represented vertically, with the top ellipse corresponding to v's side and the bottom to w's side of the edge. Solid lines connect clauses to their literals. Dashed lines connect literals and their negations.

Now, we define a *twisted* formula $\tilde{f}_G$, and the corresponding twisted graph $\tilde{X}(G)$ as follows. An edge $\{v, w\} \in E(G)$ is chosen arbitrarily. The literal-literal connections are twisted, so that $a_{v,w}$ becomes the negation of $b_{w,v}$ and $b_{v,w}$ becomes the negation of $a_{w,v}$. A twisted edge is shown in Figure 4b.

Observation 8 *The* LCN*s of f_G and $\tilde{f}_G$ are isomorphic to the graphs $X(G)$ and $\tilde{X}(G)$ in [7], respectively.*

To state the result of [7], we need the concept of a separator:

Definition 5. *A separator of a graph G is a set $S \subset V(G)$ such that the induced subgraph on $V \setminus S$ has no connected component with more than $|V(G)|/2$ vertices.*

Theorem 9 (Theorem 6.4 in [7]). *Let G be a graph such that every separator of G has at least $k+1$ vertices. Then $X(G)$ and $\tilde{X}(G)$ are non-isomorphic but k-WL indistinguishable.*

B.2 Satisfiability of f_G and $\tilde{f}_G$

By construction, literals have their negations on the other side of each edge:

Remark 2. For a normal (not twisted) edge $\{v, w\} \in E(G)$, $a_{v,w} = \neg a_{w,v}$ and $b_{v,w} = \neg b_{w,v}$. If the edge is twisted, $a_{v,w} = \neg b_{w,v}$ and $b_{v,w} = \neg a_{w,v}$.

The following is true for f_G and $\tilde{f}_G$, for any edge $\{v, w\}$ (twisted or not):

Observation 10 *The literals $a_{v,w}, b_{v,w}$ are non-equal in any satisfying assignment.*

Proof. This is forced by $(a_{v,w} \vee b_{v,w}) \wedge (a_{w,v} \vee b_{w,v}) \equiv (a_{v,w} \vee b_{v,w}) \wedge (\neg a_{v,w} \vee \neg b_{v,w})$.

This allows us to talk about satisfying assignments of f_G and $\tilde{f}_G$ as orientations of edges of G:

Remark 3. Any satisfying assignment is uniquely characterized by the values of A. On a graph without twists, assignments to A correspond one-to-one with orientations of G. On a twisted edge, either both $a_{v,w}$ and $a_{w,v}$ are true, or both are false.

Consequently, the number of A-literals set to true in f_G is m, while in $\tilde{f}_G$ it is $m-1$ or $m+1$.

The next lemma characterizes the satisfiability of X_k:

Lemma 3. *Let k be an odd integer and assume that $a_i \neq b_i$ for $1 \leq i \leq k$. X_k is satisfied if and only if an even number of a_i's (or equivalently b_i's) are set to true.*

Proof. Let $T \subset \{1,\dots,k\}$ be the set of indices i such that a_i is set to true. We start by proving that X_k is satisfied whenever $|T|$ is even. The clauses in D_k are satisfied whenever $a_i \neq b_i$, which is guaranteed by our assumption. Consider any clause $c_S = \big(\vee_{i\in S}\, a_i\big) \vee \big(\vee_{i\notin S}\, b_i\big)$. Since k is odd, $|S|$ is even, and $|T|$ is even, $S \cup T$ is not a partition of $\{1,\dots,k\}$. Hence, there is some index i such that i is in both sets or neither of the sets. If $i \in S \cap T$, then a_i satisfies c_S. Else $i \notin S \cup T$, and b_i satisfies c_S.

Conversely, suppose that $|T|$ is odd. Consider the clause c_S with $S = \{1,\dots,k\} \setminus T$. It is a clause because $|S|$ is even. It is unsatisfied, since $a_i = F$ for each $i \in S$ and $b_i = F$ for each $i \notin S$.

The following observation is a direct consequence of Lemma 3 and Theorem 10:

Observation 11 *Assume the degree of every node in G is odd. In any solution of f_G or $\tilde{f}_G$, the number of A-literals set to true is even.*

We say that a simple graph G has an *even orientation* if there is an orientation of the edges such that all nodes have even outdegree. The following is a simple fact characterizing the existence of even orientations:

Lemma 4. *Let H be a simple connected graph. H has an even orientation if and only if $m = |E(H)|$ is even.*

Proof. Let $d^{\text{out}}(v)$ denote the outdegree of a node v. It always holds that $\sum_{v\in V} d^{\text{out}}(v) = m$.

Assume there exists an even orientation of H. Then $d^{\text{out}}(v)$ is even for all v. Since the sum of even terms is always even, m must be even.

Now assume that m is even. Start with an arbitrary orientation of the edges. For any two vertices v, w with odd outdegree, take an arbitrary path connecting v and w and reorient the edges on the path. This changes the outdegrees of v and w from odd to even, while not changing the parity of other nodes. Repeat this process until all nodes have even outdegree. Suppose that this is not possible, that is, we are left with a single node v with odd outdegree. We have $d^{\text{out}}(v) = m - \sum_{w\in V\setminus\{v\}} d^{\text{out}}(w)$. The left side is odd, while the right side is even because m and the terms of the sum are even. Hence, we can always find an even orientation.

Using this, we can prove the main lemma characterizing the satisfiability of f_G and $\tilde{f}_G$:

Lemma 5. *Let G be a connected graph where all degrees are odd.*[7] *If $m = |E(G)|$ is even (odd), f_G is satisfiable (unsatisfiable), and $\tilde{f}_G$ is unsatisfiable (satisfiable).*

Proof. Suppose that m is even. G has an even orientation by Lemma 4. We can use this to extract a satisfying assignment of f_G by setting $a_{v,w} = T$ for each outedge of v. This assignment satisfies every subformula by Lemma 3. On the other hand, the twisted formula must have $m-1$ or $m+1$ A-literals true (Remark 3), making it unsatisfiable (Theorem 11).

If m is odd, f_G is unsatisfiable by Theorem 11. On the other hand, the twisted edge allows us to set both a-literals of the edge to false, effectively removing the edge. The graph G with $\{v, w\}$ removed has an even number of edges, so we can find a satisfying assignment by computing an even orientation of the remaining edges.

Proof of Theorem 4. Let G be a graph with odd degrees, where the size of the smallest separator is linear in n. We can use the construction of [1] for 3-regular expanders. The LCNs of f and f' are isomorphic to $X(G)$ and $\tilde{X}(G)$, respectively. These graphs are indistinguishable by Theorem 9. By the above Lemma 5, exactly one of f_G and $\tilde{f}_G$ is satisfiable. Note that both f_G and $\tilde{f}_G$ have at most 3 literals per clause. The number of variables is $n \cdot \Delta(G) = O(n)$ and the number of clauses $n \cdot (2^{\Delta(G)-1} + 1) = O(n)$. □

Tseitin Formulas. Interestingly, this construction is related to Tseitin formulas, which are known as hard instances for resolution refutation proofs [34].

Definition 6. *A Tseitin formula is constructed by taking a graph G and a charge function $c : V(G) \to \{0, 1\}$ labeling the vertices. Each edge $e \in E(G)$ is associated with a variable x_e. For each vertex, there is a constraint $\xi_v = \sum_{w \in N(v)} x_{\{v,w\}} = c(v) \mod 2$, meaning that the parity of the sum of variables of v's edges is equal to the charge. The full formula is defined as $\wedge_{v \in V} \xi_v$.*

It is known that a Tseitin formula is satisfiable if and only if the sum of charges is even. Hence, satisfiability is a global property of the graph. When the underlying graph is an expander (with small degrees), Tseitin formulas are known to be hard instances for resolution [34].

C Hardness of Distinguishing SAT Instances (Proof of Theorem 5)

We show that, in general, distinguishing LCNs is as hard as graph isomorphism. The graph isomorphism problem (GI) asks whether there is an edge-preserving

[7] The proof can be generalized to a mix of odd and even degrees, but we chose to do this for simplicity of the argument.

bijection of the nodes. Formally, the decision problem is

$$\text{GI} = \{(G, H) : G, H \text{ are isomorphic graphs}\}$$

Let $\mathcal{G} = \{\text{LCN}(f) : f \in \text{3-SAT}\}$ be the set of LCNs of all 3-SAT formulas. We define the graph isomorphism problem on LCNs as the language

$$\text{GI}_{\text{SAT}} = \{(G, H) : G, H \in \mathcal{G} \text{ are isomorphic graphs}\}$$

We use the following formula to encode all relevant information about a graph in a CNF formula:

Definition 7 (f_G). *For each $v \in V(G)$, let x_v be a variable. The edges of G (and $|V(G)|$) can be encoded as a* CNF *formula $f_G = \left(\bigwedge_{\{v,w\}\in E(G)}(x_v \vee x_w)\right) \wedge \left(\bigwedge_{v\in V} x_v\right)$.*

This formula is trivially satisfiable and not meaningful from a logical perspective, but it uniquely encodes G up to isomorphism.

Observation 12 *The* LCN *of f_G is equal to G with the following modifications. Each edge is subdivided with the corresponding clause node added in the middle. The original nodes correspond to the positive literals. Two leaf nodes corresponding to the negative literal and the unit clause is connected to each positive literal.*

Theorem 5. *The graph isomorphism problem on* LCN*s of* 3-SAT *formulas is equally hard as graph isomorphism on general graphs.*

Proof. $\text{GI} \leq \text{GI}_{\text{SAT}}$: Let G, H be two graphs and let f_G, f_H be the corresponding formulas encoding the graph structure, as in Definition 7. These two formulas are isomorphic iff G, H are isomorphic: the two-variable clauses encode edges and the single-variable clauses encode nodes. The LCNs of f_G, f_H are isomorphic iff the two formulas are isomorphic (Theorem 1).

$\text{GI}_{\text{SAT}} < \text{GI}$: Let $G_f, G_{f'}$ be two LCN*s*. This direction is easy, since G_f and $G_{f'}$ are just two graphs. Note that graph isomorphism between edge-colored graphs can be reduced to graph isomorphism between uncolored graphs by replacing each edge with a special gadget that does not occur anywhere else in the graph, e.g. a K_4 since an LCN is tripartite. Specifically, for each literal-literal edge $\{x, \neg x\}$, remove the edge and add a K_4, connecting x and $\neg x$ to the same vertex in the K_4.

D 3-regular SAT (Proof of Theorem 2)

Recall that a bipartite graph is (a, b)-regular if all nodes in the left partition have degree a and all nodes in the right partition have degree b.

Observation 13 *(a, b)-regular bipartite graphs with n_A and n_B nodes in the partitions are indistinguishable by 1-WL.*

We use the following lemma to manipulate the formula:

Remark 4. $(x \vee y) \iff (x \vee y \vee z) \wedge (x \vee y \vee \neg z)$

Lemma 6 (Theorem 2.1 [32] modified). *For any $\delta \geq 2$, given a 3-SAT formula f with maximum literal degree Δ, there is an equisatisfiable formula f' with maximum literal degree δ. The formula f' has at most $n' = n + 4\Delta n$ variables and $m' = m + 4\Delta n$ clauses.*

Proof. Let x be a variable in f with x or $\neg x$ appearing more than δ times. We break the variable into $s = d(x) + d(\neg x)$ variables $x_1, \dots, x_s$, where $s \leq 2\Delta$. The constraint $x_1 = x_2 = \dots, x_s$ is equivalent to $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge \dots \wedge (x_s \vee \neg x_1)$. We use Remark 4 on each clause to make them 3-regular. The end result uses $2s$ variables and has $2s$ clauses. Each original literal is used once in the constraint clauses, so it can be used $\delta - 1$ times outside of it.

Lemma 7. *Given a 3-SAT formula f where each literal appears in at most 3 clauses, there is an equisatisfiable 3-regular formula f' where each literal is in exactly 3 clauses, with at most $n' = 5n$ variables and $m' = m + 9n$ clauses.*

Proof. Let $L_r \subseteq L$ be the literals in f that appear in exactly 3 clauses. First, note that the number of edges in the literal-clause graph of f is

$$3m = 3|L_r| + \sum_{\ell \in L \setminus L_r} d(\ell)$$

Here, the sum of degrees of non-regular literals must be a multiple of 3. We need to add $s = \sum_{\ell \in L \setminus L_r} 3 - d(\ell) = 3|L \setminus L_r| - \sum_{\ell \in L \setminus L_r} d(\ell)$ connections to make the literals 3-regular. As shown above, s is a multiple of 3, so it is enough to show how to add 3 connections for literals ℓ_1, ℓ_2, ℓ_3 (possibly some of these are equal). We introduce auxillary variables a, b, c, d and set $a = b = c = 1$, while the value of d does not matter. We add the following clauses to form f':

$$\begin{array}{llll}
(\ell_1, a, \neg b), & (a, \neg c, d), & (b, \neg a, d), & (c, \neg b, d), \\
(\ell_2, \neg a, c), & (a, \neg c, \neg d), & (b, \neg a, \neg d), & (c, \neg b, \neg d), \\
(\ell_3, b, \neg c) & & &
\end{array}$$

Note that all 9 clauses are satisfied by either a, b or c. Also, all clauses are unique (even when $\ell_1 = \ell_2 = \ell_3$) and non-trivial (no clauses of type $x \vee \neg x$). All auxillary literals appear exactly three times. The set of solutions of f' projected to the variables of f is the same as the set of solutions for f.

The number of missing connections s is at most $3n$, so the number of added variables and clauses is at most $4n$ and $9n$.

E Distinguishing Random SAT Instances (Proof of Theorem 7)

This section uses the seminal results of [4] on random graph isomorphism. To state our results, we need some related definitions. Let $\mathcal{K}$ be a class of graphs with n vertices. A *canonical labeling algorithm* of $\mathcal{K}$ is an algorithm which assigns numbers $1, \dots, n$ to each graph in $\mathcal{K}$, such that two graphs are isomorphic if and only if the labeled graphs coincide. Note that the labeling must be permutation invariant. Given a canonical labeling, we can test if two graphs are isomorphic in linear time by comparing the edges of the two graphs.

[4] gave a canonical labeling algorithm for isomorphism testing on random graphs:

Theorem 14 ([4]). *There is a class of n-node graphs $\mathcal{K}$ and a linear-time algorithm that decides whether a given graph G belongs to $\mathcal{K}$ and computes a canonical labeling of $\mathcal{K}$. The probability that a uniformly random n-node graph belongs to $\mathcal{K}$ is at least $1 - n^{-1/7}$, for large enough n.*

Proof sketch. In short, their algorithm computes a unique identifier for each node based on adjacency to high-degree nodes. For this to work, the top $r := 3 \log n / \log 2$ degrees must be unique. Assuming unique degrees for nodes $v_1, \dots, v_r$ ordered by degree, the adjacency patterns to these nodes gives an $O(\log n)$-bit identifier x_v to each node v, where $(x_v)_i = \mathbb{1}(v_i \in N(v))$. If the top degrees and the generated IDs are unique, this labeling is returned. Otherwise, the graph does not belong to $\mathcal{K}$. It can be shown that both conditions hold with probability at least $1 - n^{-1/7}$ over all n-node graphs.

It is known that the WL test identifies all graphs in $\mathcal{K}$. For clarity, we add the following lemma to make this formal:

Lemma 8. *The WL test distinguishes any two graphs G, H where $G \in \mathcal{K}$ and H is any graph non-isomorphic to G.*

Proof. The first round labels χ_v^1 of WL partition nodes by degree. The partition given by labels χ_v^2 after the second round is clearly at least as fine as the partition given by x_v in the algorithm of Theorem 14. Hence, if x_v is unique for each node, then so is χ_v^2. Hence, the multiset of second-round labels is different for any $G \in \mathcal{K}$ and $H \notin \mathcal{K}$. In the third round, the unique labels encode all information about edges, so any differences in the adjacency between graphs in $\mathcal{K}$ is detected.

Distinguishing Instances Extracted from Random LIGs Recall the literal-incidence graph representation of a CNF formula, where each literal is a node and two nodes are connected if they appear in the same clause. A principled way of extracting a CNF formula from a random literal-incidence graph is given by [39] (see Lemma 2). We show that a CNF formula constructed this way is likely identified by the WL test.

Theorem 7. *A* CNF *formula extracted from a uniformly random literal-incidence graph with n literals is identified by the WL test with probability at least $1-(n)^{-1/7}$, over the choice of a LIG, for a large-enough n.*

Proof. Let G be the corresponding literal-incidence graph. We show that the LCN is identified if G is identified. Color refinement identifies G, i.e. each node gets a unique identifier (Theorem 14), with probability at least $1-(n)^{-1/7}$.

Consider color refinement on the LCN. We can ignore literal-literal edges – due to their different color, they only add expressivity. In two iterations, the information traveling from literals to literals via clauses on the LCN is exactly the same as in one iteration on G. By construction, the set of two-hop literal neighbors on the LCN (ignoring literal-literal edges) is exactly the same as the set of neighbors in G. Hence, color refinement on the LCN produces unique identifiers for the literals (in at most twice the number of iterations). Since there are no duplicate clauses, clause nodes become identified by the unique set of literals they are connected to. Since all nodes have a unique identifier, the LCN is identified (as in Lemma 8).

F Experimental results

F.1 Details on Instance Generation

The default size of instances in G4SATBench is limited (200-400 variables for hard instances). To generate very large instances with number of variables in the thousands, we adapt the generation script for 3-SAT by slightly reducing the clause-to-variable ratio from the known satisfiability threshold [9] of $\lfloor 4.258 \cdot n_V + 58.26 \cdot n_V^{-2/3} \rfloor$, where n_V is the number of variables. This is known as the threshold number of clauses for 3-SAT instances, where the ratio of satisfiable to unsatisfiable formulas is approximately 50/50, and also where the hardest instances are typically found. To produce very large instances for the 3-SAT family, we change the multiplier from 4.258 to 4.158, which reduces the number of clauses slightly. This enables faster generation of satisfiable instances, while still maintaining approximately the same complexity.

F.2 Full Results

Below are Tables 3 and 4, presenting the full results of our experiments on the 2024 SAT competition and G4SAT benchmark instances.

Table 3: Results on the instances from the 2024 SAT competition. All values are reported as mean ± standard deviation. The WL-expressible (WL-exp) column indicates whether predicting a satisfying assignment is possible with a WL-powerful GNN. r_{crit} denotes the WL iteration where the WL-partition-constrained formula becomes satisfiable. $r_{\text{converged}}$ is the number of iterations for the WL algorithm to converge. The number of variables and clauses is given as average over all instances in the family. For some families, we were only able to test on one example due to the large size of the instances.

Family	WL-exp	r_{crit}	$r_{\text{converged}}$	Variables	Clauses	Count
argumentation	True	2.00	3.00	300	17540	1
battleship	False	-	1.00	364	2562	1
binary-tree-parity	True	22.00	22.00	511	2039	1
circuit-multiplier	False	-	7.00	1013	18793	1
crafted-cec	True	19.00	20.00	30587	91703	1
cryptography	True	28.00 ± 25.40	30.80 ± 23.86	46433 ± 42323	196086 ± 121145	5
	False	-	12.00 ± 0.00	1637 ± 130	19651 ± 2923	2
cryptography-ascon	False	-	54.00 ± 0.00	158772 ± 32	373432 ± 124	2
cryptography-simon	True	17.67 ± 2.73	18.33 ± 2.88	3301 ± 602	10981 ± 2048	6
hamiltonian	True	4.16 ± 0.50	5.47 ± 0.51	553 ± 46	4974 ± 409	19
heule-folkman	False	-	4.91 ± 0.30	16614 ± 1103	20147 ± 1492	11
heule-nol	False	-	8.82 ± 1.40	1419 ± 0	7831 ± 7	11
knights-problem	False	-	25.00	122472	442903	1
maxsat-optimum	True	28.60 ± 2.51	30.20 ± 2.68	23282 ± 6054	277737 ± 121915	5
minimum-disagreement-parity	False	-	8.50 ± 0.93	1040 ± 191	5846 ± 1123	8
polynomial-multiplication	False	-	62.00 ± 38.18	29546 ± 32789	117068 ± 129921	2
pythagorean-triples	True	7.00	7.00	3692	13727	1
quasigroup-completion	True	4.00 ± 0.00	4.00 ± 0.00	20940 ± 5970	436860 ± 149323	2
random-circuits	False	-	5.33 ± 1.15	3115 ± 231	184064 ± 13856	3
random-csp	True	4.00 ± 0.00	4.00 ± 0.00	862 ± 117	43293 ± 26251	2
random-modularity	True	3.00	4.00	2200	9086	1
rbsat	True	4.00 ± 0.00	4.00 ± 0.00	806 ± 93	48089 ± 8880	4
scheduling	True	12.00 ± 2.37	16.33 ± 3.78	16717 ± 3451	67014 ± 12872	6
	False	-	28.29 ± 14.26	31646 ± 21671	162269 ± 114314	7
sgen	False	-	1.00	180	432	1
stedman-triples	True	9.00	10.00	2345	61876	1
subgraph-isomorphism	True	4.00	5.00	1285	166723	1
summle	False	-	32.00 ± 0.00	94160 ± 17675	199503 ± 34558	3
tree-decomposition	False	-	42.00	74804	393322	1
unknown	False	-	8.00	1385	27650	1

Table 4: Results on the instances from the G4SAT benchmark. Instances are divided by family and hardness (which is based on size). All instances were initially satisfiable. All values are reported as mean ± standard deviation. The WL-expressible column indicates whether predicting a satisfying assignment is possible with a WL-powerful GNN. r_{crit} denotes the WL iteration where the WL-partition constrained formula becomes satisfiable. $r_{\text{converged}}$ is the number of iterations for the WL algorithm to converge. On average, a few iterations is enough for WL to sufficiently distinguish literals, except for a few outlier instances in the k-clique, k-vertex cover and ca families. The PS family is omitted due to problems with the generation script.

family	difficulty	sat	r_{crit}	$r_{\text{converged}}$	Variables	Clauses	Count
3-sat	easy	True	2.97 ± 0.18	3.68 ± 0.47	26 ± 9	119 ± 36	1000
	medium	True	3.00 ± 0.04	3.92 ± 0.28	119 ± 47	509 ± 198	1000
	hard	True	3.00 ± 0.00	4.00 ± 0.00	250 ± 29	1065 ± 125	100
	hard+	True	3.00 ± 0.00	4.00 ± 0.00	921 ± 48	3775 ± 196	24
	hard++	True	3.08 ± 0.28	4.00 ± 0.00	5001 ± 62	20504 ± 256	25
k-clique	easy	True	4.12 ± 0.73	6.26 ± 0.83	33 ± 13	543 ± 426	960
		False	-	6.00 ± 0.78	22 ± 7	217 ± 194	40
	medium	True	4.11 ± 0.52	6.33 ± 0.95	68 ± 17	2156 ± 960	999
		False	-	6.00	45	939	1
	hard	True	4.00 ± 0.00	6.00 ± 0.00	114 ± 20	5554 ± 1718	100
k-domset	easy	True	4.11 ± 0.71	5.61 ± 0.93	39 ± 12	329 ± 186	1000
	medium	True	4.33 ± 0.58	5.50 ± 0.72	88 ± 18	1647 ± 687	1000
	hard	True	4.42 ± 0.67	5.54 ± 0.69	137 ± 22	3986 ± 1315	100
k-vercov	easy	True	4.75 ± 1.14	6.18 ± 1.38	39 ± 13	358 ± 245	993
		False	-	4.00 ± 0.00	26 ± 8	159 ± 108	7
	medium	True	4.94 ± 1.00	5.88 ± 1.08	96 ± 20	2052 ± 936	1000
	hard	True	5.00 ± 1.01	5.80 ± 0.85	179 ± 25	7198 ± 2159	100
sr	easy	True	2.00 ± 0.05	3.00 ± 0.06	25 ± 9	146 ± 54	1000
	medium	True	2.01 ± 0.12	3.00 ± 0.00	118 ± 47	644 ± 249	1000
	hard	True	2.05 ± 0.22	3.00 ± 0.00	299 ± 62	1613 ± 343	100