Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"
Consiglio Nazionale delle Ricerche

ISTI Technical Reports

MILP-SAT-GNN: yet another neural SAT solver

Giuseppe Riccardo Leone, ILC-CNR, Pisa, Italy
Hamza Khyari, USMBA-LSIA, Fez, Morocco
Umberto Straccia, ISTI-CNR, Pisa, Italy

ISTI-TR-2025/020

Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"
Consiglio Nazionale delle Ricerche

MILP-SAT-GNN: yet another neural SAT solver
Franco Alberto Cardillo, Hamza Khyari, Umberto Straccia
ISTI-TR-2025/018

Abstract
We proposes a novel method that enables Graph Neural Networks (GNNs) to solve SAT problems by leveraging a technique developed for applying GNNs to Mixed Integer Linear Programming (MILP). Specifically, k-CNF formulae are mapped into MILP problems, which are then encoded as weighted bipartite graphs and subsequently fed into a GNN for training and testing. From a theoretical perspective: (i) we establish permutation and equivalence invariance results, demonstrating that the method produces outputs that are stable under reordering of clauses and variables; (ii) we identify a theoretical limitation, showing that for a class of formulae called foldable formulae, standard GNNs cannot always distinguish satisfiable from unsatisfiable instances; (iii) we prove a universal approximation theorem, establishing that with Random Node Initialization (RNI), the method can approximate SAT solving to arbitrary precision on finite datasets, that is, the GNN becomes approximately sound and complete on such datasets. Furthermore, we show that for unfoldable formulae, the same approximation guarantee can be achieved without the need for RNI. Finally, we conduct an experimental evaluation of our approach, which show that, despite the simplicity of the neural architecture, the method achieves promising results.

Keywords: Computer Science – Learning, Computer Science - Artificial Intelligence.

Citation
Cardillo F.A., Khyari H., Straccia U. *MILP-SAT-GNN: yet another neural SAT solver*. ISTI Technical Reports 2025/020. DOI: 10.32079/ISTI-TR-2025/020.

Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"
Area della Ricerca CNR di Pisa
Via G. Moruzzi 1
56124 Pisa Italy
http://www.isti.cnr.it

arXiv:2507.01825v1 [cs.LG] 2 Jul 2025

MILP-SAT-GNN: YET ANOTHER NEURAL SAT SOLVER

FRANCO ALBERTO CARDILLO[1], HAMZA KHYARI[2], AND UMBERTO STRACCIA[3]

[1]CNR-ILC, PISA, ITALY
[2]USMBA-LSIA, FEZ, MOROCCO
[3]CNR-ISTI, PISA, ITALY

Abstract. We proposes a novel method that enables Graph Neural Networks (GNNs) to solve SAT problems by leveraging a technique developed for applying GNNs to Mixed Integer Linear Programming (MILP). Specifically, k-CNF formulae are mapped into MILP problems, which are then encoded as weighted bipartite graphs and subsequently fed into a GNN for training and testing. From a theoretical perspective: (i) we establish permutation and equivalence invariance results, demonstrating that the method produces outputs that are stable under reordering of clauses and variables; (ii) we identify a theoretical limitation, showing that for a class of formulae called foldable formulae, standard GNNs cannot always distinguish satisfiable from unsatisfiable instances; (iii) we prove a universal approximation theorem, establishing that with Random Node Initialization (RNI), the method can approximate SAT solving to arbitrary precision on finite datasets—that is, the GNN becomes approximately sound and complete on such datasets. Furthermore, we show that for unfoldable formulae, the same approximation guarantee can be achieved without the need for RNI. Finally, we conduct an experimental evaluation of our approach, which show that, despite the simplicity of the neural architecture, the method achieves promising results.

1. Introduction

The *Boolean Satisfiability Problem* (SAT), and particularly its restricted variant 3-SAT (see definition later on), stands as a fundamental challenge in computer science, underpinning numerous applications across formal verification, artificial intelligence, theory of computation, and optimization (see, e.g. [11]). While decades of research have yielded highly sophisticated SAT solvers based on conflict-driven clause learning, heuristics, and systematic search, these solvers often face difficulties when tackling large-scale, structurally complex, or adversarially generated instances (e.g. designing heuristics remains a highly non-trivial and time-consuming task). In parallel, the field of machine learning, and in particular *Graph Neural Networks* (GNNs) [27, 36], has demonstrated remarkable capacity to model and reason about structured, relational data, offering a promising opportunity to enhance or even replace modern SAT solvers. However, applying GNNs directly to SAT presents some challenges: SAT formulae have rich, logical structure that is difficult to fully capture with standard GNN architectures. Also, GNNs, due to their inherent limitations may fail to distinguish between critical differences in satisfiability status. Furthermore, achieving soundness (if the solver's decision is SAT then the satisfiability status is indeed SAT) and completeness (if the satisfiability

status is SAT then the solver's decision is SAT) is particularly difficult in learned settings.

Related Work. The use of GNNs for SAT solving is not novel as documented by the GNN SAT solvers that have been proposed in the literature (see [13, 16, 20], for an overview). Existing GNN-based SAT solvers can be broadly categorized into two branches: *stand-alone neural solvers* and *neural-guided solvers*. Stand-alone neural solvers utilize GNNs to solve SAT instances directly. For instance, [38, 5, 39] focuses on predicting the satisfiability of a given formula, while several alternative approaches [3, 32, 19] aim to construct a satisfying assignment. Neural-guided solvers, on the other hand, integrate GNNs with modern SAT solvers, trying to improve e.g. their search heuristics with the prediction of GNNs. In this category fall methods such as [21, 37, 44, 45, 47]. Eventually, in [20] a benchmark dataset has been proposed to evaluate some of these GNN models. **Contribution.** Our contribution falls in the former stand-alone neural solvers category and is as follows. We propose a new method that enables GNNs to solve SAT problems by leveraging a technique developed for applying GNNs to Mixed Integer Linear Programming (MILP). Specifically, k-CNF formulae are mapped into MILP problems, which are then encoded as weighted bipartite graphs and subsequently fed into a GNN for training and testing. We believe that, having an underlying MILP representation, may allow us, at least in theory (with appropriate and more involved transformations), to deal with some *many-valued* [14], paraconsistent [2], probabilistic and/or logics combining logical constraints with numerical constraints as well [23, 26, 34], opening up to new perspectives of GNNs usages.

From a theoretical perspective: (i) we establish permutation and equivalence invariance results, demonstrating that the method produces outputs that are stable under reordering of clauses and variables; (ii) we prove a universal approximation theorem, establishing that with *Random Node Initialization* (RNI) [1, 12, 35], the method can approximate SAT solving to arbitrary precision on finite datasets—that is, the GNN becomes approximately sound and complete on such datasets. Furthermore, we show that for unfoldable formulae, the same approximation guarantee can be achieved without the need for RNI; and (iii) finally, we have implemented our method and have conducted an experimental evaluation, demonstrating that, despite the simplicity of the neural architecture, the method achieves promising results.

In the following, we proceed as follows. In the next section, we introduce the background notions we will rely on. In Section 3 we illustrate our method and theoretical results, while in Section 4 we address the experimental validation. Section 5 summarise our contribution and illustrates topics for future work.

2. Background

For readers not be familiar with both CNFs and GNNs, we provide here a succinct summary of the basic notions we rely on. The informed reader may just skip a part and come back if needed.

CNFs. Let $\Sigma = \{p_1, p_2, \ldots, p_n\}$ be a finite non-empty set of n propositional *letters* (or *variables*). Let $\mathcal{L}_\Sigma = \{\alpha, \beta, \ldots\}$ be the set of propositional formulae based on the set of Boolean operators $\{\neg, \wedge, \vee\}$ and Σ inductively defined as follows: (i) every propositional letter is a formula; and (ii) if α, β are formulae, so are $\neg\alpha, \alpha \wedge \beta$ and $\alpha \vee \beta$. The Boolean operator $\neg$ (resp. $\wedge, \vee, \rightarrow, \leftrightarrow$ is called the

negation (resp. *conjunction, disjunction*) operator. A *literal* (denoted l) is either a propositional letter p_i (called *positive literal*) or its negation $\neg p_i$ (called *negative literal*) and with $\bar{l}$ we denote $\neg p_i$ (resp. p_i) if $l = p_i$ (resp. if $l = \neg p_i$). A *clause* is a (finite) disjunction of literals, while a *k-clause* is a clause of k literals ($1 < k \leq n$). In a clause no letter may occur more than once. A formula is in *Conjunctive Normal Form* (CNF) if it is a (finite) conjunction of clauses. It is a k-CNF formula if it is a (finite) conjunction of k-clauses. All clauses in a CNF formula are assumed to be distinct. A k-CNF formula ϕ of m clauses $\phi = \bigwedge_{i=1}^{m} C_i$, with $C_i = \bigvee_{j=1}^{k} l_{ij}$ can also be represented as a *set of sets*, by representing each k-clause as a set of literals and the conjunction of clauses as the set of clauses represented as sets. That is, $\phi = \{C_1, \ldots, C_m\}$, with $C_i = \{l_{i1}, \ldots, l_{ik}\}$. Note that there are at most $k_{\mathtt{CNF}} = 2^k \binom{n}{k}$ distinct k-clauses and, thus, any k-CNF formula can not have more than $k_{\mathtt{CNF}}$ k-clauses. With $\mathcal{L}^k_{\mathtt{CNF}}$ we denote the set of all k-CNF formulae, while with $\mathcal{L}^k_{\mathtt{CNF}}(n, m)$ we denote k-CNF formulae having at most n variables and m clauses. Given a formula ϕ, with $\Sigma_\phi \subseteq \Sigma$ we denote the set of propositional letters occurring in ϕ. An *interpretation*, or *world*, w w.r.t. Σ is a set of literals such that all propositional letters in Σ occur exactly once in w. We may also denote a world w as the concatenation of the literals occurring in w by replacing a negative literal $\neg p$ with $\bar{p}$ (e.g. $\{\neg p_1, p_2\}$ may be denoted also as $\bar{p}_1 p_2$). With $w \Vdash \phi$ we indicate that the world w *satisfies* the formula ϕ, i.e. w is a *model* of ϕ, which is inductively defined as usual: $w \Vdash l$ iff $l \in w$, $w \Vdash \neg\phi$ iff $w \nVdash \phi$, $w \Vdash \phi \wedge \psi$ iff $w \Vdash \phi$ and $w \Vdash \psi$ and $w \Vdash \phi \vee \psi$ iff $w \Vdash \phi$ or $w \Vdash \psi$. Furthermore, ϕ is *satisfiable* (resp. *unsatisfiable*) if it has (resp. has no) model. The *k-SAT* problem consists in deciding whether a k-CNF formula is satisfiable or not and is known to be a NP-complete problem for $k \geq 3$ [7].[1] A *dataset* is a finite set of formulae-label pairs ($s \geq 2$) $\mathcal{D} = \{\langle \phi_1, y_1 \rangle, \ldots, \langle \phi_s, y_s \rangle\}$, such that $y_i \in \{0, 1\}$ and $y_i = 1$ iff ϕ satisfiable.[2] A *CNF* (resp. k-CNF) *dataset* is a dataset of CNF (resp. k-CNF) formulae.

Graph Neural Networks. *Graph Neural Networks* [27, 36] (GNNs) are deep learning architectures for graph structured data (see, e.g. [15, 18]) for a recent overview). In general terms, each data point consists of the structure of a graph represented by the adjacency matrix and properties of the graphs. The properties of these graphs are stored in node feature vectors, and can sometimes include edge feature vectors.

Formally, a *graph* is a tuple $G = \langle V, E, \mathbf{X} \rangle$, where $V = \{v_1, \ldots, v_{|V|}\}$ is the set of nodes, $E = \{e_1, \ldots, e_{|E|}\}$ is the set of edges, and the edge $e = \langle v_i, v_j \rangle \in E$ represents the connection relationship between nodes v_i and v_j in the graph. $\mathbf{X} \in \mathbb{R}^{|V| \times M}$ is the node feature matrix with M being the dimension of each node feature. With $\mathbf{x}_v$ we denote the feature vector of v.[3] The adjacency matrix of a graph can be defined as $\mathbf{A} \in \mathbb{R}^{|V| \times |V|}$, where $\mathbf{A}_{ij} = 1$ if $\langle v_i, v_j \rangle \in E$, otherwise $\mathbf{A}_{ij} = 0$. We use $N(v)$ to denote the *neighbours* of a node v and with d_v its degree. The graph G is

[1] The k-SAT problem can be solved by a nondeterministic Turing machine in polynomial time.

[2] Ideally, we would like to have that there is no pair involving equivalent formulae, i.e. $\phi_i \not\equiv \phi_j$ for each $1 \leq i < j \leq s$ and, thus, $|\mathcal{D}| \leq 2^{2^n}$. However, for large size dataset, e.g. in the order of 10^5, producing such a dataset is unfeasible (time consuming) and, we are not going to consider this further constraint.

[3] Edges may also have an associated feature vector and being embedded, but for ease of representation we do not consider this case here.

undirected if $\mathbf{A}$ is *symmetric*, i.e. $\mathbf{A}_{ij} = \mathbf{A}_{ji}$ for all $1 \leq i, j \leq n$. G has a *self-loop* if $\mathbf{A}_{ii} = 1$ for some $1 \leq i \leq n$. The *order* of a graph is the number of nodes in it.

The adjacency matrix $\mathbf{A}$ of a graph G is often highly sparse, making it inefficient for storage, computation, and machine learning tasks. To address this, *graph embedding* techniques—specifically *node embeddings*—have been developed [18] to reduce dimensionality by learning a compact vector representation $\mathbf{h}_v \in \mathbb{R}^d$ for each node $v \in V$, where $d \ll |V|$.

Many different versions of the GNNs have been considered in the literature (see, e.g. [18]). A somewhat standard and core model is the one that goes under the name *Message Passing Neural Network* (MPNN) [10] or *Aggregate-Combine Graph Neural Network* (AC-GNN) [4]. AC-GNN are also called *1-dimensional GNNs* in [31].

It consists of a sequence of layers that iteratively combine the representation vectors of every node with the multiset of representation vectors of its neighbours. Formally, consider a sequence $\langle L_1, ..., L_l \rangle$ of layers L_k. L_1 is called the *input layer*, $L_2, ..., L_{l-1}$ the *hidden layers* and L_l the *output layer*. Let $\texttt{agg}$ and $\texttt{comb}$ be *aggregation* and *combination* functions. The function $\texttt{agg}$ aggregates the information from the neighbours of each node, while $\texttt{comb}$ updates the node representations by combining the aggregated information from neighbours with the current node representations. We assume that each node v has an associated initial embedding vector $\mathbf{h}_v^1 = f_{\texttt{in}}(\mathbf{x}_v)$, where $f_{\texttt{in}}$ is a learnable function. A GNN computes a vector $\mathbf{h}_v^k$, called the node representation or *node embedding* of v at the k-th layer, via the following recursive formula $(1 \leq k < l)$:[4]

$$\mathbf{h}_v^{k+1} \quad = \quad \texttt{comb}(\mathbf{h}_v^k, \texttt{agg}(\{\!\{\mathbf{h}_w^k \mid w \in N(v)\}\!\})) \tag{1}$$

The node representations h_v^l in the last layer can be treated as the final node representations. Then, the vectors h_v^l of the last layer L_l are then *pooled* together via

$$\hat{\mathbf{y}} \quad = \quad \texttt{pool}(\{\!\{\mathbf{h}_v^l \mid v \in V\}\!\}) \tag{2}$$

to give a single graph vector $\hat{\mathbf{y}}$, the output of the GNN.[5]

In order to translate the abstract GNN framework defined above into something we can implement, we must give concrete instantiations to these update and aggregate functions. The most basic GNN framework, which is a simplification of the original GNN models proposed in [27, 36], is as follows $(1 \leq k < l)$:

$$\begin{array}{rcl} \mathbf{h}_v^1 & = & f_{\texttt{in}}(\mathbf{x}_v) \\ \mathbf{h}_v^{k+1} & = & \sigma(\mathbf{W}_{self}^k \mathbf{h}_v^k + \mathbf{W}_{neigh}^k \sum_{w \in N(v)} \mathbf{h}_w^k + \mathbf{b}_v^k) \\ \hat{\mathbf{y}}_v & = & f_{\texttt{out}}(\mathbf{h}_v^l) \ , \text{ for all } v \in V \ , \end{array} \tag{3}$$

where $\mathbf{W}_{self}^k, \mathbf{W}_{neigh}^k \in \mathbb{R}^{d \times d}$ are learnable weight matrices, $f_{\texttt{out}}$ is a learnable function and σ is a non-linear *activation function*. Commonly used activation functions include the sigmoid, tanh, and ReLU functions.[6] The bias term $\mathbf{b}_v^k \in \mathbb{R}^d$ is often omitted for notional simplicity, but including the bias term can be important to

[4] $\{\!\{\cdot\}\!\}$ denotes a multiset.

[5] The $\texttt{agg}$, $\texttt{comb}$ and $\texttt{pool}$ functions may also take into account the feature vectors of the involved nodes, but for ease of presentation we do not consider it here.

[6] https://en.wikipedia.org/wiki/Activation_function.

improve effectiveness. Many more involved GNNs model haven been developed so far. We refer the reader to e.g. [15] for more insights on GNNs.

In the context of GNNs, two tasks are typically addressed. Informally, the *node classification problem* is the task of predicting the label y_v of each node v in a graph, while the *graph classification problem* is the task of predicting the label y_G of a graph G, where a label may be a categorical class (binary classification or multiclass classification), or a continuous number (regression). We anticipate that, in this work, we are considering the binary graph classification problem only in which the output of a GNN is always a value in $\{0, 1\}$.[7] Specifically, a *graph dataset* consists of set of graph-label pairs ($s \geq 1$) $\mathcal{D} = \{\langle G_1, y_1\rangle, \ldots, \langle G_s, y_s\rangle\}$, where $y_i \in \{0, 1\}$. From such a dataset of labelled graphs (the *training set*), $\langle G_i, y_i\rangle$, with $y_i \in \{0, 1\}$, we would like to induce (learn) a function $f_{\texttt{GNN}} : \mathcal{G} \to \{0, 1\}$ that given an unlabelled graph $G \in \mathcal{G}$ as input predicts its label $y_G \in \{0, 1\}$. To do so, in general one tries learn the parameters of the network, e.g. the weights and bias in Eq. 3, by minimising a *loss function*. That is, to learn a function $f_{\texttt{GNN}}$ from $\mathcal{D}$ such that it minimises the *loss* $L = \sum_{G_i \in \mathcal{D}} loss(\hat{y}_i, y_i)$, where $\hat{y}_i$ is the output of $f_{\texttt{GNN}}$ applied to G_i and y_i is the so-called *the ground truth label* of G_i and *loss* quantifies the disagreement between $\hat{y}$ and y (with $loss(\hat{y}, y) = 0$ if $\hat{y} = y$).

3. MILP-based GNNs for k-SAT

As anticipated, we leverage on the recently introduced method to learn GNNs that solve *Mixed Integer Linear Programming* (MILP) problems [6][8] to decide whether a k-CNF formula is satisfiable or not. Specifically, k-CNF formulae are mapped into MILP problems, which are then encoded as weighted bipartite graphs and subsequently fed into a GNN for training and testing.

3.1. From k-CNFs to MILPs.

We recall that a general MILP problem M is defined as follows:

$$\min \mathbf{c}^\intercal \mathbf{x} \text{ s.t. } \mathbf{A}\mathbf{x} \geq \mathbf{b}\ , \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}\ , x_j \in \mathbb{Z}, \forall j \in I\ , \tag{4}$$

where $\mathbf{c}^\intercal \mathbf{x} = \sum_{i=1}^{n} c_i \cdot x_i$ is the *objective function* to be minimised; $\mathbf{x} \in \mathbb{R}^n$ are the *variables*; $\mathbf{c} \in \mathbb{R}^n$ are the *coefficients* of the variables in the objective function; $\mathbf{A} \in \mathbb{R}^{m \times n}$ is a matrix of coefficients such that $\mathbf{A}\mathbf{x}$ is the set of linear *constraints* bounded below by $\mathbf{b} \in \mathbb{R}^m$, which is a vector of coefficients; $\mathbf{l}, \mathbf{u} \in (\mathbb{R} \cup \{-\infty, +\infty\})^n$ express lower and upper bounds of the variables; and the index set $I \subseteq \{1, 2, \ldots, n\}$ denotes those indices j where the x_j are constrained to be an integer.
The *feasible set* of a MILP M is defined as $M_{\texttt{feasible}} = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{A}\mathbf{x} \geq \mathbf{b}, \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, x_j \in \mathbb{Z}, \forall j \in I\}$.

We say that M is *feasible* if $M_{\texttt{feasible}} \neq \emptyset$, and *unfeasible* otherwise. It is well-known that the feasibility problem, like the k-SAT problem, is a NP-complete problem as well. In this work we are only interested in the feasibility/unfeasibility problem and, thus, the minimisation of the objective function is omitted from a

[7] The output 1 will denote that a formula is satisfiable, while 0 will denote that the formula is unsatifiable.

[8] We refer the reader e.g. to [46] for an overview on methods using machine learning in the context of MILP.

MILP. We also recall that the *MPS* file format[9] is a standard way of representing and archiving MILP problems.

Now, consider a k-CNF formula ϕ represented as set of sets. We transform now the k-SAT problem of ϕ into a feasibility problem of a MILP M_ϕ. The transformation follows a well-known method in transforming propositional formulae into a set of linear equations (see, e.g. [14, 17]). Specifically, for each propositional letter p_i occurring in ϕ, consider a binary integer variable x_i. The intuition here is that $x_i = 1$ (resp. $x_i = 0$) means that p_i is true (resp. p_i is false). For a literal l, we use the following transformation

$$\tau(l) \quad = \quad \begin{cases} x_i & \text{if } l = p_i \\ 1 - x_i & \text{if } l = \neg p_i \ . \end{cases} \tag{5}$$

A clause $C_i = \{l_{i1}, \ldots, l_{ik}\}$ is transformed into the linear formula

$$\tau(C_i) = \sum_{j=1}^{k} \tau(l_{ij}) \ . \tag{6}$$

Eventually, the formula ϕ is translated into the MILP M_ϕ:

$$\tau(C_1) \geq 1 \ , \ldots , \tau(C_m) \geq 1 \ , x_i \in \{0,1\}, \forall p_i \in \Sigma_\phi \ , \tag{7}$$

which we may succinctly represent as the MILP M_ϕ ($n = |\Sigma_\phi|$)

$$\mathbf{A}_\phi \mathbf{x}_\phi \geq \mathbf{b}_\phi \ , \mathbf{x}_\phi \in \{0,1\}^n \ , \tag{8}$$

where $\mathbf{x}_\phi \in \{0,1\}^n$ is the vector of variables related to the propositional letters occurring in ϕ, $\mathbf{A}_\phi \in \{-1, 0, 1\}^{m \times n}$ and $\mathbf{b}_\phi \in \{1-k, \ldots, -1, 0, 1\}^m$ are respectively, the matrix and vector of integers resulting from the encoding of the linear expressions as from Eq. 6. That is, for $1 \leq i \leq m$, $b_i = 1 - neg_i$, where neg_i is the number of negative literals in C_i and, thus, $0 \leq neg_i \leq k$.

The following is an easy provable result:

Proposition 1. *Consider a k-CNF formula ϕ and let M_ϕ its transformation M_ϕ into MILP according to Eq. 8. Then ϕ is satisfiable iff M_ϕ is feasible.*

Example 1. *Consider the 2-CNF (Krom) formula $\phi = (p_1 \vee p_2) \wedge (p_1 \vee \neg p_2) \wedge (\neg p_1 \vee p_2)$, which is equivalent to $p \wedge q$, i.e. $\phi \equiv p_1 \wedge p_2$. ϕ's set of set representation is $\phi = \{\{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_2\}\}$. It can easily be verified that the MILP transformation M_ϕ, consisting of two variables and three constraints is $x_1 + x_2 \geq 1, x_1 - x_2 \geq 0, -x_1 + x_2 \geq 0, x_1, x_2 \in \{0,1\}$ and, thus, we have*

$$\mathbf{A}_\phi = \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ -1 & 1 \end{bmatrix} \ , \ \mathbf{b}_\phi = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \ . \tag{9}$$

[9] https://lpsolve.sourceforge.net/5.5/mps-format.htm.

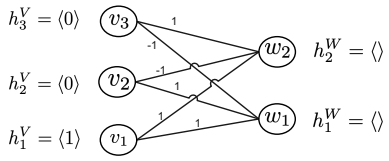


FIGURE 1. Bipartite graph representation of the MILP in Example 1.

3.2. **From MILPs to GNNs.** Now, we adapt the GNN method to solve MILP problems proposed in [6, 9], to our specific case. It is based on a MILP representation in terms of weighted bipartite graphs, which is a quite typical approach when using MILP in connection with GNNs (see [46] for an overview). Specifically, a MILP problem M is represented as an undirected *Weighted bipartite graph* $G = (V \cup W, E, \mathbf{X})$ with nodes in $V \cup W$ partitioned into two groups $V = \{v_1, \ldots, v_m\}$ and $W = \{w_1, \ldots, w_n\}$, with $V \cap W = \emptyset$, and E is a set of weighted edges where each edge has one end in V and one end in W. E can also be seen as a function $E : V \times W \to \mathbb{R}$.

Consider a finite dataset $\mathcal{D}$ of formulae in $\mathcal{L}^k_{\texttt{CNF}}(n, m)$ for fixed $n, m \in \mathbb{N}$ (i.e. k-CNF formulae of maximal n variables and m clauses). Consider $\phi \in \mathcal{D}$ and its MILP transformation M_ϕ (Eq. 8), its weighted bipartite representation $G_\phi = (V_\phi \cup W_\phi, E_\phi, \mathbf{X}_\phi)$, is as follows:

(1) W_ϕ is a set of nodes w_j representing variable x_j occurring in M_ϕ (i.e. propositional letter p_j occurs in ϕ). We call w_j a *variable node*;
(2) V_ϕ is a set of nodes v_i representing the i-th constraint occurring in M_ϕ. We call v_i a *constraint node*;
(3) the edge connecting node constraint v_i and variable node w_j has weight $E_{ij} = (\mathbf{A}_\phi)_{ij}$. Essentially, there is an edge $\langle v_i, w_j \rangle$ if the variable x_j occurs in the i-th constraint (i.e. $(\mathbf{A}_\phi)_{ij} \neq 0$) and the weight of this edge is the value $(\mathbf{A}_\phi)_{ij}$;
(4) the feature vector $\mathbf{x}_{v_i}$ of constraint node v_i is $\mathbf{b}_i$ (so, has dimension 1);
(5) the feature vector $\mathbf{x}_{w_j}$ of variable node w_j is $\langle \rangle$ (so, has dimension 0).

We call the graph G_ϕ so constructed a *MILP-graph*. Note that the degree of G_ϕ is at most $n + m$.

Example 2 (Example 2 cont.)**.** *The bipartite graph representation of the MILP problem in Example 1 is shown in Fig. 1. It has two variable nodes w_1, w_2 and three constraint nodes v_1, v_2, v_3. There is an edge $\langle v_i, w_j \rangle$ if $(\mathbf{A}_\phi)_{ij} \neq 0$ and the weight of this edge is $(\mathbf{A}_\phi)_{ij}$. Each constraint node v_i has a feature of dimension 1, which is $\langle b_i \rangle$.*

Now, the GNNs in this paper always take an MILP-Graph as input, having at most n variable nodes and at most m constraint nodes, and the output is a single real number used to predict the property of the whole graph (cf. the case of graph classification), i.e. whether the underlying MILP problem (resp. k-CNF formula) is feasible (resp. satisfiable).

Remark 1. *Another case would be that each node in W has an output and, thus, the GNN has an output in $\mathbb{R}^n$, which is used to predict the property of each variable (cf. the case of node classification). This may used in case we would like to predict*

also the model of a k-CNF formula, i.e. the truth of the variables in that model (if it exists). We will leave this task for future work.

Now we define the GNN structure precisely, which consist of the following steps. For a sequence $\langle L_1, ..., L_l \rangle$ of layers L_k:

Input layer:: the initial mapping at the input layer L_1 ($k = 1$) consists of an encoding of input feature $\mathbf{x}_i^V$ (resp. $\mathbf{x}_j^W$) of the i-th constraint node (resp. j-variable node) into the embedding space by means of learnable function $f_{\mathtt{in}}^V$ (resp. $f_{\mathtt{in}}^W$), where $1 \leq i \leq m, 1 \leq j \leq n$:

$$\mathbf{h}_i^{1,V} = f_{\mathtt{in}}^V(\mathbf{x}_i^V) \ , \mathbf{h}_j^{1,W} = f_{\mathtt{in}}^W(\mathbf{x}_j^W) \ ,$$

where $\mathbf{h}_i^{1,V}, \mathbf{h}_j^{1,W} \in \mathbb{R}^d$ are initial embedded constraint and variable nodes features and d is their dimension;

Hidden layer:: the hidden layers L_k ($1 < k < l$), are updated via learnable functions $f_k^V, f_k^W \colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ and $g_k^V, g_k^W \colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ as follows ($1 \leq i \leq m, 1 \leq j \leq n$) :

$$\mathbf{h}_i^{k+1,V} = g_{k+1}^V(\mathbf{h}_i^{k,V}, \sum_{j=1}^{n} E_{ij} f_{k+1}^W(\mathbf{h}_i^{k,W})) \tag{10}$$

$$\mathbf{h}_j^{k+1,W} = g_{k+1}^W(\mathbf{h}_i^{k,W}, \sum_{i=1}^{m} E_{ij} f_{k+1}^V(\mathbf{h}_j^{k,V})) \ . \tag{11}$$

Output layer:: for the (single) graph-level output, we use a learnable function $f_{\mathtt{out}} \colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ that maps the variable node embedding space of the last layer L_l into a value $\hat{y}_G \in \mathbb{R}$ as follows:

$$\hat{y}_G = f_{\mathtt{out}}(\sum_{i=1}^{m} \mathbf{h}_i^{l,V}, \sum_{j=1}^{n} \mathbf{h}_j^{l,W}) \ . \tag{12}$$

All functions in the so defined GNN, namely $f_{\mathtt{in}}^V$, $f_{\mathtt{in}}^W$, $f_{\mathtt{out}}$, $\{f_k^V, f_k^W, = g_k^V, g_k^W\}_{k=2}^{l-1}$ are learnable functions. Moreover, please note that, in our the current implementation, there is no update of edge features, which remain constant to their initial weight.

We anticipate that we will classify a formula ϕ as satisfiable iff $\hat{y}_{G_\phi} > 0.5$, that is the output of the learned GNN applied to the bipartite input graph G_ϕ is greater than 0.5. We call the so obtained model MILP-SAT-GNN. We also define $\mathcal{F}_{\mathtt{GNN}}^{n,m}$ as the set of all mappings $\mathcal{F}_{\mathtt{GNN}}^{n,m} = \{F \colon \mathcal{L}_{\mathtt{CNF}}^k(n,m) \to \mathbb{R} \mid F(\phi) = \hat{y}_{G_\phi}, \forall \phi \in \mathcal{L}_{\mathtt{CNF}}^k(n,m)\}$, that is the set of mappings F that take as input a k-CNF formula $\phi \in \mathcal{L}_{\mathtt{CNF}}^k(n,m)$ (as set of sets); transform ϕ into the MILP problem M_ϕ; represent M_ϕ as weighted bipartite graph G_ϕ; give G_ϕ as input to the GNN; and has $\hat{y}_{G_\phi} \in \mathbb{R}$ as output.
Note that G_ϕ's order is at most $n + m$.

3.3. **Some Properties.** In what follows, let $\Phi_{\mathtt{sat}}^{n,m}$ be the classification function $\Phi_{\mathtt{sat}}^{n,m} \colon \mathcal{L}_{\mathtt{CNF}}^k(n,m) \to \{0,1\}$, where $\Phi_{\mathtt{sat}}^{n,m}(\phi) = 1$ if the k-CNF formula $\phi \in \mathcal{L}_{\mathtt{CNF}}^k(n,m)$ is satisfiable, $\Phi_{\mathtt{sat}}^{n,m}(\phi) = 0$, otherwise. We are going now to address the following question: does there exist $F \in \mathcal{F}_{\mathtt{GNN}}^{n,m}$ that well approximates $\Phi_{\mathtt{sat}}^{n,m}$? Proposition 7

later on will provide an answer to this question. But, beforehand we address some properties.

Invariance. Consider $\phi \in \mathcal{L}^k_{\texttt{CNF}}(n,m)$. Let $\Sigma_\phi = \{p_1, ..., p_n\}$ be the alphabet of propositional letters occurring in ϕ, define $S_m = \{1, \ldots, m\}$ and $S_n = \{1, \ldots, n\}$. Let $\mathcal{S}_m$ be the set of permutations τ^c on S_m and let $\mathcal{S}_n$ be the set of permutations τ^p on S_n.[10] Essentially, a permutation τ^c is a rearrangement of the clauses in ϕ, i.e. each clause C_i is replaced with $C_{\tau^c(i)}$ $(1 \leq i \leq m)$, while τ^p is a rearrangement of the propositional letters occurring in ϕ, i.e. each propositional letter p_j occurring in ϕ is replaced with $p_{\tau^p(j)}$ $(1 \leq i \leq n)$. With $\tau^c(\phi)$ (resp. $\tau^p(\phi)$) we denote the k-CNF formula obtained from ϕ by applying τ^c (resp. τ^p) to ϕ in the obvious way. Moreover, for a world (interpretation) w over Σ_ϕ, we denote with $\tau^p(w)$ the interpretation obtained from w by applying τ^p to w (i.e., each propositional letter p_j occurring in w is replaced with $p_{\tau^p(j)}$). If $W \subseteq W_\Sigma$ is a set of worlds, then $\tau^p(W) = \{\tau^p(w) \mid w \in W\}$.

Now, define $\mathcal{S}^{m\times n} = \mathcal{S}_m \times \mathcal{S}_n$ and consider $\tau = \langle \tau^c, \tau^p \rangle \in \mathcal{S}^{m\times n}$. With $\tau(\phi)$ we denote the k-CNF formula ψ obtained from ϕ by applying τ^c and τ^p to ϕ. Obviously, $\tau(\phi) \in \mathcal{L}^k_{\texttt{CNF}}(n,m)$. The following proposition can be shown.

Proposition 2. *Consider $\phi \in \mathcal{L}^k_{\texttt{CNF}}(n,m)$ and let $\tau = \langle \tau^c, \tau^p \rangle \in \mathcal{S}^{m\times n}$. Then the following hold: (i) w is a model of ϕ iff w is a model of $\tau^c(\phi)$; (ii) w is a model of ϕ iff $\tau^p(w)$ is a model of $\tau^p(\phi)$; (iii) w is a model of ϕ iff $\tau^p(w)$ is a model of $\tau(\phi)$; and (iv) ϕ is satisfiable iff $\tau(\phi)$ is satisfiable.*

The above proposition tells us that the satisfiability of formula is *permutation invariant*, in the sense that satisfiability is preserved if we rearrange clauses, variables or both. As a consequence, we obviously have $\Phi^{n,m}_{\texttt{sat}}(\phi) = \Phi^{n,m}_{\texttt{sat}}(\tau(\phi))$, $\forall \tau \in \mathcal{S}^{m\times n}$.

Another property that we are going show is that if we make a permutation $\sigma(\phi)$ on a formula ϕ, the output of the GNN does not change. Formally, we say that a mapping $F \in \mathcal{F}^{n,m}_{\texttt{GNN}}$ is *permutation invariant* if it satisfies $F(\phi) = F(\sigma(\phi)), \forall \sigma \in \mathcal{S}^{m\times n}$ while we say that $F \in \mathcal{F}^{n,m}_{\texttt{GNN}}$ is *equivalence invariant* if it satisfies $F(\phi) = F(\psi), \forall \phi, \psi \in \mathcal{L}^k_{\texttt{CNF}}(n,m)$ s.t. $\phi \equiv \psi$. Clearly, the feasibility of a MILP problem M does not change if the variables and/or constraints in M are reordered. In particular, the MILP problem M_ϕ is feasible iff $M_{\sigma(\phi)}$ is feasible. A direct consequence of Proposition 2 above and [6] is the following stating that if we rearrange clauses and/or propositional letters the result of the GNN does not change.

Proposition 3. *Consider $\phi \in \mathcal{L}^k_{\texttt{CNF}}(n,m)$. Let $\sigma \in \mathcal{S}^{m\times n}$ be a permutation of clauses and/or propositional letters in ϕ. Then all $F \in \mathcal{F}^{n,m}_{\texttt{GNN}}$ are permutation invariant, i.e. $F(\phi) = F(\sigma(\phi))$.*

Similarly, with a slightly more involved proof, it can be shown that:

Proposition 4. *Consider two equivalent formulae ϕ and ψ in $\mathcal{L}^k_{\texttt{CNF}}(n,m)$. Then $\Phi^{n,m}_{\texttt{sat}}(\phi) = \Phi^{n,m}_{\texttt{sat}}(\psi)$; and all $F \in \mathcal{F}^{n,m}_{\texttt{GNN}}$ are equivalence invariant, i.e. $F(\phi) = F(\psi)$.*

On GNN limitations to represent k-CNF formulae. We now show a limitation of GNNs to represent k-CNF formulae. Specifically, we will show that the

[10] A permutation of a set $S \subseteq \mathbb{N}$ is defined as a bijection from S to itself. That is, $\tau\colon S \to S$ is equivalent to the rearrangement of the elements of S in which each element i is replaced by the corresponding $\tau(i)$.

Algorithm 1 $\mathtt{WL}_{\mathtt{MILP}}$

Input: A bipartite labelled graph $G = \langle V \cup W, E, l \rangle$
Output: A colouring of the nodes in G

$d \leftarrow |V \cup W|$ ▷ d is the degree of G
for all $v \in V$ **do**
  $C_0^V(v) \leftarrow \mathtt{Colour}_0^V(l(v))$ ▷ Initialise constraint node colour
end for
for all $w \in W$ **do**
  $C_0^W(w) = \mathtt{Colour}_0^W(l(w))$ ▷ Initialise variable node colour
end for
for $i = 1, 2, \ldots, d$ **do**
  for all $v \in V$ **do**
    $C_i^V(v) \leftarrow \mathtt{Colour}_i^V(C_{i-1}^V(v), \{\!\{\langle C_{i-1}^W(w), E_{vw}\rangle \mid w \in N(v)\}\!\})$ ▷ Update constraint node colour
  end for
  for all $w \in W$ **do**
    $C_i^W(w) \leftarrow \mathtt{Colour}_i^W(C_{i-1}^W(w), \{\!\{\langle C_{i-1}^V(v), E_{vw}\rangle \mid v \in N(w)\}\!\})$ ▷ Update variable node colour
  end for
end for
return $\langle C_d^V, C_d^W \rangle$

whole GNN family $\mathcal{F}_{\mathtt{GNN}}^{n,m}$ cannot distinguish two formulae that differ w.r.t. satisfiability.

It has recently been shown that the *expressiveness* of GNNs can be related to the *Weisfeiler-Leman* (WL) graph isomorphism test [41] and logics, such as e.g. *finite variable counting logic* $\mathbf{C}_2$ (see, e.g. [4, 12, 22, 31, 29, 30, 33, 40, 43]). Let us recall that the WL graph isomorphism test is a heuristic test for the existence of an *isomorphism* between two graphs G and H. Specifically, in graph theory, an *isomorphism* of graphs G and H is a bijection f between the nodes of G and H, such that any two nodes v_1 and v_2 of G are adjacent in G if and only if $f(v_1)$ and $f(v_2)$ are adjacent in H. We say that f is a *edge preserving* bijection. If an isomorphism exists between two graphs, then the graphs are called *isomorphic*, and is denoted $G \cong H$. For *labelled graphs*, we consider the following definition. A *labelled graph* is of the form $G = \langle V, E, l \rangle$, where l is a *label function* $l\colon V \cup E \to S$, where $S \subseteq \mathbb{N}$. $l(r)$ is a *label* of $r \in V \cup E$. In the case of labelled graphs, an isomorphism is a node bijection f which is both edge-preserving and label-preserving, that is, we additionally require that $l(w) = l(f(w))$ for $w \in V \cup E$.

The algorithm $\mathtt{WL}_{\mathtt{MILP}}$ below is a variant of the WL (viz.. 1-WL) algorithm adapted to our case.[11] In the algorithm, $\mathtt{Colour}_0^V$ (resp. $\mathtt{Colour}_0^W$) injectively maps the label of a constraint (resp. variable) node to a unique natural number; and $\mathtt{Colour}_i^V$ and $\mathtt{Colour}_i^W$ injectively map a pair to a unique natural number, which has not been used in previous iterations. After i iterations the colour of a node v captures some structure of its k-hop neighbourhood, i.e. the subgraph induced by all nodes reachable by walks of length at most k. To test if two graphs G and H are non-isomorphic, we run the above algorithm in parallel on both graphs. If the two graphs have a different number of nodes coloured $c \in \mathbb{N}$ at some iteration, the 1-WL concludes that the graphs are not isomorphic. Now, as any k-CNF formula ϕ is encoded as a labelled bipartite graph G_ϕ, the above colouring of a labelled graph induces a colouring of propositional letters and clauses in ϕ. as illustrated by Algorithm $\mathtt{WL}_{\mathtt{kCNF}}$.

We can now use this colouring to define a sort of isomorphism among formulae. Specifically, given two k-CNF formulae ϕ and ψ, we say that ϕ and ψ are

[11] See [6] for similar algorithms for the MILP case.

Algorithm 2 $\mathtt{WL}_{\mathtt{kCNF}}$

Input: A k-CNF formula ϕ
Output: A colouring of propositional letters and clauses in ϕ
1: $G_\phi \leftarrow \langle V_\phi \cup W_\phi, E_\phi, l_\phi \rangle$ ▷ Initialise G_ϕ as the labelled bipartite graph representation of ϕ
2: **return** $\mathtt{WL}_{\mathtt{MILP}}(G_\phi)$

graph-isomorphic iff their labelled bipartite graph representations G_ϕ and G_ψ are isomorphic.

Unfortunately, there exist some non-isomorphic graph pairs that WL test fail to distinguish. That is, it is well-known that the WL test is known to be correct, but not complete, in the sense that if the WL algorithm says that two graphs are not isomorphic, then they are indeed so. However, the other direction does not hold: that is, there are non-isomorphic graphs where the difference is not detected by the WL algorithm. Therefore, it is not surprising that this applies to the case of k-CNF formulae as well.

Specifically, throughout this paper, we use the notation $\phi \sim \psi$ to denote that ϕ and ψ *cannot be distinguished by the 1-WL test*, that is, $\mathtt{WL}_{\mathtt{kCNF}}(\phi) = \mathtt{WL}_{\mathtt{kCNF}}(\psi)$, for all colouring functions $\mathtt{Colour}_0^V$, $\mathtt{Colour}_0^W$, $\mathtt{Colour}_i^V$ and $\mathtt{Colour}_j^W$ $(i, j > 0)$.

The following proposition indicates that $\mathcal{F}_{\mathtt{GNN}}^{n,m}$ actually has the same expressive power as the 1-WL test, which is a direct consequence of [6, Theorem 3.1] proven for arbitrary MILP problems.

Proposition 5 (Theorem 3.1 in [6])**.** *For all $\phi, \psi \in \mathcal{L}_{\mathtt{CNF}}^k(n, m)$, it holds that $\phi \sim \psi$ iff $F(\phi) = F(\psi)$, for all $F \in \mathcal{F}_{\mathtt{GNN}}^{n,m}$.*

Unfortunately, the following proposition reveals that the 1-WL test is less expressive than $\Phi_{\mathtt{sat}}^{n,m}$.

Proposition 6. *There are two k-CNF formulae ϕ and ψ, the former satisfiable while the latter is not, but $\phi \sim \psi$, i.e. they cannot be distinguished by the 1-WL test.*

Proof. In the following, let us define the *exclusive or* operator as follows: $\varphi \veebar \varphi' \stackrel{\text{def}}{=} (\varphi \vee \varphi') \wedge (\neg\varphi \vee \neg\varphi')$. Now, let ϕ and ψ defined as follows

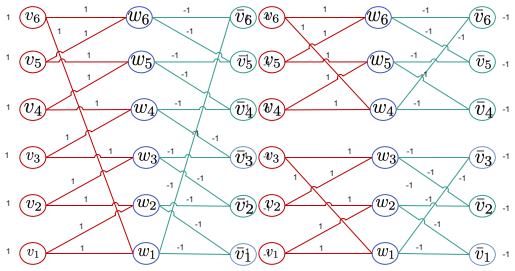
$$\begin{aligned} \phi &= (p_1 \veebar p_2) \wedge (p_2 \veebar p_3) \wedge (p_3 \veebar p_4) \wedge (p_4 \veebar p_5) \\ &\quad \wedge (p_5 \veebar p_6) \wedge (p_6 \veebar p_1) \end{aligned} \tag{13}$$

$$\begin{aligned} \psi &= (p_1 \veebar p_2) \wedge (p_2 \veebar p_3) \wedge (p_3 \veebar p_1) \wedge (p_4 \veebar p_5) \\ &\quad \wedge (p_5 \veebar p_6) \wedge (p_6 \veebar p_4) \end{aligned} \tag{14}$$

One may easily verify that ϕ is satisfiable ($w = p_1\bar{p}_2p_3\bar{p}_4p_5\bar{p}_6$ is model), while ψ is not. Therefore, $\Phi_{\mathtt{sat}}(\phi) \neq \Phi_{\mathtt{sat}}(\psi)$. Now, we show however that $\phi \sim \psi$, i.e. $\mathtt{WL}_{\mathtt{kCNF}}(\phi) = \mathtt{WL}_{\mathtt{kCNF}}(\psi)$. That is, ϕ and ψ cannot be distinguished by the 1-WL test.

The MILP representations M_ϕ and M_ψ are illustrated below (a_h, with $h = 1, 2$, are constants with $a_1 = 1, a_2 = -1$):

$$M_\phi \stackrel{\text{def}}{=} \begin{cases} a_h x_i + a_h x_{i+1} \geq a_h & 1 \leq i < 6, x_j \in \{0, 1\}, \\ a_h x_6 + a_h x_1 \geq a_h & 1 \leq j \leq 6 \end{cases} \tag{15}$$



(A) Bipartite graph G_ϕ. (B) Bipartite graph G_ψ.

FIGURE 2. Bipartite graph representations of ϕ and ψ of Proposition 6.

$$M_\psi \stackrel{\text{def}}{=} \begin{cases} a_h x_i + a_h x_{i+1} \geq a_h & \\ x_3 + x_1 \geq a_h & 1 \leq i < 3, 4 \leq k < 6 \\ a_h x_k + a_h x_{k+1} \geq a_h & x_j \in \{0,1\}, 1 \leq j \leq 6 \\ x_6 + x_4 \geq a_h & \end{cases} \tag{16}$$

It can easily verified that indeed, M_ϕ is feasible ($\bar{\mathbf{x}} = \langle 1,0,1,0,1,0\rangle$ is a solution), while M_ψ is not.

In Fig. 2 we illustrate the labelled bipartite graph representations G_ϕ (Fig. 2a) and G_ψ (Fig. 2b) of ϕ (viz. M_ϕ) and ψ (viz. M_ψ), respectively (w_j are node variables, v_i and $\bar{v}_i$ are node constraints).

Now, we show that the two graphs in Fig. 2 can not be distinguished by WL test, which can be proved by induction. In fact, it is not difficult to see that we may end up with a colouring as illustrated in Fig. 2: each constraint node v_i ($1 \leq i \leq 6$), labelled with 1, is connected to 2 variable nodes w_i ($1 \leq i \leq 6$) via edges labelled 1; and each constraint node $\bar{v}_i$ ($1 \leq i \leq 6$), labelled with -1, is connected to 2 variable nodes w_i ($1 \leq i \leq 6$) via edges labelled -1.
Consequently, one cannot distinguish the two graphs by checking the outputs of the 1-WL test. □

The above proposition tells us that there are two formulae ϕ and ψ such that $\Phi^{n,m}_{\mathtt{sat}}(\phi) \neq \Phi^{n,m}_{\mathtt{sat}}(\psi)$ (for some n, m), but $\phi \sim \psi$ and, thus, by Proposition 5, $F(\phi) = F(\psi)$, for all $F \in \mathcal{F}^{n,m}_{\mathtt{GNN}}$.

GNNs with RNI for k-CNFs. We next show how one may easily get rid of the above mentioned shortcomings. To do so, one may rely on so-called GNNs with *Random Node Initialisation* (RNI), which allows to increasing the expressiveness of GNNs [1, 12, 35]. Not surprisingly, this technique has also been used in fact in [6] for MILP problems, which we illustrate shortly here. Specifically, given graph $G = \langle V, E, \mathbf{X}\rangle$, the idea of RNI consists of extending each node's feature node $\mathbf{x}_v$ with an additional *random value*, e.g. drawn *uniformly* from the interval $[0, 1]$. The intuition is that by appending an additional random feature, each node will have distinct features and the resulting bipartite graph can be distinguished by the 1-WL algorithm. In the following, we call such GNNs as *GNNs with RNI.* and define $\mathcal{F}^{n,m}_{\mathtt{RNIGNN}}$ as the set of all mappings $\mathcal{F}^{n,m}_{\mathtt{RNIGNN}} = \{F \colon \mathcal{L}^k_{\mathtt{CNF}}(n,m) \to \mathbb{R} \mid F(\phi) = \hat{y}_{G_\phi}, \forall \phi \in \mathcal{L}^k_{\mathtt{CNF}}(n,m)\}$ that rely on GNNs with RNI. As a consequence, the output

$\hat{y}_G$ (see Eq. 12) of GNN with RNI is no longer deterministic, but becomes a random variable. Nevertheless, as explained in [6], GNNs with RNI still remain permutation invariant in the sense that the outcome of the computation of GNN with RNI does not depend on the specific representation of the input graph. Consequently, Proposition 3 (permutation invariance) and Proposition 4 (equivalence invariance) continue to hold for $\mathcal{F}^{n,m}_{\texttt{RNIGNN}}$ as well.

The following proposition states that, by adding random features to the weighted bipartite graph representation of a formula ϕ provides sufficient power to represent ϕ's satisfiability status. That is, let $\mathbb{I}^{n,m}_{\texttt{SAT}} : \mathcal{F}^{n,m}_{\texttt{RNIGNN}} \times \mathcal{L}^{k}_{\texttt{CNF}}(n,m) \to \{0,1\}$ be the binary classification function defined as

$$\mathbb{I}^{n,m}_{\texttt{SAT}}(F,\phi) = \begin{cases} 1 & \text{if } F(\phi) > \frac{1}{2} \\ 0 & \text{otherwise .} \end{cases} \tag{17}$$

Specifically, given a GNN with RNI $F \in \mathcal{F}^{n,m}_{\texttt{RNIGNN}}$ and a k-CNF formula $\phi \in \mathcal{L}^{k}_{\texttt{CNF}}(n,m)$, F predicts ϕ to be satisfiable if the output $\hat{y}_{G_\phi}$ of F applied to ϕ is above $\frac{1}{2}$. If $\mathbb{I}^{n,m}_{\texttt{SAT}}(F,\phi) \neq \Phi^{n,m}_{\texttt{sat}}(\phi))$ then we say that F *misclassified* ϕ. Then using [6, Theorem 5.1], the following *Universal SAT Approximation Property* can be shown.

Proposition 7. *Consider a finite dataset $\mathcal{D}$ of formulae in $\mathcal{L}^{k}_{\texttt{CNF}}(n,m)$. For all $\epsilon > 0$, there exists $F \in \mathcal{F}^{n,m}_{\texttt{RNIGNN}}$ such that for all $\phi \in \mathcal{D}$, $Pr(\mathbb{I}^{n,m}_{\texttt{SAT}}(F,\phi) \neq \Phi^{n,m}_{\texttt{sat}}(\phi)) < \epsilon$.*

In other words, the proposition asserts that for any finite dataset $\mathcal{D}$ of formulas in $\mathcal{L}^{k}_{\texttt{CNF}}(n,m)$ and any $\epsilon > 0$, there exists a GNN with RNI such that the misclassification probability for any formula $\phi \in \mathcal{D}$ is less than ϵ, which is consistent with the results shown in [1, 6, 12, 35].

GNNs for Unfoldable k-CNFs.
In this section we show that we may get an analogous result as Proposition 7 for GNNs without RNI, provided that we restrict the dataset to some specific formulae, called *unfoldable* formulae. Specifically, consider a formula $\phi \in \mathcal{L}^{k}_{\texttt{CNF}}(n,m)$ and let us considering the colouring of propositional letters and clauses of ϕ computed by $\texttt{WL}_{\texttt{kCNF}}(\phi)$. We say that ϕ is *foldable* if there are two clauses or two propositional letters in ϕ having the same colour. Otherwise, we call ϕ *unfoldable*. For instance, both formulae ϕ and ψ in Eqs. 13 and 14, respectively, are foldable, while the formula ϕ in Example 1 is not. Note that our definition follows that of [6, Definition 4.1], adapted to formulae.

Now, by using [6, Theorem 4.2.], we may prove the analogue of Proposition 7 for AC-GNNs, provided that we restrict the dataset to unfoldable formulae. That is,

Proposition 8. *Consider a finite dataset $\mathcal{D}$ of unfoldable formulae in $\mathcal{L}^{k}_{\texttt{CNF}}(n,m)$. For all $\epsilon > 0$, there exists $F \in \mathcal{F}^{n,m}_{\texttt{GNN}}$ such that for all $\phi \in \mathcal{D}$ $Pr(\mathbb{I}^{n,m}_{\texttt{SAT}}(F,\phi) \neq \Phi^{n,m}_{\texttt{sat}}(\phi)) < \epsilon$.*

It essentially states that for unfoldable formulae we do not need the extra expressive power of RNI. This concludes this section.

4. Evaluation

The k-SAT problem exhibits the well-known *easy-hard-easy* phase transition [8, 28], based on the *clauses-to-variables ratio* $r = \frac{m}{n}$, where m and n are the number of

clauses and variables, respectively. The hardest instances occur near $m \approx 4.5 \cdot n = h$. For $r \leq h$, formulae are under-constrained, with many models and short solving times. For $r \geq h$, they are over-constrained, typically unsatisfiable, and also quickly resolved. The hardest cases lie near $r = h$, where formulae have few or a unique solution, making them computationally demanding. As such, current GNN SAT solvers often focuses on hard problems, as we do here as well.

We employ here two types of SAT dataset generators. The first one is G4SATBench [20], which, generates, among others, 3-SAT datasets for GNN-based SAT solvers focussing on hard problems. G4SATBench produces sat/unsat balanced random 3-SAT formulae at the phase-transition region (hard region), $h' = 4.258n + 58.26n^{2/3}$ [8]. The other one is our own one. As SAT is NP-complete, building a labelled-dataset can be very time-consuming. A peculiarity of our dataset generator is that, given k, maximal number of variables n, maximal clause length m and dataset size s, we generate a labelled dataset of s random k-CNF formulae not exceeding the given parameters. Then, during the training phase, we may specify, according to some parameters, which formulae to select (always balanced sat/unsat), among which we have e.g. the training, validation and test sizes, the fraction f_p of hard formulae in the area $[h' - f_l \cdot h', h' + f_r \cdot h']$, where $f_l, f_r \in [0, 1]$ are two parameters allowing to define the extend of the hard area. We run our experiments on eight 3-SAT datasets, 4 generated by G4SATBench and 4 generated by MILP-SAT-GNN, of sizes 2K and 20K, each split into *easy* (number of variables $n \in [10, 40]$) and *medium* ($n \in [40, 200]$), in MILP-SAT-GNN we have $f_l = f_r = 0.01$ and $f_p = 1.0$ (all formulae are in the hard area). The validation and test size is 10% of the dataset size. The results are reported in Table 1. On all eight datasets we run our MILP-SAT-GNN solver[12] and the NeuroSAT solver [38][13] as comparison, where the latter has been executed as indicated and implemented in the G4SATBench source code.[14] For MILP-SAT-GNN we made some runs with varying hyperparameters. Given the limited resources we had access to, the number of epochs has been fixed to 150. Concerning RNI, we considered the following percentage of nodes that have a random feature: 0.0% (no RNI), 50% (half of the nodes have a random feature) and 100% (all nodes have a random feature). Concerning the embedding size d and the number of layers l, we considered $d \in \{32, 64, 128\}$ and $l \in \{2, 16, 32\}$, respectively. The batch size is fixed to 64 and the learning rate is 10^{-4}. We considered *Mean Squared-Error* (MSE) and *Binary Cross Entropy* (BCE) loss functions (see, e.g. [15, 42]), but our results showed lower effectiveness with MSE. The only a notable exception for MSE is however the case of (d-l)=128-32 with no RNI of the G4Satbench, case 20K-easy dataset, for which we found an accuracy of 0.84.

As one may see, our preliminary effectiveness results of MILP-SAT-GNN, despite its current simple architecture, are promising and somewhat comparable to the one of NeuroSat[15] with a more notable difference for the cases 20K-easy. Concerning MILP-SAT-GNN, the case (d-l) = 32-16 with 50% RNI has more winning

[12]The source code of MILP-SAT-GNN and the k-SAT generator can be found here: https://www.umbertostraccia.it/cs/ftp/milp-sat-gnn.arxive.zip

[13]In [20] reported among the best one for 3-SAT.

[14]Among others, with 32 hidden layers, 128 embedding size, learning rate 10^{-4}

[15]Not surprisingly, e.g. for the case 128-32 on the G4Satbench 2000 easy dataset, the average learning time per epoch of NeuroSat (70sec) is around 4.5 times longer than the one for MILP-SAT-GNN(15sec).

TABLE 1. Experimental results (accuracy, where loss function is BCE).

			Datasets - Loss BCE							
			G4SATBench				MILP-SAT-GNN			
			2K		20K		2K		20K	
Solver	d-l	RF	easy	medium	easy	medium	easy	medium	easy	medium
Neuro-SAT	-	-	.80	.75	.89	.79	.80	.77	.89	.79
MILP-SAT-GNN	32-2	0	.67	.73	.68	.71	.71	.73	.68	.72
	32-16		.70	.68	.78	.75	.73	**.75**	.78	**.75**
	32-32		.74	.74	**.79**	.75	.70	.73	.78	.74
	64-2		.70	**.75**	.67	.71	.70	.70	.68	.72
	64-16		.75	.66	.77	.75	.70	.73	.77	.74
	64-32		**.79**	.64	.76	.75	.69	.71	.77	**.75**
	128-2		.67	.71	.66	.71	.70	.70	.69	.71
	128-16		.72	.69	.75	.75	.74	.70	.75	.72
	128-32		.73	.67	.75 (**.84**)	.75	.71	.70	.73	.74
	32-2	.5	.71	.73	.67	.71	.69	.70	.67	.71
	32-16		.73	.69	**.79**	.74	.69	**.75**	**.79**	**.75**
	32-32		.76	.66	**.79**	.75	.72	.73	.77	.74
	64-2		.61	.73	.68	.71	.73	.67	.69	.71
	64-16		.73	.70	.77	.75	.72	.70	.77	**.75**
	64-32		.71	.66	.77	**.76**	.70	.72	.76	.74
	128-2		.70	.69	.67	.71	.68	.71	.68	.71
	128-16		.77	.69	.76	.74	.70	.72	.75	.73
	128-32		.74	.75	.76	.75	.72	.74	.75	.73
	32-2	1.0	.69	.72	.68	.71	.72	.70	.68	.73
	32-16		.75	.65	**.79**	.74	.73	.64	.78	.74
	32-32		.77	.73	.78	.74	.74	.72	.78	.73
	64-2		.68	.73	.68	.71	.69	.73	.68	.72
	64-16		.76	.73	.76	.75	.73	.67	.78	.74
	64-32		.71	.71	.77	.74	.68	.73	.76	**.75**
	128-2		.70	.74	.68	.71	.68	.69	.67	.70
	128-16		.76	**.75**	.75	.74	.70	.73	.75	**.75**
	128-32		.70	.74	.76	.75	**.75**	.71	.76	**.75**

conditions. For the considered datasets, having RNI indeed has slight more winning columns than not having RNI, compliant with the results reported e.g. [1, 6].

5. Conclusions

Summary. This paper introduced MILP-SAT-GNN, a novel method that enables Graph Neural Networks (GNNs) to solve Boolean SAT (Satisfiability) problems by leveraging GNNs used for Mixed Integer Linear Programming (MILP) via a transformation of k-CNF formulae into MILPs, which are then encoded as weighted bipartite graphs and processed by a GNN. We contributed with some theoretical properties of the method (invariance, approximate soundness and completeness) and performed also an experimental validation showing promising results and comparable to best performing solver, despite the simple GNN architecture adopted.

Future work. One line, obviously, relates to increase the complexity of GNN architecture with the objective to improve the effectiveness. Also, we are going to address the satisfiability assignment problem within MILP-SAT-GNN (find a model if it exists). Moreover, we would like also to investigate how to train our system on a dataset covering the whole spectrum of formulae with varying sizes k, number of variables and number of clauses, to generalise the system to any input. Another line, by noting that Proposition 1, with an appropriate and more involved transformation into MILP, may be extended to some *many-valued* [14] and paraconsistent [2] logics, relates to the development of GNNs for these logics as well. Additionally, using MILP allows us also to combine logical constraints with numerical constraints, such as "the price is $\geq 15k$". We also would like to investigate whether we can directly encode also some probabilistic logic [23, 34, 26] and used in a neuro-symbolic approach such as [24] (see, e.g. [25] for an overview) in which, rather that having two separated systems (so-called, system1-system2 model), we aim at encoding both as a neural network.

ACKNOWLEDGMENTS

This research was partially supported by the FAIR (Future Artificial Intelligence Research) project funded by the NextGenerationEU program within the PNRR-PE-AI scheme (M4C2, investment 1.3, line on Artificial Intelligence). This work has also been partially supported by the H2020 STARWARS Project (GA No. 101086252), a type of action HORIZON TMA MSCA Staff Exchanges.

References

[1] Ralph Abboud, .Ismail .Ilkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. In *IJCAI*, 2020.

[2] Jair Minoro Abe, Seiki Akama, and Kazumi Nakamatsu. *Introduction to Annotated Logics - Foundations for Paracomplete and Paraconsistent Reasoning*, volume 88 of *Intelligent Systems Reference Library*. Springer, 2015.

[3] Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. Learning to solve circuit-sat: An unsupervised differentiable approach. In *ICLR*, 2019.

[4] Pablo Barceló, Egor V. Kostylev, Mikaël Monet, Jorge Pérez, Juan L. Reutter, and Juan Pablo Silva. The logical expressiveness of graph neural networks. In *ICLR*, 2020.

[5] Chris Cameron, Rex Chen, Jason S. Hartford, and Kevin Leyton-Brown. Predicting propositional satisfiability via end-to-end learning. In *AAAI*, pages 3324–3331, 2020.

[6] Ziang Chen, Jialin Liu, Xinshang Wang, and Wotao Yin. On representing mixed-integer linear programs by graph neural networks. In *ICLR*, 2023.

[7] Stephen Arthur Cook. The complexity of theorem proving procedures. In *STOC*, pages 151–158, 1971.

[8] James M. Crawford and Larry D. Auton. Experimental results on the crossover point in random 3-SAT. *Artificial Intelligence Journal*, 81(31–57), 1996.

[9] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *NeurIPS*, pages 15554–15566, 2019.

[10] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *ICML*, pages 1263–1272, 2017.

[11] Carla P. Gomes, Henry A. Kautz, Ashish Sabharwal, and Bart Selman. Satisfiability solvers. In *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 89–134. Elsevier, 2008.

[12] Martin Grohe. The logic of graph neural networks. *LICS*, pages 1–17, 2021.

[13] Wenxuan Guo, Hui-Ling Zhen, Xijun Li, Wanqian Luo, Mingxuan Yuan, Yaohui Jin, and Junchi Yan. Machine learning methods in solving the boolean satisfiability problem. *Machine Intelligence Research*, 20(5):640–655, 2023.

[14] Reiner Hähnle. Many-valued logics and mixed integer programming. *Annals of Mathematics and Artificial Intelligence*, 3,4(12):231–264, 1994.

[15] William L. Hamilton. *Graph Representation Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2020.

[16] Sean B. Holden. Machine learning for automated theorem proving: Learning to solve SAT and QSAT. *Foundations and Trends in Machine Learning*, 14(6):807–989, 2021.

[17] Robert G. Jeroslow. *Logic-based Decision Support. Mixed Integer Model Formulation*. Elsevier, Amsterdam, Holland, 1989.

[18] Wei Ju, Zheng Fang, Yiyang Gu, Zequn Liu, Qingqing Long, Ziyue Qiao, Yifang Qin, Jianhao Shen, Fang Sun, Zhiping Xiao, Junwei Yang, Jingyang Yuan, Yusheng Zhao, Yifan Wang, Xiao Luo, and Ming Zhang. A comprehensive survey on deep graph representation learning. *Neural Networks*, 173:106207, 2024.

[19] Min Li, Zhengyuan Shi, Qiuxia Lai, Sadaf Khan, Shaowei Cai, and Qiang Xu. On eda-driven learning for SAT solving. In *DAC*, pages 1–6, 2023.

[20] Zhaoyu Li, Jinpei Guo, and Xujie Si. G4SATBench: Benchmarking and advancing SAT solving with graph neural networks. *Transactions on Machine Learning Research*, 2024.

[21] Zhaoyu Li and Xujie Si. Nsnet: A general neural probabilistic framework for satisfiability problems. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *NeurIPS*, 2022.

[22] Andreas Loukas. What graph neural networks cannot learn: depth vs width. In *ICLR*, 2020.
[23] Thomas Lukasiewicz. Probabilistic logic programming. In *ECAI*, pages 388–392, 1998.
[24] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. In *NeurIPS*, pages 3753–3763, 2018.
[25] Giuseppe Marra, Sebastijan Dumancic, Robin Manhaeve, and Luc De Raedt. From statistical relational to neurosymbolic artificial intelligence: A survey. *Artificial Intelligence*, 328:104062, 2024.
[26] Pedro Zuidberg Dos Martires, Luc De Raedt, and Angelika Kimmig. Declarative probabilistic logic programming in discrete-continuous domains. *ArtifICIAL Intelligence*, 337:104227, 2024.
[27] Christian Merkwirth and Thomas Lengauer. Automatic generation of complementary descriptors with molecular graph networks. *Journal of Chemical Information and Modeling*, 45(5):1159–1168, 2005.
[28] David Mitchell, Bart Selman, and Hector Levesque. Hard and easy distribution of SAT problems. In *AAAI*, pages 459–465, 1992.
[29] Christopher Morris, Matthias Fey, and Nils Kriege. The power of the Weisfeiler-Leman algorithm for machine learning with graphs. In Zhi-Hua Zhou, editor, *IJCAI*, pages 4543–4550, 8 2021. Survey Track.
[30] Christopher Morris, Yaron Lipman, Haggai Maron, Bastian Rieck, Nils M. Kriege, Martin Grohe, Matthias Fey, and Karsten M. Borgwardt. Weisfeiler and Leman go machine learning: The story so far. *Journal of Machine Learning Research*, 24:333:1–333:59, 2023.
[31] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *AAAI*, 2019.
[32] Emils Ozolins, Karlis Freivalds, Andis Draguns, Eliza Gaile, Ronalds Zakovskis, and Sergejs Kozlovics. Goal-aware neural SAT solver. In *IJCNN*, pages 1–8, 2022.
[33] Maximilian Pflueger, David Tena Cucala, and Egor V. Kostylev. Recurrent graph neural networks and their connections to bisimulation and logic. In *AAAI*, pages 14608–14616, 2024.
[34] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. ProbLog: A probabilistic prolog and its application in link discovery. In *IJCAI*, pages 2462–2467, 2007.
[35] Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Random features strengthen graph neural networks. In *SDM*, pages 333–341. SIAM, 2021.
[36] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Trans. Neural Networks*, 20(1):61–80, 2009.
[37] Daniel Selsam and Nikolaj S. Bjørner. Guiding high-performance SAT solvers with unsat-core predictions. In *SAT*, volume 11628 of *Lecture Notes in Computer Science*, pages 336–353, 2019.
[38] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. In *ICLR*, 2019.
[39] Zhengyuan Shi, Min Li, Yi Liu, Sadaf Khan, Junhua Huang, Hui-Ling Zhen, Mingxuan Yuan, and Qiang Xu. Satformer: Transformer-based UNSAT core learning. In *ICCAD*, pages 1–4. IEEE, 2023.
[40] Behrooz Tahmasebi, Derek Lim, and Stefanie Jegelka. The power of recursion in graph neural networks for counting substructures. In *ICAIS*, volume 206 of *Proceedings of Machine Learning Research*, pages 11023–11042, 2023.
[41] B. Weisfeiler and A. Leman. The reduction of a graph to canonical form and the algebra which appears therein. *Nauchno-Technicheskaya Informatsia*, 9(2):12–16, 1968. English translation by G. Ryabov is available at https://www.iti.zcu.cz/wl2018/pdf/wl_paper_translation.pdf.
[42] Lingfei Wu, Peng Cui, Jian Pei, and Liang Zhao, editors. *Graph Neural Networks: Foundations, Frontiers, and Applications*. Springer Nature Singapore, 2022.
[43] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.
[44] Zhiyuan Yan, Min Li, Zhengyuan Shi, Wenjie Zhang, Yingcong Chen, and Hongce Zhang. Asymsat: Accelerating SAT solving with asymmetric graph-based model prediction. In *DATE*, pages 1–2, 2024.
[45] Emre Yolcu and Barnabás Póczos. Learning local search heuristics for boolean satisfiability. In *NeurIPS*, pages 7990–8001, 2019.

[46] Jiayi Zhang, Chang Liu, Xijun Li, Hui-Ling Zhen, Mingxuan Yuan, Yawen Li, and Junchi Yan. A survey for solving mixed integer programming via machine learning. *Neurocomputing*, 519:205–217, 2023.

[47] Wenjie Zhang, Zeyu Sun, Qihao Zhu, Ge Li, Shaowei Cai, Yingfei Xiong, and Lu Zhang. Nlocalsat: Boosting local search with solution prediction. In *IJCAI*, pages 1177–1183, 2020.