Continuous-Discrete Message Passing for Graph Logic Reasoning

Cristóbal Corvalán [1] **Francesco Alesiani** [1] **Markus Zopf** [1]

Abstract

The message-passing principle is used in the most popular neural networks for graph-structured data. However, current message-passing approaches use black-box neural models that transform features over continuous domain, thus limiting the reasoning capability of GNNs. Traditional neural networks fail to model reasoning over discrete variables. In this work, we explore a novel type of message passing based on a differentiable satisfiability solver. Our model learns logical rules that encode which and how messages are passed from one node to another node. The rules are learned in a relaxed continuous space, which renders the training process end-to-end differentiable and thus enables standard gradient-based training. Our experiments show that MAXSAT-GNN learns arithmetic operations and that is on par with state-of-the-art GNNs, when tested on graph structured data.

1. Introduction

Graph-structured data can be found in many domains such as biology, chemistry, and computer science Bronstein et al. (2021); Scarselli et al. (2008). Consequently, machine learning for graph-structured data is gaining more interest from the machine learning community.

A key component of neural networks for graph-structured data, so-called graph neural networks, is the message passing principle Gilmer et al. (2017). The key idea of message passing is to exchange messages between nodes in a graph such that representations for nodes or the graph can be learned. The obtained representations are used to address tasks such as node classification Kipf and Welling (2016), graph classification Errica et al. (2019), and missing node feature prediction Rossi et al. (2021).

Even though message passing is used in many graph neural networks, it is far from perfect. On the contrary, several issues with message passing have been reported in prior works. While current graph neural networks exhibit over-smoothing Nt and Maehara (2019); Chen et al. (2020), over-squashing Topping et al. (2021), under-reaching Alon and Yahav (2020) or limited expressive power Morris et al. (2019); Maron et al. (2019); GNNs also provide limited graph reasoning capabilities, since features are manipulated over continuous domain. In addition to these short hands, as we show in our experiments (subsection 4.2), traditional neural network fails to reason over discrete variables (or combinatorial problems Pogančić et al. (2019); Cappart et al. (2021)), as for example in learning and generalizing elementary arithmetic operations.

In this work, we explore an alternative to standard message passing. We propose to learn logic rules (which could model for example binary arithmetics) end-to-end with a differentiable satisfiability solver to encode how messages are distributed within the graph. By modeling the node features as logical variables, we describe the relationship of those features over the neighbor nodes using one or more logical sentences. A feature is propagated over neighbor nodes only if correct according to the graph logic rules.

For example in Figure 1, the variable describing if the molecule is Alanine or Glycine is set based on the number of hydrogen atoms around the carbon atom and the presence of three or two carbon atoms. We thus assume that a collection of logical rules can be collected at the level of the single atom and then verified by pooling the logical variables at the level of the whole graph.

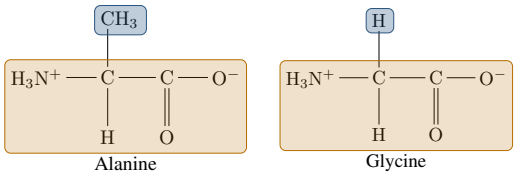

Figure 1: Alanine and Glycine Molecules; the difference is in the presence of a specific sub-structure or the abundance of specific atoms.

Our approach MAXSAT-GNN is a continuous-discrete approach and thus enjoys several benefits such as data effi-

*Equal contribution [1]NEC Laboratories Europe, Heidelberg, Germany. Correspondence to: Francesco Alesiani <Francesco.Alesiani@neclab.eu>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

ciency and interpretability. For example in the arithmetics experiments (subsection 4.2) the number of sentences is limited. Moreover, our experiments show (section 4) that our approach exceeds the accuracy of standard message passing approaches in several tasks.

2. Background

2.1. Notation

An *undirected graph* is a pair $G = (V_G, E_G)$, where $V_G = \{v_1, \dots, v_N\}$ is a finite set of *vertices* (also called nodes), and $E_G \subseteq \{\{u, v\} : u, v \in V_G, u \neq v\}$ is a symmetric, irreflexive, binary relation on V_G. The elements in E_G are called *edges*. $\mathcal{N}(v) = \{u : \{v, u\}, u \in V, \{v, u\} \in E_G\}$ denotes the *neighborhood* of v and $|\cdot|$ denotes the size of a set. For a column vector h, h^T is its transpose.

2.2. SAT and MaxSAT Problems

The motivation for this work is satisfiability problems (SAT) which consists of a set of boolean variables that are related by a logical structure, in other words, elements related by logic rules.

In general, the rules that govern the relationship between those elements can be represented in conjunctive normal form (CNF), which consists of a series of clauses joined by AND operators. CNF can represent any propositional logic (sec.7.5 of Russell (2010)) Each of the clauses may contain some of the variables, or their negation, as follows:

$$(s_{11}x_1 \vee \ldots \vee s_{1n}x_n) \wedge \ldots \wedge (s_{m1}x_1 \vee \ldots \vee s_{mn}x_n), \quad (1)$$

where s_{ji} determines whether the variable $x_i \in \{\bot, \top\}$[1] is present and/or negated in clause j , for example if $s_{11} = 1$ then x_1 participates in the first clause, while if $s_{11} = -1$ then x_1 is negated into $\neg x_1$, while if $s_{11} = 0$ then x_1 is not present. The objective of the SAT problem is to find the truth values of the variables so that the CNF statement is fulfilled.

We consider the optimization analog of this problem (MaxSAT), where the goal is to find a configuration of variables so that the amount of fulfilled clauses is maximized. SATNet Wang et al. (2019) is a MaxSAT solver that can be incorporated into more complex network architectures to solve the MaxSAT problem while it learns the logical structure of the MaxSAT in a continuous and differentiable way. SATNet shows great success in binary encoded prediction problems such as the parity problem and sudoku puzzles.

[1] $\bot$ is the logic false value, and $\top$ is the logic true value. In the following, the true value will be mapped to $+1$, while the false value into -1.

2.3. SATNet

The SATNet Wang et al. (2019) is a satisfiability solver that maps the variables and parameters of the MaxSAT problem into a continuous high-dimensional space. This relaxation allows us to write the MaxSAT problem as Semi-Definite Programming (SDP) problem and solve the relaxation using fast block coordinate descent techniques. Using SDP formulation to solve SAT problem has been shown to have approximation guarantees for MAXCUT and MAX-2SAT, while its use with low rank SDP solver for Max2SAT has been proposed in (Wang et al., 2017; Wang and Kolter, 2019). The use of the SDP formulation allows the algorithm to be integrated as a layer in machine learning systems.

Given a MAXSAT problem with n variables m clauses, we denote the variables of the SAT as $x_i \in \{-1, 1\}$ for $i \in \{1, \dots n\}$, where x_i represent the truth value of each of the i-th variable. Let $s_{ji} \in \{-1, 0, 1\}$ denote the parameters of the SAT for $i \in \{1, \dots n\}$ and $j \in \{1, \dots m\}$. The value of s_{ji} represents thus the sign (if present) of variable x_i in clause j. The MaxSAT problem consists of finding the values of x_i so that the sum of fulfilled clauses is maximized as follows:

$$\max_{x \in \{-1,1\}^n} \sum_{j=1}^{m} \bigvee_{i=1}^{n} \begin{cases} 1 & s_{ji}x_i > 0 \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

The MaxSAT problem is relaxed to form an SDP as done by Goemans and Williamson (1995) and Wang and Kolter (2019). First, the SAT variables x_i are given a probabilistic interpretation, allowing them to be in the interval $P(x_i = 1) \in [0, 1]$. Usually, inputs are binary encoded and are discrete, but the MaxSAT solver based on SATNet allows non-discrete inputs. Second, the probabilistic variables are relaxed by a map into the k-dimensional sphere: $P(x_i = 1) \in [0, 1] \rightarrow \bar{x}_i \in S^{k-1} \subset \mathbb{R}^k$, with $\|\bar{x}_i\| = 1$ and $S^{k-1} = \{\bar{x} \in \mathbb{R}^k : \|\bar{x}\| = 1\}$. The probabilistic variables and the relaxed ones are related by an auxiliary variable $\bar{x}_0$ that is introduced as a truth-direction. The probability of x_i being true will be related to the projection of the relaxed variable in the truth direction:

$$P(x_i = 1) = \cos^{-1}(-\bar{x}_i^T \bar{x}_0)/\pi \quad (3)$$

Additionally, the coefficients s_{ji} are also mapped into the real numbers $\bar{s}_{ji} \in \mathbb{R}$, and an additional coefficient $\bar{s}_{j0} = -1$ is introduced. The MaxSAT in equation 2 can be expressed Wang and Kolter (2019) in the following Semi-Definite Programming (SDP) form:

$$\min_{\bar{X}} \langle S^T S, \bar{X}^T \bar{X} \rangle, \quad \text{such that } \|\bar{x}_i\| = 1 \quad \forall i \quad (4)$$

where $\bar{X} \in \mathbb{R}^{k \times (n+1)}$ and $S \in \mathbb{R}^{m \times (n+1)}$ are the matrices formed by the column vectors $\bar{x}_i$ and $s_i = \bar{s}_i / \sqrt{4\|\bar{s}_i\|}$ respectively. Given a set of known parameters S, the MaxSAT

represented as in Equation 4 is solved via a block coordinate descent method that converges to the optimal global point of the SDP Wang et al. (2017). The solutions of the relaxed MaxSAT $\bar{x}_i \in S^{k-1}$ are mapped back to a probabilistic value using Equation 3.

To solve Equation 4, we first map the logic variable x_i to the relaxed variables $\bar{x}_i$. To improve convergence, the vectors $\bar{x}_i$ are generated from the logical values as $\bar{x}_i = -\cos(\pi x_i)\bar{x}_0 + \sin(\pi x_i)P_\top \bar{x}_i^{\text{rand}}$, where $P_i = I_K - \bar{x}_i\bar{x}_i^T$ is the projection matrix on the vector $\bar{x}_i$, while $\bar{x}_i^{\text{rand}}$ is a random unit vector. The solution of Equation 4 is given as the fix point Wang et al. (2019)

$$\bar{x}_i = -\frac{g_i}{\|g_i\|} \tag{5}$$

where $g_i = \bar{X}S^T s_i - \|s_i\|^2 \bar{x}_i = \bar{X}S^T s_i - v_i s_i^T s_i$.

Given an assignment of the learnable parameters S, the SATNet solver solves in a forward pass the MaxSAT problem. Wang et al. (2019) provides an efficient way to back-propagate their gradients with respect to S. In other words, this module can be combined with already known machine learning differentiable methods to learn the rules of a MaxSAT encoded in the parameters of the S matrix. The complexity of solving Equation 12 Wang et al. (2019), for both forward and backward steps, is $O(knmT)$, with T being the maximum number of iterations. Additional information is provided in the Appendix D. In the following, we will use $y = \text{MAXSAT}_M^N(x)$ to denote a MAXSAT problem with N logic variables and M clauses, where the input variable $x \in [0,1]^{d_x}$ and output variables $y \in [0,1]^{d_y}$ have a combined size of $d_x + d_y = N$. Whenever multiple inputs $x_1, x_2, \ldots$ are presented to MAXSAT, these are concatenated in a single input x.

2.4. Message Passing

Message passing Gilmer et al. (2017) consists of three steps. First, for each pair of connected nodes u, v, a message $m(v,u)$ is computed. Second, for each node v, all messages $m(v,u)$ with $u \in \mathcal{N}(v)$ are aggregated. Third, the node representation of v is updated based on the aggregated messages. In this work, we do not distinguish between the node's feature h_v and the edge message $m(v,u)$ during aggregation.

3. MaxSAT-based Message Passing

We propose a message aggregation procedure where neighboring nodes' features, associated with a central node, are logically related to the updated central node's feature through an unknown MaxSAT, see Figure 2, i.e. a set of logic rules. Our motivation lies in the intuition that the information carried across graph edges and the updated nodes

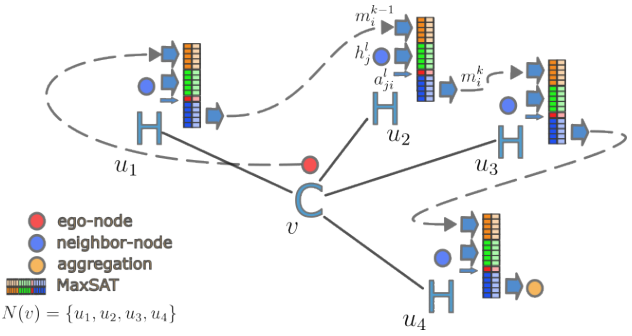

Figure 2: MAXSAT Message passing: Visualization of the Message Passing aggregation using a MaxSAT problem. We use the Methane molecule from Figure 1, where the carbon atom is the current node. The neighbor nodes are visualized according to the adopted notation. The neighbor nodes $N(v)$ are first ordered (subsection 3.2) and then aggregated using Equation 6.

can be represented as a set of truth variables. In noise free application, we could encode all the rules of the graph in a single SAT, however in reality, we have only access to nosy data or simply samples from an underlay discrete distribution. For this reason, maximising the satisfied clauses (i.e. MaxSAT) is a more reasonable approach. The logic rules that fulfills the MaxSAT related to them can in principle be learned and computed from the neighbor nodes and is inherent to the nature of information represented in the graph.

MaxSAT-based message passing (or MAXSAT-GNN) as we will introduce benefits from two features: first, it allows to capture relationships that only involve local interaction (i.e. node's neighbours). Second, our model is capable of carrying interactions between neighboring node features through a memory. Those interactions can be captured at the moment of aggregation.

In the proposed model, we use a differentiable rule learning approach to learn the MaxSAT behind the aggregation. Node features and aggregated messages will therefore acquire a probabilistic nature according to the relaxation process discussed in the previous section.

3.1. Aggregation function over neighbors and message passing using Recursive MaxSAT

We start by describing in more detail our aggregation method on a single GNN layer. Given a central node i, the input of our model is the set of all neighbors ($\mathcal{N}(i)$) node features h_j^l of that node plus the central node feature itself, encoded as binary truth values: $h_i^l, h_j^l \in [0,1]^{d_l}$ for $j \in \mathcal{N}(i)$, where d_l is the dimensionality of the features at the l-layer, where the logic value is represented as a proba-

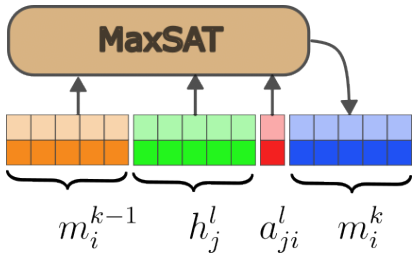

Figure 3: MAXSAT variables: Visualization of the Message Passing aggregation using a MaxSAT problem at the l-layer of the network. m_i^k, m_i^{k-1} are the messages when considering the i-node of the graph, while h_j^l is the feature of the j-neighbour node. The attention bit a_{ji}^l, described in subsection 3.3, helps the MaxSAT to select the relevant features.

bility (Equation 3).

The aggregation function over the neighbors of a node is implemented recursively similar to Recurrent Networks Hochreiter and Schmidhuber (1997), where the aggregation step uses a MaxSAT solver, for the experiments we use SATNet (subsection 2.3). We call it R-MAXSAT-GNN, i.e. Recursive MaxSAT Graph Neural Network. Using node i and the set of its neighbor features $h_i^l, h_j^l : j \in \mathcal{N}(i)$, the R-MAXSAT-GNN applies a logic rule to all of those elements in a recursive manner, in resemblance to an addition operation with multiple inputs. It starts operating on two of them and the output is used as a carry or memory for the next operation with the next element until the whole set takes part in the aggregation. The memory is a key element of our aggregation since it contains the important information from all neighbor nodes to help to compute a logic-related output. Let $\{h_j^l : j \in \mathcal{N}(i)\}$ be the set of features entering the node v_i, where j is the neighbor node index. With reference to Figure 3, we proposed aggregation of the following form:

$$m_i^k = \text{MAXSAT}_M^{3d_l}(m_i^{k-1}, h_j^l) \quad \forall j \in \mathcal{N}(i) \tag{6}$$

$$h_i^{l+1} = m_i^{|\mathcal{N}(i)|}, \quad m_i^0 = h_i^l \tag{7}$$

where m_i^k is the message/memory that aggregates the information from the neighboring nodes for the ego-node, whose feature, h_i^l, can be used as the initial state. The center node feature h_i^l in Equation 7 can be removed, as for example in the Node Missing data experiment (subsection 4.3), and replaced with the first neighbor node's feature.

3.2. Canonical ordering

In Equation 6, the nodes do not have a predefined order, thus, to implement an equivariant or invariant Message Passing method for graph data Bronstein et al. (2021), i.e. to the group of permutations over the nodes, we propose to order the features before they are processed sequentially Niepert et al. (2016). This ordering consists of mapping the binary representation encoded in the features to the real numbers and sorting the neighbors in decreasing order. Whenever two or more nodes have the same feature's values, the relative order is not relevant for the permutation invariant property, since the result of the node's features aggregation of Equation 6 is independent of the permutation of these nodes. While this ordering is fixed, it could be easily extended using a self-attention mechanism, similar to the attention bit (subsection 3.3).

3.3. Logic attention bit

When aggregating the features we consider also the use of an attention bit. This bit is used to help the solver to decide if the message should be processed or not. We call RA-MAXSAT-GNN, the model that uses the attention bit. The attention bit is computed between the center node and each of its neighbors. The attention bit is an additional input to Equation 6 as follows:

$$a_{ji}^l = \sigma(h_j^{lT} W^l h_i^l - b^l) \tag{8}$$

$$m_i^k = \text{MAXSAT}_M^{3d_l+1}(m_i^{k-1}, a_{ji}^l, h_j^l) \quad \forall j \in \mathcal{N}(i) \tag{9}$$

where σ is the non-linear Sigmoid function, $W^l \in \mathbb{R}^{d_l \times d_l}, b^l \in \mathbb{R}^{d_l}$ are trained parameters and m_i^k is the memory of the aggregation related to node i at step k, and d_l is the feature dimension ath l-layer. For the case that the central node feature is missing, we consider a self-attention bit $a_j^l = \sigma(h_j^{lT} W^l h_j^l)$, and the attention would provide self-filtering information for the SAT. As mentioned in the previous section, the attention bit a_{ji}^l could also be used for ordering the nodes. For simplicity, we use the fixed ordering from subsection 3.2.

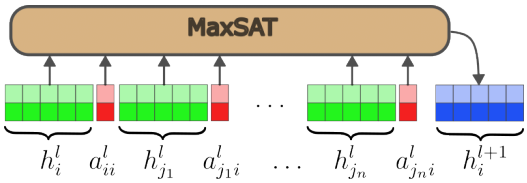

Figure 4: Visualization of the batch version of the MAXSAT, where multiple nodes' features are considered at once.

3.4. Batch aggregation

Recursive aggregation of Equation 6 suffers from various limitations typical of recursive architectures Hochreiter and Schmidhuber (1997), since the output is only observed after the last iteration or the probability uncertainties of the variables grows at each iteration. In both cases, the Satnet is forced to work with non-deterministic features multiple

times which makes the problem highly non-convex and potentially suffers from a vanishing gradient similar to Recurrent Networks. This makes a logic-based decision less accurate.

To evaluate the capability of the recursive aggregation to be trained end-to-end over multiple recursions steps, we introduce an additional model Batch MAXSAT-GNN (B-MAXSAT-GNN) that computes outputs over K neighboring nodes' features at once in a single forward pass, where K is fixed to the maximum node degree of the network. Therefore, node features are ordered and concatenated.

$$h_i^{l+1} = \text{MAXSAT}_M^{(n+2)d_l}(\phi(h_i^l, h_{j_1}^l, \dots, h_{j_n}^l)), \quad (10)$$

where ϕ refers to an ordering function, $j \in \mathcal{N}(I)$, and $n = |\mathcal{N}(i)|$. This model only requires one evaluation and does not require hidden states, thus improving training stability. However, when the degree of the node increases, the size of the MaxSAT problem increases. For larger graphs, we can thus resort to K-neighbors sampling to reduce the size of the MaxSAT problem. When a node has less than K neighbors, the missing node's feature inputs are substituted with a default value $\bar{h}_0$, which can either be set to all zeros vector or learned during training. A visualization of the B-MAXSAT-GNN with also attention bits is proposed in Figure 4.

4. Experiments

In this section, we test experimentally the learning and prediction capabilities of our MAXSAT-GNNs models. They focus on the ability to aggregate features, assign a suitable node label, and finally, find out if these node updates can be used for graph classification.

4.1. Baselines

We evaluate the MAXSAT-GNNs against a variety of baselines that have the features we are interested in. To evaluate the sequential processing, based on recursions with internal states, we consider two recursive networks such as LSTM Hochreiter and Schmidhuber (1997) and GRU Cho et al. (2014). They use a hidden state that is passed to further recursions and regulates the conservation and propagation of information. For message passing on graph structures, we consider the standard graph GCN convolution Kipf and Welling (2016), the GAT convolution which contains an attention mechanism to assign weights to edge messages Veličković et al. (2017) and the Graph Isomorphism Network (GIN) Xu et al. (2018), which improves GNN's expressive power. Tranformer based model have been adapter to work with graph data, for example the Graph Transformer of the Unified Message Passaging Model (UniMP) (Shi et al., 2021).

4.2. Learn arithmetic: addition and multiplication

To support our motivation that logical reasoning can be found in common machine learning problems, we compare the learning capability of arithmetic operations of both MAXSAT and existing approaches. Basic arithmetics over interger numbers are implmented, even in modern computer architectures, using logic rules, we test the ability of the MAXSAT-GNNs in learning those rules. Therefore we take into account two experiments: 1) *Addition* of $2, 3$, and 4 numbers; 2) *Multiplication* with modulo of $2, 3$, and 4 numbers. The synthetic datasets consist of numbers in binary representation with a length of five bits (i.e. integer numbers from 0 to 31). For the addition, we take all possible pairs, triplets, and quadruplets whose sum does not exceed 31. For multiplication, we consider all possible pairs, triplets, and quadruplets. The labels are set to be the result of addition/multiplication with modulo 32 of those numbers. The recurrent networks and the MAXSAT-GNNs are tested by a simple forward pass on the set of numbers. To test those sets on our graph-based benchmarks, we construct a star graph dataset (from the previous sets) with an unlabeled center node whose neighbors correspond to the numbers to be operated. The output after a message aggregation should give us an insight into their ability to learn the arithmetic operation we are studying.

4.2.1. RESULTS OF LEARNING ARITHMETIC

The results of learning arithmetic are summarized in Table 1 for addition and for multiplication. We report the mean accuracy per bit of the binary rounded results given by the models. In general, we notice that our model learns much better arithmetic operations than recurrent networks such as GRU and LSTM. This is evidence of that the MAXSAT-GNNs are more capable of encoding logic functions and carrying them across a memory state. Also, if we take a general view of the results of the graph-based convolutions (GCN and GAT) we discover that MAXSAT-GNNs have more power to aggregate messages on a logic-based setting, which is not based only on a sum aggregation such as the GCN's and GAT's. Taking a look at the specific results, in addition, we find out that MAXSAT-GNNs give satisfactory results in this task. When training on pairs of numbers the R-MAXSAT-GNN achieves a perfect score together with the GIN.

We notice also that adding elements to the recursion makes the performance of the R-MAXSAT-GNN drop by 7.2% and 18.0%. GIN maintains its almost perfect score when training on quadruplets. The B-MAXSAT-GNN is still capable to capture the addition operation maintaining its performance over $98,5\%$ when it is trained on quadruplets.

We conclude from this that the R-MAXSAT-GNN is sensible to lose information as explained in subsection 3.4 when

Method / task	Addition			Multiplication		
Trained/ tested on	$2 \to 2$	$3 \to 3$	$4 \to 4$	$2 \to 2$	$3 \to 3$	$4 \to 4$
R-MAXSAT-GNN	**1.0000(000)**	0.9284(532)	0.8196(493)	**0.9633(055)**	0.9554(42)	0.9030(2)
B-MAXSAT-GNN	-	0.9958(008)	0.9859(105)	-	**0.9586(3)**	**0.9758(018)**
LSTM	0.8254(267)	0.4787(310)	0.7109(47)	0.8000(286)	0.8392(64)	0.8859(58)
GRU	0.8894(280)	0.7779(600)	0.7670(30)	0.8196(54)	0.8509(54)	0.8848(19)
GCN	0.5753(158)	0.6427(56)	0.6735(21)	0.6291(51)	0.7171(40)	0.7907(2)
GAT	0.7946(247)	0.7713(24)	0.7690(78)	0.7917(204)	0.8313(35)	0.8709(31)
GIN	**1.0000(000)**	**0.9999(001)**	**0.9990(011)**	0.8070(106)	0.8321(15)	0.8433(17)

Table 1: In this table we report the performance in terms of accuracy for the Addition of 5-bit numbers and Multiplication with modulo of 5-bit numbers. The best and second-best results (if overlaps statistically) are reported, where the top results are also underlined. The error, expressed as standard deviation, is reported in parenthesis and represents the last relevant digits. For example 1.234 ± 0.050 is represented as $1.234(50)$. The dash represents that B-MAXSAT-GNN is equivalent to R-MAXSAT-GNN.

Training on $2, 3, 4$ number set (addition, out-domain evaluation)						
Tested on	$2 \to 3$	$2 \to 4$	$3 \to 2$	$3 \to 4$	$4 \to 2$	$4 \to 3$
R-MAXSAT-GNN	**1.0000(000)**	**0.9990(015)**	0.8010(1783)	0.7875(1379)	0.6508(1529)	0.6438(1927)
B-MAXSAT-GNN	-	-	0.9938(059)	-	0.9379(578)	0.9709(247)
LSTM	0.6027(1481)	0.5337(1268)	0.5870(526)	0.4787(310)	0.5886(40)	0.6393(73)
GRU	0.6903(233)	0.6034(261)	0.5957(1301)	0.5332(1624)	0.5577(519)	0.6304(167)
GCN	0.6387(61)	0.6727(36)	0.5918(258)	0.6736(16)	0.6013(340)	0.6384(39)
GAT	0.6075(91)	0.5440(150)	0.6421(189)	0.6654(76)	0.7038(147)	0.6688(79)
GIN	0.9199(44)	0.8228(82)	**1.0000(000)**	**0.9679(039)**	**0.9987(022)**	**0.9999(002)**

Table 2: In this table we show the accuracy results for the Addition of 5-bit numbers, where we test the generalization (out-distribution), where the models are trained on $2, 3, 4$ number-sets. The best and second-best results (if overlaps statistically) are reported, where the top results are also underlined. The accuracy is reported as in Table 1. The dash represents when B-MAXSAT-GNN can not be used since the number of operations is larger than K.

the input has more elements. This idea is supported when looking at the uncertainty of the results. B-MAXSAT-GNN has more stable results while the recursive version did only sometimes achieve similar scores (and could not learn in the others). For the multiplication task, R-MAXSAT-GNN achieves the best accuracy score when training with pairs. As before, B-MAXSAT-GNN has the peculiarity that their results stay similar, over 95.8%, with the three datasets.

4.2.2. GENERALIZATION OF ARITHMETIC LEARNED OPERATIONS

We explore if our models can generalize the arithmetic operation on a different aggregation size, by testing them with the other datasets that were not used for training. (For example, the MAXSAT-GNN that was trained with pairs of numbers tested on triplets and quadruplets). We report these results on Table 2 and Table 3. For the addition, we observe in general that the R-MAXSAT-GNN is able to generalize when it was trained on pairs, but in the other experiments, they are not, showing decreases in performance over 12%. On the other hand, B-MAXSAT-GNN proves to be more successful in this task, maintaining their ability to learn pair multiplication at 99.4% and 93.7% in the triplet and quadruplet experiments respectively. Unfortunately, this achievement is obscured by the fact the GIN is able to generalize in all cases with scores over 99%.

We report a similar generalization behavior on the multiplication task with MAXSAT-GNN. In contrast to its recursive version, the accuracy of B-MAXSAT-GNN does not decrease more than 3% for the pairs and triplet experiments and it decreases slightly more, by 5.4%, when it is trained on four numbers and tested on two.

4.3. Node Missing Data

As a second step, knowing that the R-MAXSAT-GNN is capable to learn an arithmetic operation on binary numbers, we move on to real datasets whose features are represented in binary or one-hot encoding. We follow the assumption, that there is some logical operation, similar to an arithmetic operation, that can be performed on messages toward a specific node. This operation would help to discern newer or missing node representations on a graph, for instance, to find node labels when data is not available, from the neighborhood information. *Missing node data prediction* consists in predicting node features based on the information that can be gathered from their neighborhood. It is a useful task when the dataset is incomplete, but there is still information

Training on $2, 3, 4$ number sets (multiplication, out-distribution)						
Tested on	$2 \to 3$	$2 \to 4$	$3 \to 2$	$3 \to 4$	$4 \to 2$	$4 \to 3$
R-MAXSAT-GNN	**0.9445(048)**	**0.9409(066)**	0.7221(1060)	**0.9013(543)**	0.7884(39)	0.8544(1)
B-MAXSAT-GNN	-	-	**0.9306(016)**	-	**0.9218(036)**	**0,9541(014)**
LSTM	0.7654(334)	0.7462(367)	0.6514(134)	0.8001(171)	0.6556(313)	0.7676(443)
GRU	0.7539(288)	0.7707(489)	0.7672(178)	0.8699(167)	0.7063(568)	0.8037(230)
GCN	0.7160(16)	0.7909(4)	0.6116(127)	0.7903(7)	0.6128(73)	0.7121(28)
GAT	0.7426(174)	0.7226(440)	0.7122(78)	0.8499(33)	0.6963(183)	0.7891(45)
GIN	0.5290(59)	0.4970(68)	0.5945(256)	0.6138(45)	0.6278(43)	0.7165(25)

Table 3: In this table we show the accuracy results for the Multiplication with modulo of 5 bit numbers, where we test the generalization (out-distribution), and where the models are trained on $2, 3, 4$ number-sets. The best and second best results (if overlaps statistically) are reported, where the top results are also underlined. The accuracy is reported as in Table 1.

Node Missing Data			
	MUTAG	Mutagenicity	ENZYMES
R-MAXSAT-GNN	**0.9372(031)**	**0.8968(009)**	0.7123(178)
*B-MAXSAT-GNN	0.9201(22)	0.8221(18)	**0.7266(045)**
RA-MAXSAT-GNN	**0.9365(002)**	**0.8962(010)**	**0.7269(046)**
GCN	0.9165(17)	0.7637(92)	0.7164(36)
GAT	0.9086(40)	0.8182(1)	0.7202(43)
GIN	0.9134(109)	0.8263(8)	0.7152(41)

Table 4: The accuracy performance in recovering the missing node features on the molecular datasets. The best and second best results (if overlaps statistically) are reported, where the top results are also underlined. The accuracy is reported as in Table 1.

enough to capture the missing data.

We set up our experiment on three datasets from the benchmarks for graph learning TUDataset Morris et al. (2020a) MUTAG, Mutagenicity, and ENZYMES. For training, we set 20% of all the nodes to be test nodes: their features are set to zero, meaning that they are unknown. The rest of the nodes are the training nodes. During each training iteration (mini-batch), 10% of the training nodes are set to zero, and their features are inferred at training time. We use a similar architecture as in the previous experiment, composed of one layer of message aggregation with our three models and the baselines to gather neighborhood information; and one linear layer for non-probabilistic outputs. The labels of the nodes are one-hot encoded features. Therefore we optimize the cross entropy loss for multiclass classification and evaluate performance using classification accuracy after applying a soft-max layer to the output.

4.3.1. RESULTS OF NODE MISSING FEATURE TASK

As shown in Table 4, the ability of the MAXSAT-GNN to find the correct label based on closest neighbor message passing is similar to or slightly better that the other models.

On the MUTAG dataset, the Satnet achieves an accuracy of 93.7% which is somewhat better than the results of the baselines which reach 91.3%. This difference is more remarkable in the case of the Mutagenicity dataset where the difference is over 7% with respect to the best of the GNN networks. The results achieved on the ENZYMES dataset do not exhibit a particular improvement over the baselines.

4.4. Graph Classification

We further investigate the performance of the proposed method for the task of graph classification. We consider three datasets from the same graph learning benchmarks Morris et al. (2020a): MUTAG, Mutagenicity, and PROTEINS. We additionally tested on the ZINC dataset (Sterling and Irwin, 2015; Gómez-Bombarelli et al., 2018). The first two contain graphs with one-hot encoded features. The PROTEINS dataset consists of an integer number plus a one-hot encoded three-class features. That integer number was "clamped" between the values 0 and 31, the interval where most of the values lie, and subsequently was converted into a binary 5-bit vector and was eventually concatenated to the rest of the features.

All those datasets have a global graph label with two different classes. For training and metric evaluation, we split them into a training set (80%) and a test set (20%) respectively.

The architecture for our MaxSAT-based message passing consisted of 2 layers of message aggregation with RA-MAXSAT-GNN, and B-MAXSAT-GNN. A global pooling using the $\max$ function, which should resemble an OR gate. One linear dense layer followed by a Sigmoid function, for probabilistic outputs. The baselines (GCN, GAT, and GIN) use the same architecture. We train our models using Adam Kingma and Ba (2014) optimizer and the binary cross entropy loss Zhang and Sabuncu (2018). For the Zinc dataset we additional compared with the Graph Transformer. We also evaluate our results using the accuracy metric.

4.4.1. RESULTS OF GRAPH CLASSIFICATION TASK

In complex tasks such as graph classification where multiple aggregations are involved, the MaxSAT-based models are capable of performing similarly to the baselines. The results are shown in Table 5. We report the performance of B-MAXSAT-GNN, although they are not an adequate model for performing aggregation, especially for datasets such as PROTEINS, where the maximum graph degree is considerably larger than the other datasets. While for the ZINC dataset, our MaxSAT model shows lower performance, we see that our models outperform on average the other baselines on the MUTAG dataset reaching 92.1% in accuracy, while in the others the results overlap. This demonstrates that graph classification can be modeled with SAT solvers where an internal logical representation of the nodes is capable of classifying the graphs.

	Graph Classification			
	MUTAG	Mutagenicity	PROTEINS	ZINC
B-MAXSAT-GNN	**0.9211(456)**	0.7949(123)	-	-
RA-MAXSAT-GNN	0.9035(152)	**0.8078(130)**	**0.7227(262)**	0.8690 (0.0206)
GCN	0.7632(912)	0.7919(131)	0.7047(324)	0.9025 (0.0035)
GAT	0.7979(405)	0.7905(87)	0.6811(419)	0.9028 (0.0055)
GIN	0.8502(402)	**0.8150(149)**	**0.6922(475)**	**0.9114** (0.0036)
Graph Transformer -	-	-	-	0.9012 (0.0023)

Table 5: Standard deviation is reported with the last n position. The best and second-best results (if it overlaps statistically) are reported, where the top results are also underlined. The accuracy is reported as in Table 1.

5. Related Work

Deep learning on graphs and in particular graph neural networks (GNNs) has been extensively studied in the last few years Sperduti and Starita (1997); Scarselli et al. (2008); Kipf and Welling (2016); Gilmer et al. (2017). The predominant paradigm is message passing Gilmer et al. (2017), which propagates information using a learnable non-linear function on the graph. Among the most popular architecture is GCN Kipf and Welling (2016), where the graph is represented using the normalized adjacent matrix, GAT Veličković et al. (2017), where the weights of multiple heads on the node are mixed with learnable functions, and Graph Isomorphism Network (GIN) Xu et al. (2018), which achieves the same discriminative power level of Weisfeiler-Lehman (WL) isomorphism test. RNNLogic Qu et al. (2021) uses an EM-based algorithm to learn a set of rules for reasoning on knowledge graphs. Contrary to our approach, the model is not differentiable. In Niepert et al. (2016) the authors consider a fixed canonical ordering, while we use a fixed function, which depends on the current node's feature. To overcome the limited expressive power of GNN, recently alternative approaches have been proposed as in Maron et al. (2019; 2018); Morris et al. (2020b); Morris and Mutzel (2019), where WL-k ($k \geq 1$) networks are described, whose complexity, however, increases exponentially with the expressive level k.

Learning Aggregation Function and Learning on Sets An alternative line of work addresses the improvement in the aggregation function in current graph neural netwok and message passing architectures. (Corso et al., 2020),(Pellegrini et al., 2021). Another alternative direction is discard the graph strauctures and directly learn over sets (Lee et al., 2019). These works do not attempt to model the learning process using logic clauses.

Reasoning in Knoledge Graphs Reasoning over categorical data is a critical task in Knowledge Graphs (Zhang et al., 2020),(Qiu et al., 2023). In this application the information in organized in triplet and infgerence of new relationships or links an important task.

Discrete latent variables An alternative way to model reasoning is to use discrete latent variables. To integrate discrete variables into traditional differentiable architectures, various gradient estimations have been proposed Jang et al. (2016); Maddison et al. (2016); Paulus et al. (2021). Unfortunately, these models only mimic the discrete nature of the variables but do not capture the underlying reasoning mechanism.

Neural Combinatorial Optimization (NCO) The combinatorial problem can be solved using heuristics, NCO methods use deep neural networks to learn adaptable heuristics either using supervised learning or reinforcement learning Joshi et al. (2019); Kool et al. (2019). These approaches could be alternatives to the one proposed in this work.

6. Conclusions

In this paper, we propose to model the properties of graph structure data using logic rules which can be learned through end-to-end training. We exploit the structure of message passing and proposed an invariant-equivariant architecture based on an ordering function and a flexible attention mechanism. We prove with multiple experiments that MAXSAT-GNN learns rules for arithmetic operations, while on molecular datasets is capable of estimating missing node features and classifying graphs. Limitations of this work are: we expect input features to be discrete and data generated at least partially according to some logic rules. Further, the training is longer because of its recursive nature.

REFERENCES

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205*, 2020.

Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges, May 2021. URL http://arxiv.org/abs/2104.13478.

Quentin Cappart, Didier Chételat, Elias Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. *arXiv preprint arXiv:2102.09544*, 2021.

Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3438–3445, 2020.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets, 2020.

Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. *arXiv preprint arXiv:1912.09893*, 2019.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural Message Passing for Quantum Chemistry, June 2017. URL http://arxiv.org/abs/1704.01212.

Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, nov 1995. ISSN 0004-5411. doi: 10.1145/227683.227684. URL https://doi.org/10.1145/227683.227684.

Rafael Gómez-Bombarelli, Jennifer N. Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276, Feb 2018. ISSN 2374-7943. doi: 10.1021/acscentsci.7b00572.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv:1611.01144*, 2016.

Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv:1906.01227*, 2019.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2016. URL https://arxiv.org/abs/1609.02907.

Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! *ICLR*, 2019.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks, 2019.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv:1611.00712*, 2016.

Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. In *International Conference on Learning Representations*, 2018.

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. *Advances in neural information processing systems*, 32, 2019.

Christopher Morris and Petra Mutzel. Towards a practical k-dimensional weisfeiler-leman algorithm. *arXiv preprint arXiv:1904.01543*, 2019.

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 *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4602–4609, 2019.

Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tu-dataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*, 2020a.

Christopher Morris, Gaurav Rattan, and Petra Mutzel. Weisfeiler and leman go sparse: Towards scalable higher-order graph embeddings. *Advances in Neural Information Processing Systems*, 33:21824–21840, 2020b.

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pages 2014–2023. PMLR, 2016.

Hoang Nt and Takanori Maehara. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*, 2019.

Gábor Pataki. On the rank of extreme matrices in semidefinite programs and the multiplicity of optimal eigenvalues. *Mathematics of operations research*, 23(2):339–358, 1998.

Max B Paulus, Chris J Maddison, and Andreas Krause. Rao-blackwellizing the straight-through gumbel-softmax gradient estimator. page 11, 2021.

Giovanni Pellegrini, Alessandro Tibo, Paolo Frasconi, Andrea Passerini, and Manfred Jaeger. Learning aggregation functions, 2021.

Marin Vlastelica Pogančić, Anselm Paulus, Vit Musil, Georg Martius, and Michal Rolinek. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*, 2019.

Haiquan Qiu, Yongqi Zhang, Yong Li, and Quanming Yao. Logical expressiveness of graph neural network for knowledge graph reasoning, 2023.

Meng Qu, Junkun Chen, Louis-Pascal Xhonneux, Yoshua Bengio, and Jian Tang. RNNLogic: Learning Logic Rules for Reasoning on Knowledge Graphs. In *International Conference on Learning Representations*, 2021.

Emanuele Rossi, Henry Kenlay, Maria I Gorinova, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael Bronstein. On the unreasonable effectiveness of feature propagation in learning on graphs with missing node features. *arXiv preprint arXiv:2111.12128*, 2021.

Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification, 2021.

Alessandro Sperduti and Antonina Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.

Teague Sterling and John J. Irwin. Zinc 15 – ligand discovery for everyone. *Journal of Chemical Information and Modeling*, 55(11):2324–2337, Nov 2015. ISSN 1549-9596. doi: 10.1021/acs.jcim.5b00559.

Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. *arXiv preprint arXiv:2111.14522*, 2021.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2017. URL https://arxiv.org/abs/1710.10903.

Po-Wei Wang and J Zico Kolter. Low-rank semidefinite programming for the max2sat problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1641–1649, 2019.

Po-Wei Wang, Wei-Cheng Chang, and J. Zico Kolter. The mixing method: low-rank coordinate descent for semidefinite programming with diagonal constraints, 2017. URL https://arxiv.org/abs/1706.00476.

Po-Wei Wang, Priya Donti, Bryan Wilder, and Zico Kolter. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *International Conference on Machine Learning*, pages 6545–6554. PMLR, 2019.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks?, 2018. URL https://arxiv.org/abs/1810.00826.

Yuyu Zhang, Xinshi Chen, Yuan Yang, Arun Ramamurthy, Bo Li, Yuan Qi, and Le Song. Efficient probabilistic logic reasoning with graph neural networks. *arXiv preprint arXiv:2001.11850*, 2020.

Zhilu Zhang and Mert Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. *Advances in neural information processing systems*, 31, 2018.

Supplementary Materials

A. Limitations of the proposed approach

We recognize some limitations of the proposed approach. First, the dataset's input features are considered discrete and the dataset is generated at least partially according to some logic rules. If the input data is described with continuous variables and quantization of the input values does not introduce high distortion, then the model can be used. In some situations, we can employ an initial nonlinear layer to encode the features either into discrete features (for example using Jang et al. (2016); Maddison et al. (2016); Paulus et al. (2021)) or into continuous value in $[0, 1]$. Otherwise, other approaches are more appropriate. Training of the model is longer because of the recursive nature of the model and proportional to the number of neighbors of the nodes, but the number of variables is similar to alternative methods. With the batch architecture, the computational time and convergence are comparable with the classical forward neural network.

B. Logic expressive power

We model the relationship of nodes' (or edges') features in the neighborhood of a node of a graph. When we use multiple layers, we can extend the scope of the learned rules to a larger number of features. While we model extended logic rules over the features of a graph, we do know if we cover all possible logic rules. This is left for future work.

C. Experimental details

C.1. Addition

For the addition experiments, we set the number of bits to 5, thus the total number of variables is $n = 15$, where two numbers are used as input and one variable is the output. We set the number of auxiliary variables Wang et al. (2019) to aux $= 12$, while the number of clauses $m = 40$. The number of applications of the MAXSAT depends on the experiment $N = 1, 2, 3$. The same network is applied recursively. With the B-MAXSAT-GNN, the missing input variables are set to zero.

C.2. Multiplication

For the multiplication experiments, we set the number of bits to 5, thus the total number of variables is $n = 15$, where two numbers are used as input and one variable is the output. We set the number of auxiliary variables Wang et al. (2019) to aux $= 16$, while the number of clauses $m = 88$. The number of applications of the MAXSAT depends on the experiment $N = 1, 2, 3$. The same network is applied recursively. With the B-MAXSAT-GNN, the missing input variables are set to zero, while aux $= 100, m = 100$, and $n = 5 + 5N$.

C.3. Graph Classification

For the Graph Classification experiments the total number of variables is n, the number of auxiliary variables Wang et al. (2019) aux, and the number of clauses m, the number of applications of the MAXSAT depends on the dataset, for Mutagenicity $N = 5, \text{aux} = 20, m = 20, n = 42$, for PROTEINS $N = 26, \text{aux} = 12, m = [12, 20], n = 24$ and for MUTAG $N = 28, \text{aux} = 12, m = [24, 24], n = 27$. The same network is applied recursively as an aggregation function, while we use 2 layers in the experiments. With the B-MAXSAT-GNN, the missing input variables are set to zero. GCN has a similar architecture with two layers and 64 channels, while GAT has 16 channels, and GIN has 7 channels. An additional network generates the graph classification from the node features. For training, we use ADAM Kingma and Ba (2014) gradient update and $lr = 1e^{-3}$, while the training loss function is the binary cross entropy loss Zhang and Sabuncu (2018).

C.4. Node missing features

As for the Graph Classification experiments, also for the Node missing features experiments the total number of variables is n, the number of auxiliary variables Wang et al. (2019) aux, the number of clauses m, the number of application of the MAXSAT depends on the dataset, for Mutagenicity $N = 5, \text{aux} = 20, m = 20, n = 42$, for PROTEINS $N = 26, \text{aux} = 12, m = [12, 20], n = 24$ and for MUTAG $N = 28, \text{aux} = 12, m = [24, 24], n = 27$. The same network is applied recursively as an aggregation function, while we use 2 layers in the experiments. With the B-MAXSAT-GNN, the missing input variables are set to zero. GCN has a similar architecture with two layers and 64 channels, while GAT has 16 channels, and GIN has 7 channels. For training, we use ADAM Kingma and Ba (2014) gradient update and $lr = 1e^{-3}$, while the training loss function is the binary cross entropy loss. The difference with respect to the graph classification is that we do not have a graph pooling function, but we predict the node features for the missing node features directly.

D. Differentiable Satisfiability Network

D.1. MAX-SAT Problem

In Maximal Satisfiability Problems (MAX-SAT), we are interested to find the assignment of n binary variables $x_i \in \{-1, 1\}, i = 1, \ldots, n$ concerning m given clauses, or

$$\max_{x \in \{-1,1\}^n} \sum_{j \in [m]} \vee_{i \in [n]} 1_{s_{ji} x_i > 0} \tag{11}$$

where $s_{ji} \in \{-1, 0, +1\}$ are the clauses of the MAX-SAT problem. If $s_{ji} = 0$ the variable i is ignored in the j clause, while $x_i = +1$ is associated with a true value and $x_i = -1$ to a false value, thus $s_{ji} = -1$ negates the variable x_i. MAX-SAT is one of the extensions of the Satisfiability (SAT) problem, where all the clauses need to be true. Relaxing the SAT is useful when we want to find the closest solution that satisfies most of the clauses.

D.2. SAT-Net: differentiable MAX-SAT relaxation via Semi-definitive programming (SDP)

The problem in Equation 11 can be relaxed into a Semi-Definitive Programming (SDP) problem **?**Wang and Kolter (2019); Wang et al. (2017)

$$\min_{V \in \mathbb{R}^{k \times (n+1)}} \langle S^T S, V^T V \rangle \quad \text{s.t.} \quad \|v_i\| = 1, \forall i \in \{\top, 1, \dots, n\}, \tag{12}$$

where for each input variable x_i is associated with unitary vector $v_i \in \mathbb{R}^K$ of dimension k, with some $k > \sqrt{2n}$ Pataki (1998), with k is the size of the embedded space, while n is the number of variables. The variable $v_\top$ is used as a reference and is associated with true logic value. The normalized matrix $S = [s_\top, s_1, \dots, s_n] / \operatorname{diag}(1/\sqrt{4|s_j|}) \in \mathbb{R}^{m \times (n+1)}$ encodes the clauses, while the unitary matrix $V \in \mathbb{R}^{K \times (k+1)}$ encodes the variables.

Reading the logic variables Once we solve the relaxed problem, we need to compute the logic variables from the vectors that minimize Equation 12.

$$P(x_i = 1) = \frac{1}{\pi} \arccos(-v_i^T v_\top)$$

The probability measures the angle between the vector associated with the true value and the vector associated with the i variable, indeed $v_i^T v_\top = \cos(\pi x_i)$. If we want to recover the discrete value, we compute the sign of the probability, i.e. $x_i = \operatorname{sign}(P(x_i = 1))$.

Transforming the logic variables to the relaxed vectors We generate the vectors from the logical values as $v_i = -\cos(\pi x_i) v_\top + \sin(\pi x_i) P_\top v_i^{\text{rand}}$, where $P_i = I_K - v_i v_i^T$ is the projection matrix on the vector v_i, while v_i^{rand} is a random unit vector.

Solving the SDP relaxation The solution of Equation 12 is given as the fix point Wang et al. (2019)

$$v_i = -\frac{g_i}{\|g_i\|} \tag{13}$$

where $g_i = V S^T s_i - \|s_i\|^2 v_i = V S^T s_i - v_i s_i^T s_i$.

Algorithm 1 Forward pass algorithm: coodrinate descent

Input: V_I, where I is the set of input variables
Output: V_O, where O is the set of output variables

```
G = V S^T
while not converged do
    for i ∈ O do
        g_i = V S^T s_i − ||s_i||^2 v_i
        v_i = −g_i/||g_i||   As described in section subsection D.2
        G = G + (v_i − v_i^prev) s_i^T
    end for
end while
```

Auxiliary variables As noted in Wang et al. (2019), additional variables (*aux*) are necessary to help the SDP relaxation to converge to the minimal point. These variables do not have a specific meaning, but we notice that they are akin to reformation using additional variables of the original problem, this reformulation, while not changing the original truth table, helps the underlying minimization procedure to converge.

Computational complexity of solving SDP relaxation The complexity of the algorithm depends on the solution of Equation 12. As shown in (Wang et al., 2019) the solution of the SDP relaxation can be computed using coordinate descent and the integration as differentiable is implemented using two separated, but similar algorithms Alg.1 and Alg.2 for the forward and backward passes.

Since the algorithms require only rank-one updates, the overall complexity of the two algorithms is $O(Tkmn)$, with k the expanded dimension, n the number of variables and m the number of clauses. At the same time, T represents the number of iterations of the algorithm. During the experiments, T is set to a small number, e.g. $T = 40$.

Computational complexity of solving SDP relaxation on graphs When solving MAXSAT-GNN, we need to solve a MAXSAT problem for each node and for all the neighbours. The computational complexity of in this scenario is $O(TkmnNML)$, where N is the number of nodes, M is the number of edges, and L number of layers. The batch version of MAXSAT-GNN has the same time complexity, but it is more stable during training since the MAXSAT modules are not connected in series but in parallel.

E. Motivational Example: 1-WL Isomorphism Test

In Figure 5, we show two graphs that can not be distinguished according to the 1-WL isomorphism test and, consequently, by a standard GNN. Indeed the neighborhood of

Algorithm 2 Backward pass algorithm: coodrinate descent

Input: $\nabla_O \mathcal{L}$, where O is the set of output variables
Output: $\nabla_I \mathcal{L}$, where I is the set of input variables
$U_O = 0, F = U_O S_O^T = 0$
while not converged **do**
  for $i \in O$ **do**
    $\nabla g_i = F s_i - \|s_i\|^2 u_i - \partial \mathcal{L}/\partial v_i$
    $u_i = -P_i \nabla g_i / \|g_i\|$
    $F = F + (u_i - u_i^{\text{prev}}) s_i^T$
  end for
end while

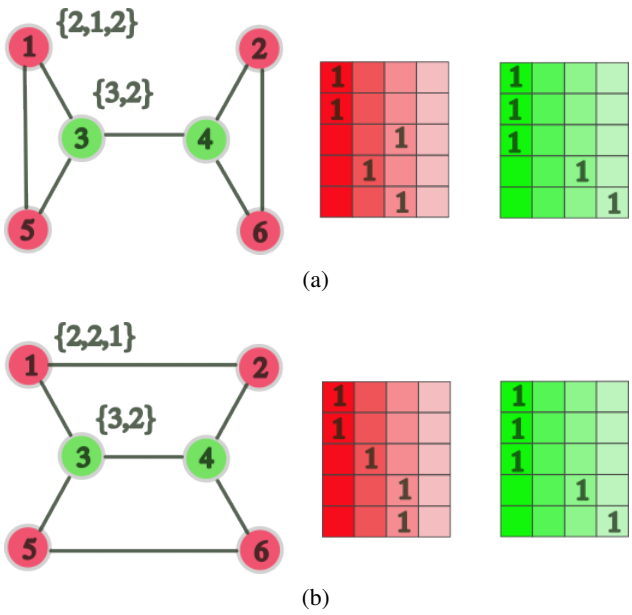

Figure 5: Graph (a) and graph (b) are not distinguishable for a standard GNN or 1-WL test.

the red and green nodes in both graphs is the same, so the aggregation function will return the same result. If we are able to propagate the multi-hop distance in binary format, then we can reason on the relative distance of nodes. In a simplified example, consider the adjacent matrix A of the two networks, we then thus use the one-hop and two-hop adjacent matrices $A_{\text{a}}, A_{\text{a}}^2$ of the first graph (Equation 14) and $A_{\text{b}}, A_{\text{b}}^2$ of the second graph (Equation 15). We can use the rows of the two-hop adjacent matrix to reason on the node contribution. For example, the node 1 (as highlighted in Equation 14) has one entry equal to 2 and three equal to 1, while in the second graph (as highlighted in Equation 15), two entries equal to 2 and one equal to 1. We can thus use this information to classify the node or the graph. A standard GNN would not be able to count the entries.

$$A_{\text{a}} = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}, A_{\text{a}}^2 = \begin{pmatrix} 2 & 0 & 1 & 1 & 1 & 0 \\ 0 & 2 & 1 & 1 & 0 & 1 \\ 1 & 1 & 3 & 0 & 1 & 1 \\ 1 & 1 & 0 & 3 & 1 & 1 \\ 1 & 0 & 1 & 1 & 2 & 0 \\ 0 & 1 & 1 & 1 & 0 & 2 \end{pmatrix} \tag{14}$$

$$A_{\text{b}} = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}, A_{\text{b}}^2 = \begin{pmatrix} 2 & 0 & 0 & 2 & 1 & 0 \\ 0 & 2 & 2 & 0 & 0 & 1 \\ 0 & 2 & 3 & 0 & 0 & 2 \\ 2 & 0 & 0 & 3 & 2 & 0 \\ 1 & 0 & 0 & 2 & 2 & 0 \\ 0 & 1 & 2 & 0 & 0 & 2 \end{pmatrix} \tag{15}$$

F. Zinc Experiments

We propose also experiments with the Zinc dataset, where we model the regression tasks as a classification into 5 classes representing different intervals defined by the intervals of probabilities $[0.7, 0.3, 0.1, .01]$ and train on the first $12'000$ samples. We additional compared with a graph transformer model (Shi et al., 2021). For this dataset, the GIN network provides higher accuracy.

Graph Classification	ZINC
B-MAXSAT-GNN	0.8690(0206)
Graph Transformer	0.9012(0023)
GCN	0.9025(0035)
GAT	0.9028(0055)
GIN	**0.9114(0036)**

Table 6: Standard deviation is reported with the last n position. The best and second-best results (if it overlaps statistically) are reported, where the top results are also underlined. The accuracy is reported as in Table 1.