On the Performance of Deep Generative Models of Realistic SAT Instances

Iván Garzón
LSI, DaSCI, University of Granada, Spain

Pablo Mesejo
DECSAI, DaSCI, University of Granada, Spain

Jesús Giráldez-Cru[1]
DECSAI, DaSCI, University of Granada, Spain

Abstract

Generating realistic random SAT instances – random SAT formulas with computational characteristics similar to the ones of application SAT benchmarks – is a challenging problem in order to understand the success of modern SAT solvers solving this kind of problems. Traditional approaches are based on probabilistic models, where a probability distribution characterizes the occurrences of variables into clauses in order to mimic a certain feature exhibited in most application formulas (e.g., community structure), but they may be unable to reproduce others. Alternatively, deep generative models have been recently proposed to address this problem. The goal of these models is to learn the whole structure of the formula without focusing on any predefined feature, in order to reproduce all its computational characteristics at once. In this work, we propose two new deep generative models of realistic SAT instances, and carry out an exhaustive experimental evaluation of these and other existing models in order to analyze their performances. Our results show that models based on graph convolutional networks, possibly enhanced with edge features, return the best results in terms of structural properties and SAT solver performance.

2012 ACM Subject Classification Computing methodologies → Artificial intelligence

Keywords and phrases Realistic SAT generators, pseudo-industrial random SAT, deep generative models, deep learning

Digital Object Identifier 10.4230/LIPIcs.SAT.2022.3

Funding This work is funded by the University of Granada (Spain) under project PPJIA2021-05.
Jesús Giráldez-Cru: is also supported through the Juan de la Cierva program, fellowship IJC2019-040489-I, funded by MCIN/AEI /10.13039/501100011033.

1 Introduction

The Boolean Satisfiability Problem (SAT) is one of the fundamental problems in Computer Science and Artificial Intelligence. It has been extensively studied from a theoretical point of view due to its complexity; SAT is the first known NP-complete problem. This means that in the worst case, existing algorithms to solve SAT may run during exponentially long executions. However, despite its worst-case complexity, modern SAT solvers are nowadays able to solve large application SAT benchmarks in a reasonable amount of time. This is due to the breakthrough contributions in the so-known Conflict-Driven Clause Learning (CDCL) algorithm [29]. Nevertheless, the reasons explaining the success of CDCL solving large application SAT formulas are not completely understood yet [19].

In the last years, there have been some attempts of explaining such a success. In the literature, we can distinguish between rigorous and correlative approaches. On the one hand, rigorous metrics generally establish theoretical upper-bounds on certain classes of SAT

[1] Corresponding author

formulas. Some examples of rigorous metrics are backdoors [31], and branch-width [1], among others. Unfortunately, these rigorous metrics do not usually provide much information on the actual hardness of existing application SAT formulas. On the other hand, correlative metrics focus on features exhibited in the vast majority of these benchmarks that correlate to CDCL performance. Some examples are the scale-free structure [3] and the community structure [2]. In general, formulas with this kind of structures are easy in practice, although there is no theoretical guarantee about their hardness.

A common approach in order to better understand the efficiency of SAT solving algorithms is using random SAT models. In the classical random SAT model, a formula $F_k(n,m)$ is a set of m clauses over n variables, where clauses are chosen uniformly and independently among all the $2^k\binom{n}{k}$ non-trivial clauses of length k.[2] Although the empirical hardness of this model has been extensively studied [25, 8, 24, 34], it is unable to reproduce the computational properties observed in real-world SAT instances, and hence, it hardly explains their hardness.

Alternatively, there are some models in the literature focused on reproducing certain correlative metrics, which are exhibited by most of real-world SAT instances (e.g., scale-free structure [3] and community structure [2]). In order to generate realistic SAT instances (i.e., random SAT formulas with these features observed in real-world benchmarks), a non-uniform probability distribution can be defined to assign variables to clauses in a certain manner, resulting in some models such as the Scale-Free (SF) model [4], the Community Attachment (CA) model [13], and the Popularity-Similarity (PS) model [14]. For this reason, we refer them as probabilistic models. Unfortunately, these probabilistic models are only able to reproduce a few features at once.

Recently, deep generative models have been proposed as a new paradigm to address the generation of realistic SAT instances. The goal of these models is to learn the whole structure of the formula in order to reproduce all its properties at once. G2SAT [36] is one example of these models. The main idea of G2SAT is representing the SAT formula as a graph in order to perform a number of node-splitting operations to generate the set of training examples which is used to train a model based on a graph neural network (GNN). This (trained) model is used afterwards to generate random formulas of the same size (i.e., same number of nodes and edges) with an expected resembling structure. A previous model that inspires the architecture of G2SAT can be found in [32].

A key aspect of deep generative models is the computation of node embeddings, i.e., the information associated to each node in the graph, which is computed by the GNN. Obviously, this information depends on the graph representation of the SAT formula used by the model. In G2SAT, this is done with a convolution layer based on GraphSAGE [16] over a bipartite graph with literals and clauses. Moreover, the experimental evaluation of G2SAT presented in [36] is only performed on a limited set of 10 heterogeneous SAT instances, training the model with all of them at once. Therefore, it remains unclear whether G2SAT is able to learn *any* kind of structure and how the convolutional layer used to compute node embeddings affects its performance. These open questions on the performance of deep generative models of realistic SAT instances motivate the present work.

In order to address the previous questions, in this work we present a much more extensive experimental evaluation of models based on the G2SAT framework. Besides G2SAT, we also analyze a model that uses Graph Convolutional Networks (GCN) [9, 10, 11] in order to compute node embeddings, as already suggested in [36]. Also, we propose two new models based on edge enhanced graph neural networks (EGNN) [15], and on edge-conditioned

[2] A non-trivial clause of length k contains k distinct, non-complementary literals.

convolution (ECC) [30]. In contrast to G2SAT, these new models are able to handle a graph representation of the SAT formula which captures its whole structure in a more transparent manner, without the need of *ad hoc* artifacts. In particular, these new models do not require to create additional message passing paths between a literal and its negation in the GNN (as G2SAT requires). To do so, they use a graph representation of the SAT formula where edges model the sign of each variable within each clause, and this edge information is naturally processed by the GNN. To the best of our knowledge, they are the first deep generative models of realistic SAT instances that consider edge features to represent this semantic.

In our experimental study, we analyze the performance of these models in terms of structural properties and practical hardness. The structural analysis is based on two crucial features of application of SAT formulas. Namely, they are the community structure and the clustering coefficient. Notice that the G2SAT framework does not alter the distribution of variable occurrences in the generated formulas, hence it does not make sense to study features such as the scale-free structure. On the other hand, the hardness of the instances is evaluated with five well-known CDCL SAT solvers. In both cases, we analyze the structure and the hardness of the synthetic instances generated by these models with respect to a set of heterogeneous (input) real-world SAT instances. In contrast to [36], we train the models separately for each formula, which allows us to evaluate with a finer precision the performance of the generators. Our analysis shows that the models based on GCN and EGNN can improve the performance of G2SAT in many cases. We emphasize that the whole experimentation carried out in this work took more than 240 days, thus this extensive experimental evaluation is the main contribution of the present work.

The rest of this work is organized as follows. In Section 2 we summarize some related works, while Section 3 defines some preliminary concepts. Section 4 describes the framework of G2SAT, and Section 5 describes the new deep generative models of realistic SAT instances proposed in this work. Section 6 is devoted to the extensive experimental evaluation of these deep generative models. Finally, we conclude in Section 7.

2 Related works

There are some works in the literature facing the problem of the generation of realistic SAT instances, i.e., SAT formulas with computational properties similar to the ones observed in real-world benchmarks. In the SF model [4], SAT formulas are generated with scale-free structure, i.e., variable occurrences follow a power-law distribution, thus with high variability. The CA model [13] is able to generate SAT instances with any desired community structure. In a formula with clear community structure, variables are grouped into communities such that they mainly co-occur in clauses with other variables of the same community. The goal of the PS model [14] is to reproduce in a single model these two structures. In particular, it generates formulas with scale-free structure and high clustering coefficient, where the community structure emerges as a results of them. G2SAT [36] is a pioneer approach of deep generative models to the generation of realistic SAT instances, where a GNN is trained to *learn* the structure of a (set of) input instance in order to generate formulas similar to it. A preliminary version of it can be found in [32].

Machine learning has been already used in the context of SAT. One example is SATzilla [35, 17], where it is proposed a per-instance algorithm portfolio that estimates the best solver to solve a given formula from a predefined set. This portfolio approach has also been successfully applied in other works [23, 22]. More recently, deep learning has been introduced in order to solve SAT [28, 27].

3 Preliminaries

The SAT problem consists of deciding whether there exists a satisfying assignment for a given propositional formula. A SAT formula is in Conjunctive Normal Form (CNF) if it is written as a conjunction of clauses, where a clause is a disjunction of literals, and a literal is either a Boolean variable or its negation. Let $var(\varphi)$, $lit(\varphi)$, and $cl(\varphi)$ be respectively the functions that return the set of variables, literals, and clauses of a SAT formula φ. Let v_i, l_j, and ω_k refer to variables, literals, and clauses, respectively.

For any graph $G(V, E)$, where V is the set of nodes and $E \subseteq V \times V$ is the set of edges, let $N(u)$ be the neighborhood of node $u \in V$, i.e., the set $\{v \in V | (u, v) \in E\}$, and let E_u the set of adjacent edges to node $u \in V$, i.e., $\{(u, w) \in E\}$ for any node $w \in V$.

A Literal-Clause Graph LCG_φ of a SAT formula φ is a bipartite graph $G(V_1, V_2, E)$, where $V_1 = lit(\varphi)$ (i.e., literal-nodes), $V_2 = cl(\varphi)$ (i.e., clause-nodes), and $E \subseteq V_1 \times V_2$ is the set of edges indicating the occurrence of a literal in a clause of φ, i.e., $(l_i, \omega_k) \in E \leftrightarrow l_i \in \omega_k$ for any clause $\omega_k \in cl(\varphi)$. A Variable-Incidence Graph VIG_φ of a SAT formula φ is a graph $G(V, ew)$, where $V = var(\varphi)$ (i.e., variable-nodes), and $ew : V \times V \to \mathbb{R}$ is the edge weight function defined as $ew(v_i, v_j) = \sum_{\substack{\omega \in \varphi \\ v_i, v_j \in \omega}} 1/\binom{|\omega|}{2}$. This is, the sum of all the weights generated by a clause is equal to 1, hence it considers the length of the clauses. A Signed Variable-Clause Graph $SVCG_\varphi$ of a SAT formula φ is a bipartite graph $G(V_1, V_2, E)$, where $V_1 = lit(\varphi)$ (i.e., literal-nodes), $V_2 = cl(\varphi)$ (i.e., clause-nodes), and $E \subseteq V_1 \times V_2 \times \{+, -\}$ is the set of edges indicating the occurrence and the sign of a variable in a clause of φ, i.e., $(v_i, \omega_k, +) \in E \leftrightarrow l_i \in \omega_k$ and $(v_i, \omega_k, -) \in E \leftrightarrow \neg l_i \in \omega_k$, for any clause $\omega_k \in cl(\varphi)$. For the sake of clarity, we remove the subindex φ of the previous graph representations whenever it is clear the SAT formulas φ they refer to.

4 The G2SAT framework

In this section we first provide a brief overview of GNN. Then, we summarize the G2SAT framework [36] based on GraphSAGE, and then a modification of it based on GCN. For further details we address the reader to the original reference.

4.1 Graph Neural Networks

Graphs represent a challenging structure to be addressed with conventional machine learning techniques. These types of techniques are specialized in simple data structures, with fixed sizes and structure (i.e., images). However, graphs are a type of data without a fixed form, with a indeterminate number of unordered nodes, each having a different number of neighbors. Therefore, the need of developing a new type of neural networks capable of working on graphs is addressed by GNN.

The key aspect of GNN are node embeddings, which map the features of each node into a low-dimensional space. This space is expected to provide information about node similarity and (global) graph structure. As a consequence, the main operation of GNN consists of learning a non-linear transformation function able of computing node embeddings based on the aggregation of itself and its neighbor embeddings. This operation is known as *message passing* [12], and can be visualized in Figure 1. The learning procedure consists of learning the (trainable) weights of both the aggregation and the transformation functions. This process is repeated in a number of layers (including linear and pooling layers), until the GNN

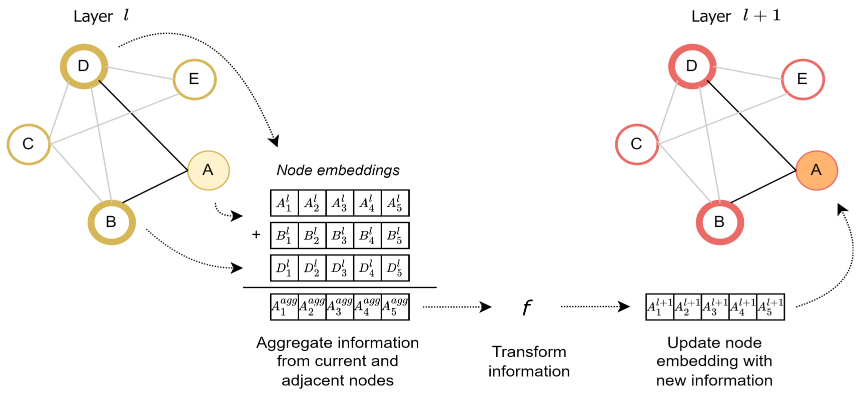
Layer l
Layer $l+1$
Node embeddings
Aggregate information from current and adjacent nodes
Transform information
Update node embedding with new information
f

Figure 1 Computation of node embeddings based on the *message passing* operation on a graph.

is trained to solve different types of problems like node classification, graph classification and link prediction, among others. A more in-depth review of these types of methods and architectures can be found in [37, 33].

4.2 Overview of G2SAT

The general idea of G2SAT is to *learn* a probability distribution $p(G)$ of the input graph(s) G over the set of bipartite graphs $\mathbb{G}$, i.e., learning the complex dependencies between nodes and edges in the input data. In order to learn $p(G)$, G2SAT applies an n-step iterative process: $p(G) = \Pi_{i=1}^{n} p(G_i|G_1,\ldots,G_{i-1})$ where G_i is the intermediate graph at step i. Nevertheless, this can be simplified as $p(G_i|G_1,\ldots,G_{i-1}) = p(G_i|G_{i-1})$ since the order of generating the intermediate graphs does not alter the final resulting graph. To learn this distribution, G2SAT relies on two operations: node splitting and node merging. They are always applied to clause-nodes in the LCG of the formula, so we only define them to bipartite graphs as follows.

▶ **Definition 1** (Node splitting)**.** *Given a bipartite graph $G(V_1, V_2, E)$ and a node $u \in V_2$, the $nodeSplit(G, u)$ operation returns a graph $G(V_1, V_2', E')$ where $V_2' = V_2 \cup \{u'\}$ and $E' = E \cup \{(u', v)\} \setminus \{(u, v)\}$ for some nodes $v \in N(u)$. This is, the node u is split into u and u' and its adjacent edges are distributed among them. In G2SAT this edge distribution is randomly chosen.*

▶ **Definition 2** (Node merging)**.** *Given a bipartite graph $G(V_1, V_2, E)$ and two nodes $u, v \in V_2$, the $nodeMerge(G, u, v)$ operation returns a graph a graph $G(V_1, V_2', E')$ where $V_2' = V_2 \setminus \{v\}$ and $E' = E \cup \{(u, w)\} \setminus E_v$ for all nodes $w \in N(v)$. This is, the node v is collapsed into u, including all its adjacent edges.*

Any bipartite graph can be transformed into a set of trees by successively applying the node splitting operation over the nodes in one of its partitions (e.g., clause-nodes). Also, a bipartite graph can be always generated from a set of trees by applying a sequence of node merging operations. Due to the extremely high computational cost of computing $p(G_{i+1}|G_i)$, G2SAT proposes to divide this process into two steps: $p(G_i, u, v) = p(u, v \,|\, G_{i-1}) \cdot p(G_i \,|\, G_{i-1}, u, v)$. The new step $p(u, v \,|\, G_{i-1})$ consists of selecting two random variables u and v which represent a random pair of node candidates that may be merged in the second step (slightly modified to consider them). In G2SAT, a discrete uniform probability distribution is used to select such a random pair (u, v) as per Equation (1).

$$\begin{aligned} p(G_{i+1}, u, v \mid G_i) &= p\,(u, v \mid G_i) \cdot p\,(nodeMerge(G_i, u, v) \mid G_i, u, v) \\ &= \text{Uniform}\left(\{(u, v) \mid \forall u, v \in V_2^{G_i}\}\right) \cdot \text{Bernoulli}\left(\sigma(\mathbf{h}_u^T \mathbf{h}_v \mid u, v)\right) \end{aligned} \tag{1}$$

where $\sigma(\cdot)$ is the sigmoid function, and $\mathbf{h}_u$ and $\mathbf{h}_v$ are the node embeddings for nodes u and v. Therefore, the key question is how to obtain the node embeddings in order to correctly capture the structure of the input graph(s).

In G2SAT, node embeddings are computed with GraphSAGE [16], a variant of GCNs with strong inductive capabilities across different graphs. In this framework, the node embedding of a node in a certain layer l depends on the embeddings of its neighbors and of itself in the previous layer $l-1$. In particular, the node embedding $\mathbf{h}_u^{(l+1)}$ of a node u in the $(l+1)$-th layer is given by Equation (2).

$$\mathbf{h}_u^{(l+1)} = \text{ReLU}\left(\mathbf{W}^{(l)} \cdot \text{CONCAT}\left(\mathbf{h}_u^{(l)}, \text{AGG}\left(\text{ReLU}(\mathbf{Q}^{(l)}\mathbf{h}_v^{(l)} + \mathbf{q}^{(l)} \mid v \in N(u)\right)\right)\right) \tag{2}$$

where $\mathbf{Q}^{(l)}$, $\mathbf{q}^{(l)}$, and $\mathbf{W}^{(l)}$ are the trainable parameters of the model, and AGG$(\cdot)$ is the mean pooling aggregation function. The input node features are one-hot vectors of length 3, indicating the type of node (clause, positive, or negative literal). In this process, an additional message passing path is created between every literal and its negation, due to their strong relation.

Notice that the previous procedure defined in Equation (1) can be seen as a classification task, where the model has to decide whether a pair of nodes (u, v) must be merged or not. In order to train the model, during the training phase a number of node splitting operations is performed. When a node s is split into a pair of nodes (s, s^+), a positive example is generated. Also, a negative example (s, s^-) can be easily generated by just selecting a randomly chosen node $s^- \in V_2 \backslash \{s, s^+\}$. This dataset of positive and negative examples is then used to compute the node embedding, by minimizing the binary cross entropy loss of Equation (3):

$$\mathcal{L} = -\mathbb{E}_{u,v \sim p_{pos}}\left[log\left(\sigma(\mathbf{h}_u^T \mathbf{h}_v)\right)\right] - \mathbb{E}_{u,v \sim p_{neg}}\left[log\left(1 - \sigma(\mathbf{h}_u^T \mathbf{h}_v)\right)\right] \tag{3}$$

where p_{pos} and p_{neg} are distributions over positive and negative training examples. In order to guarantee a tree-like structure from the original bipartite graph, the number of node splitting operations is $|E| - |V_2|$. Moreover, this tree-like decomposition from the original graph is repeated r times. Therefore, the number of training examples is $2r(|E| - |V_2|)$. Finally, the resulting r trees are saved into a template in order to start the generation procedure from one of them.

In the generation phase, G2SAT starts with a randomly chosen tree from the previous template, and iteratively performs node merging operations between two nodes $u, v \in V_2$, i.e., clause-nodes. For each node merging, a number of distinct pairs (u, v) is randomly generated with the only constraint that clause-node v cannot be connected to any variable (both literal-nodes, with any sign) already connected to clause-node u. From this set of pairs, the selected pair is $(u, v) = \text{argmax}_{u,v}\{\mathbf{h}_u^T \mathbf{h}_v\}$. Notice that this does not exactly follow Equation (1), but uses a greedy method to select the most likely pair (u, v) among the set of possible candidates. This is due to the large amount of time required to sample the true distribution. Experimental results reported in [36] show that this change also produces reasonable results in a much shorter time.

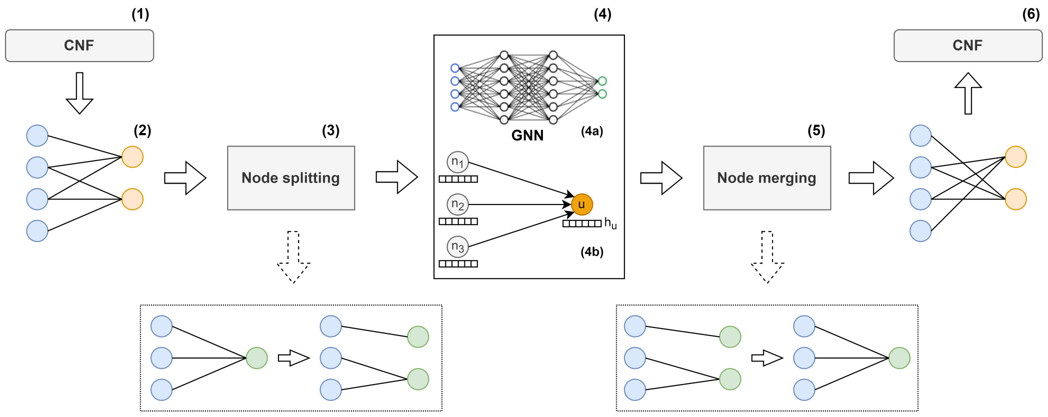

Figure 2 General overview of GSAT-based methods. First, an input CNF (1) is represented as a bipartite graph (2). Then, this graph is converted into a set of trees after a number of node splitting operations (3), from which training examples are generated. In the training phase (4), a GNN (4a) is used to compute node embeddings according to the neighbors of each node (4b). Finally, in the generation phase, node merging operations (5) are performed to generate a bipartite graph which represents the output CNF (6). The methods analyzed in this work differ in the graph representation of the CNF formulas (steps 2 and 5) as well as in the way node embeddings are represented and computed (steps 4a and 4b).

In Figure 2, we depict the general overview of the G2SAT framework, which will be used by all the methods analyzed in this work. They all differ in the graph representation of the CNF formulas (see steps 2 and 5) as well as in the way node embeddings are represented and computed (see steps 4a and 4b).

4.3 Node embeddings based on GCN

The original version of G2SAT uses GraphSAGE to compute node embeddings. However, as suggested in [36], other GCN-based convolutional layers can be used to this end. In this work, we analyze the graph convolution proposed in [18]. Equation (4) describes the computation of the node embeddings $\mathbf{H}$ (in matrix notation):

$$\mathbf{H}^{(l+1)} = \mathrm{ReLU}\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right) \tag{4}$$

where $\tilde{A}$ is the adjacency matrix with added self-connections, i.e., $\tilde{A} = A + I_N$ (where I_N is the identity matrix of size N), $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, and $\mathbf{W}$ is the trainable matrix of weights.

This is the only difference of this model with respect to G2SAT. This model will be named as GCN2S in the rest of this work.

5 Deep generative models of realistic SAT instances based on G2SAT

In this section we define the two new alternatives of deep generative models of realistic SAT instances proposed in this work. They both follow the general ideas from G2SAT, but they differ in the way node embeddings are computed. Recall that node embeddings is the key aspect in the process of G2SAT, where pairs of nodes are selected to be merged based on those embeddings. Our first model is based on EGNN [15], where a number of edge features (e.g., sign of the variables within the clauses) is used to compute node embeddings. In our second model, we use an ECC [30] where edge filter weights are dynamically generated according to the neighborhood of each node.

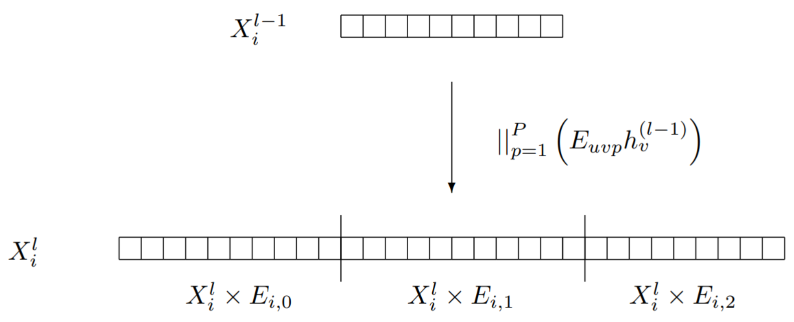

Figure 3 Computation of node embeddings based on EGNN, as a concatenation of adjacent edge embeddings. In this example, $P = 3$, i.e., each edge has three features.

5.1 Enhancing G2SAT edge representation with EGNN

A drawback of G2SAT and, in particular, of the graph model used to represent SAT formulas (i.e., the LCG), is the inability to explicitly represent the relation between a literal and its negation. This limitation forces G2SAT to create additional message passing paths between these two nodes, due to the strong dependency between them. As a consequence, the bipartite structure of the LCG is destroyed in order to introduce this *ad hoc* solution.

Alternatively, we present a model where this dependency is directly captured from the graph representation of the formula. Our model is based on EGNN [15], where edges are allowed to have several features. This is useful to represent graphs with multiple type of edges. In this model, we use the SVCG to represent SAT formulas. Recall that this representation has two types of edges (positive and negative) in order to represent the sign of each variable within each clause. Therefore, we can compute node embedding with EGNN using this information of the edges. Notice that this does not require to create additional message passing paths between (literal) nodes, since this dependency is directly captured by the edge features.

As in G2SAT, each node u is characterized by a node embedding $\mathbf{h}_u$. Additionally, EGNN also includes a tensor representing the edge features. In particular, E_{uvp} represents the feature $p \in [1, P]$ on the edge between nodes u and v. These edge features can be seen as a *filter* to produce the new node embeddings. This way, the node embedding is computed as an aggregation between itself and its adjacent edges. This process is performed in a multi-layer feed-forward architecture, and is summarized in Equation (5) for a layer l.

$$\mathbf{h}_u^{(l)} = \text{ReLU}\left(\mathbf{W}^{(l-1)} \cdot \text{AGG}(||_{p=1}^{P} E_{uvp}\mathbf{h}_v^{(l-1)})\right) \qquad \forall v \in N(u) \cup \{u\} \tag{5}$$

where $||\cdot$ is the concatenation operator, and the aggregation function is the average. Notice that the dimensionality of the concatenation depends on the number of edge features P, but it is reduced by the aggregation function to the number of weights in that layer.

In Figure 3, we represent the computation of node embeddings based on EGNN. In particular, the embedding of a certain node depends on the embeddings of all the adjacent edges to it and their edge features, and all these edges features are aggregated in order to produce the new node embedding in the next layer.

In our model, node embeddings are computed using the previous procedure. This allows us to integrate edge features (i.e., the sign of the variables within the clauses) in a transparent manner. The rest of the model follows the same ideas from G2SAT. First, a sequence of node-splitting operations transforms the bipartite SVCG into a set of trees. This produces a

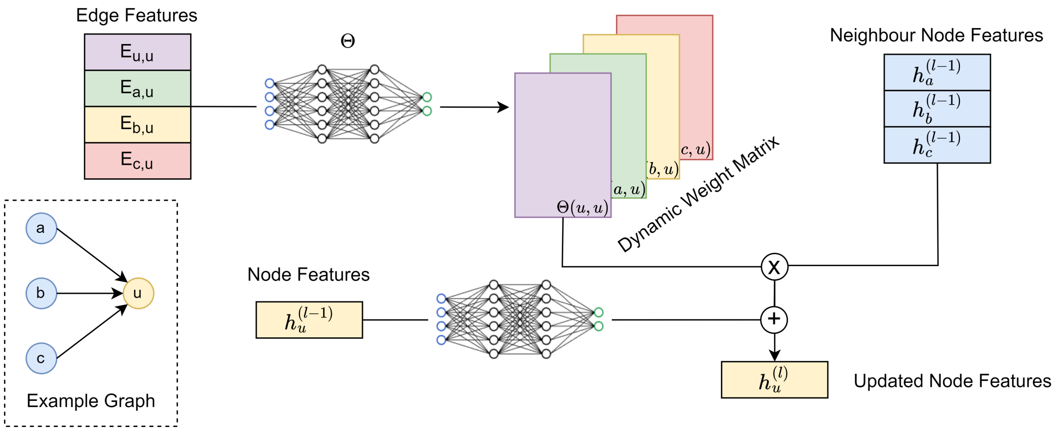

Figure 4 Computation of node embeddings based on ECC, where a dynamic weight matrix (based on edge features) is used to aggregate neighbor embeddings.

number of positive and negative examples which will be used to train the model afterwards. Finally, in the generation phase, a random formula is generated using that trained model after a sequence of node merging operations. Notice that the edge type is not altered during the node splitting and merging operations, so the total number of literals remains unaltered by this model. In the rest of this work we refer this model as EGNN2S.

5.2 Adjusting G2SAT node embeddings with ECC

Another alternative to capture the strong dependencies between literals of a SAT formula in the graph representation is ECC [30]. Again, this model does not require any *ad hoc* artifacts to learn those dependencies. To this end, node embeddings are computed according to their neighbors in the graph, but using a dynamic weight matrix that depends on edge features. To get this dynamic matrix, a neural network is trained in order to transform the edge features (of an arbitrary size) to the specific size required in the convolution layer.

Specifically, the node embedding $\mathbf{h}_u^{(l)}$ of a node u for a layer (l) is computed in Equation (6).

$$\mathbf{h}_u^{(l)} = \mathbf{W}^{(l-1)}\mathbf{h}_u^{(l-1)} + \sum_{v \in N(u)} \Theta_{\mathbf{W}^{(l-1)}}(E_{uv})\mathbf{h}_v^{(l-1)} \tag{6}$$

where $\mathbf{W}^{(l)}$ is the weight matrix in layer l, $\Theta_{\mathbf{W}}$ is the neural network that transforms edge features into a dynamic weight matrix used by the neighbor node embeddings, and E_{uv} is the edge feature between nodes u and v.

In Figure 4, we depict this architecture. It can be seen that the dynamic weight matrix is computed with Θ from the edge features. Based on it, the aggregation of neighbors is computed and added to the current node embedding.

In our implementation, Θ is a multi-layer perceptron (MLP) with 2 dense layers, using a ReLU activation layer between them. This MLP maps edge features of dimension $|E|$ to a dimension of $(in\_channels \cdot out\_channels)$, where both $in\_channels$ and $out\_channels$ depend on the dimensionality of the corresponding convolutional layer.

In this model, we use the SVCG to represent SAT formulas, and compute node embeddings according to Equation (6). The rest of the model follows the same strategy than G2SAT, i.e., a sequence of node splitting operations during the training phase, and a sequence of node merging actions during the generation step. From now on, we refer this model as ECC2S.

6 Experimental evaluation

In this section, we provide an extensive experimental evaluation of the four deep generative models of realistic SAT instances analyzed in this work. Namely, they are G2SAT (Sect. 4.2), GCN2S (Sect. 4.3), EGNN2S (Sect. 5.1), and ECC2S (Sect. 5.2). The four models are implemented in PyTorch 1.9 and their code is publicly available.[3] We first analyze their performance in terms of the structure observed in the generated formulas with respect to the original structure. Second, we analyze the hardness of the generated benchmarks in terms of SAT solver performance.

In our analysis we select a total of 41 real-world SAT instances to train the models (see Appendix A for more details on these formulas). Notice that the results on G2SAT presented in [36] are only computed for 10 smallest formulas of our analysis. Therefore, this experimental evaluation provides new and better insights about the performance of G2SAT.

For every input formula, we separately train each model and generate 3 distinct random instances. The four models are trained with learning rate of 0.001, batch size of 64, and 200 training epochs, using Adam optimizer. All of them are composed of 3 convolutional layers with 32 filters each. Models have been trained on a single GTX Titan Xp GPU, and Intel® Xeon® E5-2630 CPU.

We must emphasize that the whole experimentation, including the training and the generation phases performed by the 4 models on these 41 SAT formulas, took more than 240 days. Therefore, the first conclusion drawn from our experiments is the poor scalability of these G2SAT-based models. Notice that the formula size of the largest instance is only 88,849 literals in 22,792 clauses, a small size compared to the size of actual benchmarks used in the last SAT Competitions. As a consequence, an interesting direction of future work on deep generative models of realistic SAT instances is the adaptation of these models to handle larger real-world benchmarks. This phenomenon is summarized in Observation 3.

▶ **Observation 3.** *G2SAT-based frameworks exhibit a limited scalability due to the large amount of time spent in the pair selection procedure.*

6.1 Performance on structural properties

In the first part of our experimental analysis, we analyze the structure of the generated formulas with respect to the one of the original formulas used to train the models. In contrast to [36], we train the models separately for each SAT formula because of the heterogeneity of structures found in the input data. For instance, if a model is trained with graphs (or SAT formulas) having both low and high values for a specific feature, the synthetic graphs generated with it may have an average value for this feature, thus not resembling to any of the input graphs.

In our analysis, we focus on two structural properties that have been extensively studied in the context of SAT formulas due to their correlation with respect to solving times. Namely, they are the community structure [2, 19] and the clustering coefficient [34, 14]. They both are studied on the VIG of the formulas.[4] Although both structures represent the density and the form of connections within the graph, they differ in its scale. In particular, the clustering coefficient focuses on local densities, whereas the community structure determines the global density of the graph within communities of nodes.

[3] `https://github.com/i4vk/SAT_generators`
[4] To study the clustering coefficient, we use the unweighted version of the VIG.

On the one hand, to study the community structure we analyze the modularity Q [26]. This value represents the density of connections in a partition of the nodes of a graph into communities. Specifically, the modularity of a (weighted) graph G and a partition P of its nodes is defined as:

$$Q(G(V,w),P) = \sum_{P_i \in P} \frac{\sum\limits_{x,y \in P_i} w(x,y)}{\sum\limits_{x,y \in V} w(x,y)} - \left(\frac{\sum\limits_{x \in P_i} deg(x)}{\sum\limits_{x \in V} deg(x)} \right)^2 \tag{7}$$

The optimal modularity Q is a value in $[0,1]$, with higher values indicating a more clear community structure.

On the other hand, the clustering coefficient of a node determines the number of neighbors of that node that are also connected. As in [36], we analyze the average clustering coefficient of the graph. This value is defined as:

$$CC(G(V,E)) = \frac{1}{|V|} \sum_{i \in V} \frac{|\{e_{jk} : v_j, v_k \in N(i), e_{jk} \in E\}|}{|N(i)| \cdot (|N(i)| - 1)} \tag{8}$$

This CC value is defined in the interval $[0,1]$, with higher values representing graphs with a higher clustering coefficient.

Also, we must emphasize that the design of G2SAT does not alter the number of occurrences of each literal. Notice that node splitting and merging operations are only performed on clause-nodes. As a consequence, the degree of literal- or variable-nodes (depending on the model) remains unaltered. Therefore, we omitted in our experimental evaluation the study of distributions of variables occurrences, such as the scale-free structure [3], since all the generated formulas have exactly the same distribution than the one in the input data.

In Table 1 we report the results on this analysis of structural properties, remarking in bold the model obtaining the best performance for each metric. The first observation is that there is a very low, almost negligible, variability in the generated formulas of every model. In a large majority of the cases, the standard deviation in the analyzed metrics is smaller than 0.01. Therefore, this suggests that all the models are robust to learn a particular structure, regardless whether it is close to the structure in the input data or not. This is summarized in Observation 4.

▶ **Observation 4.** *G2SAT-based frameworks exhibit a robust performance in terms of the achieved structural features, due to the small variability of these features in the generated formulas.*

For each model and metric, we consider that its performance is adequate whenever the difference between the generated formulas and the input one is smaller than 0.05, i.e., both structures are resembling. Based on that, the results on the community structure show that the four models exhibit a reasonably good performance, since the modularity in the generated instances is, in general, similar to the one in the input formulas. However, comparing the accuracy of the models, we observe that G2SAT is the one with the worst performance, whereas EGNN2S and GCN2S achieve very good results. In particular, our proposed model EGNN2S is the one with the best accuracy in 12 of the 41 input formulas. On the contrary, our model ECC2S shows a modest performance. Interestingly, in 16 formulas no model is able to correctly reproduce the original structure, including formulas with both low and high modularity (see, e.g., `I1` and `I17`). These observations on the community structure are summarized in Observation 5.

Table 1 Performance of deep generative models of realistic SAT instances on the community structure (modularity) and the (average) clustering coefficient (in the VIG) of a set of real-world SAT instances. For each model and input formula, best results are marked in bold whenever the precision w.r.t. the original instances (Or.) is smaller than 0.05. #i stands for the number of instances for which each generator shows the best performance.

	Community structure (modularity)					Clustering coefficient				
ID	Or.	G2SAT	GCN2S	EGNN2S	ECC2S	Or.	G2SAT	GCN2S	EGNN2S	ECC2S
1	0.84	0.49 ± .00	0.47 ± .00	0.29 ± .00	0.23 ± .00	0.66	0.88 ± .00	0.75 ± .01	0.89 ± .00	0.88 ± .00
2	0.64	0.90 ± .00	0.84 ± .01	**0.66 ± .00**	0.77 ± .00	0.44	0.71 ± .00	**0.47 ± .01**	0.30 ± .01	0.28 ± .00
3	0.66	0.87 ± .00	0.83 ± .01	0.62 ± .00	**0.67 ± .00**	0.46	0.82 ± .01	0.52 ± .02	0.33 ± .00	0.33 ± .00
4	0.63	0.91 ± .00	0.85 ± .00	**0.64 ± .00**	0.79 ± .00	0.45	0.52 ± .00	0.33 ± .00	0.29 ± .00	0.20 ± .00
5	0.73	0.93 ± .00	0.86 ± .00	**0.77 ± .00**	0.82 ± .00	0.44	0.37 ± .00	0.25 ± .00	0.26 ± .00	0.17 ± .00
6	0.68	0.94 ± .00	0.81 ± .01	0.89 ± .00	0.84 ± .00	0.49	0.74 ± .00	0.41 ± .00	**0.45 ± .01**	0.33 ± .01
7	0.60	0.74 ± .01	0.53 ± .01	0.75 ± .01	0.70 ± .01	0.63	**0.65 ± .01**	0.40 ± .01	0.42 ± .01	0.40 ± .01
8	0.79	0.93 ± .00	**0.80 ± .00**	0.93 ± .00	0.85 ± .00	0.58	0.69 ± .00	0.37 ± .00	0.47 ± .01	0.28 ± .00
9	0.72	0.91 ± .00	0.78 ± .00	0.90 ± .00	0.83 ± .00	0.61	0.70 ± .00	0.39 ± .00	0.41 ± .00	0.29 ± .00
10	0.72	0.60 ± .00	0.59 ± .00	0.38 ± .00	0.46 ± .00	0.59	0.54 ± .00	0.42 ± .00	0.33 ± .00	0.31 ± .00
11	0.73	0.91 ± .00	0.87 ± .00	0.84 ± .00	0.80 ± .00	0.54	0.37 ± .00	0.27 ± .00	0.24 ± .00	0.18 ± .00
12	0.84	0.86 ± .00	**0.83 ± .00**	0.69 ± .00	0.78 ± .00	0.51	0.23 ± .00	0.17 ± .00	0.16 ± .00	0.13 ± .00
13	0.69	0.86 ± .01	0.84 ± .00	0.77 ± .00	0.79 ± .00	0.47	0.25 ± .00	0.21 ± .00	0.26 ± .00	0.16 ± .00
14	0.76	0.87 ± .00	0.83 ± .00	0.79 ± .00	**0.77 ± .00**	0.54	0.30 ± .00	0.20 ± .00	0.17 ± .00	0.15 ± .00
15	0.73	0.88 ± .00	0.86 ± .00	**0.74 ± .00**	0.81 ± .00	0.50	0.20 ± .00	0.15 ± .00	0.21 ± .00	0.13 ± .00
16	0.72	0.88 ± .00	0.86 ± .00	**0.74 ± .00**	0.81 ± .00	0.51	0.19 ± .00	0.15 ± .00	0.21 ± .00	0.13 ± .00
17	0.15	0.58 ± .00	0.64 ± .00	0.71 ± .00	0.65 ± .00	0.50	0.72 ± .00	**0.53 ± .00**	0.41 ± .00	0.41 ± .00
18	0.17	0.57 ± .00	0.65 ± .00	0.69 ± .00	0.66 ± .00	0.64	0.56 ± .00	0.49 ± .00	0.34 ± .00	0.33 ± .00
19	0.18	0.53 ± .00	0.65 ± .00	0.62 ± .00	0.55 ± .00	0.62	0.82 ± .00	**0.64 ± .00**	0.54 ± .00	0.56 ± .00
20	0.61	0.85 ± .00	0.70 ± .01	**0.57 ± .00**	0.50 ± .00	0.62	0.67 ± .00	0.41 ± .01	0.40 ± .00	0.37 ± .00
21	0.72	0.88 ± .00	0.77 ± .00	0.60 ± .00	0.50 ± .00	0.54	0.66 ± .00	0.39 ± .00	0.31 ± .00	0.31 ± .00
22	0.66	0.76 ± .00	0.59 ± .01	0.80 ± .00	0.73 ± .00	0.45	0.53 ± .01	0.37 ± .01	0.33 ± .00	0.32 ± .01
23	0.67	0.79 ± .00	**0.63 ± .01**	0.80 ± .00	0.74 ± .00	0.43	0.57 ± .00	0.35 ± .01	0.30 ± .00	0.28 ± .01
24	0.80	0.75 ± .00	0.82 ± .00	**0.79 ± .00**	0.64 ± .00	0.38	**0.42 ± .00**	**0.34 ± .00**	0.20 ± .00	0.14 ± .00
25	0.65	0.83 ± .00	0.70 ± .00	0.81 ± .00	0.74 ± .00	0.43	0.63 ± .00	0.38 ± .00	0.30 ± .00	0.28 ± .00
26	0.77	0.75 ± .00	0.81 ± .00	**0.78 ± .00**	0.63 ± .00	0.39	0.52 ± .00	**0.39 ± .00**	0.21 ± .00	0.16 ± .00
27	0.78	0.75 ± .00	0.82 ± .00	**0.78 ± .00**	0.63 ± .00	0.39	0.48 ± .00	**0.36 ± .00**	0.20 ± .00	0.15 ± .00
28	0.69	0.85 ± .00	0.78 ± .01	0.82 ± .00	0.77 ± .00	0.39	0.66 ± .00	**0.42 ± .00**	0.27 ± .00	0.24 ± .00
29	0.73	**0.72 ± .00**	0.78 ± .00	0.80 ± .00	0.62 ± .00	0.40	0.47 ± .00	**0.43 ± .00**	0.27 ± .00	0.19 ± .00
30	0.79	0.74 ± .00	0.82 ± .00	**0.80 ± .00**	0.63 ± .00	0.39	**0.38 ± .00**	**0.38 ± .00**	0.22 ± .00	0.14 ± .00
31	0.75	**0.73 ± .00**	0.80 ± .00	0.80 ± .00	0.62 ± .00	0.39	0.45 ± .00	**0.42 ± .00**	0.26 ± .00	0.17 ± .00
32	0.77	0.69 ± .00	**0.76 ± .00**	0.81 ± .00	0.63 ± .00	0.38	**0.37 ± .00**	0.35 ± .00	0.23 ± .00	0.15 ± .00
33	0.78	0.69 ± .00	**0.77 ± .00**	0.82 ± .00	0.62 ± .00	0.38	**0.34 ± .00**	0.33 ± .00	0.24 ± .00	0.14 ± .00
34	0.77	0.66 ± .00	**0.73 ± .00**	**0.81 ± .00**	0.63 ± .00	0.38	**0.36 ± .00**	0.32 ± .00	0.22 ± .00	0.16 ± .00
35	0.77	0.68 ± .00	**0.76 ± .00**	0.81 ± .00	0.63 ± .00	0.37	**0.33 ± .00**	0.32 ± .00	0.22 ± .00	0.14 ± .00
36	0.77	0.89 ± .00	0.86 ± .00	0.87 ± .00	**0.81 ± .00**	0.46	0.20 ± .00	0.17 ± .00	0.27 ± .00	0.13 ± .00
37	0.79	0.93 ± .00	0.87 ± .00	**0.81 ± .00**	0.82 ± .00	0.44	0.28 ± .00	0.20 ± .00	0.20 ± .01	0.14 ± .00
38	0.81	0.65 ± .00	**0.77 ± .00**	0.41 ± .00	0.72 ± .00	0.44	0.31 ± .00	0.21 ± .00	0.21 ± .00	0.15 ± .00
39	0.71	0.89 ± .00	0.85 ± .00	0.87 ± .00	0.82 ± .00	0.49	0.24 ± .00	0.21 ± .00	0.27 ± .00	0.15 ± .00
40	0.53	0.82 ± .00	0.60 ± .02	0.78 ± .01	0.73 ± .01	0.59	0.73 ± .02	0.40 ± .02	0.51 ± .04	0.43 ± .01
41	0.49	0.83 ± .01	0.62 ± .02	0.79 ± .00	0.75 ± .01	0.58	0.75 ± .03	0.39 ± .02	**0.55 ± .01**	0.43 ± .03
#i	-	8	2	**12**	3	-	**10**	7	2	0

▶ **Observation 5.** *The model EGNN2S shows the best performance in terms of community structure. While G2SAT also exhibits a reasonably good performance, the other models GCN2S and ECC2S perform very poorly on this feature. Moreover, in a large fraction of formulas none of the analyzed methods is able to correctly mimic their community structure.*

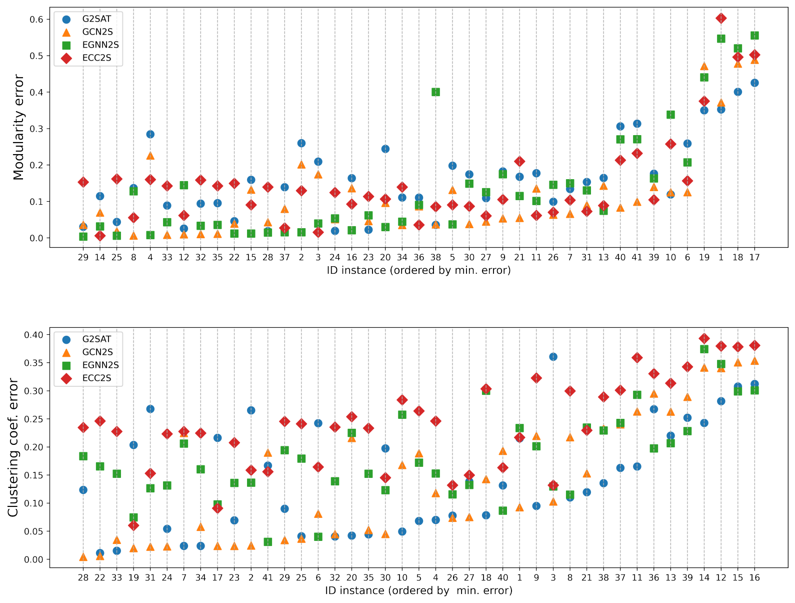

Figure 5 Error on the community structure and the clustering coefficient of deep generative models of realistic SAT instances w.r.t. to original benchmarks.

In the case of the clustering coefficient, the differences with respect to the original formulas are slightly bigger, suggesting that this structure is harder to mimic. For instance, in 21 of the 41 formulas, the clustering coefficient is not correctly reproduced by any model. In this case, the model with the best performance is GCN2S, but G2SAT also shows a reasonably good accuracy. On the contrary, the model ECC2S seems to be unable to learn this kind of structure, with a very poor performance in almost all the benchmarks. These results are summarized in Observation 6.

▶ **Observation 6.** *In the task of reproducing the clustering coefficient of input formulas, the model G2SAT shows the best performance. The model GCN2S also exhibits a relatively good performance, but the other models EGNN2S and ECC2S do not. However, it is worth noticing that no model is able to correctly reproduce this feature in 21 of the 41 analyzed formulas.*

Moreover, in Figure 5 we depict the performance of the analyzed deep generative models on the community structure (top) and the clustering coefficient (bottom). In these plots, each column represents a SAT instance of our dataset (ordered by error), and the Y values represent the error between the original instance and the generated ones (the lower, the better). We emphasize that the results of each instance must not be aggregated. Notice that our dataset has formulas with modularity ranging from 0.15 to 0.84; a model trained with all of them (as [36]) would possibly learn something spurious.

In summary, these results show that in many cases deep generative models based on GCN2S and EGNN2S are able to improve the results of the existing model G2SAT based on G2SAT, in terms of structural properties. However, it is also observed that there is room for improvements due to the large number of real-world formulas for which the structure is hardly mimicked.

6.2 Performance on SAT solver hardness

In the second part of our experimental evaluation, we analyze the hardness of the generated formulas with respect to the one in the original instances. The goal of this analysis is to compare the hardness and the satisfiability of the original formulas with respect to the instances generated by the analyzed models. To this end, we solve both the original and the generated formulas (for each generator) with several SAT solvers and analyze whether the percentages of SAT/UNSAT formulas are similar, and compare the cumulative CPU time and the rankings of the solvers in each dataset. Therefore, this analysis characterizes the ability of each generator to reproduce the hardness of the original instances. In contrast to [36], we only select CDCL SAT solvers specialized in application benchmarks. In [36], this experiment was carried out using solvers specialized in both real-world and random formulas, and it was observed that solvers specialized in random formulas exhibit a very poor performance on realistic SAT instances, as expected. However, those results do not provide much information about the ability of deep generative models to actually reproduce the hardness and the satisfiability of the input formulas.

In our study, we use the following CDCL SAT solvers: Glucose[5] [5], Lingenling[6] [6], MapleSAT[7] [20], MapleLCM[8] [21], and CaDiCaL[9] [7]. They all are executed with a timeout of 5000 seconds. In order to evaluate each generator, we measure the percentages of SAT and UNSAT formulas with respect to the input dataset, as well as the ranking of these five solvers on the generated formulas compared to the ranking obtained in the original benchmark. Additionally, we study whether the resulting formulas are as hard as the original ones in terms of timeouts and CPU time.

To compare two rankings x and y, we use the Kendall rank correlation coefficient τ:

$$\tau(x,y) = \frac{P-Q}{\sqrt{(P+Q+T)\cdot(P+Q+U)}} \tag{9}$$

where P and Q are the number of concordant and discordant pairs, respectively, whereas T and U are the number of ties in x and y, respectively. This coefficient returns a value in $[-1,1]$, where 1 indicates full concordance, and -1 indicates full discordance. Therefore, higher values of τ indicate a better ability to reproduce the practical hardness of the input set of formulas.

In Table 2 we report the performance of CDCL SAT solvers on the formulas generated by the analyzed models. First, we observe that the four models hardly generate satisfiable formulas. In particular, 47.3% of the input SAT instances are satisfiable. However, the four models are only able to generate around 8% of satisfiable formulas. This suggests that, although the randomizations that this kind of models introduce in the structure of the formulas have a small impact on its structure, they can have a major impact on the solution space they affect, which ultimately remove all the solutions until making the generated formula unsatisfiable, as summarized in Observation 7.

▶ **Observation 7.** *Deep generative models based on G2SAT are unable to reproduce the satisfiability of the input formula in most of the cases. In particular, most of the generated formulas are unsatisfiable regardless the satisfiability of the input formulas.*

[5] `https://www.labri.fr/perso/lsimon/glucose/`
[6] `https://github.com/arminbiere/lingeling`
[7] `https://bitbucket.org/JLiangWaterloo/maplesat/src/master/maplesat/`
[8] `http://sat-race-2019.ciirc.cvut.cz/solvers/MapleLCMDiscChronoBT-DL-v3.zip`
[9] `https://github.com/arminbiere/cadical`

Table 2 Results on SAT solver performance of deep generative models of realistic SAT instances. τ stands for the Kendall rank correlation coefficient, computed for both SAT and UNSAT formulas. TO stands for timeout within 5000 seconds.

	%SAT	τ(SAT)	%UNSAT	τ(UNSAT)	%TO	CPU time
Original formulas	47.3	–	42.9	–	9.8	44503.55
G2SAT	**8.1**	**0.2**	91.9	-0.6	0.0	1.38
GCN2S	7.4	0.0	**91.6**	**-0.4**	**1.0**	**14910.86**
EGNN2S	7.4	**0.2**	92.6	-0.6	0.0	1.48
ECC2S	7.3	0.0	92.7	-0.8	0.0	2.22

The previous observation is also an important drawback in order to mimic the hardness of the input benchmarks. In fact, only GCN2S is able to produce formulas of a reasonable hardness. However, these instances are much easier than the actual input set (the accumulated CPU time to solve these synthetic formulas is around three times lower than for real-world instances). The other three models always generate very easy formulas. We summarize these results in Observation 8.

▶ **Observation 8.** *SAT instances generated with deep generative models based on G2SAT are, in general, much easier than the original formulas used to train the models. The only model able to generate formulas of a certain hardness –although much easier than original instances– is GCN2S.*

Finally, if we compare the rankings obtained with the five CDCL SAT solvers through the τ coefficients for SAT and UNSAT instances, we observe that G2SAT and EGNN2S are the best models to reproduce the computational properties in satisfiable formulas, whereas GCN2S is the best in unsatisfiable ones. However, this observation must be interpreted cautiously due to the two previous observations, i.e., the inability of this kind of models to mimic both the satisfiability and the hardness of the input set of benchmarks. Therefore, we consider these results are not conclusive. Notice that this is an important drawback with respect to probabilistic models (e.g., SF [3], CA [13], and PS [14]), where both the satisfiability and the hardness of the generated formulas can be easily controlled.

As future work, we plan to extend this hardness and satisfiability analysis through the lens of MaxSAT. In particular, we conjecture that the reason that most of the generated formulas (for every model) are trivially UNSAT may be due to a small (sub)set of highly connected variables, which produces a very small UNSAT core in the generation phase (i.e., many links are repeatedly generated between those variables). As a consequence, the satisfiability of the formula can be altered, as well as its hardness. However, the rest of the formula may be able to preserve the global structure of the formula. Therefore, detecting those highly connected variables may be a first step towards improving deep generative models in terms of practical hardness.

7 Conclusions and future work

In this work, we presented an extensive experimental evaluation of four deep generative models of realistic SAT instances based on the G2SAT framework [36]. In this framework, SAT formulas are represented as bipartite graphs, and a number of node splitting operations are performed in order to generate training examples to train a model based on a GNN. Afterwards, in the generation phase, this trained model is used to generate random SAT instances whose computational properties are expected to be similar to the ones in the original benchmark. The four models differ in the graph representation of the formulas as well as in the computation of the node embeddings in the GNN.

The original G2SAT model uses the LCG graph representation of the SAT formula, and a convolutional layer based on GraphSAGE [16]. We also analyzed this model using a convolutional layer based on GCN [18]. Additionally, we proposed two models that consider edge features in order to represent all the semantics in the SAT formula (e.g., the sign of the variables in the clauses) in a more transparent manner. Specifically, they are based on EGNN [15], where a number of edge features is used to compute node embeddings, and on ECC [30], where edge filter weights are dynamically generated according to the neighborhood of each node. To the best of our knowledge, they are the first deep generative models of realistic SAT instances that consider edge features to represent the sign of the variables within the clauses.

In our experimental evaluation, we analyzed the performance of these models in terms of reproducing the structure and the hardness of an input set of instances. In terms of structural properties, we analyzed the community structure and the clustering coefficient, two crucial features extensively studied in the literature. Since this kind of models do not alter the number of variable occurrences, we omitted other kind of structures, such as scale-free. In terms of hardness, we analyzed the robustness of the models using five well-known CDCL SAT solvers.

Our analysis showed that, although the models exhibit a robust performance in terms of variability (the results show an almost negligible deviation), they all have a very poor scalability. In particular, the experimental evaluation was performed on a set of 41 real-world SAT instances of small size, and took more than 240 days. Furthermore, we observed that, in general, the models based on GCN and EGNN are the most robust ones in terms of structural properties. In particular, our generator based on EGNN is the most accurate model to reproduce the community structure of the input formulas it tries to mimic. However, reproducing the clustering coefficient appears to be a much more challenging task. On the other hand, we also observed that the four models have a very poor performance in terms of practical hardness. In particular, they all are biased towards the generation of very easy unsatisfiable instances. This is an important drawback with respect to probabilistic models (e.g., SF [3], CA [13], and PS [14]), where both the satisfiability and the hardness of the generated formulas can be easily controlled.

As a consequence, this experimental study showed some potential lines of future research on deep generative models of realistic SAT instances. Namely, the scalability of the models in order to train them with larger instances, the randomizations in the generation phase through the lens of MaxSAT in order to preserve the satisfiability of the formulas as well as their hardness, and a much more robust performance in terms of structural features (especially in the clustering coefficient), since we found a large number of real-world formulas for which no model is able to correctly reproduce their structure.

References

1 Michael Alekhnovich and Alexander A. Razborov. Satisfiability, branch-width and Tseitin tautologies. *Computational Complexity*, 20(4):649–678, 2011.

2 Carlos Ansótegui, Maria Luisa Bonet, Jesús Giráldez-Cru, Jordi Levy, and Laurent Simon. Community structure in industrial SAT instances. *Journal of Artificial Intelligence Research*, 66:443–472, 2019.

3 Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. On the structure of industrial SAT instances. In *Proc. of the 15th International Conference on Principles and Practice of Constraint Programming (CP 2009)*, pages 127–141, 2009.

4 Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. Towards industrial-like random SAT instances. In *Proc. of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 387–392, 2009.

5 Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *Proc. of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 399–404, 2009.

6 Armin Biere. CaDiCaL, Lingeling, Plingeling, Treengeling, YalSAT Entering the SAT Competition 2017. In *Proc. of SAT Competition 2017 : Solver and Benchmark Descriptions*, pages 14–15. Department of Computer Science Series of Publications B, University of Helsinki, 2017.

7 Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In *Proc. of SAT Competition 2020 : Solver and Benchmark Descriptions*, pages 51–53. Department of Computer Science Report Series B, University of Helsinki, 2020.

8 Vasek Chvátal and Bruce A. Reed. Mick gets some (the odds are on his side). In *Proc. of the 33rd Annual Symposium on Foundations of Computer Science (FOCS 1992)*, pages 620–627, 1992.

9 Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proc. of the Annual Conference on Neural Information Processing Systems (NIPS 2016)*, pages 3837–3845, 2016.

10 David K. Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Proc. of the Annual Conference on Neural Information Processing Systems (NIPS 2015)*, volume 28, pages 224–2232, 2015.

11 Fernando Gama, Elvin Isufi, Geert Leus, and Alejandro Ribeiro. Graphs, convolutions, and neural networks: From graph filters to graph neural networks. *IEEE Signal Processing Magazine*, 37(6):128–138, 2020.

12 Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proc. of the 34th International Conference on Machine Learning (ICML 2017)*, pages 1263–1272, 2017.

13 Jesús Giráldez-Cru and Jordi Levy. Generating SAT instances with community structure. *Artificial Intelligence*, 238:119–134, 2016.

14 Jesús Giráldez-Cru and Jordi Levy. Popularity-similarity random SAT formulas. *Artificial Intelligence*, 299:103537, 2021.

15 Liyu Gong and Qiang Cheng. Exploiting edge features for graph neural networks. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2019)*, pages 9211–9219, 2019.

16 William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proc. of the Annual Conference on Neural Information Processing Systems (NIPS 2017)*, pages 1025–1035, 2017.

17 Frank Hutter, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, 2014.

18 Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proc. of the 5th International Conference on Learning Representations (ICLR 2017)*, 2017.

19 Chunxiao Li, Jonathan Chung, Soham Mukherjee, Marc Vinyals, Noah Fleming, Antonina Kolokolova, Alice Mu, and Vijay Ganesh. On the hierarchical community structure of practical boolean formulas. In *Proc. of the 24th International Conference on Theory and Applications of Satisfiability Testing (SAT 2021)*, pages 359–376, 2021.

20 Jia Hui Liang. *Machine Learning for SAT Solvers.* PhD thesis, University of Waterloo, Ontario, Canada, 2018.

21 Mao Luo, Chu-Min Li, Fan Xiao, Felip Manyà, and Zhipeng Lü. An effective learnt clause minimization approach for CDCL SAT solvers. In *Proc. of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017)*, pages 703–711, 2017.

22 Yuri Malitsky. Instance-specific algorithm configuration. *Constraints*, 20(4):474, 2015.

23 Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Non-model-based algorithm portfolios for SAT. In *Proc. of the 14th International Conference on Theory and Applications of Satisfiability Testing (SAT 2011)*, pages 369–370, 2011.

24 David G. Mitchell and Hector J. Levesque. Some pitfalls for experimenters with random SAT. *Artificial Intelligence*, 81(1-2):111–125, 1996.

25 David G. Mitchell, Bart Selman, and Hector J. Levesque. Hard and easy distributions of SAT problems. In *Proc. of the 10th National Conference on Artificial Intelligence (AAAI 1992)*, pages 459–465, 1992.

26 Mark E.J. Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(2):026113, 2004.

27 Daniel Selsam and Nikolaj Bjørner. Guiding high-performance SAT solvers with unsat-core predictions. In *Proc. of the 22nd International Conference on Theory and Applications of Satisfiability Testing (SAT 2019)*, pages 336–353, 2019.

28 Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. In *Proc. of the 7th International Conference on Learning Representations (ICLR 2019)*, 2019.

29 João P. Marques Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, pages 131–153. IOS Press, 2009.

30 Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017)*, pages 29–38, 2017.

31 Ryan Williams, Carla P. Gomes, and Bart Selman. Backdoors to typical case complexity. In *Proc. of the 18th International Joint Conference on Artificial Intelligence (IJCAI 2003)*, pages 1173–1178, 2003.

32 Haoze Wu and Raghuram Ramanujan. Learning to generate industrial SAT instances. In *Proc. of the 12th International Symposium on Combinatorial Search (SOCS 2019)*, pages 206–207, 2019.

33 Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

34 Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. Predicting satisfiability at the phase transition. In *Proc. of 26th National Conference on Artificial Intelligence (AAAI 2012)*, 2012.

35 Lin Xu, Frank Hutter, Holger H. Hoos, and Keving Leyton-Brown. Satzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, 2008.

36 Jiaxuan You, Haoze Wu, Clark W. Barrett, Raghuram Ramanujan, and Jure Leskovec. G2SAT: learning to generate SAT formulas. In *Proc. of the Annual Conference on Neural Information Processing Systems (NeurIPS 2019)*, pages 10552–10563, 2019.

37 Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

A Real-world SAT instances used in the experimental analysis

In this appendix we provide the details of the 41 real-world SAT instances used in our experimental analysis. In particular, in Table 3 we detail these benchmarks as well as their characteristics (number of variables and clauses, and formula size).

Table 3 Features of the real-world SAT instances used in our experimental analysis, including number of variables (#vars), number of clauses (#clau.) and formula size (f. size).

ID	SAT Instance	#vars	#clau.	f. size
1	`aes_24_4_keyfind_2`	320	5424	33136
2	`aes_32_3_keyfind_2`	450	2204	7772
3	`aes_64_1_keyfind_1`	320	2088	8648
4	`AProVE07-03`	1317	7448	25057
5	`AProVE07-08`	2481	12625	39603
6	`bf0432-007`	473	2038	5989
7	`bmc-ibm-2`	119	573	1887
8	`bmc-ibm-5`	1068	6042	17685
9	`bmc-ibm-7`	860	4797	14634
10	`cmu-bmc-barrel6`	602	4533	20440
11	`cmu-bmc-longmult15`	1731	9791	32002
12	`countbitsarray02_32`	2202	9416	27628
13	`countbitsrotate016`	1122	4555	12887
14	`countbitssrl016`	1691	8378	25855
15	`g2-hwmcc15deep-6s399b02-k02`	3394	14336	40763
16	`g2-hwmcc15deep-6s399b03-k02`	3516	14837	42129
17	`i32a1`	370	9123	35584
18	`i32d2`	367	5116	20160
19	`i32e3`	284	4974	25280
20	`minor032`	751	5130	29532
21	`minxorminand032`	1751	13938	80477
22	`mrpp_4x4#10_20`	2135	21720	85231
23	`mrpp_4x4#10_9`	859	8333	31990
24	`mrpp_4x4#12_12`	1197	11737	45696
25	`mrpp_4x4#4_24`	2217	21443	83138
26	`mrpp_4x4#4_4`	208	1538	5340
27	`mrpp_4x4#4_5`	309	2517	8970
28	`mrpp_4x4#6_16`	1446	13684	53264
29	`mrpp_4x4#6_20`	1846	17576	68672
30	`mrpp_4x4#6_5`	330	2721	9733
31	`mrpp_4x4#8_8`	717	6773	24667
32	`mrpp_6x6#10_10`	2144	22792	88849
33	`mrpp_6x6#10_8`	1630	17009	65607
34	`mrpp_6x6#12_8`	1551	15788	60713
35	`mrpp_6x6#16_9`	2085	22775	88141
36	`mulhs016`	2589	10400	29015
37	`sat_prob_3`	2860	13998	44208
38	`sat_prob_83`	1759	8012	27063
39	`smulo016`	1459	6288	17894
40	`ssa2670-130`	82	327	1011
41	`ssa2670-141`	91	377	1161