Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Expressive GNNs for SAT Solving through Substructure Counting

Semester Thesis

Jeremy Gleixner

jgleixner@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Saku Peltonen, Joël Mathys
Prof. Dr. Roger Wattenhofer

June 17, 2025

Acknowledgements

I thank my supervisors Saku Peltonen and Joël Mathys for their guidance and support throughout this thesis, as well as Prof. Dr. Roger Wattenhofer for the opportunity to conduct this work. Further thanks go to the Computer Engineering and Networks Laboratory (TIK) for providing the computational resources used throughout the project.

Abstract

Graph Neural Networks (GNNs) have emerged as a promising tool for solving the Boolean Satisfiability Problem (SAT) with the help of machine learning techniques. To further the performance of GNN based SAT solvers, we propose the introduction of substructure counts to message passing neural networks. We implement cycle count node features in the G4SATBench benchmark framework and analyze their potential to increase the performance of SAT solvers. Our findings indicate that simple cycle counts introduced to initial embeddings of nodes may not be enough to significantly increase expressivity of GNNs in SAT solving settings.

Contents

Chapter 1

Introduction

The Boolean Satisifablity Problem (SAT) is a fundamental problem of computer science and logic. It was the first problem to be proven NP-complete [1] and has a wide range of real-world applications, including hardware and software verification, automated planning and scheduling, cryptographic analysis, optimization, and artificial intelligence, among others. Consequently, there has been significant interest over the last few decades in solving SAT instances efficiently. However, modern SAT solvers often rely on heuristics designed by hand, which is both time-consuming and challenging.

Graph Neural Networks (GNNs) have been proposed as a potential alternative to advance the performance of SAT solvers. In recent years, they have been evolving rapidly and have shown promising results in processing complex structured data. Multiple attempts have been made to leverage GNNs to solve SAT instances.

Despite their ability to analyze structured data, GNNs are fundamentally limited in their expressive power. They fail to capture certain structures in the graph due to the local and iterative nature of message passing. One proposed mitigation is to enhance the expressive power of these models by reintroducing certain structural information back into the model. In this work, we analyze whether cycle counts as node features offer an increase in performance for SAT solving tasks.

Chapter 2

Related Work

In the following sections, we briefly present an overview of the current state of modern SAT solvers and prior attempts at using GNNs for the problem.

2.1 SAT solvers

Despite the high relevance of the SAT problem, performance increases in modern SAT solvers remain limited, with a lack of major breakthroughs in recent years [2]. Solvers have been following established paradigms for multiple decades, with incremental improvements primarily drawn from sophisticated heuristics to efficiently navigate the vast search space. Prominent modern SAT solvers mainly use conflict-driven clause learning (CDCL) and local search (LS) methods [3]. Crafting such heuristics is challenging and requires a comprehensive understanding of SAT solvers. Additionally, these heuristics are often tailored to specific problems and do not generalize well to other SAT instances, requiring continuous effort for novel applications.

Considering these challenges, machine learning based approaches offer a significantly less labor intensive alternative to traditional methods. Especially deep learning techniques have been evolving rapidly and advancing into the field of combinatorial problems, opening up new possibilities for learning based SAT solvers [4].

2.2 SAT solving with GNNs

In recent years, GNNs have demonstrated strong capabilities in learning from structured data [5, 6, 7], inspiring a surge of research attempting to leverage them for SAT solving. GNNs offer an inherent way to process SAT instances, given that Boolean formulas in CNF can be represented as bipartite graphs.

One influential attempt at SAT solving with GNNs is NeuroSAT [8]. It demonstrates the ability of a pure GNN architecture to predict the satisfiability

of SAT instances and even produce satisfying assignments in some cases. A more recent example is QuerySAT [9], which introduces a query mechanism instead of calculating the output from input values alone.

In contrast to these standalone neural solvers, there have been several alternative approaches which combine GNN architectures with traditional SAT solvers. These *neural-guided* solvers introduce learning-directed heuristics, helping in the search procedure and reducing the need for manual intervention [10, 11, 12, 13].

In an attempt to create a systematic way to quantify the advancement in the field of GNN based SAT solvers, the benchmark framework G4SATBench was proposed [14]. G4SATBench provides the ability to evaluate GNN SAT solvers on a diverse range of datasets related to different areas of SAT problems. It re-implements popular GNN SAT solver architectures and supports multiple prediction tasks and training objectives to compare the SAT solving performance of different models. G4SATBench is the foundation of our work, as we extend it's functionality to support substructe counts as node features.

Chapter 3

Preliminaries

3.1 SAT Problem

A Boolean formula consists of Boolean variables, logical operators and parentheses. The variables can take the value of either *true* or *false*. Logical operators such as AND/conjunction ($\wedge$), OR/disjunction ($\vee$), and NOT/negations ($\neg$) connect these variables together and construct more complex expressions, which again evaluate to either *true* or *false*. A formula is labeled as *satisfiable* if there exists an assignment of Boolean values to its variables so that the entire formula evaluates to *true*. If there is no such assignment, the formula is labeled *unsatisfiable*. The Boolean Satisfiability (SAT) Problem asks, if a given formula is satisfiable.

3.1.1 CNF Formula

It is common to give Boolean formulas in *conjunctive normal form* (CNF), where the formula is written as conjunctions ($\wedge$) of clauses. Each clause consists of disjunctions of literals, where a literal is a Boolean variable or its negation. A Boolean formula in CNF form is satisfied if at least one literal of each clause evaluates to *true*.

The majority of modern SAT solvers operate on CNF inputs, including the G4SATBench framework and the work we present here.

3.1.2 Graph Representation

In order to utilize GNNs for SAT solving, we need a way to represent CNF formulas as graphs. G4SATBench provides two implementations for this task: LCG* and VCG*. Our work is using the VCG* format, which is a variation of the Variable-Clause Graph (VCG). The VCG* is a bipartite graph with variables on one side and clauses on the other side. A variable is connected to a clause with an edge if it appears in the clause of the underlying CNF formula. VCG* has two types of edges, corresponding to the polarity of the variable in the clause.

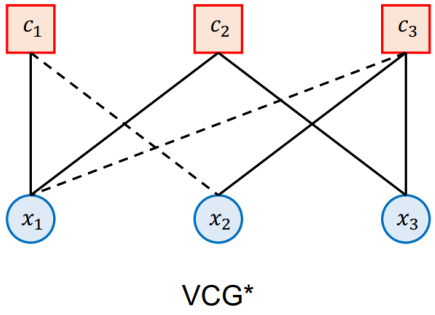

Figure 3.1: VCG* of the CNF formula $(x_1 \vee \neg x_2) \wedge (x_1 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$, reproduced from [14]

An example of this encoding can be seen in figure 3.1.

3.2 Graph Neural Networks

3.2.1 Limits on expressivity

GNNs based on message passing are fundamentally limited in their expressivity, in particular in their ability to distinguish certain non-isomorphic graphs. This is an inherent property of message passing. Nodes aggregate information from their local neighborhood using permutation-invariant aggregation functions. This process can be shown to be at most as expressive as the Weisfeiler-Lehman (WL) test, meaning in cases where the WL-test is unable to distinguish two graphs, GNNs also fail to do so. This restricts their ability to learn certain structural patterns in graph data.

3.2.2 Substructure Counts

As GNNs are blind to certain structural patterns, we lose potentially relevant information encoded in the structure of the graph. One proposed mitigation to this problem is to explicitly inject structural information directly into the message passing process. We count substructures in the graph and store this information as node or graph features, potentially providing useful information the model would not be able to learn otherwise. This information can then be propagated through normal message passing iterations. Substructure counts can be useful in tasks such as community detection, prediction of molecule properties, and classification of document structures. *Simple Cycles* are commonly used substructures for these applications.

3.2.3 Simple Cycles

In graph theory, a simple cycle, or just cycle, is a closed path where no node appears twice. The length of a cycle is the number of nodes on the path.

In the case of an undirected graph, two simple cycles are said to be distinct if they are **1)** not cyclic permutations of each other and **2)** not cyclic permutations of the other's reversal.

3.3 G4SATBench

In this section, we provide a brief overview of the functionality and implementation of the used benchmark framework G4SATBench. We cover only parts with relevance to our modifications.

3.3.1 Prediction Tasks

In the evaluation of our implementation, we used two types of prediction tasks: satisfiability predictions and satisfying assignment predictions.

Satisfiability

Satisfiability tasks are modeled as binary graph classifications tasks, where the graph representation of a given SAT instance is either classified as satisfiable or as unsatisfiable.

Assignment

Assignment tasks predict valid assignments for given SAT instances. This is implemented as binary node classification task, where the variables of the CNF formula are either classified as *true* or as *false*. Note that we only use satisfiable SAT instances for assignment tasks.

3.3.2 Models

G4SATBench implements four different GNN architectures for SAT solving: NeuroSAT, GCN, GGNN, and GIN. Our modifications are compatible with all models supporting the VCG* format, meaning all models expect NeuroSAT which only works with LCG*.

3.3.3 Loss Functions

For satisfiability tasks, G4SATBench implements binary cross-entropy loss between predictions and labels. For assignment tasks, three different training paradigms are provided: supervised, unsupervised 1 and unsupervised 2. We only briefly show the actual formulas here and refer to the G4SATBench paper for a more in-depth explanation [14].

Supervised

Analogously to the satisfiablilty task, supervised training loss for assignment tasks is implemented as binary cross-entropy loss between predictions and labels. However, predictions are performed as classifications for variables in this case and not for the entire graph. This means that the ground truth of the label is a single valid assignment and may not represent all possible valid assignments.

Unsupervised 1

The first unsupervised loss aims to differentiate and maximize the satisfiability value of the CNF formula [15]. It introduces smooth max and min functions given in (3.1) and minimizes the loss according to (3.2).

$$S_{\max}(x_1, x_2, \ldots, x_d) = \frac{\sum_{i=1}^{d} x_i \cdot e^{x_i/\tau}}{\sum_{i=1}^{d} e^{x_i/\tau}}, \quad S_{\min}(x_1, x_2, \ldots, x_d) = \frac{\sum_{i=1}^{d} x_i \cdot e^{-x_i/\tau}}{\sum_{i=1}^{d} e^{-x_i/\tau}}, \tag{3.1}$$

$$\mathcal{L}_\phi(x) = \frac{(1 - S(x))^\kappa}{(1 - S(x))^\kappa + S(x)^\kappa} \quad (\kappa \geq 1 \text{ is a predefined constant}) \tag{3.2}$$

Unsupervised 2

The second unsupervised loss is calculated as follows [9]:

$$V_c(x) = 1 - \prod_{i \in c^+} (1 - x_i) \prod_{i \in c^-} x_i, \quad \mathcal{L}_\phi(x) = -\log\left(\prod_{c \in \phi} V_c(x)\right) = -\sum_{c \in \phi} \log(V_c(x)), \tag{3.3}$$

c^+ and c^- are the sets of variables that appear in clause c with the respective polarity.

Initial Embeddings

G4SATBench supports two ways of initializing the embeddings of nodes in the GNNs: *Random* and *Learned*. *Random* initializes the embeddings of each node with random values, whereas with *Learned*, the initial values are trainable parameters.

To further test the expressivity of models, we introduce a third initialization mode *Ones* which initializes all embeddings to 1.

3.3.4 Evaluation

For Satisfiability, the testing accuracy is the fraction of correctly classified SAT instances. In assignment tasks, the testing accuracy is the fraction of SAT instances that received a satisfying assignment. We perform all read-outs for the satisfying assignment using the standard decoding option provided by G4SATBench. All configurations are run with 3 different seeds.

Chapter 4

Implementation

In the following sections, we provide a detailed explanation of our implementation.

4.1 Datasets

4.1.1 Base Datasets

We tested our implementation on 6 out of the 7 base datasets provided in G4SATBench, shown in Figure 4.1. We had to exclude the PS dataset due to non-functioning generation code. Furthermore, we were unable to evaluate the *Patterns* edge mode on the CA dataset due to the inherently high cycle count of the dataset, rendering the full traversal of each cycle computationally unfeasible.

4.1.2 Regular Vertex Cover Dataset

In addition to the base datasets of G4SATBench, we created a variation of the k-Vercov dataset termed k-Vercov-reg. In k-Vercov-reg, we generate vertex cover

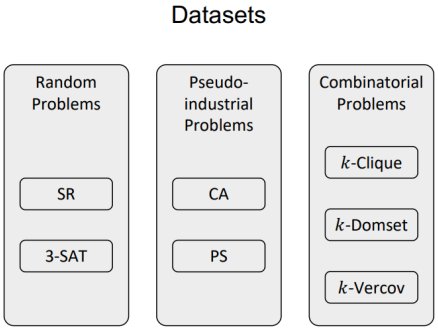

Figure 4.1: Overview of G4SATBench datasets, reproduced from [14]

problems of regular graphs with a set degree. We then find the smallest possible k for each graph and create vertex cover instances for k−2, k−1, k, k+1. Consequently, two of the resulting SAT instances will be satisfiable, and the other two will be unsatisfiable. We fixed the degree of regular graphs to 3 for all experiments.

Our rationale for including this dataset is the relevance of substructure counts for the problem. Cycle counts of length 6 in the SAT graph encoding should correlate with the number of triangles a node is part of in the original graph, which seems relevant to the problem.

4.2 Node Features

In the original G4SATBench implementation, embedding of nodes relied only on initialization method and aggregated data via message passing. We introduce individual node features depending on the local structures of the graph.

4.2.1 Cycle Counts

Our used substructure count is cycle counting. Concretely, we get all the simple cycles of a graph of a specified set of cycle lengths. For each node, we then count the number of cycles it is a part of and store the information as node features for each specified length.

4.2.2 Edge Modes

As described in Section 3.1.2, the used VCG* graph representation has positive and negative edge types, depending on the polarity the variable has in the respective clause. Since that differentiation is highly relevant to the satisfiability problem, failing to take edge types into account when counting cycles could lead to the loss of potentially valuable information.

We introduce four different kinds of edge modes, relating to how edge types are handled in cycle counting:

- Combined
- Positive
- Negative
- Patterns

Combined

Edge mode *Combined* is the general case where we ignore edge types. For this mode, the graph is converted to a default bipartite graph with a single edge type and both positive and negative connections result in a normal edge. Cycles are then counted on the resulting graph.

Positive and Negative

Similarly to the *Combined* mode, we convert the graph to a default bipartite graph with a single edge type. However, in contrast to the previous mode, here we dismiss edges that are not of the specified type. The resulting graph contains only a subset of the edges of the original graph. Cycles are then counted on this new subgraph.

4.2.3 Patterns

In this mode, we evaluate all cycles present in the graph, but we discriminate based on the edge type pattern of the cycles. This means we count cycles separately for every possible combination of positive and negative edges. We achieve this through the following steps:

Step 1: We calculate all possible combinations of edge types depending on the relevant cycle length. The resulting combinations are termed *Pattern Groups*. Note that we count the reversal of a pattern as the same pattern. This is due to the undirected nature of the graphs we are working with.

Step 2: We find all cycles of the relevant length as we do in the *Combined* mode.

Step 3: We traverse the cycles found and track the type of edges used.

Step 4: For each node in the cycle, we rotate the pattern according to the position of the node in the cycle. We then check to see which pattern group the edge pattern belongs to. Note that, as previously mentioned, we also check for the reversal of the pattern.

Step 5: Finally, we increase the count of the respective pattern group for that node.

4.2.4 Merging Feature Data with Embeddings

In order to maintain full compatibility with all VCG* models of G4SATBench, we merge the calculated node features with the initial embeddings before passing them through the models. To achieve this, we introduce an additional linear layer projecting the initial embeddings vector and the node features vector to

the dimensionality of the embeddings of the model. We mark the linear layer as trainable parameter.

Chapter 5

Results & Evaluation

In this chapter, we present and evaluate the results of our experiments. Our evaluation is divided into two parts: model performances on the datasets included in G4SATBench, and model performances on our implemented regular vertex cover dataset. All configurations are run with 3 different seeds, the standard deviation is indicated as error bars in the figures. All cycle counts on the G4SATBench datasets have been run with cycle length of 4. Experiments on our added regular vertex cover dataset were run with cycle length of 6 due to the explanation given in Section 4.1.2.

5.1 G4SATBench Datasets

5.1.1 Satisfiabilty Prediction

Table 5.1 contains the results of the satisfiability prediction tasks. We provide a graphical representation in Figure 5.1. We evaluated three different edge types for the node features. *Combined, Pos, Neg* means that each node has 3 features with substructure counts for the respective edge types. Refer to Section 4.2.2 for a detailed explanation.

From the data, we can see that the models with node features do not perform better than the models without. Furthermore, we note that certain datasets are more prone to random variation in accuracy outcomes between runs.

Table 5.1: Testing accuracy of satisfiability prediction tasks.

Init Emb	**Node Features**	**Dataset**				
		3-sat	**k-clique**	**k-domset**	**k-vercov**	**sr**
Learned	None	0.83	0.77	0.94	0.98	0.73
	Combined/Neg/Pos	0.85	0.72	0.79	0.96	0.68
	Patterns	0.86	0.65	0.86	0.97	0.61
Ones	None	0.81	0.73	0.87	0.88	0.59
	Combined/Neg/Pos	0.80	0.67	0.89	0.98	0.78
	Patterns	0.87	0.62	0.74	0.94	0.81

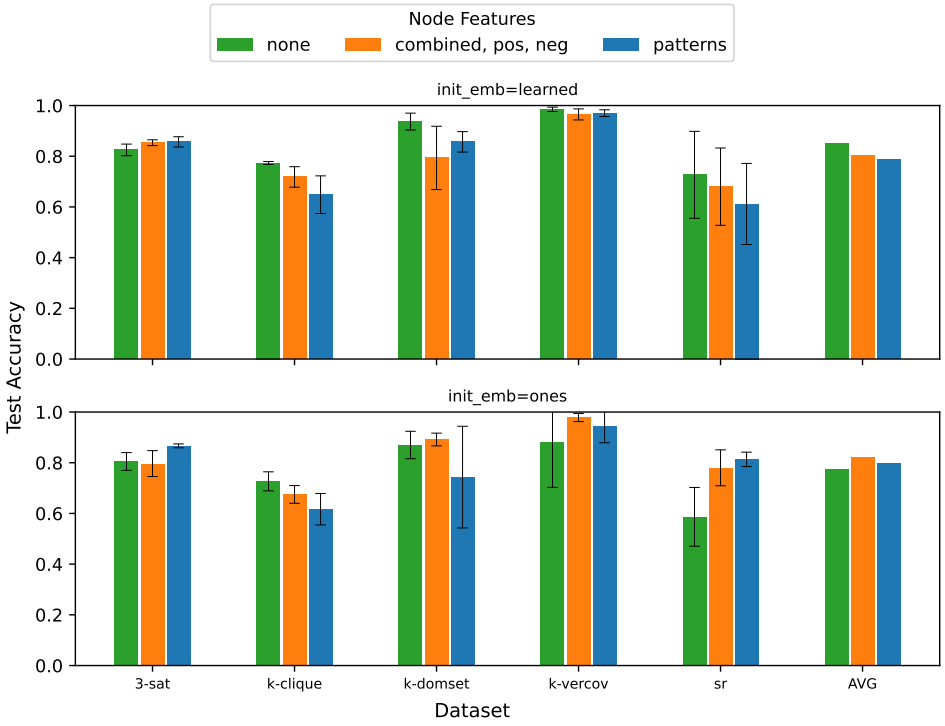

Figure 5.1: Testing accuracy of satisfiability prediction tasks. Colored bars represent the different cycle count node features used.

5.1.2 Assignment Prediction

We present the results for the assignment prediction tasks in Table 5.2, a graphical representation can be seen in Figure 5.2. In addition, we provide a comparison that includes the CA dataset and in turn excludes the edge mode *Patterns* in Figure 5.3. Reasons for the exclusion of the CA dataset can be found in Section 4.1.1.

Again, we did not observe an increase in the test accuracy with the inclusion of node features of either type. Note that the runs on k-Domset and k-Vercov dataset fail for the Unsupervised 1 loss and show significant variations for the Unsupervised 2 loss. This is in line with previous findings of the authors of the G4SATBench paper [14]. Lastly, we point out the test accuracy on the CA dataset in the Supervised loss case, presented in Figure 5.3. A possible explanation for this low accuracy is that the CA dataset by design has many possible valid assignments. The supervised loss, however, only trains the model on one possible solution and thus may hinder the ability of the model to generalize well.

Table 5.2: Testing accuracy of assignment prediction tasks.

Init Emb	Loss	Node Features	Dataset				
			3-sat	k-clique	k-domset	k-vercov	sr
Learned	SUP	None	0.66	0.53	0.46	0.71	0.56
		Combined/Neg/Pos	0.59	0.48	0.33	0.65	0.58
		Patterns	0.59	0.55	0.41	0.64	0.55
	UNSUP 1	None	0.78	0.41	0.00	0.00	0.69
		Combined/Neg/Pos	0.77	0.42	0.00	0.00	0.65
		Patterns	0.77	0.55	0.00	0.00	0.65
	UNSUP 2	None	0.76	0.61	0.32	0.31	0.64
		Combined/Neg/Pos	0.73	0.64	0.26	0.52	0.65
		Patterns	0.74	0.59	0.00	0.31	0.62
Ones	SUP	None	0.63	0.41	0.29	0.64	0.60
		Combined/Neg/Pos	0.57	0.57	0.32	0.68	0.59
		Patterns	0.61	0.41	0.35	0.65	0.52
	UNSUP 1	None	0.76	0.39	0.00	0.00	0.69
		Combined/Neg/Pos	0.76	0.38	0.00	0.00	0.67
		Patterns	0.75	0.49	0.00	0.00	0.64
	UNSUP 2	None	0.74	0.59	0.62	0.56	0.64
		Combined/Neg/Pos	0.73	0.49	0.31	0.52	0.62
		Patterns	0.72	0.62	0.60	0.31	0.58

5.2 Regular Vertex Cover

5.2.1 Satisfiabilty Prediction

We present the result of the satisfiability prediction tasks evaluated on the regular vertex cover dataset in Table 5.3 with a graphical representation in Figure 5.4. In this case, we only use edge mode *Combined* for node features since the relevant information is encoded only with negative literals in k-Vercov-reg. Recall that we use cycle length of 6 for k-Vercov-reg.

We see no improvement on testing accuracy when using node features on the regular vertex cover dataset. While there appears to be a trend of features negatively impacting accuracy, the correlation did not reach significance.

We mention that for all results, the accuracy was either 1, meaning fully correct classification of all SAT instances, or 0.5, meaning unable to make any predictions beyond random chance. This indicates that the models evaluated on the regular vertex cover either correctly learn to classify all instances or none. We hypothesize that this is due to regular nature of the dataset.

5.2.2 Assignment Prediction

The assignment prediction for the regular vertex cover failed for all attempted runs, regardless of the configuration. As shown in Table 5.4, the models produced no valid assignments. We omit the graphic representation of these results.

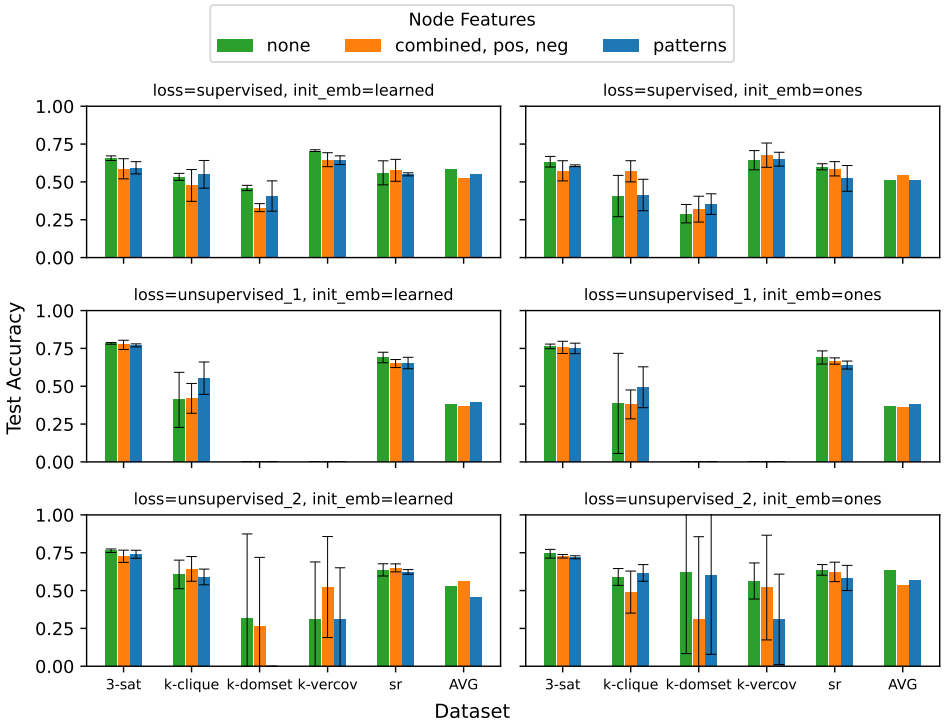

Figure 5.2: Testing accuracy of assignment prediction tasks. Colored bars represent the different cycle count node features used.

Table 5.3: Testing accuracy of satisfiability prediction tasks, evaluated on the k-Vercov-reg dataset.

Node Features	**Init Embeddings**	**Accuracy**	
		Mean	**Std**
None	Learned	1.00	0.00
	Ones	0.67	0.29
	Random	1.00	0.00
Combined	Learned	0.83	0.29
	Ones	0.67	0.28
	Random	0.67	0.28

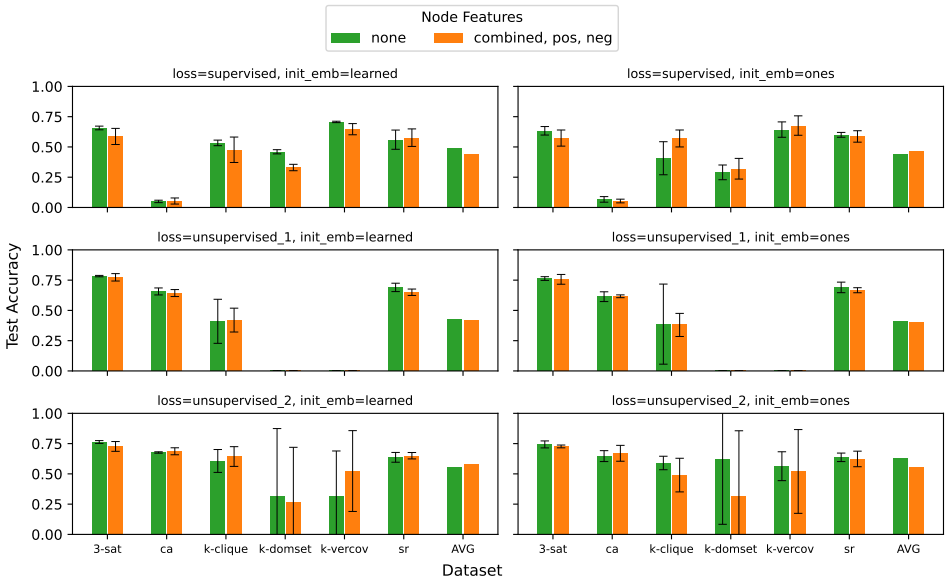

Figure 5.3: Testing accuracy of assignment prediction tasks. Colored bars represent the different cycle count node features used. Variant of Figure 5.2 with dataset CA included and pattern cycle count features excluded.

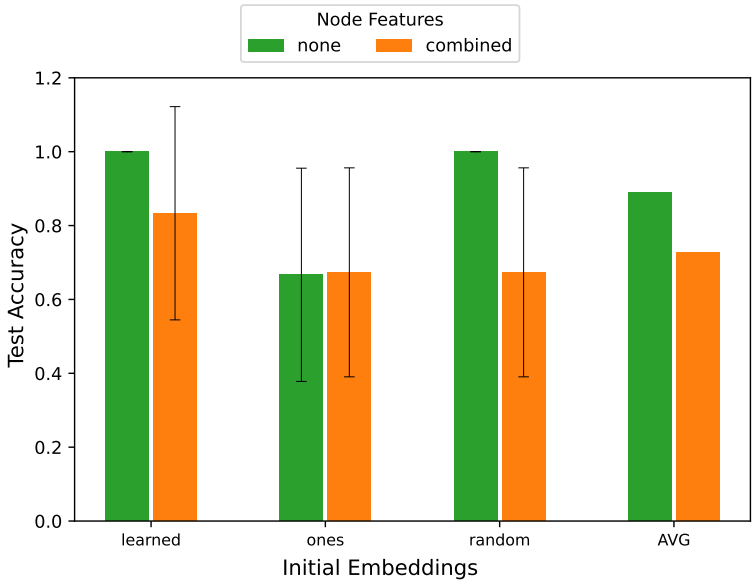

Figure 5.4: Testing accuracy of satisfiability prediction task, evaluated on the regular vertex cover dataset. The color of the bars indicate whether cycle count node features were used.

Table 5.4: Testing accuracy of assignment prediction tasks, evaluated on the regular vertex cover dataset. All runs were performed using the *Learned* initialization

Node Features	**Loss**	**Accuracy**	
		Mean	**SD**
None	Supervised	0.00	0.00
	Unsupervised 1	0.00	0.00
	Unsupervised 2	0.00	0.00
Combined	Supervised	0.00	0.00
	Unsupervised 1	0.00	0.00
	Unsupervised 2	0.00	0.00

Chapter 6

Conclusion

6.1 Summary

In this work, we analyze the viability of cycle count node features to increase the performance of GNN based SAT solvers. We introduce node features to G4SATBench and provide functionality to count cycles while taking into account the different kind of edge types of the VCG* representation. We also introduce a new dataset with regular vertex cover instances of set degree and provide an additional initialization of node embeddings with ones.

Our findings indicate that providing cycle counts as initial node features alone is not enough to significantly increase the performance of GNNs in the setting of SAT solving. We hypothesize that differentiating based on edge type might not be enough and more advanced substructure counts might be necessary to achieve an increase in performance. Further, we acknowledge the possibility that only merging the node features in the initial embeddings of the models might not be enough, and more aggressive injection of substructure information could potentially yield better results. Lastly, the datasets implemented in G4SATBench might not be a good representation of SAT instances where substructure counts provide useful information.

6.2 Future Work

Although our results did not suggest an increase in performance, we believe that further exploration of the topic is merited. We propose the following starting points for future work in this area:

- Expand the parameters of cycle counts to other lengths than the ones we explored.
- Introduce node features into message passing beyond initialization of embeddings.

- Introduce substructure counts beyond counting simple cycles of different edge patterns.
- Evaluate on datasets with instances where substructure counts are known to provide crucial information.

Bibliography

[1] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, ser. STOC '71. New York, NY, USA: Association for Computing Machinery, 1971, p. 151–158. [Online]. Available: https://doi.org/10.1145/800157.805047

[2] W. Guo, H.-L. Zhen, X. Li, W. Luo, M. Yuan, Y. Jin, and J. Yan, "Machine learning methods in solving the boolean satisfiability problem," *Machine Intelligence Research*, vol. 20, no. 5, p. 640–655, Jun. 2023. [Online]. Available: http://dx.doi.org/10.1007/s11633-022-1396-2

[3] N. Eén and N. Sörensson, "An extensible sat-solver," in *SAT*, ser. Lecture Notes in Computer Science, E. Giunchiglia and A. Tacchella, Eds., vol. 2919. Springer, 2003, pp. 502–518. [Online]. Available: http://dblp.uni-trier.de/db/conf/sat/sat2003.html#EenS03

[4] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: a methodological tour d'horizon," 2020. [Online]. Available: https://arxiv.org/abs/1811.06128

[5] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," 2017. [Online]. Available: https://arxiv.org/abs/1511.05493

[6] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2017. [Online]. Available: https://arxiv.org/abs/1609.02907

[7] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" 2019. [Online]. Available: https://arxiv.org/abs/1810.00826

[8] D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, and D. L. Dill, "Learning a sat solver from single-bit supervision," 2019. [Online]. Available: https://arxiv.org/abs/1802.03685

[9] E. Ozolins, K. Freivalds, A. Draguns, E. Gaile, R. Zakovskis, and S. Kozlovics, "Goal-aware neural sat solver," in *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, Jul. 2022, p. 1–8. [Online]. Available: http://dx.doi.org/10.1109/IJCNN55064.2022.9892733

[10] D. Selsam and N. Bjørner, "Guiding high-performance sat solvers with unsat-core predictions," 2019. [Online]. Available: https://arxiv.org/abs/1903.04671

[11] W. Zhang, Z. Sun, Q. Zhu, G. Li, S. Cai, Y. Xiong, and L. Zhang, "Nlocalsat: Boosting local search with solution prediction," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, ser. IJCAI-PRICAI-2020. International Joint Conferences on Artificial Intelligence Organization, Jul. 2020, p. 1177–1183. [Online]. Available: http://dx.doi.org/10.24963/ijcai.2020/164

[12] J. M. Han, "Enhancing sat solvers with glue variable predictions," 2020. [Online]. Available: https://arxiv.org/abs/2007.02559

[13] Z. Li and X. Si, "Nsnet: A general neural probabilistic framework for satisfiability problems," 2022. [Online]. Available: https://arxiv.org/abs/2211.03880

[14] Z. Li, J. Guo, and X. Si, "G4SATBench: Benchmarking and advancing SAT solving with graph neural networks," *Transactions on Machine Learning Research*, 2024. [Online]. Available: https://openreview.net/forum?id=7VB5db72lr

[15] S. Amizadeh, S. Matusevych, and M. Weimer, "Learning to solve circuit-sat: An unsupervised differentiable approach," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:53544639