Machine Learning Model for Selecting Assignments of Variables for SAT Problems

Jonathan Oliva
Department of Computer Science, University of York, United Kingdom

Peter Nightingale
Department of Computer Science, University of York, United Kingdom

Abstract

Machine learning (ML) methodologies have been evolving as they have been applied in various fields. One area of application is Boolean Satisfiability problems (SAT problems), which has many applications including machinery scheduling in assembly lines, or timing analysis in digital circuits. In general, SAT solvers are applied to reduce costs and optimize resource production in industry.

In this area, most ML research has focused on predicting the satisfiability of SAT instances or predicting an UNSAT core. The challenging problem of predicting a value for each variable in a SAT instance has not been the main focus of research in this area. However, the ability to predict values for SAT variables with good accuracy could be applied in several ways to improve performance of conventional SAT or MaxSAT solvers.

In this paper we propose VAPS-GCN (Variable Assignment Prediction for Satisfiability using a Graph Convolutional Network), a machine learning model to aid in the prediction of variable assignments for SAT problems. We adapted and improved a model for Abstract Argumentation to work on SAT problems. The process of the proposed model starts with converting a problem from CNF into a heterogeneous graph representation of SAT problem components. We propose a specialized network of GCN layers for aggregation between three distinct kinds of nodes. In the final layer, the model produces a prediction of the probability of each literal appearing in a solution. The model is first trained on small random SAT instances, then fine-tuned for the specific problem of interest (where the training data may be scarce).

The model produces a prediction of the assignment of each variable, combined with its confidence score. The output of the model may be useful in several ways, such as warm-starting solvers or deciding on some variables to assign prior to running a classical solver. In this paper, we evaluate the model on two classes of MaxSAT problems. In each case we fine-tune the model for the problem class, then use it to pre-assign some of the variables prior to applying a classical MaxSAT solver. We show substantial performance improvements with a very small loss of optimality.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming

Keywords and phrases Machine Learning, Artificial Intelligence, Graph Neural Network, Boolean Satisfiability Problems, Combinatorial Problems

Funding *Peter Nightingale*: Partially supported by UK EPSRC grant EP/W001977/1

1 Introduction

SAT and MaxSAT are foundational problems in AI with many applications. In this paper we focus on predicting values for the Boolean variables in SAT and MaxSAT problems, with the goal of speeding up conventional solvers for these problems. The ability to predict values of SAT or MaxSAT variables with good accuracy could potentially be applied in several ways to improve performance of conventional solvers. To give an example, a solver could be warm-started by providing an initial phase for each variable for use in a phase heuristic [28]. Furthermore, if values can be predicted alongside a measure of confidence, then the predicted assignments with the highest confidence can be applied to reduce the size and difficulty of SAT and MaxSAT instances (with the risk of making the instance unsatisfiable or ruling out all the optimal solutions).

In this area, most ML research has focused on predicting the satisfiability of SAT instances or predicting an UNSAT core. The challenging problem of predicting a value for each variable in a SAT instance has not been the main focus of research so far. Therefore a key motivation for this research is to explore whether values can be predicted with good accuracy.

There are several approaches to help with the process of solving SAT problems, for example, end-to-end machine learning models such as NeuroSAT [31] and DG-DAGRNN [2], both of which focus on predicting satisfiability of SAT instances. Information about satisfiability can be used by traditional solvers to assist in reducing solving time by choosing an appropriate configuration for the solver. Approaches such as SATformer [32] predict satisfiability and the UNSAT core composed of clauses and their literals. An UNSAT core (if one exists) provides an explanation for the unsatisfiability of an instance. Our research is broadly concerned with applying machine learning in solving constraint satisfaction problems that are encoded into SAT or MaxSAT. In this and the next section, a brief introduction and an overall explanation are provided.

1.1 Constraint Satisfaction Problems

The Constraint Satisfaction Problem (CSP) consists of a problem composed of a finite set of variables, each of which is associated with a finite domain, and a set of constraints that restricts the values which variables can simultaneously take [36]. CSPs are not only present in Artificial Intelligence (AI), but are also prevalent in industry. A common example is scheduling problems where there are shared and sometimes limited resources. Typically in a scheduling problem the start times of a set of tasks are decided while respecting resource constraints and precedences between tasks [36]. A Constrained Optimisation Problem (COP) is a CSP with an optimisation function (to be minimised or maximised). For example, a scheduling problem could minimise the end time of the final task.

There are many applications outside AI where constraint programming can be and is applied. For instance, in molecular biology and medicine for building 3D models of proteins, testing their properties regarding specific settings and relations between atoms. Apt surveyed applications of constraint programming [4], including helping to locate faults in hardware, examining the quality of a product, and helping to reduce the time and cost of processes by finding the most optimal schedules. Apt covered practical problems in electrical engineering and manufacture.

1.2 Machine Learning Applied to SAT

Current SAT solvers are remarkably powerful and successful on structured instances, but can still exhibit exponential scaling as the size of instances is increased. Machine learning models have the capacity to learn and to exploit patterns present in SAT instances. Machine learning methods based on graphs have attracted attention recently in CP and SAT, as well as other areas of combinatorial solving and optimisation. Research on SAT with machine learning has focused on predicting satisfiability, or predicting the clauses and their literals that cause an instance to be unsatisfiable (i.e., predicting an UNSAT core) [14].

Several distinct ML models have been developed for SAT. Examples include the unsupervised learning model of QuerySAT [27] to determine an instance as satisfiable by finding first the correct assignment of variables, or declaring a problem as unsatisfiable if no such assignment can be found. NeuroSAT [31] is a supervised ML model to predict satisfiability using a message-passing network. Later, NeuroCore [30] (based on NeuroSAT) is an example of an ML model used as a branching heuristic for a Conflict-Driven Clause Learning (CDCL) solver (MiniSAT in this case). In NeuroCore, the key idea is to direct the solver towards finding an UNSAT core.

In the next section we will discuss some of the works most relevant to this research. These ML-based approaches to be described in this paper follow three major architectures. Recurrent Neural Networks (RNN) known by their capacity to remember previous processed data that passed by [13]. Graph Neural Networks (GNN), a kind of neural network designed to find patterns of incoming graphs, making generalizations of graph structured data [38]. Reinforcement Learning (RL) where an agent is rewarded or punished for its actions in an environment [33]. In addiction, GNN models described in this paper tend to follow a remarkable approach in ML applied to SAT solving, using a subtype of GNN called Message Passing Neural Networks (MPNN). MPNN are models where vertices of a graph update their states by incoming messages regarding their neighbouring vertices [12, 34].

2 Background and Related Work

In this section, we present an outline of the work in the area of machine learning for SAT solving. We focus particularly on those methods with an approach similar to ours to predict assignments for variables. We also cover ML models for the maximum satisfiability problem.

2.1 Preliminaries

2.1.1 Conjunctive Normal Form

A formula in Conjunctive Normal Form (CNF) consists of a conjunction of clauses, where each clause is a disjunction of literals (a variable or its negation) [31]. In a more formal definition, a formula is in CNF when it is of the form:

$$C_1 \land C_2 \land C_3 \land ... \land C_n$$

With each conjunct C_i being a disjunction of literals:

$$L_{i1} \lor L_{i2} \lor L_{i3} \lor ... \lor L_{ij}$$

Where L_{ij} as a literal that is a variable x_{ij} or its negation $\neg x_{ij}$ [16].

2.1.2 Boolean Satisfiability

A Boolean Satisfiability problem (SAT problem) aims to determine whether there exists a way of assigning variables *True* or *False* values so that a given Boolean formula evaluates to *True*. If it succeeds, then the formula is called satisfiable [14].

Following the Cook–Levin theorem, SAT was the first proven NP-Complete problem in computational complexity theory. Decision problems in NP can be reduced to NP-Complete problems, and they can be solved in polynomial time on a non-deterministic Turing machine [14, 10]. The theorem opens a window of possibilities. It explains that combinatorial problems can be translated into SAT problems. Furthermore, SAT problems can be translated into a CNF formula. It offers opportunities to research methods to resolve a variety of problems, by letting researchers represent complex problems into the restricted form of an SAT problem that with a SAT solver can get a resolution. A SAT solver is an algorithm to solve SAT problems [14, 10].

2.2 Machine Learning Approaches to SAT

NeuroSAT [31] is an approach to predicting satisfiability that has been influential and is widely cited. It is a Graph Neural Network (GNN) and Long Short-Term Memory (LSTM) model based on the message passing strategy of MPNN to communicate and update nodes, which represent clauses and literals of an SAT problem. It uses an embedding layer for capturing network activations. A model trained on random problems (3-SAT instances), it predicts whether a problem is satisfiable. NeuroSAT is relevant to this paper because of its description of the possibility of obtaining the assignment of variables from the embedding when a problem is satisfiable. However, it only offers a brief description about how to implement this method, being a model and paper more focused on prediction of satisfiability.

An interesting work is DG-DAGRNN [2], an unsupervised learning approach to resolve a special kind of SAT problems, the Circuit-SAT. It explores and implements a mimicry of a reinforcement learning approach to train an end-to-end model by means of an exploit-exploit strategy. In addition, it uses an embedding architecture to capture the patterns typically present in a Circuit-SAT problem. Finally, instead of focusing in determining if a problem is satisfiable such as NeuroSAT, it finds the solutions. Once it discovers the solutions, it classifies a problem as satisfiable. This strategy avoids false positives generated by models. However, the model design is focused in a particular kind of SAT problem, making it difficult to adapt to others.

Another work of relevance for this research is QuerySAT [27], a model based on NeuroSAT [31] and DG-DAGRNN [2]. It provides solutions to SAT problems by making use of a query mechanism which helps the model to learn the current structure of a problem in runtime by using a loss function to evaluate several trials produced for feedback about the solution for variable assignments. The process is repeated several times in a recurrent step iteration. Nonetheless, there are crucial differences with MPNN strategy [31, 14], the query mechanism replaces the step associated with message passing from variables to clauses [27]. Furthermore, a crucial difference with NeuroSAT, QuerySAT was designed to learn the correct assignment for the variables, once it found a solution, it checks whether the assignments produced a satisfiable state in the problem. However, because of its training in an unsupervised learning method of satisfiable problems, detecting unsatisfiability is a challenge for the model tending to become an almost infinite loop [27].

A remarkable work is SATformer [32], it is a model based on attention networks to solve SAT problems. However, its focus is on UNSAT problems, especially the UNSAT core, the main group of variables and clauses that make a problem unsatisfiable. It uses an architecture formed by a combination of transformers and a graph neural network to calculate a prediction score for clauses. Internally, the model sorts and categorizes the clauses in incremental order and in groups to compare them later on. This methodology is what enables their model to predict the most likely clauses and variables to be part of the UNSAT core. Furthermore, it allows us to predict whether a problem has a solution.

2.3 Learning to Prune

A remarkable approach for this research is learning to prune for constraint programming (CP-LTP) [1]. In a Learning to Prune (LTP) approach, an ML model is trained to predict variable assignments that are likely to be part of an optimal solution. Each predicted assignment is accompanied by a confidence score. To apply LTP to an unseen instance, the set of predicted assignments are computed then the ones with the highest confidence scores are applied to the instance prior to solving with a conventional solver (e.g. a MaxSAT solver).

The aim is to sparsify the instance, speeding up solving while having minimal or no impact on the optimality of the solution found. The CP-LTP approach is currently restricted to boolean decision variables, and is trained for a problem class. CP-LTP first translates a constraint model into a graph representation. Then a feature vector is computed for each decision variable from the graph representation (using a standard graph embedding), and some custom features for the decision variable are appended. A simple ML model is trained to predict a boolean value (with a confidence score) for a decision variable given the feature vector of the variable. To apply a trained CP-LTP model to a new instance of the same class, the predicted assignments are generated for each boolean variable. They are sorted by confidence, and the most confident predictions are applied to the problem instance prior to solving. In contrast to our work, CP-LTP is focused on individual variables and uses a relatively simple ML model (gradient boosted trees).

In this work, we also use an LTP methodology to predict assignments for MaxSAT instances, looking to find a good solution in less processing time and with a modest optimality gap compared to the real optimal value of the instance. An important difference is that the mentioned CP-LTP approach is a general framework that allows utilizing LTP methodology on problems expressed in a CP modelling language while our ML model does not work directly with a CP modelling language, but instead generalizes SAT instances presented in CNF.

2.4 Machine Learning Approaches to MaxSAT

MaxSAT adds optimisation to SAT. Current research on practical MaxSAT solving focuses on partial weighted MaxSAT, where the CNF clauses are divided into hard clauses (which must be satisfied), and soft clauses with a weight. An optimal solution to partial weighted MaxSAT maximises the sum of the weights of the satisfied soft clauses.

A remarkable work on assignment prediction is the research of Tönshoff et al. [35]. Their model is defined for binary COP, and was evaluated on Max-2-SAT instances (among others). Training is unsupervised and uses a set of instances of a given problem class, and the model is a randomised recurrent neural network based on LSTM. Some promising results are reported for four problem classes when compared to classical solvers such as CPLEX and Loandra. Their approach has a hyperparameter for the number of iterations (where each iteration uses the same model parameters), while we use a GCN with a fixed depth and the layers do not share parameters.

InitPMS [23] is proposed to work with local search solvers for MaxSAT by aiding in the initialization of variables as an alternative to random initialization. Simply put, it is an ML approach to predict assignments for partial weighted MaxSAT. It is trained in three rounds, where the first two focus on hard clauses while the final one trains with both hard and soft clauses.

The InitPMS model is formed by four components or steps. First, a graph that incorporates a differentiation between nodes representing hard and soft clauses, and an adjacency matrix as final product of this graph. Second, a differentiation and extraction of information to create vectors (features) of the hard and soft clauses. Third, exchange of information between vectors. The fourth and last component is a calculation of the probability that a variable is *True* in the optimal solution.

InitPMS is the most similar ML model to our work. However, its focus is on the initialization of variables in a local search MaxSAT solver, while our approach focuses on reducing space search by predicting an early direct assignment of variables. Another difference is that our model architecture is composed of a series of GCN heterogeneous convolution layers while InitPMS applies RGCN [29] to recognize the edges representing the hard and soft clause relations, and GGSN or GGCN [21] to extract the variable information. Furthermore, our architecture focuses on the components of an instance in CNF, represented as a graph with nodes being clauses, literals, and their negations; and six types of edge being the relations between them.

2.5 Machine Learning for Abstract Argumentation

Abstract Argumentation is an approach to computational modelling of argument that is deliberately extremely simple: arguments are represented as nodes of a graph, and attacking relationships between arguments are represented as directed edges [8]. No other kind of relationship is represented in the graph, which is called an Argumentation Framework (AF). A key problem in abstract argumentation is determining the set of arguments from an AF that can be accepted together (under a variety of different semantics). SAT solvers are frequently used as components of solvers for problems in abstract argumentation. Logic problems in abstract argumentation have been addressed using various graph-based machine learning methods, some of which have inspired our work.

An example of a end-to-end ML heuristic solver for abstract argumentations is EGNN [11]. Proposed as a reinforcement learning solution to dynamic argumentation based on message passing strategy, with a notable focus on efficiency against more traditional solvers.

An important work for this paper is the approach of Lars Malmqvist [24]. He created a model called AFGCN that depicts an end-to-end solver for resolution of AF, a machine learning model based on the application of several layers of GCN, based on the work of Kipf and Welling [17]. A notable factor is how the convolutions of the GCN layers define a first-order approximation of a localized spectral filter on the graphs, capturing the behavior of the variables. GCN layers are remarkable by displaying a good level of performance in homogeneous graphs. However, when they are used in multiple layers, they tend to converge to fixed points. This problem is minimized in the AFGCN model by using residual connections to feed the GCN layers. Based on AFGCN, there are works like AFGAT [9] that offers an analysis of how to improve the model. For instance, by avoiding using dropout layers in the model. AFGAT was focused on increasing the speed of AFGCN on large datasets, and using attention networks for increasing accuracy.

However, for this research AFGCN was adapted to generalize predictions of assignments for SAT. It is important to mention some key differences. AFGCN was designed to work on a homogeneous graph, that is a graph with only one kind of node while the VAPS-GCN model was designed to work on a heterogeneous graph, i.e. a graph with more than one type of node. Therefore, it was necessary to adapt the layers of the model to work on multiple node types. Additionally, we have different kinds of edges to represent the different relationships between types of nodes. Also, it was necessary to have a clear separation of distinct node types in the input feature vectors. Another key difference is the utilization of an embedding layer added to improve the performance when dealing with random problems in our first training round.

3 Predicting Assignments with a Machine Learning Model

The main part of the machine learning model approach is AFGCN [24], a model adapted to predict assignments. However, the process of selecting the assignments of the literals includes several parts. The machine learning model, and process associated with the input and selection of information to the model can be summarized in six parts:

- CSP instances converted to SAT in CNF. They are introduced into the framework created around the VAPS-GCN model to prepare the data for the model.
- SAT instances are converted into graph representations following a heterogeneous graph scheme called LCG*. The clauses and literals of the CNF are used to create three types of nodes. That is, clauses, positive literals (without the negation sign) and negative literals (literals with the negation sign).
- Several metrics from the area of graph network analysis, for instance, eigenvector and closeness centrality, are computed to categorize the nodes and the relationships that they exhibit in the connections between nodes in the graph scheme generated from a respective SAT problem. Together with other handmade values representing the type of node, they are added and grouped in a list by node. Then they are submitted to an embedding layer to capture the information patterns as input features.
- The graph representation and node feature vectors are input into the VAPS-GCN model. VAPS-GCN is a heterogeneous adaptation of the AFGCN model. VAPS-GCN generates a continuous 0-1 value for each node in the LCG* graph, which is a prediction of the probability that the node is *True* in a solution (or optimal solution for MaxSAT). The predicted values for the literal nodes are collected for the next step.
- For each SAT variable x, a confidence score is computed from the predicted values of the two literals x and $\neg x$. The SAT variables are then sorted by confidence (most confident first).
- A threshold is used to separate the variables in the set of confident predictions. A copy of the original SAT or MaxSAT instance is made and unary clauses are added to the copy to assign the respective variables.

Each part of this brief summary is described in detail in the following subsections. Also, the overall design is summarized in Figure 1.

3.1 Heterogeneous Graph Scheme

CSP instances can be represented as Boolean formulas in the CNF representation. Then they can be stored in DIMACS files to be resolved by conventional SAT solvers. The first step to apply our model is to convert the clauses and literals of the CNF format into a graph. Several graph representations have been proposed [22]. It is important to note that the graph representation can lack some of the key relationships that exist between literals and clauses present in the CNF. Furthermore, this loss of information can negatively affect the performance of GNN models.

As an approach to minimizing the problem described in the previous paragraph, LCG* was chosen as the graph scheme. LCG* (described in the G4SATBench [22] paper) is an extension of the original Literal-Clause Graph (LCG), and it can represent the relationship between positive and negative literals as an added edge connecting a positive literal with its negative counterpart. The left graph in Figure 2 shows a heterogeneous graph in the LCG* scheme, the dotted lines represent the relationship that exists between positive and negative literals.

Our dataset of heterogeneous graphs is assembled by transforming our SAT instances from CNF to LCG* graphs. The right graph in Figure 2 presents an assembled heterogeneous bidirectional directed graph composed of three kinds of nodes; clauses, literals, and the negation of the respective literal, and six different edges. In addition, this bidirectional strategy will let the model process the data following a similar form to other ML SAT

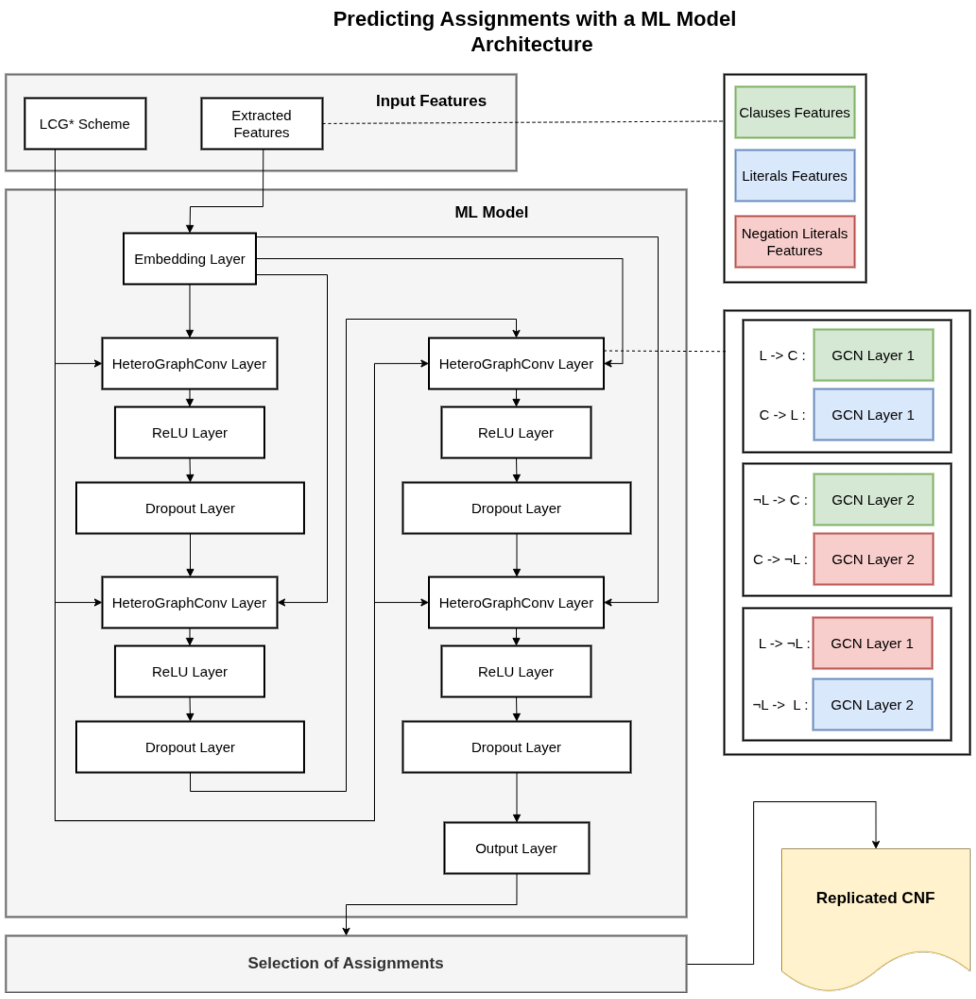


Figure 1 Machine learning design model and features. L represents the positive literals, C the clauses, and ¬L the negative literals.

solvers of the MPNN family. Furthermore, this strategy allows an interchange of information between clauses and literals (negations included), and literals and their respective negations.

For graph construction, and for other parts of the research such as the architecture of the ML model and pre-processing of information, the Deep Graph Library (DGL) [37] was used. DGL is a library dedicated to extending and collecting in a framework several tools and modules to help in the implementation and design of graph neural network models.

3.2 Feature Extraction and Graph Network Analysis

The LCG* graph described above is used directly by the VAPS-GCN model, but we also compute a collection of features for each node of the graph. The features represent the type of the node and its location relative to other nodes. These extracted features are composed of the results of graph network analysis (GNA) metrics applied to each node of a graph. Features are obtained using the AFGCN [24] model as a reference, and with the application of metrics of the NetworkX library [15]. We use metrics of centrality, such as the eigenvector, closeness, and degree. Furthermore, NetworkX does not support heterogeneous graphs so we distinguish the different types of edges (clause to positive literal, clause to negative literal, etc) by giving them different weights. The feature vector also includes a sequence of 0 or 1 values representing the type of node (clause, positive literal, or negative literal) with a one-hot encoding.

In conclusion, the final structure of extracted features obtained to be added as part of our dataset is a collection of lists, where each list is filled with the features calculated for a respective node in the LCG* graph. The main purpose of these extracted features is to include the relevant information patterns present in the graph. In addition, the extracted features of each problem instance are divided into three groups corresponding to the node types of the graph. Altogether, the dataset is formed by the assembled heterogeneous bidirectional directed graphs and their respective extracted features.

3.3 Model Architecture

Our ML model, Variable Assignment Prediction for Satisfiability using GCN (VAPS-GCN), is based on a heterogeneous graph adaptation of the architecture of the AFGCN model. The model architecture starts with an embedding layer to encode the extracted features in an extended form that allows the model to learn more effectively. The processing of the information continues sequentially through four blocks of layers, each composed of a heterogeneous convolution layer, an activation function layer (ReLU layer), and a dropout layer. Finally we have a linear function layer, followed by a sigmoid activation function to generate the predicted probabilities.

An important aspect to explain are the graph convolution layers and their propagation rule used in this model. Originally, the GCN model of Kipf and Welling [17] was designed for homogeneous graphs. It outputs vectors representing the relevance and behaviour that exist between the nodes and their neighbourhood. The propagation rule for a GCN convolution layer is as follows:

$$f(H^{(l)}, A) = \sigma(\widehat{D}^{-1/2}\widehat{A}\widehat{D}^{-1/2}H^{(l)}W^{(l)})$$

Where A is the adjacency matrix, and $\widehat{A}$ is the result of adding self-loops represented as an identity matrix to the adjacency matrix. $H^{(l)}$ are input features of layer l, and $W^{(l)}$ is the weight matrix of layer l. The $\widehat{D}^{-1/2}$ are a symmetric normalization of the rows and columns of the adjacency matrix. σ is an activation function applied to the layer [24, 17].

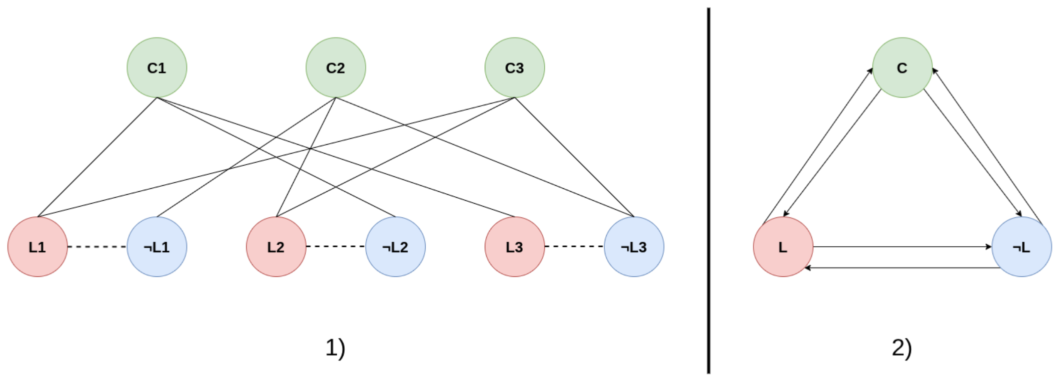


Figure 2 To the left the LCG* scheme using as example a SAT instance; C1: (L1, -L2, L3), C2: (-L1, L2, -L3), C3: (L1, L2, -L3). To the right the bidirectional heterogeneous graph assembled with the DGL library.

GCN layers are notable for capturing relevant information from nodes when processing homogeneous graphs. However, the VAPS-GCN model architecture is oriented to process information from heterogeneous graphs. There are six different relationships (edge types) within the LCG* graph: positive literal to clause and vice versa; negative literal to clause and vice versa; positive literal to negative literal and vice versa. Each of the four heterogeneous convolution layers (named HeteroGraphConv Layer in Figure 1) contains two submodules (named in Figure 1 as GCN Layer 1 and GCN Layer 2) each with its own set of parameters to train.

Each submodule sm implements the following formula, which sums over each directed edge in the set $\mathcal{R}$ of edges r from r_{src} to r_{dst}, where r_{dst} is the same for all edges in $\mathcal{R}$ and all edges in $\mathcal{R}$ are of a type associated with sm:

$$f_{sm,r_{dst}}(H^{(l)}) = b_{sm}^{(l)} + \sum_{r=(r_{src},r_{dst})\in\mathcal{R}} \frac{1}{c_{src,dst}} h^{l}_{r_{src}} W_{sm}^{(l)}$$

Where $W_{sm}^{(l)}$ is the vector of weights for submodule sm in layer l, $b_{sm}^{(l)}$ is the vector of biases for submodule sm in layer l, $h^{l}_{r_{src}}$ is the vector of the source node, and the type of the edge r is associated with the submodule sm. The scaling constant $c_{src,dst} = \sqrt{\text{degree}(src)}\sqrt{\text{degree}(dst)}$.

We do not have six submodules for the six different relationships (edge types). Instead we use two submodules and these are associated to the edge types as shown in Figure 1 (right). Each submodule is used for three distinct edge types. The number of submodules, and association of submodules to edge types, is the result of many preliminary experiments.

Where a node r_{dst} has multiple incoming edges that are handled by more than one submodule, the vectors $f_{sm,r_{dst}}(H^{l})$ need to be aggregated into a single vector. We chose to use an element-wise sum, implemented in the DGL module named HeteroConvLayer [6].

VAPS-GCN model architecture is composed of four heterogeneous convolution layers. Each heterogeneous convolution layer contains only two submodules (as described above), associated with the six relationships of our heterogeneous graphs as shown in the right diagram in Figure 2. The design allows for an interchange of information similar to SAT solvers based on message passing (MPNN) such as NeuroSAT.

GCN layers have a well-known problem of stopping learning the patterns of information present in the data, a problem caused by applying multiple GCN layers in the architecture of a model. In our model, this problem is addressed by using residuals added in each heterogeneous convolution layer except the first one. These residuals are the original extracted features processed after the embedding layer. The residuals are added to the output of the previous layer by applying a matrix addition.

After the four blocks of heterogeneous convolution layers, we apply a linear function. Then it passes through a process to fix dimensions of the data, after which it is possible to calculate the loss function and the prediction of assignments. A sigmoid function is applied to generate the predicted probability of Boolean *True* or *False* for the respective nodes of clauses, positive literals and negative literals.

3.4 Selection of Assignments

The selection of assignments and the computation of the loss function for training the model use the processed data and the relationship that exists between literals and their negations in CNF. These results correspond to the graph and the extracted features processed by the model and are divided by node type. However, only the results of the nodes corresponding to the literals and their negation are used in the loss function computation. This strategy allows the model to focus on the direct but opposite relationship that exists between these nodes. Furthermore, our datasets are composed of satisfiable instances, which means that the nodes corresponding to their clauses are always *True*.

The relationship between positive literals and their negation is used when selecting the most confident assignments. It lets us choose assignments that exhibit a high dual inverse probability. For example, predicting with a probability of 90% that a specific literal is Boolean *True* in the CNF of the respective SAT instance, while predicting that its negation is Boolean *True* with a probability of 10%, that is, a probability of 90% as Boolean *False*. This relationship is used to calculate a confidence score in the prediction of assignment for the respective literal. The confidence score of a variable x is calculated as:

$$C_x = |x_l - x_n|$$

Where x_l is the predicted probability of the positive literal, and x_n is the predicted probability of the negative literal. We also exclude any variable where either $x_l \in [0.3, 0.7]$ or $x_n \in [0.3, 0.7]$.

The confidence score is used to sort the remaining variables from most to least confident. When using VAPS-GCN to do learning-to-prune (as we do in the experiments below), we need to decide a set of variables to assign. We use a threshold which is a manually assigned hyperparameter and is based on the problem class (we report the threshold for the two problem classes below). Each variable with a confidence score above the threshold is assigned to its predicted value (simply by adding a unary hard clause to the SAT or MaxSAT instance).

In summary, the selection of assignments corresponds to the learning-to-prune methodology. We add the best unary assignments in the CNF, which guide a conventional MaxSAT or SAT solver toward a solution with the goal of reducing CPU time with minimal impact on the quality of the optimal solution.

3.5 Multi-Label Training Approach

VAPS-GCN is a supervised learning model; this means that, for training, it needs to compare the prediction computed by learning the information patterns of a problem instance against the respective solution (label). However, SAT instances can have more than one solution. The initial training of the model included multiple solutions by instance as an approach to improve the performance of the learning process.

Multiple solutions are stored per instance in the dataset. This approach selects the most similar solution based on a comparison against the prediction using the cosine similarity metric. The closest solution will be used when computing the loss function for training purposes.

In summary, the purpose of this multi-label training method is to allow the model to learn the structure of solutions while avoiding over-fitting onto a single solution for each instance in the training set. However, this approach is not practical in every case. We use it for the random SAT and RCPSP MaxSAT classes (described below). However it was impractical to generate multiple solutions for training instances of the Combinatorial Auctions problem, and in that case we used a single optimal solution.

4 Experimental Evaluation

In this section, the problems to be used to apply the model are described, together with the results of using the model already trained to prune them. Furthermore, the chosen metric to be used to measure the model overall performance is the Matthews Correlation Coefficient (MCC). A balanced metric for binary data that works well when managing unbalanced datasets. The setting for all the cases of training in this research is Adam as optimizer, and binary cross-entropy as loss function for the ML model.

The model was trained using a fine-tuning approach for each problem class. An approach in which the model initially learns patterns of the data by using randomly generated, easily available data (random SAT instances). Subsequently, training is continued with more useful but scarcer datasets. In this paper, we focus on two optimization (MaxSAT) problem classes, and fine-tune the model for each one.

In addition, balancing was performed over the correct assignments (labels) of the problem instances in the datasets. The loss function was balanced with the Boolean *True* and *False* assignments of the real solutions present in the list of solutions for an instance. It was done with the intention of avoiding fitting problems caused by having a binary dataset with a higher number of ones than zeros, or the other way around.

4.1 Random Problems

Random problems represent the general or initial data used to train the VAPS-GCN model. These instances were generated by the SR generator present in the G4SATBench framework [22].

The generated random problems do not have a defined internal structure compared to real-world problems. However, they can be used to guide the model in learning the dynamics of a SAT instance in CNF. ML models learn the patterns present in the data of the problem instances, but the absence of defined patterns in the structure of random problems is a well-known challenge for ML models to solve and to assert model performance when solving SAT instances. However, this challenge is easier to solve by conventional solvers.

The dataset was generated in this initial evaluation and for pre-training the model to be used in the next combinatorial problems in this section. The dataset is divided into two sets, training, and validation. They are composed of 50000 and 10000 satisfiable problem instances, respectively. Each instance has between 40 and 100 variables. The model was trained over 250 epochs, and with a batch of three hundred instances. In addition, the multi-label approach was applied to this dataset because they are SAT instances that potentially have multiple solutions. At the end of each epoch, the current trained model is evaluated using the validation set, and retained if it is the best trained model seen so far. The final trained model produced by the training process is the best one found during the 250 epochs.

The evaluation of the final trained model produced a value of 46.10% for the mean MCC. The accuracy was 73.05%.

4.2 Combinatorial Optimization Problems

4.2.1 Resource Constrained Project Scheduling Problem

The resource-constrained project scheduling problem (RCPSP) [5] is a combinatorial optimization problem where the objective is to find the shortest possible schedule containing a set of non-interruptible tasks. Each task has a given constant duration (defined by the instance). There are two types of constraints: precedence constraints between tasks (i.e. task A must be completed before task B can begin); and renewable resource constraints that place an upper bound on the amount of a resource that may be consumed at any given time step. The set of precedences is a problem class parameter. For each resource, the upper bound is a problem class parameter, and so is the resource usage of each task. For each resource, for each time step t, the sum of resource usage of tasks that are active at t must be within the upper bound. We use a MaxSAT encoding of RCPSP.

We use a set of RCPSP instances derived from the j30.mm set in the PSPLIB [18] library. To encode to MaxSAT, an upper bound on the makespan is needed (the *horizon*). We used the set of instances from Ansotegui et al [3] with their horizons which were computed with a greedy procedure. However, j30.mm are multi-mode (MRCPSP) instances (tasks may have multiple modes of execution with different durations and resource usage). We converted the instances to RCPSP by fixing the mode to 1 in all cases. Any unsatisfiable instances were filtered out. The instances were encoded into MaxSAT using Savile Row [26] 1.10.1 with the MDD encoding used for the resource constraints [7] with automatic detection of at-most-one groups of variables enabled [3]. Otherwise, encoding options are the defaults as described in the Savile Row manual [25].

Finally, WMaxCDCL2024 MaxSAT solver was applied with twenty different random seeds to add some level of randomness to the labels of the training and validation sets. It is important to mention that only the hard clauses from the instances were used in the construction of the LCG* graphs. Incorporating soft clauses is part of our future work. Following some preliminary experiments, we set the confidence threshold to 0.97. The training set consists of 400 instances, while the validation set and test set are the same set of 148 instances. While not ideal, re-using instances in the test and validation sets allows us to obtain some indicative results despite having a small dataset. In future work we intend to translate a larger set of RCPSP instances.

The VAPS-GCN model showed good performance in predicting the assignments for the MaxSAT instances, generating a visible decrease in solving time. The results of the evaluation are presented in Figure 3. The instances were separated according to the level of hardness. The hardness of an instance for this research is defined as the time that a conventional MaxSAT solver needs to find a solution to the instance without the application of the VAPS-GCN model.

There is a clear indication of improving speed in solving MaxSAT problems which in most cases generated a decrease around 50% in solving time (CPU time), including some of the hardest instances when applying our ML model. Furthermore, in terms of pruning, regardless of the variance in the number of variables in an instance, we found that the pruning of the hardest instances oscillates between 12% and 25%. We found that 10 instances no longer have a solution after applying VAPS-GCN pruning. Finally, for the instances where both original and pruned models have a solution, we obtained the optimality gap (i.e. the objective difference between the original and pruned instance when both are solved to optimality). The mean optimality gap is 0.30%, while the median gap is 0.00%.

However, although there is a decrease in CPU time to solve an instance when using our VAPS-GCN model, the solutions tend to be slightly sub-optimal or the instance becomes UNSAT after pruning for the hardest instances. Only the easiest instances (less than five hundred milliseconds original CPU time) tend to keep the original optimal value after pruning.

4.2.2 Combinatorial Auction Problem

The Winner Determination problem [19, 1] (WDP) is also called the Combinatorial Auction problem in the literature. In the WDP (in its simplest form), a set of items are offered for sale. Bids are received with a given value to purchase a subset of the items. The problem is to determine the set of bids that are accepted, maximising the total value of the accepted bids while avoiding pairs of bids that are in conflict. Two bids are in conflict if they share an item.

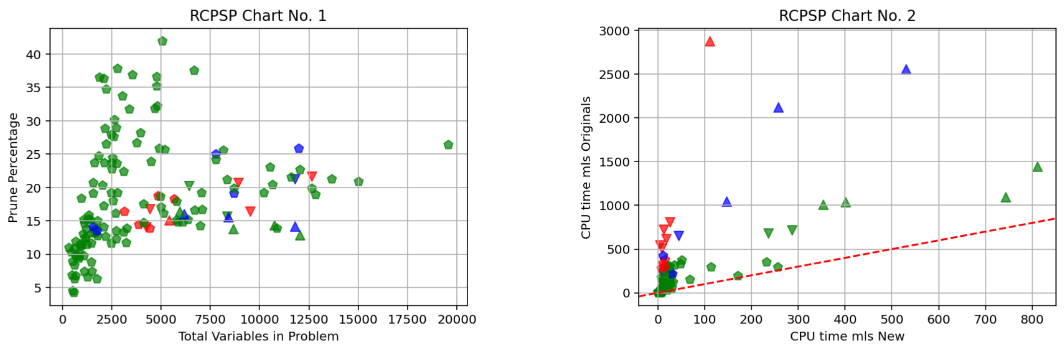


Figure 3 To the left, a chart showing the level of pruning. To the right, a chart showing the speed increase of solving the pruned instances in comparison with the original instances. The shape of the marker indicates the difficulty, that is, time used by the MaxSAT solver to find the solution for the instance without pruning. A pentagon indicates instances that took less than 500 ms, a triangle for more than 1000 ms, and an upside-down triangle for those between 500 and 1000 ms. The color indicates the state of the solution to the pruned model: blue if it is sub-optimal, red if no solution was found, and green if it is optimal.

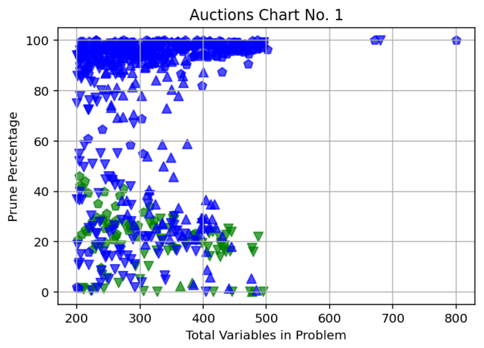
WDP is represented in MaxSAT with one Boolean variable per bid indicating whether the bid is accepted. Binary hard clauses rule out pairs of bids that are in conflict, while the optimization function is easily encoded using unary soft clauses, one for each bid and weighted by the value of the bid.

Generating the dataset to train (fine-tune) the VAPS-GCN model for the WDP required similar steps to the RCPSP case. The problem instances for the training, validation and testing sets of the model were generated using the Combinatorial Auction Test Suite [20], then they were solved, creating the labels using an earlier version of the MaxSAT solver WMaxCDCL. Only instances which were solved to optimality within one hour were added to the dataset. As in RCPSP, Savile Row was used to generate a MaxSAT encoding for each instance. Finally, similarly to RCPSP, only the hard clauses of the instances were used in the LCG* graph fed into the VAPS-GCN model.

The combinatorial auctions dataset to be used in the fine-tuning process can be divided into different distributions or collections. Distributions such as *arbitrary*, *matching*, *regions*, and *scheduling*. Furthermore, legacy distributions such as *L4*, *L5*, and *L6*. Another distribution example is *paths*. However, it was added from a previous example [1] of LTP. All the previously mentioned distributions were generated as one set which was then divided at random into the training, validation and test sets. The training set contains 2000 instances, the validation set has 350, and the test set has 667.

After training VAPS-GCN for the problem class, we applied it as in RCPSP but with a much lower confidence threshold of 0.60 (decided after some preliminary experimentation). For 23 instances in the test set we found that no predicted assignments met the confidence threshold, so these instances were excluded. Of the 644 remaining instances in the test set, we found that 102 of the pruned instances retained the same optimal solution value as the original instances. 542 of the pruned instances had an optimality gap compared to the original instances. None of the pruned instances were unsatisfiable. The mean average optimality gap was 5.10% and the median was 3.70%. Speed improvements are strong on average, as shown by the results plotted in Figure 4.

Chart No.1 in Figure 4 indicates that the pruning of instances does not follow a clear pattern. Furthermore, instances that retain an optimal solution after pruning tend to have



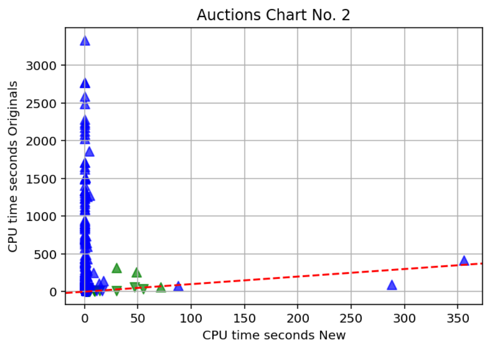


Figure 4 To the left, a chart showing the level of pruning. To the right, a chart showing the speed increase of solving the pruned instances in comparison with the original instances. The shape of the marker indicates the difficulty, that is, time used by the MaxSAT solver to find the solution for the instance without pruning. A pentagon indicates instances that took less than 1 minute, a triangle for more than 10 minutes, and an upside-down triangle for those between 1 minute and 10 minutes. The color indicates the state of the solution to the pruned model: blue if it is sub-optimal, and green if it is optimal.

between 1% and 40% of their literals assigned by pruning.

In chart No.2 of Figure 4 we can observe substantial reductions in CPU time with the MaxSAT solver. Some of the hardest instances (triangle markers) are showing the biggest speed-ups from pruning. However, the instances exhibiting very large speed-ups have also lost all their optimal solutions by pruning.

5 Conclusions

We have presented the VAPS-GCN graph-based supervised machine learning model for SAT (to predict assignments for SAT variables), and demonstrated its application on two realistic structured problems: RCPSP, and Combinatorial Auctions. The model (once trained) is applied in a learning-to-prune fashion, where the most confident (variable-value) predictions from the model are directly assigned to create a pruned problem instance that may be faster to solve (with the risk of losing all solutions, or all optimal solutions). The method offers a very promising approach to reduce CPU time in solving combinatorial problems such as our benchmarks, RCPSP and Combinatorial Auctions. In both cases, applying learning-to-prune with VAPS-GCN sped up the MaxSAT solver. Combinatorial Auctions showed much larger speed-ups but with a bigger optimality gap.

However, both problem classes still need to improve on recognizing assignments of variables that are part of the optimal solution for the hardest instances. Furthermore, in spite of the fact that solutions were found for most or all of the pruned problem instances, in both cases the optimality gap can be improved if we can increase the accuracy of the ML model.

Our future work includes: testing the consistency of the predicted assignments using unit propagation to avoid losing all solutions for a trivial reason; focusing on pruning sets of variables that have a clear meaning in the problem definition (e.g. avoiding auxiliaries introduced during encoding); and representing soft clauses in VAPS-GCN.

References

1 Deepak Ajwani, Peter Nightingale, and Felix Ulrich-Oltean. Generalizing learning-to-prune for constraint programming. In *Proceedings of the 23rd workshop on Constraint Modelling and*

Reformulation (ModRef 2024), 2024. URL: `https://modref.github.io/ModRef2024.html`.

2 Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. Learning to solve circuit-sat: An unsupervised differentiable approach. In *International Conference on Learning Representations*, 2018.

3 Carlos Ansótegui, Miquel Bofill, Jordi Coll, Nguyen Dang, Juan Luis Esteban, Ian Miguel, Peter Nightingale, András Z Salamon, Josep Suy, and Mateu Villaret. Automatic detection of at-most-one and exactly-one relations for improved SAT encodings of pseudo-boolean constraints. In *International Conference on Principles and Practice of Constraint Programming*, pages 20–36. Springer, 2019.

4 1949 Apt, Krzysztof R. *Principles of constraint programming / Krzysztof R. Apt.* Cambridge University Press, Cambridge, 2003.

5 Christian Artigues, Sophie Demassey, and Emmanuel Neron. *Resource-constrained project scheduling: models, algorithms, extensions and applications.* John Wiley & Sons, 2013.

6 DGL Library Authors. Heterogeneous graphconv module, 2018. Accessed: 1st August 2025. URL: `https://www.dgl.ai/dgl_docs/guide/nn-heterograph.html`.

7 Miquel Bofill, Jordi Coll, Peter Nightingale, Josep Suy, Felix Ulrich-Oltean, and Mateu Villaret. SAT encodings for Pseudo-Boolean constraints together with at-most-one constraints. *Artificial Intelligence*, 302:103604, 2022.

8 Günther Charwat, Wolfgang Dvořák, Sarah A Gaggl, Johannes P Wallner, and Stefan Woltran. Methods for solving reasoning problems in abstract argumentation–a survey. *Artificial intelligence*, 220:28–63, 2015.

9 Paul Cibier and Jean-Guy Mailly. Graph convolutional networks and graph attention networks for approximating arguments acceptability. In *Computational Models of Argument*, pages 25–36. IOS Press, 2024.

10 Stephen A Cook. The complexity of theorem-proving procedures (1971). 2021.

11 Dennis Craandijk and Floris Bex. Enforcement heuristics for argumentation with deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 5573–5581, 2022.

12 Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

13 Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

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

15 Aric Hagberg, Pieter J Swart, and Daniel A Schult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Laboratory (LANL), Los Alamos, NM (United States), 2008.

16 John Harrison. *Handbook of practical logic and automated reasoning.* Cambridge University Press, 2009.

17 Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

18 Rainer Kolisch and Arno Sprecher. Psplib-a project scheduling problem library: Or software-orsep operations research software exchange program. *European journal of operational research*, 96(1):205–216, 1997.

19 Daniel Lehmann, Rudolf Müller, and Tuomas Sandholm. The winner determination problem. *Combinatorial auctions*, pages 297–318, 2006.

20 Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham. Towards a universal test suite for combinatorial auction algorithms. In *Proceedings of the 2nd ACM Conference on Electronic Commerce - EC '00*, pages 66–76, Minneapolis, Minnesota, United States, 2000. ACM Press. `doi:10.1145/352871.352879`.

,

21 Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

22 Zhaoyu Li, Jinpei Guo, and Xujie Si. G4satbench: Benchmarking and advancing sat solving with graph neural networks. *arXiv preprint arXiv:2309.16941*, 2023.

23 Chanjuan Liu, Guangyuan Liu, Chuan Luo, Shaowei Cai, Zhendong Lei, Wenjie Zhang, Yi Chu, and Guojing Zhang. Optimizing local search-based partial maxsat solving via initial assignment prediction. *Science China Information Sciences*, 68(2):1–15, 2025.

24 Lars Malmqvist. *Approximate solutions to abstract argumentation problems using graph neural networks*. PhD thesis, University of York, 2022.

25 Peter Nightingale. Savile row manual. *arXiv preprint arXiv:2201.03472*, 2021.

26 Peter Nightingale, Özgür Akgün, Ian P Gent, Christopher Jefferson, Ian Miguel, and Patrick Spracklen. Automatically improving constraint models in savile row. *Artificial Intelligence*, 251:35–61, 2017.

27 Emils Ozolins, Karlis Freivalds, Andis Draguns, Eliza Gaile, Ronalds Zakovskis, and Sergejs Kozlovics. Goal-aware neural sat solver. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2022.

28 Knot Pipatsrisawat and Adnan Darwiche. A lightweight component caching scheme for satisfiability solvers. In *International conference on theory and applications of satisfiability testing*, pages 294–299. Springer, 2007.

29 Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pages 593–607. Springer, 2018.

30 Daniel Selsam and Nikolaj Bjørner. Guiding high-performance sat solvers with unsat-core predictions. In *Theory and Applications of Satisfiability Testing–SAT 2019: 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9–12, 2019, Proceedings 22*, pages 336–353. Springer, 2019.

31 Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L Dill. Learning a SAT solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*, 2018.

32 Zhengyuan Shi, Min Li, Yi Liu, Sadaf Khan, Junhua Huang, Hui-Ling Zhen, Mingxuan Yuan, and Qiang Xu. Satformer: transformer-based unsat core learning. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–4. IEEE, 2023.

33 Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

34 Miru Tang, Baiqing Li, and Hongming Chen. Application of message passing neural networks for molecular property prediction. *Current Opinion in Structural Biology*, 81:102616, 2023. URL: `https://www.sciencedirect.com/science/article/pii/S0959440X23000908`, `doi:10.1016/j.sbi.2023.102616`.

35 Jan Toenshoff, Martin Ritzert, Hinrikus Wolf, and Martin Grohe. Graph neural networks for maximum constraint satisfaction. *Frontiers in artificial intelligence*, 3:580607, 2021.

36 E. Tsang. *Foundations of Constraint Satisfaction*. Computation in cognitive science. Academic Press, 1993. URL: `https://books.google.co.uk/books?id=TnxQAAAAMAAJ`.

37 Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

38 Lingfei Wu, Peng Cui, Jian Pei, Liang Zhao, and Xiaojie Guo. Graph neural networks: foundation, frontiers and applications. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 4840–4841, 2022.