# Predicting Assignments of MaxSAT Variables with Graph Neural Networks

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

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

**Felix Ulrich-Oltean**
Department of Computer Science, University of York, United Kingdom

## Abstract

The partial weighted MaxSAT problem is an important generalization of SAT, useful for expressing and solving a variety of constrained optimization problems. Efficient solving methods have been developed based on SAT and MIP, including core-guided search, the implicit hitting set approach, and branch-and-bound. The problem is naturally expressed as a graph, with vertices representing literals and clauses, while a set of edges connect literals to the clauses that contain them. In this paper we propose a graph neural network architecture that is trained to predict the truth value of each MaxSAT variable in an optimal solution (with a confidence score). The machine learning model is initially trained on small random MaxSAT instances, then fine-tuned for a particular class of MaxSAT instances. A model such as this could be used to warm-start a solver or to guide search in some way. We chose to apply the model as part of a learning-to-prune method, where some literals (the most confident predictions) are chosen to assign before applying a conventional solver. Using learning-to-prune, we demonstrate significant improvements in performance compared to a current branch-and-bound MaxSAT solver.

**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
*Felix Ulrich-Oltean*: Partially supported by UK EPSRC grant EP/W001977/1

**Acknowledgements** We thank Jordi Coll for assistance with the WMaxCDCL solver. The Viking cluster was used during this project, which is a high performance compute facility provided by the University of York. We are grateful for computational support from the University of York, IT Services and the Research IT team.

## 1 Introduction

MaxSAT (in particular, partial weighted MaxSAT) is an important generalization of SAT, useful for expressing and solving a wide variety of constrained optimization problems. Efficient solvers have been developed based on SAT and Mixed Integer Programming (MIP), including core-guided search, the implicit hitting set method, and branch-and-bound. In this paper we focus on predicting the values of the Boolean variables in MaxSAT problems, with the goal of speeding up conventional solvers for these problems. The ability to predict values of MaxSAT variables with good accuracy could potentially be applied in several ways to improve performance of conventional solvers. For example, a solver could be warm-started by providing an initial phase for each variable for use in a phase heuristic [33]. Furthermore, if values can be predicted alongside a confidence score, then the predicted assignments with the highest confidence can be applied to reduce the size and difficulty of a given MaxSAT instance (at risk of making the instance unsatisfiable or ruling out all optimal solutions).

In this area, most machine learning 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 or MaxSAT instance has not been the main focus of research. Therefore a key motivation for this research is to explore whether values can be predicted with a useful degree of accuracy. There are several approaches targeting SAT, for example, end-to-end machine learning models such as NeuroSAT [36] and DG-DAGRNN [4], 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 [37] predict satisfiability and an UNSAT core composed of clauses and their literals. An UNSAT core (if one exists) provides an explanation for the unsatisfiability of an instance. InitPMS [28] is the most closely related work: it predicts an assignment for partial MaxSAT which is then used as an initial assignment for a local search solver as an alternative to a random initial assignment.

MaxSAT can be quite naturally expressed as a graph, with subsets of the vertices to represent clauses and literals, suggesting that graph neural networks will be useful in this context. We propose a graph neural network architecture named VAPSGCN (ultimately based on Kipf and Welling's Graph Convolutional Network [21]), a set of input features, and a two-stage training method. The model is trained to predict the truth value of each MaxSAT variable (i.e. the value it would be assigned in an optimal solution). It also produces a confidence score for each variable. We report accuracy statistics for the model on four sets of benchmark instances (across three problem classes).

We have also evaluated the model experimentally by applying it in a learning-to-prune framework, where the predicted values with a high confidence score are assigned to reduce the size of the MaxSAT instance (i.e. the model acts as a pruning heuristic). The pruned instance and the original instance are both solved with the same conventional partial weighted MaxSAT solver, and the results are compared (in terms of solution quality and time). Applying the model resulted in a substantial improvement overall, mainly because many more instances could be solved (i.e. a solution could be found) with pruning than without.

## 2 Background and Related Work

Machine learning methods based on graphs have attracted attention recently in CP and SAT, as well as other areas of combinatorial solving and optimisation. ML approaches follow three major architectures: Recurrent Neural Networks (RNN) that process a stream and have the capacity for memory [18]; Graph Neural Networks (GNN), a kind of neural network designed to recognise patterns in graphs and generalize from graph structured data [45]; and Reinforcement Learning (RL) where an agent is rewarded or punished for its actions in an environment [38]. In addiction, GNN models (as applied to SAT or MaxSAT) tend to be 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 [17, 39]. In this section we briefly survey the work in the area of machine learning for SAT and MaxSAT solving. We focus particularly on those methods with an approach similar to ours to predict assignments for variables.

### 2.1 Machine Learning Approaches to SAT

Research on SAT with machine learning has mainly focused on predicting satisfiability, or predicting the clauses and their literals that cause an instance to be unsatisfiable (i.e., predicting an UNSAT core) [20]. NeuroSAT [36] 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 a SAT instance. After training on random 3-SAT instances, it predicts whether a problem is satisfiable. NeuroCore [35] (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 an UNSAT core. DG-DAGRNN [4] takes a different approach (to predict solutions to Circuit-SAT in this case): it uses a training strategy similar to reinforcement learning, with an architecture designed to embed the graph structure of Circuit-SAT. QuerySAT [32] builds on NeuroSAT and DG-DAGRNN to predict solutions to CNF SAT. It uses a query mechanism which helps the model to learn the structure of an instance at runtime. The query process is iterated with the intention of converging to a solution. There are crucial differences to the MPNN strategy [36, 20]: the query mechanism replaces the message passing step from variables to clauses [32]. Detecting unsatisfiability is a particular challenge for QuerySAT because it is similar to local search. SATformer [37] adds an attention mechanism to a graph neural network architecture (improving on the accuracy of NeuroSAT) to predict UNSAT cores and satisfiability.

## 2.2 Machine Learning Approaches to MaxSAT

Tönshoff et al. [40] proposed a model to predict optimal solutions to binary constraint optimization problems. They evaluated their model on Max-2-SAT, among other problems, with promising results compared to classical solvers (e.g. CPLEX). The model is a recurrent neural network based on LSTM and trained in an unsupervised fashion.

Liu et al. proposed InitPMS [28], a model to predict variable assignments for partial (unweighted) MaxSAT. InitPMS is intended to replace the random initial assignment conventionally used by local search partial MaxSAT solvers. The authors report substantial improvements in performance when testing on MaxSAT Evaluation (MSE) 2020 and 2021 benchmark instances (having trained on MSE benchmarks prior to 2020). Their machine learning model is quite different to ours. InitPMS applies RGCN [34] to capture structural information from the heterogeneous input graph (where hard and soft clauses are represented by two distinct edge types), and GGCN [26] to update the nodes representing MaxSAT variables. Finally a multilayer perceptron is used to predict the assignment. While InitPMS is the most closely related work it cannot be applied to partial weighted MaxSAT, so we have not compared our model to InitPMS experimentally.

Recently, HyperSAT [11] and SGAT [2] have addressed weighted MaxSAT (without hard clauses) using graph neural networks with attention mechanisms. HyperSAT also uses a hypergraph representation (i.e. each clause becomes a hyperedge) which may be interesting to explore for partial weighted MaxSAT.

## 2.3 Machine Learning for Abstract Argumentation

Abstract Argumentation is an approach to computational modelling of argument that is deliberately minimal: arguments are represented as nodes of a graph, and attack relationships between arguments are represented as directed edges [10]. 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.

Malmqvist et al. [29] proposed AFGCN, a model to predict the acceptability of arguments in an AF, based on a four-layer GCN [21]. 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 behaviour of the variables. GCN layers display a good level of performance in homogeneous graphs. However over 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. AFGCN proved to be more accurate than a similar earlier approach named AGNN [14]. Based on AFGCN, AFGAT [12] offers an analysis of how to improve the model (e.g. by avoiding using dropout layers). AFGAT was focused on increasing the speed of AFGCN on large datasets, and using attention networks for increasing accuracy. Another example of GNNs applied in this area is EGNN [15] for dynamic argumentation (where knowledge of attacks between arguments is incomplete or changing).

We have adapted the AFGCN architecture to MaxSAT. 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 VAPSGCN model works on a heterogeneous graph. 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 addition of an embedding layer to improve performance when dealing with random problems in our first training round.

### 2.4 Learning to Prune

In the "Learning to Prune" approach, an ML model is trained to predict variable assignments that are likely to be part of an optimal solution; by implementing the predictions as a partial assignment, the search space can be reduced. Success has been shown for graph-based problems, such as the Maximum Clique Enumeration problem (MCE). Grassia et al. [19] apply repeated passes to "sparsify" a graph by removing nodes which are unlikely to be in a maximum clique; their predictions are made by a gradient boosting classifier. Carchiolo et al. [9] also tackle the MCE problem, but use graph neural networks to prune the graphs before applying an exact solver. Going beyond just graph-based problems, CP-LTP [3] applied this approach more broadly to combinatorial optimization problems expressed in a general constraint modelling language; CP-LTP was applied to problem classes which had linear objective functions defined over Boolean variables. Although graph embeddings were used as features, the classifier used was once again gradient boosted trees. Similar to this work, the problem is encoded into MaxSAT and solved using a MaxSAT solver.

## 3 VAPSGCN: A GCN-based Model for Partial Weighted MaxSAT

In this section we describe the architecture and training of the proposed model, named Variable Assignment Prediction for SAT/MaxSAT with GCN (VAPSGCN). The model is based on AFGCN [29], extensively adapted to the partial weighted MaxSAT problem. Without loss of generality, the soft clauses must be unary. The overall process is summarised below and detailed in the following subsections. The architecture is also summarised in Figure 1.

- The MaxSAT instance is converted into a heterogeneous graph following the LCG*

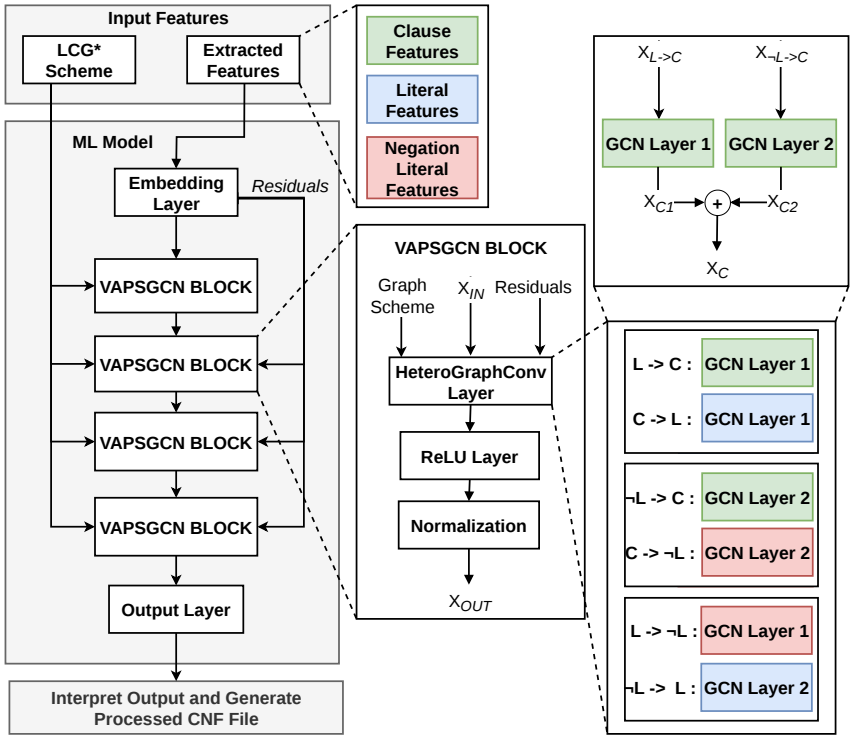


**Figure 1** VAPSGCN architecture. L represents the positive literal nodes, C the clause nodes, and ¬L the negative literal nodes.

scheme [27] with nodes representing clauses, positive literals, and negative literals.

- Metrics from the area of graph network analysis, for instance, eigenvector and Katz centrality, are computed as node features representing the relationships that they exhibit in the graph. These are combined with features encoding the type of node (clause, positive literal, negative literal).
- The graph representation and node feature vectors are input into the VAPSGCN model. VAPSGCN is a heterogeneous adaptation of the AFGCN model. VAPSGCN 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 an optimal solution.
- For each MaxSAT variable $x$, a confidence score is computed from the predicted values of the two literals $x$ and $\neg x$. The MaxSAT variables are then sorted by confidence (most confident first). Literals below a confidence threshold are discarded. The remaining literals are assigned in order (highest confidence score first), except where assigning the literal causes unit propagation to fail.

## 3.1 Heterogeneous Graph Scheme

The first step to apply VAPSGCN is to convert the MaxSAT instance (except soft clauses, which will be encoded as node features) into a graph. Several graph representations have been proposed for SAT and are surveyed by Li et al. [27]. We chose to use the LCG* representation for the hard clauses. Figure 2 shows a heterogeneous graph in the LCG* scheme. The dotted lines are the edges between positive and negative literals, while solid lines represent inclusion

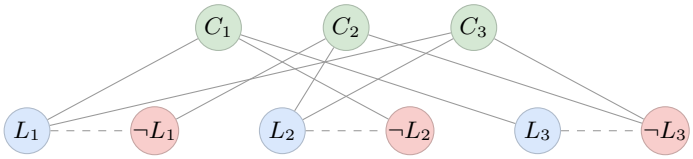


**Figure 2** The LCG* graph of the SAT instance $C_1 : (L_1 \vee \neg L_2 \vee L_3)$, $C_2 : (\neg L_1 \vee L_2 \vee \neg L_3)$, $C_3 : (L_1 \vee L_2 \vee \neg L_3)$

of a literal in a clause. LCG* extends the original Literal-Clause Graph (LCG) by adding a distinct type of edge between each positive literal and its negation. LCG* was chosen because it preserves all the information in the CNF, and allows direct communication between each literal and its negation. Finally, the LCG* graph is turned into a directed graph by replacing each undirected edge $a \leftrightarrow b$ with two directed edges $a \rightarrow b$, $b \rightarrow a$. We then have a graph with three node types (clause, positive literal, and negative literal) and six types of edges (see Figure 1 lower-right panel). VAPSGCN is in the MPNN family of models which can be understood as passing messages along the edges of a directed graph. The Deep Graph Library (DGL) [44] was used to construct the graph and the ML model.

## 3.2 Feature Extraction and Graph Network Analysis

The LCG* graph described above is used directly by the VAPSGCN model, but we also compute a collection of features for each node. The features represent the type of the node and its location relative to other nodes. Some of the extracted features are graph network analysis metrics applied to each node. We use a similar set of features to those successfully used in AFGCN [29]. We use the eigenvector and Katz measures of centrality, the node degree, indegree, outdegree, page rank, and greedy graph colouring (implemented in the Rustworkx library [41]). The feature vector for the node 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. The type is also represented as a value from $\{1 \ldots 3\}$. Finally, we include the weight for the respective literal in a (unary) soft clause converted to a percentage of the largest weight, or 0 if the literal is in no soft clause. Each node has a vector of 12 features.

## 3.3 Model Architecture

VAPSGCN (illustrated in Figure 1) is loosely based on AFGCN [29], adapted for heterogeneous graphs. The model architecture starts with an embedding layer to encode the feature vector (containing 12 features) into an extended vector containing 128 values, allowing the model to learn more effectively. The embedding layer is a simple linear mapping that is applied to each node independently (but with the same weights in each case). Then there are four blocks of layers, each composed of a heterogeneous graph convolution layer (explained in detail below), an activation function layer (ReLU layer), and an instance D1 normalization layer [42]. Finally we have a linear function layer, followed by a sigmoid activation function to generate the predicted probabilities.

The GCN model of Kipf and Welling [21] was originally designed for homogeneous graphs. Intuitively, a GCN convolution layer is a function (with trainable parameters) that updates the feature vector of each node $n$ in the graph, taking as input the existing feature vector for $n$ and those of its neighbours (where there is a directed edge from the neighbour to $n$). By applying the same function to each node simultaneously, a GCN layer can scale to any size of graph without changing the number of parameters to train. 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. $\sigma$ is an activation function applied to the layer [29, 21].

There are six 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 $sm$:

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

Where $W^{(l)}_{sm}$ is the vector of weights for submodule $sm$ in layer $l$, $b^{(l)}_{sm}$ 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 (lower 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 [7], as illustrated in Figure 1 (upper right, for a node of type $C$).

The VAPSGCN model is composed of four heterogeneous convolution layers. Each one contains two submodules (as described above), associated with the six relationships of our heterogeneous graphs. The design allows for an interchange of information similar to SAT models based on message passing (MPNN) such as NeuroSAT.

GCNs with several layers have a well-known problem that they tend to average out the feature differences between nodes (known as *over-smoothing*). In our model this problem is addressed by using residuals added in each heterogeneous convolution layer except the first one. The residuals are the original features produced by the embedding layer. The residuals are added to the output of the previous layer by applying matrix addition. After the four blocks of heterogeneous convolution layers, we have an output layer consisting of two parts. The first is a linear function from a feature vector to a single value. The linear function is applied to each node independently, with the same weights in each case. Finally, a sigmoid (logistic) function is applied to generate the predicted probability that the clause, positive literal, or negative literal is *true* in an optimal solution.

## 3.4 Interpreting Output and Generating Learning-to-Prune Assignments

The network produces a predicted probability for each node in the LCG* graph. We discard the values for the clause nodes. During training, the loss function (binary cross-entropy) is

applied to the values produced for positive literal nodes and negative literal nodes. When using the trained model we are most interested in variables where the model makes a prediction with high confidence. The relationship between positive and negative literals is used when selecting the most confident assignments. For example, if the output was 0.9 for the literal $x$, and 0.1 for the literal $\neg x$ (i.e. the model predicts that $x$ is *true* and that $\neg x$ is *false*) we could conclude that $x$ is *true* with a reasonably high degree of confidence. The confidence score of $x$ is as follows (where $p_l$ is the predicted probability of literal $l$):

$$C_x = |p_x - p_{\neg x}|$$

When using VAPSGCN to do learning-to-prune (as 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 can be tuned to the problem class (although we use the same value throughout). For each variable, we take the literal of that variable with the higher predicted probability and add the literal to a list $L$, which is then sorted by confidence score.

We start with an empty set of assignments $A$, and iterate over the literals $l_i \in L$ in order (from highest to lowest confidence score). If $l_i$ has a confidence score above (or equal to) the threshold, then it is added to $A$. Unit propagation is applied to the set of hard clauses from the MaxSAT instance and the set of literals (as unit clauses) $A$. If unit propagation detects inconsistency, then $l_i$ is discarded; otherwise $l_i$ is retained in $A$ and the iteration through $L$ continues. Finally the pruned MaxSAT instance is a copy of the original instance with the literals in $A$ added (as unary hard clauses).

## 4 Training

The VAPSGCN model is trained in two stages. In the first stage, we train the model on a large set of randomly generated (weighted partial) MaxSAT instances, with the intention that it learns the basic dynamics of MaxSAT. In the second stage, it is fine-tuned for a particular problem class, where training data is scarcer or more difficult to generate. In the second stage we expect the model to learn from the structure of the problem class. To give a simple example, in the encoding of the Combinatorial Auctions problem (described in Section 5) we have boolean variables representing auction bids with a given bid value. In this case, the model could learn to rule out those bids with a low value that would preclude accepting a large number of other bids. We focus on three problem classes in this paper, and for one of the problem classes we have two different instance distributions. We fine-tune the model for each one of these four benchmark sets and evaluate its accuracy in Section 5 below.

Recall that the ML model is a binary classifier, it predicts the class *true* or *false* for each MaxSAT variable. The main metric for model performance is Matthews Correlation Coefficient (MCC), a metric for binary classification that works well with unbalanced datasets and takes account of both classes. We also report accuracy because of its prevalence in the machine learning literature. Both stages of training are performed with stochastic gradient descent using the Adam algorithm with learning rate 0.001, and binary cross-entropy as the loss function. Balancing is performed over the two classes by adjusting the loss function to give a higher weighting to errors when predicting the minority class.

### 4.1 Random Partial Weighted MaxSAT

We used the *SR* generator [36] (available from G4SATBench [27]) to generate 40,000 satisfiable SAT instances with the number of variables chosen uniformly at random in the range 40

to 100. (SR generates pairs of instances with one satisfiable and the other unsatisfiable. We simply take the satisfiable one in each case.) SR has only one parameter (the number of variables). It is designed such that simple statistical properties of the instance should not indicate whether it is satisfiable because it was originally used for training NeuroSAT. Instances generated by SR have no regular structure.

For each instance, we converted it into partial weighted MaxSAT by first choosing a percentage $p$ at random from 30% to 70%. Then we sample $p$% of the literals at random. For each literal in the sample, a soft unary clause is added containing the literal with a weight sampled uniformly at random in the range $1 \ldots 100$. The 40,000 instances were divided into a training set with 30,000 and validation set with 10,000.

The model was trained for 250 epochs, with a batch size of 300 instances. 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 is the best one found during the 250 epochs. Training took 10.5 hours on an NVIDIA A40.

## 5 Benchmarks and VAPSGCN Accuracy

In this section we describe the benchmark problem classes and instances, and explore the accuracy of VAPSGCN when trained on them. In each case, the problem instances were encoded into partial weighted MaxSAT using Savile Row [31] 1.11.1 with the MDD encoding [8] used for pseudo-Boolean constraints with automatic detection of at-most-one groups of variables enabled [5]. Otherwise, encoding options are the defaults as described in the Savile Row manual [30].

The **Resource-Constrained Project Scheduling Problem** (RCPSP) [6] 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 used the 480 j60.sm instances from PSPLIB [22]. To encode to MaxSAT, an upper bound on the makespan is needed (the *horizon*), and we compute this by constructing a schedule with a simple greedy heuristic and using its makespan as the horizon. We augmented the set by taking an instance and shuffling the columns corresponding to durations and resource usages (with a different random reordering for each column). The MaxSAT encoding of RCPSP has a boolean variable $x_{i,t}$ for each task $i$ and timestep $t$ within the horizon, indicating whether the task is running during the timestep. It also has an order encoding [16, 30] of the integer start time variable for each task, channelled to the $x_{i,t}$ variables. The encodings of precedence constraints use the start time variables, while the encodings of (pseudo-boolean) resource constraints use the $x_{i,t}$ variables. The optimization function is encoded with unary soft clauses (each with weight 1), each containing one variable from the order encoding of the start time of the final task. The MDD-based encoding of resource constraints introduces auxiliary variables [8]. Learning-to-prune may assign any of the MaxSAT variables, including those introduced when encoding the MDDs.

Each instance was solved with WMaxCDCL [13] with a time limit of 2 hours; the training

| Problem Class | MCC | Accuracy |
|---|---|---|
| Random MaxSAT (prior to fine-tuning) | 0.4099 | 0.7050 |
| RCPSP | 0.6155 | 0.8345 |
| CA Arbitrary | 0.2322 | 0.9041 |
| CA Regions | 0.2732 | 0.9075 |
| NSP | 0.5281 | 0.7598 |

**Table 1** Accuracy and Matthews Correlation Coefficient (MCC) for the VAPSGCN model on the validation set when fine-tuned for each problem class.

set of 1000 instances and validation set of 215 instances were sampled at random from the set of instances that were solved to optimality. The test set of 100 instances were sampled from the set where WMaxCDCL timed out. Fine tuning was performed for 150 epochs with the same training procedure (and hyperparameters) as for the random instances. We used an NVIDIA H100 and the fine-tuning process took 57.3 hours (substantially longer than other problem classes because the graphs are larger, with the largest having over 100,000 nodes).

The **Nurse Scheduling Problem** [43] (NSP) consists of creating a duty schedule that meets staffing requirements and optimises for the preferences of staff in a hospital environment. We use a simple form of the NSP. For each day in the schedule, each staff member is allocated one of four shift types. One of the shift types represents a day off, and staff have two days off for every 5 days at work. For each day and shift, the number of staff on duty must be at least a given number (which is provided in the instance data). Staff express their preferences by giving each day and each shift a score from $\{1 \dots 4\}$. The objective function is to minimise the sum of the preference scores of assigned shifts. NSP is modelled as partial weighted MaxSAT as follows. For each day $d$, shift $s$, and staff member $m$ we have a boolean variable $x_{d,s,m}$ which is *true* iff shift $s$ is allocated to staff member $m$ on day $d$. The hard constraints are all cardinality constraints on subsets of the $x_{d,s,m}$ variables. The optimization function is encoded with unary soft clauses ($\neg x_{d,s,m}$) weighted by the relevant preference score.

Benchmark instances were obtained from earlier work in learning-to-prune for CP [3] (originally from NSPLIB [43]). All 3000 instances were solved with WMaxCDCL [13] with a 2 hour time limit. From the set of instances that were solved to optimality, we sampled 1100 at random and divided them into 900 for training and 200 for validation. We use a batch size of 10. The test set consists of 100 instances chosen at random from the instances where WMaxCDCL timed out. The model was fine-tuned for 150 epochs with the same training procedure and hyperparameters as for the random instances. We used an NVIDIA H100 and the fine-tuning process took 14.7 hours.

The **Combinatorial Auction problem** [24, 3] (CA) is also called the Winner Determination problem. 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. CA 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 encoded using unary soft clauses, one for each bid and weighted by the value of the bid.

We used the Combinatorial Auctions Test Suite [25] benchmark sets named *arbitrary* and *regions* because they contain instances with a range of difficulties for conventional solvers. We used WMaxCDCL [13] to solve each instance with a time limit of two hours. For each benchmark set separately, from the instances that were solved to optimality, we sampled 2100

| | RCPSP | CA Arbitrary | CA Regions | NSP |
|---|---|---|---|---|
| V. Feasible Solution Found | **70** | **85** | **89** | 100 |
| W. Feasible Solution Found | 42 | 78 | 87 | 100 |
| V. No Soln, W. No Soln | 23 | 7 | 1 | 0 |
| V. Soln, W. No Soln | 35 | 15 | 12 | 0 |
| V. No Soln, W. Soln | 7 | 8 | 10 | 0 |
| Soln Quality Equal | 31 | 16 | 9 | 69 |
| V. Better Soln | 0 | 21 | 33 | 17 |
| W. Better Soln | 4 | 33 | 35 | 14 |
| V. Timeout | 18 | 89 | 89 | 94 |
| W. Timeout | 69 | 98 | 100 | 99 |

**Table 2** Two hour time limit experiment. V. is VAPSGCN-LTP, W. is WMaxCDCL, Soln is Solution. Total number solved (out of 100) and number of instances in each of 6 classes, for each problem. Instance classes correspond to point types in Fig. 3.

for the training set and 450 for the validation set at random. The test sets of 100 instances each were sampled at random from the instances where WMaxCDCL timed out. Fine tuning was performed for 250 epochs with the same training procedure (and hyperparameters) as for the random instances. In this case we used an NVIDIA A40 and the fine-tuning process took 4.2 hours for *arbitrary* and 3.3 hours for *regions*.

### Fine-Tuning Accuracy Results

The fine-tuning procedure is described in Section 4, and we applied it to each of the four benchmark sets. The accuracy and MCC of each fine-tuned model is reported in Table 1. The accuracy and MCC of the model prior to fine-tuning is also included for reference. Since random MaxSAT instances have no patterns to recognise in their graph structure, we expected a GCN-based model to perform better for the structured instances. It is the case that accuracy is higher following fine-tuning on all problem classes. MCC is also higher for RCPSP and NSP than for random MaxSAT. However, we found that MCC values for the two types of CA benchmarks are low. The model does have some predictive power (MCC is above zero) but high accuracy and low MCC indicates that the model is not highly accurate for the less frequent class (*true* in this case). However, these statistics do not take account of the confidence score. If the model is used in a learning-to-prune solver, inaccurate predictions with low confidence are harmless. Our evaluation with learning-to-prune (in the next section) shows that the CA models are sufficiently accurate overall to be useful in that context.

We were not able to compare to InitPMS [28] (arguably the most closely related prior work) because their model is for partial MaxSAT whereas three of our benchmark sets (CA and NSP) are partial weighted MaxSAT. Also, they reported no model accuracy statistics.

## 6 Learning-to-Prune Experiments

In this section we evaluate whether VAPSGCN (fine-tuned for each problem class as described above) performs well practically as part of a learning-to-prune solver. Learning-to-prune uses a ML model to reduce the size of a problem instance (by assigning some variables in this case), then solves the pruned instance with a conventional solver. The goal is to solve the instance faster, but guaranteed optimality is lost. Section 3.4 describes how the output of the VAPSGCN model is interpreted to decide on a set of literals to assign.

WMaxCDCL 2024 [13] was chosen as the backend solver because of its strong overall performance compared to other solvers (a version of WMaxCDCL won the MaxSAT Evalu-

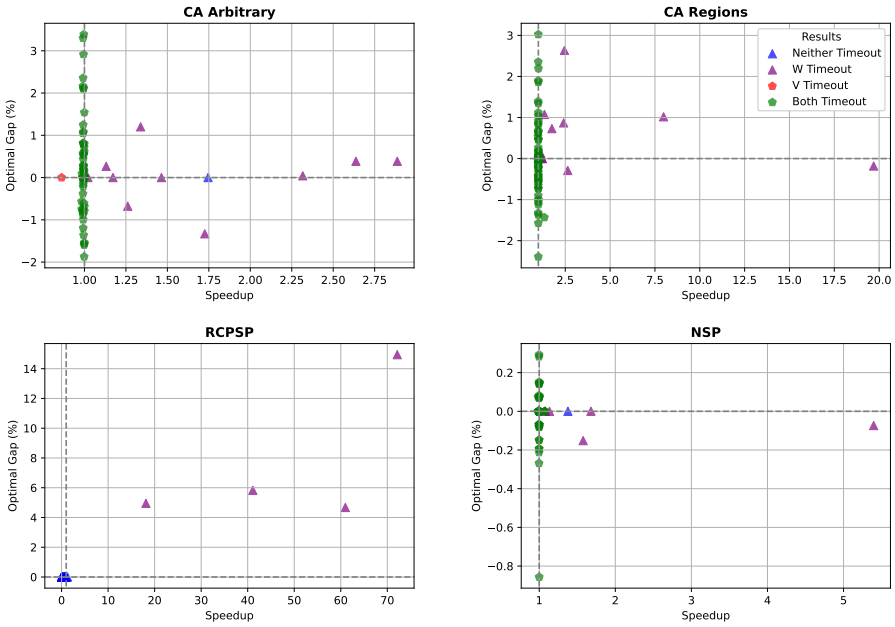


**Figure 3** Experiment with two hour time limit. Speedup from LTP and loss of optimality for each problem class. Only instances where *both* solvers found a solution are included here. The $x$-axis is the speedup quotient of VAPSGCN-LTP (V) compared to WMaxCDCL (W). The $y$-axis is the percentage difference of the objective value of V compared to W as the baseline. A positive value indicates the best solution found by V is worse than the best solution found by W.

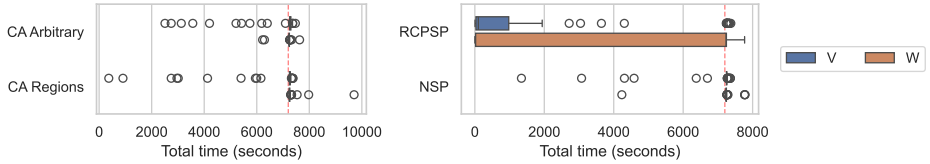


**Figure 4** Distribution of time taken by the two solvers in the two hour time limit experiment. For each problem class, the upper box-and-whisker plot is for VAPSGCN-LTP (V) and lower plot is for WMaxCDCL (W). The red dashed line is the time limit. For a few instances the solver overruns the time limit quite substantially.

ation 2023 exact weighted track [1]), and the fact that it is based on branch-and-bound so it generates intermediate feasible solutions. When WMaxCDCL times out, we record the objective value of the last solution produced (if a feasible solution was found), so we are able to compare results between two solvers even when one or both timed out.

Following some preliminary experiments, we set the confidence threshold to 0.95. We have a test set of 100 instances of each class. We compare the following solver configurations:

**VAPSGCN-LTP** Apply VAPSGCN to the MaxSAT instance, then prune the instance (as in Section 3.4). Solve the pruned instance with WMaxCDCL.

**WMaxCDCL** Solve the original instance with WMaxCDCL alone.

All reported results are a median of 5 runs (with distinct random seeds). Experiments ran on the York Viking cluster with AMD EPYC3 CPU cores. Both solvers use a single CPU process and no GPU. VAPSGCN-LTP timings include the entire process of computing features, applying the VAPSGCN model, pruning the MaxSAT instance, and solving.

| | RCPSP | CA Arbitrary | CA Regions | NSP |
|---|---|---|---|---|
| V. Feasible Solution Found | **68** | **79** | **85** | 100 |
| W. Feasible Solution Found | 28 | 67 | 82 | 100 |
| V. No Soln, W. No Soln | 32 | 9 | 2 | 0 |
| V. Soln, W. No Soln | 40 | 24 | 16 | 0 |
| V. No Soln, W. Soln | 0 | 12 | 13 | 0 |
| Soln Quality Equal | 28 | 10 | 4 | 70 |
| V. Better Soln | 0 | 19 | 27 | 22 |
| W. Better Soln | 0 | 26 | 38 | 8 |
| V. Better \| W. Better | **40** \| 0 | **43** \| 38 | 43 \| **51** | **22** \| 8 |

**Table 3** Results with equal time for both solvers. V. is VAPSGCN-LTP, W. is WMaxCDCL, Soln is Solution. Total number solved (out of 100) and number of instances in each of 6 classes, for each problem. Instance classes correspond to point types in Fig. 5.

## 6.1 Fixed Time Limit Experiment

In our first experiment we simply ran both solvers with a time limit of 2 hours, and recorded the time they took and the objective value of the best solution they produced. Table 2 shows how many instances were solved (i.e. where the solver produced any feasible solution). The set of 100 instances is then broken down into six categories based on whether each of the solvers produced a solution at all, and (if they both produced a solution) which solver produced a better solution.

Table 2 shows that VAPSGCN-LTP is able to find solutions for many more of the instances for three of the problem classes (and all instances for the other problem class). Also there were at most 10 instances of each class where VAPSGCN-LTP did not find a solution and WMaxCDCL did. However, when both solvers produced a solution, WMaxCDCL was able to produce a better solution in the majority of cases for RCPSP and both types of CA. It is only for the NSP that VAPSGCN-LTP produces a better solution for more instances than WMaxCDCL.

Figure 3 shows the speedup obtained by pruning compared to the loss of solution quality (where both solvers produced a solution). Results vary widely for the different problem classes. We found that many instances have a speedup of approximately 1 (which can occur when both solvers reach the time limit). For RCPSP, it seems the pruning is aggressive, causing optimality gaps that may not be acceptable depending on the application (e.g. more than 2-3% may be unacceptable). However the threshold can be adjusted per problem class (and could be decided based on the training data, alongside fine-tuning). Some instances show very large speedups, although these instances also tend to have large optimality gaps.

Figure 4 shows the time distribution for each solver and problem class. It shows that pruning tends to reduce the CPU time and (given that VAPSGCN-LTP found a feasible solution for many more instances) seems to indicate that VAPSGCN-LTP performs comparatively well. Since LTP can be applied with any (complete or incomplete) solver, it seems likely that the approach would contribute positively to a solver portfolio [23].

## 6.2 Equal Time Experiment

In the previous experiment, WMaxCDCL tends to take more CPU time than VAPSGCN, with more runs of WMaxCDCL reaching the time limit (see Table 2). It is perhaps not surprising that WMaxCDCL can find better quality solutions in many cases since it is using the full two hours while VAPSGCN-LTP stops earlier. Here we compare the two solvers in

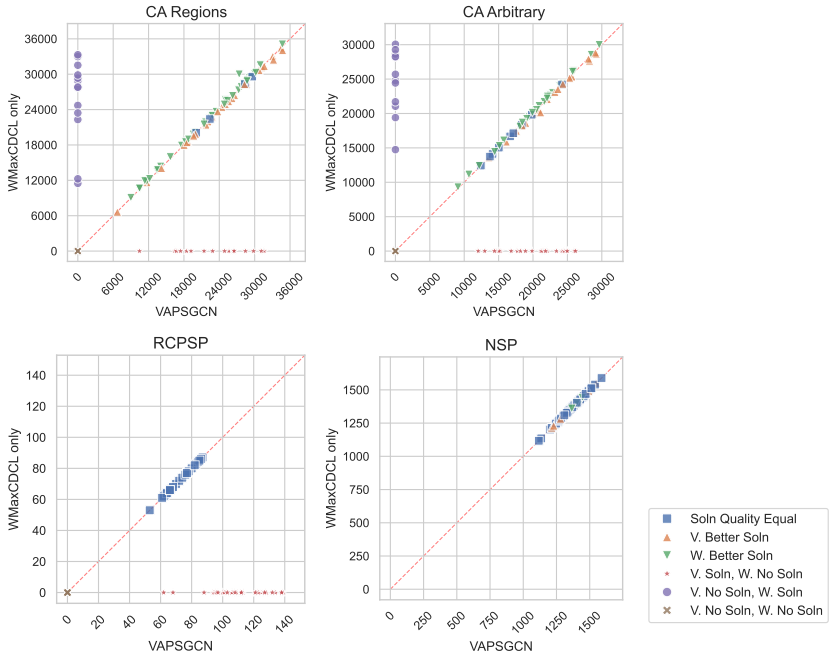


**Figure 5** Results with equal time for both solvers. The $x$-axis is the objective value of VAPSGCN-LTP (V), or 0 if it produced no solution. The $y$-axis is the objective value of WMaxCDCL (W), or 0 if it produced no solution. The shape and colour of points on the plots indicate the type of instance.

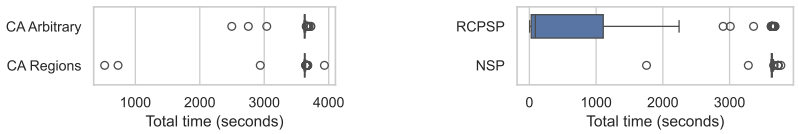


**Figure 6** Distribution of time taken by VAPSGCN-LTP in the equal time experiment.

a different way, giving them the same amount of time by adjusting the WMaxCDCL time limit to match the time taken by VAPSGCN-LTP.

For each instance $i$ of each problem class, we ran VAPSGCN-LTP (with a time limit of 1 hour) and recorded the total time $t_i$ (as a median of five runs with distinct random seeds). We then ran WMaxCDCL with a time limit of $t_i$, giving the two solvers approximately the same amount of time. (In some cases we found that WMaxCDCL overran its time limit by a small percentage. For example with CA Regions it overran by at most 1.9%.) Table 3 shows the number of instances solved (i.e. any feasible solution found) for each class. Then the 100 instances are broken down into the same six categories as in Table 2. Finally, the table summarises the results by showing the number of instances where each solver performed better than the other (i.e. produced a solution when the other did not, or produced a better solution). VAPSGCN-LTP significantly outperforms the other solver for three benchmark sets, in two cases because it is able to find a solution for more instances.

For both Combinatorial Auction classes, WMaxCDCL found better solutions for the majority of instances (where both solvers found a solution). However for NSP, VAPSGCN-LTP produces better solutions for the majority of instances. Figure 5 directly compares solution quality of the two solvers, showing that they produce solutions of very similar quality when given the same amount of time. The only substantial difference between

VAPSGCN-LTP and WMaxCDCL shown in Figure 5 is that VAPSGCN-LTP finds a feasible solution for more instances. Figure 6 shows the time distribution of VAPSGCN-LTP. Once again it shows that the four problem classes have different time distributions (particularly RCPSP), and may benefit from different confidence score thresholds.

### 6.3 Comparison to Random Pruning

In the experiments above, VAPSGCN-LTP increases the number of instances where a feasible solution is found. The mechanism is unknown: it could be that simply reducing the sizes of the problem instances makes it possible to find more feasible solutions within the time limit. Without further experimentation, it is not clear that the VAPSGCN model is providing any benefit. Therefore we performed a simple experiment with random pruning on the RCPSP class. Random pruning (Random-LTP) works as follows: first perform VAPSGCN-LTP and record the number of literals pruned ($p$); then create a list $L$ containing all literals in a random order; finally construct the set of assignments $A$ from $L$ as described in Section 3.4 (last paragraph), iterating until there are exactly $p$ literals in $A$. The result is that literals in $A$ are chosen uniformly at random, except that some literals may be rejected by the unit propagation filter (Section 3.4). The pruned MaxSAT instance is a copy of the original with literals in $A$ added as unary hard clauses.

When applying Random-LTP to RCPSP with a two-hour time limit, a feasible solution was found for 11 instances (compared to 70 for VAPSGCN-LTP and 42 for WMaxCDCL alone), with the other 89 being unsatisfiable. There were no time-outs. For the 11 satisfiable instances, Random-LTP produced the same objective value as the other two methods and its solver time was always greater than WMaxCDCL alone. The VAPSGCN model is clearly performing much better than random selection for RCPSP. The relative importance of predicting the correct value for variables versus picking a good set of variables to assign (via confidence scores) remains to be seen.

## 7 Conclusions

We have presented a novel machine learning model named VAPSGCN to predict the assignments of variables in partial weighted MaxSAT. The model exploits the natural graph structure of MaxSAT. It is based on graph convolutional networks (a type of graph neural network). VAPSGCN can be scaled to instances of different sizes without retraining. It is intended to be trained for a problem class using a set of training instances, then applied to speed up solving of other instances of the same class. We evaluated it with four benchmark sets, of three problem classes. Our experiments show that applying VAPSGCN as part of a learning-to-prune system did improve solver efficiency, in a somewhat unexpected way: we were able to find feasible solutions for more instances with VAPSGCN pruning than without, when using the same solver (WMaxCDCL, a complete solver based on branch-and-bound).

Our future work includes exploring potential improvements to the model (e.g. generalising to hypergraphs as in HyperSAT [11], or applying an attention mechanism), and looking at ways to improve the application of the model (e.g. using the predicted assignment as a solver heuristic, either alone or in conjunction with learning-to-prune). Another interesting avenue could be to prune only the variables that have a clear meaning in the original problem class (avoiding variables introduced during encoding).

## References


1 MaxSAT evaluation 2023. https://maxsat-evaluations.github.io/2023/. Accessed: 2026-03-04.

2 Graph-based attention for differentiable maxsat solving. In *Proceedings of the 39th Conference on Neural Information Processing Systems (NeurIPS 2025)*, 2025.

3 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.

4 Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. Learning to solve circuit-sat: An unsupervised differentiable approach. In *International Conference on Learning Representations*, 2019.

5 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.

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

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

8 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.

9 Vincenza Carchiolo, Marco Grassia, Michele Malgeri, and Giuseppe Mangioni. Geometric Deep Learning sub-network extraction for Maximum Clique Enumeration. *PLOS ONE*, 19(1):e0296185, January 2024. doi:10.1371/journal.pone.0296185.

10 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.

11 Qiyue Chen, Shaolin Tan, Suixiang Gao, and Jinhu Lü. HyperSAT: Unsupervised hypergraph neural networks for weighted MaxSAT problems. *arXiv preprint arXiv:2504.11885*, 2025.

12 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.

13 Jordi Coll, Chu-Min Li, Shuolin Li, Djamal Habet, and Felip Manyà. Solving weighted Maximum Satisfiability with Branch and Bound and clause learning. *Computers & Operations Research*, page 107195, 2025.

14 Dennis Craandijk and Floris Bex. Deep learning for abstract argumentation semantics. *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, Yokohama*, pages 1667–1673, 2020. Main track.

15 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.

16 James M Crawford and Andrew B Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *AAAI*, volume 2, pages 1092–1097, 1994.

17 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.

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

**19** Marco Grassia, Juho Lauri, Sourav Dutta, and Deepak Ajwani. Learning Multi-Stage Sparsification for Maximum Clique Enumeration, September 2019. arXiv:1910.00517, doi:10.48550/arXiv.1910.00517.

**20** 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.

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

**22** 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.

**23** Lars Kotthoff, Alexandre Fréchette, Tomasz P Michalak, Talal Rahwan, Holger H Hoos, and Kevin Leyton-Brown. Quantifying algorithmic improvements over time. In *IJCAI*, pages 5165–5171, 2018.

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

**25** 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.

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

**27** Zhaoyu Li, Jinpei Guo, and Xujie Si. G4SATBench: Benchmarking and advancing SAT solving with graph neural networks. *Transactions on Machine Learning Research*, 2024. URL: https://openreview.net/forum?id=7VB5db72lr.

**28** 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.

**29** Lars Malmqvist, Tangming Yuan, and Peter Nightingale. Approximating problems in abstract argumentation with graph convolutional networks. *Artificial Intelligence*, 336:104209, 2024.

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

**31** 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.

**32** 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.

**33** 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.

**34** 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.

**35** 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.

**36** Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. In *International Conference on Learning Representations*, 2019. URL: https://openreview.net/forum?id=HJMC_iA5tm.

,




**37** 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.

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

**39** 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`.

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

**41** Matthew Treinish, Ivan Carvalho, Georgios Tsilimigkounakis, and Nahum Sá. rustworkx: A high-performance graph library for python. *arXiv preprint arXiv:2110.15221*, 2021.

**42** Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

**43** Mario Vanhoucke and Broos Maenhout. Nsplib–a nurse scheduling problem library: A tool to evaluate (meta-) heuristic procedures. In *Operational research for health policy: making better decisions, proceedings of the 31st annual meeting of the working group on operations research applied to health services*, pages 151–165, 2007.

**44** 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.

**45** 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.