Can Q-Learning with Graph Networks Learn a Generalizable Branching Heuristic for a SAT Solver?

Vitaly Kurin[*]
Department of Computer Science
University of Oxford
Oxford, United Kingdom
vitaly.kurin@cs.ox.ac.uk

Saad Godil
NVIDIA
Santa Clara, California
United States
sgodil@nvidia.com

Shimon Whiteson
Department of Computer Science
University of Oxford
Oxford, United Kingdom
shimon.whiteson@cs.ox.ac.uk

Bryan Catanzaro
NVIDIA
Santa Clara, California
United States
bcatanzaro@nvidia.com

Abstract

We present Graph-Q-SAT, a branching heuristic for a Boolean SAT solver trained with value-based reinforcement learning (RL) using Graph Neural Networks for function approximation. Solvers using Graph-Q-SAT are complete SAT solvers that either provide a satisfying assignment or proof of unsatisfiability, which is required for many SAT applications. The branching heuristics commonly used in SAT solvers make poor decisions during their warm-up period, whereas Graph-Q-SAT is trained to examine the structure of the particular problem instance to make better decisions early in the search. Training Graph-Q-SAT is data efficient and does not require elaborate dataset preparation or feature engineering. We train Graph-Q-SAT using RL interfacing with MiniSat solver and show that Graph-Q-SAT can reduce the number of iterations required to solve SAT problems by 2-3X. Furthermore, it generalizes to unsatisfiable SAT instances, as well as to problems with 5X more variables than it was trained on. We show that for larger problems, reductions in the number of iterations lead to wall clock time reductions, the ultimate goal when designing heuristics. We also show positive zero-shot transfer behavior when testing Graph-Q-SAT on a task family different from that used for training. While more work is needed to apply Graph-Q-SAT to reduce wall clock time in modern SAT solving settings, it is a compelling proof-of-concept showing that RL equipped with Graph Neural Networks can learn a generalizable branching heuristic for SAT search.

1   Introduction

Boolean satisfiability (SAT) is an important problem for both industry and academia that impacts various fields, including circuit design, computer security, artificial intelligence and automatic theorem proving. As a result, modern SAT solvers are well crafted, sophisticated, reliable pieces of software that can scale to problems with hundreds of thousands of variables [33].

---

[*]The work was done when the author was a research intern at NVIDIA.

SAT is known to be NP-complete [22], and most state-of-the-art open-source and commercial solvers rely on multiple *heuristics* to speed up the exhaustive search, which is otherwise intractable. These heuristics are usually meticulously crafted using expert domain knowledge and are often iteratively refined via trial and error. In this paper, we investigate how we can use machine learning to improve upon an existing branching heuristic without leveraging domain expertise.

We present Graph-Q-SAT, a branching heuristic in a Conflict Driven Clause Learning [40, 21, CDCL] SAT solver trained with value-based reinforcement learning (RL), based on deep Q-networks [30, DQN]. Graph-Q-SAT uses a graph representation of SAT problems similar to Selsam et al. [39] which provides permutation and variable relabeling invariance. Graph-Q-SAT uses a Graph Neural Network [13, 4, GNN] as a function approximator to provide generalization as well as support for a dynamic state-action space. Graph-Q-SAT uses a simple state representation and a binary reward that requires no feature engineering or problem domain knowledge. Graph-Q-SAT modifies only part of the CDCL based solver, keeping it *complete*, i.e., always yielding a correct solution.

We demonstrate that Graph-Q-SAT outperforms Variable State Independent Decaying Sum [31, VSIDS], the most frequently used CDCL branching heuristic, reducing the number of iterations required to solve SAT problems by 2-3X. Graph-Q-SAT is trained to examine the structure of the particular problem instance to make better decisions at the beginning of the search, whereas the VSIDS heuristic suffers from poor decisions during the warm-up period.

Our work primarily focuses on the machine learning perspective and thus more work would be required to apply Graph-Q-SAT in industrial-scale SAT settings. However, Graph-Q-SAT exhibits intriguing properties which might eventually be useful for practical applications. We show that our method generalizes to problems five times larger than those it was trained on. We also show that Graph-Q-SAT generalizes across problem types from satisfiable (SAT) to unsatisfiable instances (unSAT). We show that reducing the number of iterations, in turn, could reduce wall clock time, the ultimate goal when designing heuristics. We also show positive zero-shot transfer properties of Graph-Q-SAT when the testing task family is different from the training one. Finally, we show that some of these improvements are achieved even when training is limited to a single SAT problem, demonstrating data efficiency.

2 Background

2.1 SAT problem

A SAT problem involves finding variable assignments such that a propositional logic formula is satisfied or showing that such an assignment does not exist. A propositional formula is a Boolean expression, including Boolean variables, ANDs, ORs and negations. The term literal is used to refer to a variable or its negation. It is convenient to represent Boolean formulas in conjunctive normal form (CNF), i.e., conjunctions (AND) of clauses, where a clause is a disjunction (OR) of literals. An example of a CNF is $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3)$, where $\wedge, \vee, \neg$ are AND, OR, and negation respectively. This CNF formula has two clauses: $(x_1 \vee \neg x_2)$ and $(x_2 \vee \neg x_3)$. In this work, we use SAT to denote both the Boolean Satisfiability problem and a satisfiable instance, which should be clear from the context. We use unSAT to denote unsatisfiable instances.

There are many types of SAT solvers. We focus on CDCL solvers, MiniSat [10] in particular, because it is an open-source, minimal, but powerful implementation. A CDCL solver repeats the following steps: every iteration it chooses a variable and assigns it a binary value. This is called a decision. Then, the solver simplifies the formula building an implication graph and checks whether a conflict emerged. Given a conflict, the solver can infer (learn) new clauses and backtrack to the variable assignments where the newly learned clause becomes unit (consisting of a single literal). Learnt clauses force a variable assignment which avoids the previous conflict. Sometimes, CDCL solver undoes all the variable assignments keeping the learned clauses to escape futile regions of the search space. This is called a restart.

We focus on the branching heuristic because it is one of the most heavily used during the solution procedure. The branching heuristic is responsible for picking the variable and assigning some value to it. VSIDS [31] is one of the most used CDCL branching heuristics. It is a counter-based heuristic that keeps a scalar value for each literal or variable (MiniSat uses the latter). These values are increased

every time a variable is involved in a conflict. The algorithm behaves greedily with respect to these values called *activities*. Activities are usually initialized with zeroes [27].

2.2 Reinforcement Learning

We formulate the RL problem as a Markov decision process (MDP). An MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma, \rho \rangle$ with a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a reward function $\mathcal{R}(s, a, s')$ and the transition function $\mathcal{T}(s, a, s') = p(s, a, s')$, where $p(s, a, s')$ is a probability distribution, $s, s' \in \mathcal{S}, \ a \in \mathcal{A}$. Discount factor $\gamma \in [0, 1)$ weights preferences for immediate reward relative to future reward. The last element of the tuple ρ is the probability distribution over initial states. In the case of *episodic tasks*, the state space is split into the set of non-terminal states and the terminal state $\mathcal{S}^+$. To solve an MDP means to find an optimal policy, a mapping that outputs an action or distribution over actions given a state and which maximizes the expected discounted return $R = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t]$, where $r_t = \mathcal{R}(s_t, a_t, s_{t+1})$ is the reward for the transition from s_t to s_{t+1}. In Section 3 we apply DQN, a value-based RL algorithm that approximates an optimal Q-function, an action-value function that estimates the sum of future rewards after taking an action a in state s and following an optimal policy π thereafter: $Q^*(s, a) = \mathbb{E}_{\pi, \mathcal{T}, \rho}[\mathcal{R}(s, a, s') + \gamma \max_{a'} Q^*(s', a')]$. A mean squared temporal difference (TD) error is used to make an update step: $L(\theta) = (Q_\theta(s, a) - r - \gamma \max_{a'} Q_{\bar{\theta}}(s', a'))^2$, where θ parametrizes the Q-function. Target network [30] $Q_{\bar{\theta}}$ is used to stabilize DQN. Its weights are copied from the main network Q_θ after each k minibatch updates.

2.3 Graph Neural Networks

Boolean formulas can be of arbitrary size. Moreover, during the solution procedure, some parts of the formula are eliminated and new clauses are added. We need a network architecture which does not assume the input to be of fixed size. Moreover, a Boolean formula should be invariant to the permutation of the clauses, variables and their renaming. To accommodate these requirements and also to take the problem structure into account, we use Graph Neural Networks [13, GNN] to approximate our Q-function. We use the formalism of Battaglia et al. [4], which unifies most existing GNN approaches. Under this formalism, GNN is a set of functions that take an annotated graph as input and output a graph with modified annotations but the same topology.

Here, a graph is a directed graph $\langle V, E, U \rangle$, where V is the set of vertices, E is the set of directed edges with $e_{ij} = (i, j) \in E$, $v_i, v_j \in V$, and U is a global attribute which contains the information relevant to the whole graph. We call vertices, edges, and the global attribute entities. Each entity has an associated annotation: $\boldsymbol{e}_{ij} \in \mathbb{R}^e$, $\boldsymbol{v}_i \in \mathbb{R}^v$ or $\boldsymbol{u} \in \mathbb{R}^u$. A GNN changes these annotations as a result of its operations.

A GNN is as a set of six functions: update functions ϕ_e, ϕ_v, ϕ_u and aggregation functions $\rho_{e \to v}, \rho_{e \to u}, \rho_{v \to u}$. The information propagates between vertices along graph edges. Update functions compute new entity annotations. Aggregation functions enable GNN to process graphs of arbitrary topology, compressing multiple entities features into vectors of fixed size. Summation, averaging, taking $\max$ or $\min$ are popular choices of aggregation functions.
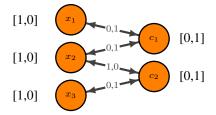
More formally, within one iteration, a GNN does the following computations (in order):

$$
\begin{aligned}
\boldsymbol{e}'_{ij} &= \phi_e(\boldsymbol{u}, \boldsymbol{e}, \boldsymbol{v}_i, \boldsymbol{v}_j) \, \forall e_{ij} \in E \\
\boldsymbol{v}'_i &= \phi_v \big[\boldsymbol{u}, \boldsymbol{v}_i, \rho_{e \to v}(\{ \boldsymbol{e}_{ki} \mid \forall e_{ki} \in E \}) \big] \, \forall v_i \in V \\
\boldsymbol{u}' &= \phi_u \big[\boldsymbol{u}, \rho_{e \to u}(\{ \boldsymbol{e}_{ij} \mid \forall e_{ij} \in E \}), \rho_{v \to u}(\{ \boldsymbol{v}_i \mid \forall v_i \in V \}) \big].
\end{aligned}
$$

A GNN performs multiple iterations to further propagate information in the graph. Neural networks that represent update functions, can be optimised end-to-end using backpropagation.

3 Graph-Q-SAT

As noted in Section 2.2, we use the MDP formalism for our purposes. Each SAT problem is an MDP sampled from a distribution of SAT problems of a specific family (e.g., random 3-SAT or graph coloring). Moreover, each problem is either satisfiable or unsatisfiable. Hence, a task is defined as follows: $\tau \sim \mathcal{D}(\phi, (un)SAT, n_{vars}, n_{clauses})$, where $\mathcal{D}$ is the distribution of SAT problems with
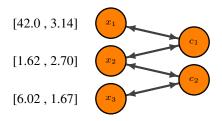
Figure 1: Bipartite graph representation of the Boolean formula $(x_1 \lor x_2) \land (\neg x_2 \lor x_3)$. The numbers next to the vertices distinguish variables and clauses. Edge labels encode literal polarities.

Figure 2: Q-function values for setting variables to *true* and *false* respectively. Taking $\arg\max$ across all Q-values of variable nodes gives an action.

ϕ defining the task family, the second argument defining the problem satisfiability and n_{vars} and $n_{clauses}$ are the number of variables and clauses respectively. Each state of the MDP consists of unassigned variables and unsatisfied clauses containing these variables. The MDP is episodic, and a terminal state is reached when a satisfying assignment is found, or the all possible options have been exhausted, proving unSAT. The action space includes two actions for each unassigned variable: assigning it to true or false. We build upon the MiniSat-based environment of [44]. We modified it to support arbitrary SAT problems and generate a graph representation of the state. It takes the actions, modifies its implication graph internally and returns a new state containing newly learned clauses and without the variables removed during propagation. Strictly speaking, this state is not fully observable. In the case of a conflict, the solver undoes the assignments for variables that are not observed by the agent. However, in practice, this should not inhibit the goal of quickly pruning the search tree: the information in the state is enough to pick a variable that leads to more propagation in the remaining formula. We use a simple reward function: the agent gets a negative reward of p for each non-terminal transition and 0 for reaching the terminal state. This reward encourages an agent to finish an episode as quickly as possible and does not require elaborate reward shaping.

SAT is an appealing problem from the RL perspective. It has the features that are hard to find in conventional RL environments. First, elements of the state/action set are of different dimensions, which is a challenging case for conventional function approximation techniques. Second, the state-space has a structured, object-oriented representation. Third, SAT allows to vary problem sizes without changing the task family, and to change the task family without changing problem sizes. Lastly, with Hoos and Stützle [18] benchmarks we use, experiments do not take weeks and are easy to iterate on.

3.1 State Representation

We represent a SAT problem as a graph similar to Selsam et al. [39]. We make it more compact, using vertices to denote variables instead of literals. We use vertices to encode clauses as well. As Figure 1 shows, our state representation is simple and does not require feature engineering. An edge (x_i, c_i) means that a clause c_i contains literal x_i. If a literal contains a negation, a corresponding edge has a $[1, 0]$ label and $[0, 1]$ otherwise. GNNs process directed graphs, so we create two directed edges with the same labels: from a variable to a clause and vice-versa. Vertex features are two-dimensional one-hot vectors, denoting either a variable or a clause. We do not provide any other information to the model. The global attribute input is empty and is only used for message passing.

3.2 Q-Function Representation

We use the encode-process-decode architecture [4], which we discuss in more detail in Appendix C.1. Similarly to Bapst et al. [3], our GNN labels variable vertices with Q-values. Each variable vertex has two actions: set the variable to true or false as shown on Figure 2. We choose the action that gives the maximal Q-value across all variable vertices. The graph contains only unassigned variables, so all actions are valid. We use DQN with common techniques such as memory replay, target network, and ϵ-greedy exploration. To expose the agent to more episodes and prevent it from getting stuck, we cap the maximum number of actions per episode similarly to the *episode length* parameter in *gym* [6]. We implement our models using Pytorch [35] and Pytorch Geometric [11].

Table 1: Number of MiniSat iterations (no restarts) to solve random 3-SAT instances.

dataset	median	mean
SAT 50-218	38	42
SAT 100-430	232	286
SAT 250-1065	62 192	76 120
unSAT 50-218	68	68
unSAT 100-430	587	596
unSAT 250-1065	178 956	182 799

Table 2: Graph-Q-SAT MRIR trained on SAT-50-218. SAT-50-218 results are for a separate validation set.

dataset	mean	min	max
SAT 50-218	2.46	2.26	2.72
SAT 100-430	3.94	3.53	4.41
SAT 250-1065	3.91	2.88	5.22
unSAT 50-218	2.34	2.07	2.51
unSAT 100-430	2.24	1.85	2.66
unSAT 250-1065	1.54	1.30	1.64

3.3 Training and Evaluation

We train our agent using Random 3-SAT instances from the SATLIB benchmark [18]. To measure generalization, we split these data into training, validation, and test sets. To illustrate the problem complexities, Table 1 provides the number of steps it takes MiniSat to solve the problem. Each random 3-SAT problem is denoted as SAT-X-Y or unSAT-X-Y, where SAT means that all problems are satisfiable, unSAT means all problems are unsatisfiable. X and Y stand for the number of variables and clauses in the initial formula. We provide more details about the datasets in Appendix C.2.

While random 3-SAT problems have relatively few variables and clauses, they have an interesting property that makes them more challenging for a solver. For this dataset, the ratio of clauses to variables is close to 4.3 to 1 which is near the *phase transition* at which it is hard to say whether the problem is SAT or unSAT [9]. In 3-SAT problems, each clause has exactly 3 variables. However, learned clauses might be of arbitrary size.

We use Median Relative Iteration Reduction (MRIR) w.r.t. MiniSat as our main performance metric: the number of iterations it takes MiniSat to solve a problem divided by Graph-Q-SAT's number of iterations. Similarly to the *median human normalized score* adopted in the Atari domain [16], we use the median instead of the mean to avoid skew from outliers. By one iteration we mean one *decision*, i.e., choosing a variable and setting it to a value. We compare ourselves with the best MiniSat results having run MiniSat with and without restarts. We cap the number of decisions our method takes at the beginning of the solution procedure and then we give control to MiniSat.

We are not interested in the absolute number of iterations per se or the total ratio between VSIDS and Graph-Q-SAT. We use these numbers as a common scale to show the generalisation, transfer and data efficiency properties of our approach.

When training, we evaluate the model every 1000 batch updates on the validation instances and pick the model with the best validation results. After that, we evaluate this model on the test set and report the results. For each model we do 5 training runs and report the average MRIR results, the maximum, and the minimum. We provide all the hyperparameters needed to reproduce our results in Appendix C. Our experimental code as well as the MiniSat *gym* environment can be found at `https://github.com/NVIDIA/GraphQSat`.

4 Experimental Results

In this section, we present empirical results for Graph-Q-SAT.

4.1 Improving upon VSIDS

In our first experiment, we consider whether it is possible to improve upon VSIDS using no domain knowledge, a simple state representation, and a simple reward function. The first row in Table 2 gives a positive answer to that question. DQN equipped with a GNN solves the problems in fewer than half the iterations of MiniSat. Graph-Q-SAT makes decisions resulting in more propagations, i.e., inferring variable values based on other variable assignments and clauses. This helps Graph-Q-SAT prune the search tree faster. For SAT-50-218, Graph-Q-SAT does on average 2.44 more propagations than MiniSat (6.62 versus 4.18). We plot the average number of variable assignments for each problem individually in the Appendix B.

These results raise the question: Why does Graph-Q-SAT outperform VSIDS? VSIDS is a counter-based heuristic that takes time to warm up. Our model, on the other hand, perceives the whole problem structure and can make more informed decisions from the beginning. To test this hypothesis, we vary the number of decisions our model makes at the beginning of the solution procedure before we hand the control back to VSIDS. The results in Figure 3 support this hypothesis. Even if our model is used for only the first ten iterations, it still improves performance over VSIDS.

One strength of Graph-Q-SAT is that VSIDS keeps being updated while the decisions are made with Graph-Q-SAT. We believe that Graph-Q-SAT complements VSIDS by providing better quality decisions in the initial phase while VSIDS is warming up. Capping the number of model calls also significantly reduces the main bottleneck of our approach – wall clock time spent on model evaluation.

4.2 Generalization Properties of Graph-Q-SAT

Next, we consider Graph-Q-SAT's generalization properties.

4.2.1 Generalization across Problem Sizes

Table 2 shows that Graph-Q-SAT has no difficulty generalizing to larger problems, showing almost 4X improvement in iterations for a dataset 5 times bigger than the training set. Graph-Q-SAT on average leads to more variable assignments changes per step, e.g., 7.58 vs 5.89 on SAT-100-430 (refer to Appendix B for detailed plots). It might seem surprising that the model performs better for larger problems. However, an increase in score for different problem sizes might also mean that the base solver scales worse than our method does for this benchmark.

4.2.2 Generalization from SAT to unSAT

An important characteristic of Graph-Q-SAT is that the problem formulation and representation makes it possible to solve unSAT problems when training only on SAT, which is problematic for some existing approaches [39]. The performance is, however, worse than the performance on satisfiable problems. On the one hand, SAT and unSAT problems are different. When the solver finds one satisfying assignment, the problem is solved. For unSAT, the algorithm needs to exhaust all possible options to prove that there is no such assignment. On the other hand, there is one important similarity between the two: an algorithm has to prune the search tree as fast as possible. Our measurements of the average number of propagations per step demonstrate that Graph-Q-SAT learns how to prune the tree more efficiently than VSIDS (6.36 vs 4.17 for unSAT-50-218, detailed plots are in Appendix B).
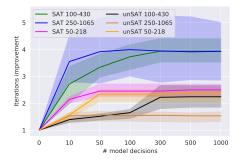
4.2.3 Transfer across Task Families

So far, we have examined the generalization properties of Graph-Q-SAT varying only the last three arguments of the task distribution defined in Section 3 ($\mathcal{D}(\phi, (un)SAT, n_{vars}, n_{clauses})$. In this section we go one step further and study Graph-Q-SAT's *zero-shot transfer* to a new task family ϕ.

This is a challenging problem. SAT problems have distinct structures, e.g., the graph representation of a random 3-SAT problem looks different than that of a graph coloring problem. GNNs learn graph local properties, i.e. how neighbouring entities' features have a global implication on Q-values. It is reasonable to expect a performance drop when changing the task family ϕ, but the magnitude of the drop gives some indication of the method's ability to transfer across task families. Therefore, we evaluate a model trained on SAT-50-218 on the flat graph coloring benchmark from SATLIB [18]. All the problems in the benchmark are satisfiable.

Table 3 shows positive transfer for Graph-Q-SAT on the graph coloring benchmark, with MRIR above 1 in five out of eight cases. As expected, MRIR is lower if than in Table 2, where the model was evaluated on the tasks sampled from the same distribution.

Training directly on the graph coloring benchmark indeed improves performance. Graph coloring benchmarks have only 100 problems each, so we do not split them into training/validation/test sets using *flat-75-180* for training and *flat-100-239* to do model selection. Table 4 shows that Graph-Q-SAT, trained on flat75-180 shows higher MRIR compared to the transferred model. Additionally, this experiment shows that Graph-Q-SAT can scale when training on larger graphs.
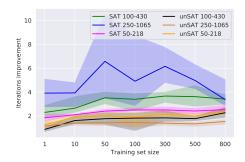
Figure 3: Graph-Q-SAT number of maximum first decisions vs performance. Graph-Q-SAT shows improvement starting from 10 iterations confirming our hypothesis of VSIDS initialization problem. The shades mark *min* and *max* values.

Figure 4: Dataset size effect on generalization. While Graph-Q-SAT profits from more data in most of the cases, it is able to generalize even from one data point. Model is trained on SAT-50-218. The shades mark *min* and *max* values.

Another intriguing property of Graph-Q-SAT generalization is that sometimes Graph-Q-SAT shows better performance when generalizing in comparison to training from scratch. Learning on SAT-100-430 requires more resources, does not generalize as well, and is generally less stable than training on SAT-50-218 and then transferring to SAT-100-430 and SAT-250-1065. Hence, generalizing with Graph-Q-SAT not only reduces the time and samples spent on training, but yields models hardly achievable by learning. We suppose the reason is that transfer is not directly affected by all the issues an RL agent faces when training: higher variance in the returns caused by longer episodes, challenges for temporal credit assignment, and difficulties with exploration.

4.3 Data Efficiency

We design our next experiment to understand how many different SAT problems Graph-Q-SAT needs to learn from. We varied the SAT-50-218 training set from a single problem to 800 problems. Figure 4 shows that Graph-Q-SAT is extremely data efficient. Having more data helps in most cases but, even with a single problem, Graph-Q-SAT generalizes across problem sizes and to unSAT instances. This should allow Graph-Q-SAT to generalize to new benchmarks without access to many problems from them. We assume that Graph-Q-SAT's data efficiency is one of the benefits of using RL. The environment allows the agent to explore diverse regions of state-action space, making it possible to learn useful policies even from a single instance. In supervised learning, data diversity is addressed at the training data generation step.

4.4 Wall-Clock Time Bottleneck

The main goal of this work is to show that RL can learn a value function that can be used as a branching heuristic in a SAT solver, and to study the model's generalisation properties. In its current form, more work would be required to apply Graph-Q-SAT in an industrial setting, where wall-clock time is the metric of success, and problem sizes are extremely large. However, we believe that Graph-Q-SAT should be of interest to the SAT community because reduction of iterations can reduce wall clock time when the number of saved iterations is large enough to tolerate the network inference timings. Due to a shortage of space, we present the wall-clock time and scaling analysis in Appendix A. This analysis shows that MRIR reduction leads to wall clock time improvements on SAT-250 and unSAT-250.

5 Related Work

Using machine learning for the SAT problem is not a new idea [14, 15, 12, 42, 47, 28]. Recently, SAT has attracted interest in the deep learning community. There are two main approaches: solving a problem end-to-end or learning heuristics while keeping the algorithm backbone the same. Selsam et al. [39, NeuroSAT] take an end-to-end supervised learning approach demonstrating that GNN can generalize to SAT problems bigger than those used for training. NeuroSAT finds satisfying assign-

Table 3: SAT-50 model's performance on SATLIB flat graph coloring benchmark. The comparison is w.r.t. MiniSat with restarts, since MiniSat performs better in this mode for this benchmark.

dataset	variables	clauses	Graph-Q-SAT MRIR		
			average	min	max
30-60	90	300	1.51	1.25	1.65
50-115	150	545	1.36	0.47	1.80
75-180	225	840	1.40	0.31	2.06
100-239	300	1117	1.44	0.31	2.38
125-301	375	1403	1.02	0.32	1.87
150-360	450	1680	0.76	0.37	1.40
175-417	525	1951	0.67	0.44	1.36
200-479	600	2237	0.67	0.54	0.87

Table 4: Graph-Q-SAT MRIR (5 training runs on 75-180, model selection with 100-239).

dataset	Graph-Q-SAT MRIR		
	average	min	max
75-180	2.44	2.25	2.70
100-239	2.89	2.77	2.98
30-60	1.74	1.33	2.00
50-115	2.08	2.00	2.13
125-301	2.43	2.20	2.66
150-360	2.07	2.00	2.11
175-417	1.98	1.69	2.21
200-479	1.70	1.38	1.98

ments for the SAT formulae and thus cannot generalize from SAT to unSAT problems. Moreover, the method is incomplete and might generate incorrect results, which is extremely important, especially for unSAT problems. Selsam and Bjørner [38] modify NeuroSAT and integrate it into popular SAT solvers to improve timing on SATCOMP-2018 benchmark. While the approach shows its potential to scale to large problems, it requires an extensive training set including over 150,000 data points. Amizadeh et al. [2] propose an end-to-end GNN architecture to solve circuit-SAT problems. While their model never produces false positives, it cannot solve unSAT problems.

The following methods take the second approach: learning a branching heuristic instead of learning an algorithm end-to-end. Jaszczur et al. [19] take the supervised learning approach using the same graph representation as Selsam et al. [39]. The authors show a positive effect of combining DPLL/CDCL solver with the learnt model. As in Selsam et al. [39], their approach requires diligent crafting of the test set. Also, the authors do not compare their approach to the VSIDS heuristic, which is known to be a crucial component of CDCL [23]. Wang and Rompf [44], whose environment we took as a starting point, show that DQN does not generalize for 20-91 3-SAT problems, whereas Alpha(Go) Zero [41] does. Our results show that the issue is related to state representation. They use CNNs, which are not invariant to variable renaming or permutations. Moreover, CNNs require a fixed input size which makes it infeasible when applying to problems with different numbers of variables or clauses.

Yolcu and Póczos [48] use REINFORCE [46] to learn the variable selection heuristic of a local search SAT solver [37]. Their algorithm is an incomplete solver and cannot work with unsatisfiable instances. They also investigate the generalisation over problem sizes on random instances near the phase transition. However, in this experiment, the training problems have ten variables only, and the number of variables in the test set does not exceed 80 with the success ratio of the algorithm staying below the baseline for the latter case.

Lederman et al. [26] train a REINFORCE [46] agent applying GNNs to replace the branching heuristic for Quantified Boolean Formulas (QBF). QBF considers a different problem allowing existential and universal quantifiers. Lederman et al. [26] note positive generalization properties across problem size for problems from similar distributions. Our work focuses more on the generalization and transfer properties of a GNN value-based RL algorithm. We investigate data efficiency properties and merge VSIDS with a trained RL agent, looking into the trade-off between the model use and its effect on the final solution. Apart from that, we show that it is possible to achieve good performance and generalization properties with a simpler state representation. Finally, doing more message propagations per step and using a GNN as a Q-function (in their case, a GNN only computes node embeddings) allows us to consider more subtle dependencies in the graph.

Look-ahead SAT solvers [17] perform more computations compared to VSIDS to evaluate the consequences of their decisions. In some of the cases, e.g. random k-SAT, this pays off. Difference heuristics used for making a decision measure reduction in the formulae before and after the decision. LRB heuristic [28] uses multi-armed bandits to explicitly optimise for the ability of the variables' to generate learnt clauses. We hypothesise, that Graph-Q-SAT might have learnt some aspects of those heuristics (Figure 8 in Appendix B). We believe that integrating Graph-Q-SAT with other types of solvers is a promising direction for future research.

Vinyals et al. [43] introduce a recurrent architecture for approximately solving complex problems, such as the Traveling Salesman Problem, approaching it in a supervised way. Bello et al. [5] consider combinatorial optimization problems with RL. Khalil et al. [24] approach combinatorial optimization using GNNs and DQN, learning a heuristic that is later used greedily. It differs from our approach in that their heuristic is effectively the algorithm itself. The environment dynamics in Khalil et al. [24] is straightforward with the next state easily inferred, given the current state and the chosen action. In the case of SAT, there are CDCL steps after the decision, and the next state might be totally different from the current one making the problem harder in terms of learning the Q-function. In addition, we use Battaglia et al. [4] which is more expressive than `structure2vec` used in Khalil et al. [24]. The global attribute in Battaglia et al. [4] can facilitate message passing in case of a bigger graph. Having separate updaters for edges and nodes leads to more powerful representations. And, finally, an edge updater of Battaglia et al. [4] can learn better pairwise interaction between the sender and the receiver, enabling sending different messages to different nodes.

Paliwal et al. [34] use GNNs with imitation learning for theorem proving. Carbune et al. [8] propose a general framework of injecting an RL agent into existing algorithms. Cai et al. [7] use RL to find a suboptimal solution that is further refined by another optimization algorithm, in their case, simulated annealing [25, SA]. It is not restricted to SA, and this modularity is valuable. However, it is also a drawback because the second optimization algorithm might benefit more from the first if they were interleaved. For instance, Graph-Q-SAT can guide search before VSIDS overcomes its initialization bias.

GNNs have enabled the study of RL agents in state/action spaces of dynamic size, which is crucial for generalization beyond the given task. Wang et al. [45] and Sanchez-Gonzalez et al. [36] consider GNNs for the control problem generalization. Bapst et al. [3] report strong generalization capabilities for the construction task. Multi-agent research [20, 29, 1] shows that GNN benefits from invariance to the number of agents in the team or other environmental entities.

6 Conclusions and Future Work

In this paper, we demonstrated that Q-learning can be used to learn the branching heuristic of a SAT solver. Graph-Q-SAT uses a simple state representation and does not require elaborate reward shaping. We show empirically that Graph-Q-SAT causes more variable propagations per step, solving the SAT problem in fewer iterations than VSIDS. For larger problems, we showed that fewer iterations could, in turn, reduce wall-clock time.We demonstrated its generalization abilities, showing more than 2-3X reduction in iterations for problems up to 5X larger and 1.5-2X from SAT to unSAT. We showed how Graph-Q-SAT improves VSIDS and that it is data-efficient. We also demonstrated positive transfer properties when changing the task family and showed that training on data from other distributions could lead to further performance improvements.

Although we showed the powerful generalization properties of graph-based RL on SAT, we believe the problem is still far from solved. More work is needed before Graph-Q-SAT is ready to compete with branching heuristics in a modern industrial setting. The two main direction of future applied research are scaling and wall-clock time reduction. Some possible ways of tackling these issues include combining the machine learning improvements from above together with an efficient C++ implementation, using a smaller network, reducing the network polling frequency, and replacing the variable activities with Graph-Q-SAT's output, similarly to Selsam and Bjørner [38].

From the machine learning perspective, it is intriguing to study how combining benchmarks from different domains might improve the transfer behavior. Further research will focus on scaling Graph-Q-SAT using the latest stabilizing techniques [16] and more sophisticated exploration methods. Building an efficient curriculum is another important step towards further scaling the method, motivated by Bapst et al. [3]. Newsham et al. [32] show that the graph structure of SAT problems affects the problem complexity. We are interested in understanding how the structure influences the performance of Graph-Q-SAT and how we can exploit this knowledge to improve Graph-Q-SAT.

Broader Impact

We believe that further progress in machine learning can have a profound economic, societal and political impact. It is hard to predict a particular effect of our method on society but, in general, we

believe that the society might benefit from our research through its impact on industry and academia. We consider two examples below.

SAT has a profound impact on circuit design, computer security, artificial intelligence, automatic theorem proving, and combinatorial optimisation, among others. For academia, Graph-Q-SAT code and results give a playground to work on GNN scaling, generalisation in RL, transfer and multitask learning, and incorporating a machine learning model with a well established algorithm. It can encourage collaboration between the applied ML and SAT communities. Analysing the behaviour of learned models might give human designers more insights to boost further research.

For industry, having faster SAT solvers would lead to faster production cycles and faster rate of progress as well as to more robust products. In circuitry design, for example, SAT is used for hardware verification. As a result, faster SAT solvers will eventually lead to fewer faults in hardware.

Like any technology, our method also carries potential risks. Further automation might reduce the need for human labour. If not managed and regulated properly, machine learning progress might also exacerbate social and economic inequality.

Acknowledgments and Disclosure of Funding

References

[1] A. Agarwal, S. Kumar, K. P. Sycara, and M. Lewis. Learning transferable cooperative behavior in multi-agent teams. In A. E. F. Seghrouchni, G. Sukthankar, B. An, and N. Yorke-Smith, editors, *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20, Auckland, New Zealand, May 9-13, 2020*, pages 1741–1743. International Foundation for Autonomous Agents and Multiagent Systems, 2020. URL `https://dl.acm.org/doi/abs/10.5555/3398761.3398967`.

[2] S. Amizadeh, S. Matusevych, and M. Weimer. Learning to solve circuit-sat: An unsupervised differentiable approach. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL `https://openreview.net/forum?id=BJxgz2R9t7`.

[3] V. Bapst, A. Sanchez-Gonzalez, C. Doersch, K. L. Stachenfeld, P. Kohli, P. W. Battaglia, and J. B. Hamrick. Structured agents for physical construction. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 464–474. PMLR, 2019. URL `http://proceedings.mlr.press/v97/bapst19a.html`.

[4] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. F. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, Ç. Gülçehre, H. F. Song, A. J. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. R. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu. Relational inductive biases, deep learning, and graph networks. *CoRR*, abs/1806.01261, 2018. URL `http://arxiv.org/abs/1806.01261`.
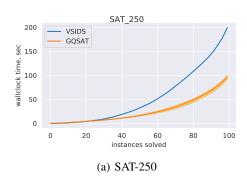
[5] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio. Neural combinatorial optimization with reinforcement learning. *CoRR*, abs/1611.09940, 2016. URL http://arxiv.org/abs/1611.09940.

[6] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016. URL http://arxiv.org/abs/1606.01540.

[7] Q. Cai, W. Hang, A. Mirhoseini, G. Tucker, J. Wang, and W. Wei. Reinforcement learning driven heuristic optimization. *CoRR*, abs/1906.06639, 2019. URL http://arxiv.org/abs/1906.06639.

[8] V. Carbune, T. Coppey, A. Daryin, T. Deselaers, N. Sarda, and J. Yagnik. Smartchoices: Hybridizing programming and machine learning, 2018.

[9] P. C. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. In J. Mylopoulos and R. Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence. Sydney, Australia, August 24-30, 1991*, pages 331–340. Morgan Kaufmann, 1991. URL http://ijcai.org/Proceedings/91-1/Papers/052.pdf.

[10] N. Eén and N. Sörensson. An extensible sat-solver. In E. Giunchiglia and A. Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003. doi: 10.1007/978-3-540-24605-3_37. URL https://doi.org/10.1007/978-3-540-24605-3_37.

[11] M. Fey and J. E. Lenssen. Fast graph representation learning with pytorch geometric. *CoRR*, abs/1903.02428, 2019. URL http://arxiv.org/abs/1903.02428.

[12] A. Flint and M. B. Blaschko. Perceptron learning of SAT. In P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 2780–2788, 2012. URL http://papers.nips.cc/paper/4533-perceptron-learning-of-sat.

[13] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE, 2005.

[14] C. Grozea and M. Popescu. Can machine learning learn a decision oracle for NP problems? A test on SAT. *Fundam. Informaticae*, 131(3-4):441–450, 2014. doi: 10.3233/FI-2014-1024. URL https://doi.org/10.3233/FI-2014-1024.

[15] S. Haim and T. Walsh. Restart strategy selection using machine learning techniques. In O. Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 312–325. Springer, 2009. doi: 10.1007/978-3-642-02777-2_30. URL https://doi.org/10.1007/978-3-642-02777-2_30.

[16] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. G. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. In S. A. McIlraith and K. Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 3215–3222. AAAI Press, 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17204.

[17] M. Heule and H. van Maaren. Look-ahead based SAT solvers. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 155–184. IOS Press, 2009. doi: 10.3233/978-1-58603-929-5-155. URL https://doi.org/10.3233/978-1-58603-929-5-155.

[18] H. H. Hoos and T. Stützle. Satlib: An online resource for research on sat. *Sat*, 2000:283–292, 2000. URL `https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html`. accessed September 18, 2019.

[19] S. Jaszczur, M. Luszczyk, and H. Michalewski. Neural heuristics for SAT solving. *CoRR*, abs/2005.13406, 2020. URL `https://arxiv.org/abs/2005.13406`.

[20] J. Jiang, C. Dun, and Z. Lu. Graph convolutional reinforcement learning for multi-agent cooperation. *CoRR*, abs/1810.09202, 2018. URL `http://arxiv.org/abs/1810.09202`.

[21] R. J. B. Jr. and R. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In B. Kuipers and B. L. Webber, editors, *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97, July 27-31, 1997, Providence, Rhode Island, USA*, pages 203–208. AAAI Press / The MIT Press, 1997. URL `http://www.aaai.org/Library/AAAI/1997/aaai97-032.php`.

[22] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi: 10.1007/978-1-4684-2001-2_9. URL `https://doi.org/10.1007/978-1-4684-2001-2_9`.

[23] H. Katebi, K. A. Sakallah, and J. P. M. Silva. Empirical study of the anatomy of modern sat solvers. In K. A. Sakallah and L. Simon, editors, *Theory and Applications of Satisfiability Testing - SAT 2011 - 14th International Conference, SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011. Proceedings*, volume 6695 of *Lecture Notes in Computer Science*, pages 343–356. Springer, 2011. doi: 10.1007/978-3-642-21581-0_27. URL `https://doi.org/10.1007/978-3-642-21581-0_27`.

[24] E. B. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song. Learning combinatorial optimization algorithms over graphs. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 6348–6358, 2017. URL `http://papers.nips.cc/paper/7214-learning-combinatorial-optimization-algorithms-over-graphs`.

[25] S. Kirkpatrick, D. G. Jr., and M. P. Vecchi. Optimization by simmulated annealing. *Sci.*, 220 (4598):671–680, 1983.

[26] G. Lederman, M. N. Rabe, S. Seshia, and E. A. Lee. Learning heuristics for quantified boolean formulas through reinforcement learning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=BJluxREKDB`.

[27] J. H. Liang, V. Ganesh, E. Zulkoski, A. Zaman, and K. Czarnecki. Understanding VSIDS branching heuristics in conflict-driven clause-learning SAT solvers. In N. Piterman, editor, *Hardware and Software: Verification and Testing - 11th International Haifa Verification Conference, HVC 2015, Haifa, Israel, November 17-19, 2015, Proceedings*, volume 9434 of *Lecture Notes in Computer Science*, pages 225–241. Springer, 2015. doi: 10.1007/978-3-319-26287-1_14. URL `https://doi.org/10.1007/978-3-319-26287-1_14`.

[28] J. H. Liang, V. Ganesh, P. Poupart, and K. Czarnecki. Learning rate based branching heuristic for SAT solvers. In N. Creignou and D. L. Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 123–140. Springer, 2016. doi: 10.1007/978-3-319-40970-2_9. URL `https://doi.org/10.1007/978-3-319-40970-2_9`.

[29] A. Malysheva, T. T. K. Sung, C. Sohn, D. Kudenko, and A. Shpilman. Deep multi-agent reinforcement learning with relevance graphs. *CoRR*, abs/1811.12557, 2018. URL `http://arxiv.org/abs/1811.12557`.

[30] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nat.*, 518(7540):529–533, 2015. doi: 10.1038/nature14236. URL https://doi.org/10.1038/nature14236.

[31] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, pages 530–535. ACM, 2001. doi: 10.1145/378239.379017. URL https://doi.org/10.1145/378239.379017.

[32] Z. Newsham, V. Ganesh, S. Fischmeister, G. Audemard, and L. Simon. Impact of community structure on SAT solver performance. In C. Sinz and U. Egly, editors, *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 252–268. Springer, 2014. doi: 10.1007/978-3-319-09284-3_20. URL https://doi.org/10.1007/978-3-319-09284-3_20.

[33] O. Ohrimenko, P. J. Stuckey, and M. Codish. Propagation via lazy clause generation. *Constraints An Int. J.*, 14(3):357–391, 2009. doi: 10.1007/s10601-008-9064-x. URL https://doi.org/10.1007/s10601-008-9064-x.

[34] A. Paliwal, S. M. Loos, M. N. Rabe, K. Bansal, and C. Szegedy. Graph representations for higher-order logic and theorem proving. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 2967–2974. AAAI Press, 2020. URL https://aaai.org/ojs/index.php/AAAI/article/view/5689.

[35] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 8024–8035, 2019. URL http://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.

[36] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. A. Riedmiller, R. Hadsell, and P. W. Battaglia. Graph networks as learnable physics engines for inference and control. In J. G. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4467–4476. PMLR, 2018. URL http://proceedings.mlr.press/v80/sanchez-gonzalez18a.html.

[37] B. Selman, H. A. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In D. S. Johnson and M. A. Trick, editors, *Cliques, Coloring, and Satisfiability, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, October 11-13, 1993*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 521–531. DIMACS/AMS, 1993. doi: 10.1090/dimacs/026/25. URL https://doi.org/10.1090/dimacs/026/25.

[38] D. Selsam and N. Bjørner. Guiding high-performance SAT solvers with unsat-core predictions. In M. Janota and I. Lynce, editors, *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pages 336–353. Springer, 2019. doi: 10.1007/978-3-030-24258-9_24. URL https://doi.org/10.1007/978-3-030-24258-9_24.

[39] D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, and D. L. Dill. Learning a SAT solver from single-bit supervision. In *7th International Conference on Learning Representations,*

13

ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net, 2019. URL `https://openreview.net/forum?id=HJMC_iA5tm`.

[40] J. P. M. Silva and K. A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, 48(5):506–521, 1999. doi: 10.1109/12.769433. URL `https://doi.org/10.1109/12.769433`.

[41] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. P. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of go without human knowledge. *Nat.*, 550(7676):354–359, 2017. doi: 10.1038/nature24270. URL `https://doi.org/10.1038/nature24270`.

[42] R. Singh, J. P. Near, V. Ganesh, and M. Rinard. Avatarsat: An auto-tuning boolean sat solver. 2009.

[43] O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2692–2700, 2015. URL `http://papers.nips.cc/paper/5866-pointer-networks`.

[44] F. Wang and T. Rompf. From gameplay to symbolic reasoning: Learning SAT solver heuristics in the style of alpha(go) zero. *CoRR*, abs/1802.05340, 2018. URL `http://arxiv.org/abs/1802.05340`.

[45] T. Wang, R. Liao, J. Ba, and S. Fidler. Nervenet: Learning structured policy with graph neural networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL `https://openreview.net/forum?id=S1sqHMZCb`.

[46] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8:229–256, 1992. doi: 10.1007/BF00992696. URL `https://doi.org/10.1007/BF00992696`.

[47] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Satzilla: Portfolio-based algorithm selection for SAT. *CoRR*, abs/1111.2249, 2011. URL `http://arxiv.org/abs/1111.2249`.

[48] E. Yolcu and B. Póczos. Learning local search heuristics for boolean satisfiability. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 7990–8001, 2019. URL `http://papers.nips.cc/paper/9012-learning-local-search-heuristics-for-boolean-satisfiability`.

A Wall-clock Time and Scaling Analysis

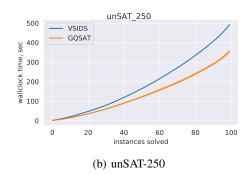
(a) SAT-250

(b) unSAT-250

Figure 5: Graph-Q-SAT's MRIR improvement (10 model calls) results in the wall clock time reduction. The curves show the averaged performance across five runs with the shade denoting the worst and the best runs.

Figure 5 demonstrates that reduction in the number of iterations together with limiting the number of model calls results in wall clock time improvements for the datasets where the number of saved iterations is large enough to tolerate the network inference timings.

More generally, we can anticipate the settings in which Graph-Q-SAT will yield an improvement in wall clock time over VSIDS by analyzing the factors contributing to their runtime performance. Assuming VSIDS's computational cost is negligible, we can compute the wall clock time of MiniSat with VSIDS as follows: $W_{VSIDS} = \sum_{t=1}^{T} P(t)$, where $P(t)$ is the unit propagation time (a procedure of formula simplification after the branching decision is made). Similarly, Graph-Q-SAT saves some fraction of iterations at the cost of added neural network inference time: $W_{\text{Graph-}Q\text{-SAT}} = \sum_{t=1}^{T/S} P(t) + \sum_{t=1}^{K} I(t)$, where S is the reduction of the number of iterations, K is the number of our model forward passes ($K << T$ for larger problems), and $I(t)$ is the network inference time. Thus, Graph-Q-SAT leads to wall clock speed ups when the total inference time stays below the time spent on propagation for the reduced number of VSIDS decisions. This seems plausible assuming that T's growth is unbounded, $K << T$ and linear dependence of $I(t)$ on the number of vertices. To check the linear dependence, we generated 10^5 graphs with characteristics similar to random 3-SAT problems (bipartite graph, each variable is connected to 13 clauses, and clause/variable ratio is 4). Figure 6 confirms that the dependence is linear.

B Propagations per step

Figure 8 shows that on average using Graph-Q-SAT leads to more propagations per step than VSIDS.

C Reproducibility

C.1 Model architecture

We use Encoder-Process-Decode architecture. Encoder and decoder are independent graph networks, i.e. MLPs taking whole vertex or edge feature matrix as a batch without message passing. We call the middle part 'the core'. The output of the core is concatenated with the output of the encoder and gets fed to the core again. We describe all the hyperparameters in Appendix C.3. We also plan to release the experimental code and the modified version of MiniSat to use as a gym environment.

C.2 Dataset

We split SAT-50-218 into three subsets: 800 training problems, 100 validation and 100 test problems. For generalization experiments, we use 100 problems from all the other benchmarks.

For graph colouring experiments, we train our models using all problems from flat-75-180 dataset. We select a model, given the performance on all 100 problems from flat-100-239. So, evaluation on
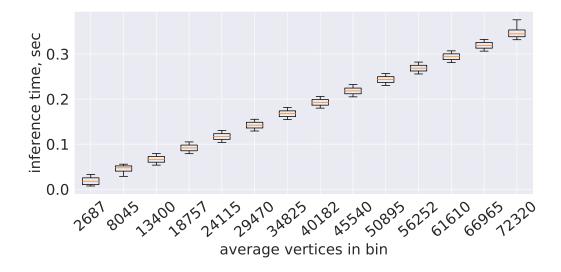
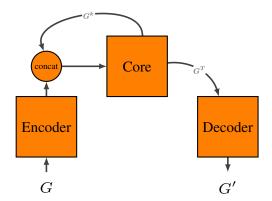Figure 6: Graph-Q-SAT inference time linearly depends on the number of vertices in the graph.

Figure 7: Encode-Process-Decode architecture. Encoder and Decoder are independent graph networks, i.e. MLPs taking whole vertex/edge data array as a batch. k is the index of a message passing iteration. When concatenating for the first time, encoder output is concatenated with zeros.

these two datasets should not be used to judge the performance of the method, and they are shown separately in Table 4. All the data from the second part of the table was not seen by the model during training (flat-30-60, flat-50-115, flat-125-301, flat-150-360, flat-175-417, flat-200-479).

C.3   Hyperparameters

Table 5 contains all the hyperparameters necessary to replicate our results.
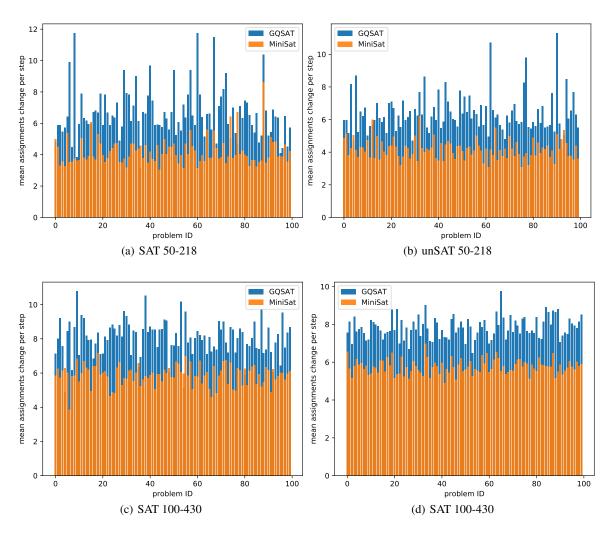
C.4   Graph-Q-SAT pseudocode

16

Figure 8: Average number of variable assignments change per step for (un)SAT-50-218 and (un)SAT-100-430.

Algorithm 1 Graph-Q-SAT Action Selection

Input: graph network GN_θ, state graph $G_s := (V, E, U)$,
with vertex features $V = [V_{vars}, V_{clauses}]$, edge features E, and a global compontent U.

$V', E', U' = GN(V, E, U)$;
$VarIndex, VarPolarity = \arg\max_{ij} V'_{vars}$;
Return $VarIndex, VarPolarity$;

Table 5: Graph-Q-SAT hyperparameters.

Hyperparameter	Value	Comment
DQN		
– Batch updates	50 000	
– Learning rate	0.00002	
– Batch size	64	
– Memory replay size	20 000	
– Initial exploration ϵ	1.0	
– Final exploration ϵ	0.01	
– Exploration decay	30 000	Environment steps.
– Initial exploration steps	5000	Environment steps, filling the buffer, no training.
– Discounting γ	0.99	
– Update frequency	4	Every 4th environment step.
– Target update frequency	10	
– Max decisions allowed for training	500	Used a safety against being stuck at the episode.
– Max decisions allowed for testing	500	Varied among [0, 10, 50, 100, 300, 500, 1000] for the experiment on Figure 3.
– Step penalty size p	-0.1	
Optimization		
– Optimizer	Adam	
– Adam betas	0.9, 0.999	Pytorch default.
– Adam eps	1e-08	Pytorch default.
– Gradient clipping	1.0	0.1 for training on the graph coloring dataset.
– Gradient clipping norm	L_2	
– Evaluation frequency	1000	
Graph Network		
– Message passing iterations	4	
– Number of hidden layers for GN core	1	
– Number of units in GN core	64	
– Encoder output dimensions	32	For vertex, edge and global updater.
– Core output dimensions	64,64,32	For vertex, edge and global respectively.
– Decoder output dimensions	32	For vertex updater, since only Q values are used, no need for edge/global updater.
– Activation function	ReLU	For everything but the output transformation.
– Edge to vertex aggregator $\rho_{e \to v}$	sum	
– Variable to global aggregator $\rho_{v \to u}$	average	
– Edge to global aggregator $\rho_{e \to u}$	average	
– Normalization	Layer Normalization	After each GN updater

Algorithm 2 Graph-Q-SAT Training Procedure

Input: Set of tasks $\mathcal{S} \sim \mathcal{D}(\phi, (un)SAT, n_{vars}, n_{clauses})$ split into $\{\mathcal{S}_{train}, \mathcal{S}_{validation}, \mathcal{S}_{test}\}$, ϕ is the task family (e.g. random 3-SAT, graph coloring). All hyperparameters are from Table 5.
Randomly Initialize Q-network GN_θ;
$updates = 0$;
$totalEnvSteps = 0$;
repeat
  repeat
    Sample a SAT problem $p \sim \mathcal{S}_{train}$;
    Initialize the environment $env = SatEnv(p)$;
    Reset the environment $s = env.reset()$;
    take action
    $a = \begin{cases} random(\mathcal{A}), \text{with probability } \epsilon \\ selectAction(s), \text{with probability } 1 - \epsilon \end{cases}$
    Take env step $s', r, done = env.step(a)$;
    $totalEnvSteps += 1$;
    dump experience $buffer.add(s, s', r, done, a)$;
    if $totalEnvSteps$ mod $updateFreq == 0$; **then**
      Do a DQN update;
    end if
    if $totalEnvSteps$ mod $validateFreq == 0$; **then**
      Evaluate GN_θ on $\mathcal{S}_{validation}$;
    end if
  until Proved SAT/unSAT ($done$ is *True*)
until $updates == totalBatchUpdates$
Pick the best model GN_θ given validation scores;
Test the model GN_θ on $\mathcal{S}_{test}$;