Report on *Can Q-Learning with Graph Networks Learn a Generalizable Branching Heuristic for a SAT Solver*

1 Summary and Review

The original paper[1] was authored by Vitaly Kurin, Saad Godil, Shimon Whiteson and Bryan Catanzaro. It was accepted in NeurIPS, 2020. The authors have presented Graph-Q-SAT, a new branching heuristic which when used upon the existing branching heuristic such as Variable State Independent Decaying Sum[5, VSIDS] in a Conflict Driven Clause Learning[2,3, CDCL] SAT solver improves the performance of the solver in terms of number of iterations or variable assignment decisions(selecting a variable in propositional formula and assigining it either True or False). This report is a part of assessment for cse 257: Search and Optimization course at UCSD. The section 1.1 gives the introduction of the SAT solving task, other background knowledge about Graph Neural Networks[7, GNN] and Reinforcement learning and summarizes the proposed Graph-Q-SAT approach. The related work is discussed in the section 1.2 and Technical details of Graph-Q-SAT, final results and review of this paper are presented in the sections 1.3, 1.4 and 1.5 respectively. The custom experiments undertaken in this project to verify the author's claims are discussed in Section 2. Section 3 discusses the extension to the original work.

1.1 Introduction

1.1.1 Background

Boolean Satisfiability is a well known problem in computer science and logic which aims at finding an assignment for all the variables which satisfies a propositional logic formula or show that no joint assignment exists that satisfies the formula. It is an important problem with applications in product configuration, hardware verification, automatic theorem proving, scheduling, cryptanalysis etc. A problem is termed as SAT if a satisfying assignment of variables exists and otherwise it is called unSAT. In the paper[1] as well as in this report, the term SAT is used for the complete boolean satisfiability problem(problem of finding whether a satisfiable assisgnment exists or not) except otherwise stated. A propositional logic formula is composed of variables and operators - AND(denoted by $\wedge$, also called conjunction), OR(denoted by $\vee$, also called disjunction), NOT(denoted by $\neg$, also called negation). A literal is defined as a form in which a variable appears which can be its original form or its negated form example:$\neg$ x,x both are termed as literals in this literature. A clause is defined as the disjunction of literals. The propositional logic formulas are represented as conjunction of multiple clauses which is also called conjunctive normal form(CNF). In this report as well as the original paper, the SAT problems under consideration are always present in CNF.

There are many different types of SAT solvers, both open-source and commercial, available to solve the SAT problem. In the paper, a Conflict Driven Clause Learning(CDCL) solver, MiniSAT[8, minisat] that employs Variable State Independent Decaying Sum(VSIDS) as branching heuristic which uses counter based heuristic to make decisions, has been considered upon which Graph-Q-SAT is used as branching heuristic. At every step or iteration(the term iteration refers to variable assignment by solver and both are used interchangeably in this report), CDCL selects a variable and assigns it a value randomly. This causes force assignment of other variables which creates an implication graph. Conflicts arise in these implication graph when one variable is being forced to

35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

take two different values. In case of conflict, the solver learns new clauses from the ones involved in the conflict and performs non-chronological backtrack. VSIDS maintains a score or counter for each variable in each polarity and updates this score whenever the solver learn new clauses due to conflict. In each iteration, VSIDS selects an unassigned variable and polarity with maximum counter value, performs the assignment and carry out unit propagations, further enhancing implication graph and check for conflicts.

Reinforcement learning formulates a decision making problem. It models a decision making problem as a Markov Decision Process(MDP). An MDP is usually represented by a tuple <S, A, R, T, γ, ρ > models an environment with set of states(S) and actions(A). When an agent is in state s and takes an action a, the transition function T determines the next state the agent reaches. The reward r is sampled from the reward distribution R(r | s,a,s') and the transition function samples the next state s' from the distribution P(s'|s,a). ρ determines the initial probability distribution over the states. The γ represents the discounting factor which is used in the calculation of cumulative reward. It is used to reduce the weightage of future rewards in the cumulative reward sum at each state s. The ultimate goal of an agent is to learn a good policy π(als), probability distribution over actions that an agent can take in state s, which can lead to the maximum expected cumulative reward given by $\sum_{t=0}^{\infty} \gamma^t r_t$. An action value function(also known as q-function) for a policy π denoted by Q_π(s,a) is defined as the expected cumulative reward an agent will receive by taking action a in state s and following policy π. There are two kinds of tasks - (a) continuous which never terminates or don't have a terminal state and (b) episodic - which has a terminal state and the environment-agent interaction terminates after finite number of time steps. When the state space becomes very large or infinite(continuous state space), we approximate the q-value function using a function approximator such as a linear function, tile coding or a neural network. The function approximation facilitates both generalization and discrimination. There are multiple ways of learning the optimal policy and Q-value functions. When the MDP is fully specified or known, dynamic programming approaches can be used such as value iteration, policy iteration etc. When the dynamics of MDP(transition function and reward distribution) is not known, sample based methods are used which approximate the q-values using the average rewards received by the agent for each state and action. Examples of sample based methods include - monte carlo policy iteration etc. Temporal Difference(TD) learning algorithms combines the best of both dynamic programming and the sample based methods. TD is an incremental learning algorithm which updates the past estimate of an action-value estimate using the current action-value estimate. Q-learning is an example of TD learning which updates the previous state-action value estimate using the maximum action value estimate possible from the current state. Following is the q-learning update rule: $Q(s,a) \leftarrow Q(s,a) + \alpha\big(r_t + \gamma \max_{a'} Q(s',a') - Q(s,a)\big)$ where α is the learning rate. Q-learning approaches employing deep neural networks as q-value function approximators are referred to as Deep Q-Network(DQN)-[9]. DQN is trained by computing the derivatives of a mean squared temporal error loss given by $L(\theta) = \big(Q_\theta(s,a) - r_t - \gamma \max_{a'} Q_{\overline{\theta}}(s',a')\big)^2$ where θ are the parameters of DQN and $\overline{\theta}$ are the parameters of target DQN which is the copy of DQN and used to compute the labels i.e. Q(s',a'). DQN agent keeps an experience replay buffer with the recent transitions of the agent. The DQN samples a minibatch of transitions from this buffer to make the q-learning update. The target DQN works in an evaluation mode and its parameters are updated by copying the agent's parameters after fixed interval of minibatch updates. This is done in order to make the targets stable while learning.

Graph Neural Networks(GNN) offers a function approximation to the data that has an inherent property of permutation invariance. Such data normally has a graphical structure or can be modelled by a graphical structure. The input to the GNN is an annotated directed graph G <V,E,U> where V is the set of vertices, E is the set of directed edges with $e_{ij} \in E, v_i, v_j \in V$ and U is a global graph level attribute. Each vertex, edge and global attribute has an associated feature vector and GNN processes this input graph and return an output graph which has same structure but updated feature vectors. GNN comprises of two class of operations - (a) update functions - ϕ_e, ϕ_v, ϕ_u and (b) aggregate functions - $\rho_{e\to v}, \rho_{e\to u}, \rho_{v\to u}$. The information propagates between the vertices through the edges and from edges and vertices to the global attribute. The aggregate functions are used to collect or summarize the information from multiple entities such as for a vertex v and edges e, $\rho_{e\to v}$ is used to compress all the information in the form of feature vectors from the incoming edges of vertex v. The update functions use this aggregated information and the previous value of the features to compute new feature vectors. The graph topology plays a crucial role here. The feature vector of a vertex is most and directly affected by those vertices which are direct neighbours of a vertex. In each message passing step of GNN, the receptive field of every node increases as it begins to cater the

information from the nodes which are several hops away. In general, a full Graph Neural Network block[7] uses equations (1),(2) and (3) to update the feature vectors of the nodes, edges and the global attribute in one message passing iteration. v_i is the initial feature vector and v_i' is the updated feature vector of vertex $i \in V$ respectively. v is used to denote all the vertex vectors combined maybe in stacked form. Similarly e_{ij} are the initial and e_{ij}' are the updated feature vectors respectively. e is used to denote the feature vectors of all edges combined may be in stacked form. Similarly, u and u' denote initial and updated feature vectors. The equation (1) aggregates the current global feature, edge and feature vectors of source v_i and destination v_j vertices and return updated edge feature vector e_{ij}'. Similarly, equation (2) and (3) updates feature vectors of each vertex and global attribute respectively. GNN can perform multiple message passing iterations. It depends on the application which output features are useful in the end - node, edge or global attribute. This output attribute is finally used in classification, regression etc tasks and the neural network parameters are updated using backpropagation in end-to-end fashion. compute There are several variants of GNN which modifies these equation, may or may not use feature vectors for vertex, egdes or global attributes. Some of these examples can be found in [7] as well. We will talk about the type of GNN used in this paper in subsection 1.3.

$$e'_{ij} = \phi_e(u, e, v_i, v_j) \forall e_{ij} \in E \tag{1}$$

$$v'_i = \phi_v \left[u, v_i, \rho_{e\to v}\big(\{e_{ki} | \forall e_{ki} \in E\}\big)\right] \forall v_i \epsilon V \tag{2}$$

$$u' = \phi_u \left[u, \rho_{e\to u}\big(\{e_{ij} | \forall e_{ij} \in E\}\big), \rho_{v\to u}\big(\{v_i | \forall v_i \in V\}\big)\right] \tag{3}$$

1.1.2 Proposed Approach

The main motivation behind this work is to use machine learning to enhance the performance of the existing solver. Graph-Q-SAT has been proposed to be used as a decision heuristic to guide the search for few initial decisions and then giving back the control to the base solver. VSIDS heuristic based CDCL solver is used as base solver in the original paper[1] as well as in this report. Improving the performance of SAT solvers has many advantages due to wide applications of boolean satisfiability problem in industry and it also gives us strong motivation to use machine learning for combinatorial search and optimization problems.

The authors have made the following claims in their paper -

- Using Graph-Q-SAT as initial heuristic can reduce the number of iterations by 2-3 times as compared to using only minisat solver
- Graph-Q-SAT generalizes to unsatisfiable SAT instances(after trained on SAT instances), as well as to problems with 5X more variables than it was trained on and demonstrate improvement in performance(in terms of number of iterations) as well which leads to reduction in wallclock time
- positive zero-shot transfer behavior when testing Graph-Q-SAT on a task family different from that used for training
- Improvements are achieved even when training is limited to a single SAT problem, demonstrating data efficiency

1.2 Related Work

The authors have done a good job in citing the papers which are highly relevant to the methods used to create Graph-Q-SAT. The authors discussed the two main approaches of using machine learning for solving SAT problems - training end-to-end supervised models or learning generalized branching heuristics and using with an existing solver. The paper does not overclaim the novelty of the proposed approach. The authors have given the due credits for their ideas of developing different components of Graph-Q-Sat to other related research work. This graphical representation of the CNF formula used in this paper has been inspired from the NeuroSAT[4] - which employs end-to-end training of a Graph Neural Network based classifier which predicts whether a given CNF formula is satisfiable or not. Though certain changes have been made to the representation used in this paper.

The authors also refers to Jaszczur el al. [10] which use message-passing Graph Neural Network, exploring attention based aggregation as well, as choose_literal heuristics in DPLL and CDCL solvers. In this work, the trained GNN is used as branching heuristic which is highly related to the use of GNN based DQN agent in Graph-Q-SAT which is used as branching heuristic over a CDCL solver. Authors also mentioned about Lederman et al.[14], who trains REINFORCE agent by employing GNN as a branching heuristic for Quantified Booelan Formulas(QBF) which consider different problems involving universal and existential quantifiers. Although their work was similar in applying reinforcement learning algorithm by employing GNN as branching heuristic, their work is focussed on QBF whereas the author's work is more focussed towards families of SAT problems. To relate the modified minisat based gym environment, the authors refer to the Wang and Rompf[11], who modified the MiniSat SAT solver which uses VSIDS branching heuristics to create a gym environment and they employed CNN based DQN and Alpha Go model similar to the ones available in OpenAI Baselines, accepting a sparse matrix of variable and clauses as input and exploiting the models to make the branching decision and converting whole SAT solving problem into a game playing problem using reinforcement learning. Their work is similar to the author's work in the sense that the author also used a sort of minisat gym environment and converted the SAT solving in a typical reinforcement learning based game playing problem. The authors pointed that in Wang and Rompf[11], DQN fails to generalize to 20-91 3SAT problems because of using CNN which is not invariant to variable or clause permutation. The authors claimed in their work that using GNN as DQN agent scales well for problems with different sizes.

The authors also mention about Khalil et al. [12] who uses DQN and GNN to learn greedy meta-algortihm as a heuristic algorithm for solving graph optimization problem. Though the approach looks similar to Graph-Q-Sat, the authors have pointed out that it differs from Graph-Q-Sat in the sense that it creates an entire heuristic algorithm whereas Graph-Q-Sat is just a branching heuristic used on top of VSIDS based CDCL SAT solver.

There is a slight bias that is observed in the related work section of the original paper. For the SAT solving related papers, the authors have mentioned only those papers that address only small SAT problems and haven't explored the large industrial SAT problems. It is fine since the authors have indicated that more work is required to address the large industry SAT problems involving thousands of variables and millions of clauses.

Overall the authors have given the due credits to the prior work in relating it to their proposed approach. The authors have also pointed out the shortcomings of the related papers and claimed to address them using Graph-Q-Sat in section 1.1.2.

1.3 Technical results

1.3.1 Technical Details

The authors have introduced Graph-Q-SAT as a branching heuristic for the SAT solvers. Some initial decisions or iterations of variable selection and assignment are taken by a trained Graph-Q-SAT and then the control is given back to the Minisat solver to solve rest of the problem. Since minisat is a complete solver(one which eventually solves boolean satisfiability problem), using Graph-Q-SAT as a branching heuristic over minisat results in a overall complete solver. Graph-Q-SAT is used in the early exploration of the search space.

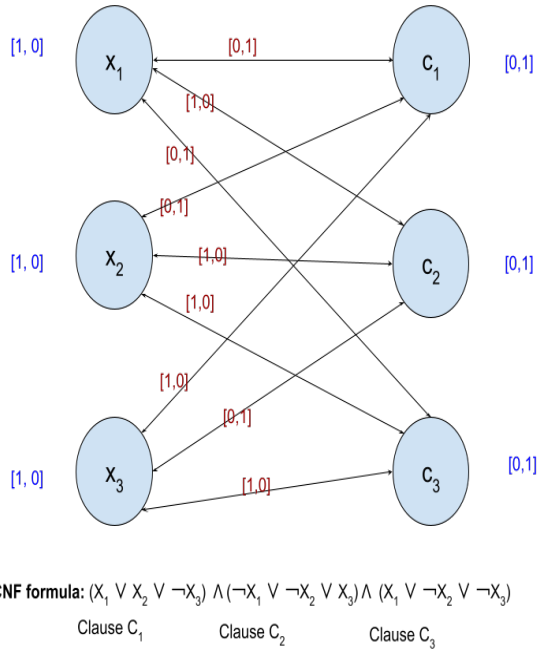
In the proposed approach, a SAT problem is being treated as an MDP which is sampled from a distribution of SAT problems of a specific family(e.g. Random 3-SAT or graph coloring). Each problem is either satisfiable or unsatisfiable. A task(an instance of SAT problem) is defined as $\tau \sim D(\phi, (un)SAT, n_{vars}, n_{clauses})$, where D is the distribution of SAT problems with ϕ defining the class family(e.g. Random 3-SAT, flat-graph-coloring), second argument tells whether it is satisfiable or not , n_{vars} and $n_{clauses}$ are the number of variables and number of clauses respectively. A satisfying assignment is a terminal state. Moreover when all the assignments have been exhausted or the environment learns that no assignment can lead to satisfying result at any stage(in case of CDCL, example if at level 0 a conflict arises), it can lead to a terminal state. Therefore, this MDP is episodic. A state of a problem is created from its currently unassigned variables and unsatisfied clauses in CNF. An interesting point of this paper is the conversion of this boolean expression in CNF to graph representation. A graph vertex is created for every variable and clause in the boolean expression instance. The directed edges are created between the variable vertex and the clause vertex,

two directed edges from vertex to clause and vice-versa if the variable belongs to the clause. The edge between a vertex x_1 and clause c_1 is a two-dimensional vector. if x_1 is present in the negated form in c_1, then the value of this feature vector is [1,0] and [0,1] otherwise. The vertex features are two-dimensional feature vectors as well. For a variable, the value of this feature vector is [1,0] and for a clause the value is [0,1]. The global attribute is empty and only used for message passing. Figure 1 shows the example of a graph representation of a boolean expression given in CNF.

In this work, the authors have used the minisat gym environment of [11]. The authors modified this environment to make it support arbitrary SAT problems and generate a graph representation of the state. The environment accepts a propositional formula problem in graphical representation and an action(unassigned variable-polarity decision), performs assignment, unit propagations, updates implication graph, learns new clauses in case of conflicts and backtracks and returns the updated propositional formula with learned clause which is the next state, a negative reward if problem is not solved and 0 otherwise, and a boolean variable indicating whether the SAT problem has been solved or not. If an action equal to -1 is given to the environment, the environment gives control to the minisat agent to take further actions.

The Graph-Q-SAT is a reinforcement learning agent which is trained in a DQN[9] setting discussed in the section 1.1.1. Graph-Q-SAT uses GNN as the action value(q-value) function approximation. The Graph-Q-SAT accepts a graphical representation of the instance of the SAT problem. The authors have used encode-process-decode architecture[7] which comprises of three GNN models - encoder, core and decoder. The encoder and decoder are independent GNN i.e. there is no message passing involved. Figure 2 and 4 shows the internal structure of encoder and decoder GNN. Both encoder and decoder GNN comprises of three independent perceptrons(no weight sharing, different parameters). Encoder encodes every 2-dimentional vertex and edge feature vector to 32 dimensional updated vertex and edge feature vectors which is called encoder output. In figure 2 and 3, ϕ_u, ϕ_v and ϕ_e denotes perceptron used for encoding global attribute, every vertex and every edge features respectively. The core GNN is a full GNN block[7] which performs message passing and updates vertex, edge and global features using the equations 1,2 and 3. This is one message passing iteration. The output of the core is concatenated with the output of the encoder and recurrently passed to the core again. Figure 4 shows the internal structure of the core GNN. The red circles in the core diagram denotes the concatenation operation. ρ denotes aggregation operation(examples sum, max, min, average etc) and ϕ denotes the perceptrons consistent with equations (1),(2) and (3)(slight change in subscript and superscipt but ϕ and ρ are same). After all message passing iterations are completed the core output is sent to the decoder. The decoder output for each vertex is transformed by a perceptron to a two-dimentional vector. These final feature vectors for all variable vertices are extracted and for each variable x, the output vector [qx_1, qx_2] denotes the q-value for assigning polarity True to variable x(qx_1) and False(qx_2). This is also summarized in Algorithm 1. When an agent is in the evaluation mode, it finds the variable and the polarity corresponding to the maximum q-value and return this variable-polarity decision as the next action to the environment. The additional details about the encoder, core and decoder GNN are given in appendix, table 5[1].

The algorithm 2[1] provides the details of training Graph-Q-SAT. For a graphical representation of CNF boolean formula state, all the vertex features are stacked together as vertices vector and all the edge features are stacked together as edge vector. Minibatch gradient descent has been used to train GNN parameters of DQN using temporal loss function discussed in section 1.1.1. A batch of different graphs is created by treating different graphs as a single complete graph consisting of more than one connected components and passing appropriate indices for the vertex and edges of each graph in the stacked vertex and edge vectors. The GNN parameters in DQN are randomly initialized and DQN is trained for *totalBatchUpdates* tracked by updates. The complete training consists of episodes of interaction between Graph-Q-SAT agent and the environment. Every episode begins by resetting the environment by calling env.reset() which samples an instance of problem from training set. The episode ends when environment returns *done* as true which means the problem has been solved. Each Graph-Q-SAT and environment interaction is called environment step and tracked by *totalEnvSteps*. Within an episode, the Graph-Q-SAT agent uses epsilon greedy policy to select an action(with probability 1-ϵ, pass the state to the GNN and compute next action using algorithm 1) and receives new state and reward in return. This transition is stored in an experience replay buffer. The batch updates are performed after every *updateFreq* environment steps and the Graph-Q-SAT agent samples a batch of transition from this replay buffer to perform GNN parameters update in DQN. After *validateFreq*, Graph-Q-SAT agent is used to solve the problems from a validation set



*

Figure 1: Graphical representation of CNF formula

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 $VarIndex, VarPolarity$;

and the number of environment steps are recorded which are then compared with the number of environment steps took only using minisat. From the validation data scores, the best model is selected and evaluated on test data. For more information about the training hyperparamters, refer to appendix, table 5[1]. Note that during evaluation, the number of decisions taken by Graph-Q-SAT are capped but the authors have not done the same during training.

1.3.2 Results

Overall the authors have been able to support all their claims using diverse set of experiments in [1]. Referring to the experimental results in the original paper, a trained Graph-Q-SAT on SAT50_218 datasetfrom SATLIB benchmark[13] does achieve 2-3 times improvement in iterations over the minisat on both SAT and unSAT problems from the same family of Random Uniform 3-SAT problems. The drastic reduction in iterations over the unSAT problems using the model trained solely on the satisfiable instances strongly claims that the Graph-Q-SAT is highly generalizable from SAT to unSAT and also to problems involving larger number of variables and clauses. Moreover the same model also achieved around 1.5 times reduction in iterations on some of the flat-graph-coloring tasks especially the ones which have lesser variables and clauses with demostrates positive zero-shot generalization of Graph-Q-SAT. The authors demonstrated that a model trained on the flat coloring

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 $\mathcal{S}_{test}$;

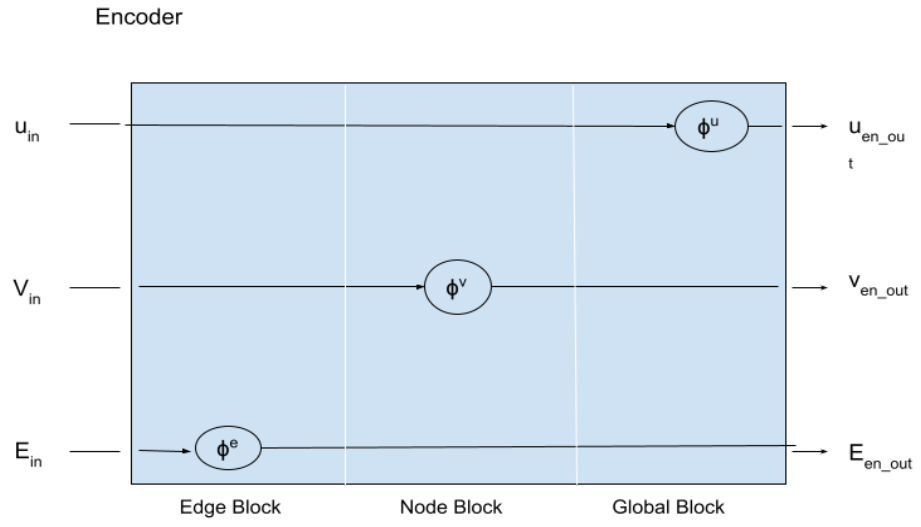


Figure 2: Encoder GNN Internal Structure

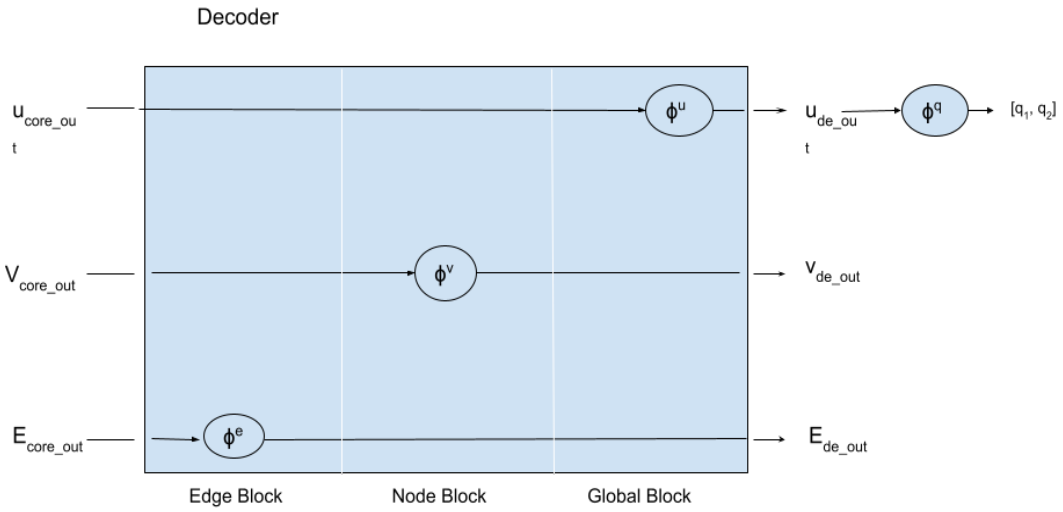


Figure 3: Decoder GNN Internal Structure

datasets results in better reduction in the iterations as compared to the transferred model on the flat graph coloring benchmarks giving more than twice reduction in iterations on three benchmarks. The authors trained different Graph-Q-SAT models by varying the size of the training SAT50-218 datasets and reported the iteration improvements for each model for different Random 3-SAT and unSAT datasets from SATLIB benchmarks. It is observed that by training the model with even one training instance results in twice the iterations reduction over minisat in some cases which buttress the data efficacy of the author's approach. The authors have also presented the plots of mean assignment changes per step with the problem ID which shows that Graph-Q-SAT results in more variable changes per step as compared to minisat and which is responsible for the fast searching in search space and high reduction of iterations. The authors have also shown the reduction in the wallclock time by plotting the wallclock time for SAT250-1065 and unSAT250-1065 which shows the effectiveness of Graph-Q-SAT branching heuristic in reducing the overall wallclock time. Refer to appendix in [1] for details on the experimental settings.

1.4 Review

Overall the authors have established a proper relationship between the prior related work and their work. The authors have also backed all their claims about their approach using an extensive set of experiments which completely support their claims. The authors have acknowledged the limitation of their approach in application to large SAT problems that are encountered in industry and competing with branch heuristic in the industry setting. They also acknowledged that increasing the number of decision by Graph-Q-SAT takes model evaluation time which could overall increase the wall clock time. The authors have mentioned the future research directions in this regard. The authors have clearly established the relationship of their ideas in developing this paper to the prior work in related work section (which can be used to estimate the novelty of their approach), pointed out their limitations and addressed them in this paper with strong experimental results.

The authors have been very succinct in mentioning the models and the algorithm used in Graph-Q-SAT which makes it difficult for readers to comprehend the finer details of the overall algorithm and components used in their approach. For example, the authors have not mentioned clearly the internal structures of the Graph Neural Networks used in the papers and just gave the reference to the original paper. Almost every paper that uses Graph Neural Networks, makes modifications to the approach mentioned in [7]. Moreover the overall branching heuristic algorithm has been mentioned in Appendix which should ideally be mentioned in the main paper and explained in detail. Moreover, the authors could have tried to address the large SAT problems such as the ones which have few

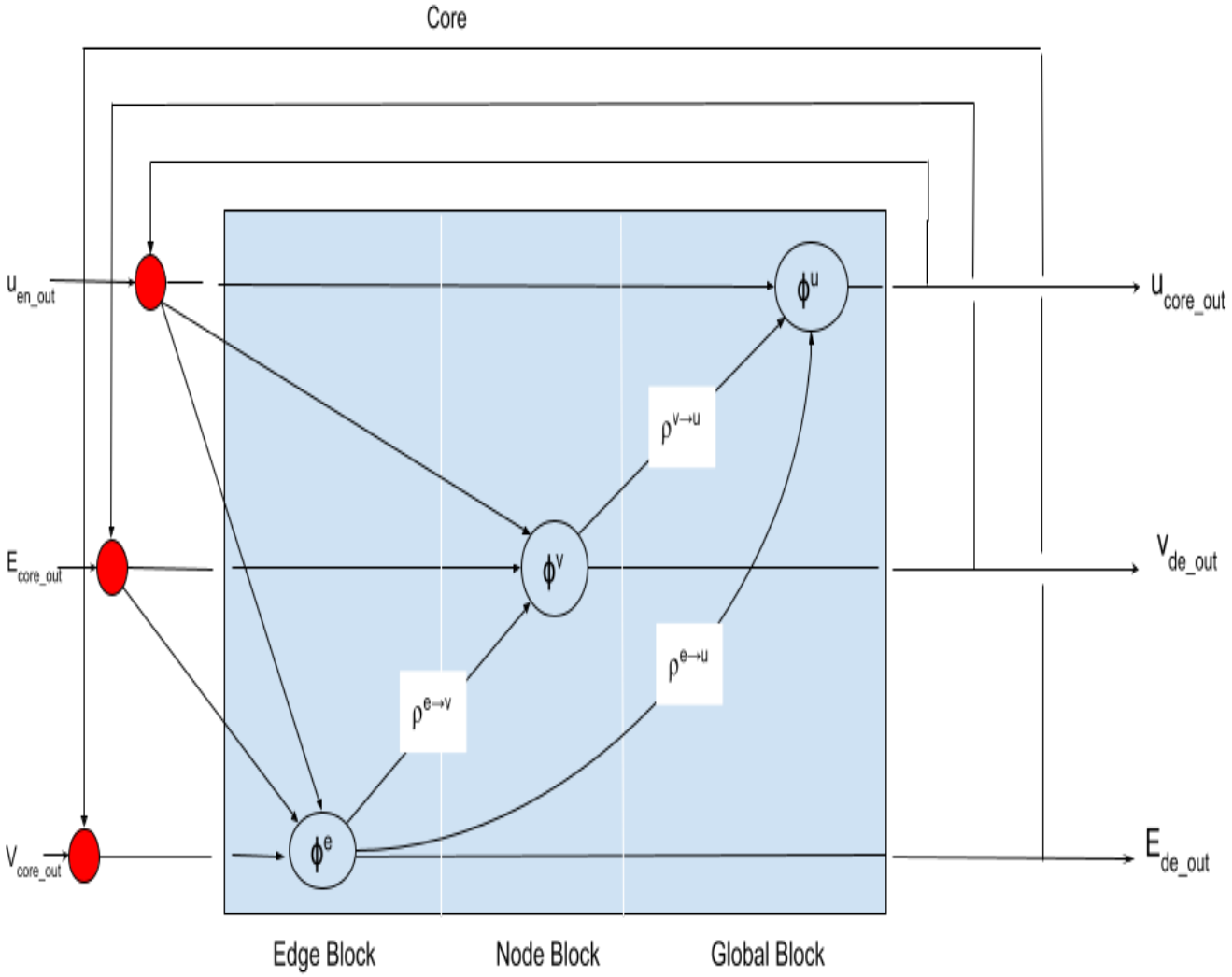


Figure 4: Core GNN Internal Structure

thousand of variables and how Graph-Q-SAT branching heuristic could deal with it or some kind of experimental results so that a reader could estimate where Graph-Q-SAT heuristic stand in dealing with industry SAT problems.

2 Experiments

This section comprises of various experiments to verify the different claims of the original paper. In this section, the enlisted experiments are similar to the ones given in the original paper. Similar experimental parameters mentioned in appendix of [1] are used in these experiments and if there are changes, then they are clearly mentioned. These experiments are performed by modifying the code released by the authors according to the requirements of different experiments. Kindly note that the authors only released the basic code and no trained models were released. The code was highly undocumented. The released code didn't have the appropriate framework to carry out all experiments so appropriate frameworks are constructed. Moreover, the authors didn't provide a lot of experimental details required to reproduce the results of the original paper such as strategy to split the train, validation and test data which in turn affects the choice of model used to generate

results for different experiments. This could make the experimental results in this paper different from the ones reported by the author. Since all these experiments involve training and evaluating deep models, I haven't been able to perform all sets of experiments mentioned in the original papers due to lack of continuous access to computing resources such as GPU although these experiments are performed using Google Colab which has its own usage limitations. Just like in original paper, SATLIB[13] benchmarks are used for training, validation and evaluation consisting of Random-3SAT problems(SAT50-218, unSAT50-218 etc) and flat graph coloring problems. All the instances in flat graph coloring datasets are satisfiable.

In most of these experiments, the performance of trained model is reported in terms of Median Relative Iteration Reduction(MIMR) i.e. number of iterations taken by minisat to solve the problem divided by number of iterations taken by Graph-Q-SAT branching heuristic equipped Minisat solver.

2.1 Improvement upon VSIDS

This subsection verifies the claim that Graph-Q-SAT can reduce the number of iterations(decisions) by 2-3 times as compared to VSIDS based CDCL solver used by Minisat. Graph-Q-SAT agent is trained on 800 SAT instances from SAT-50-218 dataset by sequentially using 800 first instances for training, next 100 for validation and last 100 for evalution. Similar splitting has been done in other datasets as well wherever required. The number of iterations taken by minisat agent with no restarts to evaluate different datasets are given in Table 1. The evaluation datasets mentioned in the table 1 have 100 instances each which model hasn't seen during training. The Graph-Q-SAT MRIR is reported for the same datasets in Table 2.

dataset	median	mean
SAT 50-218	34.5	38.65
SAT 100-430	231.5	284.73
unSAT 50-218	67.5	67.7
unSAT 100-430	587	596.39

Table 1: MiniSat iterations(without restarts)

dataset	median	mean	max	min
SAT 50-218	2.02	2.90	14.16	0.29
SAT 100-430	2.85	4.99	54.06	0.33
SAT 250-1065	0.613	2.394	53.34	0.008
unSAT 50-218	1.97	2.20	5.86	0.81
unSAT 100-430	1.22	1.31	2.88	0.56

Table 2: Graph-Q-SAT MRIR trained on SAT50-218 (without restarts)

2.2 Generalization properties of Graph-Q-SAT

In this subsection, we explore how a model trained on SAT-50-218 instances perform on tasks sampled from distribution different from the distribution of its training instances. We will also observe the how a model generalizes over tasks of different problem sizes belonging to the family of training dataset.

2.2.1 Transfer to higher problem sizes

We can observe around 3-4 times improvements in iterations on random 3-SAT datasets. The iterations reduction on the SAT100-430 dataset is around 5 times which is even better than the SAT50-218 dataset. This demonstrates the generalization power of Graph-Q-SAT. Around 2.4 times iteration reduction is observed on the SAT250-1065 dataset as well.

2.2.2 Transfer to unSAT

We can see the results of evaluating a model trained on 800 instances from SAT-50-218 on the unSAT datasets in table 2. Around 1-2 times improvement over Random-3 unSAT datasets is observed which is still better than other prior deep learning approaches such as NeuroSAT that do not work on the unSAT instances if trained only on SAT instances. This demonstartes the generalization of

Graph-Q-SAT from SAT to unSAT. Performance drop is observed with increasing number of variables and clauses in unSAT datasets similar to what was observed in the original paper.

2.2.3 Transfer across task families - flat graph coloring tasks

In this section, we evaluate a model trained on 800 instances from SAT-50-218 on the flat-coloring datasets. The MIMR results are presented in table 3. Similar to the original paper, Graph-Q-SAT has been able to improve the iterations in some flat graph coloring datasets which demonstrates the generalizability of Graph-Q-SAT over different task families. Similar to the original paper, for comparison, Graph-Q-SAT is trained solely on the flat-75-180 dataset and validated on flat100-239 dataset. Results are presented in Table 4. The MRIR is computed with restarts similar to the original paper. It is observed that this trained model acheives 2-3 times iterations reduction on the flat-graph-coloring benchmarks which is quite better than what was reported in the original paper.

dataset	median	mean	max	min
30-60	1.6	1.614	5.5,	0.148
75-180	0.508	1.388	11.0,	0.096
100-239	0.459	1.177	13.125,	0.095
150-360	0.600	0.680	4.204,	0.053
175-417	0.507	0.949	19.785,	0.065

Table 3: GraphQSAT MRIR trained on SAT50-218 (without restarts) on flat coloring problems, MRIR calculated using with_restarts

dataset	median	mean	max	min
75-180	2.4	2.951	9.625	0.761
100-239	2.491	3.188	12.1875	0.2711
30-60	1.585	1.797	5.5	0.470
150-360	1.596	2.750	23.0	0.134
175-417	0.998	2.759	22.736	0.059

Table 4: GraphQSAT MRIR trained on flat 75-180 and validated on flat100-239 (without restarts) on flat coloring problems, MRIR calculated using with_restarts

2.3 Data Efficiency

In this subsection, we explore how the size of training set affect the model's performance. This experiment is performed to verify the author's claim on data efficiency. We train Graph-Q-SAT on different number of SAT-50-218 instances and evaluate it on Random-3 SAT and unSAT SATLIB benchmarks. The results are plotted in Figure 5. It can be observed that even using a single training instance, the trained model is able to reduce the iterations by 2 times. A peak in the iteration improvement is observed for the training set size of 50 for all the tasks which is quite difficult to explain. This result is nearly similar to the one we observed in the original paper as well.

2.4 Capping maximum number of model evaluations

This experiment does not deal with any claim made by the author. This experiment is conducted so as to explore how limiting the number of model evaluations i.e. capping the number of decisions Graph-Q-SAT could take before passing control to the VSIDS affects the iteration improvement. A Graph-Q-SAT model is trained on the 800 insatnces of SAT50-218 dataset and the evaluation is performed on the SAT50-218, SAT100-430, unSAT50-218, unSAT100-430 datasets by using different values of maximum decisions allowed for Graph-Q-SAT. The results are plotted in figure 6. We can observe that after 300 iterations there isn't any improvement in the iterations. This might be due to the fact we are performing experiments on the relatively smaller datasets. On the other hand, even if Graph-Q-SAT is allowed to take 10 decisions, one can observe 1.5-4 times improvement in the iterations. Since we are aware of the fact that model evaluation takes time(especially if there's no GPU available), such curves can be studied to find the maximum number of decisions to allow Graph-Q-SAT to take so as to save computational time in model evaluations.

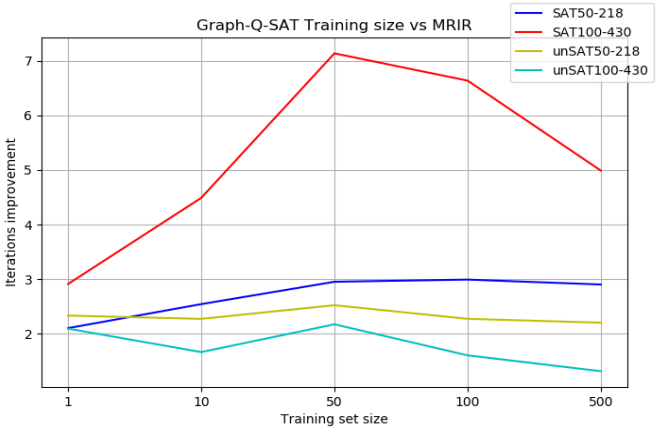


Figure 5

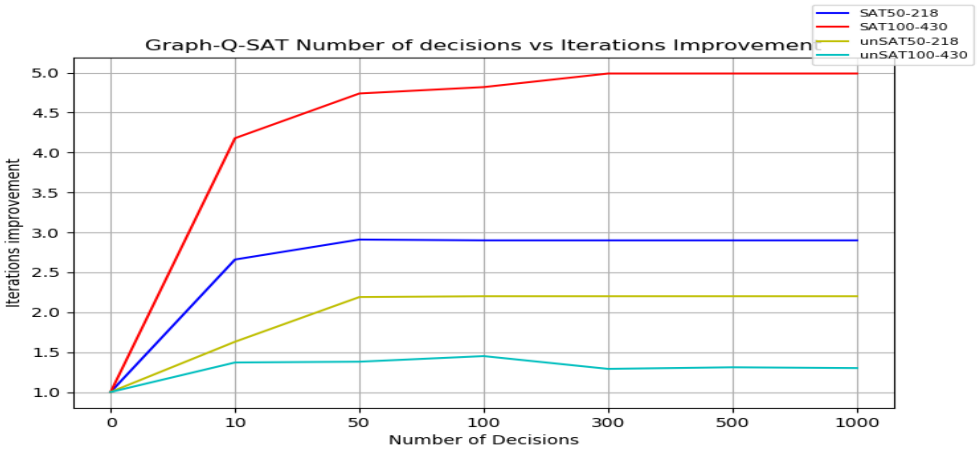


Figure 6

3 Extension

In this section, we explore the generalization properties of training Graph-Q-SAT on 800 instances of unSAT 50-218 dataset. This experiment was not present in the original paper and it has different direction since unSAT problems have different structures as compared to SAT problems or satiusfiable flat graph coloring datasets. The trained model was evaluated on different random-3 SAT and unSAT test datasets. The results are reported in the table. From table 6, we can observe that it is surprising that Graph-Q-SAT trained on unSAT instances performed better on SAT dataset as compared to unSAT dataset. The average iteration reduction on SAT100-430 dataset is around 3.7 times whereas the its 2.5 times on the unSAT datasets. This show higher generalization behaviour of training on unSAT instances as compared to SAT instances. This behaviour is quite opposite to what we observe when we performed evaluation on unSAT dataset using Graph-Q-SAT trained on SAT50-218 dataset where the performance was relatively lower on unSAT. This further gives the motivation to explore how the trained model behaves differently when trained on SAT and unSAT instances and if such models can be transfered over other tasks.

dataset	median	mean	max	min
SAT 50-218	1.850	2.525	12.428	0.288
SAT 100-430	2.779	3.787	27.029	0.313
unSAT 50-218	2.331	2.523	7.285	1.159
unSAT 100-430	2.395	2.525	4.858	1.317

Table 5: GraphQSAT MRIR trained on unSAT50-218 (without restarts)

References

[1] Kurin, Vitaly and Godil, Saad and Whiteson, Shimon and Catanzaro, Bryan *Can Q-Learning with Graph Networks Learn a Generalizable Branching Heuristic for a SAT Solver?* Advances in Neural Information Processing Systems 32, 2020

[2] 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.

[3] 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

[4] 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.

[5] 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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.