Elimination Mechanism of Glue Variables for Solving SAT Problems in Linguistics

Ziwei Zhang, Beijing University of Posts and Telecommunications, China
Yang Zhang, Beijing University of Posts and Telecommunications, China

The Asian Conference on Language 2021
Official Conference Proceedings

Abstract
We propose GVE(Glue Variables Elimination), an algorithm that organically combines neural networks with a deterministic solver to solve SAT(Boolean satisfiability problem) in the field of linguistics. It gives full play to their respective advantages by following steps: (a) applying a graph learning algorithm to learn the structure of the CNF formula; (b) finding the glue variables of the problem; (c) determining their values; (d) simplifying the original formula; (e) using a deterministic solver to solve the simplified problem. We use SATCOMP 2003-2019 benchmarks as the test data sets, and compare our model with the SAT solver CADICAL that has performed well in SATCOMP 2019 as well as the neural network model PDP proposed in recent years. GVE model shows good performance. As the complexity of the problem increases, the solution time can be about 20%-95% quicker than the deterministic solver, while at the same time around 72% more accurate than PDP model.

Keywords: Linguistics, Boolean Satisfiability Problem, Graph Learning, Survey Propagation, Reinforcement Learning, Glue Variables

Introduction

In industrial production, many questions can be converted into satisfiability questions (Constraint Satisfaction Problems (CSP)(Kumar, 2015)), especially Boolean satisfaction problem(SAT). Boolean Satisfiability, in particular, is the most fundamental NP-complete (Garey & Johnson, 1979) problem in computer science with a wide range of applications in various areas(Biere, Heule & van Maaren, 2009; Knuth, 1997). There is no deterministic Turing algorithm to solve the SAT problem in polynomial complexity(Van Leeuwen, 1991). However, SAT problems are inevitable in practical application, for example, in the fields of engineering technology, complexity theory(Karp, 1972; Aho & Hopcroft, 1974), military, artificial intelligence(Vizel, Weissenbacher & Malik, 2015), concurrency control, transportation, intelligent traffic control and so on. From the point of view of problem solving, complete algorithm, represented by Davis-Putnam proposes based on the recollection search(Nieuwenhuis, Oliveras & Tinelli, 2005; Zhang & Malik, 2002), can solve both satisfiability problems and unsatisfiable problems, though it can take a long time.

Machine Learning has been used for different aspects of CSP and SAT solving from branch prediction(Liang, Ganesh, Poupart & Czarnecki, 2016) algorithm to hyper-parameter selection(Xu, Hutter, Hoos & Leyton-Brown, 2008). These algorithms all encode the input SAT instances to different degrees while interconnectedness of the CNF formula exists in its original structure. There is a serious risk that those feature extraction techniques will lose the hidden information of the formula during procedure of data processing. For instance, frameworks such as Graph Neural Network(Li, Tarlow, Brockschmidt & Zemel, 2015), PossibleWorldNets(Evans, Saxton, Amos, Kohli & Grefenstette, 2018), NeuroSAT framework(Selsam et al., 2018), the Circuit-SAT framework(Amizadeh, Matusevych & Weimer, 2018) and Recurrent Relational Networks for Sudoku(Palm, Paquet & Winther, 2018) et al. have been quite successful in capturing the inherent structure of the SAT instances by embedding them into traditional vector spaces that are suitable for Machine Learning models. However, they don't have a persuasive explanation why their networks effective and how they work. Most importantly, their accuracy has a sharp decline as the number of variables or complexity of the SAT problems increases. Furthermore, the previous researches of solving SAT problems that use pure neural network (e.g. PDP(Amizadeh, Matusevych & Weimer, 2019)), show a decline trend of performance with the increment of problem scale, most of which are limited by the structures of CNF formulas. The uncertainty of their results also make them have little practical value.

Our contributions are as follows:
(a) The method of graph learning and representation proposed by Hamilton, Ying and Leskovec(2017) is used for the sake of learning internal features of SAT problems. Different from the previous work, which only has one kind of node in the graph, our graph has two kinds of nodes. We see the CNF formula as a bipartite graph with literals and clauses as the two type of nodes, and train a set of aggregator functions based on Random Walk(Spitzer, 2013) and Graph Convolution Network(Geng et al., 2019) that learn to aggregate feature information from a node's local neighborhood that contains both variable nodes and clause nodes. In this way, we are able to adapt to the different CNF structures and train a general solver to solve all kinds of questions which can significantly improve efficiency.

(b) Considering that the performance of neural model is limited by the complexity of the problem, we do not attempt to solve the whole question, but rather focus on finding out glue variables and their values -- those likely to occur in glue clauses(Audemard & Simon, 2009),

a type of conflict clauses known to be extremely important to the reasoning of modern CDCL(Heule, Kullmann, Wieringa & Biere, 2011) SAT solvers as a reinforcement task and then solve them through a Survey Propagation Algorithm idea based neural network. Finally we use the complete solver CADICAL (QUEUE, 2019) to solve the simplified formula. Furthermore, our framework is designed in unsupervised manner that allows GVE to be trained without training data markup.

(c) Unlike previous works only give a answer either “satisfiable” or “unsatisfiable”, for the UNSAT CNF, we not only show the result, but also the *UnsatCore* -- the combination of clauses whose subsets are still unsatisfiable, is derived as a proof of the answer.

We evaluate our algorithm on thousand near CNF instances that chosen from SATCOMP 2013-2019 benchmarks. Our experimental results show the superiority of the GVE framework compared to both neural and classical solvers.

Preliminaries

A propositional logic formula, also called Boolean expression, is built from variables, operators AND (conjunction, also denoted by $\wedge$), OR (disjunction, $\vee$), NOT (negation, $\neg$), and parentheses. The SAT problem is to check whether a given formula is satisfiable or not. If a variable assignment exists which can make every clause TRUE, the formula is called satisfiable. On the other hand, if no such assignment exists, the function expressed by the formula is FALSE for all possible variable assignments and the SAT problem is unsatisfiable. A disjunctive expression of a finite number of variables is called a clause. A formula is in conjunctive normal form (CNF) if it is a conjunction of clauses (or a single clause).

1.1. Solving Algorithms Bases

Glue levels and glue variables *Glue variable* is an important concept in CDCL algorithm (Conflict-Driven Clause Learning) proposed by Heule et al.(2011). Here we introduce the concept of *unit propagation*(Nieuwenhuis et al., 2005): given a clause $C = l_1 \vee l_2 ... \vee l_n$, if all literals but l_n are set to 0, then C is equivalent to the unit clause (l_n), and the value of l_n is forced to 1. *Glue level* counts the number of decision levels involved in the clause. A clause with low glue level requires fewer decisions to become unit. Variables in clauses with glue level≤2 are called *glue variables*. Finding glue variables had greatly improved the performance of existing technologies and has now become a standard practice.

SP Algorithm

SP (Survey Propagation) (Braunstein, Mézard & Zecchina, 2005; Mezard & Montanari, 2009) algorithm is a typical incomplete algorithm of which principle is derived from the theory of the spin glass system (Sherrington & Kirkpatrick, 1975) in statistical physics. SP applies a message passing technology to pass the probability of the variable to make the clause it located unsatisfiable, and then uses the investigation iteration function to repeatedly calculate the message passed in the factor graph. If this process converges, a set of marginal probability distributions of the values of all variables will be finally obtained. Through the probability distribution, the possible value of each variable can be known. Then the SP algorithm uses the Walksat(Hoos, 2002) algorithm to fix one or some variables to simplify the original problem. Walksat is an extension of Random Walk. Their fundamental idea is to randomly generate an initial assignment, and then select a variable to flip under certain conditions until

the maximum number of flips is reached or a solution is found. If the problem is satisfiable, a set of solutions can be finally obtained after repeating the above process.

1.2. Graph Representation

Figure 1 shows the graph representation model of clauses and variables. Circles represent variable nodes, and squares represent clause nodes. There are two types of edges, which represent the sign of the variable in the clause. This graph model can establish the adjacency matrix between variables and clauses, and can intuitively reflect which clauses a certain variable is related to. For example, clause a has three variables x_1, x_2, x_4, so the origin formula can be represent as $a = x_1 \vee x_2 \vee \overline{x_4}$. In the same way, the formula F in the figure can be restored to the original mathematical expression:

$$F = \{(x_1 \vee x_2 \vee \overline{x_4}), (x_3 \vee x_4), (\overline{x_1} \vee x_5)\} = \{a, b, c\}$$

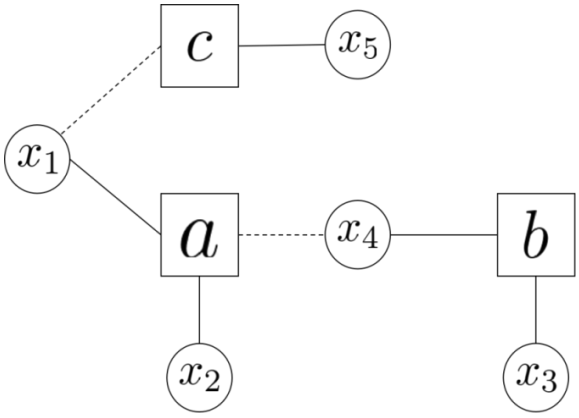
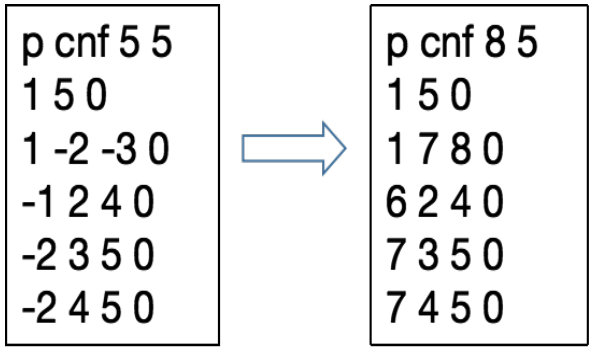


Figure 1: Graph Representation Structure of Clauses and Variables



(a) (b)

Figure 2: A CNF Conversion Example

In order to ignore the signs in the original formula during training, we split the variables into positive and negative terms, and regard the negative form of each variable as a new variable. Figure 2 shows a CNF conversion example intuitively. As for training, for a CNF formula with M clauses and N variables, it will eventually be expressed as an $m \times n$ 0-1 matrix $G^{+}_{<\chi,\varepsilon>}(G^{+})$ which contains all positive variables and $m \times n$ matrix $G^{-}_{<\chi,\varepsilon>}(G^{-})$ which contains their negations.

Instead of training a distinct embedding vector for each node as traditional graph representation methods do, we train a set of aggregator functions that learn to aggregate feature information from a node's local neighborhood. Each aggregator function aggregates information from a different number of hops, or search depth, away from a given node.

2. Network Architecture

Since it is difficult to solve the difficult SAT problems by using pure neural networks, we introduce the idea of model compound -- the combination of neural networks and a deterministic solver. The key to solve the SAT problem is to find out the key variables in CNF so as to simplify the original formula. When interacting with the deterministic solver, the model continuously modifies the prediction results of key variables through Reinforcement Learning. Meanwhile, to adapt the solver to different structures of CNFs and

find glue variables of SAT problems, we design a suitable network pattern that could solve CNFs via neural networks. Figure 3 shows the overall flow chart of our algorithm. We will describe the steps that need to take to reach the ultimate goal in this chapter .

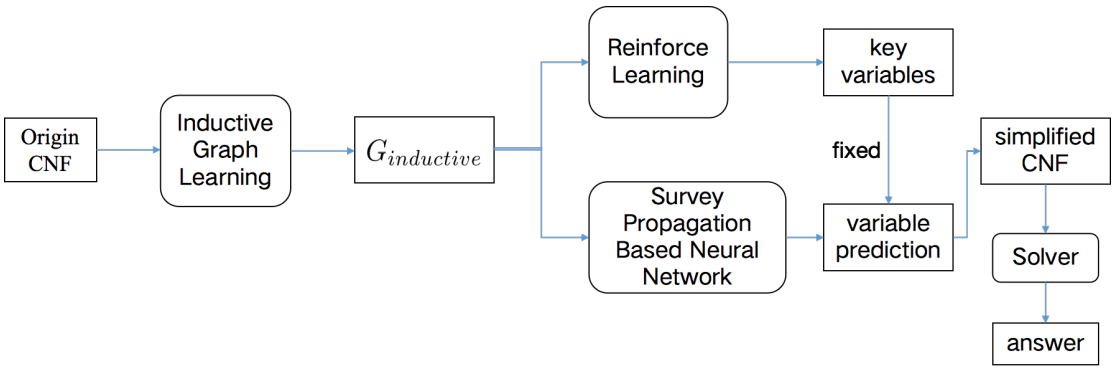


Figure 3: Overall Flow of the GVE Algorithm

The CNF formula is transformed into a bipartite graph $G(\chi, \epsilon)$ where χ and ϵ are a collection of variables and Boolean symbols, respectively. There are two types of nodes representing variables and clauses in graph G. The Boolean symbol ϵ_{ij} where $\{\epsilon_{ij} \in \epsilon, 1 \leq i \leq M, 1 \leq j \leq N\}$ connecting the j_{th} clause and the variable x_i indicate that the clause contains the variable.

Based on the above, for a SAT problem with M clauses, we define a way of measuring model as

$$W(G) = \frac{1}{M} \prod_{f=1}^{M} L_f(G_{\partial f}, V_{\partial f}) \tag{1}$$

where G to represent the graph structure, that is, the combination of the output result of the previous step and the corresponding graph structure of the original CNF formula. $V_{\partial f}$ is the variable embedding results that related to some clause f. L is an abstract representation of our entire network structure which contains graph representation layer, glue variables prediction layer and variables value prediction layer.

2.1. The Inductive Graph Learning

We adopts an unsupervised training method, assuming that each variable takes a value of true to determine the label of the clause, that is, if all the literals of the clause are negative, the clause is negative. A group of aggregator functions is trained to aggregate feature information from adjacent nodes list of a node from which we randomly sample in the process of graph learning. In our model, clauses and variables are used as learning nodes respectively. ***The list of adjacent nodes is the set of all clauses that are related to the variables in the clause, and the set of all the variables that appear in the same clause with a certain variable.*** In the figure below, orange circles represent clause nodes, and blue circles represent variable nodes. Take node 1 as an example, it will aggregate the characteristics of all adjacent nodes, and the arrow represents its source of information. In summary, the adjacent list of variable 1 contains clause nodes a, b and variable nodes 2,3,4,5. Thus, we complete the bidirectional transmission of feature information. Figure 5 indicates the binary encoding method of a CNF. As shown, the left and right half of Figure 5(b) represent V^+ and V^-, respectively.

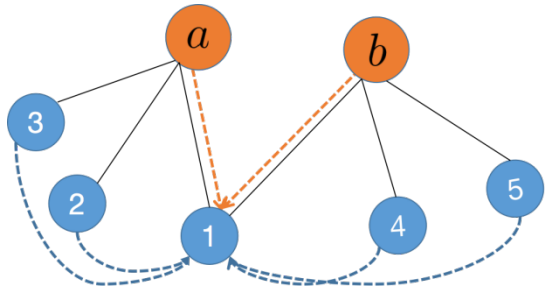
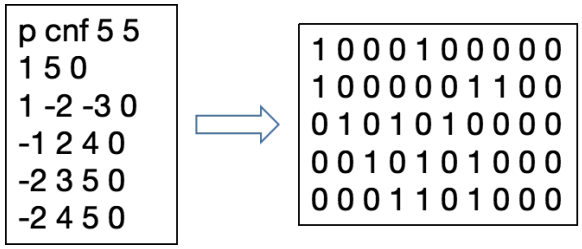


Figure 4: Schematic Diagram of the Two Sources of Feature Aggregation



(a) (b)

Figure 5: Binary Encoding Example of a CNF

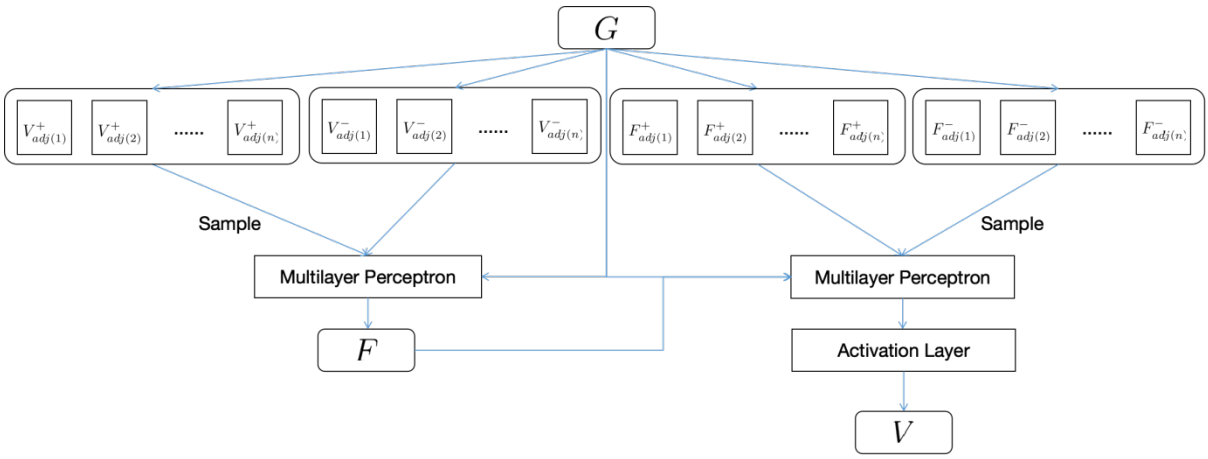


Figure 6: Architecture Diagram of Graph Aggregation

Figure 6 shows the architecture diagram of graph aggregation. For each variable v there will be a list v_{adj} "adjacent" to it, which contains all variables that have a direct or indirect relationship with it. Therefore, according to the aforementioned classification, the adjacent list is also expressed as V^{+}_{adj} and V^{-}_{adj}. We encode them so that each variable can fully learn the characteristics of the variables related to it. We use the formula below to update V^{+} and V^{-}

$$V = \Gamma_{\kappa}(G, V, V_{adj}) \tag{2}$$

where Γ_{κ} is a multi-layer linear neural network with κ as the parameter. Since the graph transformed by CNF is a bipartite graph with two kinds of nodes, we need to aggregate again

$$V = \Gamma_{\rho}(G, V, F_{adj}) \tag{3}$$

where F_{adj} represents the list of clauses that contains the variable -- that is, all the clauses in which the variable appears and ρ is a learnable parameter. Take the CNF formula in figure 2(a) as an example, variable 1 corresponds to clause 1, 2. In this way, we get the updated V^{+} and V^{-}, and finally stitch them together to get the summarized graph structure

$$V = \Upsilon_{\alpha}(concat(V^{+},\ V^{-})) \tag{4}$$

where Υ is the activation layer with α as the parameter.

2.2. The Survey-Propagation Based Neural Model

After the steps of graph learning, we can be compatible with different CNF structures. Next, we designed a neural network based on the idea of Survey Propagation algorithm so that messages are passed between clauses and variables. For the convenience of representation, we define matrix $E_v \in \mathbb{R}^{e\times 2\cdot n}$, that is, $E_{ij} = 1$ if there is an edge from variable x_j to some clause c. Likewise, $E_f \in \mathbb{R}^{e\times m}$, which represents the relationship from clauses to edges.

$$F_{x\to c} = \Gamma_\delta(V, E_v, G) \tag{5}$$
$$V_{c\to x} = \Psi_\beta(\Gamma_\eta(V, E_f, G)) \tag{6}$$

where E_v and E_f are matrices representing the relationship among edges, variables and clauses, and Γ_δ, Γ_η are linear neural networks parameterized by different vectors δ and η. Γ_δ and Γ_η are designed for calculating the value of "survey information" transmitted to each other. Notably, we have used the main idea of survey propagation algorithm to model Ψ_β. Next we divide $V_{c\to x}$ into positive and negative parts: $V^+_{c\to x}$, $V^-_{c\to x}$. Thus the main steps in Ψ_β can be described as:

$$V^0_{x\to c} = V^+_{c\to x} + V^-_{c\to x} \tag{7}$$

here the variable $V^0_{x\to c}$ is for no effect to some clause. Then we update $V_{c\to x}$ as follows:

$$V^u_{x\to c} = V^+_{c\to x} \cdot (1 - V^-_{c\to x}) \tag{8}$$
$$V^s_{x\to c} = V^-_{c\to x} \cdot (1 - V^+_{c\to x}) \tag{9}$$
$$V_{c\to x} = \frac{V^u_{x\to c}}{V^u_{x\to c} + V^s_{x\to c} + V^0_{x\to c}} \tag{10}$$

Finally, in order to keep track of arbitrary long-term dependencies in the messaging queue, we apply recurrent neural network(Mikolov et al., 2011; Hochreiter & Schmidhuber, 1997) units to Φ_θ and Φ_ι, that is

$$F = \Phi_\iota(F_{x\to c}) \tag{11}$$
$$V = \Phi_\theta(V_{c\to x}) \tag{12}$$

where ι and θ are parameter vectors. In the process of solving the CNF problem, we use the Sigmoid function(Finney, 1952) as the activation layer of the model, hence all the variable value can be mapped to (0, 1).

2.3. The Reinforcement Learning Model

This step and the previous step are executed synchronously and can be seen as the typical framing of Reinforcement Learning(Boctor, 2013; Levin, Pieraccini & Eckert, 1998) scenario: our model generates some variables just like the actions taken by the agent, which are interpreted into a reward and a representation of the state, and then fed back into the agent.

We apply batch normalization(Bjorck., Gomes, Selman & Weinberger, 2018) to the result of graph learning step to improve learning rate. Therefore, the embedding of all clauses can be

obtained as

$$F = \Theta_{\varepsilon}(\Omega_{v}(G, V)) \tag{13}$$

where Θ_{ε} refers to Batch Normalization layer as described in (Ioffe & Szegedy, 2015) and ε is a parameter that used to ensure the legality of calculations. After obtaining the embedding related to all clauses, we use this value F in turn to derive the embedding value of the variables, and apply a normalization to the intermediate in order to prepare for getting probability distribution of all variables, that is

$$V = \Theta_{\zeta}(\Omega_{\lambda}(G^{T}, F)) \tag{14}$$

where Θ_{ζ} refers to normalization layer parameterized by the parameter ζ and Ω is a multi-layer neural network with adjustable parameters λ. Finally, we obtain the score distribution $\widehat{f}$ of the all clauses, which represents the probability that the clause is a *glue clause*

$$\widehat{f} = Softmax(\Omega_{\mu}(F)) \tag{15}$$

Similarly, we can calculate the probability distribution $\widehat{\nu}$ by following

$$\widehat{\nu} = Softmax(\Omega_{\lambda}(V, \overline{V})) \tag{16}$$

here μ and λ are different parameters of neural network Ω. The value $\widehat{v}$ has provided some inspiration for finding glue variables so that we can use it to combined with the prediction worked out from Section 3.2 to simplify the origin CNF formula.

3. Training a GVE Solver

Figure 7 below shows the schematic flow of GVE algorithm. It can be seen that GVE is mainly divided into three parts, graph learning, variable prediction and CNF simplification.

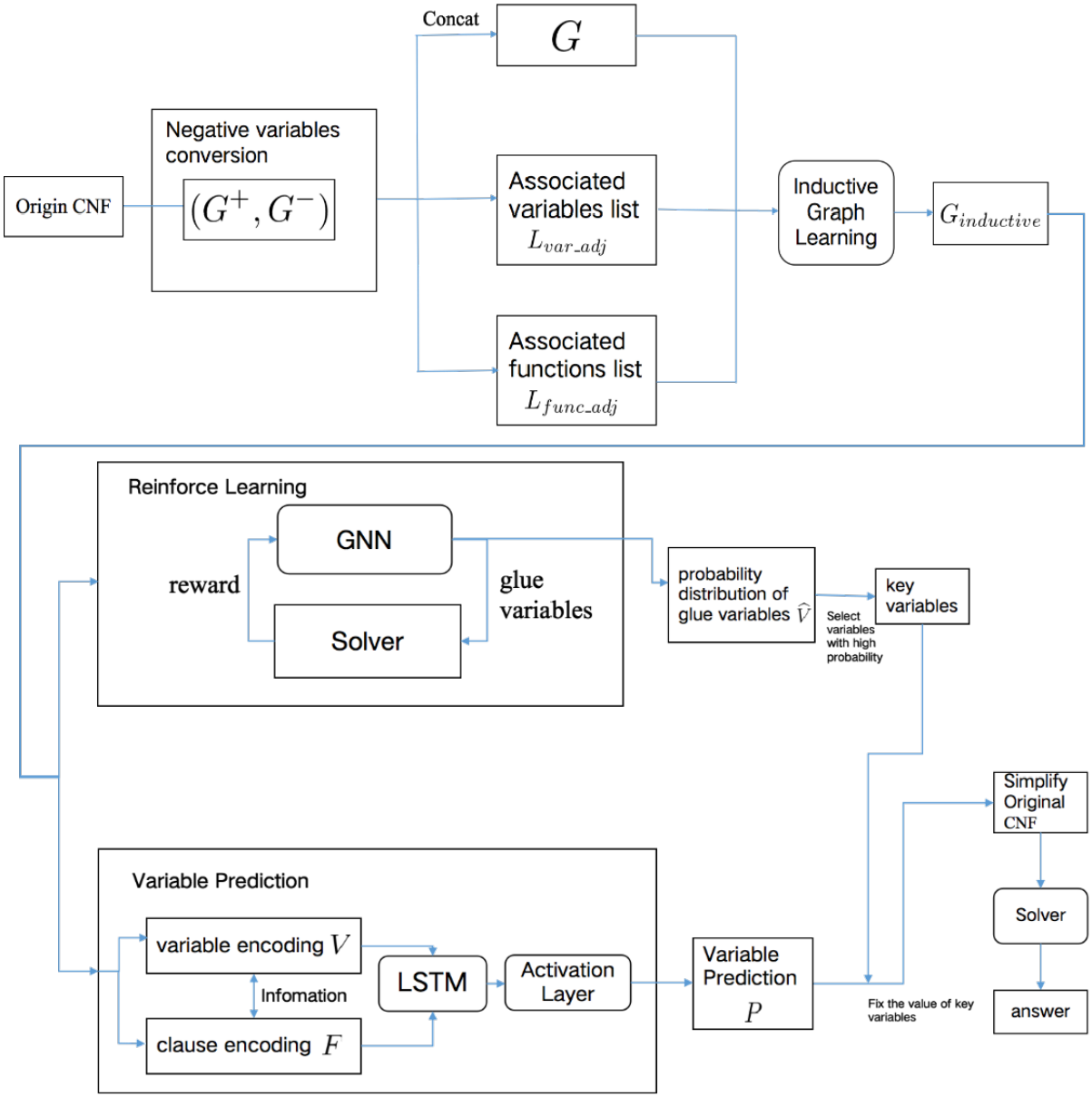


Figure 7: Schematic Flow of the GVE Algorithm

In order to train a GVE model, we would reduce losses from the above three parts, so we define the final loss as

$$loss = loss_{rl} + loss_{sp} \tag{17}$$

where $loss_{rl}$ represents the loss of the glue clause prediction model and $loss_{sp}$ represents the loss of the variable value prediction of the model. We modified the classic glucose solver(Audemard & Simon, 2014) so that we can used it to calculate the LBD scores(Audemard & Simon, 2009) of all clauses, denoted as S_{lbd}. We add up the scores of all the clauses where the variable is located to get the score of the variable. After a softmax operation, we get P_χ

$$P_{\chi i} = Softmax(\frac{\sum\limits_{f_j \ni \partial v_i} S_{lbdj}}{\sum\limits_{m=1}^{M} S_{lbdm}}) \tag{18}$$

For all variables, we can get the probability distribution that if a variable is a glue variable Q_χ, so that

$$loss_{rl} = -\sum_{i=1}^{N} P_{\chi i} \log \frac{Q_{\chi i}}{P_{\chi i}} \tag{19}$$

where P_χ is the probability distribution of glue variables given by the deterministic solver. Hence we achieve the purpose of model adjustment by minimizing the difference between the fitting probability and the target probability.

For reasons of avoiding to label the data before training, we treat the final result as a classification problem for subcategories -- each clause has two categories, corresponding to true or false. We use the most classic cross -- entropy loss function(Parsian & Nematollahi, 1996) of the classification algorithm as the prototype of the loss function

$$loss_{sp} = -\frac{1}{M} \sum_{i=1}^{M} \log \frac{\exp(H(f_i)/\tau)}{\sum\limits_{j=1}^{M} \exp(H(f_i^{(j)})/\tau)} \tag{20}$$

$$H(f_i) = G^{+}_{\partial f_i} \cdot V_{\partial f_i} + G^{-}_{\partial f_i} \cdot (1 - V_{\partial f_i}) \tag{21}$$

where f_i represents a clause in CNF, and $G_{\partial f_i}$, $V_{\partial f_i}$ represents the graph structure and variables related to clause f_i. Note that τ is a temperature parameter(Amizadeh et al., 2019) in the formula, and we will take a larger value as its initial value, and then gradually decrease to 0 during the training process.

Algorithm 1 The GVE Computation Algorithm

Input: Graph $G \in R^{M \times 2 \cdot N}$; Train iterations T; Number of simplified variables n; Deterministic solver S

Output: $V^{(t)}$, $\widehat{v}^{(t)}$

for $t = 1$ to T **do**
  /*Graph Learning */
  for $V_i \in (V^+, V^-)$ **do**
    Compute V using Eqs.(2),(3)
  end for
  $V = \Upsilon_\alpha(\text{Concat}(V^+, V^-))$
  /*Variable Prediction */
  for $v_i \in G$ **do**
    Compute F, V using Eqs. (5)-(12)
  end for
  /*Glue Variables Prediction */
  Compute clause embedding F using Eq. (13)
  Compute probability distribution $\widehat{f}$, $\widehat{v}$ using Eqs. (14),(15),(16)
  $V^{(t)} \leftarrow \{v_i : i \in 1...N\}$
  $\widehat{v}^{(t)} \leftarrow \{\widehat{v}_i : i \in 1...N\}$
  /*Solve CNF when not Training */
  if $Training = False$ **then**
    $v \leftarrow \{\widehat{v}_i : i \in 1...n\}$
    $CNF_\xi \leftarrow simplify(V, v)$
    solve CNF_ξ using S
  end if
end for
return $V^{(t)}$, $\widehat{v}^{(t)}$

Algorithm 1 describes the calculation process of the GVE model. After the step of graph induction learning, the representation V of all variables is obtained. We use V as the basis for obtaining the scores of glue variables and the prediction of all variable values. In the prediction step, we compound the results of the two, select variables with higher scores from the results of the glued variables, where the number of selected variables is related to the number of variables in the original CNF, and then fix these variables according to the results of the variable prediction model. Next we perform the step of "simplification": if the value of a variable is positive after compounding with its sign, we delete the related clause from the original CNF; otherwise, we delete the variable from the clause where it is located. This step greatly simplifies the original problem, so that the simplified CNF can be solved quickly with a deterministic solver.

4. Experiments

We compared our model with two different solutions: (a) recently proposed model PDP(Amizadeh et al., 2019) using pure neural network method, (b) CADICAL(QUEUE, 2019), the top solver in the SATCOMP competition. The experimental data are from the official data sets provided by the SATCOMP 2003-2019. We have compared the above solutions in terms of accuracy and solving speed.

PDP

Since the PDP model has different model training parameters for different data types, we selected representative "modular"(Ansótegui, Giráldez-Cru & Levy, 2012; Walsh, 1999) data as the training set to train the PDP model. The model structure is set with the author's default parameters, and the best model trained is used as the experimental solver. The PDP model has the advantage of batch solving, so in order to facilitate comparison, we record the time to solve each SAT problem separately.

CADICAL

The solver has a lot of parameters that can be configured, and the default settings are used here. We use the "total process time since initialization" in the CADICAL output as its solving time for comparison.

4.1. Data sets

In order to compare the effect of GVE with different solvers, we selected the CNF data set with the number of variables in the range of (0, 3000) from the SATCOMP 2003-2019 competition benchmarks. In several data intervals of the number of variables, it is divided into the following aspects: gradual increase in the number of variables, gradual increase in the the ratio of the number of clauses to the number of variables, and the same number of variables and clauses for comparison.

4.2. Results

Accuracy

Since CADICAL is a deterministic solver, no comparison is made in the part of correctness. The main object of comparison here is the accuracy of GVE and PDP model in multi-interval data sets. To ensure the performance of the trained PDP model, a large number of data sets are needed as training sets. Therefore, in addition to the data provided by SATCOMP, our training sets also contain part of the data generated by the generator. At least 1000 CNFs are prepared for training and 200 CNFs are selected for each data interval in testing process. For the sake of improving the accuracy of PDP, in particular, we use the optional parameter “local_search_iteration” of the model which increases the number of times of randomly flipping unsatisfiable variables after the model is applied. We also set another variable “batch_replication” to increase the number of repeated attempts.

Table 1 is a comparison result of the accuracy of two models. An explanation is needed here. Since PDP is a pure neural network model which does have a good performance when solving ordinary CNFs, but due to the high difficulty of the data we use, it is very difficult to determine all variables only by machine learning. Therefore, its performance is far from satisfactory. Meanwhile, it also shows the original intention of our proposed model from another direction: it is difficult to solve the whole problems, so we can settle for second beat and only solve the core part of the SAT problem, thereby reducing the overall workload.

Name	(0, 500]	(500, 1000]	(1000, 1500]	(1500, 3000)
GVE	81.1%	76.7%	75.01%	72.3%
PDP	6.7%	5.1%	3.2%	1.2%

Table 1: Comparison of the Accuracy of GVE and PDP

In order to improve the accuracy rate, we will try to change the determined variables for the CNF that is "UNSAT" for the first time, and then repeat the simplification and solution process. Figure 8 shows accuracy and average number of retries, which indicates GVE can basically control the number of retries below ten times, which is directly related to the difficulty of the problem.

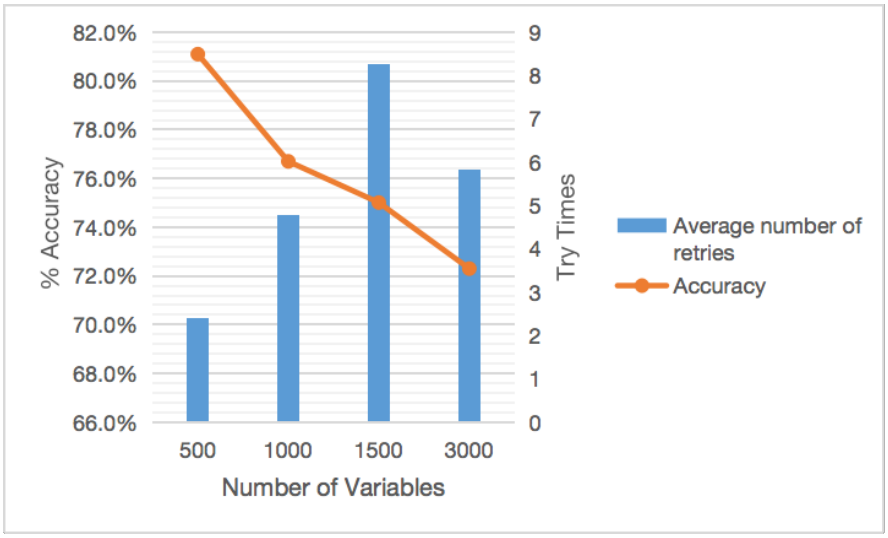


Figure 8: Accuracy of GVE and Retry Times in Different Data Intervals. Retry Times Means that We Change the Number of Variables We Have Determined, Re-simplify and then Pass the New CNF to the Solver for Solution. the Abscissa Represents the Upper Bound of the Clause Number Interval, for Example, 500 Represents the Interval (0, 500).

Degree of Simplification

Due to the fact that GVE uses the idea of machine learning to solve part of the problem, the number of removed clauses which is determined according to the number of simplified variables indicates whether it is good enough to help reduce the workload of the original problem. Figure 9 shows the average reduction ratio of GVE to original CNF formula in different interval data sets. The experimental results shows that after fixing some variables, GVE not only reduces these fixed variables, but also some associated variables are deleted indirectly during the process of simplification because of the deletion of their related clauses.

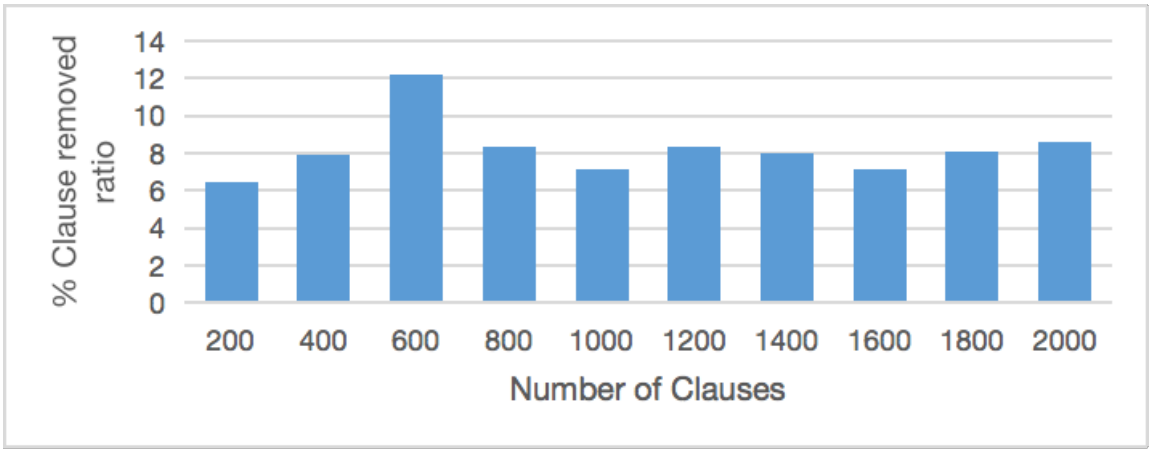
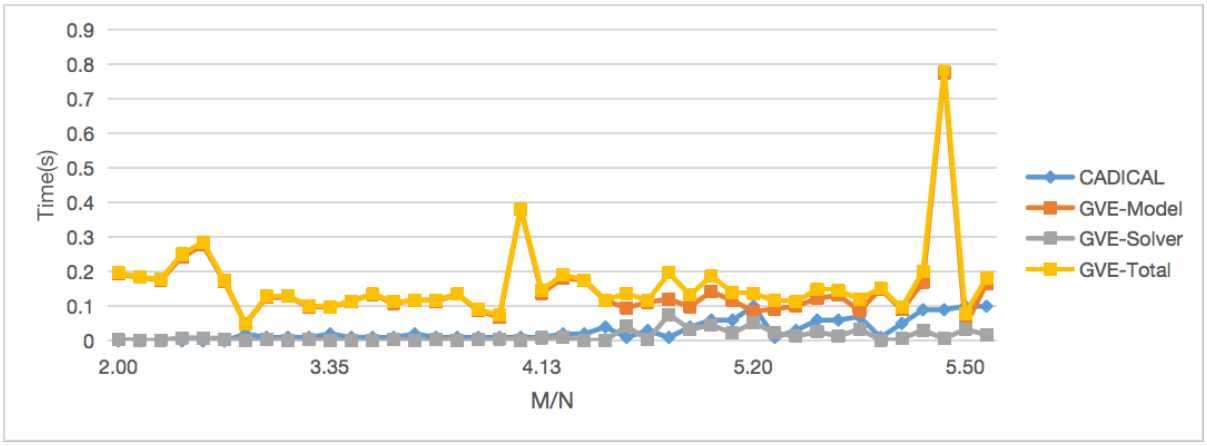


Figure 9: Reduction ratio of GVE to original CNF formula.

Time

The main comparison objects in this part are the GVE model and the CADICAL solver. The following results can be obtained based on the same data set. Note that here we only compare among the data sets where the GVE model can get the correct results, and the data sets for which the correct answer is not available will be ignored. The time of GVE consists of two parts: model solution time(the time to determine the glue variable plus the time to derive the value of the variable), and the time to use the deterministic solver to solve the simplified CNF. In order to ensure the fairness of the comparison experiment, the deterministic solver here also uses CADICAL. We use the "total process time since initialization" value in the output result as the time for this part.

Figure 10-12 shows the comparison results of multiple data intervals. It can be seen from the figures that in the case of a simpler SAT problems, the performance of GVE is similar to the CADICAL solver, and they both takes very little time; as the difficulty of the problem increases, that is, when the complexity of CNF formula gradually increases, the GVE model has a stable performance.



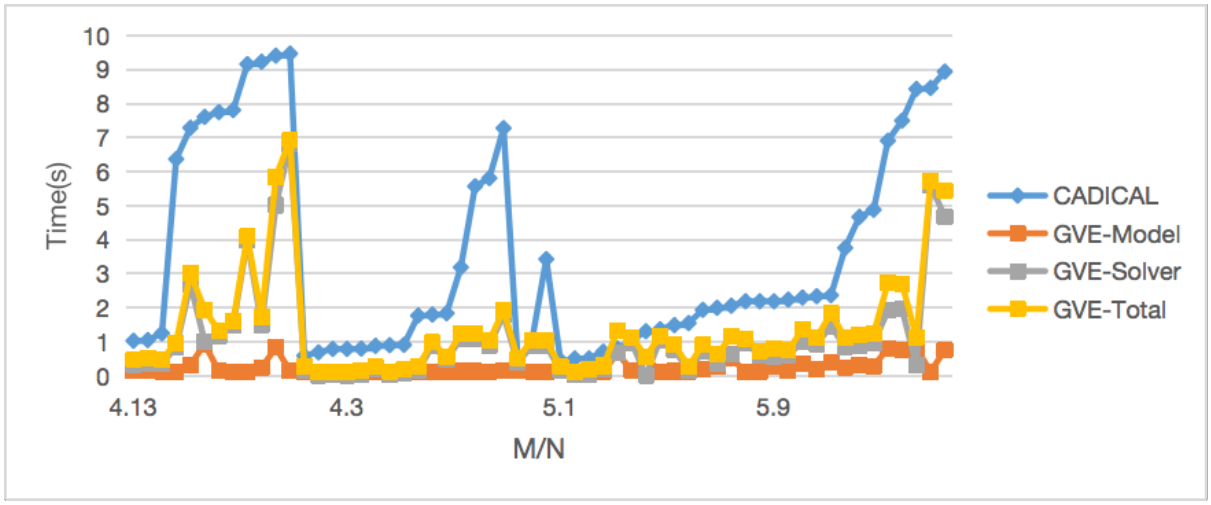


Figure 10: The Time for CADICAL vs. GVE on Simple Cnfs. as the Complexity of the Problem Increases, the Advantages of GVE Gradually Appears.

For those complicated ones, especially the problems that CADICAL needs hundreds of seconds to solve, GVE can also control the solution time to dozens of orders of magnitude, which can be said to have outstanding performance on more complex problems.

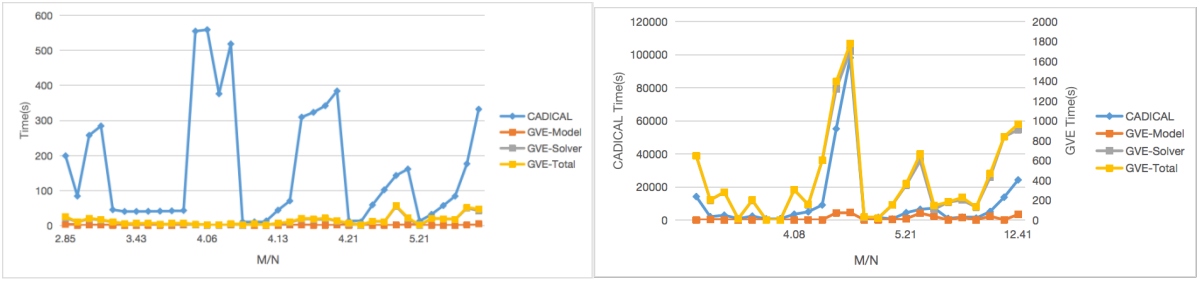
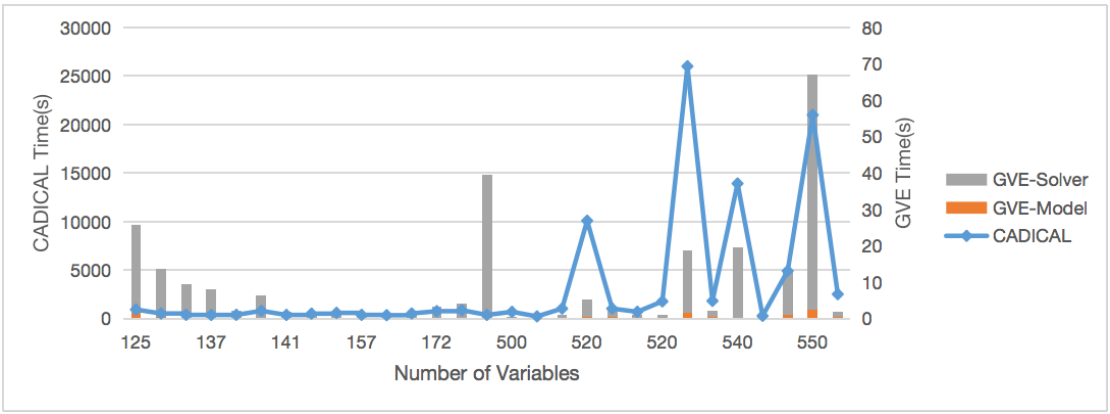


Figure 11: The Time for CADICAL vs. GVE on Hard Cnfs. although CADICAL Performance Fluctuates Greatly, GVE Always Performs Relatively Stable.

In order to analyze the performance of GVE, we also compared the solution time of GVE model and the deterministic solver from the perspective of variable changes.



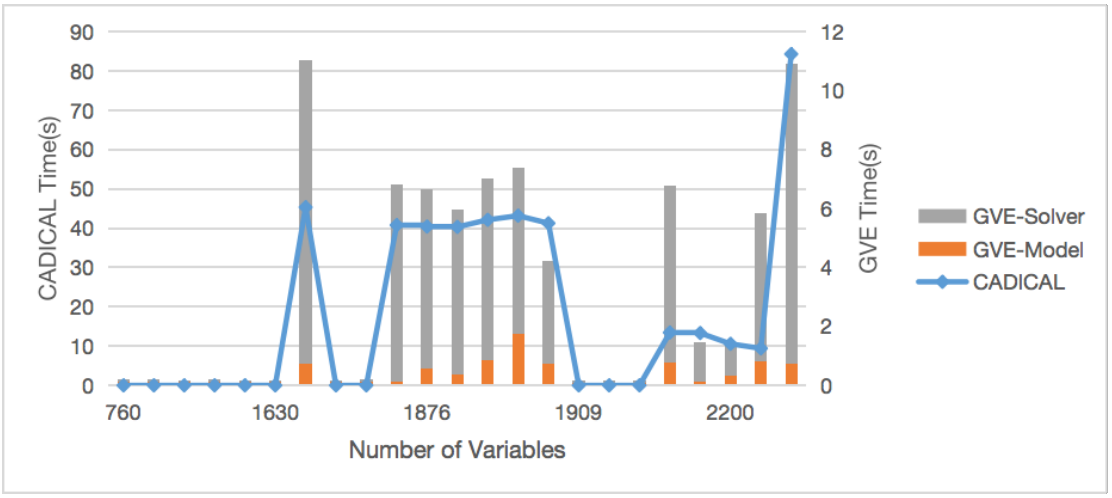


Figure 12: The Time Changes for CADICAL vs. GVE with the Number of Variables.

For a group of CNFs with similar variables, the solution time of the deterministic solver fluctuates greatly. GVE makes the solution time stable within a certain interval by finding and then fixing the glue variables.

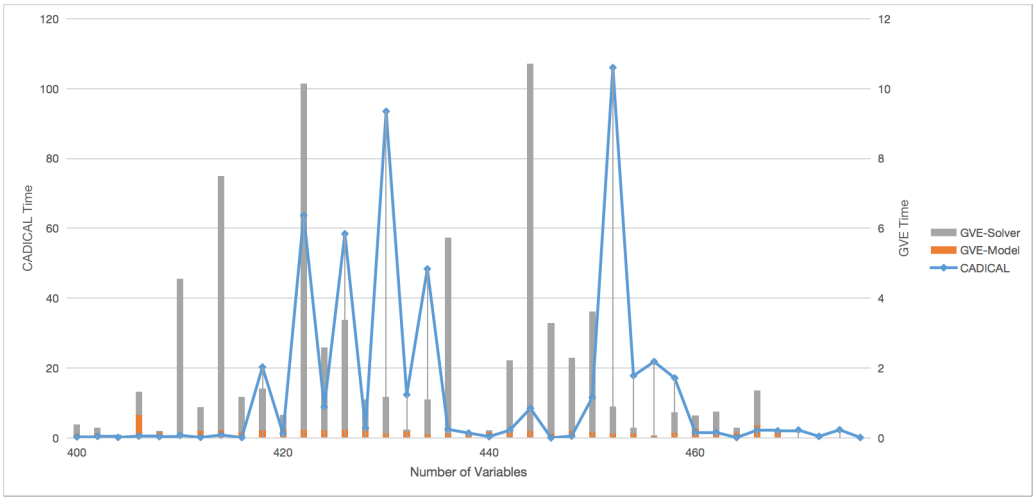


Figure 13: The Time for CADICAL vs. GVE On Cnfs that Have Similar Number of Variables. It Can Be Seen that The Pure Model Part of GVE Performs Stable, that Is, We Can Determine the Glue Variables of a CNF and its Value ina Short Time.

If the solution result of a CNF is UNSAT, in order to improve the accuracy of the GVE model as well as avoiding erroneous results caused by model errors, we will perform multiple calculations on such CNFs. This process is called “downgrade calculation”. In the process of downgrade calculation, GVE will reduce the number of fixed variables, re-simplify the original CNF, and then solve it until the correct result is obtained. Figure 14 shows the change in the number of deterministic variables and the time taken for the downgrade solution.

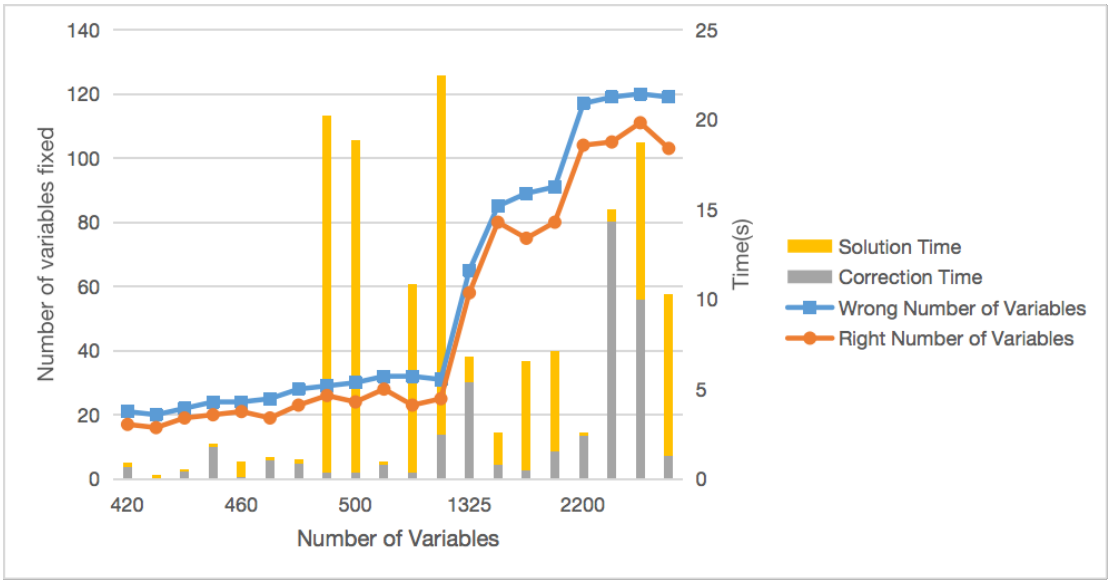


Figure 14: Number of Variables Fixed and Time Required after Downgrade Solution

UNSAT Instances

Excluding errors caused by model errors, there is another possibility that this CNF is indeed UNSAT. Therefore, after a few failed attempts, GVE will start from another angle and try to find the "UnsatCore" of the original problem. Due to the characteristics of UnsatCore, we use the obtained glue variables to form new CNF, designated CNF_{Λ}, which is formed by the original CNF and new clauses that consists of the different combinations of glue variables as well as their signs. Figure 15 shows the comparison of some GVE and CADICAL solution results and solving time they need on UNSAT CNFs.

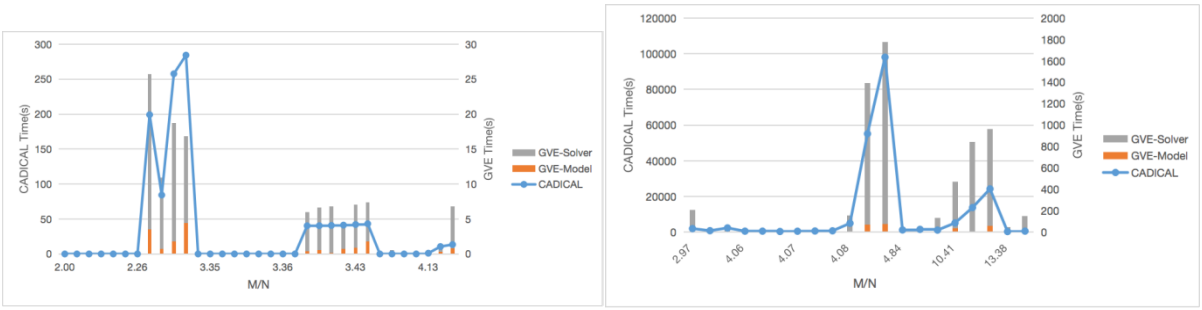


Figure 15: The Time for CADICAL vs. GVE on UNSAT Cnfs. GVE-Solver Time Is the Sum of Time that Solving All Possible CNF Included in CNF_{Λ}

From the above experimental results, it can be seen that GVE is a very useful compound solver. We have also given full play to the advantages of model learning algorithms and deterministic solvers. The idea of partial simplification can greatly reduce the solution time for complex problems. At the same time, UnsatCore, which can solve unsolvable problems, also greatly improves the credibility of the model.

Conclusion

In this article, we propose a composite model GVE to solve complex SAT problems. We apply the idea of graph induction to the learning of CNF structure which avoids the complicated operation of training different models for different types of SAT problems, so

there is no need to classify the data first. Secondly, we use reinforcement learning to find the glue variables and glue clauses of the CNF formula, and then use a neural network model designed based on the idea of SP algorithm to determine the values of the glue variables which are used to process the original SAT problem. Finally we use a deterministic solver to solve the simplified CNF. This approach allows us to fix only the most critical part of the variables for the entire problem, reducing the impact of model errors on problem solving, and at the same time improving the speed of solving complex problems.

Note that GVE may not be effective for originally simple problems, even slower than deterministic solver, but it has a significant performance improvement for those complex problems. This is also our intention of designing this model. GVE model provides an idea of combining neural networks and deterministic solvers, and our results also reveal the tremendous potential for pursuing this goal.

References

Aho, A. V., & Hopcroft, J. E. (1974). The design and analysis of computer algorithms. Pearson Education India.

Amizadeh, S., Matusevych, S., & Weimer, M. (2018, September). Learning to solve circuit-SAT: An unsupervised differentiable approach. In International Conference on Learning Representations.

Amizadeh, S., Matusevych, S., & Weimer, M. (2019). PDP: A general neural framework for learning constraint satisfaction solvers. arXiv preprint arXiv:1903.01969.

Ansótegui, C., Giráldez-Cru, J., & Levy, J. (2012, June). The community structure of SAT formulas. In International Conference on Theory and Applications of Satisfiability Testing (pp. 410-423). Springer, Berlin, Heidelberg.

Audemard, G., & Simon, L. (2009, June). Predicting learnt clauses quality in modern SAT solvers. In Twenty-first International Joint Conference on Artificial Intelligence.

Audemard, G., & Simon, L. (2009). Glucose: a solver that predicts learnt clauses quality. SAT Competition, 7-8.

Audemard, G., & Simon, L. (2014). Glucose in the SAT 2014 Competition. SAT COMPETITION 2014, 31.

Biere, A., Heule, M., & van Maaren, H. (Eds.). (2009). Handbook of satisfiability (Vol. 185). IOS press.

Bjorck, N., Gomes, C. P., Selman, B., & Weinberger, K. Q. (2018). Understanding batch normalization. In Advances in Neural Information Processing Systems (pp. 7694-7705).

Boctor, L. (2013). Active-learning strategies: The use of a game to reinforce learning in nursing education. A case study. Nurse education in practice, 13(2), 96-100.

Braunstein, A., Mézard, M., & Zecchina, R. (2005). Survey propagation: An algorithm for satisfiability. Random Structures & Algorithms, 27(2), 201-226.

Evans, R., Saxton, D., Amos, D., Kohli, P., & Grefenstette, E. (2018). Can neural networks understand logical entailment?. arXiv preprint arXiv:1802.08535.

Finney, D. J. (1952). Probit analysis: a statistical treatment of the sigmoid response curve. Cambridge university press, Cambridge.

Garey, M. R., & Johnson, D. S. (1979). Computers and intractability (Vol. 174). San Francisco: freeman.

Geng, X., Li, Y., Wang, L., Zhang, L., Yang, Q., Ye, J., & Liu, Y. (2019, July). Spatiotemporal multi-graph convolution network for ride-hailing demand forecasting. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 33, pp. 3656-3663).

Hamilton, W., Ying, Z., & Leskovec, J. (2017). Inductive representation learning on large graphs. In Advances in neural information processing systems (pp. 1024-1034).

Heule, M. J., Kullmann, O., Wieringa, S., & Biere, A. (2011, December). Cube and conquer: Guiding CDCL SAT solvers by lookaheads. In Haifa Verification Conference (pp. 50-65). Springer, Berlin, Heidelberg.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

Hoos, H. H. (2002, July). An adaptive noise mechanism for WalkSAT. In AAAI/IAAI (pp. 655-660).

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.

Karp, R. M. (1972). Reducibility among combinatorial problems. In Complexity of computer computations (pp. 85-103). Springer, Boston, MA.

Knuth, D. E. (1997). The art of computer programming (Vol. 3). Pearson Education.

Kumar, V. (1992). Algorithms for constraint-satisfaction problems: A survey.AI magazine,13(1), 32-32.

Levin, E., Pieraccini, R., & Eckert, W. (1998, May). Using Markov decision process for learning dialogue strategies. In Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'98 (Cat. No. 98CH36181) (Vol. 1, pp. 201-204). IEEE.

Li, Y., Tarlow, D., Brockschmidt, M., & Zemel, R. (2015). Gated graph sequence neural networks. arXiv preprint arXiv:1511.05493.

Liang, J. H., Ganesh, V., Poupart, P., & Czarnecki, K. (2016, July). Learning rate based branching heuristic for SAT solvers. In International Conference on Theory and Applications of Satisfiability Testing (pp. 123-140). Springer, Cham.

Mezard, M., & Montanari, A. (2009). Information, physics, and computation. Oxford University Press.

Mikolov, T., Kombrink, S., Burget, L., Černocký, J., & Khudanpur, S. (2011, May). Extensions of recurrent neural network language model. In 2011 IEEE international conference on acoustics, speech and signal processing (ICASSP) (pp. 5528-5531). IEEE.

Nieuwenhuis, R., Oliveras, A., & Tinelli, C. (2005, March). Abstract DPLL and abstract DPLL modulo theories. In International Conference on Logic for Programming Artificial Intelligence and Reasoning (pp. 36-50). Springer, Berlin, Heidelberg.

Palm, R., Paquet, U., & Winther, O. (2018). Recurrent relational networks. In Advances in Neural Information Processing Systems (pp. 3368-3378).

Parsian, A., & Nematollahi, N. (1996). Estimation of scale parameter under entropy loss function. Journal of Statistical Planning and Inference, 52(1), 77-91.

QUEUE, S. D. CADICAL at the SAT Race 2019. SAT RACE 2019, 8.

Selsam, D., Lamm, M., Bünz, B., Liang, P., de Moura, L., & Dill, D. L. (2018). Learning a SAT solver from single-bit supervision. arXiv preprint arXiv:1802.03685.

Sherrington, D., & Kirkpatrick, S. (1975). Solvable model of a spin-glass. Physical review letters, 35(26), 1792.

Spitzer, F. (2013). Principles of random walk (Vol. 34). Springer Science & Business Media.

Van Leeuwen, J. (Ed.). (1991). Handbook of theoretical computer science (vol. A) algorithms and complexity. Mit Press.

Vizel, Y., Weissenbacher, G., & Malik, S. (2015). Boolean satisfiability solvers and their applications in model checking. Proceedings of the IEEE, 103(11), 2021-2035.

Walsh, T. (1999, July). Search in a small world. In Ijcai (Vol. 99, pp. 1172-1177).

Xu, L., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2008). SATzilla: portfolio-based algorithm selection for SAT. Journal of artificial intelligence research, 32, 565-606.

Zhang, L., & Malik, S. (2002, July). The quest for efficient boolean satisfiability solvers. In International Conference on Computer Aided Verification (pp. 17-36). Springer, Berlin, Heidelberg.

Contact email: zwzhang@bupt.edu.cn