Original Research Paper

Variable Selection with PageRank for SAT Solvers

Tomohiro Sonobe

Global Research Center for Big Data Mathematics, National Institute of Informatics, Japan

Article history
Received: 01-05-2019
Revised: 25-07-2019
Accepted: 05-08-2019

Email: tomohiro_sonobe@nii.ac.jp

Abstract: How to choose decision variables often determines the performance of SAT solvers. In state-of-the-art SAT solvers, Variable State Independent Decaying Sum (VSIDS) has been used as a standard technique in the decision process. In this study, we analyze the VSIDS from the point of view of PageRank and propose a technique for improving the VSIDS. While the VSIDS focuses on local search spaces, the PageRank values are based on the relative importance from a global point of view. From this fact, we utilize the PageRank values for controlling the VSIDS and improve the performances of representative SAT solvers, MiniSAT and Glucose.

Keywords: SAT, Solver, PageRank, Variable Selection

Introduction

When a Boolean formula is given, the Boolean Satisfiability (SAT) problem asks whether an assignment of variables exists, which evaluates the formula as true. A SAT problem is known as a classical NP-complete problem and is believed not to be solvable in polynomial time. In general, a formula is given in Conjunctive Normal Form (CNF). The solvers for this kind of problem are called SAT solvers. Today, they are used for real-world applications (Marques-Silva, 2008), such as circuit design (Stephan *et al*., 2006) and neural network verification (Narodytska *et al*., 2018).

Many SAT solvers adopt the Davis-Putnam-Logemann-Loveland (DPLL) algorithm (Davis *et al*., 1962), which is based on a backtrack search. During the last dozen years, various important methods have been proposed to improve the performance of DPLL, such as, Conflict-Driven-Clause-Learning (CDCL) to prevent reappearance of similar searches, restart (Gomes *et al*., 1998) instead of backtracking to start different search from the first beginning in order to avoid heavy-tail behavior, and Variable State Independent Decaying Sum (VSIDS) decision heuristic (Moskewicz *et al*., 2001) to determine the priority to select variables to be assigned. The VSIDS is independent from the state of the variable, thus its management is simple and easy. Many CDCL/VSIDS-based solvers give scores to prioritize a set of variables that appear in learnt clauses in order to fully utilize the obtained learnt clauses.

SAT instances from real-world applications have an internal structure, where specific variables have strong relations to each other. For example in software verification, a variable in a program can have At Least One (ALO) value and At Most One (AMO) value from a given range. By using a direct encoding, all the candidate values are encoded into multiple Boolean variables (e.g., $v_1, v_2, \ldots, v_r$ for the range [1..r]) in the SAT instance and ALO/AMO constraints are encoded into clauses. Amongst these Boolean variables, if one of them is assigned to true, then the others are assigned to false. Such variables exist in the instances from real-world applications, which shapes the structure of SAT instance.

The VSIDS decision heuristic can select the related variables to currently assigned variables, which boost the efficiency of the search. This is because the VSIDS prioritizes variables in learnt clauses. The variables in a learnt clause are considered to be the culprits of a conflict and these variables can be a part of the structure. The VSIDS efficiently manages the priority of each variable (as a simple score) in a dynamic manner. Many state-of-the-art SAT solvers use the VSIDS.

In this study, we analyze the VSIDS with PageRank (Page *et al*., 1999) and improve the performance of SAT solvers by combining the PageRank with the VSIDS. The PageRank values stand for the relative importance of vertices in a graph. When we convert a SAT instance to a Variable Incidence Graph (VIG) proposed by Ansótegui *et al*. (2012), we can calculate the PageRank value of each Boolean variable. As Katsirelos and Simon (2012) have already revealed the relation between the Boolean variables and their PageRanks, we also observe the relation from another point of view. We implemented a function to compute the PageRank of a VIG in MiniSAT 2.2 (Eén and Sörensson, 2003) and conducted experiments with 300 instances from the SAT Competition 2014 application track to observe the relation between the VSIDS and PageRank. As a result, although we confirm that variables with a high PageRank are often selected as decision variables, we found that the VSIDS scores did not completely correlate with the PageRank.

From this observation, we can utilize the PageRank to enhance the VSIDS. It is difficult to gain good performance by combining the PageRank directly with the VSIDS scores because the PageRank values stand for relative importance from a global point of view. In contrast, the VSIDS often focuses on a limited part of structure, not a global structure. In addition, the calculation of PageRank often needs a long computational time. Hence we should not calculate it so frequently. In our method, we reflect the PageRank to the VSIDS scores for every a certain number of restarts. For the instances whose PageRank distribution has an almost uniform shape, we avoid using the method since these values have no information. In order to utilize the learnt clauses, the proposed method periodically reconstructs the VIG and recalculate the PageRank. We implemented the proposed method to MiniSAT 2.2 and Glucose (Audemard and Simon, 2009) version 3 and conducted experiments for 300 instances from the SAT Competition 2014. The experimental results indicates that the proposed method can improve both solvers.

Our contribution is summerized as follows:

- We analyze VSIDS with PageRank by converting SAT instances into variable incidence graphs. It figures out that both scores are highly correlated and variables with high PageRank are often selected as decision variables
- VSIDS can often focus on a local part of the search space. In contrast, PageRank scores stand for the relative importance of variables from a global point of view. The proposed method can bring VSIDS to escape from the local part when a restart is invoked
- The experimental results exhibit that the proposed method can boost the performance of state-of-the-art SAT solvers

Related Work

Katsirelos and Simon (2012) firstly analyzed the solver activity with eigenvector (PageRank). They also showed that variables with high PageRanks tended to be assigned values. This paper sheds light on the local view of the VSIDS and global view of the PageRank and improves the decision heuristic by utilizing it.

The way to select the decision variables is the fatal part of SAT solvers. Various types of decision heuristics are overviewed in (Biere and Fröhlich, 2015). In specific, Variable State Independent Decaying Sum (VSIDS) is the most basic one for the recent SAT solvers and was implemented first in Chaff (Moskewicz *et al*., 2001). The VSIDS chooses the decision variable with the highest score. Before the advent of the VSIDS, the computational cost of decision heuristics was quite high. For example, dynamic largest combined sum proposed by Silva (1999) is one of them. Huang and Darwiche (2003) proposed a decision heuristic based on a tree decomposition technique, however its computational cost was also high. Bruni and Santori (2008) modified the VSIDS by adding more scores to variables involved in the conflicts. In recent years, some techniques (Liang *et al*., 2016; 2017; Nejati *et al*., 2017; Selsam and Bjørner, 2019) from the discipline of machine learning enhanced the decision heuristics in order to more dynamically select the suitable variabes.

The VSIDS decision heuristic is considered to boost the intensive search for the internal structure of the SAT instance. It is confirmed that there are specific sets of variables. For exmaple, backbone proposed by Monasson *et al*. (1999) and backdoor proposed by Williams *et al*. (2003) can make the instance easy to solve. The structure is analyzed from a point of view of graph theory, especially by using modularity (Clauset *et al*., 2004). The modularity is a value for a partition of vertices into communities and a high modularity value indicates that a high density of edges in the communities and a low density of edges between the communities. Ansótegui *et al*. (2012) showed that SAT instances from real-world applications had quite high modularity values and randomly generated instances had low modularity values. Newsham *et al*. (2014) pointed out that the number of detected communities correlated with literal block distance (Audemard and Simon, 2009) and the modularity values were useful for predicting the performance of the SAT solvers. There are some works that improve the solver performance. Martins *et al*. (2013) utilized the detected community for MAX-SAT solvers to select relaxation variables. Sonobe *et al*. (2014) proposed a diversification technique, called community branching, for portfolio-based parallel SAT solvers. Jamali and Mitchel (2018) incorporated betweenness centrality into the decision process aiming at prioritizing variables appearing in many shortest paths, which means that these variables have strong influence on many other variables. Such variables were also introduced as bridge variable in (Liang *et al*., 2015). Other than graph theory, Ansótegui *et al*. (2014) described self-similarity property of instances from real-world applications.

Analysis of SAT Solver

In this section, we explain technical background and some analysis results.

Technical Background

A SAT instance Π is a conjunction of clauses, where a clause $c = (l_1 \vee l_2 \vee \ldots l_n)$ is a disjunction of literals. A literal is a positive or negative form of a Boolean variable. An empty clause is always false and an empty SAT instance is always true. A conflict occurs when an empty clause appears under a certain variable assignment. A SAT instance is satisfiable if there is a solution in the formula otherwise it is unsatisfiable:

Algorithm 1: Pseudo code of CDCL

```
Input: a CNF formula $\Pi$
Output: satisfiable or unsatisfiable
1:  level = 0 // decision level
2:  trail = $\emptyset$ // assignment of variables
3:  learnts = $\emptyset$ // learnt clauses
4:  inc_score = 1.0 // incremental value for VSIDS scores
5:  conf // conflicting clause
6:  learnt // learnt clause
7:  blevel // level to backtrack
8:  next // next decision variable
9:  while true do
10:     conf = unitPropagation($\Pi$, trail) // a conflicting variable
11:     if conf $\neq$ NULL then
12:         learnt = conflictAnalysis($\Pi$, trail, conf)
13:         blevel = calcBackjumpLevel(learnt)
14:         if blevel < 0 then
15:             return unsatisfiable
16:         end if
17:         for each var in learnt do
18:             increaseVSIDSScores(var, inc_score)
19:         end for
20:         inc_score = inc_score / 0.95
21:         learnts = learnts $\cup$ learnt
22:         if restart() then
23:             blevel = 0
24:         end if
25:         backjump(blevel);
26:         level = blevel
27:     else
28:         next = chooseDecisionVariable($\Pi$, trail)
29:         if next == NULL then
30:             return satisfiable
31:         end if
32:         trail = trail $\cup$ next
33:         level = level + 1
34:     end if
35: end while
```

The state-of-the-art SAT solvers for application instances are based on Conflict-Driven-Clause-Learning (CDCL) which was derived from the DPLL algorithm. The pseudo code of CDCL is shown in Algorithm 1. Given a CNF formula, this algorithm determines whether the formula is satisfiable or unsatisfiable. This code also includes the part of VSIDS and is based on MiniSAT (Eén and Sörensson, 2003). The variable "level" is the decision level that stands for the depth of the search tree. The function "unitPropagation" conducts the propagation for unit clauses and returns a conflicting variable if a conflict occurs. After a conflict, the function "conflictAnalysis" conducts the clause learning. From the learnt clause, the function "calcBackjumpLevel" calculates a level to which the solver jumps back. If the backjump level is less than zero, the given formula is unsatisfiable. Then the scores of the variables in the learnt clause are increased by the function "increaseVSIDSScores" and the degree of increment ("inc_score") is updated by being divided by 0.95 (this value is used in MiniSAT). Note that all the VSIDS scores and "inc_score" are decreased before a overflow. The backjump level becomes zero when restarting of the search is invoked. Finally, the backjumping is conducted and the decision level is renewed. If there is no conflict, the function "chooseDecisionVariable" picks up a decision variable with the highest VSIDS score. If there is no variable to assign, the search is ended and the formula is turned out to be satisfiable.

The procedure of VSIDS is as follows:

1. Initialize all the scores as 0 (or randomly)
2. Choose a variable with the highest score as a decision variable
3. Increase the scores of variables in learnt clauses
4. Decrease all the scores periodically

Note that the scores are increased not only when the variables are in learnt clauses, but also when the variables are involved in the learning process in recent solvers. By increasing the scores of learnt variables, the VSIDS achieves intensive searches for local parts of the structure. For the purpose of reduction of computational costs, the recent implementation increases the degree of increment instead of the decrement of all the scores.

The PageRank is one of basic metrics for calculating the importance of each vertex in a graph. Although it is originally used to rank Web pages in the search engine, today it is applied to other networks such as bioinformatics (Morrison *et al*., 2005) and image categorization (Pan *et al*., 2004). Assume that we have a weighted directed graph $G = (V, E)$ with n vertices and m edges. We denote a weight of an edge as $w : V \times V \to \mathbb{R}^+$, satisfying:

$$\sum_{v|(u,v)\in E} w(u,v) = 1 \tag{1}$$

for a vertex u. The PageRank is the stationary distribution of the random walk with random jumping with probability c, called the teleportation probability. The walking goes on to outgoing edges from the current vertex with probability $1-c$. In general, the PageRank π for vertex v is calculated as follows:

$$\pi(v) = c\delta(v) + (1-c)\sum_{u|(u,v)\in E} \pi(u)w(u,v) \tag{2}$$

where, $\delta(v)$ is a probability to be selected as the destination vertex from the random jumping and is set to 1.0 / n. The teleportation probability c is set to 0.15 in the original paper (Page *et al*., 1999). This calculation can converge in several dozen of iterations.

We create a Variable Incidence Graph (VIG) from the given CNF (Ansótegui *et al*., 2012). In the VIG, the vertices correspond to the Boolean variables and the edges correspond to the relations between the variables in the same clause. A clause "C" generates ${}_{|C|}C_2$ edges ($|C|(|C|-1)/2$) between every pair of the variables in the clause C. The weight of each edge is ($1/{}_{|C|}C_2$), therefore, the sum of the weights of the edges added by each clause is always 1. We create a VIG by traversing all the clauses and learnt clauses.

VSIDS and PageRank

The VSIDS tries to choose important variables and the importance is expressed as a score. The score is increased when the variable appears in learnt clauses. The more variables appear in clauses, the more frequently they are assigned values and appear in learnt clauses. From this perspective, PageRanks of VIG can be highly related with the VSIDS. This is because the variables appearing in many clauses have many edges, which leads to high PageRanks. However, they have different views. The PageRank values are based on a global view. The value stands for a relative importance. In contrast, the VSIDS may focus on local parts of the instance structure. The scores are high if corresponding variables are current targets of the search. In fact, if the top two highest PageRank variables are placed on opposite sides of the VIG, one can have a high VSIDS score while the other can have a low score. Whereas the PageRank does not completely correspond to the VSIDS score, there is a high correlation between them.

Analysis Result

To confirm our inference, we have conducted experiments. We have implemented a PageRank calculation function to MiniSAT 2.2. We conducted 20 iterations for the PageRank calculation setting the teleportation ratio to 0.15. We chose variables of the top 10% highest PageRank and calculated a decision ratio (the number of decisions for these variables / the total number of decisions) for each instance. We used 300 instances from the SAT Competition 2014 application category. We set the time limit for each instance to 5000 seconds and the experiments were conducted on a Linux PC with Intel Core i7 4770 (3.40GHz, quad-core hyper-threading) and 16 GB memory. We used the GNU compiler (gcc) version 4.8.2.

The results are exhibited in Fig. 1 for 100 satisfiable instances and Fig. 2 for 76 unsatisfiable instances. We excluded instances that could not be solved within the time limit. The x-axis indicates each instance, the left y-axis indicates the decision ratio and the right y-axis indicates the processing time. The instances sorted by ascending order of the decision ratio. For satisfiable instances, 89 out of 100 instances exhibited over 10% decision ratio and 72 out of 76 unsatisfiable instances did. From these results, we found that the variables with high PageRank values tend to be selected as decision variables by the VSIDS. Hence there is a high correlation between them. In addition, the instances with the high decision ratio seem to be solved within a short processing time for both satisfiable and unsatisfiable instances.

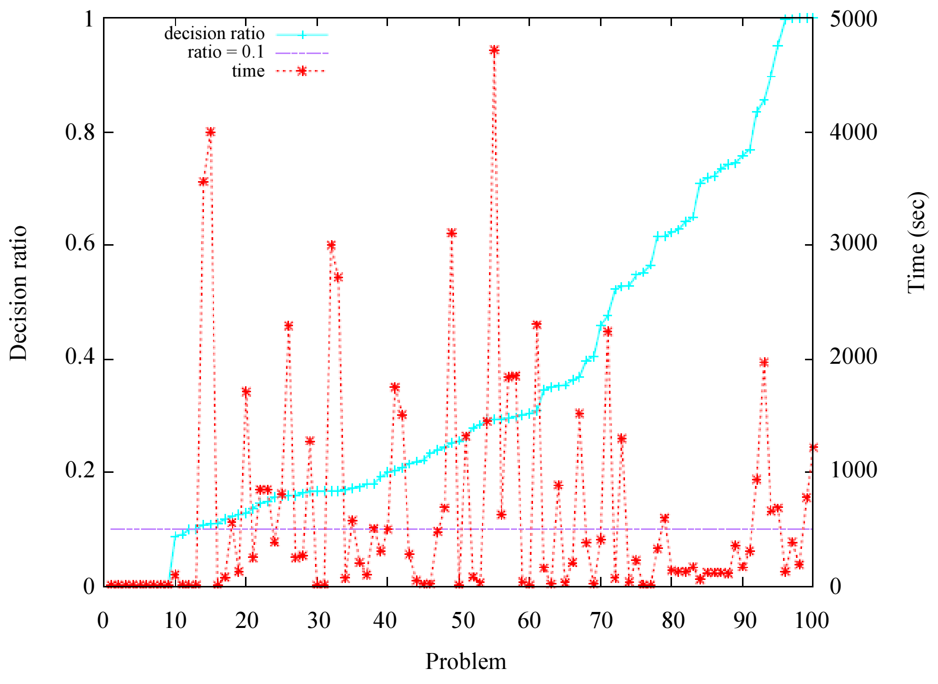

Fig. 1: Decision ratio of variables with top 10% highest PageRank and processing time for 100 satisfiable instances

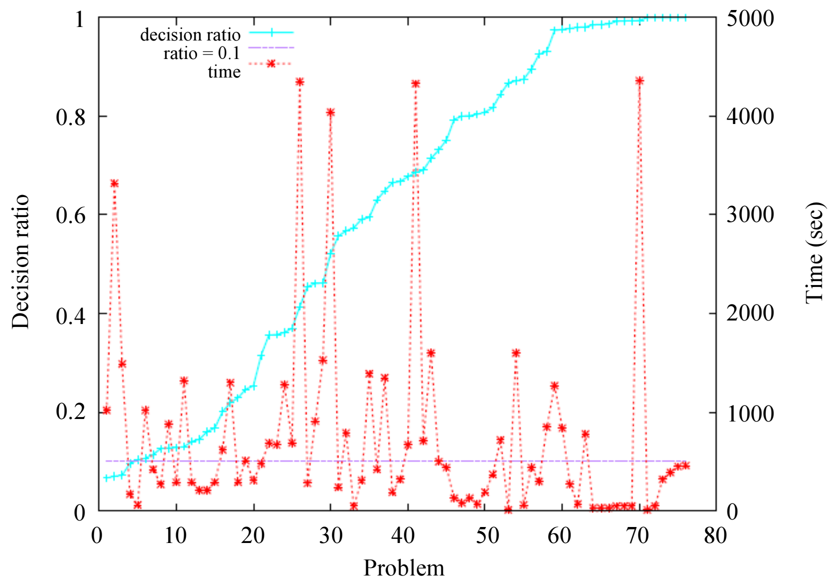

Fig. 2: Decision ratio of variables with top 10% highest PageRank and processing time for 76 unsatisfiable instances

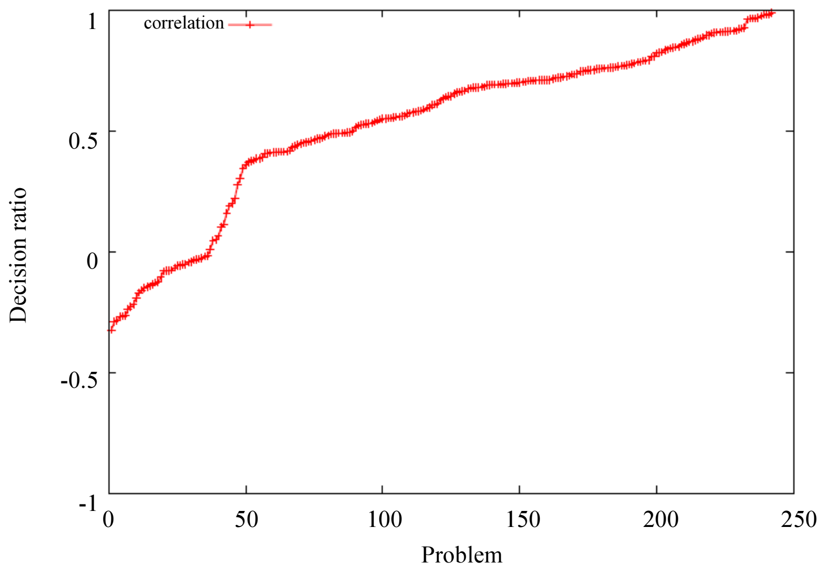

Fig. 3: Spearman's rank correlation coefficient between VSIDS and PageRank for 242 instances (x-axis indicates each instance and y-axis indicates the correlation coefficient)

However, we can see that the PageRank values does not completely coincide with the VSIDS values from Fig. 3. This figure shows the Spearman's rank correlation coefficient between the VSIDS and PageRank for each instance. Each value of each instance is calculated as follows. First, we compute ranks (positions in the ascending order) of VSIDS and PageRank for each variable after 100000 conflicts occur. Then we calculate Spearman's rank correlation coefficient between all the ranked variables. If the correlation coefficient is 1, the VSIDS and PageRank are completely related. We calculated them for 300 instances of the SAT Competition 2014 application category and showed the results in the Fig. 3 (we excluded 58 instances because they were solved within 100000 conflicts). From this figure, we can see that the VSIDS and PageRank hardly correlate in several tens of the instances. In fact, we observed negative correlation coefficients in 36 instances. The reason why the value is not so high is that the VSIDS and PageRank have different views. While the VSIDS focuses on variables in a specific part of the structure, the PageRank calculates the global importance of each variable. Thus, we should not simply replace the VSIDS with the PageRank. In the next section, we propose a method to combine them.

Proposed Method

The VSIDS tends to focus on a specific part of the structure in the given instance. This feature has advantages and disadvantages. While it can efficiently search variables that are strongly related, it could lead to a certain search spaces excessively. The restarting of the search can be a remedy of this issue, however the VSIDS scores are taken over to the search and the same search can repeat in vain because the scores are not changed after the restart. There is a possibility that the VSIDS guides the solver to a certain search space where no useful learnt clauses can be extracted.

We can solve this problem by combining the PageRank values with the VSIDS scores. PageRank has a global view and ranks all the vertices (variables). Hence this perspective enables the VSIDS to escape from the local structure and have a look at the global structure. However, in order not to spoil the advantage of VSIDS, we should not apply the PageRank to the VSIDS scores so frequently. We have to control the number of the applications. In addition, we should limit the application only for the variables with a high PageRank because the variables with a low PageRank are considered as not important. In order to convert the search space effectively, we should increase the VSIDS scores for the variables with a high PageRank and a low VSIDS score. For this issue, we first calculate rankings of the VSIDS scores and PageRank for each variable and increase the scores by considering those rankings.

We should also observe the distribution of the PageRank value. If the PageRank values are uniformly distributed, the variables have almost same importance and we have no idea which variables we should choose first. In our preliminary experiments, we figured out that the proposed method had almost no effect for instances with the uniform PageRank distribution. Thus, we first calculate the degree of unification of the PageRank and decide whether we use our proposal or not. The timing of the execution of the proposed method is suitable for the moment of restarting because our aim is to refresh the search. We implement Algorithm 2 right after the restarting routine.

The PageRank values are calculated outside. The “run_count” stands for the number of calls of this function. In fact, the main part of this function is conducted every “INTERVAL” restarts because this function modifies the VSIDS scores drastically. We control the number of the applications by adjusting this parameter. The “inc_score” is the degree of increment of the VSIDS scores, calculated by the solver. Note that the second argument of the function “increaseVSIDSScore” is just “inc_score” when clause learning is conducted. The function “calcRanking” returns the corresponding rank of each element of the given array (greater values rank higher). We calculate rankings of the PageRank and VSIDS score of each variable, “p_rank” and “a_rank”. The main part of this function increases the VSIDS scores of variables with top-(“nv”×“TARGET_RATIO”) PageRanks (e.g., if “TARGET_RATIO” is set as 0.05, 5% of variables). The function “selectTopKthIndex” returns an index of top *k*-th (the second argument) index of the given array (the first argument).

Algorithm 2: Pseudo code of the proposal method
Input: array of PageRank of each variable: pr
Input: array of VSIDS score of each variable: act
Input: the number of calls of this function: run_count
Input: incremental value for VSIDS: inc_score
Input: interval of this function: INTERVAL
Input: ratio of increment: INC_RATIO
Input: ratio of target variables: TARGET_RATIO

```
1:  nv = the number of variables
2:  p_rank // rank of variables w.r.t. PageRank
3:  a_rank // rank of variables w.r.t. VSIDS score
4:  if run_count % INTERVAL ≠ 0 then
5:      return
6:  end if
7:  p_rank = calcRanking(pr)
8:  a_rank = calcRanking(act)
9:  for k = 0 to nv * TARGET_RATIO do
10:     var_index = selectTopKthIndex(p_rank, k);
11:     if p_rank[var_index] < a_rank[var_index] then
12:         increaseVSIDSScore(var_index, inc_score
              * INC_RATIO * a_rank[var_index] /
              p_rank[var_index])
13:     end if
14: end for
```

By comparing the rankings of the PageRank and VSIDS score, we increase the VSIDS score of the target variable with a low VSIDS score and a high PageRank. For example, when $k = 0$, the “var_index” stands for the variable index with the highest (top-0th) PageRank. Then, “p_rank[var_index]” and “a_rank[var_index]” indicate the variable’s rank of PageRank and the rank of VSIDS score, respectively. The value “a_rank[var_index]”/“p_rank[var_index]” can be high when the PageRank is high and the VSIDS score is low. In this manner, we convert the search direction to other search spaces where important, but not focused on so far, variables exist from the point of view of the PageRank. We limit the number of the variables by setting the “TARGET_RATIO” because we do not have to observe all the variables (the variables with low PageRank are not important). We have to set the parameter “INC_RATIO” to relatively large number in order to increase the scores vigorously.

The procedure of PageRank calculation is in Algorithm 3. This function is called before Algorithm 2. Note that the function “calcPageRank” makes a VIG from the clauses in the given CNF and learnt clauses that the solver currently preserves. We conduct the power iteration method for the PageRank calculation.

```
Algorithm 3: Pseudo code of the PageRank calculation
Input: the number of calls of this function: run_count
Input: PageRank recalculation interval:
   REC_INTERVAL
 1: nv = the number of variables
 2: pr = [1.0/nv] * nv // PageRank array (global scope)
 3: fcp // first cumulative percentage
 4: if run_count % REC_INTERAVAL ≠ 0 then
 5:     return
 6: end if
 7: pr = calcPageRank(pr)
 8: fcp = calcFirstCumulativePercentage(pr, nv/100)
 9: if fcp < 3 then
10:     [turn off the proposed method]
11: end if
```

Since the calculation of PageRank is expensive, we limit the number of the power iterations to 5 and we limit the length of clauses to less than or equal to 10 for constructing VIGs. In practice, we do not need exact PageRank values in order to apply them for the VSIDS scores. Besides, this function is also executed every "REC_INTERVAL" restarts to reduce the computational cost.

We also consider the distribution of the PageRank value. The function "calcFirstCumulativePercentage" calculates the cumulative percentage of the PageRank value ("fcp") for the variables with top 1% highest PageRank. Let V_0 a set of variables with top 1% highest PageRank. Then:

$$fcp = 100 \times \sum_{u \in V'} \pi(u) / \sum_{v \in V'} \pi(v) \quad (3)$$

If this value is low, the distribution seems to be uniform. In an extreme case when "fcp" is less than three, we do not use our proposed method because we did not see any positive effect in our preliminary experiments.

The whole flow of the proposed method is as follows. After setting four parameters ("INC_RATIO", "INTERVAL", "REC_INTERVAL" and "TARGET_RATIO"), Algorithm 3 and 2 are called when a restart is invoked (Algorithm 1). Note that the variable "run_count" corresponds to the number of invoked restarts.

Although there are four parameters to determine the behavior of our method, we found that these values were not so sensitive to the performance of solvers in the preliminary experiments. The key idea is that we should use the PageRank value if its distribution is skewed and increase the score of the variables with a low VSIDS scores and a high PageRank value.

Results

We have implemented the proposed method in MiniSAT 2.2 and Glucose version 3 and conducted experiments. The experimental conditions are same as the previous section. However, there were 46 instances whose top 1% cumulative percentage of PageRank is less than three. We did not apply our method to these instances. Hence we excluded them from the results below.

We conducted comparison experiments with our methods, a randomized version of our method and MiniSAT 2.2. We set the "INC_RATIO" to 10000, "INTERVAL" to 10, "REC_INTERVAL" to 500 and "TARGET_RATIO" to 0.05 for the proposed method. The number of solved instances and their total time are shown in Table 1. Each column indicates the number of solved instances and its total time in seconds. The 46 instances whose "fcp" value is less than three, instances solved only by preprocessing and instances that were not solved by any solver are excluded. The instance not solved within the time limit is calculated as 5000. The "baseline", "proposal", "no-recalc" and "random" stands for the original MiniSAT 2.2, our proposal, our proposal without no recalculation of PageRank ("REC_INTERVAL" is ∞), our proposal with randomly selection of target variables and randomly increment of VSIDS scores, respectively. The "random" method is a modified version of our method that selects the variables randomly (according to the "TARGET_RATIO") with the same parameter setting of our proposal. From this result, we can see that the proposed method could solve the most instances within the shortest time. We can also see that we should recalculate the PageRank values in the search by using learnt clauses. By comparing "proposal" and "random", we figured out that using the PageRank values is more effective than at least random selection. Note that the total calculation time of the PageRank values was negligible for all the instances (0.77 seconds per solved instance on average).

Figure 4 (satisfiable instances) and 5 (unsatisfiable instances) show the result of each instance for the original MiniSAT 2.2 and our method. They show the processing time and the cumulative percentage of the PageRank values of the variables with the top 1% highest PageRank (the "fcp" value). The x-axis indicates each instance sorted by the time of the original MiniSAT 2.2 in ascending order. Hence the points on the same x-axis show the result for a same instance. The left y-axis indicates the time and the right y-axis indicates the cumulative percentage of PageRank. For satisfiable instances, although the total time was longer, we could solve two more instances than the baseline. This is because the original MiniSAT 2.2 is good at solving them intrinsically. Hence there are few rooms to improve the performance. In contrast, we could solve eight more unsatisfiable instances within a shorter time. We could see that instances with high "fcp" values (around 50%) could be solved in a short time.

We also implemented our method to Glucose 3. We set the "INC_RATIO" to 10, "INTERVAL" to 50, "REC_INTERVAL" to 5000 and "TARGET_RATIO" to 0.05 for our proposal. We set higher "INTERVAL" and "REC_INTERVAL" because Glucose 3 uses a dynamic restart policy (Audemard and Simon, 2012) and it can conduct the restart more frequently than MiniSAT 2.2. Table 2 shows the number of solved instances (total time) of each solver and Fig. 6 and 7 show the result of each instance. We can see that our proposal exhibits the best performance among the four solvers, same as in the case of MiniSAT 2.2. We could achieve good performance for the satisfiable instances and a little improvement for the unsatisfiable instances. This situation is opposite from MiniSAT 2.2. Glucose is good at solving unsatisfiable instances and has difficulties to solve satisfiable instances. In fact, there are few unsatisfiable instances not solved by original Glucose 3 except for the instances with a low "fcp" value.

Table 1: The result of each solver based on MiniSAT 2.2. for 71 satisfiable instances and 86 unsatisfiable instances

Solver	SAT (71)	UNSAT (86)	Total (157)
Baseline	67 (**52550**)	75 (109370)	142 (161920)
Proposal	**69** (52690)	**83 (85450)**	**152 (136140)**
No-recalc	67 (63070)	80 (91730)	147 (154800)
Random	67 (61210)	77 (117890)	144 (179100)

Table 2: The result of each solver based on Glucose 3 for 63 satisfiable instances and 126 unsatisfiable instances

Solver	SAT (63)	UNSAT (126)	Total (189)
Baseline	55 (68130)	**125 (90520)**	180 (158650)
Proposal	**60 (51920)**	**125** (90680)	**185 (142600)**
No-recalc	57 (59540)	**125** (91650)	182 (151190)
Random	57 (59780)	123 (92870)	180 (152650)

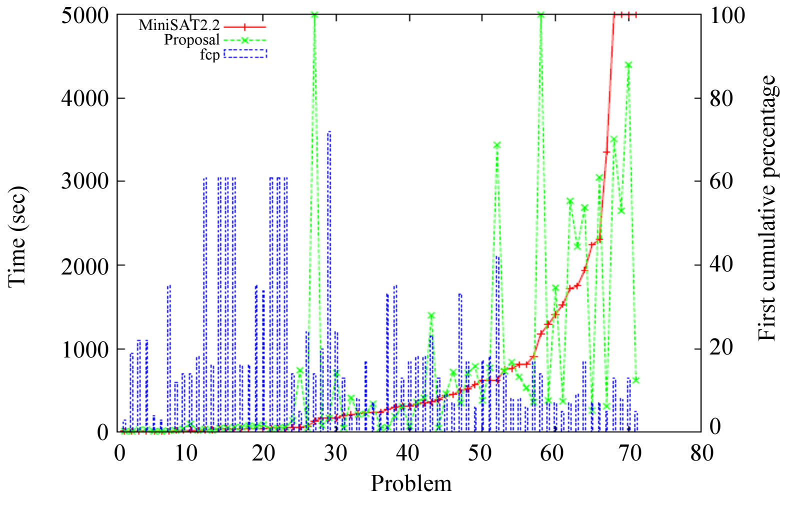

Fig. 4: The result of the original MiniSAT 2.2 and MiniSAT 2.2 with the proposed method for 71 satisfiable instances

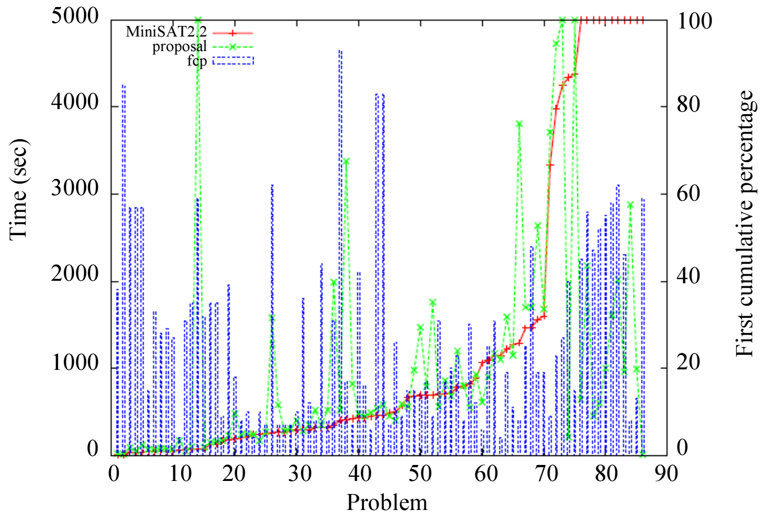

Fig. 5: The result of the original MiniSAT 2.2 and MiniSAT 2.2 with the proposed method for 86 unsatisfiable instances

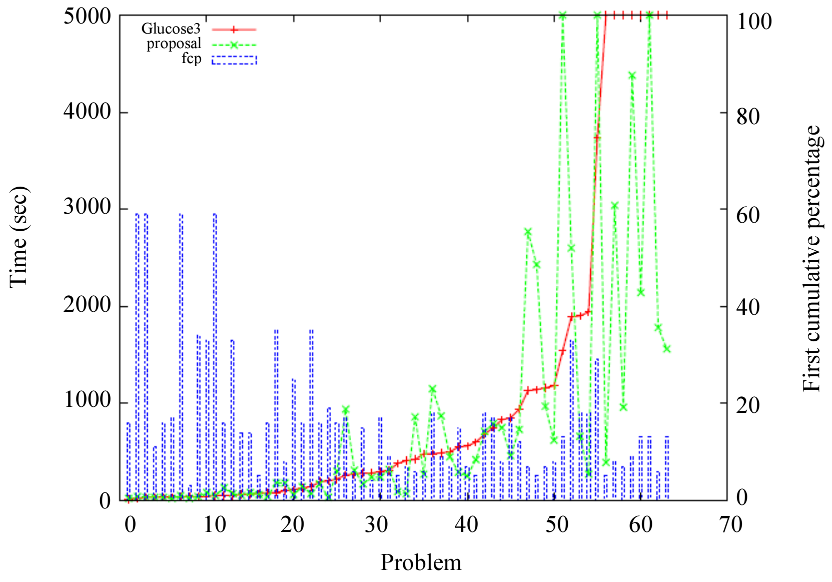

Fig. 6: The result of the original Glucose 3 and Glucose 3 with the proposed method for 63 satisfiable instances; Note that the instances solved before the "fcp" calculation (i.e., solved only by preprocessing) are not included

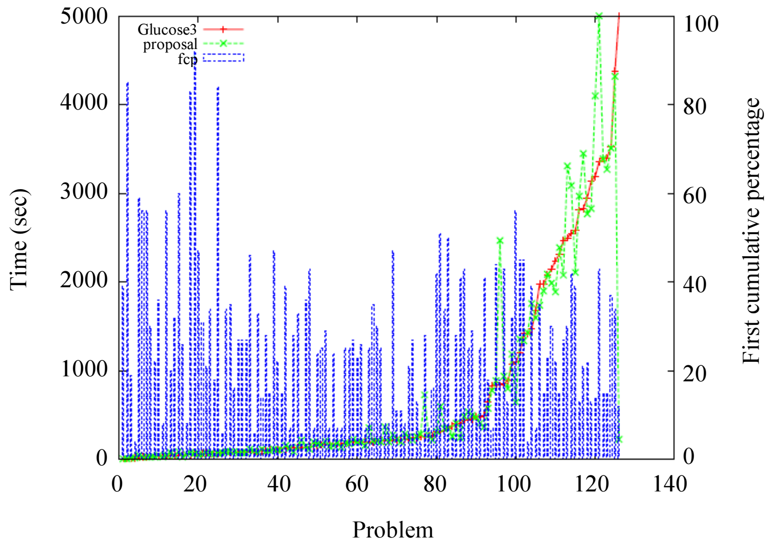

Fig. 7: The result of the original Glucose 3 and Glucose 3 with the proposed method for 126 unsatisfiable instances

Conclusion

We investigated the relation between PageRank and VSIDS and applied the PageRank value to the VSIDS score. From observational experiments, we found that variables with a high PageRank tend to be selected as decision variables. However, we also observed that they did not completely correlate when we saw the Spearman's rank correlation coefficient between them. It is because the VSIDS focuses on the local part of the structure of given instance, while PageRank gives the global view of the importance of Boolean variables. We utilize this advantage of the PageRank to convert the search space effectively by comparing the VSIDS and PageRank of each variable. In the computational experiments, we could boost the efficiency of MiniSAT 2.2 and Glucose 3. Our method does not depend on a specific implementation, thus we can embed it to any CDCL solver. We are planning to apply this method to parallel SAT solvers as future work.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number JP17K12742.

Ethics

This article is original and contains unpublished material. The corresponding author confirms that all of the other authors have read and approved the manuscript and no ethical issues involved.

References

Ansótegui, C., J. Giráldez-Cru and J. Levy, 2012. The Community Structure of Sat Formulas. In: Theory and Applications of Satisfiability Testing, Ganesh, V. (Ed.), Springer, Berlin, Heidelberg, ISBN-13: 978-3-642-31611-1, pp: 410-423.

Ansótegui, C., M. Bonet, J. Giráldez-Cru and J. Levy, 2014. The fractal dimension of sat formulas. Automated Reason., 6: 107-121. DOI: 10.1007/978-3- 319-08587-6_8

Audemard, G. and L. Simon, 2009. Predicting learnt clauses quality in modern sat solvers. Proceedings of the 21st International Jont Conference on Artifical Intelligence, Jul. 11-17, Pasadena, California, USA, pp: 399-404.

Audemard, G. and L. Simon, 2012. Refining Restarts Strategies for Sat and Unsat. In: Principles and Practice of Constraint Programming, Milano, M. (Ed.), Springer, Berlin, Heidelberg, ISBN-13: 978-3-642-33557-0, pp: 118-126.

Biere, A. and A. Fröhlich, 2015. Evaluating Cdcl Variable Scoring Schemes. In: Theory and Applications of Satisfiability Testing, Heule, M. and S. Weaver (Eds.), ISBN-10: 978-3-319-24318-4, pp: 405-422.

Bruni, R. and A. Santori, 2008. New updating criteria for conflict-based branching heuristics in dpll algorithms for satisfiability. Discrete Optimizat., 5: 569-583. DOI: 10.1016/j.disopt.2006.10.012

Clauset, A., M.E.J. Newman and C. Moore, 2004. Finding community structure in very large networks. Phys. Rev., 70: 6-36. DOI: 10.1103/PhysRevE.70.066111

Davis, M., G. Logemann and D. Loveland, 1962. A machine program for theorem-proving. Mach. Program Theorem-Prov., 5: 394-397. DOI: 10.1145/368273.368557

Eén, N. and N. Sörensson, 2003. An Extensible Sat-Solver. In: Theory and Applications of Satisfiability Testing, Giunchiglia, E. and A. Tacchella (Eds.), ISBN-10: 978-3-540-20851-8, pp: 502-518.

Gomes, C.P., B. Selman and H. Kautz, 1998. Boosting combinatorial search through randomization. Proceedings of the 15th International Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence, (AAI' 98), ACM, Madison, Wisconsin, USA, pp: 431-437.

Huang, J. and A. Darwiche, 2003. A structure-based variable ordering heuristic for sat. Proceedings of the 18th International Joint Conference on Artificial Intelligence, Aug. 09-15, ACM, Acapulco, Mexico pp: 1167-1172.

Jamali, S. and D. Mitchell, 2018. Centrality-Based Improvements to Cdcl Heuristics. In: Theory and Applications of Satisfiability Testing, Beyersdorff, O. and C.M. Wintersteiger (Eds.), Springer International Publishing, ISBN-13: 978-3-319-94143-1, pp: 122-131.

Katsirelos, G. and L. Simon, 2012. Eigenvector Centrality in Industrial Sat Instances. In: Principles and Practice of Constraint Programming, Milano, M. (Ed.), Springer, Cham, ISBN-13: 978-3-642-33557-0, pp: 348-356.

Liang, J.H., V. Ganesh, E. Zulkoski, A. Zaman and K. Czarnecki, 2015. Understanding Vsids Branching Heuristics in Conflict-Driven Clause-Learning Sat Solvers. In: Hardware and Software: Verification and Testing, Piterman, N. (Ed.), Springer, Cham, ISBN-13: 978-3-319-26287-1, pp: 225-241.

Liang, J.H., V. Ganesh, P. Poupart and K. Czarnecki, 2016. Exponential recency weighted average branching heuristic for sat solvers. Proceedings of the 30th AAAI conference on artificial intelligence, Feb. 12-17, ACM, Phoenix, Arizona, pp: 3434-3440.

Liang, J.H., H.G. Poupart, K. Czarnecki and V. Ganesh, 2017. An Empirical Study of Branching Heuristics through the Lens of Global Learning Rate. In: Theory and Applications of Satisfiability Testing, Gaspers, S. and T. Walsh (Eds.), Springer, Berlin, Heidelberg, ISBN-13: 978-3-319-66263-3, pp: 119-135.

Marques-Silva, J., 2008. Practical applications of boolean satisfiability. Proceedings of the 9th International Workshop on Discrete Event Systems, May 28-30, IEEE Xplore Press, Goteborg, Sweden, pp: 74-80. DOI: 10.1109/WODES.2008.4605925

Martins, R., V. Manquinho and I. Lynce, 2013. Community-Based Partitioning for Maxsat Solving. In: Theory and Applications of Satisfiability Testing, Järvisalo, M. and A. Van Gelder (Eds.), Springer, Berlin, Heidelberg, ISBN-13: 978-3-642-39070-8, pp: 182-191.

Monasson, R., R. Zecchina, S. Kirkpatrick, B. Selman and L. Troyansky, 1999. Determining computational complexity from characteristic phase transitions. Nature, 16: 133-137. DOI: 10.1038/22055.

Morrison, J.L., R. Breitling, D.J. Higham and D.R. Gilbert, 2005. Generank: Using search engine technology for the analysis of microarray experiments. BMC Bioinformat., 6: 21-29. DOI: 10.1186/1471-2105-6-233

Moskewicz, M.W., C.F. Madigan, Y. Zhao, L. Zhang and S. Malik, 2001. Chaff: Engineering an efficient sat solver. Proceedings of the 38th Annual Design Automation Conference, Jun. 22-22, IEEE Xplore Press, Las Vegas, NV, USA, pp: 530-535. DOI: 10.1145/378239.379017

Narodytska, N., S. Kasiviswanathan, L. Ryzhyk, M. Sagiv and T. Walsh, 2018. Verifying properties of binarized deep neural networks. Proceedings of the 32nd AAAI Conference on Artificial Intelligence, (CAI' 18), pp: 6615-6624.

Nejati, S., Z. Newsham, J. Scott, J.H. Liang and C. Gebotys *et al*., 2017. A Propagation Rate Based Splitting Heuristic for Divide-and-Conquer Solvers. In: Theory and Applications of Satisfiability Testing, Gaspers, S. and T. Walsh (Eds.), Springer, Cham, ISBN 978-3-319-66263-3, pp: 251-260.

Newsham, Z., V. Ganesh, S. Fischmeister, G. Audemard and L. Simon, 2014. Impact of Community Structure on Sat Solver Performance. In: Theory and Applications of Satisfiability Testing, Sinz, C. and U. Egly (Eds.), Springer, Cham, ISBN-13: 978-3-319-09283-6 pp: 252-268.

Page, L., S. Brin, R. Motwani and T. Winograd, 1999. The pagerank citation ranking: Bringing order to the web. The PageRank Citation Ranking: Bringing Order to the Web, Technical Report, Stanford InfoLab.

Pan, J., H. Yang, C. Faloutsos and P. Duygulu, 2004. Automatic multimedia cross-modal correlation discovery. Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, May 5-6, ACM Las Vegas, NV, USA, pp: 653-658. DOI: 10.1007/978-3-540- 24775-3_62

Selsam, D. and N. Bjørner, 2019. Guiding High-Performance Sat Solvers With Unsat-Core Predictions. In: Theory and Applications of Satisfiability Testing, Janota, M. and I. Lynce (Eds.), Springer, Cham, ISBN-13: 978-3-030-24258-9, pp: 336- 353.

Silva, J.P.M., 1999. The Impact of Branching Heuristics in Propositional Satisfiability algorithms. In: Progress in Artificial Intelligence, Barahona, P. and J.J. Alferes (Eds), Springer, Berlin, Heidelberg, ISBN-13: 978-3-540-48159-1, pp: 62-74.

Sonobe, T., S. Kondoh and M. Inaba, 2014. Community Branching for Parallel Portfolio Sat Solvers. In: Theory and Applications of Satisfiability Testing, Sinz, C. and U. Egly (Eds.), Springer, Cham, ISBN-13: 978-3-319-09283-6, pp: 188-196.

Stephan, P., R.K. Brayton and A.L. Sangiovanni-Vincentelli, 2006. Combinational test generation using satisfiability. Trans. Comput. Des. Integrat. Circu. Syst., 15: pp: 1167-1176. DOI: 10.1109/43.536723

Williams, R., C.P. Gomes and B. Selman, 2003. Backdoors to typical case complexity. Proceedings of the 18th international Joint Conference on Artificial Intelligence, Aug. 09-15, ACM, Acapulco, Mexico, pp: 1173-1178.