Boost SAT Solver with Hybrid Branching Heuristic

Seongsoo Moon, Mary Inaba
Graduate School of Information Science and Technology,
The University of Tokyo, Yayoi 1-1-1, Bunkyo-ku, Tokyo, Japan
E-mail address: logic85@hotmail.com, mary@is.s.u-tokyo.ac.jp

Abstract

Most state-of-the-art satisfiability (SAT) solvers are capable of solving large application instances with efficient branching heuristics. The VSIDS heuristic is widely used because of its robustness. This paper focuses on the inherent ties in VSIDS and proposes a new branching heuristic called TB-VSIDS, which attempts to break the ties with the consideration of the interplay between the branching heuristic and learned clauses. However, a branching heuristic cannot cover all problems, and its performance improves when combined with an appropriate configuration. Therefore, we also propose a hybrid model of branching heuristics based on random forest. The efficiencies of TBVSIDS and hybrid branching heuristics are evaluated on benchmarks in SAT Competitions. By constructing a model that reduces the overfitting problem, we hope to realize a hybrid branching heuristic that is widely applicable to other solvers.

Introduction

The satisfiability (SAT) problem is a well-known NP-complete problem; that is, it cannot be solved in polynomial-time. Despite the lack of polynomial-time solutions, SAT algorithms have substantially progressed in recent years, and state-of-the-art SAT solvers can rapidly solve software verification problems, puzzles, planning, and other application problems. One of the most influential speed-enhancers of SAT solvers is the branching heuristic, most prominently represented by the *variable state independent decaying sum* (VSIDS) (Moskewicz et al. 2001). Several researchers (Goldberg and Novikov 2007; Dershowitz, Hanna, and Nadel 2005; Liang et al. 2016a) have designed heuristics that outperform VSIDS, but VSIDS remains popular because of its robustness. Several variants of VSIDS (Biere and Fröhlich 2015) have also been proposed, but most of these variants are designed to smooth the scores after conflicts. As far as we know, nobody has actually measured tie occurrences in VSIDS and paid attention to break ties.

In this paper, we raise awareness of the frequent tie occurrences inherent in VSIDS., and propose a tie-breaking method called *Tie-breaking of VSIDS* (TBVSIDS). Because a single branching heuristic cannot generally handle the wide and expanding range of practical applications of SAT solvers, we were motivated to improve the branching heuristic by a hybrid strategy that combines several branching heuristics.

We first show the effectiveness of the hybrid strategy in our preliminary version constructed by combining two branching heuristics. Next, we construct a model by random forest. For this purpose, we prepare several branching heuristics and extract several features of the SAT formula. The objective of our model is to reduce the performance gap between the virtual best solver (VBS) and a hybrid branching heuristic, while avoiding overfitting. The VBS is hypothetically optimized for all instances and selects the best policy among several policies provided at each instance.

The remainder of this paper discusses related work, explains our proposals, and presents the experimental results. The paper ends with some concluding remarks.

Contributions

Our study makes two major contributions to SAT technology:

- Inspired by the inherent tie occurrences in VSIDS, we measured these tie occurrences and developed a new branching heuristic called TBVSIDS that breaks the ties. To our knowledge, tiebreaking has not been accomplished to date. Moreover, the heuristic is potentially applicable to other branching heuristics.
- Second, we develop a hybrid branching heuristic based on a random forest model. Because we used a single solver, our hybrid heuristic can provide a base solver for other solvers. To reduce the learning time of the model and render it universally applicable to large instances, we proposed feature extraction by random sampling. To our knowledge, random sampling has never been applied to feature extraction for SAT formulas.

Related work

Branching heuristics

Most SAT solvers find a solution by a backtracking search. During backtracking, a branching heuristic selects an unassigned variable and assigns it as true or false. The selection of the next variable for branching significantly affects the search efficiency. To pick a variable, branching heuristics invoke ranking functions that maintain a map of scores

corresponding to each variable. The VSIDS (Moskewicz et al. 2001) is the most representative branching heuristic, and has been widely used for a long time owing to its robustness, which has been demonstrated on benchmarks over the years. The score map is updated at every conflict. When a conflict occurs, the score of the variables related to that conflict is incremented by one.

Recently (Liang et al. 2016a) proposed the conflict history-based branching heuristic (CHB), which updates the scores based on rewards calculated by the conflict history. The CHB adopts the concept of reinforcement learning. The same authors also proposed learning rate branching (Liang et al. 2016b), which try to maximize learning rate. The score of the variable is updated when it is unassigned. A variable gets higher score the more it is participated in analysis of conflicts between an assignment and unassignment. If their robustness could be demonstrated in a variety of instaces, these branching heuristics might replace VSIDS.

The present paper focuses on tie occurrences in branching heuristics, and proposes a method for breaking these ties. Especially, we propose *Tie-breaking of VSIDS* (TBVSIDS), which is designed for VSIDS.

Hybrid strategies

Parallel SAT solvers often employ multiple strategies to diversify the search. In contrast, sequential solvers normally adopt a single strategy to intensify the search. Several studies have attempted to integrate different strategies in a sequential solver. For example, the multi-solver SATzilla (Xu et al. 2007) builds an empirical model using machine learning techniques and chooses an adequate solver for each problem based on its feature values. A deep learning approach has also been attempted (Loreggia et al. 2016). This approach converts a CNF into a grayscale image and builds a classifier using a convolutional neural network. However, these methods require several state-of-the-art solvers. They achieve higher performance than single solvers, but are unsuitable as base solvers for other solvers because they already include several base solvers.

Noting that a slow Luby restart policy is superior to rapid restart policies for SAT problems, (Oh 2015) designed a hybrid restart strategy. Such approaches could be combined with other approaches.

Tie-breaking

In this section, we first demonstrate tie occurrences. If frequent ties are broken appropriately, the search efficiency might improve. We then propose a tie-breaking method and evaluate it through benchmarks of SAT Competitions.

Tie occurrences

Our objective is to observe the tie frequency in a branching heuristic. For this purpose, we ran Glucose (Simon and Audemard 2009) on 300 benchmarks from the SAT Competition 2014 Application Track. The time limit was set to 1,000 s for each instance. Figure 1 shows the tie occurrences and ratios in each instance. The gray curve represents the tie ratios sorted in ascending order. Instances solved within 1,000 s were excluded because they were too short to calculate the tie ratios. We counted the presence of ties on every decision in the branching heuristic. The numbers of ties and decisions ranged from 2 million to 200 millon, so were plotted on a logarithmic scale. This figure confirms that ties frequently occur in some instances. The median and mean ratios were 0.05 and 0.19, respectively, meaning that ties occur once per five conflicts on average. Therefore, improving the tie-breaking policy is a worthwhile exercise, at least in VSIDS.

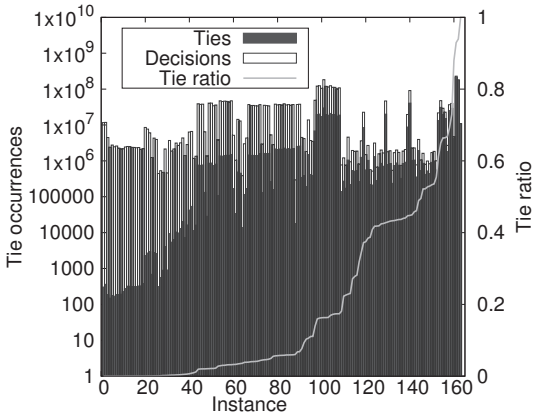


Figure 1: Tie occurrences in instances from the SAT Competition 2014 Application Track.

Proposal of TBVSIDS

In most SAT solvers with VSIDS, ties are broken randomly. In VSIDS, all variables related to resolution are incremented by 1. Our proposal provides a bonus to variables in a learned clause. We initially considered that the length of the learned clause would be a good indicator. Short clauses are more informative than long clauses in general, therefore it would be reasonable to provide more bonuses to variables in the short clauses. We implemented this idea in MiniSat (Sorensson 2010) and it worked well. However, when applied in Glucose (Simon and Audemard 2009), the results were poor. Shortly afterward, we realized that we should consider the interplay between the branching heuristic and clause learning. In Glucose, we thus replaced the length of a learned clause with the literal blocks distance (LBD) index (Simon and Audemard 2009), by which Glucose assesses a learned clause. Adding more bonuses to variables in a learned clause with smaller LBDs would improve the search within the learning scheme of Glucose. We have devised two tie-breaking methods, TBVSIDS1 and TBVSIDS2. Owing to limited space, we describe only TBVSIDS2 in detail. In the TBVSIDS2 algorithm (algorithm 1), *activity* is a floating map for the original VSIDS. TBVSIDS1 has two floating maps; *activity* for VSIDS and *activityQ* for tie-breaking. However, TBVSIDS2 directly adds small bonuses of variables in a learned clause to *activity*. We proposed TBVSIDS2 because it is more agile than TBVSIDS1 because it only updates one floating map, and we considered providing small bonuses would reduce the tie occurrences.

Algorithm 1 TBVSIDS2 branching heuristic.

for $v \in$ Vars **do**
  $activity[v] \leftarrow 0$
end for
loop
  if a conflict occurs **then**
    for $v \in$ variables resolved in conflict analysis **do**
      $activity[v] \leftarrow activity[v] + 1$
    end for
    for $v \in lc$ (*lc: learned clause*) **do**
      $quality(lc) \leftarrow 1 / dist(lc)$
      $activity[v] \leftarrow activity[v] + quality(lc)$
    end for
  else
    $unassigned \leftarrow$ unassigned variables
    $v^* \leftarrow argmax_{v \in unassigned} activity[v]$
  end if
end loop
return v^*

Experimental results

The experimental environment was a Xeon X5680 3.3 GHz CPU, 12 physical cores with 140 GB RAM. All experiments in this paper were tested in the same environment.

We ran Glucose (Simon and Audemard 2009) on 900 benchmarks from the SAT Competitions. Figure 2 compares the tie occurrences between VSIDS and TBVSIDS2 in a scatter plot. The time limit was set to 1,000 s for each instance. Instances solved in either VSIDS or TBVSIDS within a time limit were excluded for comparison of tie ratios between the different branching heuristics. As described in Figure 2, we find that tie occurrences in TBVSIDS2 were reduced in 342 out of 469 instances (72.9%). We ran these benchmarks with time limit 5,000 s and described in Table 1. In 900 benchmarks, TBVSIDS2 solved 14 more instances than VSIDS.

We measured the qualities of the learned clauses through their sizes and LBDs during searches to observe further. We considered a SAT solver with more intensive search would produce longer clauses. If a solver finds the learned clauses within the restricted areas, then the search goes deeper and deeper deriving longer clauses. Figure 3 compares the LBD distribution of the learned clauses between VSIDS and TBVSIDS2. The lowest LBD for clauses is two, there for comparison starts at $x = 2$. For example, two boxes at $x = 2$ indicate the sum of all the number of $LBD = 2$ clauses from all instances for VSIDS and TBVSIDS2 respetively. VSIDS found more clauses than TBVSIDS2 when their LBDs are under 10 and less clauses when LBDs between 11 and 15. For lack of space, we describe only LBDs distribution, but the similar results were obtained when we observed their sizes instead of LBDs. Short clauses are generally more informative than long clauses because they might invoke rapid propagations. However as we shown in Figure 3 and 1, increasing the number of more informative clauses does not necessarily improve the performance of a solver.

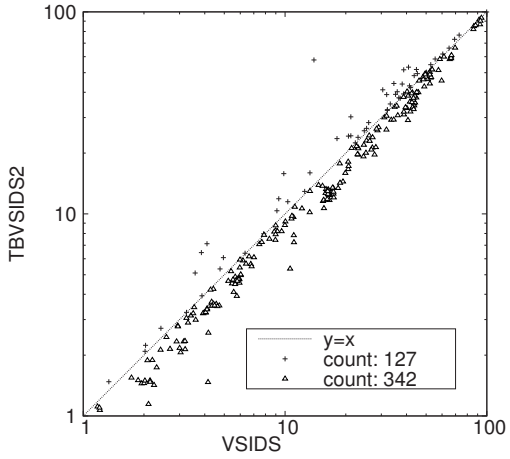


Figure 2: Comparison of tie occurrences between VSIDS and TBVSIDS2

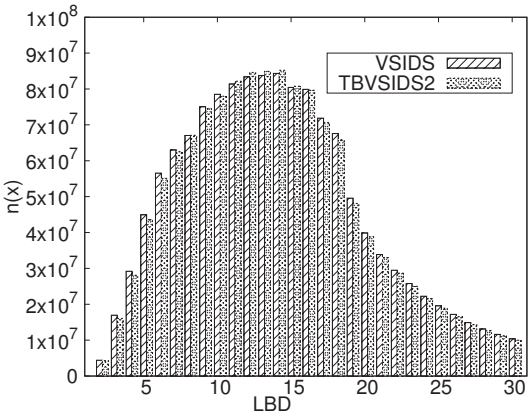


Figure 3: Comparison of LBD distribution between VSIDS and TBVSIDS2. The $n(x)$ indicates the number of learned clauses found from benchmarks, where x is LBD.

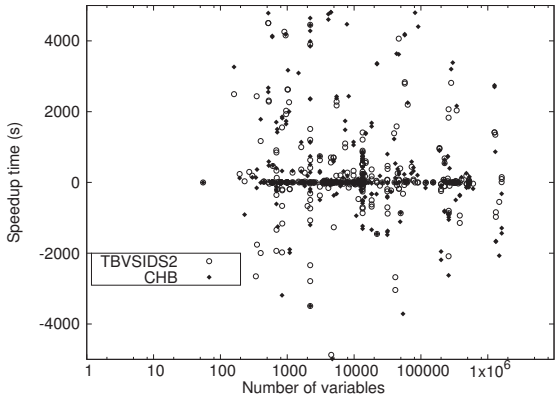
Hybrid branching heuristic - static method

This section describes a static method using TBVSIDS and a recently proposed CHB (Liang et al. 2016a). This method is preliminary to constructing a hybrid model by random forest. TBVSIDS and CHB use the same data structure as VSIDS, so are easily implemented in a solver.

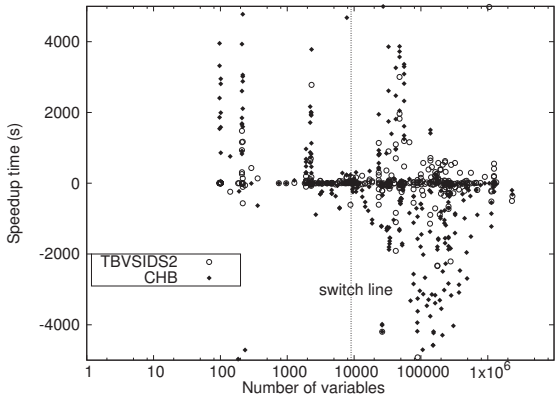
We propose a hybrid model because a single branching heuristic cannot cover all instances. The most appropriate heuristic for each instance depends on the traits of its formula and the interplay among the algorithms such as the currently learned clauses and the restart policy. Optimizing a dynamic solution that switches among branching heuristics is a difficult task. However, we can construct a model in advance, and apply it as a preprocessing method for selecting a branching heuristic.

Difference between TBVSIDS and CHB

The performances of TBVSIDS and CHB were compared for varying number of variables in the SAT formula. The



(a) SAT



(b) UNSAT

Figure 4: Speedup of CHB and TBVSIDS2 over VSIDS

speedups of CHB and TBVSIDS over VSIDS versus number of variables are presented in Figure 4. Panels (a) and (b) of this figure plot the performances in SAT and UNSAT instances, respectively.

The results show that CHB and TBVSIDS2 are relatively distant from and close to VSIDS, respectively. In UNSAT, CHB apparently performs well when the input formula has a small number of variables. Therefore, we considered that CHB and TBVSIDS2 can be substituted for VSIDS when the number of variables is small and large, respectively.

Experimental results

We implemented CHB in Glucose and selected a branching heuristic as a preprocessing method. This method counts the number of variables, and selects CHB if the variables are fewer than 9,000 (switch line in Figure 4.(b)); otherwise, it selects TBVSIDS as the branching heuristic. The results are shown in Table 1 and Figure 5. Both TBVSIDS2 and CHB outperform VSIDS, but the differences are not large. The results were improved by applying a hybrid method with an extremely simple policy. In fact, the results of the hybrid method exceeded our expectations. TBVSIDS2 and CHB solved 14 and 16 more instances than VSIDS, respectively. Therefore, we considered the gap between VSIDS and hybrid heuristic would be 30 at most (the arithmetic sum of 14 and 16). The actual gap was 47, indicating that TBVSIDS2 and CHB are complementary algorithms.

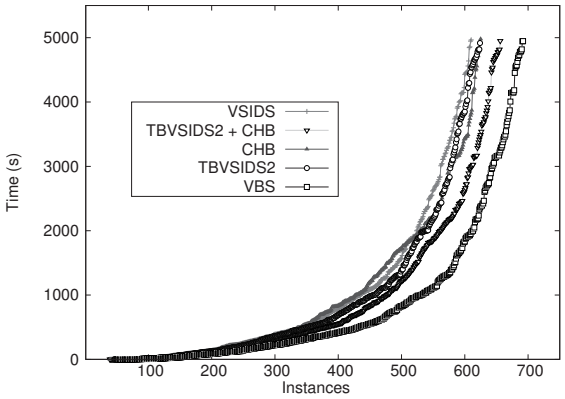


Figure 5: Cactus plot of 900 instances from SAT competitions

Table 1: Solved instances from 900 instances of SAT Competitions. Column U, T2, C, and V denote unmodified original VSIDS, TBVSIDS2, CHB, and VBS, respectively. Rows C and A denote Crafted Track and Application Track, respectively.

Solver		U	T2	C	T2+C	V
2014C	SAT	79	81	85	83	89
	UNSAT	82	88	99	111	114
	BOTH	161	169	184	194	203
2014A	SAT	100	102	102	103	108
	UNSAT	115	112	102	115	123
	BOTH	215	214	204	218	231
2015A	SAT	137	143	146	146	152
	UNSAT	101	102	96	103	107
	BOTH	238	245	242	249	259
TOTAL	SAT	316	326	333	332	349
	UNSAT	298	302	297	329	344
	BOTH	614	628	630	661	693

VBS in Figure 5 was constructed by a combination of CHB and TBVSIDS2. In the following section, we improve the VBS and the hybrid model by applying machine learning methods with several branching heuristics. to improve the VBS and a hybrid model in the following section.

Hybrid branching heuristic - random forest

In this section, we propose applying a random forest model to build a hybrid branching heuristic. We first mention why and how to apply a random forest. Next, we extract relatively simple 13 features and evaluate its models. Then we propose random sampling for SAT formulas to achive fast feature calculations. Finally, we expand feature number to 23 by applying random sampling, and evaulate our model experimentally.

Motivation & Approach

As mentioned above, a single algorithm cannot cover all SAT problems. To integrate the algorithms and boost the per-

formance of SAT solvers, researchers have proposed several algorithm selection strategies (Xu et al. 2007; Loreggia et al. 2016). Let us consider the integration of N SAT solvers (S_1, S_2, ... , S_N) with different strategies, such as restart, learning scheme, and learned clause evaluation, into a solver I. We then optimize I.

To improve I with a new policy P, we must implement P in each S_i and evaluate each case. In addition, after updating some S_i with P, we must rebuild the model for I. Applying and evaluating a new method seems to require much effort. Our final goal is to propose a base solver I with high performance and base-solver capability for other solvers such as MiniSat (Sorensson 2010) or Glucose (Simon and Audemard 2009). Such a base solver would allow continuous improvements of SAT solvers.

Existing algorithm selection strategies are only concentrated to improve their performances. They cannot be a base solver for other solvers. We considered to improve the performance of a solver by focusing on only a small part and improving that part within a reasonable timeframe. The improved solver can then be used as a base solver for other solvers. Branching heuristics is an appropriate candidate for this purpose. The recently proposed CHB and TBVSIDS can be easily implemented by using the data structure of VSIDS. This implies that these branching heuristics can be easily integrated into a solver. We constructed a random forest model that allocates an appropriate branching heuristic by training several features in an original formula.

Experimental results - 13 features

We trained our model on 1400 benchmarks of SAT Competitions from 2014 to 2016 in both the Crafted and Application Tracks. The random forest model was constructed from 13 features: vars (number of variables), clauses (number of clauses), vars/clauses, and variable-clause graph features (mean, variation coefficient, min, max, and entropy for both variable and clause node degrees). These features can be extracted within a short time without requiring a specific algorithm. The calculation time of 1400 benchmarks was under 1,000 s. This is important because when solving a formula using a SAT solver, our model must extract features as a preprocessing step. Therefore, time-consuming feature extraction is undesirable. Total of eight diffrent branching heuristics were used as classes, which are VSIDS, CHB, TBVSIDSs, and TBCHBs. We have two versions of TBVSIDS. For TBVSIDS2, we assigned a different parameter for *quality* calculation at line 10 in Algorithm 1, yielding two different TBVSIDS2s. We also implemented three versions of *Tie-breaking of CHB* (TBCHB) such as TBVSIDSs.

Our goal is to maximize the performance of the SAT solver while minimizing the overfitting problem. To reduce the overfitting, we limited the tree height in random forest to 5. We extracted 13 features for each instance, but more features do not ensure better results. Features with no relations will not improve the performance of SAT solvers. Therefore, we constructed all possible combinations of 8192 models by activating/inactivating the use of each feature in Glucose. Partial results are illustrated in Tables 2, 3, and 4. Each table shows the test results of several training data. For example, in Row 7 of each table, the model was trained on all benchmarks except those of Crafted Track in SAT Competition 2014, then tested on each track. The results in Table 2 were obtained using all 13 of the abovementioned features. When the performances of classifiers were evaluated by k-fold cross validation (Training data: $\hat{\text{A}}$X | $\hat{\text{C}}$X, Test data: AX | CX, where X = 14 | 15 | 16), the classifier in Table 2 showed the best performance, but surprisingly, used only one feature. The model in Table 3 was selected by our objective function f stated below, where X is one of the test data (AX or CX) and $N(X, Y)$ is the number of solved instances in test dataset Y when the model was trained by dataset X.

$$f = minimize(A - B) \tag{1}$$

$$A = \sum (N(X, X) - \alpha \times N(\hat{X}, X)) \tag{2}$$

$$B = \sum \sum (N(\hat{X}, Y)) \tag{3}$$

We explain the concept of our formulas. When the training data are used as the test data, i.e.,$N(X, X)$, the performance is high, but the performance of $N(\hat{X}, X)$ is low because it is trained exactly without the test data. Therefore, we seek to mimimize the gap between $N(X, X)$ and $N(\hat{X}, X)$ to achieve a good model. We also desire to reduce the gaps between the VBS results and the sum of $N(\hat{X}, X)$. Because the VBS results are fixed, they are excluded from the formulas. To combine these two ideas, a parameter α is added in Formula 1. Here we set $\alpha = 2$. In this subsection, VBS denotes a hypothetical solver selected from the eight branching heuristics mentioned above.

We named the classifiers in Tables 2, 3, and 4 as *all*, *k-fold*, and *f*, respectively. Let *n* be the number of solved instances when all instances were used as the training data and evaluated on themselves. The correlation coefficient of the performances between *f* and *n* was -0.59. Therefore, the performance of a single solver could be improved by minimizing *f*. The correlation coefficient between *k-fold* and *n* was 0.11, too low to claim a relation between these performances.

We further considered the expandability of our model. For this purpose, we trained our model using the benchmarks results on Glucose. If the model performs efficiently when applied to another solver, the model has likely reduced the overfitting problem and is satisfactorily robust. Therefore, we tested our models with another SAT solver, abcdSAT (Chen 2016), which won the Main Track in SAT-Race 2015. The performances of Glucose and abcdSAT are proved quite different; out of 1400 instances, 169 instances were solved by one solver but not by the other. Thus, we can assess the expandability of our model by applying it to abcdSAT.

Tables 5 and 6 compares the results of different models in abcdSAT and Glucose respectively. Our models were trained by the Glucose results and applied to both Glucose and abcdSAT. All of our models performed reasonably in abcdSAT, although the improvements were smaller than in Glucose. We also demonstrated a hypothetical model f_r which is robust in both results. At this time, f_r was found by a brute-force approach, namely, by traversing all 8192 models. This

Table 2: Test results with several training datasets using all 13 features. Columns: Training data. Rows: Test data. C: Crafted Track and A: Application Track. ÂX = all - AX. ĈX = all - CX, where X = 14 | 15 | 16.

	C14 (300)	A14 (300)	A15 (300)	C16 (200)	A16 (300)	all (1400)
C14 (214)	**204**	198	244	29	137	812
A14 (231)	174	**215**	244	39	139	811
A15 (261)	159	212	**251**	42	137	801
C16 (65)	152	204	224	**65**	130	775
A16 (157)	169	206	240	35	**151**	801
Ĉ14 (714)	**167**	217	249	61	143	**837**
Â14 (697)	198	**211**	247	62	143	**861**
Â15 (667)	201	213	**245**	63	145	**867**
Ĉ16 (863)	204	214	246	**38**	142	**844**
Â16 (771)	201	213	249	63	**139**	**865**
all (928)	202	213	247	63	143	868

Table 3: Test results with several training datasets using only one feature in a variable-clause graph (variable nodes: max)

	C14 (300)	A14 (300)	A15 (300)	C16 (200)	A16 (300)	all (1400)
C14	**197**	201	234	23	133	788
A14	159	**214**	241	43	136	793
A15	194	214	**249**	36	138	831
C16	139	180	199	**63**	125	706
A16	182	207	236	36	**140**	801
Ĉ14	**180**	218	247	56	142	**843**
Â14	196	**211**	245	56	139	**847**
Â15	196	215	**246**	56	142	**855**
Ĉ16	202	216	247	**51**	142	**858**
Â16	193	215	248	56	**137**	**849**
all	196	214	244	56	139	849

approach was justified because 13 features are rapidly extracted and traversing 8192 models consumes little time. In the following subsections, we increase the number of features and propose a more refined approach that finds a better model than f_r.

Random sampling

In previous section, we extracted only 13 features from each SAT formula. Extracting more features by constructing a variable graph or a clause graph is time-consuming and infeasible when the numbers of variables and clauses are very large. We considered that several features, such as average and entropy, will be conserved even when the computations are reduced by random sampling. The present experiments assess the possibility of random sampling of SAT formulas.

Figure 6 shows the correlation coefficients of feature extraction between the original formula and a randomly sampled formula. Three features (vars, clauses, and vars/clauses) were excluded because they do not need computational times. The sampling ratio was fixed at 0.1; meaning that 90% of the variables are removed from the original formula. If a clause includes several of the removed variables, then it

Table 4: Test results with several training datasets using seven features in a variable-clause graph (vars, clauses, vars/clauses, variable nodes: variation coefficient, min, max, and entropy)

	C14 (300)	A14 (300)	A15 (300)	C16 (200)	A16 (300)	all (1400)
C14	**202**	199	244	37	132	814
A14	174	**217**	251	38	134	814
A15	157	211	**252**	39	139	798
C16	160	199	214	**65**	116	754
A16	178	204	239	39	**144**	804
Ĉ14	**177**	216	251	62	140	**846**
Â14	203	**212**	251	63	141	**870**
Â15	202	214	**249**	59	141	**865**
Ĉ16	204	215	249	**37**	142	**847**
Â16	204	215	250	63	**141**	**873**
all	203	216	249	63	143	874

Table 5: Solved instances from SAT Competitions using abcdSAT. Row Average denotes the expected average performance of abcdSAT when a branching heuristic is randomly selected from 8 different branching heuristics.

Solver	C14	A14	A15	C16	A16	all
VBS	180	237	267	46	153	883
Average	162.9	221.3	254.5	37.8	141	817.5
all	168	228	256	37	143	832
k-fold	170	229	259	35	146	839
f	169	227	254	34	143	827
f_r	169	231	266	41	146	853

shrinks, and if all variables of a clause are removed, then the clause is also removed. The sampling method was applied when the formula included more than th variables, where th is a pre-set threshold. The threshold was imposed to alleviate concerns that several formulas are so small that sampling may hide their features. Each point was calculated 10 times and averaged.

Most of the coefficients were very high. The exception was cvMin at $th = 0$, because the average value of cvMin in the original formulas was very low (1.46). When applying the sampling method to all formulas, all of the cvMin values become 1.0, so the coefficient vanishes. The coefficients at vcMax became low sometimes because very few variables are connected to a large number of clauses, and if these variables are removed by random sampling, then the

Table 6: Solved instances from SAT Competitions using Glucose.

Solver	C14	A14	A15	C16	A16	all
VBS	214	231	261	65	157	928
Average	158.4	196.4	227.1	41.4	128.9	752.2
all	202	213	249	63	143	868
k-fold	196	214	244	56	137	849
f	203	216	249	63	143	874
f_r	205	215	249	62	139	870

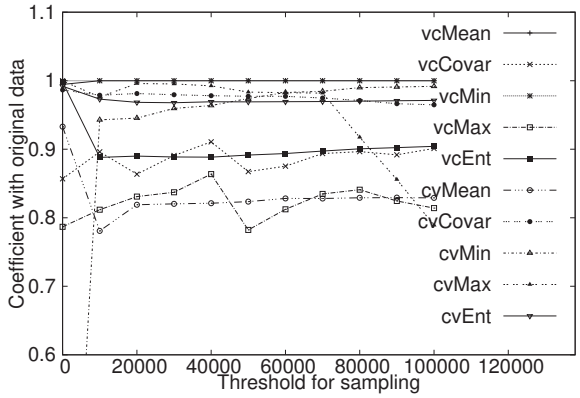


Figure 6: Correlation of features between original and sampled formulas

vcMax value becomes very small.

Overall, the performance of the sampling method was reasonable. Further studies of this approach might strengthen our intuition; in any case, the approach usefully reduces the time of extracting extra features.

Experimental results - 23 features

Finally, we expanded the feature number from 13 to 23 and applied the abovementioned random sampling method. The additional 10 features were obtained from the variable graph (mean, variation coefficient, min, max, and entropy of both node degree and diameter). The most time consuming task is extracting the diameters. For V nodes is V and E edges, diameter extraction by linked list and BFS required $O(VE)$ of runtime. From the relation $E = k \times V$ derived from the variable graph, the time of extracting the diameters is proportional to $k \times V^2$. By extracting only d_t diameters, where $d_t = N/(k \times V^2)$ and $N = 10^{10}$, we extracted 23 features for 1,400 benchmarks in under 7 hours. The longest time of each instance was 6 minutes, which is reasonable for SAT solvers. Generally, these features cannot be computed within a reasonable timeframe, because many instances have a huge variable graph. When investing over 300 SAT formulas from the SAT Competition 2014 Application Track, 116 instances contained over 10^5 variables, and 26 contained over 10^6 variables.

The 23 features extracted by random sampling must then be validated. For this purpose, we constructed random forest models from these features and benchmarks results on Glucose. Figure 7 is a flow chart of the genetic algorithm that finds a robust random forest model in both Glucose and abcdSAT SAT solvers. The features for training and evaluation were generated by different random seeds. Different seeding produces different features because of the random sampling. A good evaluation would experimentally demonstrate the robustness of the random sampling method for SAT formulas.

We selected the genetic algorithm for finding the random forest because the search space of partial acvation of features can be represented by a boolean expression of size 2^{23}. In Figure 7, $N = 10$ and $K = 3$. The constructed random forest model is evaluated by a function g. Here, g is the squared sum of the differences between VBS and the solved instances in a model selected for Glucose and abcdSAT.

Within the feature space, we searched the optimum random forest model through the above function g with the genetic algorithm (GA). We then tested the random search by

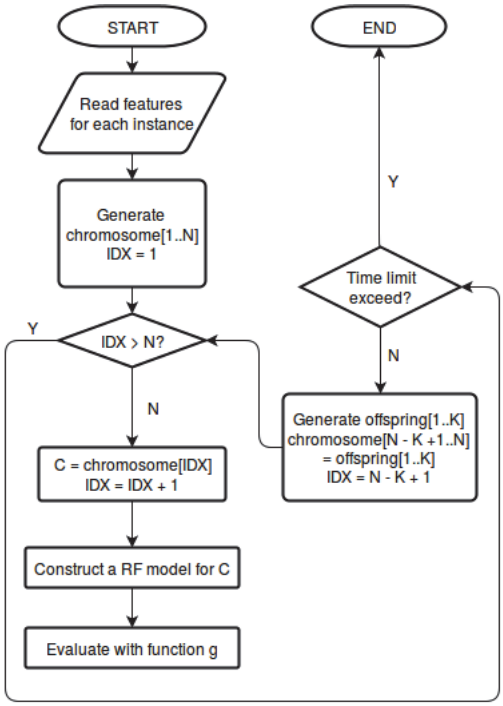


Figure 7: Flow chart for generic algorithm for searching robust random forest model

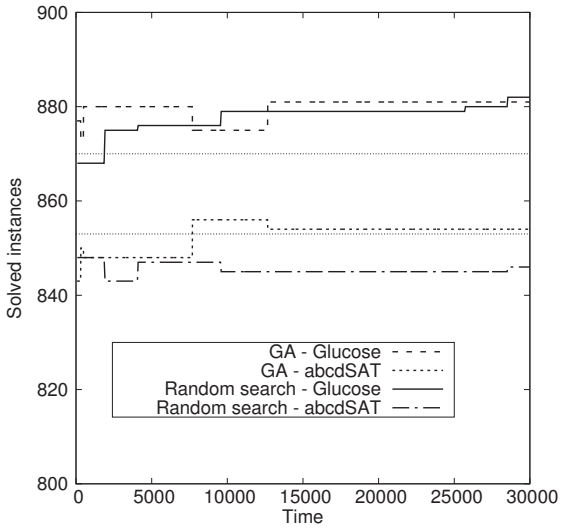


Figure 8: Comparison of genetic algorithm and random selection

constructing a point randomly in the feature space. The results are illustrated in Figure 8. The two lines at $Y = 853$ and $Y = 870$ indicate the performances of f_r in Tables 5 and 6. The GA outperformed the random search and stabilized after finding an optimal model within 8,000 s. These results are superior to those of f_r, confirming that expanding the features and extracting them within a reasonable timeframe improves the random forest model.

Concluding Remarks

First, we proposed a new branching heuristic focused on tie breakage in VSIDS. The proposed TBVSIDS showed better performance than VSIDS.

Second, we proposed a hybrid branching heuristic with higher performance than a single branching heuristic. The preliminary hybrid model used only one feature, but proved its efficiency despite its simplicity. We then improved the preliminary model by implementing random forest on the hybrid branching heuristic with 13 features. We also showed the possibility of applying a random sampling method that reduces the time of feature extraction. Although further studies are required, this approach can assist the extraction of approximate feature values that cannot be extracted from the original formulas.

We then constructed hybrid models from 23 features extracted by the random sampling method. To demonstrate the expandibility of our models, we found a robust model for two solvers through the genetic algorithm. An efficient model was identified within a reasonable timeframe.

Our method considers only branching heuristics; however, a SAT solver includes many different heuristics and their parameters. Optimizing and then combining other small parts, such as the restart policy or the evaluation of learned clauses, might further boost the performance of SAT solvers.

References

Biere, A., and Fröhlich, A. 2015. Evaluating cdcl variable scoring schemes. In *International Conference on Theory and Applications of Satisfiability Testing*, 405–422. Springer.

Chen, J. 2016. Improving abcdsat by at-least-one recently used clause management strategy. *CoRR* abs/1605.01622.

Dershowitz, N.; Hanna, Z.; and Nadel, A. 2005. A clause-based heuristic for sat solvers. In *International Conference on Theory and Applications of Satisfiability Testing*, 46–60. Springer.

Goldberg, E., and Novikov, Y. 2007. Berkmin: A fast and robust sat-solver. *Discrete Applied Mathematics* 155(12):1549–1561.

Liang, J. H.; Ganesh, V.; Poupart, P.; and Czarnecki, K. 2016a. Exponential recency weighted average branching heuristic for sat solvers. In *AAAI*, 3434–3440.

Liang, J. H.; Ganesh, V.; Poupart, P.; and Czarnecki, K. 2016b. Learning rate based branching heuristic for sat solvers. In *International Conference on Theory and Applications of Satisfiability Testing*, 123–140. Springer.

Loreggia, A.; Malitsky, Y.; Samulowitz, H.; and Saraswat, V. A. 2016. Deep learning for algorithm portfolios. In *AAAI*, 1280–1286.

Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th annual Design Automation Conference*, 530–535. ACM.

Oh, C. 2015. Between sat and unsat: the fundamental difference in cdcl sat. In *International Conference on Theory and Applications of Satisfiability Testing*, 307–323. Springer.

Simon, L., and Audemard, G. 2009. Predicting Learnt Clauses Quality in Modern SAT Solver. In *Twenty-first International Joint Conference on Artificial Intelligence (IJCAI'09)*.

Sorensson, N. 2010. Minisat 2.2 and minisat++ 1.1. In *A short description in SAT Race 2010*.

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2007. Satzilla-07: the design and analysis of an algorithm portfolio for sat. In *International Conference on Principles and Practice of Constraint Programming*, 712–727. Springer.