1M2-OS-02b-2

Boost SAT solver with hybrid branching heuristic

Moon Seongsoo   Inaba Mary

Graduate School of Information Science and Technology, The University of Tokyo

Most state-of-the-art satisfiability (SAT) solvers use the variable state independent decaying sum (VSIDS) as their branching heuristic because of its robustness. However, in general, a branching heuristic cannot cover all problems. It is axiomatic that a mixture of branching heuristics with adequate configuration performs better than single branching heuristic. In this paper, we propose a hybrid branching heuristic and show some preliminary experimental results by using benchmarks from SAT Competitions to evaluate its efficiency.

1. Introduction

The satisfiability (SAT) problem is a well-known NP-complete problem. This means that there are no polynomial-time solutions for SAT problems. Although we do not have polynomial-time solutions, there has been substantial progress in SAT algorithms; now, several application problems such as software verification, puzzles, and planning can be solved rapidly by means of state-of-the-art SAT solvers. Several elements have played a significant role in speeding up SAT solvers, and branching heuristic is one of the most influential elements. The *variable state independent decaying sum* (VSIDS) [1] is the most prominent branching heuristic. There have been several attempts [2][3] to beat VSIDS, but VSIDS is still widely used because of its robustness. Recently, several new branching heuristics [4][5] have been proposed, and they might perform better than VSIDS. However, in general, a single branching heuristic cannot handle all problems because SAT solvers are used in a wide and expanding range of practical applications. We are motivated to improve branching heuristic with a hybrid strategy that uses several branching heuristics. In the remainder of this paper, we discuss the related work, explain our proposal for a hybrid strategy, present the experimental results, and conclude the paper.

2. Related work

2.1 Branching heuristics

Most SAT solvers perform a backtracking search to find a solution. A branching heuristic selects an unassigned variable and assigns a value of true or false in a backtracking search. Selection of which variable to branch on next significantly affects the search efficiency. To pick a variable, the branching heuristics have ranking functions that maintain a map of scores corresponding to each variable. The VSIDS branching heuristic updates this map on every conflict. When a conflict occurs, variables related to that conflict obtain a score, incremented by one. Recently, *tie-breaking of VSIDS* (TBVSIDS) [5] was proposed to pick a better variable from ties or reduce the occurrence of ties. CHB [4] uses the concept of reinforcement learning and update scores based on rewards calculated by conflict history.

2.2 Hybrid strategies

It is common to choose several different strategies in a parallel SAT solver to diversify the search. However, sequential solvers normally use a single strategy to intensify the search. There are several attempts to integrate different strategies in a sequential solver. SATzilla [6] contains several different solvers, builds an empirical model using machine learning techniques and chooses an adequate solver for each problem based on its feature values. There is also a attempt to apply a deep learning approach [7] by converting CNF into a grayscale image and building a classifier using a convolutional neural network. There has been an attempt to design a hybrid restart strategy [10] because a slow Luby restart policy is superior to rapid restart policies for SAT problems.

3. Proposal

3.1 Motivation

As aforementioned, a single algorithm cannot cover all SAT problems. To integrate the algorithms and boost the performance of SAT solvers, several algorithm selection studies have been proposed. Let us consider the integration of N different SAT solvers (S_1, S_2, ... , S_N) with different strategies such as restart, learning scheme, learned clause evaluation into a solver I, and optimize I. If we want to improve I with a new policy P, it is necessary to implement P into each S_i and evaluate each of them. In addition, after updating some S_i with P, we must build a model for I again. It seems to take a lot of effort to apply and evaluate a new method. Our final goal is to maximize the performance of I, and to persistently improve I so that it becomes a base solver for other solvers such as MiniSat [8] or glucose [9].

3.2 Approach

Existing algorithm selection studies for the SAT solver might be too high-dimensional. If it is possible to focus on only a small part and improve this part easily, this refinement will lead to better performance and an improved solver can be used as a base solver for other solvers. Branching

Contact: Moon Seongsoo, Creative Informatics (Univ of Tokyo), Bunkyo-ku, Tokyo Yayoi 1-1-1 University of Tokyo Graduate School of Information Science and Technology, Graduate School of Creative Information Department of I-REF Building 4F, Tel: 03-3812-2111, E-mail address: logic85@hotmail.com

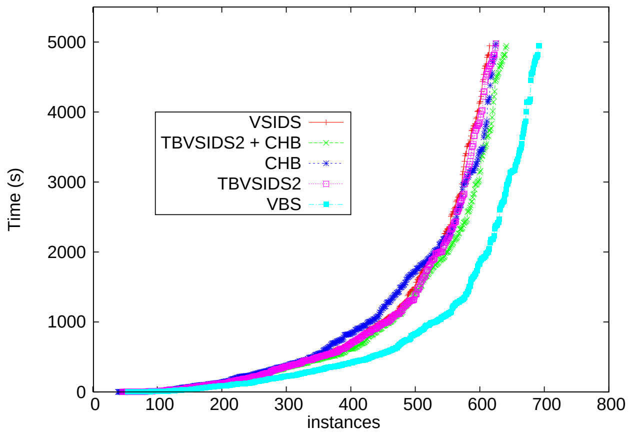

Fig. 1: Cactus plot of the 900 instances from SAT competitions

heuristics can be an appropriate candidate to obtain this. The recently proposed CHB and TBVSIDS can be implemented easily by using the data structure of VSIDS. This implies that these branching heuristics can be integrated easily into a solver. We constructed a random forest model to allocate an appropriate branching heuristic by training several features in an original formula.

4. Experimental results

4.1 Static approach

In our preliminary analysis, we noticed that CHB works well when the input has a small number of variables. As a first step towards the development of a hybrid strategy, we selected two branching heuristics, i.e., TBVSIDS and CHB. As preprocessing, we simply count the number of variables, and select CHB when the number of variables is under 9,000; otherwise, TBVSIDS is selected as the branching heuristic.

Figure 1 compares the performance of different branching heuristics. We find that our extremely simple hybrid strategy highly outperforms single branching heuristics. If we succeed in learning a more detailed policy, we could reduce the gap between our method and VBS.

4.2 Random forest

We trained our model using the benchmarks of SAT Competitions from 2014 to 2016 in both the crafted and application tracks. We used 13 features: the number of variables and clauses in the original formula and variable-clause graph features (mean, variation coefficient, min, max, and entropy for both variable node degree and clause node degree). These features can be extracted within a short time and can be obtained without a specific algorithm. This is important because when we solve a formula using an SAT solver with our model, we must extract features as preprocessing, and time-consuming feature extraction is not desirable. We used branching heuristics as classes. VSIDS, CHB, and TBVSIDS have tow versions, and we set 2 different parameters for the second version. Moreover we further implement *Tie-breaking of CHB* (TBCHB) to obtain a total of eight different branching heuristics. Our goal is to maximize the performance of the SAT solver and minimize the overfitting problem. Using more features does not ensure better results. We show partial results in Tables 1 and 2. The results in Table 1 are obtained using all 13 features mentioned above. In Table 2, we only used one feature. When we evaluated performance of these classifiers using k-fold cross validation (Training data: $\hat{A}X \mid \hat{C}X$, Test data $AX \mid CX$, where $X = 14 \mid 15 \mid 16$), Table 2 solved 25 problems more than Table 1. Therefore, a model in Table 2 might be trained better than a model in Table 1.

Table 1: Test results with several training data sets using all 13 features. Columns: Training data. Rows: Test data. C: Crafted Track and A: Application Track. $\hat{A}X$ = all - AX. $\hat{C}X$ = all - CX, where $X = 14 \mid 15 \mid 16$.

	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
$\hat{C}$14 (714)	**167**	217	249	61	143	**837**
$\hat{A}$14 (697)	198	**211**	247	62	143	**861**
$\hat{A}$15 (667)	201	213	**245**	63	145	**867**
$\hat{C}$16 (863)	204	214	246	**38**	142	**844**
$\hat{A}$16 (771)	201	213	249	63	**139**	**865**
all (928)	202	213	247	63	143	868

Table 2: Test results with several training data sets 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
$\hat{C}$14	**180**	218	247	56	142	**843**
$\hat{A}$14	196	**211**	245	56	139	**847**
$\hat{A}$15	196	215	**246**	56	142	**855**
$\hat{C}$16	202	216	247	**51**	142	**858**
$\hat{A}$16	193	215	248	56	**137**	**849**
all	196	214	244	56	139	849

5. Concluding Remarks

We proposed a hybrid branching heuristic using an algorithm selection technique. Currently, our model extracts only several features per formula, and we want to expand it to other features. By considering other features, we might reduce our overfitting problem; however, some features require too much time for calculation. To reduce the time for feature extraction, we will consider formula sampling. Our method only concentrates on branching heuristics; however, an SAT solver includes many different heuristics and their parameters. If we attempt to optimize other small parts such as a restart policy or an evaluation of learned clauses, then the combination of optimized small parts might boost the performance of SAT solvers.

References

[1] Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232). (2001).

[2] Goldberg, E., Novikov, Y.: BerkMin: A Fast and Robust Sat-Solver. Design, Automation, and Test in Europe. 465478 (2008).

[3] Dershowitz, N., Hanna, Z., Nadel, A.: A Clause-Based Heuristic for SAT Solvers. Theory and Applications of Satisfiability Testing Lecture Notes in Computer Science. 4660 (2005).

[4] Hui Liang, J., Ganesh, V., Poupart, P., Czarnecki, K. : Exponential Recency Weighted Average Branching Heuristic for SAT Solvers. Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. (2016).

[5] https://www.dropbox.com/sh/kz9hwvhtidyo4rh/AAAlNhnA8w8cnU4TeFNMMRoba/5_Doctoral_Program?dl=0&preview=Moon+(1).pdf

[6] Xu, L., Hutter, F., Hoos, H. H., Leyton-Brown, K. (n.d.). SATzilla-07: The Design and Analysis of an Algorithm Portfolio for SAT. Principles and Practice of Constraint Programming CP 2007 Lecture Notes in Computer Science, 712-727. (2007)

[7] Loreggia, A., Malitsky, Y., Samulowitz, H., Saraswat, V.: Deep Learning for Algorithm Portfolios. Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. 1280-1286 (2016).

[8] http://minisat.se/

[9] http://www.labri.fr/perso/lsimon/glucose/

[10] Oh, C.: Between SAT and UNSAT: The Fundamental Difference in CDCL SAT. Lecture Notes in Computer Science Theory and Applications of Satisfiability Testing – SAT 2015. 307323 (2015).