Exponential Recency Weighted Average Branching Heuristic for SAT Solvers

Jia Hui Liang and **Vijay Ganesh** and **Pascal Poupart** and **Krzysztof Czarnecki**
jliang@gsd.uwaterloo.ca, vijay.ganesh@uwaterloo.ca
ppoupart@uwaterloo.ca, kczarnec@gsd.uwaterloo.ca
University of Waterloo, Canada

Abstract

Modern conflict-driven clause-learning SAT solvers routinely solve large real-world instances with millions of clauses and variables in them. Their success crucially depends on effective branching heuristics. In this paper, we propose a new branching heuristic inspired by the exponential recency weighted average algorithm used to solve the bandit problem. The branching heuristic, we call CHB, learns online which variables to branch on by leveraging the feedback received from conflict analysis. We evaluated CHB on 1200 instances from the SAT Competition 2013 and 2014 instances, and showed that CHB solves significantly more instances than VSIDS, currently the most effective branching heuristic in widespread use. More precisely, we implemented CHB as part of the MiniSat and Glucose solvers, and performed an apple-to-apple comparison with their VSIDS-based variants. CHB-based MiniSat (resp. CHB-based Glucose) solved approximately 16.1% (resp. 5.6%) more instances than their VSIDS-based variants. Additionally, CHB-based solvers are much more efficient at constructing first preimage attacks on step-reduced SHA-1 and MD5 cryptographic hash functions, than their VSIDS-based counterparts. To the best of our knowledge, CHB is the first branching heuristic to solve significantly more instances than VSIDS on a large, diverse benchmark of real-world instances.

Introduction

Over the past two decades, conflict-driven clause-learning (CDCL) SAT solvers (Marques-Silva and Sakallah 1999; Moskewicz et al. 2001; Audemard and Simon 2009a; Sorensson and Een 2005; Biere 2010), designed to solve the Boolean satisfiability problem, have played a crucial role in the development of many innovative techniques in AI, software engineering, and security. Examples include solver-based automated testing with symbolic execution (Cadar et al. 2008), bounded model checking (Biere et al. 2003) for software and hardware verification, and planning in AI (Rintanen 2009). These solvers are surprisingly efficient in solving large classes of real-world instances which may contain tens of millions of variables and clauses in them, even though the Boolean satisfiability problem is known to be NP-complete and believed to be intractable in general. A key element in the success of CDCL SAT solvers is the branching heuristic, such as Variable State Independent Decaying Sum (VSIDS) (Moskewicz et al. 2001), that dynamically select variables and assign them truth values (the process also known as *branching*) as the solver searches for a solution to the input Boolean formula.

While many branching heuristics have been invented, two significant issues have hampered their study in the past. First, despite the considerable effort expended in designing branching heuristics such as DLIS (Marques-Silva 1999), BerkMin (Goldberg and Novikov 2007), and Jeroslav-Wang (Jeroslow and Wang 1990), the VSIDS branching heuristic and its variants remain as the most effective ones in widespread use today. The success of VSIDS has dramatically raised the bar that any new heuristic has to overcome. Second, until very recently, little was understood as to why the VSIDS branching heuristic and its variants are so effective. Some recent papers provide insights into the inner working of VSIDS (Liang et al. 2015; Ansótegui, Giráldez-Cru, and Levy 2012; Biere and Fröhlich 2015), and some of those insights (Liang et al. 2015) are the basis and inspiration for the work presented in this paper.

In this paper, we present a new branching heuristic, we call *conflict history-based branching heuristic* (CHB), based on the exponential recency weighted average (ERWA) algorithm used in nonstationary multi-armed bandit problems (i.e., single state reinforcement learning problems) to estimate the average reward of different actions (Sutton and Barto 1998). Inspired by the bandit framework and reinforcement learning, we learn to choose good variables to branch based on past experience. Our goal is to leverage the theory and practice of a rich sub-field of reinforcement learning to explain and design an effective branching heuristic for solving real-world problems.

The branching heuristic proposed in this paper, CHB, is completely online and learns which variable to branch on dynamically as the input instance is being solved. By online we mean that the heuristic learns only during the solving process, and there is no offline learning.

We evaluated the efficacy of CHB on the SAT Competition 2013 (Balint et al. 2013) and 2014 (Belov et al. 2014) benchmarks from the application and hand-crafted categories, and show that CHB solves significantly more instances than VSIDS. The SAT Competition is a fiercely competitive annual international competition, where dozens of the best SAT solvers in the world compete with each other on a set of benchmarks. These benchmarks, the gold standard of SAT solver research, are perhaps the most comprehensive, diverse, large, difficult-to-solve, and well-curated set of instances obtained from industrial applications such as software and hardware verification, AI, security, program analysis, and cryptography.

We implemented CHB as part of the MiniSat and Glucose solvers, and performed an apple-to-apple comparison with their VSIDS-based variants. CHB-based MiniSat (resp. CHB-based Glucose) solved approximately 16.1% (resp. 5.6%) more instances than the stock VSIDS-based solver. Additionally, CHB-based solvers are much more efficient than VSIDS-based ones in constructing first preimage attacks on step-reduced versions of SHA-1 and MD5 cryptographic hash functions. Both Glucose and MiniSat are among the best solvers in widespread use today. The Glucose solver in particular has won several recent SAT competitions in multiple categories (Audemard and Simon 2009a).

To better appreciate the advance brought about by the CHB branching heuristic, we quote two of the leading SAT solver developers Professors Audemard and Simon (Audemard and Simon 2012):

> “We must also say, as a preliminary, that improving SAT solvers is often a cruel world. To give an idea, improving a solver by solving at least ten more instances (on a fixed set of benchmarks of a competition) is generally showing a critical new feature. In general, the winner of a competition is decided based on a couple of additional solved benchmarks.”

Contributions

In this paper, we make the following contributions to the efficacy and understanding of branching heuristics in CDCL SAT solving:

- We contribute a new branching heuristic called CHB, inspired by the bandit framework in reinforcement learning, that when combined appropriately with conflict analysis solves significantly more instances than VSIDS-based solvers. VSIDS (Moskewicz et al. 2001) has been the state-of-the-art branching heuristic for the last 15 years. As far as we know, this is the first branching heuristic capable of solving more instances than VSIDS on a large and diverse real-world benchmark.
- We show that CHB-based CDCL solvers construct first preimage attacks on step-reduced versions of SHA-1 and MD5 cryptographic hash functions more efficiently than the VSIDS-based solvers (Mironov and Zhang 2006).

Background

In this section, we describe CDCL SAT solvers, the state-of-the-art branching heuristic called VSIDS (Moskewicz et al. 2001), and an algorithm for solving the nonstationary multi-armed bandit problem called ERWA (Sutton and Barto 1998) which is the basis of the branching algorithm CHB proposed in this paper.

Conflict-driven Clause-learning SAT Solvers

Conflict-driven clause-learning (CDCL) SAT solvers are the dominant solvers in practice today. They take as input formulas in Boolean logic, and decide whether they are satisfiable. The input formulas are specified in conjunctive normal form (CNF). CDCL SAT solvers perform backtracking search, where at each step, the branching heuristic picks an unassigned variable and assigns it a value of true or false (Moskewicz et al. 2001), a process called branching. The assignment given to a variable during branching is propagated via a method called Boolean constraint propagation (Moskewicz et al. 2001). Propagation is the process by which the solver simplifies the input formula, leveraging the assignment given to the *branched variable* and its logical consequences. If propagation leads to a falsified clause, the *current assignment* is not a satisfying assignment for the input formula. This state of the solver is referred to as being in *conflict*. The solver recovers from a conflict by backtracking, undoing some of the offending decisions, and trying some other assignments.

Crucial to the success of CDCL solvers is the process of clause learning (Marques-Silva and Sakallah 1999), that is triggered when the solver enters a conflict state. At a high-level, the solver computes a root cause of why the conflict occurred, i.e., a subset of currently assigned variables, such that any extensions of which is always an unsatisfying assignment to the input formula. Once this root cause has been identified, the solver remembers it in the form of a learnt clause in order to avoid the mistakes (and exponentially many similar mistakes) that led to a conflict.

The algorithm responsible for picking which variable to branch on is called the *branching heuristic*. Typically, a separate heuristic is used to pick which value, true or false, to assign to the variable. The order in which the branching heuristic picks variables has an enormous impact on the solving time. Besides clause learning, the branching heuristic is one of the most important features in modern CDCL SAT solvers (Katebi, Sakallah, and Marques-Silva 2011). The current state-of-the-art branching heuristic is called Variable State Independent Decaying Sum (VSIDS), proposed in 2001 by the authors of the Chaff solver (Moskewicz et al. 2001). A decade and a half after it was initially proposed, VSIDS and its variants continue to be the dominant branching heuristics among competitive SAT solvers such as Glucose (Audemard and Simon 2009a), Lingeling (Biere 2010), and CryptoMiniSat (Soos 2010).

The VSIDS Branching Heuristic

All branching heuristics can be characterized as ranking functions that maintain a map from variables in an input formula to floating point numbers. Abstractly speaking, the branching heuristic maintains this map in decreasing order of values assigned to the variables. The VSIDS branching heuristic maintains a floating point number, often called activity, for each Boolean variable. Whenever a learnt clause is added to the clause database by the CDCL SAT solver, the activities of all the variables present in the learnt clause are dynamically incremented by 1, also called the bump.

The activities of all variables are periodically multiplied by a floating point constant between 0 and 1, also called the decay. Modern variations of VSIDS typically bump all variables present in the clauses used by conflict analysis, not just the learnt clause variables, and decay after every conflict (Sorensson and Een 2005). VSIDS picks the unassigned variable with the highest activity to branch on.

Exponential Recency Weighted Average (ERWA)

Exponential recency weighted average (ERWA) is a simple technique to estimate a moving average incrementally by giving more weight to the more recent outcomes. Consider a stream of outcomes $x_1, x_2, x_3, ..., x_n$. We can compute an exponentially decaying weighted average of the outcomes with Equation 1.

$$\bar{x}_n = \sum_{i=1}^{n} w_i x_i \text{ where } w_i = \alpha(1-\alpha)^{n-i} \quad (1)$$

Here $\alpha \in [0, 1]$ is a factor that determines the rate at which the weights decay. To reduce the computational overhead, the moving average can be computed incrementally by updating with Equation 2 after each outcome.

$$\bar{x}_{n+1} = (1-\alpha)\bar{x}_n + \alpha x_{n+1} \quad (2)$$

ERWA has been used in the context of bandit problems (i.e., single state reinforcement learning problems) to estimate the expected reward of different actions in non-stationary environments (Sutton and Barto 1998). In bandit problems, there is a set of arms (or actions) and the agent must select which arm to play at each time step in order to maximize its long term expected reward. Since it does not know the distribution of rewards for each arm, it faces an important exploration/exploitation tradeoff. It needs to explore by trying each arm in order to estimate the expected reward of each arm and it needs to exploit by selecting arms with high expected reward. ERWA is a simple technique to estimate the empirical average of each arm. In the context of the branching heuristic we propose in this paper, we use ERWA to estimate the moving average of the "score" of each variable in the input formulas, in an online and dynamic fashion, during the entire run of the solver. Inspired by the bandit framework, we treat each variable as an arm and estimate a score for each variable that reflects the frequency and persistence of the variable in generating conflicts in the past.

The CHB Branching Heuristic

In this section, we describe our branching heuristic CHB in the context of a CDCL solver. Algorithm 1 shows the pseudocode of a simple CDCL solver with CHB as the branching heuristic, and we will refer to the line numbers of this algorithm where relevant. CHB maintains a floating point number for each Boolean variable called the Q score, initialized to 0 at the start of the search. Whenever a variable v is branched on, propagated, or asserted, the Q score is updated using Equation 3 (line 18) where α is the step-size and r_v is the reward value.

$$Q[v] = (1-\alpha)Q[v] + \alpha r_v \quad (3)$$

As is typical with exponential recency weighted average, the step-size decreases over time (Sutton and Barto 1998). The step-size is initialized to $\alpha = 0.4$ at the start of the search and decreases by 10^{-6} every conflict to a minimum of 0.06, and stays there for the run of the solver (line 26). Note that the Glucose solver implements a similar idea, where the decay factor used by VSIDS decreases over time (Audemard and Simon 2013).

r_v is the reward value, just as in the bandit problem. A low (resp. high) reward value decreases (resp. increases) the likelihood of picking v to branch on. The reward value is based on how recently variable v appeared in conflict analysis. Let $numConflicts$ be an integer variable that keeps track of how many conflicts have occured so far (line 21) and $lastConflict$ be a mapping from each variable to an integer. Initally, $lastConflict[v] = 0$ for each variable v. Whenever a variable x is present in the clauses used by conflict analysis, $lastConflict$ is updated by $lastConflict[x] = numConflicts$ (line 34). The reward value used by CHB is defined in Equation 4 (line 17).

$$r_v = \frac{multiplier}{numConflicts - lastConflict[v] + 1} \quad (4)$$

Here, $multiplier$ is either 1.0 or 0.9. If branching, propagating, or asserting the variable that triggered the update of Q encounters a conflict after propagation, then $multiplier = 1.0$ (line 12). Otherwise, $multiplier = 0.9$ (line 14).

The intuition of the reward value is similar to the intuition of VSIDS, that is to favor variables that appear recently in conflict analysis (Audemard and Simon 2009b). Additionally, the multiplier gives extra reward for producing a conflict. Based on our experience, this reward function gives good performance for CHB in practice. Empirically we have some unpublished evidence that, all else being equal, branching heuristics that have higher rate of conflict clause generation per unit time are more effective than ones that have lower rate. Leveraging this observation, we designed the reward function to maximize the rate of learnt clause generation per unit time.

During branching, CHB selects the greediest play possible by branching on the unassigned variable v with the highest Q score (line 42). The algorithm always exploits and this does not appear to be an issue in practice since the problem itself forces exploration in two ways. First, if the algorithm greedily branches on variable v, then it cannot branch on v again until the solver undoes v through backtracking/restarting since v is now assigned and the algorithm is only allowed to branch on unassigned variables. Hence the algorithm is forced to branch on other variables. Second, the propagated variables also have their Q scores updated. Hence variables with low Q scores that will not be picked for branching can still have their Q scores updated.

Differences Between CHB and VSIDS

There are several major differences between the CHB and VSIDS branching heuristics. First, VSIDS only updates the activities on each conflict, whereas CHB updates the Q

Algorithm 1 a simple CDCL solver with CHB as the branching heuristic.

```
$\alpha \leftarrow 0.4$
$numConflicts \leftarrow 0$
$plays \leftarrow \emptyset$
for $v \in Vars$ do
    $lastConflict[v] \leftarrow 0$
    $Q[v] \leftarrow 0$
end for
loop
    Boolean constraint propagation
    $plays \leftarrow plays \cup \{$variables propagated just now$\}$
    if a clause is in conflict then
        $multiplier \leftarrow 1.0$
    else
        $multiplier \leftarrow 0.9$
    end if
    for $v \in plays$ do
        $reward \leftarrow \frac{multiplier}{numConflicts - lastConflict[v] + 1}$
        $Q[v] \leftarrow (1 - \alpha) \times Q[v] + \alpha \times reward$
    end for
    if a clause is in conflict then
        $numConflicts \leftarrow numConflicts + 1$
        if $decisionLevel == 0$ then
            return UNSAT
        end if
        if $\alpha > 0.06$ then
            $\alpha \leftarrow \alpha - 10^{-6}$
        end if
        conflict analysis and learn a new clause
        $c \leftarrow \{$variables in conflict analysis$\}$
        $u \leftarrow$ the first UIP of the learnt clause
        non-chron. backtrack based on conflict analysis
        assert variable $u$ based on new learnt clause
        for $v \in c$ do
            $lastConflict[v] \leftarrow numConflicts$
        end for
        $plays \leftarrow \{u\}$
    else
        if no more unassigned variables then
            return SAT
        end if
        $unassigned \leftarrow \{$unassigned variables$\}$
        $v^* \leftarrow argmax_{v \in unassigned} Q[v]$
        assign $v^*$ to true or false based on polarity
    heuristic such as phase saving
        $plays \leftarrow \{v^*\}$
    end if
end loop
```

scores whenever a variable is branched on, propagated, or asserted. Additionally, VSIDS decays the activities of all variables whereas CHB again only decays the Q scores of branched, propagated, and asserted variables. The reward values (or bump values) in CHB are variable, whereas these values are constant in VSIDS. Also, the reward value in CHB is based on the conflict history whereas the bump value in VSIDS is determined by just the current conflict. Lastly, CHB is based on a known algorithm from reinforcement learning, thus giving us a basis for modeling and understanding branching heuristics.

Evaluation

This section describes our experimental evaluation of the practical performance of CHB versus VSIDS.

Choice of SAT Solvers

CHB was evaluated on 2 notable CDCL SAT solvers: MiniSat version 2.2.0 (Sorensson and Een 2005) and Glucose version 4.0 (Audemard and Simon 2009a). MiniSat is a popular CDCL SAT solver, which many other competitive solvers use as their basis. Additionally, it contains very few features, thus isolating the effects of the change in branching heuristics in our experiments. We used the core version of MiniSat which does not perform simplifications other than removing satisfied clauses. Glucose is a state-of-the-art CDCL SAT solver, and a winner of numerous SAT competitions. The two solvers in our evaluation gave us a comprehensive perspective of the effects of the CHB branching heuristic: on the one hand, we evaluated the effects of CHB on a very simple CDCL solver to isolate its effects, and on the other hand, a modern competitive solver to understand how CHB competes with state-of-the-art. Note that MiniSat and Glucose implement variations of VSIDS as their branching heuristics.

Methodology

For each solver-instance pair, we ran the solver on the SAT instance twice. The first run was with the default unmodified solver (which uses VSIDS as its branching heuristic) and the second run was with a modified version of the solver using CHB instead of VSIDS. The evaluation compares which solver, unmodified with VSIDS or modified with CHB, solved more instances. Each run was executed on StarExec (Stump, Sutcliffe, and Tinelli 2014), a platform designed for evaluating logic solvers. The StarExec platform uses the Intel Xeon CPU E5-2609 at 2.40GHz with 10240 KB cache and 24 GB of main memory, running on Red Hat Enterprise Linux Workstation release 6.3, and Linux kernel 2.6.32-431.1.2.el6.x86_64.

The modified solvers changed only the branching heuristic by replacing VSIDS with CHB. We left everything else in the solvers untouched, and the solvers' parameters are left to their defaults. We made no attempt to tune the solvers' parameters to work well with CHB. Additionally, the modified solvers reused the existing data structures such as the existing priority queue data structures for finding the variable ranked highest by VSIDS/CHB.

		Unmodified MiniSat	MiniSat with CHB
2013App	SAT	115	131 (**+13.9%**)
	UNSAT	59	59 (0.0%)
	BOTH	174	190 (**+9.2%**)
2013Crafted	SAT	106	106 (0.0%)
	UNSAT	62	92 (**+48.4%**)
	BOTH	168	198 (**+17.9%**)
2014App	SAT	89	113 (**+27.0%**)
	UNSAT	56	49 (-12.5%)
	BOTH	145	162 (**+11.7%**)
2014Crafted	SAT	73	84 (**+15.1%**)
	UNSAT	44	67 (**+52.3%**)
	BOTH	117	151 (**+29.1%**)
TOTAL	**SAT**	**383**	**434** (**+13.3%**)
	UNSAT	**221**	**267** (**+20.8%**)
	BOTH	**604**	**701** (**+16.1%**)

Table 1: The table presents the number of solved instances by unmodified MiniSat (with VSIDS) versus MiniSat with CHB on SAT 2013 and 2014 handcrafted and application benchmarks. Additionally, the difference between the two solvers is given as a percentage. For example, MiniSat with CHB solves 16.1% more instances than unmodified MiniSat over the entire benchmark.

		Unmodified Glucose	Glucose with CHB
2013App	SAT	103	115 (**+11.7%**)
	UNSAT	103	115 (**+11.7%**)
	BOTH	206	230 (**+11.7%**)
2013Crafted	SAT	113	119 (**+5.3%**)
	UNSAT	104	106 (**+1.9%**)
	BOTH	217	225 (**+3.7%**)
2014App	SAT	99	103 (**+4.0%**)
	UNSAT	116	106 (-8.6%)
	BOTH	215	209 (-2.8%)
2014Crafted	SAT	79	86 (**+8.9%**)
	UNSAT	89	101 (**+13.5%**)
	BOTH	168	187 (**+11.3%**)
TOTAL	**SAT**	**394**	**423** (**+7.4%**)
	UNSAT	**412**	**428** (**+3.9%**)
	BOTH	**806**	**851** (**+5.6%**)

Table 2: The table presents the number of solved instances by unmodified Glucose (with VSIDS) versus Glucose with CHB on SAT 2013 and 2014 handcrafted and application benchmarks. Additionally, the difference between the two solvers is given as a percentage. For example, Glucose with CHB solves 5.6% more instances than unmodified Glucose over the entire benchmark.

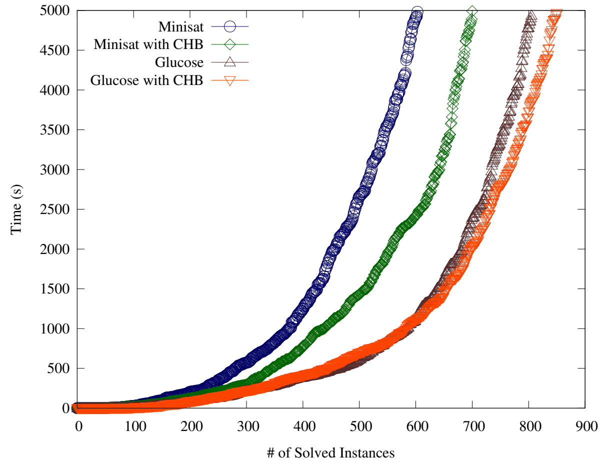

Figure 1: Cactus plot of the running times on the SAT Competition 2013 and 2014 benchmarks. A point (x, y) means that there are x instances where each one can be solved within y seconds with the given solver. A line further to the right means the solver solved more instances. A line further down means it solved instances faster.

Results: SAT Competition 2013 and 2014

First, CHB was evaluated on all 1200 instances from the application and hand-crafted categories of the SAT Competition 2013 and 2014 benchmarks. The instances from the hand-crafted category are designed to be challenging for SAT solvers. Each run of a solver was given 5000 seconds and 7.5 GB of memory to solve an instance, as per the rules of SAT Competition 2013. Tables 1 and 2 show how CHB solved more instances on this large benchmark than VSIDS.

The results show a big increase in the number of solved satisfiable instances in MiniSat and Glucose. MiniSat is a very popular solver, however it has not won any recent SAT competitions unlike Glucose. Yet MiniSat with the one addition of CHB solved 436 satisfiable instances over the entire benchmark, considerably more than the 394 solved satisfiable instances by unmodified Glucose (with VSIDS). CHB solves more satisfiable instances than VSIDS for both categories and both years. CHB also dramatically increases the number of solved unsatisfiable instances by 29 for MiniSat. Glucose with CHB solved 16 more unsatisfiable instances.

Figure 1 shows the cactus plot of the running times, a standard diagram in SAT literature for comparing the performance of solvers. Lines further down and to the right on the plot are better. The plot shows a significant improvement of CHB over VSIDS on the SAT Competition instances.

To put this into context, the solver Minsat_blbd (Chen 2014) won first place in the SAT Competition 2014 by solving the most satisfiable instances in the application track and Lingeling (Biere 2010) won that track for 2013. On the StarExec platform, Minsat_blbd solved 106 satisfiable instances in the 2014 application track, whereas MiniSat with

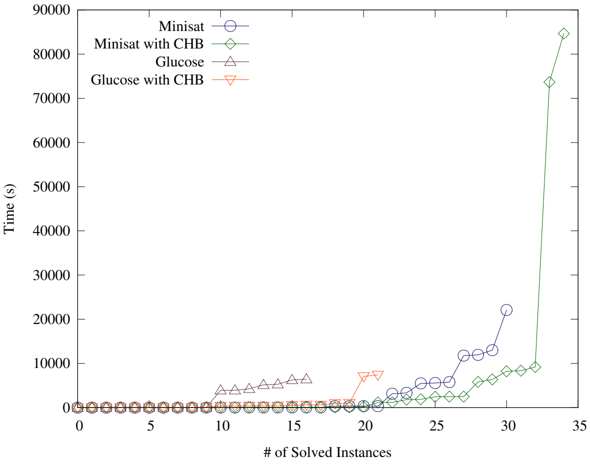

Figure 2: Cactus plot of the running times on the step-reduced SHA-1 and MD5 first preimage attacks. Refer to Figure 1 for an explanation on how to interpret cactus plots.

CHB solved 113 instances as noted in Table 1. Likewise for the 2013 application track, Lingeling solved 122 satisfiable instances, whereas MiniSat with CHB solved 131 instances.

Results: SAT-based Cryptanalysis

We also evaluated the CHB branching heuristic over 10 instances each of 21/22/23-step-reduced (resp. 27/28/29-step-reduced) encodings of a first preimage attack on SHA-1 (resp. MD5) for a total of 60 instances. SHA-1 and MD5 are cryptographic hash functions and are considered preimage resistant. That is, given an output of the hash function, it is believed to be computationally infeasible to compute an input that produces that output. In this benchmark, only a subset of the hash function steps are encoded, as the first preimage attack on the whole function is still intractable. Each instance fixed the Boolean variables corresponding to the hash function's output to a random value. The satisfying assignment of the Boolean variables, corresponding to the hash function's input, is a message that hashes to the specified output. Hence, the satisfiying assignment produced by a SAT solver is a successful first preimage attack on the step-reduced hash function.

Each run of a solver was given 24 hours and 10 GB of memory to solve each instance. Figure 2 shows the running times of CHB versus VSIDS in solving the step-reduced SHA-1 and MD5 cryptographic functions. MiniSat with CHB solved 3 more instances of 23-step-reduced SHA-1, and 1 more instance of 28-step-reduced MD5 than unmodified MiniSat (with VSIDS). Glucose with CHB solved 3 more instances of 21-step-reduced SHA-1 and 2 more instances of 22-step-reduced SHA-1 than unmodified Glucose (with VSIDS).

In conclusion, CHB-enhanced CDCL solvers significantly outperformed VSIDS-based ones on a large, real-world benchmark.

Related Work

Marques-Silva and Sakallah invented the CDCL technique (Marques-Silva and Sakallah 1999), the dominant approach for solving practical SAT problems efficiently. The VSIDS branching heuristic, the dominant branching heuristic employed in modern CDCL SAT solvers, was originally proposed by the authors of the Chaff solver (Moskewicz et al. 2001). Lagoudakis and Littman took 7 well-known SAT branching heuristics (MAXO, MOMS, MAMS, Jeroslaw-Wang, UP, GUP, SUP) and used reinforcement learning to switch between the heuristics dynamically during the run of the solver (Lagoudakis and Littman 2001). Their technique requires offline training on a class of similar instances. The algorithm proposed in this paper differs in that it learns to select good variables rather than learning to select a good branching heuristic from a fixed-set. Additionally, CHB requires no offline training.

Future Work

The connection between branching heuristics and reinforcement learning opens many new opportunities for future improvements to branching heuristics and SAT solving in general. We detail some of our future work below that builds on the foundation laid by this paper. Modern CDCL SAT solvers maintain lots of state features such as the partial assignment, trail, learnt clause database, saved phases, etc. The technique proposed in this paper is based on the multi-armed bandit setting, and it can be extended to a full Markov decision process by conditioning the choice of variables on some of the solvers' state features. More research is needed to find a stateful model that works well in practice by balancing the trade-off between the gain in information due to states and the cost of increased model complexity.

We modeled our branching heuristic on ERWA, a technique used to solve the bandit problem. However, perhaps a more powerful model will capture both the branching heuristic and clause learning. It is evident that the branching heuristic and clause learning drive each other, so a model capturing both aspects can lead to algorithms that not only choose better branching variables, but also learn higher quality clauses. It is clear that such a model is outside the bandit framework, due to the additional feedback from clause learning to the branching heuristic in the form of learnt clauses. More work is needed to construct new models that include more aspects of CDCL such as clause learning.

Conclusion

We introduced a new branching heuristic CHB inspired by the multi-armed bandit problem, a special case of reinforcement learning. We evaluated the heuristic on two popular SAT solvers, MiniSat and Glucose, and demonstrated that CHB solves more instances than VSIDS on a large and diverse benchmark containing 1200 instances from recent SAT competitions and 60 instances of first preimage attacks on cryptographic hash functions. The results show that CHB is more effective in CDCL SAT solvers at solving instances than the longstanding state-of-the-art VSIDS branching heuristic.

References

Ansótegui, C.; Giráldez-Cru, J.; and Levy, J. 2012. The community structure of SAT formulas. In *Theory and Applications of Satisfiability Testing–SAT 2012*. Springer. 410–423.

Audemard, G., and Simon, L. 2009a. Glucose: a solver that predicts learnt clauses quality. *SAT Competition* 7–8.

Audemard, G., and Simon, L. 2009b. Predicting learnt clauses quality in modern SAT solvers. In *IJCAI*, volume 9, 399–404.

Audemard, G., and Simon, L. 2012. Refining restarts strategies for SAT and UNSAT. In *Principles and Practice of Constraint Programming*, 118–126. Springer.

Audemard, G., and Simon, L. 2013. Glucose 2.3 in the sat 2013 competition. In *Proceedings of SAT Competition 2013*, 42–43.

Balint, A.; Belov, A.; Heule, M. J. H.; and Järvisalo, M. 2013. Solver and benchmark descriptions. In *Proceedings of SAT Competition 2013*, volume B-2013-1. University of Helsinki.

Belov, A.; Diepold, D.; Heule, M. J. H.; and Järvisalo, M. 2014. Solver and benchmark descriptions. In *Proceedings of SAT Competition 2014*, volume B-2014-2. University of Helsinki.

Biere, A., and Fröhlich, A. 2015. Evaluating CDCL variable scoring schemes. In *Theory and Applications of Satisfiability Testing–SAT 2015*. Springer. 405–422.

Biere, A.; Cimatti, A.; Clarke, E. M.; Strichman, O.; and Zhu, Y. 2003. Bounded model checking. *Advances in computers* 58:117–148.

Biere, A. 2010. Lingeling, plingeling, picosat and precosat at SAT race 2010. *FMV Report Series Technical Report* 10(1).

Cadar, C.; Ganesh, V.; Pawlowski, P. M.; Dill, D. L.; and Engler, D. R. 2008. EXE: automatically generating inputs of death. *ACM Transactions on Information and System Security (TISSEC)* 12(2):10.

Chen, J. 2014. Minisat_blbd. In *Proceedings of SAT Competition 2014*, volume B-2014-2, 45. University of Helsinki.

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

Jeroslow, R. G., and Wang, J. 1990. Solving propositional satisfiability problems. *Annals of mathematics and Artificial Intelligence* 1(1-4):167–187.

Katebi, H.; Sakallah, K. A.; and Marques-Silva, J. P. 2011. Empirical study of the anatomy of modern SAT solvers. In *Theory and Applications of Satisfiability Testing-SAT 2011*. Springer. 343–356.

Lagoudakis, M. G., and Littman, M. L. 2001. Learning to select branching rules in the DPLL procedure for satisfiability. *Electronic Notes in Discrete Mathematics* 9:344–359.

Liang, J. H.; Ganesh, V.; Zulkoski, E.; Zaman, A.; and Czarnecki, K. 2015. Understanding VSIDS branching heuristics in conflict-driven clause-learning SAT solvers. In *Hardware and Software: Verification and Testing*. Springer. 225–241.

Marques-Silva, J. P., and Sakallah, K. 1999. GRASP: A search algorithm for propositional satisfiability. *Computers, IEEE Transactions on* 48(5):506–521.

Marques-Silva, J. P. 1999. The impact of branching heuristics in propositional satisfiability algorithms. In *Progress in Artificial Intelligence*. Springer. 62–74.

Mironov, I., and Zhang, L. 2006. Applications of SAT solvers to cryptanalysis of hash functions. In *Theory and Applications of Satisfiability Testing-SAT 2006*. Springer. 102–115.

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.

Rintanen, J. 2009. Planning and SAT. *Handbook of Satisfiability* 185:483–504.

Soos, M. 2010. Cryptominisat 2.5.0. *SAT Race*.

Sorensson, N., and Een, N. 2005. MiniSat v1.13-a SAT solver with conflict-clause minimization. *SAT* 2005:53.

Stump, A.; Sutcliffe, G.; and Tinelli, C. 2014. StarExec: a cross-community infrastructure for logic solving. In *Automated Reasoning*. Springer. 367–373.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.