Combining VSIDS and CHB Using Restarts in SAT

Mohamed Sami Cherif
Aix-Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

Djamal Habet
Aix-Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

Cyril Terrioux
Aix-Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

Abstract

Conflict Driven Clause Learning (CDCL) solvers are known to be efficient on structured instances and manage to solve ones with a large number of variables and clauses. An important component in such solvers is the branching heuristic which picks the next variable to branch on. In this paper, we evaluate different strategies which combine two state-of-the-art heuristics, namely the Variable State Independent Decaying Sum (VSIDS) and the Conflict History-Based (CHB) branching heuristic. These strategies take advantage of the restart mechanism, which helps to deal with the heavy-tailed phenomena in SAT, to switch between these heuristics thus ensuring a better and more diverse exploration of the search space. Our experimental evaluation shows that combining VSIDS and CHB using restarts achieves competitive results and even significantly outperforms both heuristics for some chosen strategies.

2012 ACM Subject Classification Computing methodologies → Artificial intelligence

Keywords and phrases Satisfiability, Branching Heuristic, Restarts

Digital Object Identifier 10.4230/LIPIcs.CP.2021.20

Funding Supported by the French National Research Agency, project ANR-16-CE40-0028.

1 Introduction

Given a CNF Boolean formula ϕ, solving the Satisfiability (SAT) problem consists in determining whether there exists an assignment of the variables which satisfies ϕ. SAT is at the heart of many applications in different fields and is used to model a large variety of crafted and real-world problems [33, 17, 24]. It is the first decision problem proven to be NP-complete [16]. Nevertheless, modern solvers based on Conflict Driven Clause Learning (CDCL) [34] manage to solve instances involving a huge number of variables and clauses. An important component in such solvers is the branching heuristic which picks the next variable to branch on. The Variable State Independent Decaying Sum (VSIDS) [35] has been the dominant heuristic since its introduction two decades ago. Recently, Liang and al. devised a new heuristic for SAT, called Conflict History-Based (CHB) branching heuristic [29], and showed that it is competitive with VSIDS. In the last years, VSIDS and CHB have dominated the heuristics landscape as practically all the CDCL solvers presented in recent SAT competitions and races incorporate a variant of one of them.

In recent years, combining VSIDS and CHB has shown promising results. For instance, the MapleCOMSPS solver, which won several medals in the 2016 and 2017 SAT competitions, switches from VSIDS to CHB after a set amount of time, or alternates between both heuristics by allocating the same duration of restarts to each one [31, 28]. Yet, we still lack a thorough analysis on such strategies in the state of art as well as a comparison with new promising methods based on machine learning in the context of SAT solving. Indeed, recent research has also shown the relevance of machine learning in designing efficient search heuristics for SAT [29, 30, 25] as well as for other decision problems [42, 41, 36, 13]. One of the main challenges is defining a heuristic which can have high performance on any considered instance. It is well known that a heuristic can perform very well on a family of instances while failing drastically on another. To this end, several reinforcement learning techniques can be used, specifically under the Multi-Armed Bandit (MAB) framework, to pick an adequate heuristic among CHB and VSIDS for each instance. These strategies also take advantage of the restart mechanism in modern CDCL algorithms to evaluate each heuristic and choose the best one accordingly. The evaluation is usually achieved by a reward function, which has to estimate the efficiency of a heuristic by relying on information acquired during the runs between restarts. In this paper, we want to compare these different strategies and, in particular, we want to know whether incorporating strategies which switch between VSIDS and CHB can achieve a better result than both heuristics and bring further gains to practical SAT solving.

The paper is organized as follows. An overview of CDCL algorithms is given in Section 2. The heuristics VSIDS and CHB as well as the Multi-Armed Bandit Problem are recalled in Section 3. Different strategies to combine VSIDS and CHB through restarts are described in Section 4 and experimentally evaluated and compared in Section 5. Finally, we conclude and discuss future work in Section 6.

2 Preliminaries

Let X be the set of propositional variables. A literal l is a variable $x \in X$ or its negation $\overline{x}$. A clause is a disjunction of literals. A formula in Conjunctive Normal Form (CNF) is a conjunction of clauses. An assignment $I : X \rightarrow \{true, false\}$ maps each variable to a Boolean value and can be represented as a set of literals. A literal l is satisfied by an assignment I if $l \in I$, else it is falsified by I. A clause is satisfied by an assignment I if at least one of its literals is satisfied by I, otherwise it is falsified by I. A CNF formula is satisfiable if there exists an assignment I which satisfies all its clauses, else it is unsatisfiable. Solving the Satisfiability (SAT) problem consists in determining whether a given CNF formula is satisfiable.

Although SAT is NP-complete [16], Conflict Driven Clause Learning [34] (CDCL) solvers are surprisingly efficient and manage to solve instances involving a huge number of variables and clauses. Such solvers are based on backtracking algorithms which rely on powerful branching heuristics as well as several solving techniques, namely Boolean Constraint Propagation (BCP), clause learning and restarts among others. In each step, BCP is applied to simplify the formula by propagating literals in unit clauses, i.e. clauses with one literal. If BCP is no longer possible, a branching heuristic picks a variable based on information acquired throughout the search. More importantly, when a conflict is detected, i.e. a clause is falsified by the current assignment, the steps of the algorithm are retraced and clauses involved in the conflict are resolved until the First Unit Implication Point (FUIP) in the implication graph [34]. The clause produced by this process is learnt, i.e. added to the formula. This enables to avoid revisiting an explored subspace of the search tree. Restarts are also an important component in CDCL solvers, initially introduced to deal with the heavy-tailed phenomena in SAT [19]. At the beginning of each restart, the solver parameters and its data structures are reinitialized in order to start the search somewhere else in the search space without discarding learnt clauses. There are two main restart strategies namely geometric restarts [40] and Luby restarts [32]. Most modern CDCL solvers use Luby restarts as it was shown that this policy outperforms geometric restarts [20].

3 Related Work

3.1 Branching Heuristics for SAT

The branching heuristic is one of the most important components in modern CDCL solvers and has a direct impact on their efficiency. It can be considered as a function that ranks variables using a scoring function, updated throughout the search. In this section, we describe two of the main state-of-the-art branching heuristics that we will consider in this work.

3.1.1 VSIDS

The Variable State Independent Decaying Sum (VSIDS) [35] has been the most used heuristic since its introduction around two decades ago. This heuristic maintains a floating point score for each variable, called activity and initially set to 0. When a conflict occurs, the activity of some variables is bumped, i.e. increased by 1. Furthermore, the variable activities are decayed periodically, usually after each conflict. More precisely, variable activities are multiplied by a decaying factor in $]0, 1[$. There are several variants of VSIDS. For instance, MiniSat [18] bumps the activities of variables appearing in the learnt clause while Chaff [35] does it for all the variables involved in the conflict, i.e. the resolved variables including those in the learnt clause.

3.1.2 CHB

The Conflict History-Based (CHB) branching heuristic was recently introduced in [29]. This heuristic based on the Exponential Recency Weighted Average (ERWA) [38] favors the variables involved in recent conflicts as in VSIDS. CHB maintains a score (or activity) $Q(x)$ for each variable x, initially set to 0. The score $Q(x)$ is updated when a variable x is branched on, propagated, or asserted using ERWA as follows:

$$Q(x) = (1 - \alpha) \times Q(x) + \alpha \times r(x).$$

The parameter $0 < \alpha < 1$ is the step-size, initially set to 0.4 and decayed by 10^{-6} after every conflict to a minimum of 0.06. $r(x)$ is the reward value for variable x which can decrease or increase the likelihood of picking x. Higher rewards are given to variables involved in recent conflicts according to the following formula:

$$r(x) = \frac{multiplier}{Conflicts - lastConflict(x) + 1}.$$

$Conflicts$ denotes the number of conflicts that occurred since the beginning of the search. $lastConflict(x)$ is updated to the current value of $Conflicts$ whenever x is present in the clauses used by conflict analysis. $multiplier$ is set to 1.0 when branching, propagating or asserting the variable that triggered the score update lead to a conflict, else it is set to 0.9. The idea is to give extra rewards for variables producing a conflict.

3.2 Multi-Armed Bandit Problem

A Multi-Armed Bandit (MAB) is a reinforcement learning problem consisting of an agent and a set of candidate arms from which the agent has to choose while maximizing the expected gain. The agent relies on information in the form of rewards given to each arm and collected through a sequence of trials. An important dilemma in MAB is the tradeoff between exploitation and exploration as the agent needs to explore underused arms often enough to have a robust feedback while also exploiting good candidates which have the best rewards. The first MAB model, stochastic MAB, was introduced in [26] then different policies have been devised for MAB [1, 4, 6, 38, 39]. In recent years, there was a surge of interest in applying reinforcement learning techniques and specifically those related to MAB in the context of SAT solving. In particular, CHB [29] and LRB [30] (a variant of CHB) are based on ERWA [38] which is used in non-stationary MAB problems to estimate the average rewards for each arm. Furthermore, a new approach, called Bandit Ensemble for parallel SAT Solving (BESS), was devised in [27] to control the cooperation topology in parallel SAT solvers, i.e. pairs of units able to exchange clauses, by relying on a MAB formalization of the cooperation choices. MAB frameworks were also extensively used in the context of Constraint Satisfaction Problem (CSP) solving to choose a branching heuristic among a set of candidate ones at each node of the search tree [42] or at each restart [41, 13]. Finally, simple bandit-driven perturbation strategies to incorporate random choices in constraint solving with restarts were also introduced and evaluated in [36]. The MAB framework we introduce in the context of SAT in Section 4.2 is closely related to those introduced in [41, 36] in the sense that we also pick an adequate heuristic at each restart. In particular, our framework is closer to the one in [36] in terms of the number of candidate heuristics and the chosen reward function and yet it is different in the sense that we consider two efficient state-of-the-art heuristics instead of perturbing one through random choices which may deteriorate the efficiency of highly competitive SAT solvers.

4 Strategies to Combine VSIDS and CHB Using Restarts

In this section, we describe different strategies which take advantage of the restart mechanism in SAT solvers to combine VSIDS and CHB. First, we describe simple strategies which are either static or random. Then, we describe reinforcement learning strategies, in the context of a MAB framework, which rely on information acquired through the search to choose the most relevant heuristic at each restart.

4.1 Static and Random Strategies

Hereafter, we describe three different strategies, one of which is random while the other two are static. These strategies are defined as follows:

- **Random Strategy (RD_R):** This strategy randomly picks a heuristic among VSIDS and CHB at each restart with equal probabilities, i.e. each heuristic is assigned a probability of $\frac{1}{2}$. This strategy is denoted RD_R in contrast with RD_D which randomly picks a heuristic at each decision.
- **Single Switch Strategy (SS):** This strategy switches from VSIDS to CHB after a set amount of time and was used in the 2016 version of MapleCOMSPS [31]. We maintain the threshold time in which the heuristic is switched to $\frac{t}{2}$ where t is the timeout as in [31].
- **Round Robin Strategy (RR):** This strategy alternates between VSIDS and CHB in the form of a round robin. This is similar to the strategy used in the latest version of MapleCOMSPS [28]. However, since we want to consider strategies which are independent from the restart policy and which only focus on choosing the heuristics, we do not assign equal amounts of restart duration (in terms of number of conflicts) to each heuristic and, instead, let the duration of restarts augment naturally with respect to the restart policy of the solver.

4.2 Multi-Armed Bandit Strategies

In order to use MAB strategies, we first introduce a MAB framework for SAT. Let $A = \{a_1, \ldots, a_K\}$ be the set of arms for the MAB containing different candidate heuristics. The trials are the runs, i.e. executions, of the backtracking algorithm between restarts. The proposed framework selects a heuristic a_i where $i \in \{1 \ldots K\}$ at each restart of the backtracking algorithm according to two different strategies that we will describe below. To choose an arm, MAB strategies generally rely on a reward function calculated during each run to estimate the performance of the chosen arm. The reward function plays an important role in the proposed framework and has a direct impact on its efficiency. We choose a reward function that estimates the ability of a heuristic to reach conflicts quickly and efficiently. If t denotes the current run, the reward of arm $a \in A$ is calculated as follows:

$$r_t(a) = \frac{log_2(decisions_t)}{decidedVars_t}.$$

$decisions_t$ and $decidedVars_t$ respectively denote the number of decisions and the number of variables fixed by branching in the run t. Consequently, the earlier conflicts are encountered in the search tree and the fewer variables are instantiated, the greater the assigned reward value will be for the corresponding heuristic. $r_t(a)$ is clearly in $[0, 1]$ since $decisions_t \leq 2^{decidedVars_t}$. This reward function is adapted from the explored sub-tree measure introduced in [36].

Next, we describe strategies for MAB which belong to a family of well know strategies, referred to as Upper Confidence Bound (UCB) [1, 5, 4]. For this family, the following parameters are maintained for each candidate arm $a \in A$:

- $n_t(a)$ is the number of times the arm a is selected during the $t-1$ previous runs,
- $\widehat{r}_t(a)$ is the empirical mean of the rewards of arm a over the $t-1$ previous runs.

We consider two UCB strategies, UCB1 and MOSS (Minimax Optimal Strategy in the Stochastic case). These strategies select the arm $a \in A$ that respectively maximizes $UCB1(a)$ and $MOSS(a)$ defined below. The left-side terms of $UCB1(a)$ and $MOSS(a)$ are identical and aim to put emphasis on arms that received the highest rewards. Conversely, the right-side terms ensure the exploration of underused arms. The main difference between UCB1 and MOSS is that the latter also takes into account the number of arms K.

$$UCB1(a) = \widehat{r}_t(a) + \sqrt{\frac{4.ln(t)}{n_t(a)}}$$

$$MOSS(a) = \widehat{r}_t(a) + \sqrt{\frac{4}{n_t(a)} ln\left(max\left(\frac{t}{K.n_t(a)}, 1\right)\right)}$$

Finally, a strategy for MAB is evaluated by its expected cumulative regret, i.e. the difference between the cumulative expected value of the reward if the best arm is used at each restart and its cumulative value for all the runs. The expected cumulative regret R_T is formally defined below, where $a_t \in A$ denotes the arm chosen at run t and T denotes the total number of runs. In particular, UCB1 and MOSS respectively guarantee an expected cumulative regret no worse than $O(\sqrt{K.T.\ln T})$ and $O(\sqrt{K.T})$ [5, 4].

$$R_T = \max_{a \in A} \sum_{t=1}^{T} \mathbf{E}[r_t(a)] - \sum_{t=1}^{T} \mathbf{E}[r_t(a_t)]$$

5 Experimental Evaluation

In this section, we describe our experimental protocol and then we evaluate and compare the different strategies presented in Section 4.

5.1 Experimental Protocol

We consider the benchmarks from the Main Track of the last three SAT Competitions/Races, totalling to 1,200 instances. For our experiments, we use the state-of-the-art solver Kissat [10] which won first place in the main track of the SAT Competition 2020. Note that this solver is a condensed and improved reimplementation of the reference and competitive solver CaDiCaL [9, 10] in C. Data provided by Armin Bierre and Marjin Heule[1] show that Kissat is highly competitive and outperforms all-time winners of SAT competitions/Races particularly on the 2020 and 2019 Benchmarks. Kissat alternates between stable and non-stable phases as is the case in Cadical [9], renamed to stable mode and focused mode in [10]. VSIDS is used in stable phases which mainly target satisfiable instances. During non-stable phases targeting unsatisfiable instances, the solver uses the Variable Move-To-Front (VMTF) heuristic [37, 12], in which analyzed variables are moved to the front of the decision queue. It is important to note that the only modified components of the solver are the decision component and the restart component, i.e. all the other components as well as the default parameters of the solver are left untouched. Even the changes to the restart component are as minimal as possible, i.e. we maintain the phase alternation mechanism and the restart policies set for each mode as described in [10]. Furthermore, we maintain the VSIDS variant already implemented in Kissat, called Exponential VSIDS (EVSIDS) [8, 12], which is based on Chaff's where all analyzed variables are bumped after every conflict. Therefore, in the experimental evaluation, $VSIDS$ corresponds to default Kissat. Moreover, we augment the solver with the heuristic CHB as specified in [29] except that we update the scores of the variables in the last decision level after BCP. In addition, we have implemented the MAB framework specified in Section 4 with $A = \{VSIDS, CHB\}$. The rewards for UCB1 and MOSS are both initialized by launching each heuristic once during the first restarts. Finally, The experiments are performed on Dell PowerEdge M620 servers with Intel Xeon Silver E5-2609 processors under Ubuntu 18.04 with a timeout of 5,000 s for each instance.

5.2 Decisions vs Restarts

First, we would like to emphasize that taking advantage of the restart mechanism to combine VSIDS and CHB was not an arbitrary choice. Indeed, we conducted an experiment to help us choose the appropriate level, i.e. decisions or restarts, to combine VSIDS and CHB. To this end, we implemented and tested the two random strategies RD_D and RD_R which randomly chose a heuristic among VSIDS and CHB respectively in each decision and in each restart. The average results (over 10 runs with different seeds) of RD_D and RD_R on the whole benchmark are reported in Table 1 and indicate that RD_R outperforms RD_D with a gain of more than 2% in terms of solved instances and 3.5% in terms of solving time with a penalty of 10,000 s for unsolved instances. This is not surprising as the structures needed for VSIDS and CHB need to be maintained and updated simultaneously which can be quite costly. On the other hand, they are used independently in RD_R during each restart, i.e. only the chosen heuristic is used and its structures updated during the restart. Furthermore, combining both

[1] Data available on `http://fmv.jku.at/kissat/`

Table 1 Comparison between VSIDS, CHB, the different strategies and the VBS (over VSIDS and CHB) in terms of the number of solved instances in Kissat. For each row, the best results without considering the VBS are written in bold.

		VSIDS	**CHB**	**RD$_D$**	**RD$_R$**	**SS**	**RR**	**UCB1**	**MOSS**	**VBS**
Competition 2018 (400 instances)	SAT	160	159	160	164	163	165	167	**168**	169
	UNSAT	111	109	109	110	**113**	110	110	110	113
	TOTAL	271	268	268	274	276	275	277	**278**	282
Race 2019 (400 instances)	SAT	158	149	155	158	154	**162**	161	**162**	162
	UNSAT	**97**	95	95	96	96	96	96	**97**	99
	TOTAL	255	244	250	254	250	258	257	**259**	261
Competition 2020 (400 instances)	SAT	131	146	146	151	147	152	154	**156**	157
	UNSAT	121	119	117	120	118	120	120	**122**	123
	TOTAL	252	265	263	271	265	272	274	**278**	280
TOTAL (1,200 instances)	SAT	449	454	461	473	464	479	482	**486**	488
	UNSAT	**329**	323	321	326	327	326	326	**329**	335
	TOTAL	778	777	782	799	791	805	808	**815**	823

heuristics at the decision level can cause interference and may not allow each heuristic to conduct robust learning since they are being constantly interchanged. Surprisingly, both versions are competitive with CHB and VSIDS. In particular, RD_R outperforms them and solves, on average, 21 additional instances (+ 2.7%) compared to the best heuristic. This is due to randomization and diversification which help to avoid heavy tail phenomena in SAT and which can therefore improve the performance of SAT solvers [21, 19].

5.3 Comparison of Strategies

5.3.1 Number of Solved Instances

In Table 1, we present the results in terms of solved instances for CHB and VSIDS as standalone heuristics and for the different strategies presented in Section 4. We also include the results of the Virtual Best Solver (VBS) over VSIDS and CHB. Before discussing the results, we recall 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" [3].

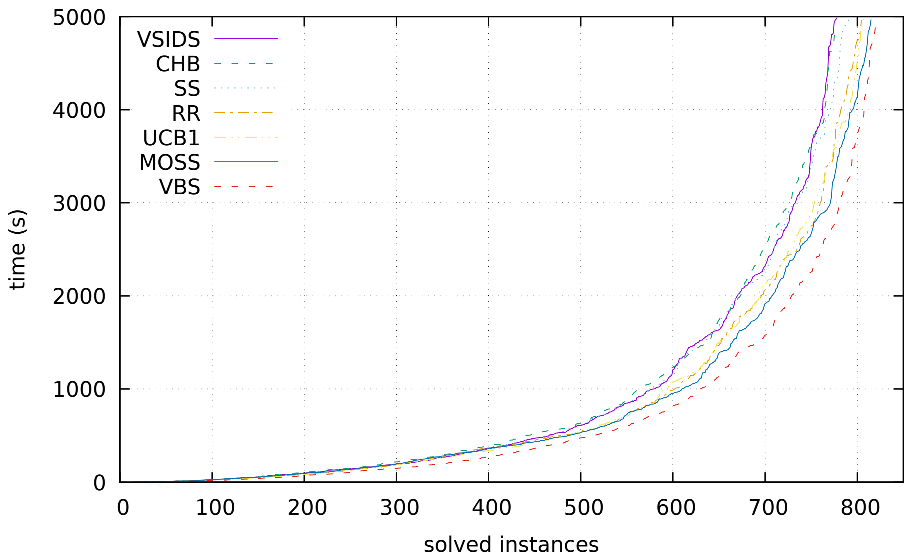
The results clearly indicate that MOSS outperforms VSIDS and CHB as well as all the other strategies. Indeed, MOSS manages to solve 37 additional instances in total (+4.8%) compared to the best heuristic (among VSIDS and CHB). The UCB1 (resp. RR) strategy is also competitive and manages to solve 30 (resp. 27) additional instances in total which corresponds to an increase of 3.9% (resp. 3.5%) in terms of solved instances compared to the best heuristic. The strategies UCB1 and RR remain comparable with a difference of 3 instances in favor of UCB1. SS also outperforms VSIDS and CHB although to a lesser degree as it solves 13 additional instances only which is worse than RD_R. If we focus on the individual yearly benchmarks, we observe that although the overall results obtained by VSIDS and CHB are comparable, they have different behaviours on each benchmark and yet MOSS, UCB1 and RR manage to capture the behaviour of the best heuristic and even outperform it on each individual benchmark. In particular, MOSS maintains its top rank on the individual benchmarks with an average of 8 (resp. 17) additional instances for each one compared to the best (resp. worst) heuristic. Moreover, the results achieved by MOSS are very close to the VBS. Indeed it achieves 99% (resp. 99.6%) of the performance of the VBS on the whole benchmark in terms of the number of solved instances (resp. satisfiable instances) while the best heuristic does not exceed 95% (resp. 93%).

However, it is important to note that the gain is mainly in satisfiable instances whereas, for unsatisfiable instances, all the strategies (except RD_D) remain comparable to both heuristics and slightly outperform CHB but not VSIDS. Nevertheless, they remain competitive with VSIDS and particularly MOSS which solves the same number of unsatisfiable instances as VSIDS. This shows that MOSS is a robust strategy as it is able to improve the performance globally and on each individual benchmark without decreasing it for unsatisfiable instances. Note that the observed behaviour of these different strategies on unsatisfiable instances may be due to different factors. First, the results in terms of unsatisfiable instances seem very homogeneous for each year and are very close to the results obtained by the VBS as both heuristics (resp. the best heuristic) achieve more than 96% (resp. 98%) of its performance in terms of the number of unsatisfiable instances. Since our motivation is to bridge the gap between the heuristics and the VBS with these strategies, it is expected that this would be very difficult for unsatisfiable instances, for which the gap is very small already. It is also very difficult to simultaneously improve the performance on both satisfiable and unsatisfiable instances. Notice how SS which seems to work better for unsatisfiable instances especially in terms of solving time (refer to Section 5.3.2) fails on satisfiable instances compared to the three top strategies. Another possible factor for this behaviour is Kissat's restarting policy which alternates between the stable mode and focused mode [10]. The heuristics VSIDS and CHB are only used in the stable mode while the focused mode targets unsatisfiable instances. This may also help to explain the homogeneity of the results obtained by the solver for unsatisfiable instances with respect to the different heuristics and strategies.

5.3.2 Solving Time

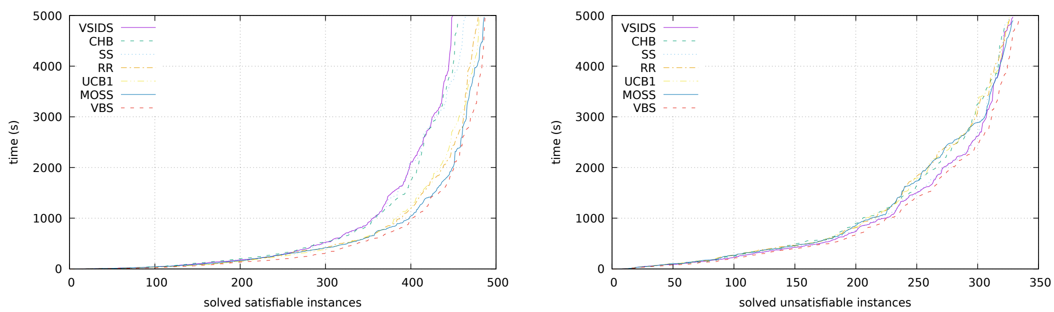
In this section, we want to evaluate the different strategies in terms of solving time. In Figure 1, we represent the number of solved instances as a function of the CPU time for VSIDS, CHB, the static and MAB strategies and the VBS on the whole benchmark. One would think that MAB based strategies in this regard would be worse than the considered heuristics and/or other strategies as UCB1 and MOSS need to conduct continuous exploration in order to ensure the selection of the most adequate arm. This does not seem to be the case. In fact, conducting exploitation with the best arm and alternating the heuristics seems to offset this disadvantage. We observe that MOSS is the best strategy as it achieves 6.1% gain in terms of solving time on the whole benchmark compared to the best heuristic if we give a penalty 10,000 s to unsolved instances while UCB1, RR and SS respectively achieve a gain of 5.7%, 5.2% and 1.7%. This gain is substantial especially considering that we are working on the solver Kissat which won the SAT competition 2020 with a remarkable performance.

We represent in Figure 2 the number of solved satisfiable and unsatisfiable instances separately as a function of the CPU time for VSIDS, CHB, the static and MAB strategies and the VBS on the whole benchmark. Notice how the gap between MOSS and the VBS (and even UCB1 and RR) narrows if we consider the satisfiable instances only. On the other hand, these top three strategies present a small gap in terms of solving time for unsatisfiable instances compared to the best heuristic, i.e. VSIDS, while remaining comparable to CHB. In particular, MOSS shows better results with respect to VSIDS and SS for instances whose solving time exceeds 4,000 s. Surprisingly, although SS seems to be the worst strategy overall and remains globally comparable to VSIDS and CHB while achieving a slight gain in solving time especially on instances whose solving time exceeds 4,000 s, it achieves the best results in terms of solving time for unsatisfiable instances and is comparable to VSIDS and the VBS in this regard. On the other hand, RR and UCB1 achieve substantial gain while remaining comparable to each other and with results slightly in favor of UCB1. To provide more detailed results, we represent in Figures 3, 4 and 5 the runtime comparison per instance with VSIDS, CHB and the VBS respectively for the top three best strategies, i.e. MOSS, UCB1 and RR. These figures confirm the trends that we observed above. More interesting, we can note that, for a noticeable number of instances, MOSS, UCB1 or RR lead to a more efficient solving than the VBS. In Figure 6, we represent the runtime comparison per instance betwenn MOSS, UCB1 and RR. These figures show that MOSS performs better than UCB1 and RR. Surprisingly, MOSS's results are closer to RR than UCB1. However, we will show in Section 5.3.4 that this is consistent with the observed behaviour of MOSS.



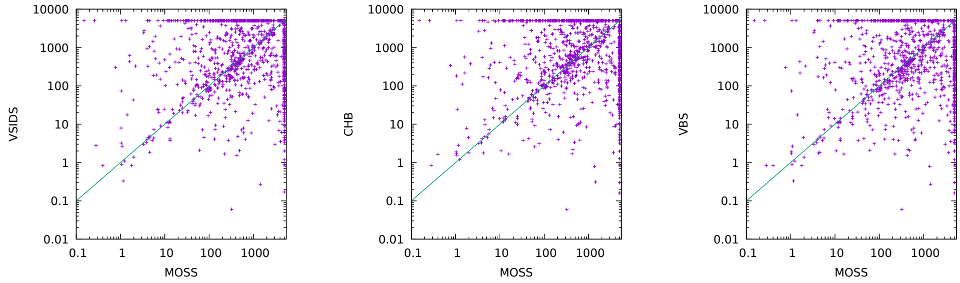
1

Figure 1 Number of solved instances as a function of CPU time for VSIDS, CHB, static and MAB strategies and the VBS with respect to the whole benchmark.



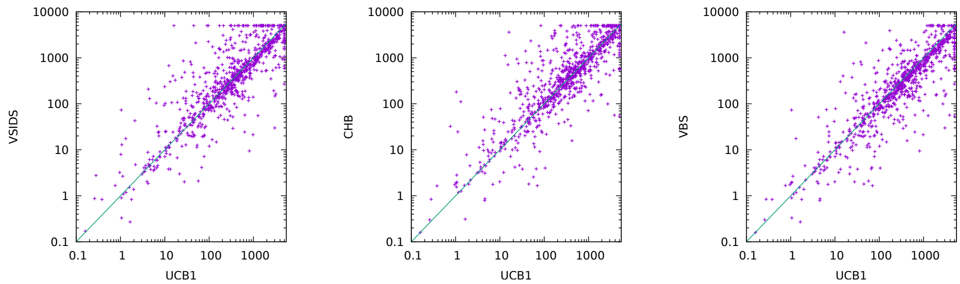
(a) (b)

Figure 2 Number of solved satisfiable (a) and unsatisfiable (b) instances as a function of CPU time for VSIDS, CHB, static and MAB strategies and the VBS w.r.t the whole benchmark.



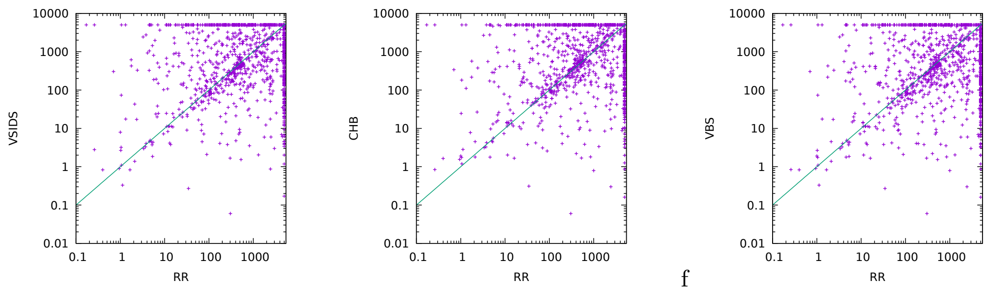
(a) (b) (c)

Figure 3 Runtime comparison (in seconds) of MOSS w.r.t. VSIDS (a), CHB (b) and VBS (c) in logarithmic scale.



(a) (b) (c)

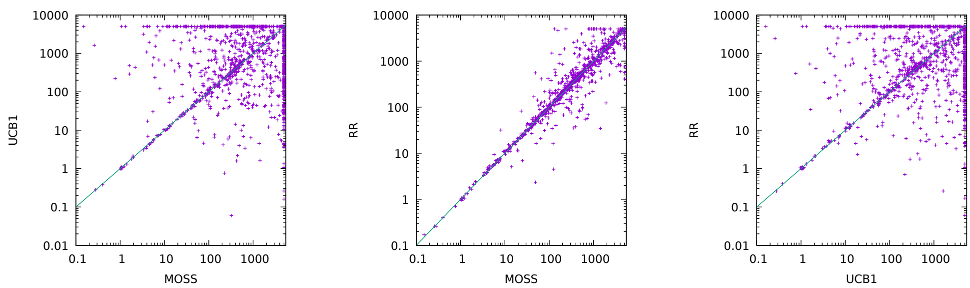
Figure 4 Runtime comparison (in seconds) of UCB1 w.r.t. VSIDS (a), CHB (b) and VBS (c) in logarithmic scale.



f

(a) (b) (c)

Figure 5 Runtime comparison (in seconds) of RR w.r.t. VSIDS (a), CHB (b) and VBS (c) in logarithmic scale.



(a) (b) (c)

Figure 6 Runtime comparison (in seconds) of MOSS w.r.t. UCB1 (a) and RR (b) and of UCB1 w.r.t. RR (c) in logarithmic scale.

Table 2 Comparison between VSIDS, CHB, static and MAB strategies and the VBS (over VSIDS and CHB) in terms of the number of solved instances (#I) and cumulative solving time (for solved instances in seconds) in Kissat for instance families in the benchmark. The results of families marked with † are joint from two different yearly benchmarks. For each row, the best results without considering the VBS are written in bold, breaking ties with milliseconds if necessary.

Family		VSIDS		CHB		SS		RR		UCB1		MOSS		VBS	
name	#I	#I	time	#I	time	#I	time	#I	time	#I	time	#I	time	#I	time
`Antibandwidth`	14	2	1,010	7	9,804	7	21,381	8	11,469	9	15,651	**9**	**14,370**	7	9,628
`Almost Perfect Non-Linear S-box Finder`	20	**11**	**15,324**	7	14,386	11	18,815	10	18,974	11	19,659	11	15,969	12	16,138
`Arithmetic Verification`	38	13	11,010	**14**	**12,268**	8	6,657	9	12,160	13	13,349	9	11,811	14	10,801
`Baseball-lineup`	13	12	3,317	12	2,949	12	3,192	12	2,645	12	2,947	**12**	**2,540**	12	2,906
`Bitcoin`	17	8	1,972	7	479	8	2,001	8	1,907	8	2,199	**8**	**1,901**	8	1,784
`Coloring`	14	6	7,501	5	7,327	**7**	**12,097**	5	3,101	5	2,753	6	7,556	6	5,876
`Core-based`	14	13	8,438	13	10,860	13	8,737	13	7,431	**14**	**14,165**	14	14,861	13	7,239
`Course Scheduling`	20	14	14,439	14	15,363	13	11,205	14	11,804	**15**	**9,323**	15	10,801	14	9,654
`Cover`	13	**4**	**9**	4	10	4	9	4	10	4	10	4	11	4	9
`Chromatic Number (CNP)`	20	20	1,708	20	1,972	20	1,743	20	1,402	**20**	**1,180**	20	1,415	20	1,194
`Divide and Unique Inverse`	20	**16**	**18,456**	16	22,537	16	19,615	16	20,864	16	23,543	16	20,931	16	18,109
`Discrete-Logarithm`	7	4	3,640	4	7,623	**4**	**3,344**	4	4,718	4	4,894	4	5,883	4	3,627
`Edge-Matching Puzzle` †	14	3	4,476	3	6,201	2	1,430	**4**	**5,546**	4	8,674	4	6,615	4	8,823
`Factoring` †	32	30	17,224	27	12,497	**30**	**16,554**	27	10,297	28	20,528	28	20,106	30	12,546
`Floating-Point Program Verification`	15	12	972	**12**	**874**	12	1,024	12	1,050	12	1,076	12	991	12	775
`Grand Tour Puzzle`	19	9	1,834	9	2,037	**9**	**1,771**	9	2,042	9	2,061	9	2,153	9	1,783
Hard 3-SAT	20	18	5,486	19	8,888	18	6,424	**19**	**3,654**	18	3,643	19	4,042	19	7,238
`Hgen`	13	12	3,168	12	2,423	12	3,134	12	1,783	**12**	**783**	12	2,590	12	2,365
`Influence Maximization`	14	12	9,617	**12**	**7,424**	12	9,472	12	10,037	12	9,989	12	10,152	12	6,865
`Kakuro Puzzle`	14	**12**	**15,705**	11	13,942	11	10,312	12	16,365	12	16,269	12	16,216	12	14,582

Table 3 Comparison between VSIDS, CHB, static and MAB strategies and the VBS (over VSIDS and CHB) in terms of the number of solved instances (#I) and cumulative solving time (for solved instances in seconds) in Kissat for some instance families in the benchmark (Table 2 continued). The results of families marked with † are joint from two different yearly benchmarks. For each row, the best results without considering the VBS are written in bold, breaking ties with milliseconds if necessary.

Family		VSIDS		CHB		SS		RR		UCB1		MOSS		VBS	
name	#I	#I	time	#I	time	#I	time	#I	time	#I	time	#I	time	#I	time
k-Colorability	15	5	7,923	5	6,528	**6**	**12,338**	5	5,167	5	6,998	5	5,540	6	10,660
Keystream Generator Cryptanalysis	18	18	14,494	14	15,104	18	14,731	18	19,433	**18**	**13,118**	18	16,828	18	13,240
Lam-Discrete-Geometry	9	**7**	**4,756**	7	7,106	7	4,899	7	7,147	7	6,856	7	6,953	7	4,680
Logical Cryptanalysis	20	20	5,606	20	10,476	20	6,748	20	5,518	20	4,241	**20**	**4,208**	20	4,946
Polynomial Multiplication †	27	20	28,884	21	27,880	21	33,016	21	27,107	20	23,653	**22**	**30,535**	25	41,331
Population Safety	15	13	2,188	**14**	**1,991**	13	2,423	12	1,624	13	3,148	13	2,417	14	1,809
Preimage	11	6	11,865	4	7,998	**7**	**13,070**	6	16,201	5	9,255	5	5,852	8	15,435
Relativized Pigeonhole Principle (RPHP)	20	11	14,890	10	11,344	11	15,229	10	9,869	11	14,065	**11**	**13,967**	11	14,890
Reversing Elementary Cellular Automata	11	11	4,046	11	4,065	**11**	**3,923**	11	4,738	11	5,151	11	4,476	11	3,664
Scrambled	20	19	7,022	18	9,278	**20**	**11,441**	19	8,114	19	14,519	20	15,141	20	6,089
SHA-1 Pre-image Attack	20	20	14,509	20	23,429	**20**	**14,085**	19	21,975	20	25,206	20	20,255	20	12,214
Social Golfer	14	2	3,008	1	3,791	1	410	2	1,202	**3**	**6,046**	2	1,530	2	3,008
Software Bounded Model Checking	19	**18**	**7,082**	18	8,959	18	7,219	18	7,966	18	8,308	18	8,435	18	6,969
Station Repacking	12	6	15,286	12	10,656	11	29,669	12	13,752	12	8,766	**12**	**6,855**	12	10,656
Stedman Triples †	27	10	8,766	11	11,508	11	11,394	12	8,215	12	7,399	**13**	**12,329**	11	6,947
SV Competition	18	18	7,522	17	3,350	17	4,227	**18**	**7,251**	18	7,935	18	7,829	18	5,577
Timetable †	26	1	1565	10	5082	10	28,417	11	5,607	11	6,247	**11**	**5,157**	10	5,082
Tree Decomposition	20	11	12,049	10	6,870	10	7,947	**11**	**4,700**	10	3,965	11	5,674	11	7,874
Vlsat	14	3	103	7	4,457	4	3,404	7	529	**7**	**500**	7	547	7	3,934

5.3.3 Instance Families

In order to provide a more thorough analysis, we describe in Tables 2 and 3 the results obtained by VSIDS, CHB, static and MAB strategies and the VBS on instance families within the benchmark [23, 22, 7]. The best strategy, i.e. MOSS, manages to rank first in 9 different families over 39 in total (23%), e.g. `Antibandwidth`, `Bitcoin` and `Stedman Triples`. Interestingly, this strategy achieves remarkable results, which are better than those of the VBS over VSIDS and CHB, for certain families such as `Logical cryptanalysis`, `RPHP` and `Station Repacking`. SS also achieves the top performance on several different families such as `Factoring`, `Scrambled` and `SHA-1 Pre-image Attack`. More precisely, SS also manages to rank on top for 9 different families which shows the interest of this strategy even though it ranks last overall compared to RR, UCB1 and MOSS. As for UCB1, it achieves top rank in 6 different families. In particular, its performance on the families `Hgen`, `CNP` and `Keystream Generator Cryptanalysis` is noteworthy since it manages to outperform the VBS. On the other hand, RR ranks top in only 4 instance families but this does not necessarily reflect its overall performance since it falls slightly behind the top ranked heuristic/strategy in other families, yet this is clearly another point in favor of UCB1 as a comparable strategy. Finally, VSIDS and CHB are ranked first in several families which shows that these heuristics remain robust as standalone heuristics.

5.3.4 MAB Behaviour

In this section, we focus on the behaviour of MAB strategies and particularly the use of arms. In Figures 7 and 8, we represent the percentage of use, i.e. percentage of restarts where each arm gets chosen respectively by UCB1 and MOSS. We observe that both strategies alternate between the heuristics but the percentages are mainly within the interval $[40\%, 60\%]$ and are often close to 50%. MOSS seems to choose in a more balanced way between VSIDS and CHB in comparison to UCB1 which introduces more variations in its choices. This behaviour is consistent with the observations made in Section 5.3.1 concerning Figure 6. The fact that the percentages are mostly within a tight interval is not surprising considering that the number of stable restarts in Kissat, during which heuristics are used, is usually very low. To give an idea, the average number of stable restarts performed by Kissat for instances solved with MOSS (resp. UCB1) is 765 (resp. 771) while the median value is much lower and amounts to 313 (resp. 338). Therefore, the obtained percentages seem adequate especially taking into account that these strategies need to achieve a good trade-off between exploration and exploitation. Notice the consecutive dents and bumps in Figures 7 and 8 which correspond to an homogeneous behaviour within the same instance family in the benchmark. It is important to note that, although the behaviour of MAB strategies may seem close to RR, this is not exactly the case. Indeed, these strategies rely on the computed reward to choose the most relevant arm during exploitation and especially when there is a large gap between the performance of the heuristics, whereas RR is a static strategy and cannot adapt its choices. This helps to explain the better results of the MAB strategies not only in terms of solved instances but also in terms of solving time and particularly in the case of MOSS. In fact, the remarkable performance of MOSS is also due to the fact that it takes into account the number of arms and has better regret than UCB1.

5.4 On MAB strategies and Branching Heuristics

In this section, we discuss the relevance of choosing Upper Confidence Bound strategies in the Multi-Armed Bandit framework and VSIDS and CHB as candidate heuristics. As mentioned in Section 3.2, many strategies were divised and theoretically studied in the context of

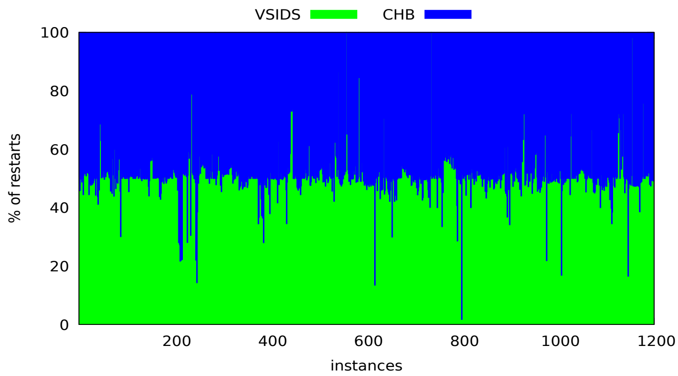


Figure 7 Percentages of use of each arm in UCB1 w.r.t the whole benchmark. The instances are reported consecutively for each yearly benchmark (from 2018 to 2020) and are alphabetically ordered. For unsolved instances, the percentages of use at the timeout are provided.

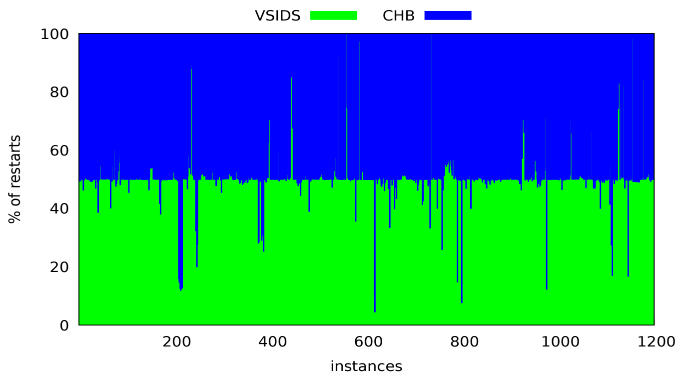


Figure 8 Percentages of use of each arm in MOSS w.r.t the whole benchmark. The instances are reported consecutively for each yearly benchmark (from 2018 to 2020) and are alphabetically ordered. For unsolved instances, the percentages of use at the timeout are provided.

MAB and can therefore be used in our framework. For instance, we can mention two other well-know strategies for MAB: ϵ-greedy [38] and EXP3 [6]. However, these strategies are not deterministic, i.e. there is a factor of uncertainty or probability. Therefore, unlike UCB strategies, they cannot always guarantee top performance and may produce different results on the same benchmark. Furthermore, UCB strategies were shown relevant and more efficient for similar MAB frameworks in the context of CSP [36, 41, 13]. This remains true in Kissat as we observed, through extensive experimentation, that ϵ-greedy and EXP3 perform poorly compared to UCB strategies and remain comparable to VSIDS and CHB.

In addition, notice that the MAB framework enables the use of several heuristics. In fact, one would argue that adding more heuristics may enable to reach more families and instances through diversification. However, recall that modern SAT solvers, and in particular Kissat, are highly competitive and rely on powerful heuristics to achieve impressive results. A bad heuristic or tuning of the parameters (e.g. the restart policy settings) can greatly deteriorate the performance of a solver. Furthermore, practically all heuristics used in modern SAT solvers are variants of VSIDS, which has been the dominant heuristic since its introduction in 2001 [35]. Only recently CHB has been introduced and shown competitive with VSIDS [29]. CHB has only one variant called LRB [30] but, through extensive experimentation, CHB turned out to be more robust with respect to different solvers and settings. The results reported in Table 1 also show that CHB can reach new instances (the VBS achieves a gain of more than 5.8% in terms of solved instances) while remaining competitive and comparable overall with respect to VSIDS in the context of a highly competitive solver such as Kissat.

5.4.1 Kissat_MAB at the SAT Competition 2021

We submitted the solver Kissat augmented with a MAB framework relying on the UCB1 strategy to the SAT competition 2021[2] under the name Kissat_MAB [14]. This solver won the Main Track of the competition and managed to solve 296 instances over 400 with a gap of 8 instances compared to the second ranked solver. Kissat_MAB also placed first in the Main SAT and NoLimits tracks. Compared to default Kissat, which also participated in the competition under the name Kissat_sc2021_default with several new improvements over its last version [11], Kissat_MAB achieves better results with 9 (resp. 11) additional solved (resp. satisfiable) instances. Furthermore, Kissat_MAB remains highly competitive on unsatisfiable instances and comparable to default Kissat as it managed to solve 148, only 2 instances less than Kissat_sc2021_default. Notice that this gap can clearly be narrowed or even turned in favor of Kissat_MAB if the MOSS strategy is used as shown in our experimental evaluation. To summarize, the results of the SAT competition 2021 seem to corroborate our experimental study and to confirm the relevance of combining VSIDS and CHB using restarts in improving the performance of highly competitive SAT solvers.

6 Conclusion and Future Work

In this paper, we evaluated different strategies which take advantage of the restart mechanism to combine two state of the art heuristics, namely VSIDS and CHB. In particular, we introduced a MAB framework for SAT and chose two known Upper Confidence Bound strategies, called UCB1 and MOSS. These strategies rely on a reward function which evaluates the capacity of the heuristics to reach conflicts quickly and efficiently. Our experimental evaluation shows that VSIDS and CHB are compatible since their combination through different strategies taking advantage of the restart mechanism is able to substantially increase the performance of the competitive solver Kissat. In particular, the MOSS strategy outperforms not only VSIDS and CHB but also all the other strategies. The strategies UCB1 and RR have also shown competitive results. These three strategies achieve substantial gain in terms of solved instances, mainly satisfiable ones, and in terms of solving time. Moreover, these strategies achieve results which are very close to the VBS over VSIDS and CHB. Our solver Kissat_MAB won the Main track of the SAT competition 2021 and placed first in the Main SAT and NoLimits tracks thus showing the relevance of combining VISDS and CHB using restarts and its ability to improve the performance of highly competitive SAT solvers.

As future work, it would be interesting to refine the reward function used in MAB strategies by relying on a combination of different criteria [15] so as to improve the MAB framework especially with respect to unsatisfiable instances. It would also be interesting to focus on one heuristic and try to refine it using a similar MAB framework, an approach which was shown relevant in the context of the Constraint Satisfaction Problem (CSP) [13]. Finally, it would be interesting to use these strategies to improve other components in modern SAT solvers such as clause deletion [2].

[2] Results and source code available on `https://satcompetition.github.io/2021/`.

References

1 Rajeev Agrawal. Sample mean based index policies by O(log n) regret for the multi-armed bandit problem. *Advances in Applied Probability*, 27(4):1054–1078, 1995. doi:10.2307/1427934.

2 Gilles Audemard and Laurent Simon. Predicting Learnt Clauses Quality in Modern SAT Solvers. In Craig Boutilier, editor, *Proceedings of the 21st International Joint Conference on Artifical Intelligence*, IJCAI'09, page 399–404, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc. URL: https://www.ijcai.org/Proceedings/09/Papers/074.pdf.

3 Gilles Audemard and Laurent Simon. Refining Restarts Strategies for SAT and UNSAT. In Michela Milano, editor, *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, volume 7514 of *Lecture Notes in Computer Science*, pages 118–126. Springer, 2012. doi:10.1007/978-3-642-33558-7_11.

4 Jean-Yves Audibert and Sébastien Bubeck. Minimax Policies for Adversarial and Stochastic Bandits. In *COLT 2009 - The 22nd Conference on Learning Theory, Montreal, Quebec, Canada, June 18-21, 2009*, 2009. URL: http://www.cs.mcgill.ca/%7Ecolt2009/papers/022.pdf.

5 Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time Analysis of the Multiarmed Bandit Problem. *Mach. Learn.*, 47(2-3):235–256, 2002. doi:10.1023/A:1013689704352.

6 Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. The nonstochastic multiarmed bandit problem. *SIAM journal on computing*, 32(1):48–77, 2002. doi:10.1137/S0097539701398375.

7 Tomáš Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors. *Proceedings of SAT Competition 2020: Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Series of Publications B*. Department of Computer Science, University of Helsinki, 2020. URL: https://helda.helsinki.fi/bitstream/handle/10138/318450/sc2020_proceedings.pdf.

8 Armin Biere. Adaptive Restart Strategies for Conflict Driven SAT Solvers. In Hans Kleine Büning and Xishun Zhao, editors, *Theory and Applications of Satisfiability Testing - SAT 2008, 11th International Conference, SAT 2008, Guangzhou, China, May 12-15, 2008. Proceedings*, volume 4996 of *Lecture Notes in Computer Science*, pages 28–33. Springer, 2008. doi:10.1007/978-3-540-79719-7_4.

9 Armin Biere. CaDiCaL, Lingeling, Plingeling, Treengeling, YalSAT Entering the SAT Competition 2017. In Tomáš Balyo, Marijn Heule, and Matti Järvisalo, editors, *Proc. of SAT Competition 2017 – Solver and Benchmark Descriptions*, volume B-2017-1 of *Department of Computer Science Series of Publications B*, pages 14–15. University of Helsinki, 2017.

10 Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020.

11 Armin Biere, Mathias Fleury, and Maximillian Heisinger. CADICAL, KISSAT, PARACOOBA Entering the SAT Competition 2021. In *Proceedings of SAT Competition 2021: Solver and Benchmark Descriptions*, volume B-2021-1 of *Department of Computer Science Series of Publications B*, pages 10–13. University of Helsinki, 2021 .

12 Armin Biere and Andreas Fröhlich. Evaluating CDCL Variable Scoring Schemes. In Marijn Heule and Sean A. Weaver, editors, *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings*, volume 9340 of *Lecture Notes in Computer Science*, pages 405–422. Springer, 2015. doi:10.1007/978-3-319-24318-4_29.

13 Mohamed Sami Cherif, Djamal Habet, and Cyril Terrioux. On the Refinement of Conflict History Search Through Multi-Armed Bandit. In *32nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2020, Baltimore, MD, USA, November 9-11, 2020*, pages 264–271. IEEE, 2020. doi:10.1109/ICTAI50040.2020.00050.

14 Mohamed Sami Cherif, Djamal Habet, and Cyril Terrioux. Kissat_MAB: Combining VSIDS and CHB through Multi-Armed Bandit. In *Proceedings of SAT Competition 2021: Solver and Benchmark Descriptions*, volume B-2021-1 of *Department of Computer Science Series of Publications B*, pages 15–16. University of Helsinki, 2021.

15 Wei Chu, Lihong Li, Lev Reyzin, and Robert E. Schapire. Contextual Bandits with Linear Payoff Functions. In Geoffrey J. Gordon, David B. Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, volume 15 of *JMLR Proceedings*, pages 208–214. JMLR.org, 2011. URL: `http://proceedings.mlr.press/v15/chu11a/chu11a.pdf`.

16 Stephen A. Cook. The Complexity of Theorem-Proving Procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, page 151–158, New York, NY, USA, 1971. Association for Computing Machinery. `doi:10.1145/800157.805047`.

17 Todd Deshane, Wenjin Hu, Patty Jablonski, Hai Lin, Christopher Lynch, and Ralph Eric McGregor. Encoding First Order Proofs in SAT. In Frank Pfenning, editor, *Automated Deduction - CADE-21, 21st International Conference on Automated Deduction, Bremen, Germany, July 17-20, 2007, Proceedings*, volume 4603 of *Lecture Notes in Computer Science*, pages 476–491. Springer, 2007. `doi:10.1007/978-3-540-73595-3_35`.

18 Niklas Eén and Niklas Sörensson. An Extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003. `doi:10.1007/978-3-540-24605-3_37`.

19 Carla P Gomes, Bart Selman, Nuno Crato, and Henry Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of automated reasoning*, 24(1-2):67–100, 2000. `doi:10.1023/A:1006314320276`.

20 Shai Haim and Marijn Heule. Towards Ultra Rapid Restarts. *CoRR*, abs/1402.4413, 2014. `arXiv:1402.4413`.

21 William D. Harvey and Matthew L. Ginsberg. Limited Discrepancy Search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*, pages 607–615. Morgan Kaufmann, 1995. URL: `http://ijcai.org/Proceedings/95-1/Papers/080.pdf`.

22 Marijn Heule, Matti Järvisalo, and Martin Suda, editors. *Proceedings of SAT Race 2019: Solver and Benchmark Descriptions*, volume B-2019-1 of *Department of Computer Science Series of Publications B*. Department of Computer Science, University of Helsinki, 2019. URL: `https://helda.helsinki.fi/bitstream/handle/10138/306988/sr2019_proceedings.pdf`.

23 Marijn Heule, Matti Juhani Järvisalo, Martin Suda, et al., editors. *Proceedings of SAT Competition 2018: Solver and Benchmark Descriptions*, volume B-2018-1 of *Department of Computer Science Series of Publications B*. Department of Computer Science, University of Helsinki, 2018. URL: `https://helda.helsinki.fi/bitstream/handle/10138/237063/sc2018_proceedings.pdf`.

24 Ted Hong, Yanjing Li, Sung-Boem Park, Diana Mui, David Lin, Ziyad Abdel Kaleq, Nagib Hakim, Helia Naeimi, Donald S. Gardner, and Subhasish Mitra. QED: Quick Error Detection tests for effective post-silicon validation. In Ron Press and Erik H. Volkerink, editors, *2011 IEEE International Test Conference, ITC 2010, Austin, TX, USA, November 2-4, 2010*, pages 154–163. IEEE Computer Society, 2010. `doi:10.1109/TEST.2010.5699215`.

25 Vitaly Kurin, Saad Godil, Shimon Whiteson, and Bryan Catanzaro. Improving SAT Solver Heuristics with Graph Networks and Reinforcement Learning. *CoRR*, abs/1909.11830, 2019. `arXiv:1909.11830`.

26 Tze Leung Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985. `doi:10.1016/0196-8858(85)90002-8`.

27 Nadjib Lazaar, Youssef Hamadi, Said Jabbour, and Michèle Sebag. Cooperation control in Parallel SAT Solving: a Multi-armed Bandit Approach. Research Report RR-8070, INRIA, 2012.

28 Jia Hui Liang, Chanseok, Vijay Ganesh, Krzysztof Czarnecki, and Pascal Poupart. MapleCOMSPS, MapleCOMSPS_LRB, MapleCOMSPS_CHB. In *Proceedings of SAT Competition 2017: Solver and Benchmark Descriptions*, page 20–21, 2017.

29 Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Exponential Recency Weighted Average Branching Heuristic for SAT Solvers. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 3434–3440. AAAI Press, 2016. URL: `http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12451`.

30 Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Learning Rate Based Branching Heuristic for SAT Solvers. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 123–140. Springer, 2016. `doi:10.1007/978-3-319-40970-2_9`.

31 Jia Hui Liang, Chanseok Oh, Vijay Ganesh, Krzysztof Czarnecki, and Pascal Poupart. MapleCOMSPS, MapleCOMSPS_LRB, MapleCOMSPS_CHB. In *Proceedings of SAT Competition 2016: Solver and Benchmark Descriptions*, page 52–53, 2016.

32 Michael Luby, Alistair Sinclair, and David Zuckerman. Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, pages 128–133, 1993. `doi:10.1109/ISTCS.1993.253477`.

33 Inês Lynce and João Marques-Silva. SAT in Bioinformatics: Making the Case with Haplotype Inference. In Armin Biere and Carla P. Gomes, editors, *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, volume 4121 of *Lecture Notes in Computer Science*, pages 136–141. Springer, 2006. `doi:10.1007/11814948_16`.

34 J.P. Marques-Silva and K.A. Sakallah. GRASP: a search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999. `doi:10.1109/12.769433`.

35 Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, pages 530–535. ACM, 2001. `doi:10.1145/378239.379017`.

36 Anastasia Paparrizou and Hugues Wattez. Perturbing Branching Heuristics in Constraint Solving. In Helmut Simonis, editor, *Principles and Practice of Constraint Programming - 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7-11, 2020, Proceedings*, volume 12333 of *Lecture Notes in Computer Science*, pages 496–513. Springer, 2020. `doi:10.1007/978-3-030-58475-7_29`.

37 Lawrence Ryan. Efficient algorithms for clause-learning SAT solvers. Master's thesis, Simon Fraser University, 2004.

38 Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, Cambridge, MA, USA, 1998. URL: `https://www.worldcat.org/oclc/37293240`.

39 William R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294, December 1933. `doi:10.1093/biomet/25.3-4.285`.

40 Toby Walsh. Search in a Small World. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'99, page 1172–1177, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. URL: `https://www.ijcai.org/Proceedings/99-2/Papers/071.pdf`.

41 Hugues Wattez, Frédéric Koriche, Christophe Lecoutre, Anastasia Paparrizou, and Sébastien Tabary. Learning Variable Ordering Heuristics with Multi-Armed Bandits and Restarts. In Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang, editors, *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 371–378. IOS Press, 2020. doi:10.3233/FAIA200115.

42 Wei Xia and Roland H. C. Yap. Learning Robust Search Strategies Using a Bandit-Based Approach. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 6657–6665. AAAI Press, 2018. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17192.