The Impact of Bounded Variable Elimination on Solving Pigeonhole Formulas

Joseph E. Reeves and Marijn J. H. Heule

Carnegie Mellon University, Pittsburgh, Pennsylvania, United States
{jereeves,mheule}@cs.cmu.edu

Abstract

Variable elimination is arguably the most important pre- and in-processing technique in state-of-the-art SAT solvers. There are hardly any problems for which variable elimination by substitution, as presented by Eén and Biere in 2005, hurts performance. However, during an experimental study, we observed that variable elimination resulted in substantial slowdowns of the 2020 SAT Competition winner Kissat on pigeonhole formulas. In this paper we evaluated the impact of different variable elimination orderings on solving pigeon hole formulas using recent SAT competition winners. The results show that some solvers are more stable than others, while the formulas obtained by some elimination orderings are consistently more difficult to solve. Further, we implemented static variable decision heuristics in the solver CaDiCaL that outperformed all other solvers.

1 Introduction

Variable elimination for conflict-driven clause learning (CDCL) SAT solvers was pioneered in the seminal paper by Niklas Eén and Armin Biere [10]. They improve Davis Putnam resolution [9], also known as variable elimination *by distribution*, by using gate extraction to allow for a more optimized elimination process. The resulting technique, called variable elimination *by substitution*, is typically combined with a bound on how much the formula is allowed to grow. This bounded variable elimination (BVE) can reduce the number of variables and clauses in a formula at any stage in a solver's execution. BVE has been incorporated into all modern CDCL solvers, and is recognized as one of the most important simplification techniques in SAT. It has been shown to be consistently beneficial across benchmarks through experimental validation.

However, we observed that BVE can have a positive or negative impact on solver performance when solving pigeonhole formulas. More specifically, certain variable elimination orderings yield more difficult formulas, while others improve the performance of solvers. We show how some top-tier solvers are stable under the variable elimination orderings, while other solvers suffer a drastic loss in performance. In addition, variable elimination has the potential to help some solvers depending on the ordering, but appears to strictly degrade performance of other solvers. To understand this phenomena, we evaluated several configurations of various solvers. We found that variable scoring heuristics have large impacts on execution time over the variable elimination instances. Further, we implemented static scoring heuristics that show improved performance, and compare them with the variable scoring internal to CaDiCaL.

2 Preliminaries

To evaluate the satisfiability of a formula, a CDCL solver [19] iteratively performs the following operations: First, the solver performs unit propagation. Then, it tests whether it has reached a conflict. If no conflict has been reached and all variables are assigned, the formula is satisfiable.

Otherwise, the solver chooses an unassigned variable through a variable selection heuristic, assigns a truth value to it through a phase selection heuristic, and continues by again performing unit propagation. If, however, a conflict has been reached, the solver performs conflict analysis potentially learning a short clause. In case this clause is the (unsatisfiable) empty clause, the unsatisfiability of the formula can be concluded. In case it is not the empty clause, the solver revokes some of its variable assignments ("backjumping") and then repeats the whole procedure. Modern solvers incorporate pre- and in-processing techniques [15] which change the formula in some way, usually reducing the number of variables and clauses or shrinking the size of clauses.

2.1 Variable Selection

An important part of the CDCL algorithm is selecting which variable to branch on. Most decision heuristics are based on variable scores [7], where each variable is assigned a score and the unassigned variable with highest score is selected for branching. Scores can be static, assigned once and never changed during runtime; dynamic without state, calculated via some property of the formula; or dynamic with state, updated via some learning metric. Of the dynamic scores with state, variable state independent decaying sum (VSIDS) [20], normalized VSIDS (NVSIDS), exponential VSIDS (EVSIDS) [2], learning rate branching (LRB) [17], and variable move-to-front (VMTF) [7] are commonly used in practice. For these scoring heuristics, bumping refers to increasing a variable's score and rescoring refers to decreasing scores.

VSIDS bumps certain variables by some constant increment, and rescores all variables after a set number of conflicts. **NVSIDS** extends this by using an exponential moving average (EMA). At each conflict a variable is either bumped with $s' = f \cdot s + (1 - f)$, or rescored with $s' = f \cdot s$, for a constant $0 < f < 1$. **EVSIDS**, first appearing in MINISAT [11], uses an exponentially increasing score increment to avoid rescoring. With conflict index i and $g = 1/f$ bumped variables are updated with $s' = s + g^i$. **LRB** scores variables based on their learning rate (LR): the fraction of conflicts a variable participates in while assigned. Given an LR of r when a variable is unassigned and step-size α, the variable is bumped with $s' = (1-\alpha) \cdot s + \alpha \cdot r$. Variables are rescored at every conflict with $s' = 0.95 \cdot s$ similar to NVSIDS. LRB uses a step-size that gets smaller over time, deemphasizing newly learned information at the later stages of an execution. **VMTF** simply maintains a list of variables where bumping a variable moves it to the head of the list. Since bumping order matters, the variables can be sorted before bumping to reflect the current state of the trail.

Variables are bumped during conflict analysis. They may then be sorted immediately, or after a set conflict interval. Typically, variables on the conflict side are bumped. It is also beneficial to bump reason side variables [17]. This can be done in a bounded way, limiting the number of variables bumped. These variable scoring heuristics can be used in combination, e.g., CADICAL switches between EVSIDS and VMTF. Additionally, some versions of MAPLE use a distance heuristic [27] for the first $50,000$ conflicts, before switching to LRB and VSIDS.

2.2 Phase Selection

After a decision variable is selected an often independent heuristic selects the phase, i.e., the value assigned to the variable. It is common to initialize phases to one value for all variables. During runtime, solvers use phase-saving [23]: a technique to cache and reuse recent phases. In addition, rephasing can reset cached phases during restarts in multiple ways including flipping values, returning to initial values, and randomizing values. More complicated techniques for rephasing involve using a local search solver to reduce the number of unsatisfied clauses, or

storing the phases with the largest trail as the best phase. Target phasing is an extension of the best phase, where the phases are updated online to promote longer assignment trails [5].

2.3 Bounded Variable Elimination

Using standard notation for CNF formulas, resolution takes two clauses $C_1 = x \vee a_1 \vee ... \vee a_n$ and $C_2 = \overline{x} \vee b_1 \vee ... \vee b_m$, and returns $a_1 \vee ... \vee a_n \vee b_1 \vee ... \vee b_m$, denoted by $C_1 \otimes C_2$. Variable elimination *by distribution* on some variable x is performed by resolving all clauses containing x, denoted by S_x, with all clauses containing its negation $\overline{x}$, denoted by $S_{\bar{x}}$,

$$\{C_1 \otimes C_2 | C_1 \in S_x, C_2 \in S_{\bar{x}}\}$$

The non-tautological clauses in the above set then replace the clauses S_x and $S_{\bar{x}}$ in the formula, eliminating all occurrences of variable x. The new formula is equisatisfiable.

Variable elimination *by substitution* [10] improves the elimination procedure by leveraging gate extraction [22]. A variable x is the output of a gate if x is functionally dependent on its defining clauses. When performing elimination, only pairwise resolvents between definition and non-definition clauses containing x are needed. The pairwise resolvents from clauses in the gate definition are tautologies, and the pairwise resolvents from clauses containing x not in the gate definition are derivable from the other resolvents. This simplified procedure can be viewed as substituting the definition of x into non-defining clauses. It both reduces the number of resolutions required and frequently reduces the number of clauses in the formula.

Bounded variable elimination (BVE) adds a criteria for when variable elimination is to be used, e.g., if the number of clauses added is less than the number of clauses removed. Note that if a literal occurs positively (resp. negatively) in a single clause, variable elimination is guaranteed to produce fewer clauses. In general, variables are selected as candidates for BVE arbitrarily if they meet some criterion. For CaDiCaL, this is all variables occurring in removed irredundant clauses since the last attempted elimination. These variables are then iterated over and eliminated if BVE succeeds. Importantly, different elimination candidate orderings can produce different formulas.

3 Pigeonhole Problem

The pigeonhole problem involves placing $n+1$ pigeons into n holes with one pigeon in each hole, or proving that such a placement is impossible. The colloquial proof is straight forward, yet resolution proofs are exponentially-sized [26], making it an interesting problem to study SAT solver performance. As a consequence, mainstream CDCL solvers that emit resolution proofs witness exponential blow-up in solving time. The solvers SaDiCaL [14], PGBDD [8], and Lingeling [3] have been designed with special reasoning techniques to solve problems that are hard for resolution, including pigeon hole formulas.

We use the direct encoding for the pigeonhole formula, with the variable $p_{x,y}$ representing pigeon x in hole y. The formula consists of at least one (ALO) constraints by pigeon and at most one (AMO) constraints by hole:

$$\begin{aligned} \text{ALO(pigeon=}x\text{)} \quad & (\bigvee_{i=1}^{n} p_{x,i}) \\ \text{AMO(hole=}y\text{)} \quad & (\overline{p}_{i,y} \vee \overline{p}_{j,y}) \quad \text{for } 1 \leq i < j \leq n+1 \end{aligned}$$

These clauses express that each pigeon will be in at-least-one hole and that each hole contains at-most-one pigeon. Formulas with these constraints are referred to as sparse. Exactly-one (EXO) constraints additionally include ALO constraints by hole and AMO constraints by pigeon. These additional constraints are redundant because they do not change the satisfiability of the problem. Formulas with the EXO constraints are referred to as full.

Applying BVE to the sparse formulas reduces the number of variables and clauses. From the original formula, $p_{x,y}$ is eliminated by resolving the single ALO clause for pigeon x with the n AMO clauses for hole y, producing n clauses. Each clause contains the positive literals from pigeon x with one negated literal from hole y. Each subsequent variable eliminated must come from a pigeon independent of the previous variables, otherwise more than n clauses would be created and the elimination would not be bounded. Therefore, there are up to $n+1$ variables that can be eliminated, and they may come from any hole. Eliminating one unique pigeon from each hole will generate clauses of the form described above. However, when multiple variables from the same hole are eliminated, the clauses added include some with only positive literals from the two pigeons sharing the hole.

We will study different formulas obtained by removing $n+1$ variables. We use parameter $h \leq n$, which denotes the number of holes from which variables are eliminated. Variables are removed starting with the first hole and then alternating between each of the first h holes. In particular, we remove the following variables: $p_{i,((i-1)\%h)+1}$ for $1 \leq i \leq n+1$. Note that as h increases there will be more binary clauses deleted from the formula, but the added clauses will have shorter average length.

With full constraints, it is not possible to choose an $h < n$ as this would add more clauses than are deleted during BVE. In fact, only n variables can be deleted and they must be selected from independent pigeons and holes. This corresponds to the formula computed with $h = n$ and $i < n+1$. Due to the symmetry of the original formula, the holes of a variable elimination ordering can be isomorphically mapped to new holes without significant changes to the generated formula. E.g., eliminating all variables from hole 2 generates the same formula as eliminating all variables from hole 5 up to variable naming and clause ordering. However, solvers are sensitive to the clause and variable orderings of the input formula and may have different execution times.

4 Evaluation

We ran our experiments on StarExec [25]. The specs for the compute nodes can be found at `https://starexec.org/starexec/public/about.jsp`. The compute nodes were Intel Xeon E5 cores with 2.4 GHz with 8 GB of memory and a 5 hour timeout. The benchmark formulas, experimental data, and solver configurations are available at `https://github.com/jreeves3/bve-evaluation-Artifact`. We thank the community at StarExec for providing these computational resources.

4.1 Solvers

CaDiCaL [4] is the CDCL solver that won the UNSAT track of the SAT competition in 2018. We use the 2020 version. CaDiCaL switches between strategies [21], with stable and unstable modes. For this work, the important difference is the use of VMTF in unstable (unsat) mode and EVSIDS with target phases [6] in stable mode. Kissat [5] is an optimized C version of CaDiCaL with a similar framework but some changes to data structures. We denote different versions of Kissat by the year they entered the SAT competition. Kissat20 won the Main

track of the 2020 SAT competition. The new 2021 version KISSAT21 adds unique implication point (UIP) shrinking, among other changes. KISSAT21 also bumps variables if on-the-fly-strengthening (OTFS) [13, 12] succeeds. OTFS is attempted during conflict analysis and if it succeeds, no new clause is learned but an existing clause is strengthened and used as the driving clause. KISSAT20 only bumps variables after deducing a learned clause.

The three MAPLE solvers that won the SAT competition Main track in 2019, 2018, and 2017 are MAPLE_LCM_DIST_CHRONOBT_DL_V3 [16], MAPLE_LCM_DIST_CHRONOBT [24] MAPLE_LCM_DIST [28], referred to as MAPLE19, MAPLE18, and MAPLE17, respectively. These solvers rest on the code base of GLUCOSE [1] and MINISAT [11], with the addition of variable selection heuristics switching between LRB and VSIDS. The solvers bump reason side variables when in LRB mode. In the solver namings, LCM is learnt clause minimization [18], DIST is the distance strategy for variable selection within the first $50,000$ conflicts, and CHRONO is chronological backtracking. MAPLE19 incorporates a new scheme for moving duplicate learnt (DL) clauses into higher tiers based on their literal block distance (LBD), preventing them from being deleted during clause reduction. MAPLE19 also has a deterministic LRB-VSIDS switching strategy (every 30 million propagations), whereas the others are time-based.

4.2 Sparse and Full Pigeonhole Formulas

Figure 1 shows solvers' execution times on the sparse (left) and full (right) pigeonhole formulas. Note these are log plots due to the exponential blowup in execution time. CADICAL outperforms other solvers for the sparse formulas solving $n = 13$ in under $1,800$ seconds. This is unexpected as KISSAT20 is the optimized C version of CADICAL, suggesting something "lucky" happening during the execution for CADICAL. But, both CADICAL and KISSAT20 outperform the three MAPLE solvers. These trends change for the full formulas. CADICAL performs worse than KISSAT20 and the MAPLE solvers outperform KISSAT20 and CADICAL at $n = 11$. This reflects a drop in performance in KISSAT20 and CADICAL, whereas the other solvers are stable under the additional constraints in the full formulas. The similarity between MAPLE18 and MAPLE17 for this and following plots is because chronological backtracking is not relevant when the formula and number of decisions before each conflict are small.

BVE is disabled for each solver in Figure 2, with execution times shown for the same formulas. Notably, almost all solvers solve more instances, with the exception of KISSAT21. In addition, for the sparse formulas the solvers group more closely together, around where CADICAL was with BVE enabled in Figure 1. The same is true for the full formulas where disabling BVE is helpful across the board. In all four plots KISSAT21 is stable but worse than the best-case performance of the other solvers. It is clear that BVE is impacting solvers in different ways for these formulas.

4.3 Pigeonhole Formulas with Bounded Variable Elimination

To understand the changes from Figure 1 to Figure 2, we looked at the specific ways BVE could be applied to the sparse formulas, i.e., what variables are selected for elimination. We generated formulas for $n = 11$ using the method described in Section 3. The 11 formulas are defined by h, where $n + 1$ variables are selected from h holes. Each variable comes from an independent pigeon to maintain the invariant that $n + 1$ clauses are deleted and n are added. Due to the additional constraints, full formulas must follow the $h = n$ elimination ordering and can only eliminate n variables.

Figure 3 shows the execution time for solvers on these 11 formulas. On the left are the execution times from Figure 1 (def.) and Figure 2 (no-e) with the sparse formula for comparison.

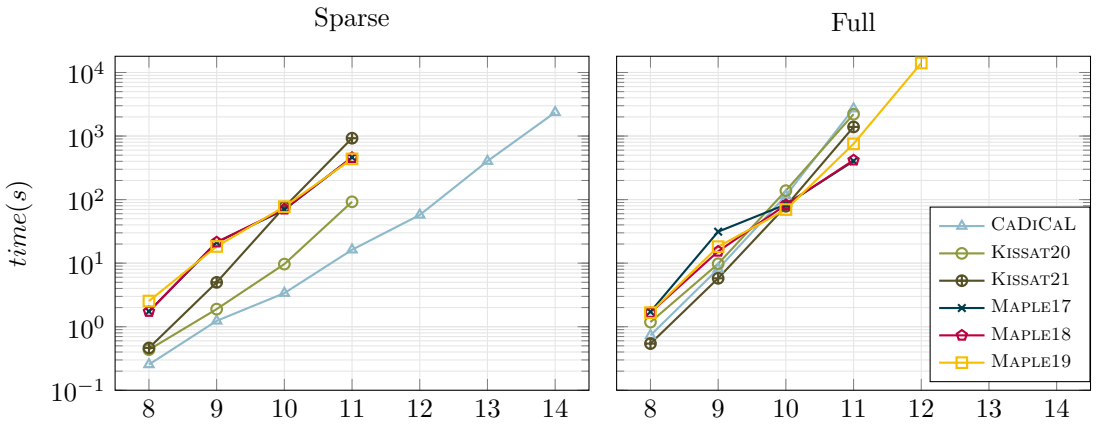

Figure 1: Log plots showing execution time on pigeonhole formulas.

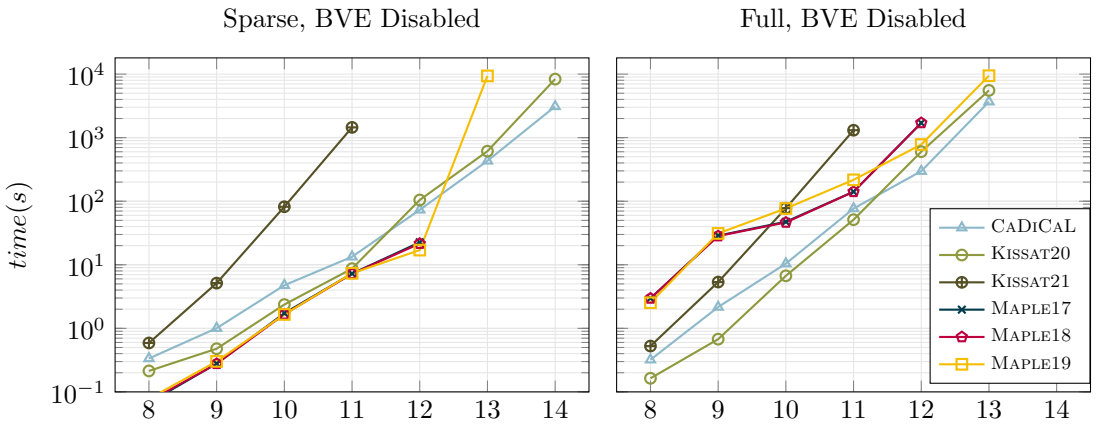

Figure 2: Log plots showing execution time on pigeonhole formulas with BVE disabled.

Kissat20 and CaDiCaL are extremely fast up to $h = 6$, then performance rapidly falls. For Kissat20 and CaDiCaL the variables selected for BVE determine whether the runtime will be less than 1 second or greater than $3,000$ seconds. This makes clear the results from Figure 1: CaDiCaL instantiates a BVE ordering that resulted in a fast execution time, whereas Kissat20 used one that resulted in a slow execution time. Both solvers were forced to use a bad BVE ordering ($h = 11$) for full formulas in Figure 3 which is why they both timed out at $n = 12$.

On the other hand, the Maple solvers are more stable, with a bump from $h = 4$ to $h = 6$, then a plateau. Maple19 follows the same trend but with slightly worse execution times for $h > 7$. The apparent tradeoff for stability is slower execution times for $h < 7$. However, disabling BVE gives the Maple solvers better performance than any BVE instance. Kissat21 is similarly stable, with worse performance. But, Kissat21 does benefit from BVE.

For all solvers, the best-case BVE elimination ordering is with $h = 1$. Figure 4 shows solvers' performances on the pigeonhole formulas after BVE has been applied with $h = 1$. CaDiCaL and Kissat20 perform better with these formulas than if no BVE was applied (Figure 3).

The MAPLE solvers perform worse up until $n = 13$, where MAPLE18 and MAPLE17 no longer timeout. So, not only is the variable ordering for BVE important on these formulas, so too is the decision to perform BVE at all. Additionally, all solvers are faster with $h < 6$, suggesting some formulas have became generally harder after BVE was applied with $h > 6$.

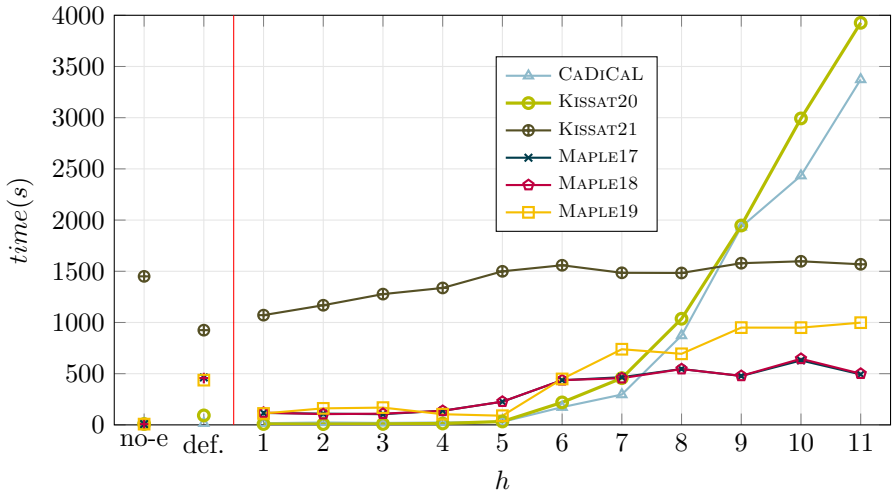

Figure 3: Execution time on BVE instances of pigeonhole formula $n = 11$. In each instance 12 variables are eliminated, selected from independent pigeons and h independent holes.

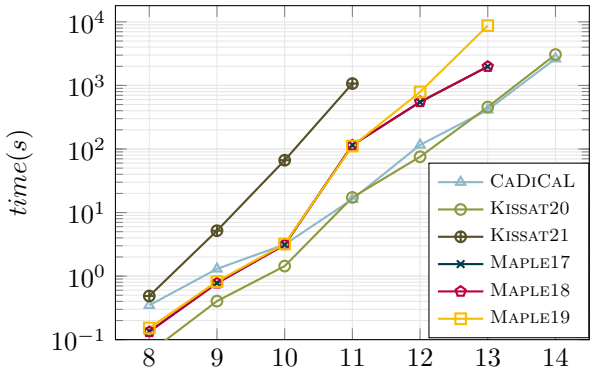

Figure 4: Log plot showing execution time on pigeonhole formulas with BVE applied for the $h = 1$ instance.

5 Branching Heuristic Analysis

Two things are clear from Figure 3: it is possible to achieve short execution times for at least some of the BVE instances (see KISSAT20 on $h = 1$), and it is possible to achieve relatively stable execution times over all instances (see MAPLE17). Understanding the reasons for this could be useful for making a solver both stable and fast over BVE instances.

There are many significant differences between the solvers, but running KISSAT20 in unsat mode hints at the likely culprit: branching heuristics. Among the statistics KISSAT20 displays during a normal execution is the average level (number of decisions made at which a conflict is found). Surprisingly, the level stays around 11 when running KISSAT20 in unsat mode on these formulas. With some instances taking thousands of seconds to solve, this means KISSAT20 is constantly changing the decision variables. Note, many formulas in practice are solved instantly once the level drops to 11. The difference in KISSAT20 and MAPLE solvers' scoring heuristics could explain this phenomenon.

To analyze the effect of the solvers' branching heuristics on the BVE instances, we evaluated the solvers under various configurations. The results show that the degree to which a decision heuristic is static greatly affects solving time. By static we mean that variables are selected in the same or close to the same order throughout the execution.

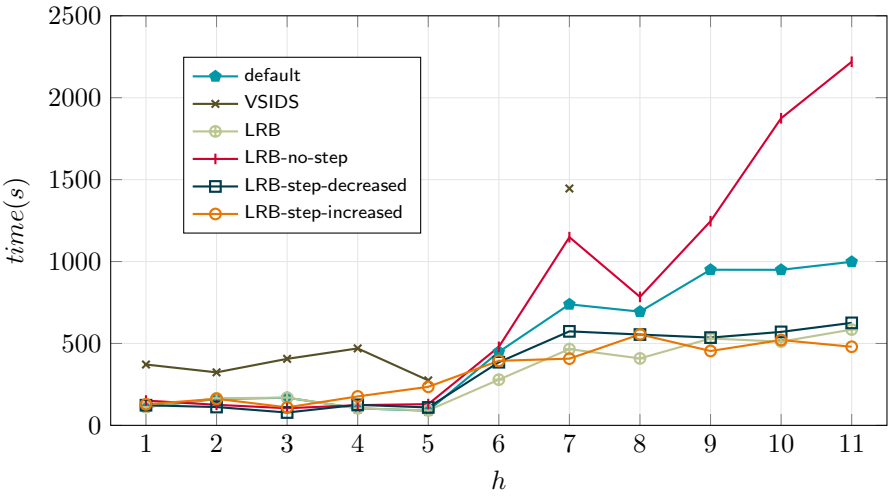

Figure 5: Execution time for MAPLE19 on BVE instances of pigeonhole formula $n = 11$. VSIDS and LRB are the decision heuristics (default switches between the two). The step refers to the step-size decrement. It is set to 0, decreased by 5 times or increased by 5 times.

5.1 The Impact of Branching Heuristics in Maple19

MAPLE19 starts with the Dist heuristic for the first $50,000$ conflicts. This scores variables based on their distance from the conflict clause. The idea is this heuristic gathers more information at the early stages of solving when little has been learned. It then uses LRB, and switches to VSIDS at 30 million propagations, unlike MAPLE18 and MAPLE17 which switch after $2,500$ seconds. Reason side bumping is only done during the LRB phase.

We set either VSIDS or LRB always on for two of the configurations in Figure 5. The VSIDS experiments timed out for $h = 6$ and $h > 8$. The LRB experiments are similar to the results of MAPLE18 and MAPLE17 in Figure 3. MAPLE18 and MAPLE17 would never switch to VSIDS if solving a problem within $2,500$ seconds. A significant difference with VSIDS is the lack of reason side bumping. A lack of reason side bumping deteriorates CADICAL's performance in a similar way.

We also configured the step-size decrement with multiple values. Recall, variables are bumped with $s' = (1 - \alpha) \cdot s + \alpha \cdot r$. The step-size α is initialized to 0.4 and decremented by 10^{-6} at each conflict until it reaches 0.06. The step-size determines the weight of newly learned information, so as it decreases the LRB relies more on past information. As it decrements, the decision ordering will become more static. This is unlike Kissat20 where newly learned information will be weighted the same over the course of an execution, and the decision ordering may be constantly changing. To evaluate the effect of step-size for these formulas, we configured the step-size decrement to be either 0 (no-step), five times more than the original value (step-increased), or five times less (step-decreased). No-step performs terribly, and step-reduced is slightly worse than the other LRB configurations for $h > 6$. This might be because the harder instances with $h > 6$ cause variable scores to fluctuate sporadically over the solving time. Having a larger step decrement forces the scores to be more static, reducing the degree of this fluctuation. In other words, for these formulas it could be beneficial to learn some decision ordering early, then stick with it for the remainder of the execution.

5.2 The Impact of Branching Heuristics in Kissat21

Kissat21 switches between EVSIDS and VMTF, with a sophisticated phase selection and restart strategy that differs between the scoring heuristics. There is no decay for the exponential score increment, i.e., variables are bumped with $s' = s + g^i$. In reference to LRB this can be seen as having a step-size of 0. This means Kissat21 does not decrement the weight of newly learned information over the course of solving. Kissat21 extends Kissat20 with UIP shrinking and bumping after OTFS.

In Figure 6 we see various configurations of Kissat21. As with Maple19, disabling reason-side bumping negatively impacts performance. Disabling UIP shrinking (no-shrink) makes this even worse. Separately, disabling bumping after OTFS (no OTFS bumping, no-otfsb) does not

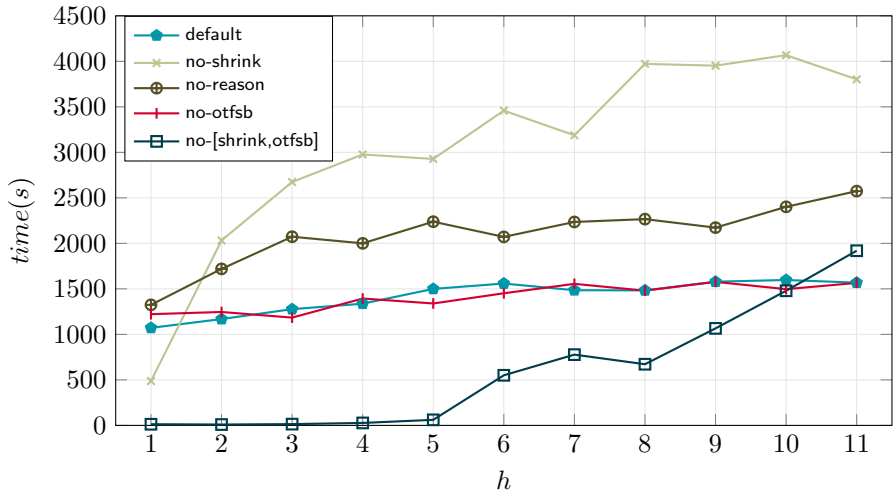

Figure 6: Execution time for Kissat21 on BVE instances of pigeonhole formula $n = 11$. Configurations disabled are OTFS bumping (otfsb), reason side bumping (reason), and UIP shrinking (shrink).

affect performance. Only when UIP shrinking and bumping after OTFS are disabled do we see performance that resembles KISSAT20. These techniques interact in complex ways that affect how and when variables are bumped. Understanding the interplay between conflict analysis techniques and bumping is an open area of research.

5.3 The Impact of Branching Heuristics in CaDiCaL

CADICAL has a default configuration similar to KISSAT21 without UIP shrinking or OTFS bumping. For CADICAL we took a deeper look at the components that affect branching, namely: reason side bumping, forced phasing, scoring strategies VMTF versus EVSIDS, and unstable (unsat) and stable mode. We also introduce static variable orderings. For a decision, the ordering is traversed until the first unassigned variable is found, and that variable becomes the next decision variable. Phasing is done independently. We considered three orderings: *static-direct* which counts the variables by pigeon then by hole, *static-by-hole* which counts the variables by hole then by pigeon, and *static-from-corner* which counts the variables by expanding from the corner $p_{1,1}$ in a pigeon then hole-wise manner. We also experiment with reversing the first two orderings, denoted by a *-rev* postfix. Reversing the static orderings would have no impact on the original formula since it is symmetric. However, the elimination orderings we compute start from $h = 1$. So, reversing the static orderings can affect whether decision variables come from holes with eliminated variables.

Figure 7 shows the results for various scoring heuristics and the static orderings. Firstly, the scoring heuristics VMTF or EVSIDS and stable or unsat mode do not seem to alter performance in a meaningful way. The stable mode performs slightly worse for $h = 10, 11$ and the unsat mode is the opposite. However, the static orderings have vastly different results. Both *static-by-hole* and *static-from-corner* do much worse than the default solver for small h, while *static-by-hole* begins coming down after $h = 7$. Reversing the ordering yields similar performance to the default solver's trend-line with *static-by-hole-rev*, but the performance degradation is not as steep. It is not clear why *static-by-hole* and *static-by-hole-rev* cross at $h = 9$. On the other hand, *static-direct* and *static-direct-rev* are stable and better than any of the solvers in Figure 3 for $h > 7$. There appears to be a tradeoff between stable performance over all instances, or better performance on $h < 7$ with a steep drop off for larger values.

Forcing variable phases (default to true), disabling target phasing, and disabling reason side variable bumping can also affect the degree to which the variable ordering is static. The results in Figure 8 show that disabling reason side bumping is extremely detrimental, with timeouts occurring at $h = 2$. Interestingly, the default configuration with no reason side bumping solves $h = 10$ within $2,000$ seconds. Disabling reason bumping has less of an impact with EVSIDS. This could be because VMTF bumps variables to the head of a list whereas EVSIDS maintains scores over time lowering the impact of poorly timed bumps. Forced phasing has the opposite effect, making the configurations improve slightly except in the case of *static-direct*. Disabling target phases has little effect.

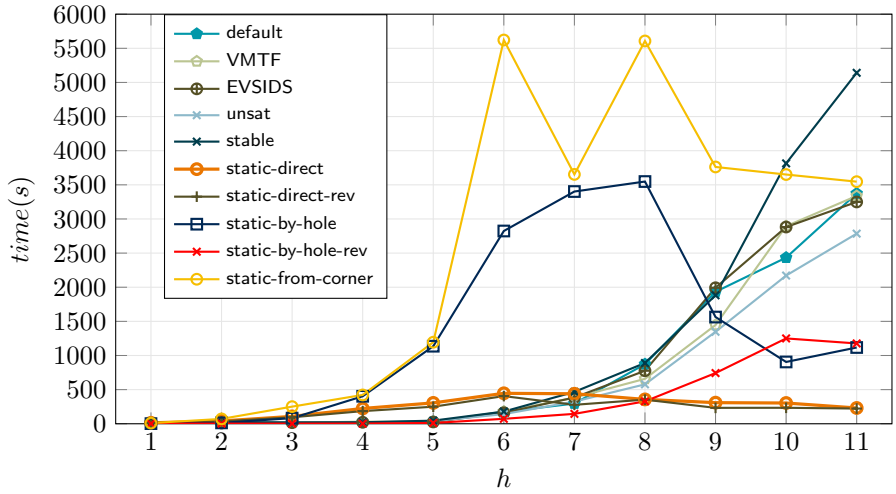

Figure 7: Execution time for CaDiCaL on BVE instances of pigeonhole formula $n = 11$. VMTF and EVSIDS affect the scoring heuristic while stable and unsat also affect restart and rephase strategies. Static- refers to a static ordering option, and rev is reversing the ordering.

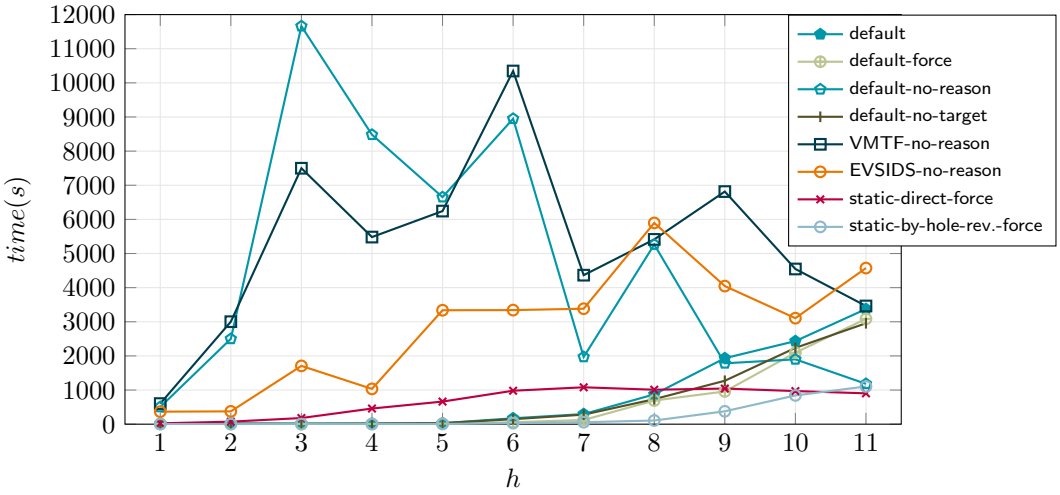

Figure 8: Execution time for CaDiCaL on BVE instances of pigeonhole formula 11. Force forces initial phases (set to true), no-reason disables reason side bumping, and no-target disables target phasing.

6 Variable Score Analysis

Based on the data, decision heuristics and bumping choices seem to be a leading cause in the performance variations on BVE instances. More static approaches, e.g., higher step-decrement for Maple19 or *static-by-hole-rev* for CaDiCaL, perform better on these instances. But,

it is not clear why *static-by-hole-rev* tracks CaDiCaL's default performance, or why disabling reason bumping hurts performance. To understand this, we take a deeper look at the branching decisions being made by CaDiCaL and Kissat21.

To aggregate branching decisions without disentangling the trail and implied variables we focused solely on variable scores. After each conflict we stored the current variable scores and sorted the variables in reverse order. For each variable we computed the average position in the sorted scores list every $100,000$ conflicts. The variables were plotted in Figure 9 by this average position, where colorings indicate the hole a variable refers too. Each consecutive column represents averages from the next $100,000$ conflicts. Solvers are run with option *stabilizeonly=1* forcing EVSIDS as VMTF does not keep scores.

Recall in Figure 3 the execution time of CaDiCaL begins to increase sharply after $h = 6$. In Figure 9 for the smaller h values, variables appear to be grouped by hole, but this trend breaks down by $h = 11$ where the colors appear random. The grouping of holes matches the holes in which variables are eliminated from. Take $h = 2$, the variables from $h = 1, 2$ are at the top of the ordering. The same is true for $h = 6$, the variables from $h = 1, 2, 3, 4, 5, 6$ (non-red variables), are at the top of the ordering. This trend is shown by the red line cutting across the graph, with variables from eliminated holes resting above the line. This image prompted the implementation of the reversed by hole static ordering in Figure 7, which in turn followed the same execution pattern as the default CaDiCaL configuration up to $h = 7$.

Figure 10 uses the same method to compute averages. The three sections are: Kissat21 for $h = 1, 5, 11$; Kissat21 for $h = 3$ with default and K1 as the no-shrink no-otfsb configuration; and CaDiCaL for $h = 3$ with default, C1 as forced phasing, and C2 as no reason bumping configurations. The first section shows that Kissat21 does not bump variables by hole for smaller h, and instead produces a mixture of colors for all h-values. However, K1 does do the by-hole sorting, and this was the configuration with the best performance. On the other hand, performance for CaDiCaL in C2 is worse than the CaDiCaL default configuration. Notice the variables start around 30, and these higher averages mean the variables move around more in the ordering. Even though holes 1,2, and 3 are at the top of the ordering, the other variables are less stable in their positions and the performance degrades. These figures show us that both the way in which variables are ordered and the degree to which the ordering is static affect performance.

7 Conclusion and Future Work

While the importance of BVE in modern SAT solvers is widely acknowledged, the impact of different BVE orderings is hardly studied. All solvers do BVE essentially the same way, yet we observed different BVE orderings can affect them in different ways. This was heavily tied to the decision heuristics used by each solver.

These problems and possible solutions should be explored in future work. On the one hand, certain BVE instances appear harder than others, and in some cases BVE consistently degrades performance. There may be certain cases, for example when a literal occurs in only a single clause for one of its parities, that are not suited for BVE. On the other hand, decision heuristics for solvers should be made robust under different BVE instances. We saw how reason side bumping and OTFS bumping could affect performance. And also how decrementing the step-size in LRB could lead to stable performance. Understanding when to stop learning from new conflicts and how to dynamically adjust scoring heuristics could greatly benefit solvers.

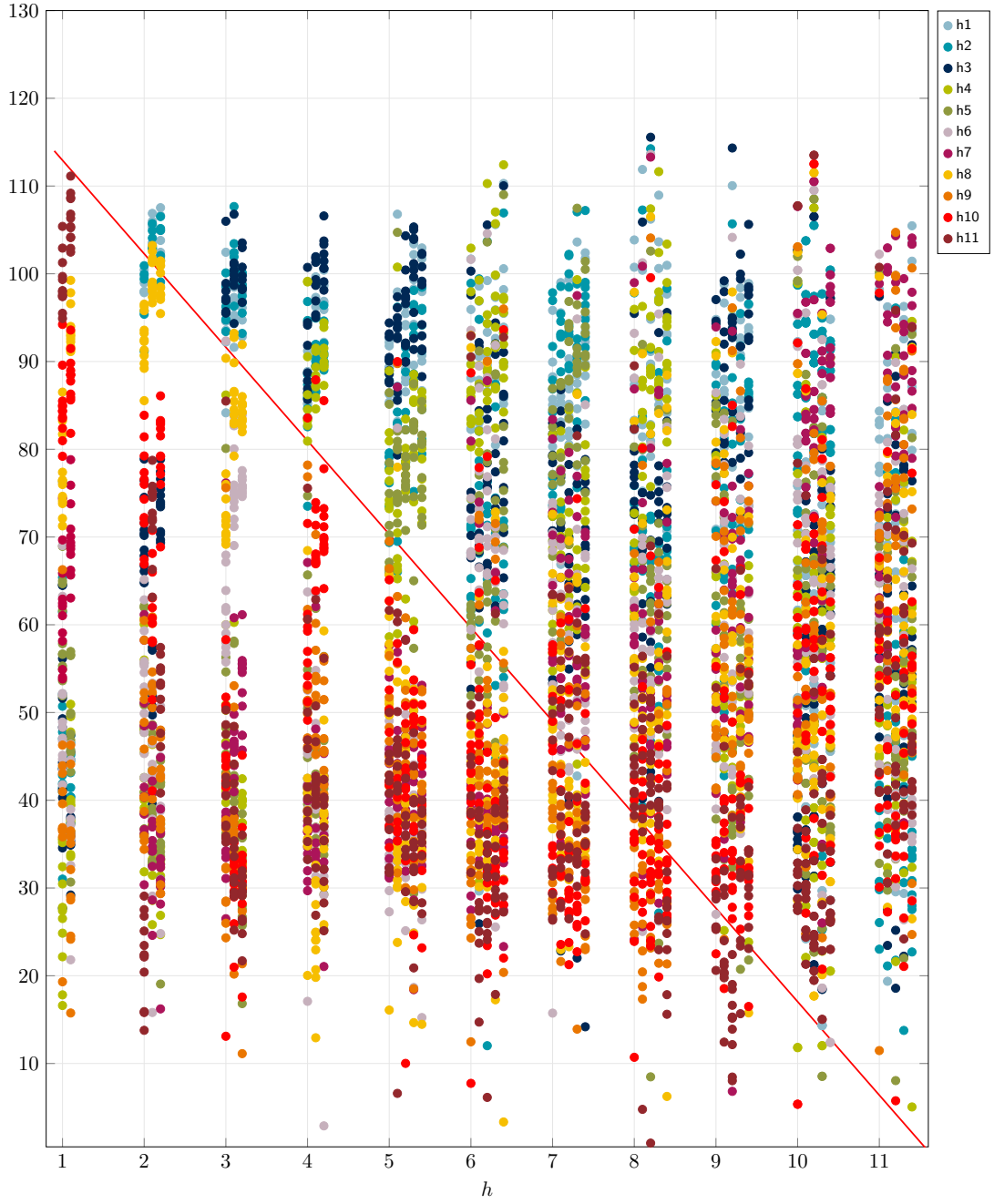

Figure 9: Variable scores are sorted at each conflict and the positions are averaged every $100,000$ conflicts, up to $500,000$ conflicts (some instances are solved before this limit). Variables are colored by hole.

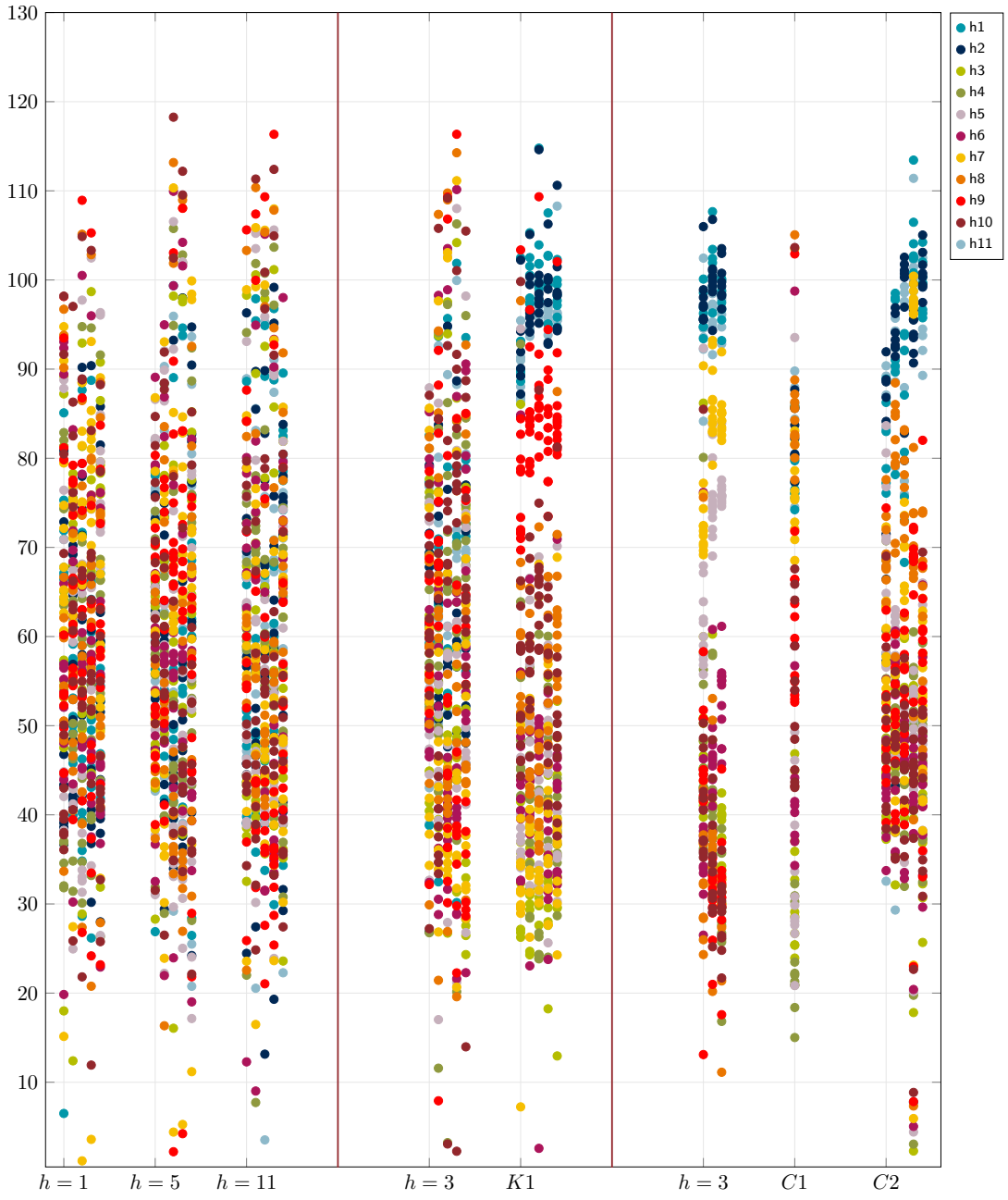

Figure 10: KISSAT21 results for BVE instance on pigeonhole formula $n = 11$ with $h = 1, 5, 11, 3$. K1 is the configuration with no shrink and no OTFS bumping. The third section has CADICAL results for $h = 3$ default, forced phasing, and no reason bumping. Variable scores are sorted at each conflict and the positions are averaged every $100,000$ conflicts, up to $500,000$ conflicts Variables are colored by hole.

References

[1] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 399–404, 2009.

[2] Armin Biere. Adaptive restart strategies for conflict driven sat solvers. In *Theory and Applications of Satisfiability Testing (SAT)*, pages 28–33, 05 2008.

[3] Armin Biere. Lingeling, Plingeling, PicoSAT, and PrecoSAT at SAT race 2010. Technical report, 2010.

[4] Armin Biere. CaDiCaL, Lingeling, Plingeling, Treengeling, and YalSAT entering the SAT competition 2017. Technical report, 2017.

[5] Armin Biere, Katalin Fazekas, M. Fleury, and Maximilian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT competition 2020. Technical report, 2020.

[6] Armin Biere and Mathias Fleury. Chasing target phases. In *Proceedings of Pragmatics of (SAT)*, 2020.

[7] Armin Biere and Andreas Fröhlich. Evaluating cdcl variable scoring schemes. In *Theory and Applications of Satisfiability Testing (SAT)*, 2015.

[8] Randal E. Bryant and Marijn J. H. Heule. Generating extended resolution proofs with a BDD-based SAT solver. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Part I*, volume 12651 of *LNCS*, pages 76–93, 2021.

[9] M. Davis and H. Putnam. A computing procedure for quantification theory. *ACM*, 7(3):394–397, July 1962.

[10] Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In *Theory and Applications of Satisfiability Testing (SAT)*, volume 3569 of *LNCS*, pages 61–75. Springer, 2005.

[11] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Theory and Applications of Satisfiability Testing (SAT)*, volume 2919 of *LNCS*, pages 502–518. Springer, 2003.

[12] Youssef Hamadi, Said Jabbour, and Lakhdar Sais. Learning for dynamic subsumption. In *IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 328 – 335, 12 2009.

[13] Hyojung Han and Fabio Somenzi. On-the-fly clause improvement. In *Theory and Applications of Satisfiability Testing (SAT)*, volume 5584, pages 209–222. Springer, 06 2009.

[14] Marijn J. H. Heule, Benjamin Kiesl, and Armin Biere. Encoding redundancy for satisfaction-driven clause learning. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 11427 of *LNCS*, pages 41–58. Springer, 2019.

[15] Matti Järvisalo, Marijn Heule, and Armin Biere. Inprocessing rules. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *IJCAR*, volume 7364 of *LNCS*, pages 355–370. Springer, 2012.

[16] Stepan Kochemazov, Oleg Zaikin, Victor Kondratiev, and Alexander Semenov. Maplelcmdistchronobt-dl, duplicate learntsheuristic-aided solvers at the sat race 2019. Technical report, 2019.

[17] Jia Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Learning rate based branching heuristic for sat solvers. In *Theory and Applications of Satisfiability Testing (SAT)*, volume 9710, pages 123–140. Springer, 07 2016.

[18] Mao Luo, Chu-Min Li, Fan Xiao, Felip Manyà, and Zhipeng Lü. An effective learnt clause minimization approach for cdcl sat solvers. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 703–711, 2017.

[19] Joao P. Marques-Silva, Ines Lynce, and Sharad Malik. *Conflict-Driven Clause Learning SAT Solvers*, chapter 4, pages 131–153. Handbook of Satisfiability, IOS Press, February 2009.

[20] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: engineering an efficient sat solver. In *Design Automation Conference*, pages 530–535, 2001.

[21] Chanseok Oh. Between sat and unsat: The fundamental difference in cdcl sat. In *Theory and*

Applications of Satisfiability Testing (SAT), pages 307–323. Springer International Publishing, 2015.

[22] Richard Ostrowski, Eric Gregoire, Bertrand Mazure, and Lakhdar Sais. Recovering and exploiting structural knowledge from cnf formulas. In *Principles and Practice of Constraint Programming*, volume 2470, pages 185–199. Springer-Verlag, 09 2002.

[23] Thammanit Pipatsrisawat and Adnan Darwiche. A lightweight component caching scheme for satisfiability solvers. In *Theory and Applications of Satisfiability Testing (SAT)*, volume 4501, pages 294–299. Springer, 05 2007.

[24] Vadim Ryvchin and Alexander Nadel. Maple-LCM-Dist-ChronoBT: Featuring chronological backtracking. Technical report, 2018.

[25] Aaron Stump, Geoff Sutcliffe, and Cesare Tinelli. StarExec: A cross-community infrastructure for logic solving. In *International Joint Conference on Automated Reasoning (IJCAR)*, volume 8562 of *LNCS*, pages 367–373. Springer, 2014.

[26] Alasdair Urquhart. Hard examples for resolution. *J.ACM*, 34(1):209–219, 1987.

[27] Fan Xiao, Chu-Min Li, Mao Luo, Felip Manyà, Zhipeng Lü, and Yu Li. A branching heuristic for SAT solvers based on complete implication graphs. *Science China Information Sciences*, 62(7), apr 2019.

[28] Fan Xiao, Mao Luo, Chu-Min Li, Felip Manyaà, and Zhipeng Lü. MapleLRB-LCM, maple-lcm, maple-lcm_dist, maplelrb-lcmoccrestart and glucose-3.0+width in sat competition 2017. Technical report, 2017.