Machine Learning-based Restart Policy for CDCL SAT Solvers

Jia Hui Liang[1], Chanseok Oh[2], Minu Mathew[3],
Ciza Thomas[3], Chunxiao Li[1], and Vijay Ganesh[1]

[1] University of Waterloo, Waterloo, Canada
[2] Google, New York, United States
[3] College Of Engineering, Thiruvananthapuram, India

Abstract. Restarts are a critically important heuristic in most modern conflict-driven clause-learning (CDCL) SAT solvers. The precise reason as to why and how restarts enable CDCL solvers to scale efficiently remains obscure. In this paper we address this question, and provide some answers that enabled us to design a new effective machine learning-based restart policy. Specifically, we provide evidence that restarts improve the quality of learnt clauses as measured by one of best known clause quality metrics, namely, literal block distance (LBD). More precisely, we show that more frequent restarts decrease the LBD of learnt clauses, which in turn improves solver performance. We also note that too many restarts can be harmful because of the computational overhead of rebuilding the search tree from scratch too frequently. With this trade-off in mind, between that of learning better clauses vs. the computational overhead of rebuilding the search tree, we introduce a new machine learning-based restart policy that predicts the quality of the next learnt clause based on the history of previously learnt clauses. The restart policy erases the solver's search tree during its run, if it predicts that the quality of the next learnt clause is below some dynamic threshold that is determined by the solver's history on the given input. Our machine learning-based restart policy is based on two observations gleaned from our study of LBDs of learnt clauses. First, we discover that high LBD percentiles can be approximated with z-scores of the normal distribution. Second, we find that LBDs, viewed as a sequence, are correlated and hence the LBDs of past learnt clauses can be used to predict the LBD of future ones. With these observations in place, and techniques to exploit them, our new restart policy is shown to be effective over a large benchmark from the SAT Competition 2014 to 2017.

1 Introduction

The Boolean satisfiability problem is a fundamental problem in computer science: given a Boolean formula in conjunctive normal form, does there exist an assignment to the Boolean variables such that the formula evaluates to true? Boolean satisfiability is the quintessential NP-complete [13] problem, and hence one might prematurely conjecture that Boolean SAT solvers cannot scale. Yet modern SAT

solvers routinely solve instances with millions of Boolean variables. In practice, many practitioners reduce a variety of NP problems to the Boolean satisfiability problem, and simply call a modern SAT solver as a black box to efficiently find a solution to their problem instance [11, 12, 19]. For precisely this reason, SAT solving has become an important tool for many industrial applications. Through decades of research, the SAT community has built surprisingly effective backtracking solvers called conflict-driven clause-learning (CDCL) [23] SAT solvers that are based on just a handful of key principles [18]: conflict-driven branching, efficient propagation, conflict analysis, preprocessing/inprocessing, and restarts.

Like all backtracking search, the run of a CDCL SAT solver can be visualized as a search tree where each distinct variable is a node with two outgoing edges marked true and false (denoting value assignments to the variable) respectively. The solver frequently restarts, that is, it discards the current search tree and begins anew (but does not throw away the learnt clauses and the variable activities). Although this may seem counterproductive, SAT solvers that restart frequently are significantly faster empirically than solvers that opt not to restart. The connection between restarts and performance is not entirely clear, although researchers have proposed a variety of hypotheses such as exploiting variance in the runtime distribution [22, 14] (similar to certain kinds of randomized algorithms). For various reasons however, we find these hypotheses do not explain the power of restarts in the CDCL SAT solver setting. In this paper, we take inspiration from Hamadi et al. who claim that the purpose of restarts is to compact the assignment stack [16]. We then further show that a compact stack tends to improve the quality of clauses learnt where we define quality in terms of the well-known metric *literal block distance* (LBD). Despite the search tree being discarded by a restart, learnt clauses are preserved so learning higher quality clauses continues to reap benefits across restarts. By learning higher quality clauses, the solver tends to find a solution quicker. However, restarting too often incurs a high overhead of constantly rebuilding the search tree. So it is imperative to balance the restart frequency to improve the LBD but avoid excessive overhead.

Based on the above-mentioned analysis, we designed a new restart policy called *machine learning-based restart* (MLR) that triggers a restart when the LBD of the next learnt clause is above a certain threshold. The motivation for this policy is that rather than investing computation into learning a low quality clause, the solver should invest that time in rebuilding the search tree instead in the hopes of learning a better clause. This restart policy is based on two key observations that we made by analyzing CDCL solvers over a large benchmark: First, we observed that recent LBDs are correlated with the next LBD. We introduce a machine learning-based technique exploiting this observation to predict the LBD of the next learnt clause. Second, we observed that the right tail of the LBD distribution is similar to the right tail of the normal distribution. We exploit this observation to set a meaningful LBD threshold for MLR based on percentiles. MLR is then shown to be competitive vis-a-vis the current state-of-the-art restart policy implemented as part of the Glucose solver [4].

Contributions: We make the following contributions in this paper:

1. We provide experimental support for the hypothesis that restarts "compact the assignment stack" as stated by the authors of ManySAT [16] (see Section 4). We then add to this hypothesis, and go on to show that a compact assignment stack correlates with learning lower LBD clauses (see Section 4.2). Lastly, learning clauses with lower LBD is shown to correlate with better solving time (see Section 4.3). Additionally we provide analytical explanations as to why we discount some previously proposed hypotheses that attempt to explain the power of restarts in practice (see Section 3).
2. We propose a method to set thresholds for the quality of a LBD of a clause. We experimentally show that the right tail of the LBD distribution closely matches a normal distribution, hence high percentiles can be accurately predicted by simply computing the mean and standard deviation. See Section 5.1 for details.
3. We show that LBDs viewed as a sequence are correlated. This is a crucial observation that we back by experimental data. The fact that LBDs viewed as a sequence are correlated enables us to take the LBDs of recent learnt clauses and predict the LBD of the next clause. See Section 5.2 for details.
4. Building on all the above-mentioned experimentally-verified observations, we introduce a new restart policy called *machine learning-based restart* (MLR) that is competitive vis-a-vis the current state-of-the-art restart policy on a comprehensive benchmark from the SAT Competition 2014 to 2017 instances. See Section 6 for details.

2 Background

We assume the reader is familiar with the Boolean satisfiability problem and SAT solver research literature [6].

LBD Clause Quality Metric: It has been shown, through the lens of proof complexity, that clause-learning SAT solvers (under perfect non-deterministic branching and restarts, and asserting clause learning schemes) are exponentially more powerful than CDCL SAT solvers without clause learning [26]. However, the memory requirement to store all the learnt clauses is too high for many instances since the number of conflicts grows very rapidly. To overcome this issue, all modern SAT solvers routinely delete some clauses to manage the memory usage. The most popular metric to measure the quality of a clause is called *literal block distance* (LBD) [3], defined as the number of distinct decision levels of the variables in the clause. Intuitively, a clause with low LBD prunes more search space than a clause with higher LBD. Hence clauses with high LBD typically are the ones prioritized for deletion. Although LBD was originally proposed for the purpose of clause deletion, it has since proven useful in other contexts where there is need to measure the quality of learnt clauses such as sharing clauses in parallel SAT solvers and restarts. Another measure of quality of a learnt clause is its length. To the best of our knowledge, we are not aware of any other universally accepted clause quality metrics at the time of writing of this paper.

In this paper we will often look at LBDs as a sequence. At any time during the search where i conflicts have occurred, we use the term "previous" LBD to refer to the LBD of the clause learnt at the i^{th} conflict and "next" LBD to refer to the LBD of the clause learnt at the $(i+1)^{th}$ conflict.

Overview of Restarts in CDCL SAT Solvers: Informally, a restart heuristic in the context of CDCL SAT solver can be defined as a method that discards parts of the solver state at certain points in time during its run. CDCL solvers restart by discarding their "current" partial assignment and starting the search over, but all other aspects of solver state (namely, the learnt clauses, variable activities, and variable phases) are preserved. Although restarts may appear unintuitive, the fact that learnt clauses are preserved means that solver continues to make progress. Restarts are implemented in practically all modern CDCL solvers because it is well known that frequent restarts greatly improve solver performance in practice.

3 Prior Hypotheses on "The Power of Restarts"

In this section, we discuss prior hypotheses on the power of restarts in the DPLL and local search setting and their connection to restarts in the CDCL setting.

Heavy-tailed Distribution and Las Vegas Algorithm Hypotheses: From the perspective of Las Vegas algorithms, some researchers have proposed that restarts in CDCL SAT solvers take advantage of the variance in solving time [22, 14]. For a given input, the running time of a Las Vegas algorithm is characterized by a probability distribution, that is, depending on random chance the algorithm can terminate quickly or slowly relatively speaking. A solver can get unlucky and have an uncharacteristically long running time, in which case, a restart gives the solver a second chance of getting a short runtime [22]. More specifically, a heavy-tailed distribution was observed for various satisfiable instances on randomized DPLL solvers [14]. However, this explanation does not lift to restarts in modern CDCL solvers. First, most modern CDCL solvers are not Las Vegas algorithms, that is, they are deterministic algorithms, and hence restarts cannot take advantage of variance in the solving time like in Las Vegas algorithms. Second, the optimal restart policy for Las Vegas algorithms has a restart interval greater than the expected solving time of the input [22], so hard instances should restart very infrequently. However in practice, even difficult instances with high solving time benefit from very frequent restarts in CDCL solvers. Third, the definition of restarts in the context of Las Vegas algorithms differs significantly from the restarts implemented in CDCL solvers. In Las Vegas algorithms, the restarts are equivalent to starting a new process, that is, the algorithm starts an independent run from scratch. Restarts in CDCL are only partial, the assignment stack is erased but everything else preserved (i.e., learnt clauses, saved phases, activity, etc.). Since the phases are saved, the CDCL SAT solver reassigns variables to the same value across restart boundaries [27]. As the authors of ManySAT [16] note: "Contrary to the common belief, restarts are not used to eliminate the heavy tailed phenomena since after restarting SAT solvers dive in the part of

the search space that they just left." Fourth, the heavy-tailed phenomena was found to be true only for satisfiable instances, and yet empirically restarts are known to be even more relevant for unsatisfiable instances.

Escaping Local Minima Hypothesis: Another explanation for restarts comes from the context of optimization. Many optimization algorithms (in particular, local search algorithms), get stuck in the local minima. Since local search only makes small moves at a time, it is unlikely to move out of a deep local minimum. The explanation is that restarts allow the optimization algorithm to escape the local minimum by randomly moving to another spot in the solution space. Certain local-search based SAT solvers (that aim to minimize the number of unsatisfied clauses) do use restarts for this very purpose [17, 28]. However, restarts in CDCL do not behave in the same manner. Instead of setting the assignment of variables to random values like in local search, rather CDCL solvers revisit the same (or nearby) search space of assignments even after restarts since the variable activities and phases are preserved across restart boundaries [27].

As we show in Section 4, our hypothesis for the power of restarts is indeed consistent with the "escaping local minima" hypothesis. However, restarts enable CDCL solvers to escape local minima in a way that works differently from local search algorithms. Specifically, CDCL solvers with restarts enabled escape local minima by jumping to a nearby space to learn "better clauses", while local search algorithms escape local minima by randomly jumping to a different part of the search space.

4 "Restarts Enable Learning Better Clauses" Hypothesis

In this section, we propose that restarts enable a CDCL solver to learn better clauses. To justify our hypothesis, we start by examining the claim by Hamadi et al. [16] stating that "In SAT, restarts policies are used to compact the assignment stack and improve the order of assumptions." Recall that in CDCL SAT solvers, the only thing that changes during a restart is the assignment stack, and hence the benefits of restarts should be observable on the assignment stack. In this paper, we show that this claim matches reality, that is, restarting frequently correlates with a smaller assignment stack. We then go one step further, and show that a compact assignment stack leads to better clause learning. That is, the solver ends up learning clauses with lower LBD, thereby supporting our hypothesis, and this in turn improves the solver performance.

Restarts do incur a cost though [27], for otherwise restart after every conflict would be the optimal policy for all inputs. After a solver restarts, it needs to make many decisions and propagations to rebuild the assignment stack from scratch. We call this the *rebuild time*. More precisely, whenever a solver performs a restart, we note the current time and the assignment stack size x right before the restart. Then the rebuild time is the time taken until either the solver encounters a new conflict or the new assignment stack size exceeds x through a series of decisions and propagations. Since we want to isolate the benefit of restart, we need to

discount the cost of rebuilding. We define *effective time* to be the solving time minus the rebuild times of every restart.

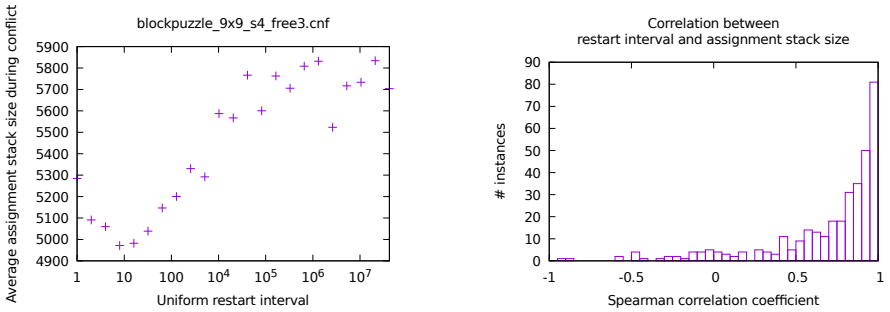
4.1 Confirming the "Compacting the Assignment Stack" Claim

We ran the Glucose 4.1 SAT solver [5] [4] with various frequencies of restarting to show that indeed restarts do compact the assignment stack. For all experiments in this paper, Glucose was run with the argument "-no-adapt" to prevent it from changing heuristics. For each instance in the SAT Competition 2017 main track, we ran Glucose 4.1 with a timeout of 3hrs of effective time on StarExec, a platform purposefully designed for evaluating SAT solvers [29]. The StarExec platform uses the Intel Xeon CPU E5-2609 at 2.40GHz with 10240 KB cache and 24 GB of main memory, running on Red Hat Enterprise Linux Server release 7.2 , and Linux kernel 3.10.0-514.16.1.el7.x86_64.

At every conflict, the assignment stack size is logged before backtracking occurs then the solver restarts after the conflict is analyzed (i.e., a uniform restart policy that restarts after every 1 conflict). We then ran the solver again on the same instance except the restart interval is doubled (i.e., a uniform restart policy that restarts after every 2 conflicts). We continue running the solver again and again, doubling the restart interval each time (i.e., a uniform restart policy that restarts after every 2^k conflicts) until the restart interval is so large that the solver never restarts before termination. For each instance, we construct a scatter plot, where the x-axis is the restart interval and the y-axis is the average assignment stack size for that restart policy on that instance, see Figure 1a for an example. We then compute the Spearman correlation between the two axes, a positive correlation denotes that smaller restart intervals correlate with smaller assignment stack size, that is evidence that frequent restarts compacts the assignment stack. We plot the Spearman correlations of all 350 instances in Figure 1b. 91.7% of the instances have a positive correlation coefficient. In conclusion, our experiments support the claim by Hamadi et al. [16] "restarts policies are used to compact the assignment stack."

It is important to note that this result is contingent on the branching heuristic implemented by the solver. If the branching heuristic is a static ordering, then the solver picks the decision variables in the same order after every restart and rebuilds the same assignment stack, hence the assignment stack does not get compacted. In our previous work [21], we showed that VSIDS-like branching heuristics "focus" on a small subset of logically related variables at any point in time. We believe a "focused" branching heuristic will see the compacting of assignment stack since a restart erases the assignment stack so a "focused" branching heuristic can reconstruct the assignment stack with only the subset of variables it is focused on.

[4] Glucose is a popular and competitive CDCL SAT solver often used in experiments because of its efficacy and simplicity (http://www.labri.fr/perso/lsimon/glucose/)



(a) (b)

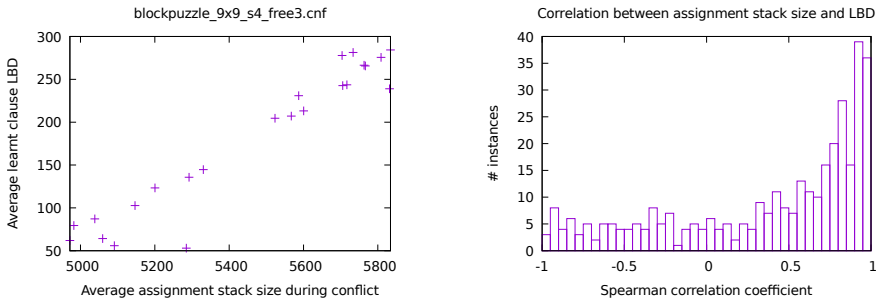
Fig. 1: (a) Scatter plot for a given instance showing increasing assignment stack size as the restarts become less frequent. (b) Histogram showing the distribution of Spearman correlations between the restart interval and the average assignment stack size for all 350 instances. The median correlation is 0.839.

4.2 Learning Better Clauses

We hypothesize that compacting the assignment stack generally leads to better learnt clauses, and that this is one of the benefits of restarts in SAT solvers in practice. Note that the clause learning schemes construct the learnt clause from a subset of variables on the assignment stack. Hence, a smaller assignment stack should lead to a learnt clause with smaller LBD than otherwise. To show this experimentally, we repeat the previous experiment where we ran Glucose 4.1 with the uniform restart policy restarting every 2^k conflicts for various parameters of k. At each conflict, we log the assignment stack size before backtracking and the LBD of the newly learnt clause. For each instance, we draw a scatter plot, where the x-axis is the average assignment stack size and the y-axis is the average LBD of learnt clauses, see Figure 2a. We compute the Spearman correlation between the two axes and plot these correlations in a histogram, see Figure 2b. 73.1% of the instances have a positive correlation coefficient.

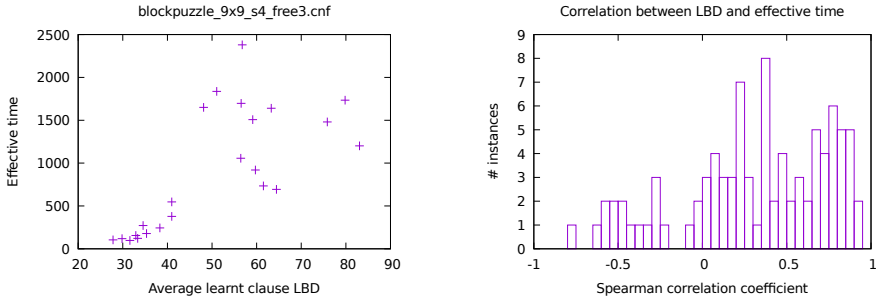
4.3 Solving Instances Faster

Although lower LBD is widely believed to be a sign of good quality clause, we empirically show that indeed lower LBD generally correlates with better effective time. This experiment is a repeat of the last two experiments, with the exception that the x-axis is the average learnt clause LBD and the y-axis is the effective time, see Figure 3a for an example. As usual, we compute the Spearman correlation between the two axes, discarding instances that timeout, and plot these correlations in a histogram, see Figure 2b. 77.8% of the instances have a positive correlation coefficient. As expected, learning lower LBD clauses tend to improve solver performance.



(a) (b)

Fig. 2: (a) Scatter plot for a given instance showing increasing assignment stack size correlates with increasing LBD of learnt clauses. (b) Histogram showing the distribution of Spearman correlations between the average assignment stack size and the average LBD of learnt clauses for all 350 instances. The median correlation is 0.607.



(a) (b)

Fig. 3: (a) Scatter plot for a given instance showing increasing average learnt clause LBD correlates with increasing effective time. (b) Histogram showing the distribution of Spearman correlations between the average learnt clause LBD and effective time for all 90 instances without timeouts. The median correlation is 0.366.

4.4 Clause Length

If the previous experiments replaced LBD with clause length, then the median Spearman correlation between the average assignment stack size and average learnt clause length is 0.822 and the median Spearman correlation between the average learnt clause length and effective time is 0.08.

4.5 Low LBD in Core Proof

We hypothesize that lower LBD clauses are preferable for unsatisfiable instances because they are more likely to be a core learnt clause, that is, a learnt clause that is actually used in the derivation of the final empty clause. We performed the following experiment to support our hypothesis. We ran Glucose with no clause deletion on all 350 instances of the SAT Competition 2017 main track with 5000 seconds timeout. We turned off clause deletion because the deletion policy in Glucose inherently biases towards low LBD clauses by deleting learnt clauses with higher LBDs. We used DRAT-trim [30] to extract the core proof from the output of Glucose, i.e, the subset of clauses used in the derivation of the empty clause. We then computed the ratio between the mean LBD of the core learnt clauses and the mean LBD of all the learnt clauses. Lastly we plotted the ratios in a histogram, see Figure 4. For the 57 instances for which core DRAT proofs were generated successfully, all but one instance has a ratio below 1. In other words, lower LBD clauses are more likely to be used in deriving the empty clause than clauses with higher LBD.

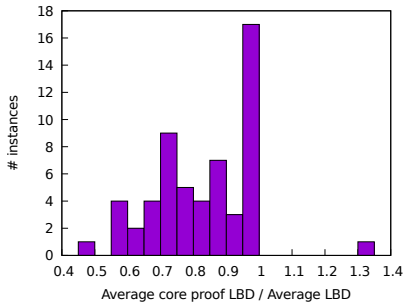


Fig. 4: Histogram for the ratio between the mean LBD of the learnt clauses in the core proof and the mean LBD of all the learnt clauses for the 57 unsatisfiable instances DRAT-trim produced a core proof.

4.6 New Restart Policy

Based on the above observations, we designed a new machine learning-based restart policy that is competitive with the state-of-the-art policies. As shown

earlier, empirically restarts reduce LBD, hence we design a restart policy that tries to avoid high LBDs by restarting. Intuitively, our restart policy does the following: restart if the next learnt clause has an LBD in the 99.9^{th} percentile. Implementing this policy requires new techniques to answer the two following questions: is an LBD in the 99.9^{th} percentile and what is the LBD of the next learnt clause. We designed techniques to estimate answers to these two questions. The answer for the first question is the normal distribution is a good approximation for the right tail of the LBD distribution. The answer for the second question is to use machine learning to predict the LBD of the next clause.

5 A Machine Learning-based Restart Policy

In this section, we describe our machine learning-based restart policy. We first start by answering the two questions posed in the last subsection regarding LBD percentile and predicting LBD of the next clause.

5.1 LBD Percentile

Given the LBD of a clause, it is unclear a priori how to label it as "good" or "bad". Some heuristics set a constant threshold and any LBDs above this threshold are considered bad. For example, Plingeling [7] considers learnt clauses with LBD greater 7 to be bad, and these clauses are not shared with the other workers. COMiniSatPS considers learnt clauses with LBD greater than 8 to be bad, and hence these clauses are readily deleted [25]. The state-of-the-art Glucose restart policy [4] on the other hand uses the mean LBD multiplied by a fixed constant as a threshold. The problem with using a fixed constant or the mean times a constant for thresholds is that we do not have a priori estimate of how many clauses exceed this threshold, and these thresholds seem arbitrary. Using arbitrary thresholds makes it harder to reason about solver heuristics, and in this context, the efficacy of restart policies.

We instead propose that for any given input it is more appropriate to use dynamic threshold that is computed based on the history of the CDCL solver's run on that input. At any point in time during the run of the solver, the dynamic threshold is computed as the 99.9^{th} percentile of LBDs of the learnt clauses seen during the run so far. Before we estimate whether an LBD is in the 99.9^{th} percentile, the first step is to analyze the distribution of LBDs seen in practice. In this experiment, the Glucose solver was run on all 350 instances in SAT Competition 2017 main track for 30 minutes and the LBDs of all the learnt clauses were recorded. Figure 5 shows the histogram of LBDs for 4 representative instances. As can be seen from the distributions of these representative instances, either their LBD distribution is close to normal or a right-skewed one.

Even though the right-skew distribution is not normal, the high percentiles can still be approximated by the normal distribution since the right tail is close to the normal curve. We conducted the following experiment to support this claim. For each instance, we computed the mean and variance of the LBD distribution

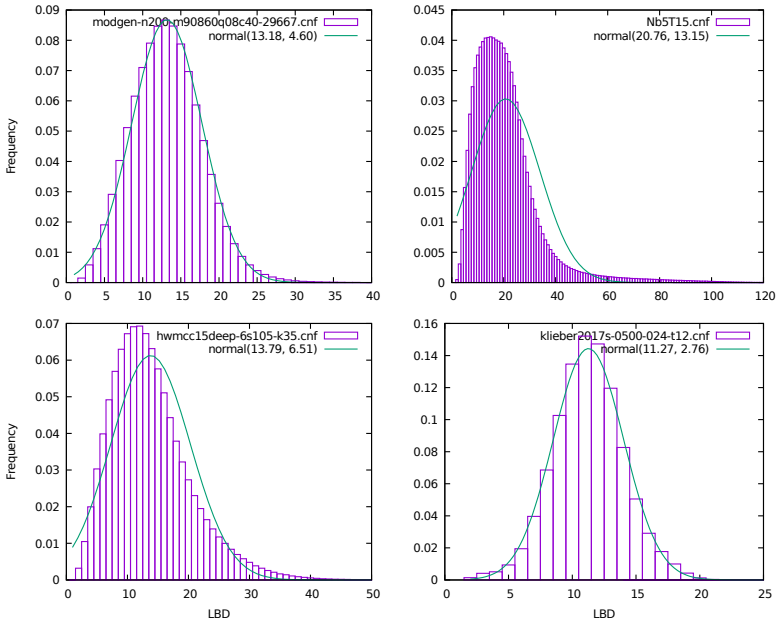


Fig. 5: Histogram of LBDs of 4 instances. A normal distribution with the same mean and variance is overlaid on top for comparison.

to draw a normal distribution with the same mean and variance. We used the normal distribution to predict the LBD x at the 99.9^{th} percentile. We then checked the recorded LBD distribution to see the actual percentile of x. Figure 6 is a histogram of all the actual percentiles. Even in the worst case, the predicted 99.9^{th} percentile turned out to be the 97.1^{th} percentile. Hence for this benchmark the prediction of the 99.9^{th} percentile using the normal distribution has an error of less than 3 percentiles. Additionally, only 6 of the 350 instances predicted an LBD that was in the 100^{th} percentile and all 6 of these instances solved in less than 130 conflicts hence the prediction was made with very little data.

These figures were created by analyzing the LBD distribution at the end of a 30 minute run of Glucose, and we note the results are similar before the 30 minutes is up. Hence the 99.9^{th} percentile of LBDs can be approximated as the 99.9^{th} percentile of $norm(\mu, \sigma^2)$. The mean μ and variance σ^2 are estimated by the sample mean and sample variance of all the LBDs seen thus far, which is computed incrementally so the computational overhead is low. The 99.9^{th} percentile of the normal distribution maps to the z-score of 3.08, that is, an LBD is estimated to be in the 99.9^{th} percentile if it is greater than $\mu + 3.08 \times \sigma$.

5.2 LBD of Next Clause

Since at any point during the run of a solver, the LBD of the "next learnt" clause is unknown, we propose the use of machine learning to predict that LBD

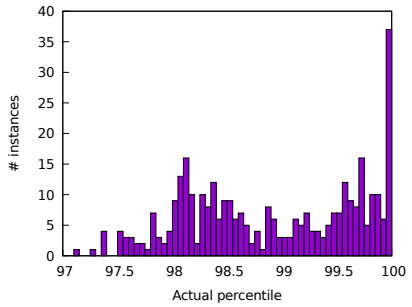


Fig. 6: Histogram of the actual percentiles of the LBD predicted to be the 99.9^{th} percentile using a normal distribution.

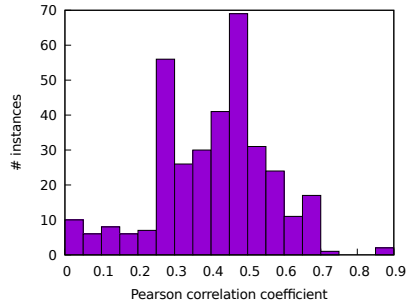


Fig. 7: Histogram of the Pearson correlation between the "previous" and "next" LBD for the instances in the SAT Competition 2017 main track benchmark.

instead. This requires finding good features that correlate with the next LBD. We hypothesize that LBDs of recent past learnt clauses correlate with the LBD of the next learnt clause.

In this experiment, Glucose was run on all 350 instances of the 2017 Competition main track and the LBDs of all the learnt clauses were recorded. Let n be the number of LBDs recorded for an instance. A table with two columns of length $n-1$ are created. For each row i in this two column table, the first column contains the LBD of the i^{th} conflict and the second column contains the LBD of the $(i+1)^{th}$ conflict. Intuitively, after the solver finishes resolving the i^{th} conflict, the i^{th} learnt clause is the "previous" learnt clause represented by the first column. Correspondingly, the "next" learnt clause is the $(i+1)^{th}$ learnt clause represented by the second column. For each instance that took more than 100 conflicts to solve, we computed the Pearson correlation between the first and second column and plot all these correlations in a histogram, see Figure 7.

Our results show that the "previous LBD" is correlated with the "next LBD" which supports the idea that recent LBDs are good features to predict the next LBD via machine learning. In addition, all the correlations are positive, meaning that if the previous LBD is high (resp. low) then the next LBD is expected to be high (resp. low). Perhaps this explains why the Glucose restart policy [4] is effective. Additionally, we note that for the average instance, the LBD of the learnt clause after a restart is smaller than the LBD of the learnt clause right before that restart 74% of the time, showing the effect of restarts on LBD.

This paper proposes learning the function $f(l_{-1}, l_{-2}, l_{-3}, l_{-1} \times l_{-2}, l_{-1} \times l_{-3}, l_{-2} \times l_{-3}) = l_{next}$ where l_{-i} is the LBD of the learnt clause from i conflicts ago and $l_{-i} \times l_{-j}$ are products of previous LBDs to incorporate their feature interaction, and l_{next} is the LBD of the next clause. This function is approximated using linear regression where θ_i are coefficients to be trained by the machine learning algorithm:

$\tilde{f}(l_{-1}, l_{-2}, l_{-3}, l_{-1} \times l_{-2}, l_{-1} \times l_{-3}, l_{-2} \times l_{-3}) = \theta_0 + \theta_1 \times l_{-1} + \theta_2 \times l_{-2} + \theta_3 \times l_{-3} + \theta_4 \times l_{-1} \times l_{-2} + \theta_5 \times l_{-1} \times l_{-3} + \theta_6 \times l_{-2} \times l_{-3}$

Since LBDs are streamed in as conflicts occur, an online algorithm that can incrementally adjust the θ_i coefficients cheaply is required. We use the state-of-the-art Adam algorithm [20] from machine learning literature because it scales well with the number of dimensions, is computationally efficient, and converges quickly for many problems. The Adam algorithm is in the family of stochastic gradient descent algorithms that adjusts the coefficients to minimize the squared error, where the error is the difference between the linear function's prediction and the actual next LBD. The algorithm computes the gradient of the squared error function and adjusts the coefficients in the opposite direction of the gradient to minimize the squared error function. For the parameters of Adam, we use the values recommended by the original authors [20].

The coefficients θ_i are all initialized to 0 at the start of the search. Whenever a new clause is learnt, one iteration of Adam is applied with the LBDs of the three previous learnt clauses and their pairwise products as features and the LBD of the new clause as the target. The coefficients θ_i are adjusted in the process. When BCP reaches a fixed point without a conflict, the function $\tilde{f}$ is queried with the current set of coefficients θ_i to predict the LBD of the next clause. If the prediction exceeds the sample mean plus 3.08 standard deviations (i.e., approximately the 99.9^{th} percentile), a restart is triggered.

The new restart policy, called *machine learning-based restart* (MLR) policy, is shown in Algorithm 1. Since the mean, variance, and coefficients are computed incrementally, MLR has a very low computational overhead.

6 Experimental Evaluation

To test how MLR performs, we conducted an experimental evaluation to see how Glucose performs with various restart policies. Two state-of-the-art restart policies are used for comparison with MLR: Glucose (named after the solver itself) [4] and Luby [22]. The benchmark consists of all instances in the application and hard combinatorial tracks from the SAT Competition 2014 to 2017 totaling 1411 unique instances. The Glucose solver with various restart policies were run over the benchmark on StarExec. For each instance, the solver was given 5000 seconds of CPU time and 8GB of RAM. The results of the experiment are shown in Figure 8. The source code of MLR and further analysis of the experimental results are available on our website [1].

The results show that MLR is in between the two state-of-the-art policies of Glucose restart and Luby restart. For this large benchmark, MLR solves 19 instances more than Luby and 20 instances fewer than Glucose. Additionally, the learned coefficients in MLR $\sigma_1, \sigma_2, \sigma_3$ corresponding to the coefficients of the features representing recent past LBDS are nonnegative 91% of the time at the end of the run. This reinforces the notion that previous LBDs are positively correlated with the next LBD.

Algorithm 1 Pseudocode for the new restart policy MLR.

function INITIALIZE ▷ Called once at the start of search.
  $\alpha \leftarrow 0.001, \epsilon \leftarrow 0.00000001, \beta_1 \leftarrow 0.9, \beta_2 \leftarrow 0.999$ ▷ Adam parameters.
  $conflicts \leftarrow 0, conflictsSinceLastRestart \leftarrow 0$
  $t \leftarrow 0$ ▷ Number of training examples.
  $prevLbd_3 \leftarrow 0, prevLbd_2 \leftarrow 0, prevLbd_1 \leftarrow 0$ ▷ LBD of clause learnt 3/2/1 conflicts ago.
  $\mu \leftarrow 0, m2 \leftarrow 0$ ▷ For computing sample mean and variance of LBDs seen.
  for v in $0..|FeatureVector()| - 1$ **do** ▷ Initialize θ, m, v to be vectors of zeros.
    $\theta_i \leftarrow 0, m_i \leftarrow 0, v_i \leftarrow 0$ ▷ Coefficients of linear function and Adam internals.
function FEATUREVECTOR
  return $[1, prevLbd_1, prevLbd_2, prevLbd_3, prevLbd_1 \times prevLbd_2, prevLbd_1 \times prevLbd_3, prevLbd_2 \times prevLbd_3]$
function AFTERCONFLICT(LearntClause) ▷ Update the coefficients θ using Adam.
  $conflicts \leftarrow conflicts + 1, conflictsSinceLastRestart \leftarrow conflictsSinceLastRestart + 1$
  $nextLbd \leftarrow LBD(LearntClause)$
  $\delta \leftarrow nextLbd - \mu, \mu \leftarrow \mu + \delta/conflicts, \Delta \leftarrow nextLbd - \mu, m2 \leftarrow m2 + \delta \times \Delta$
  if $conflicts > 3$ **then** ▷ Apply one iteration of Adam.
    $t \leftarrow t + 1$
    $features \leftarrow FeatureVector()$
    $predict \leftarrow \theta \cdot features$
    $error \leftarrow predict - nextLbd$
    $g \leftarrow error \times features$
    $m \leftarrow \beta_1 \times m + (1 - \beta_1) \times g, v \leftarrow \beta_2 \times v + (1 - \beta_2) \times g \times g$
    $\hat{m} \leftarrow m/(1 - \beta_1^t), \hat{v} \leftarrow v/(1 - \beta_2^t)$
    $\theta \leftarrow \theta - \alpha \times \hat{m}/(\sqrt{\hat{v}} + \epsilon)$
  $prevLbd_3 \leftarrow prevLbd_2, prevLbd_2 \leftarrow prevLbd_1, prevLbd_1 \leftarrow nextLbd$
function AFTERBCP(IsConflict)
  if $\neg IsConflict \wedge conflicts > 3 \wedge conflictsSinceLastRestart > 0$ **then**
    $\sigma \leftarrow \sqrt{m2/(conflicts - 1)}$
    if $\theta \cdot FeatureVector() > \mu + 3.08\sigma$ **then** ▷ Estimate if next LBD in 99.9th percentile.
      $conflictsSinceLastRestart \leftarrow 0, Restart()$

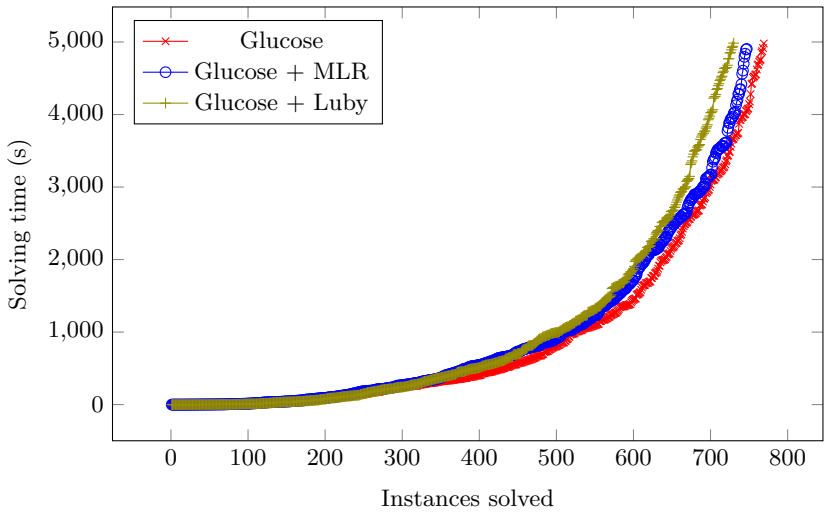


Fig. 8: Cactus plot of two state-of-the-art restart policies and MLR. A point (x, y) is interpretted as x instances have solving time less than y seconds for the given restart policy. Being further right means more instances are solved, further down means instances are solved faster.

7 Related Work

Restart policies come in two flavors: static and dynamic. Static restart policies predetermine when to restart before the search begins. The state-of-the-art for static is the Luby [22] restart heuristic which is theoretically proven to be an optimal universal restart policy for Las Vegas algorithms. Dynamic restart policies determine when to restart on-the-fly during the run of the solver, typically by analyzing solver statistics. The state-of-the-art for dynamic is the restart policy proposed by Glucose [4] that keeps a short-term and a long-term average of LBDs. The short-term is the average of the last 50 LBDs and the long-term is the average of all the LBDs encountered since the start of the search. If the short-term exceeds the long-term by a constant factor then a restart is triggered. Hence the Glucose policy triggers a restart when the recent LBDs are high on average whereas the restart policy proposed in this paper restarts when the predicted LBD of the next clause is high. Biere et al. [8] propose a variation of the Glucose restart where an exponential moving average is used to compute the short-term and long-term averages. Haim and Walsh [15] introduced a machine learning-based technique to select a restart policy from a portfolio after 2100 conflicts. The MABR policy [24] uses multi-armed bandits to minimize average LBD by dynamically switching between a portfolio of policies. Our use of machine learning differs from these previous methods in that machine learning is part of the restart policy itself, rather than using machine learning as a meta-heuristic to select between a fixed set of restart policies.

Proof-complexity theoretic Considerations: Theorists have conjectured that restarts give the solver more power in a proof-complexity sense than a solver without restarts. A CDCL solver with asserting clause learning scheme can polynomially simulate general resolution [26] with nondeterministic branching and restarts. It was independently shown that a CDCL solver with sufficiently random branching and restarts can simulate bounded-width resolution [2]. It remains an open question whether these results hold if the solvers does not restart. This question has remained stubbornly open for over two decades now. We refer the reader to the excellent articles by Buss et al. on attempts at understanding the power of restarts via proof-complexity theory [10, 9].

8 Conclusion

We showed that restarts positively impact the clause learning of CDCL solvers by decreasing the LBD of learnt clauses (thus improving their quality) compared to no restarts. However restarting too frequently is computationally expensive. We propose a new restart policy called MLR that tries to find the right balance in this trade-off. We use z-scores of the normal distribution to efficiently approximate the high percentiles of the LBD distribution. Additionally, we use machine learning to predict the LBD of the next clause, given the previous 3 LBDs and their pairwise products. Experimentally, the new restart policy is competitive with the current state-of-the-art.

References

[1] MLR source code, https://sites.google.com/a/gsd.uwaterloo.ca/maplesat/mlr

[2] Atserias, A., Fichte, J.K., Thurley, M.: Clause-Learning Algorithms with Many Restarts and Bounded-Width Resolution. In: Kullmann, O. (ed.) Theory and Applications of Satisfiability Testing - SAT 2009. pp. 114–127. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)

[3] Audemard, G., Simon, L.: Predicting Learnt Clauses Quality in Modern SAT Solvers. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence. pp. 399–404. IJCAI'09, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2009)

[4] Audemard, G., Simon, L.: Refining Restarts Strategies for SAT and UNSAT, pp. 118–126. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)

[5] Audemard, G., Simon, L.: Glucose and Syrup in the SAT17. In: Proceedings of SAT Competition 2017: Solver and Benchmark Descriptions. pp. 16–17 (2017)

[6] Biere, A., Biere, A., Heule, M., van Maaren, H., Walsh, T.: Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications. IOS Press, Amsterdam, The Netherlands, The Netherlands (2009)

[7] Biere, A.: Lingeling, Plingeling and Treengeling Entering the SAT Competition 2013. In: Proceedings of SAT Competition 2013: Solver and Benchmark Descriptions. pp. 51–52 (2013)

[8] Biere, A., Fröhlich, A.: Evaluating CDCL Variable Scoring Schemes, pp. 405–422. Springer International Publishing, Cham (2015)

[9] Bonet, M.L., Buss, S., Johannsen, J.: Improved Separations of Regular Resolution from Clause Learning Proof Systems. Journal of Artificial Intelligence Research 49, 669–703 (2014), https://doi.org/10.1613/jair.4260

[10] Buss, S.R., Kolodziejczyk, L.A.: Small Stone in Pool. Logical Methods in Computer Science 10(2) (2014)

[11] Cadar, C., Ganesh, V., Pawlowski, P.M., Dill, D.L., Engler, D.R.: EXE: Automatically Generating Inputs of Death. In: Proceedings of the 13th ACM Conference on Computer and Communications Security. pp. 322–335. CCS '06, ACM, New York, NY, USA (2006)

[12] Clarke, E., Biere, A., Raimi, R., Zhu, Y.: Bounded Model Checking Using Satisfiability Solving. Formal Methods in System Design 19(1), 7–34 (2001)

[13] Cook, S.A.: The complexity of theorem-proving procedures. In: Proceedings of the Third Annual ACM Symposium on Theory of Computing. pp. 151–158. STOC '71, ACM, New York, NY, USA (1971), http://doi.acm.org/10.1145/800157.805047

[14] Gomes, C.P., Selman, B., Crato, N., Kautz, H.: Heavy-Tailed Phenomena in Satisfiability and Constraint Satisfaction Problems. Journal of Automated Reasoning 24(1-2), 67–100 (Feb 2000)

[15] Haim, S., Walsh, T.: Restart Strategy Selection Using Machine Learning Techniques. In: Kullmann, O. (ed.) Theory and Applications of Satisfiability Testing - SAT 2009. pp. 312–325. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)

[16] Hamadi, Y., Jabbour, S., Sais, L.: ManySAT: a Parallel SAT solver. Journal on Satisfiability 6, 245–262 (2008)

[17] Hirsch, E.A., Kojevnikov, A.: UnitWalk: A new SAT solver that uses local search guided by unit clause elimination. Annals of Mathematics and Artificial Intelligence 43(1), 91–111 (Jan 2005)

[18] Katebi, H., Sakallah, K.A., Marques-Silva, J.a.P.: Empirical Study of the Anatomy of Modern Sat Solvers. In: Proceedings of the 14th International Conference on

Theory and Application of Satisfiability Testing. pp. 343–356. SAT'11, Springer-Verlag, Berlin, Heidelberg (2011)
[19] Kautz, H., Selman, B.: Planning As Satisfiability. In: Proceedings of the 10th European Conference on Artificial Intelligence. pp. 359–363. ECAI '92, John Wiley & Sons, Inc., New York, NY, USA (1992)
[20] Kingma, D.P., Ba, J.: Adam: A Method for Stochastic Optimization. CoRR abs/1412.6980 (2014), http://arxiv.org/abs/1412.6980
[21] Liang, J.H., Ganesh, V., Zulkoski, E., Zaman, A., Czarnecki, K.: Understanding VSIDS Branching Heuristics inConflict-Driven Clause-Learning SAT Solvers. In: Piterman, N. (ed.) Hardware and Software: Verification and Testing. pp. 225–241. Springer International Publishing, Cham (2015)
[22] Luby, M., Sinclair, A., Zuckerman, D.: Optimal Speedup of Las Vegas Algorithms. Information Processing Letters 47(4), 173–180 (Sep 1993)
[23] Marques-Silva, J.P., Sakallah, K.A.: GRASP-A New Search Algorithm for Satisfiability. In: Proceedings of the 1996 IEEE/ACM International Conference on Computer-aided Design. pp. 220–227. ICCAD '96, IEEE Computer Society, Washington, DC, USA (1996)
[24] Nejati, S., Liang, J.H., Gebotys, C., Czarnecki, K., Ganesh, V.: Adaptive Restart and CEGAR-Based Solver for Inverting Cryptographic Hash Functions. In: Paskevich, A., Wies, T. (eds.) Verified Software. Theories, Tools, and Experiments. pp. 120–131. Springer International Publishing, Cham (2017)
[25] Oh, C.: COMiniSatPS the Chandrasekhar Limit and GHackCOMSPS. In: Proceedings of SAT Competition 2017: Solver and Benchmark Descriptions. pp. 12–13 (2017)
[26] Pipatsrisawat, K., Darwiche, A.: On the Power of Clause-Learning SAT Solvers with Restarts, pp. 654–668. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
[27] Ramos, A., van der Tak, P., Heule, M.J.H.: Between Restarts and Backjumps. In: Sakallah, K.A., Simon, L. (eds.) Theory and Applications of Satisfiability Testing - SAT 2011. pp. 216–229. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
[28] Saitta, L., Sebag, M.: Phase Transitions in Machine Learning, pp. 767–773. Springer US, Boston, MA (2010)
[29] Stump, A., Sutcliffe, G., Tinelli, C.: Automated Reasoning: 7th International Joint Conference, IJCAR 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 19-22, 2014. Proceedings, chap. StarExec: A Cross-Community Infrastructure for Logic Solving, pp. 367–373. Springer International Publishing, Cham (2014)
[30] Wetzler, N., Heule, M.J.H., Hunt, W.A.: DRAT-trim: Efficient Checking and Trimming Using Expressive Clausal Proofs. In: Sinz, C., Egly, U. (eds.) Theory and Applications of Satisfiability Testing – SAT 2014. pp. 422–429. Springer International Publishing, Cham (2014)