Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)

An Empirical Study of Branching Heuristics through the Lens of Global Learning Rate*

Jia Liang, Hari Govind, Pascal Poupart, Krzysztof Czarnecki and **Vijay Ganesh**
University of Waterloo, Waterloo, Canada
{j8liang, hgvediramanakrishnan, ppoupart, k2czarne, vijay.ganesh}@uwaterloo.ca

Abstract

In this paper, we analyze a suite of 7 well-known branching heuristics proposed by the SAT community and show that the better heuristics tend to generate more learnt clauses per decision, a metric we define as the global learning rate (GLR). We propose GLR as a metric for the branching heuristic to optimize. We test our hypothesis by developing a new branching heuristic that *maximizes GLR* greedily. We show empirically that this heuristic achieves very high GLR and interestingly very low literal block distance (LBD) over the learnt clauses. In our experiments this greedy branching heuristic enables the solver to solve instances faster than VSIDS, when the branching time is taken out of the equation. This experiment is a good proof of concept that a branching heuristic maximizing GLR will lead to good solver performance modulo the computational overhead. Finally, we propose a new branching heuristic, called SGDB, that uses machine learning to cheapily approximate greedy maximization of GLR. We show experimentally that SGDB performs on par with the VSIDS branching heuristic.

1 Introduction

Searching through a large, potentially exponential, search space is a reoccurring problem in many fields of computer science and many researchers have come to rely on SAT solvers as a general purpose tool to efficiently perform the search. Two notable milestones that are key to the success of SAT solvers are the Variable State Independent Decaying Sum (VSIDS) branching heuristic (and its variants) [Moskewicz *et al.*, 2001] and conflict analysis techniques [Marques-Silva and Sakallah, 1996].

One of the challenges in designing branching heuristics is that it is not clear what constitutes a good decision variable. We proposed one solution to this issue in our LRB branching heuristic paper [Liang *et al.*, 2016b], which is to frame branching as an optimization problem. We defined a computable metric called *learning rate* and defined the objective as maximizing the learning rate. Good decision variables are ones with high learning rate. Since learning rate is expensive to compute a priori, we used a multi-armed bandit learning algorithm to estimate the learning rate on-the-fly as the basis for the LRB branching heuristic [Liang *et al.*, 2016b].

In this paper, we deepen our previous work and our starting point remains the same, namely, branching heuristics should be designed to solve the optimization problem of maximizing learning rate. In LRB, the learning rate metric is defined per variable. In this paper, we define a new metric, called the *global learning rate* (GLR) to measure the solver's overall propensity to generate conflicts, rather than the variable-specific metric we defined in the case of LRB. Not only do we observe that the effectiveness of extant branching heuristics correlates with their capacity to maximize GLR, we also find that forcing VSIDS to increase GLR improves its performance when the branching computation is factored out. Lastly we present a new branching heuristic called stochastic gradient descent branching (SGDB) that uses machine learning to greedily maximize GLR, and is shown experimentally to perform on par with VSIDS.

2 Background

Conflict-driven clause-learning (CDCL) [Marques-Silva and Sakallah, 1996] is currently the most effective paradigm of SAT solvers in practice for a wide range of problems. They perform an exhaustive backtracking search by assigning Boolean variables to values trying to find a solution. The branching heuristic is responsible for the order in which the variables are assigned, and this order has a huge impact on the running time of the solver. When the solver detects that its assignment of variables cannot be extended to a solution, it is in *conflict* and needs to backtrack to an earlier state.

CDCL solvers analyze every conflict to produce a new clause that prevents the same or a similar conflict from reoccurring. This requires maintaining an implication graph where the nodes are assigned literals and edges are implications forced by Boolean constraint propagation (BCP). When a clause is falsified, the CDCL solver invokes *conflict analysis* to produce a *learnt clause* from the conflict. It does so by *cutting* the implication graph, typically at the first-UIP [Marques-Silva and Sakallah, 1996], into the *reason side*

*This paper is an abridged version of a paper with the same title at the SAT 2017 conference where it won the best student paper honourable mention award.

and the *conflict side* with the condition that the decision variables appear on the reason side and the falsified clause appears on the conflict side. A new learnt clause is constructed by negating the reason side literals incident to the cut. Literal block distance (LBD) is a popular metric for measuring the "quality" of a learnt clause [Audemard and Simon, 2009]. The LBD is defined as the number of distinct decision levels of the variables in the clause, the lower the LBD the better.

3 GLR Maximization as a Branching Heuristic Objective

To frame the branching heuristic as an optimization problem, the first step is to define the objective. Ideally the objective of the branching heuristic is to minimize the total running time. However calculating the running time a priori is infeasible, which makes it unsuitable as an objective for branching. Instead, we target an easy to compute feature that correlates with solving time.

We define the *global learning rate* (GLR) of a solver as $GLR := \frac{\#\ of\ conflicts}{\#\ of\ decisions}$. We will justify why maximizing GLR is a reasonable objective for a branching heuristic. Past research concludes that clause learning is the most important feature for good performance in a CDCL solver [Katebi *et al.*, 2011], so perhaps it is not surprising that increasing the rate at which clauses are learnt is a reasonable objective. In our experiments, we assume the learning scheme is first-UIP since it is universally used by all modern CDCL solvers.

We propose the following hypothesis: *for a given instance, the branching heuristic that achieves higher GLR tends to solve that instance faster than heuristics with lower GLR*. In the following experiment, we tested the above hypothesis with 7 branching heuristics: LRB [Liang *et al.*, 2016b], CHB [Liang *et al.*, 2016a], VSIDS (MiniSat [Eén and Sörensson, 2004] variation of VSIDS), CVSIDS (Chaff [Moskewicz *et al.*, 2001] variation of VSIDS), Berkmin [Goldberg and Novikov, 2007], DLIS [Marques-Silva, 1999], and Jeroslow-Wang [Jeroslow and Wang, 1990]. We created 7 versions of MapleSAT [1], one for each branching heuristic, keeping the code unrelated to the branching heuristic untouched. We ran all 7 solvers on each application and hard combinatorial instance from every SAT Competition and SAT Race held between 2009 and 2016 with a 1800 second timeout. Duplicate instances are removed. We recorded the GLR and the average LBD of clauses learnt at termination. We then computed the median GLR and median average LBD over the entire benchmark. All experiments in this paper were conducted on StarExec [Stump *et al.*, 2014]. The code for all our experiments in this paper can be found on our website [2]. The results are presented in Table 1. Note that sorting by GLR in decreasing order, sorting by instances solved in decreasing order, and sorting by LBD in increasing order produces almost the same ranking. This gives credence to our hypothesis that GLR correlates with branching heuristic effectiveness. Additionally, the experiment shows that high GLR correlates with low LBD.

[1] https://sites.google.com/a/gsd.uwaterloo.ca/maplesat/

[2] https://sites.google.com/a/gsd.uwaterloo.ca/maplesat/sgd

Heuristic	*Median LBD*	*Median GLR*	*# Solved*
LRB	7.435	0.589	1552
CHB	8.430	0.512	1499
MVSIDS	9.548	0.524	1436
CVSIDS	9.988	0.413	1309
BERKMIN	12.235	0.359	629
DLIS	7.705	0.215	318
JW	25.061	0.068	290

Table 1: LBD, GLR, and number of instances solved for 7 different branching heuristics, sorted by the number of instances solved. Jeroslow-Wang (JW) is modified to also account for learnt clauses.

Maximizing GLR also makes intuitive sense when viewing the CDCL solver as a proof system. Every conflict generates a new lemma in the proof. Every decision is like a new "case" in the proof. Intuitively, the solver wants to generate lemmas quickly using as few cases as possible, or in other words, maximize conflicts with as few decisions as possible. This is equivalent to maximizing GLR. Of course in practice not all lemmas/learnt clauses are of equal quality, so the quality such as LBD is also an important objective.

4 Greedy Maximization of GLR

Finding the globally optimal branching sequence that maximizes GLR is intractable in general. Hence we tackle a simpler problem to maximize GLR greedily instead. Although this is too computationally expensive to be effective in practice, it provides a proof of concept for GLR maximization and a gold standard for subsequent branching heuristics.

We define the function $c : PA \to \{1, 0\}$ that maps partial assignments to either class 1 or class 0. Class 1 is the "conflict class" which means that applying BCP to the input partial assignment with the current clause database would encounter a conflict once BCP hits a fixed-point. Otherwise the input partial assignment is given the class 0 for "non-conflict class". Note that c is a mathematical function with no side-effects, that is applying it does not alter the state of the solver. The function c is clearly decidable via one call to BCP, although it is quite costly when called too often.

The greedy GLR branching (GGB) heuristic is a branching heuristic that maximizes GLR greedily. When it comes time to branch, the branching heuristic is responsible for appending a decision variable (plus a sign) to the current partial assignment. GGB prioritizes decision variables where the new partial assignment falls in class 1 according to the function c. That is, GGB branches on decision variables that cause a conflict during the subsequent call to BCP, if such variables exist. See Algorithm 1 for the implementation of GGB.

Unfortunately, GGB is very computationally expensive due to the numerous calls to the c function every time a new decision variable is needed. However, we show that GGB significantly increases the GLR relative to the base branching heuristic VSIDS. Additionally, we show that if the time to compute the decision variables was ignored, then GGB would be a more efficient heuristic than VSIDS. This suggests we need to cheaply approximate GGB to avoid the heavy computation. A cheap and accurate approximation of GGB would

Algorithm 1 Pseudocode for the GGB heuristic using the function c to greedily maximize GLR. Note that GGB is a meta-heuristic, it takes an existing branching heuristic (VSIDS in the following pseudocode) and makes it greedier by causing conflicts whenever possible. In general, VSIDS can be replaced with any other branching heuristic.

```
1: function PHASESAVING(Var)                          ▷ Return variable plus a sign.
2:     return mkLit(Var, Var_savedPolarity)
3:
4: function VSIDS(Vars) ▷ Return variable with highest VSIDS activity plus a sign.
5:     return PhaseSaving(argmax_{v∈Vars} v_activity)
6:
7: function GGB
8:     CPA ← CurrentPartialAssignment
9:     V ← UnassignedVariables
10:    oneClass ← {v ∈ V  |  c(CPA ∪ {PhaseSaving(v)}) = 1}
11:    zeroClass ← V \ oneClass
12:    if oneClass ≠ ∅ then                           ▷ Next BCP will cause a conflict.
13:        return VSIDS(oneClass)
14:    else                                           ▷ Next BCP will not cause a conflict.
15:        return VSIDS(zeroClass)
```

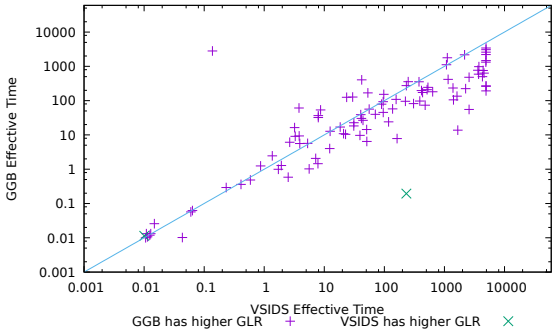

Figure 1: GGB vs VSIDS. Each point in the plot is a comparable instance. Note that the axes are in log scale. GGB has a higher GLR for all but 2 instances. GGB has a mean GLR of 0.74 for this benchmark whereas VSIDS has a mean GLR of 0.59.

in theory be a better branching heuristic than VSIDS. The following experiments were performed with MapleSAT with restarts and clause deletion turned off to minimize the effects of external heuristics. For each of the 300 instances in the SAT Competition 2016 application category, MapleSAT was ran twice, the first run configured with VSIDS and the second run configured with GGB. The run with VSIDS used a timeout of 5000 seconds. The run with GGB used a timeout of 24 hours to account for the heavy computational overhead. We define *effective time* as the solving time minus the time spent by the branching heuristic selecting variables. Figure 1 shows the results of effective time between the two heuristics. Only *comparable* instances are plotted. An instance is comparable if either both heuristics solved the instance or one heuristic solved the instance with an effective time of x seconds while the other heuristic timed out with an effective time greater than x seconds.

Of the comparable instances, GGB solved 69 instances with a lower effective time than VSIDS and 29 instances with a higher effective time. Hence if the branching was free, then GGB would solve instances faster than VSIDS 70% of the time. GGB achieves a higher GLR than VSIDS for all but

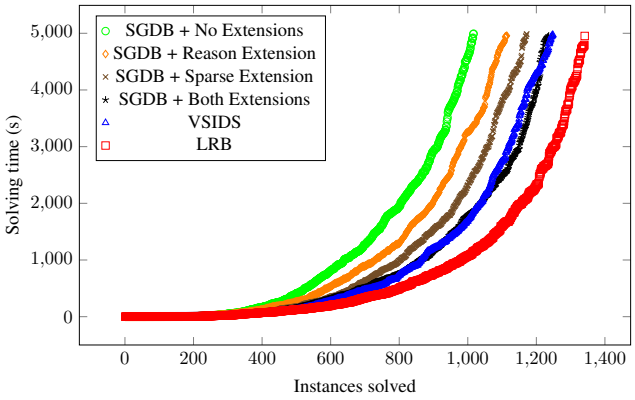

Figure 2: A cactus plot of various configurations of SGDB, VSIDS, and LRB on the entire benchmark with duplicate instances removed. A point (x, y) can be interpreted as x instances have a solving time of y seconds or less with the given heuristic. Being to the right and down is better.

2 instances, hence it does a good job increasing GLR as expected. Additionally, GGB has a lower LBD than VSIDS for 72 of the 98 comparable instances. We believe this is because GGB by design causes conflicts earlier when the decision level is low, which keeps the LBD small since LBD cannot exceed the current decision level.

5 Stochastic Gradient Descent Branching Heuristic

GGB is too expensive in practice due to the computational cost of computing the $c : PA \rightarrow \{1, 0\}$ function. Instead, we propose a new braching heuristic called stochastic gradient descent branching (SGDB) to cheaply approximate c with online stochastic gradient descent [Bottou, 1998]. SGDB learns the logistic regression [Cox, 1958] function $\tilde{c} : \mathbb{R}^n \rightarrow [0, 1]$ where $\mathbb{R}^n$ is the partial assignment's feature vector and $[0, 1]$ is the probability the partial assignment is in class 1, the conflict class. Online training is a good fit since the function c we are approximating is non-stationary due to the clause database changing over time. For an instance with n Boolean variables and a partial assignment PA, we introduce the features $x_1, ..., x_n$ defined as follows: $x_i = 1$ if variable $i \in PA$, otherwise $x_i = 0$. With logistic regression, $\tilde{c} := \sigma(w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n)$ is parameterized by the weights w_i, and the goal of SGDB is to find good weights dynamically as the solver roams through the search space. Initially the weights are set to zero since we assume no prior knowledge.

To train these weights, SGDB needs to generate training data of the form $PA \times \{1, 0\}$ where 1 signifies the conflict class. We leverage the existing conflict analysis procedure in the CDCL algorithm to create this data. Whenever the solver performs conflict analysis, SGDB creates a partial assignment PA_1 by concatenating the literals on the conflict side of conflict analysis with the negation of the literals in the learnt clause and gives this partial assignment the label 1. Clearly applying BCP to PA_1 with the current clause database leads to a conflict, hence it is assigned to the con-

Metric	Status	SGDB + No Ext	SGDB + Reason Ext	SGDB + Sparse Ext	SGDB + Both Ext	VSIDS	LRB
Mean GLR	SAT	0.324501	0.333763	0.349940	0.357161	0.343401	0.375181
	UNSAT	0.515593	0.518362	0.542679	0.545567	0.527546	0.557765
	BOTH	0.403302	0.409887	0.429420	0.434854	0.419337	0.450473
Mean Avg LBD	SAT	22.553479	20.625091	19.470764	19.242937	28.833872	16.930723
	UNSAT	17.571518	16.896552	16.249930	15.832730	22.281780	13.574527
	BOTH	20.336537	18.965914	18.037512	17.725416	25.918232	15.437237

Table 2: GLR and average LBD of various configurations of SGDB, VSIDS, and LRB on the entire benchmark with duplicate instances removed. LRB solves the most instances and achieves the highest GLR and lowest average LBD in our experiments.

flict class. SGDB creates another partial assignment PA_0 by concatenating all the literals in the current partial assignment excluding the variables in the current decision level and excluding the variables in PA_1. Applying BCP to PA_0 does not lead to a conflict with the current clause database, because if it did, the conflict would have occurred at an earlier level. Hence PA_0 is given the label 0. In summary, SGDB creates two data points at every conflict, one for each class guaranteeing a balance between the two classes. SGDB then applies one step of stochastic gradient descent on these two data points to update the weights. Since we are training in an online fashion, the two data points are discarded after the weights are updated. To reduce the computation cost, regularization is performed lazily. Regularization, if done eagerly, updates the weights of every variable on every step of stochastic gradient descent. With lazy updates, only the weights of non-zero features are updated. As is typical with stochastic gradient descent, we gradually decrease the learning rate α over time until it reaches a fixed limit. This helps to rapidly adjust the weights at the start of the search.

When it comes time to pick a new decision variable, SGDB uses the $\tilde{c}$ function to predict the decision variable that maximizes the probability of creating a partial assignment in class 1, the conflict class. Since we are using logistic regression, and the features are either 0 or 1, we can simply branch on the unassigned variable with the highest weight. By storing the weights in a max priority queue, the variable with the highest weight can be retrieved in time logarithmic to the number of unassigned variables, a big improvement over linear time.

Sparse Non-Conflict Extension: Applying stochastic gradient descent takes time proportional to $|PA_1|$ and $|PA_0|$. Unfortunately in practice, $|PA_0|$ is often quite large, about 75 times the size of $|PA_1|$ in our experiments. To shrink the size of PA_0, we introduce the sparse non-conflict extension. With this extension PA_0 is constructed by randomly sampling one assigned literal for each decision level less than the current decision level. Then the literals in PA_1 are removed from PA_0 as usual. This construction bounds the size of PA_0 to be less than the number of decision levels.

Reason-Side Extension: SGDB constructs the partial assignment PA_1 by concatenating the literals in the conflict side and the learnt clause. Although PA_1 is sufficient for causing the conflict, the literals on the reason side are the reason why PA_1 literals are set in the first place. Inspired by the LRB branching heuristic with a similar extension, the reason-side extension takes the literals on the reason side adjacent to the learnt clause in the implication graph and adds them to PA_1. This lets the learning algorithm associate these variables with the conflict class.

5.1 Experimental Results

We ran MapleSAT configured with 6 different branching heuristics (LRB, VSIDS, SGDB with four combinations of the two extensions) on all the application and hard combinatorial instances from SAT Competitions 2011, 2013, 2014, and 2016. At the end of each run, we recorded the elapsed time, the GLR at termination, and the average LBD of all clauses learnt from start to finish. Figure 2 show the effectiveness of each branching heuristic in solving the instances in the benchmark. The reason-side extension (resp. sparse non-conflict extension) increases the number of solved instances by 97 (resp. 155). The two extensions together increase the number of solved instances by 219, and in total solve just 12 instances fewer than VSIDS. LRB solves 93 more instances than VSIDS. Table 2 shows the GLR and the average LBD achieved by the branching heuristics. Both extensions individually increased the GLR and decreased the LBD. The extensions combined increased the GLR and decreased the LBD even further. The best performing heuristic, LRB, achieves the highest GLR and lowest LBD in this experiment. It should not be surprising that LRB has high GLR, our goal when designing LRB was to generate lots of conflicts by branching on variables likely to cause conflicts. By design, LRB tries to achieve high GLR albeit indirectly by branching on variables with high learning rate.

6 Related Work

The VSIDS branching heuristic, currently the most widely implemented branching heuristic in CDCL solvers, was introduced by the authors of the Chaff solver in 2001 [Moskewicz *et al.*, 2001] and later improved by the authors of the MiniSat solver in 2003 [Eén and Sörensson, 2004]. Lagoudakis and Littman introduced a new branching heuristic in 2001 that dynamically switches between 7 different branching heuristics using reinforcement learning to guide the choice [Lagoudakis and Littman, 2001]. In 2015, Biere and Fröhlich introduced a new heuristic called ACIDS [Biere and Fröhlich, 2015] that is shown to be as effective as VSIDS. Additionally, they show that the VMTF heuristic if implemented carefully is competitive with VSIDS. Liang et al. introduced two branching

heuristics, CHB and LRB, in 2016 where a stateless reinforcement learning algorithm selects the branching variables themselves. CHB does not view branching as an optimization problem, whereas LRB, GGB, SGDB do. As stated earlier, LRB optimizes for learning rate, a metric defined with respect to variables. GGB and SGDB optimize for global learning rate, a metric defined with respect to the solver.

7 Conclusion

Finding the optimal branching sequence is nigh impossible, but we show that using the simple framework of optimizing GLR has merit. The crux of the question since the success of our LRB heuristic is whether solving the learning rate optimization problem is indeed a good way of designing branching heuristics. A second question is whether machine learning algorithms are the way to go forward. We answer both questions via a thorough analysis of 7 different notable branching heuristics, wherein we provide strong empirical evidence that better branching heuristics correlate with higher GLR. Further, we show that higher GLR correlates with lower LBD, implying better clause learning. Additionally, we designed a greedy branching heuristic to maximize GLR and showed that it outperformed VSIDS, one of the most competitive branching heuristics. To answer the second question, we designed the SGDB that is competitive vis-a-vis VSIDS. With the success of LRB and SGDB, we are more confident than ever before in the wisdom of using machine learning techniques as a basis for branching heuristics in SAT solvers.

References

[Audemard and Simon, 2009] Gilles Audemard and Laurent Simon. Predicting Learnt Clauses Quality in Modern SAT Solvers. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, IJCAI'09, pages 399–404, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[Biere and Fröhlich, 2015] Armin Biere and Andreas Fröhlich. Evaluating CDCL Variable Scoring Schemes. In Marijn Heule and Sean Weaver, editors, *Theory and Applications of Satisfiability Testing – SAT 2015*, pages 405–422, Cham, 2015. Springer International Publishing.

[Bottou, 1998] Léon Bottou. On-line Learning in Neural Networks. chapter On-line Learning and Stochastic Approximations, pages 9–42. Cambridge University Press, New York, NY, USA, 1998.

[Cox, 1958] D. R. Cox. The Regression Analysis of Binary Sequences. *Journal of the Royal Statistical Society. Series B (Methodological)*, 20(2):215–242, 1958.

[Eén and Sörensson, 2004] Niklas Eén and Niklas Sörensson. *Theory and Applications of Satisfiability Testing: 6th International Conference, SAT 2003, Santa Margherita Ligure, Italy, May 5-8, 2003, Selected Revised Papers*, chapter An Extensible SAT-solver, pages 502–518. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[Goldberg and Novikov, 2007] Eugene Goldberg and Yakov Novikov. BerkMin: A Fast and Robust Sat-solver. *Discrete Appl. Math.*, 155(12):1549–1561, June 2007.

[Jeroslow and Wang, 1990] Robert G. Jeroslow and Jinchang Wang. Solving Propositional Satisfiability Problems. *Annals of Mathematics and Artificial Intelligence*, 1(1-4):167–187, September 1990.

[Katebi *et al.*, 2011] Hadi Katebi, Karem A. Sakallah, and João P. Marques-Silva. Empirical Study of the Anatomy of Modern Sat Solvers. In *Proceedings of the 14th International Conference on Theory and Application of Satisfiability Testing*, SAT'11, pages 343–356, Berlin, Heidelberg, 2011. Springer-Verlag.

[Lagoudakis and Littman, 2001] Michail G Lagoudakis and Michael L Littman. Learning to Select Branching Rules in the DPLL Procedure for Satisfiability. *Electronic Notes in Discrete Mathematics*, 9:344–359, 2001.

[Liang *et al.*, 2016a] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Exponential Recency Weighted Average Branching Heuristic for SAT Solvers. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 3434–3440. AAAI Press, 2016.

[Liang *et al.*, 2016b] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Learning Rate Based Branching Heuristic for SAT Solvers. In *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, pages 123–140, 2016.

[Marques-Silva and Sakallah, 1996] João P Marques-Silva and Karem A. Sakallah. GRASP-A New Search Algorithm for Satisfiability. In *Proceedings of the 1996 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '96, pages 220–227, Washington, DC, USA, 1996. IEEE Computer Society.

[Marques-Silva, 1999] João P Marques-Silva. The Impact of Branching Heuristics in Propositional Satisfiability Algorithms. In *Proceedings of the 9th Portuguese Conference on Artificial Intelligence: Progress in Artificial Intelligence*, EPIA '99, pages 62–74, London, UK, UK, 1999. Springer-Verlag.

[Moskewicz *et al.*, 2001] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Annual Design Automation Conference*, DAC '01, pages 530–535, New York, NY, USA, 2001. ACM.

[Stump *et al.*, 2014] Aaron Stump, Geoff Sutcliffe, and Cesare Tinelli. *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*, chapter StarExec: A Cross-Community Infrastructure for Logic Solving, pages 367–373. Springer International Publishing, Cham, 2014.