Reusing the Assignment Trail in CDCL Solvers: System description

Article · November 2011

DOI: 10.3233/SAT190082

CITATION
1

READS
96

3 authors, including:

Marijn Heule
University of Texas at Austin
64 PUBLICATIONS **2,669** CITATIONS

SEE PROFILE

Journal on Satisfiability, Boolean Modeling and Computation 7 (2011) 133-138

Reusing the Assignment Trail in CDCL Solvers

SYSTEM DESCRIPTION

Peter van der Tak peter@vdtak.eu
Antonio Ramos ramosmaurer@hotmail.com
Department of Software Technology
Delft University of Technology, Delft
The Netherlands

Marijn Heule* marijn@heule.nl
Institute for Formal Models and Verification
Johannes Kepler University, Linz
Austria

Abstract

We present the solver `RestartSAT` which includes a novel technique to reduce the cost to perform a restart in CDCL SAT solvers. This technique, called ReusedTrail, exploits the observation that CDCL solvers often reassign the same variables to the same truth values after a restart. It computes a partial restart level for which it is guaranteed that all variables below this level will be reassigned after a full restart. `RestartSAT`, an extended version of `MiniSAT`, incorporates ReusedTrail – which can be implemented easily in almost any CDCL solver. On average, it saves over a third of the decisions and propagations necessary to solve a problem using a Luby restart policy with unit run 1. Experimental results show that `RestartSAT` solves over a dozen more application instances than the default `MiniSAT`.

Keywords: *CDCL solver, restart strategy*

Submitted April 2011; revised September 2011; published November 2011

1. Introduction

Restart strategies [2, 4, 9] have been a crucial feature in conflict-driven clause learning (CDCL) solvers [5] to tackle hard industrial problems. CDCL solvers select decision variables based on their involvement in emerged conflicts [6]. In case of frequent restarts [3, 8], only a few new conflicts have been hit between two succeeding restarts. As a consequence, CDCL solvers tend to select the same variables in a similar order after the restart. Additionally, phase-saving [7] ensures that decision variables are assigned to the same truth value as the value they were assigned to before a restart. Due to these heuristics, CDCL solvers effectively perform a *partial restart* with the computational cost of a full restart.

In this paper we present `RestartSAT`, a solver based on `MiniSAT` 2.2 [1] that implements a novel technique which reduces the computational costs to perform a restart. When the solver wants to restart, it determines the first level at which the heuristics may select a different decision variable, after which it performs a partial restart by backtracking to this level, rather than perform a more costly full restart. Experiments show that `RestartSAT` solves more real-world SAT instances from the SAT 2009 application suite than `MiniSAT` 2.2.

* Supported by the Austrian Science Foundation (FWF) NFN Grant S11408-N23 (RiSE)

2. Reused Trail

In this section we present the ReusedTrail technique which is used to reduce the cost of a restart in a CDCL solver. It exploits the observation that similar assignments are made after succeeding restarts when restarting frequently.

First a short overview on how decisions are made in CDCL solvers. As long as there are unit clauses in the formula, their remaining literals are forced to true. We refer to variables that are forced to a certain value as *implied variables*. Assigning implied variables results in two possible outcomes: 1) there are no unit clauses; 2) a conflict arises (i.e. all literals of a clause are assigned to false). In case 1) the solver selects a *decision variable* using a *variable selection heuristic* and assigns it to a truth value based on a *value selection heuristic*. The most common variable selection heuristic in CDCL solvers is VSIDS [6] which selects variables based on their involvement in recent conflicts. Most CDCL solvers use phase-saving [7] to select the value. This heuristic assigns variables to their last forced value. In case 2) the conflict is analyzed and a conflict clause is added to the formula. Afterwards, the VSIDS heuristic is updated and variables are unassigned until the conflict clause becomes unit. This process continues until either a satisfying assignment is found or it learns the empty clause – a clause with no literals showing that there are no solutions.

A powerful technique in CDCL solvers is restarting [2]. This technique simply unassigns all variables and continues the process above. CDCL solvers tend to restart more and more frequently in recent years [8]. By combining VSIDS and phase-saving, the same variables tend to be assigned to the same truth values. More detailed, variables that are assigned are put on an *assignment trail*. Each decision variable starts a new *decision level*. Whereas a normal backjump would backtrack only to the *backjump level* (BJL), a *full restart* removes all variables from the trail and effectively backtracks to level 0, the *restart level* (RSL). Yet after a restart the same assignments are often added back in a similar order.

Consider our running example in Fig. 1 showing a trail just before the solver is about to restart. Notice that if the solver unassigns all variables on the trail (full restart), the above process will perform some redundant operations: the variable with the highest VSIDS score (x_5) will be assigned first (as on the trail before restarting). The same holds for the second decision (x_2) and consequently for implied variable x_{11}. Now the next decision would be x_{13}. Since it no longer matches the trail, the so-called MatchingTrail level (MTL) is two [8]. Consider the four variables with the highest scores (x_5, x_2, x_{13}, x_9) and observe that they are also the first four decision variables. The last level at which the n highest scoring decision variables are the n first decisions is called the PermutedTrail level (PTL) [8]. Performing a partial restart to MTL or PTL instead of performing a full restart reduces the number of redundant operations.

The main observation that resulted in the development of `RestartSAT` is that even some assignments after the PermutedTrail level are redundant. Let x_{next} be the unassigned variable with the highest activity – in other words the next decision variable if no restart would be performed. The key insight is that any assignment that is made before x_{next} after a restart must also have been assigned before the restart. The ReusedTrail level (RTL) is the highest level up to which all decision variables score higher than x_{next} in the VSIDS order. This means that each of these would be part of the trail after a full restart.

trail	x_5^*=1	x_2^*=0	x_{11}=0	x_9^*=0	x_{13}^*=1	x_4=0	x_{14}^*=1	x_8=1	x_1^*=1	x_6^*=1	x_3=1
level	1	2	2	3	4	4	5	5	6	7	7
score	93.5	88.2	15.9	75.4	81.2	44.0	62.8	27.7	53.6	38.1	63.9
RSL			MTL			PTL			RTL		BJL

unassigned	x_7	x_{10}	x_{12}
score	51.9	38.6	44.1

next decision at the backjump level (BJL): $x_7 = 0$ with VSIDS score 51.9

Figure 1. An example to illustrate the (partial) restart levels: restart level (RSL), MatchingTrail level (MTL), PermutedTrail level (PTL), ReusedTrail level (RTL) and the backjump level (BJL). Variables x_i^* are decision variables. The others are implied variables.

Back to the example. The first six decision variables have a higher VSIDS score than $x_{\text{next}} = x_7$. Therefore they will be reassigned before x_7 after a full restart. Reusing the trail by performing a partial restart to level 6 avoids redundantly assigning these variables to the same truth values. Note that when x_7 will be assigned – that is, after performing a partial restart and reassigning $x_3 = 1$ – the trail contains the same assignments as it would when x_7 is assigned after a full restart, albeit in a different order.

Observe that the (partial) restart levels are clearly ordered, which is also shown in the example: $0 = \text{RSL} \leq \text{MTL} \leq \text{PTL} \leq \text{RTL} \leq \text{BJL}$. Fig. 2 shows the number of times each level was found as a value for MTL, PTL, RTL, and BJL when performing a restart. MTL is often close to RSL (level 0), as indicated by the spikes at the left side of the graph. In general PTL and RTL perform much better than MTL, and especially RTL is often close to BJL, which suggests that most of the work that is performed after a restart is redundant.

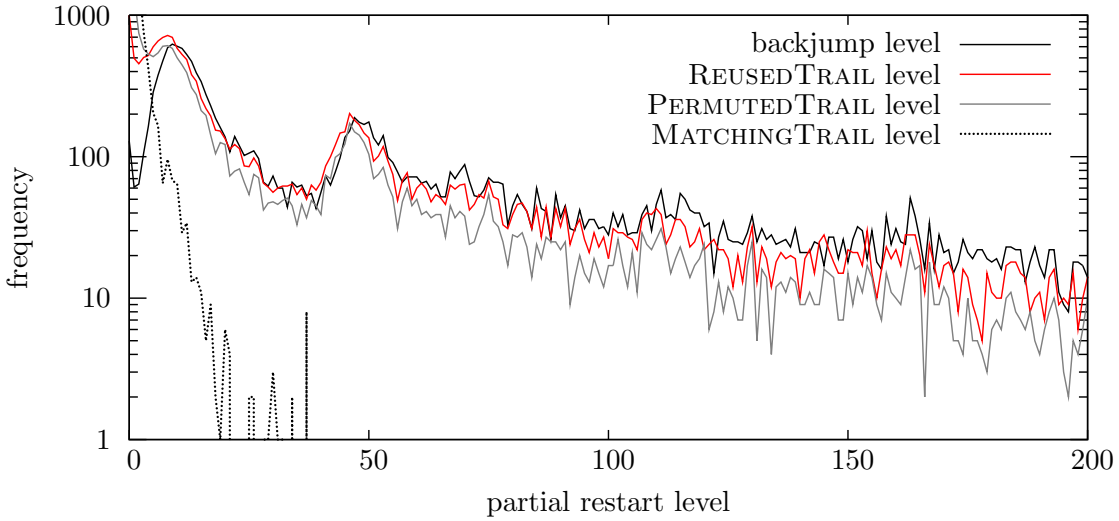

Figure 2. Histogram of the MatchingTrail, PermutedTrail, ReusedTrail and backjump levels while solving `ACG-15-10p1.cnf` of the SAT 2009 competition with Luby-1.

```
ReusedTrail (OrderHeap, Activity, DecisionVar)
1   forever do
2       if OrderHeap.empty() then return BackjumpLevel
3       v ← OrderHeap.remove()
4       if AssignmentType[x_next] = Unassigned then break
5   OrderHeap.insert(x_next)
6   for i ← 1 to BackjumpLevel
7       if Activity[DecisionVar[i]] < Activity[x_next] then return i − 1
8   return BackjumpLevel
```

Figure 3. Pseudo-code of the ReusedTrail algorithm.

ReusedTrail not only returns a higher partial restart level compared to MatchingTrail and PermutedTrail, it is also cheaper to compute. While both other algorithms require to loop over decision variables as well as implied variables [8], computing ReusedTrail only requires a loop over the decision variables (and only over those that are saved). Since the number of implied variables is in general an order of magnitude larger than the number of decision variables, this results in a big speedup. In practice, ReusedTrail is efficient, and profiling suggests that it takes less than 1% of the total solving time.

The pseudocode for ReusedTrail is listed in Fig. 3. It requires efficient access to the heap that stores the VSIDS order (OrderHeap) the VSIDS score of each variable (Activity$[x]$), and the sequence of decision variables (DecisionVar$[i]$) where x is a variable and i is the index of a decision level. In lines 1 through 5, it determines x_{next}, and reinserts it into the heap. Reinsertion is required so that all unassigned variables remain in the heap. Additionally, `MiniSAT` occasionally tries to perform a restart even when a satisfying assignment is found, resulting in an empty heap. In that case, we do not need to restart and return BackjumpLevel. Lines 6 and 7 iterate through the decision levels as described above, returning the highest level for which the decision variable has an activity of at least that of x_{next}. If all levels satisfy this criterion line 8 returns BackjumpLevel.

Although the reduction of the restart costs is the most important feature of ReusedTrail, this technique also has an interesting side-effect. In CDCL solvers, each implied variable has a *reason clause* that denotes the assignments that forced it. These influence the construction of conflict clauses. A partial restart does not affect reason clauses, whereas performing a full restart may change them. This could influence the path the solver takes through the search space either positively or negatively. A thorough analysis of the changed reason clauses requires more work, but the effect seems small. We forced reason clauses to be the same for the solved instances using a Luby-1 restart sequence, and ReusedTrail achieved a speedup of about 14% — similar to the result without forcing reason clauses. We therefore conclude that this side-effect likely does not impact the performance significantly.

3. Experimental Results

In this section we analyze the performance of `RestartSAT`[1] and the ReusedTrail technique. We compare the performance of `RestartSAT` to `MiniSAT` 2.2 [1], on which our solver

1. Available from http://www.st.ewi.tudelft.nl/sat/download.php

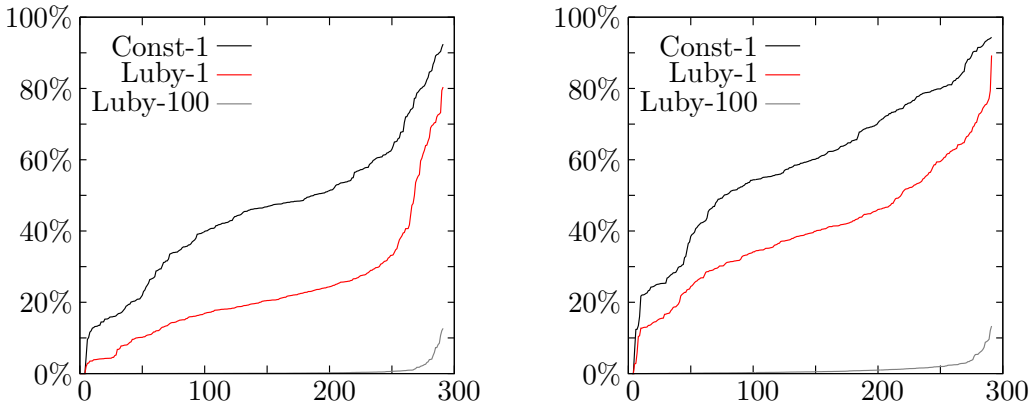

Figure 4. Fraction of saved propagations (left) and decisions (right) for three restart strategies.

was based. Our experiments were run on the (292) application instances of the SAT 2009 competition[2]. Each instance was run with a time limit of 1200 seconds using a server of 20 Intel Xeon X5570 CPUs running on 2.9 GHz with 32 GB of memory. We consider three restart strategies in our experiments. Since ReusedTrail aims to reduce restart cost, we use a radical strategy that restarts before every decision (Const-1) and a Luby sequence [4] with unit run 1 (Luby-1). We also consider Luby-100, because it is used by `MiniSAT`.

To illustrate the effectiveness of ReusedTrail, we determine the fraction of decisions and propagations that were saved (reused) by using ReusedTrail.[3] Fig. 4 shows the results for all instances sorted by this fraction (sorted separately for each of the restart strategies). Observe that ReusedTrail often significantly reduces redundant work for Luby-1 (and Const-1), by reusing over 40% of the decisions and 20% of the propagations. Unlike the differences in the two graphs, their average fractions – the average fraction of each instance weighted by the total number of propagations or decisions – are almost equal; 60% for Const-1, 35% for Luby-1, and 1% for Luby-100. The fraction is higher for larger instances, because the weighted averages are larger than the fractions of most instances.

Next, we compare the CPU time needed to solve each instance between `MiniSAT` and `RestartSAT`. Fig. 5 shows the run time for `MiniSAT` and `RestartSAT` for both restart strategies. In general, the solving time in `RestartSAT` has improved over `MiniSAT`, however for some instances RestartSAT performs worse. For these benchmarks the performance greatly depends on the seed.[4]

For Luby-100, the improvement in run time is less noticeable because restarts happen infrequently and the VSIDS order changes dramatically between restarts. However, our experiments show that Luby-1 performs better than Luby-100 in `RestartSAT`: Luby-1 solves 174 instances (168 in `MiniSAT`) and Luby-100 solves 172 (170 in `MiniSAT`). Const-1 benefits most from ReusedTrail and solves 163 instances (147 in `MiniSAT`), but is not yet competitive.

2. Available from http://www.satcompetition.org/
3. These can be computed by summing the number of decisions up to the ReusedTrail level over all iterations and dividing them by the total number of reused and performed decisions and propagations.
4. `MiniSAT` has the option to randomly initialize the VSIDS scores. For many benchmarks the seed used for the initialization has a huge impact on the performance.

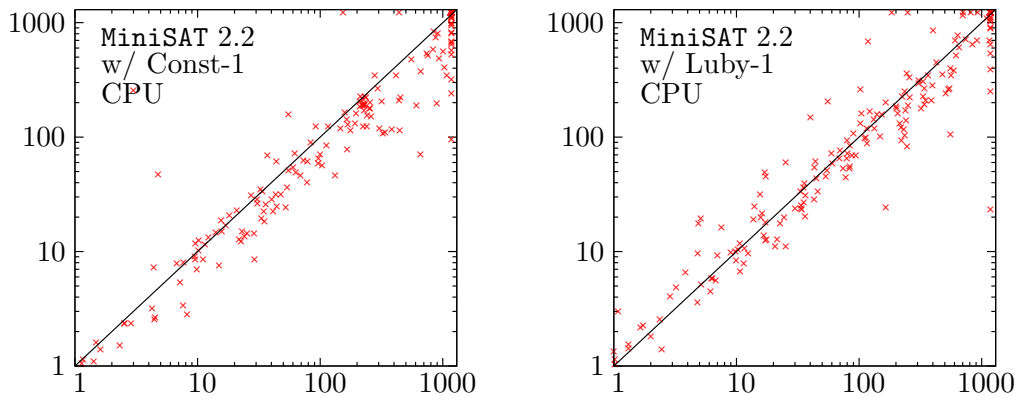

Figure 5. Solving time (s) comparison between `MiniSAT` and `RestartSAT` for the SAT 2009 application benchmarks. `RestartSAT` is faster below the diagonal.

A lower VSIDS decay value δ further improves the results. Notice that this will make ReusedTrail itself somewhat less effective because the VSIDS order will change faster. Yet, $\delta = 0.75$ improves performances with frequent restarts [8]: Const-1, Luby-1, and Luby-100 solve 169, 185, and 176 instances respectively. `RestartSAT` is thus able to solve 15 more instances than `MiniSAT` within the same time limit.

References

[1] N. Eén and N. Sörensson. An extensible SAT-solver. In *SAT'03*, pages 502–518, 2003.

[2] C.P. Gomes, B. Selman, and N. Crato. Heavy-tailed distributions in combinatorial search. In *CP'97*, pages 121–135, 1997.

[3] S. Haim and M.J.H. Heule. Towards ultra rapid restarts, 2010. Technical report, UNSW and TU Delft.

[4] M. Luby, A. Sinclair, and D. Zuckerman. Optimal speedup of las vegas algorithms. In *ISTCS*, pages 128–133, 1993.

[5] J.P. Marques-Silva, I. Lynce, and S. Malik. *Conflict-Driven Clause Learning SAT Solvers*, *FAIA* **185**, chapter 4, pages 131–153. IOS Press, February 2009.

[6] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: engineering an efficient sat solver. In *DAC '01*, pages 530–535, 2001.

[7] K. Pipatsrisawat and A. Darwiche. A lightweight component caching scheme for satisfiability solvers. In *SAT'07*, pages 294–299, 2007.

[8] A. Ramos, P. van der Tak, and M.J.H. Heule. Between restarts and backjumps. *Lecture Notes in Computer Science* **6695**, pages 216–229. Springer, 2011.

[9] T. Walsh. Search in a small world. In *IJCAI 99*, pages 1172–1177, 1999.