Adaptive Restart Strategies for Conflict Driven SAT Solvers

Armin Biere

Johannes Kepler University, Linz, Austria

Abstract. As the SAT competition has shown, frequent restarts improve the speed of SAT solvers tremendously, particularly on satisfiable industrial instances. This paper presents a novel adaptive technique that measures the agility of the search process dynamically, which in turn is used to control the restart frequency. Experiments demonstrate, that this new dynamic restart strategy improves speed of our SAT solver PicoSAT on crafted instances considerably and on industrial instances slightly.

1 Introduction

SAT solvers may benefit from restarts [3]. Particularly on satisfiable industrial examples frequent restarts improved the performance of our SAT solver PicoSAT [1] tremendously. Even though PicoSAT is a winner of the SAT competition 2007 in the category of satisfiable industrial instances, an analysis of PicoSAT's performance on unsatisfiable instances in general and on crafted instances in particular reveals, that frequent restarts can also be harmful.

In this short paper we address this issue and present a novel adaptive technique that measures the "agility" of the SAT solver as it traverses the search space, based on the rate of recently flipped assignments. The level of agility dynamically determines the restart frequency. Low agility enforces frequent restarts, high agility prohibits restarts. Our experiments demonstrate, that this new dynamic restart strategy improves the speed of PicoSAT on crafted instances considerably and on industrial instances slightly.

As has been argued in [3] combinatorial search has heavy-tail behavior. Even if an instance is easy to satisfy (or to refute), the search may get stuck in a complex part of the search space. As a solution to this problem, the authors suggest to use randomization, and in particular *restarts*. To *restart* means to stop the current search after a certain time has passed and start over again.

Our focus is on industrial and crafted instances. For random benchmarks randomized algorithms are more successful. There has been work on dynamic restart algorithms for randomized search, see for instance [4,6]. This work is not applicable to our setting. We want to improve the performance of conflict driven SAT solvers with learning, such as RSAT [9] and PicoSAT [1]. Additionally these solvers always pick the last assignment for a decision variable. Enforcing these heuristics without learning makes restarts useless. Furthermore, statistics, such as the number of satisfied clauses, which are crucial in adaptive restart scheduling for local search [4], are not available in the solvers we want to improve.

Techniques, as implemented in the SAT solver TiniSAT [5] inspired by [7] and further improved in RSAT [9] and PicoSAT [1], show, that frequent restarts in combination with saving and reusing the previous phase can speed up SAT solvers on industrial instances tremendously, particularly on satisfiable ones. In this category PicoSAT was a clear winner of the SAT'07 Competition.

Beside fast low level data structures [1], the major improvement in version 535 of PicoSAT as submitted to the SAT'07 Competition, is an aggressive restart schedule in combination with saving and reusing phases of assigned variables: The first restart occurs after 100 conflicts. Then this restart interval is increased by 10%, which means the next restart happens after another 110 conflicts, then after another 121 conflicts etc. However, this sequence of longer and longer inner restart intervals is reset to its initial value of 100 conflicts after the end of an outer restart interval is reached. Then the outer restart interval is also increased by 10%. This results in "bursts" of restarts. The restart frequency in one burst sequence slows down at the end and its length, the burst duration, slowly increases over time. More details can be found in [1].

RSAT [9] follows TiniSAT [5] with respect to restarts. Both have a less aggressive restart strategy than PicoSAT. They also do use the same kind of preprocessing [2] as MiniSAT. As a result RSAT, TiniSAT and MiniSAT turned out to be faster than PicoSAT on unsatisfiable industrial instances. On unsatisfiable *crafted* instances the situation is even worse. PicoSAT and in this case also RSAT can solve far less benchmarks than MiniSAT.

After this analysis it seems a valid conjecture, that frequent restarts may also be harmful, particularly on unsatisfiable crafted instances. The question then is, how to measure the effectiveness of frequent restarts, or better, to determine criteria, when to disable restarts.

2 Measuring Agility

In all our recent SAT solvers we monitor the average decision height and print it as a kind of progress report. The average decision height is calculated by summing up the decision levels at decision points and dividing the result by the number of decisions. If the average decision height is going up, we are "close" to a satisfying assignment. If the average decision height goes down[1], the solver will eventually resolve the empty clause, or at least some new unit clauses, and its getting "closer" to a refutation. Intuitively the solver is stuck if the average height is not changing much, and it may be a good idea to restart. On the other hand restarts should not happen if the average decision height is changing fast.

Our first *failed* attempt to dynamically control restarts was based on this observation. Restarts are disabled if the derivative of the average decision height becomes small. However, we were not able to get any positive results. In particularly, it seems to be impossible to come up with good "magic constants". The absolute values of the derivative of the average decision height varies considerably from instance to instance.

[1] This only applies to a conflict driven SAT solver with learning.

2.1 Flips

As pioneered by RSAT [9], PicoSAT always picks the last phase resp. direction to which a variable was assigned when assigning a decision variable. For instance if a decision variable was assigned to *true*, the last time it was assigned, then again it is assigned to *true*. If a variable is picked as decision variable and was not assigned before, then the phase is picked depending on the number of positive resp. negative occurrences.

Therefore, whenever a variable becomes assigned to a certain value, in particular if the assignment is forced by some other decision, PicoSAT and RSAT have to remember this value. During backtracking the variable is unassigned again, but the old value is *saved*.

This apparatus easily allows to determine when a new forced[2] assignment to a variable *flips* the old value of the variable. Flipping the value of a variable means, that it is assigned to the opposite value, as it was assigned the last time.

Clearly, if the frequency of flips is small, then the SAT solver literally does not move much, using for instance hamming distance in the boolean space as metric. This may be a good time to restart. On the other hand if many flips have occurred recently then there is no point in restarting, it may be even counterproductive.

2.2 A Fresh Look at VSIDS

In order to obtain a robust metric for measuring agility, we follow a reformulation of the seminal work on VSIDS [8]. The basic idea of VSIDS is to concentrate on those variables that recently were involved in conflicts: a variable v is *involved* in a conflict, if v is resolved in the conflict analysis to produce the learned clause or is contained in the learned clause.

Every variable has a counter, called the VSIDS score, which counts how often this variable was used in deriving a learned clause. This counter essentially sums up all these involvements. However, and this is the intriguing idea of VSIDS, it is much better to slowly forget past involvement. Variables with higher VSIDS score are picked as decision variable, which increases the focus of the search. Explaining the effectiveness of VSIDS is out of the scope of this paper.

One way to implement this scheme, is to multiply the VSIDS counters of all variables *not* involved in the current conflict by a constant factor[3] $0 < f < 1$, but not change the counters of involved variables. However, this does not quite work, because the counters will never increase. The solution is to first punish *all* variables by multiplying their score with f, including variables involved in the conflict, and only then additionally increment the score of the latter by $1 - f$.

$$s, f \leq 1, \quad \text{then} \quad s' \;\leq\; s \overbrace{\cdot\, f}^{\text{decay in any case}} \underbrace{+\,1 - f}_{\text{increment if involved}} \;\leq\; f + 1 - f \;=\; 1$$

[2] An assignment for a decision variable will always use the old value according to the direction resp. phase saving and reusing heuristics.

[3] MiniSAT, RSAT: $f = 95\% \approx 1/1.05$, PicoSAT: $f = 1/1.1 \approx 91\%$.

This reformulation of VSIDS [8] has the benefit that it produces a rational number between 0 and 1, and can be interpreted as the percentage of the number of times a variable was involved in a conflict "recently". Unfortunately we do not have a more precise definition for "recently" at this moment.

The details are as follows. Let δ_n denote the normalized n^{th} increment of a variable v in the n^{th} conflict. It is either 0 if v is *not* involved in the n^{th} conflict, or 1 if v is involved, and we have $i_n = (1 - f) \cdot \delta_n$ for the actual increment i_n. Then the n^{th} score s_n of v after conflict n can be calculated as

$$s_n = (\dots(i_1 \cdot f + i_2) \cdot f + i_3) \cdot f \cdots) \cdot f + i_n = \sum_{k=1}^{n} i_k \cdot f^{n-k} = (1-f) \cdot \sum_{k=1}^{n} \delta_k \cdot f^{n-k}$$

which we call *normalized* VSIDS (NVSIDS).

In practice it is too costly to update the VSIDS resp. NVSIDS score of all variables at every conflict, in particular for industrial examples. In the original Chaff implementation, this overhead is avoided, by accumulating and delaying punishment: variables are only punished after 256 conflicts have passed, by multiplying their score by 0.5. Meanwhile involvements increment the score by 1.

MiniSAT 1.13 has shown that it is also possible, much more accurate, more efficient and *more effective* to just update the scores of variables involved in the conflict. The same scheme is used in PicoSAT and in the following we explain and relate this optimized score calculation to our NVSIDS.

In MiniSAT's new *exponential* VSIDS scheme (EVSIDS) variables are *not* punished, but the EVSIDS score S_n has to be interpreted as $s_n \cdot f^{-n}/(1 - f)$, where n is the number of conflicts and s_n is the NVSIDS score. The increment becomes f^n at the n^{th} conflict and with $I_k = \delta_k \cdot f^{-k}$ we get

$$s_n = (1-f) \cdot f^n \cdot \sum_{k=1}^{n} \delta_k \cdot f^{-k} = (1-f) \cdot f^n \cdot \sum_{k=1}^{n} I_k = (1-f) \cdot f^n \cdot S_n$$

As the equation shows the EVSIDS score is linearly related to NVSIDS and thus can be used instead of NVSIDS to compare activity of variables. Moreover, it can be kept up-to-date by just adding f^{-k} to the score of those variables involved in the k^{th} conflict. The EVSIDS scores of other variables, which are usually many more, do not have to be touched.

2.3 Average Number of Recently Flipped Assignments (ANRFA)

To obtain a concrete metric for the agility a we follow the same idea as our NVSIDS reformulation of VSIDS. The global variable a is initialized to zero and intuitively measures the average number of recently flipped assignments.

Whenever a variable v is forced to be assigned, a is updated. First a is multiplied by $0 < g < 1$. If the assignment is a *flip*, e.g. it assigns the opposite value as in the previous assignment to v, then we increment a by $1 - g$. Assignments of decision variables and variables not assigned before do not increment a. As discussed for NVSIDS this enforces $0 \leq a \leq 1$, if we start with $a = 0$:

$$a, g \leq 1, \quad \text{then} \quad a' \leq \overbrace{a \cdot g}^{\text{decay in any case}} \underbrace{+1 - g}_{\text{increment if flipped}} \leq g + 1 - g = 1$$

Also note that we do not need an "exponential" reformulation of EVSIDS as for VSIDS, because there is only one single global agility counter.

A value of $g = 0.9999 = 1 - 1/10000$ was effective in our experiments. Slightly different values did not change the result much (in contrast to f in VSIDS). Note, that there are orders of magnitude more assignments than conflicts in a SAT run and therefore g naturally has to be much closer to 1 than f.

We logged a over industrial and crafted benchmarks on which the old version of PicoSAT performed much worse than competitors. It turned out that in those cases, where we conjectured that restarts should be slowed down, the agility a varied between 15% and 40%. For many industrial benchmarks a was way below 20%. Therefore we picked 20% as the limit at which a scheduled inner restart is disabled. Outer restarts are only disabled if the agility reaches 25% and more. Slightly different values do not change experimental results much.

The restart schedule controls the garbage collection limit for learned clauses, as in MiniSAT. Thus the restart schedule per se should *not* change. If a scheduled restart is disabled resp. skipped the solver simply does not backtrack and continues at the same decision level.

Table 1. Number of solved instances: "adaptive = *no*" is without dynamic restart control, "adaptive = **yes**" uses the ANRFA agility a to disable backtracking. Columns *sat*, *unsat*, and *solved* denote the number of solved satisfiable instances, then the number of unsatisfiable instances, and the sum of these two numbers. Time out is only 900 seconds which matches the one used in the SAT Race'06, but is much less than the time limit in the SAT Competition'07. The three rows with AAS-RSAT, show the number of solved instances for a modified version of RSAT, which is more similar to PicoSAT. The percentages "25%" and "30%" are the two values on the limit of the ANRFA agility a. Above this limit AAS-RSAT does not backtrack if a restart is scheduled.

		SAT Race'06			SAT Competition'07					
					industrial			crafted		
	adaptive	*sat*	*unsat*	*solved*	*sat*	*unsat*	*solved*	*sat*	*unsat*	*solved*
MiniSAT 2.0	*no*	32	38	70	37	57	94	22	46	68
orig. RSAT 2.0	*no*	38	36	74	41	51	92	10	20	30
AAS-RSAT	*no*	33	33	66	45	48	93	11	21	32
AAS-RSAT 25%	**yes**	34	32	66	44	49	93	11	24	35
AAS-RSAT 30%	**yes**	36	33	69	48	48	96	12	23	35
PicoSAT 741	*no*	35	39	74	43	54	97	14	24	38
PicoSAT 741	**yes**	36	39	75	44	57	101	16	36	52

3 Experiments

We added calculating ANRFA and the adaptive restart strategy to PicoSAT and measured its effect on the SAT Race'06 instances and the SAT'07 Competition benchmarks with a time out of 900 seconds and a memory limit of 1.5 GB on Linux PCs with 3 GHz Pentium IV. As Tab. 1 shows we slightly improved on industrial examples. PicoSAT with the adaptive restart schedule can solve 36% more crafted instances. This is mainly due to the improvement on unsatisfiable instances, where 50% more instances are solved.

We also implemented the suggested adaptive technique in RSAT 2.0, the version submitted to the SAT'07 Competition. Before we changed the basic restart interval from 512 to 100 as in PicoSAT and always enforced saving and reusing phases to match PicoSAT more closely. This results in an "aggressive always saving" RSAT, called AAS-RSAT, with and without adaptive restart control. Using adaptive control for restarts in RSAT is not as impressive as for PicoSAT, but we did not spend much time to optimize magic constants either.

4 Conclusion and Future Work

We presented a new adaptive restart strategy, which slows down restarts if the agility of the SAT solver is high. The key insight is to apply the same filtering technique to the number of flipped assignments as in a new reformulation of VSIDS. For PicoSAT considerable performance improvements have been achieved. In future work we want to apply similar ideas to dynamically control the number of garbage collected clauses resp. the limit on the number of conflicts.

References

1. Biere, A.: PicoSAT essentials. Journal on Satisfiability, Boolean Modeling and Computation (submitted, 2008)
2. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, Springer, Heidelberg (2005)
3. Gomes, C., Selman, B., Kautz, H.: Boosting combinatorial search through randomization. In: Proc. AAAI 1998 (1998)
4. Hoos, H.: An adaptive noise mechanism for WalkSAT. In: Proc. AAAI 2002 (2002)
5. Huang, J.: The effect of restarts on the eff. of clause learning. In: Proc. IJCAI 2007 (2007)
6. Kautz, H., Horvitz, E., Ruan, Y., Selman, B., Gomes, C.: Dynamic restart policies. In: Proc. AAAI 2002 (2002)
7. Luby, M., Sinclair, A., Zuckerman, D.: Optimal speedup of Las Vegas algorithms. Information Processing Letters 47 (1993)
8. Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proc. DAC 2001 (2001)
9. Pipatsrisawat, K., Darwiche, A.: RSat 2.0: SAT solver description. Technical Report D–153, Automated Reasoning Group, Comp. Science Dept., UCLA (2007)