Towards Ultra Rapid Restarts*

Shai Haim[1] and Marijn Heule[2]

[1] University of New South Wales and NICTA, Sydney, Australia
[2] Delft University of Technology, Delft, The Netherlands

Abstract. We observe a trend regarding restart strategies used in SAT solvers. A few years ago, most state-of-the-art solvers restarted on average after a few thousands of backtracks. Currently, restarting after a dozen backtracks results in much better performance. The main reason for this trend is that heuristics and data structures have become more restart-friendly. We expect further continuation of this trend, so future SAT solvers will restart even more rapidly. Additionally, we present experimental results to support our observations.

1 Introduction

Restarts have been proposed for satisfiability (SAT) solvers to counter heavy-tail behavior [1]. Initially, branching heuristics were randomized to make sure that the search-tree would be different after each restart. Also, restarts should not be applied too frequently to guarantee that a solver can explore the entire search-tree between two restarts in case a problem has no solutions. For modern conflict-driven clause learning (CDCL) solvers [2] this is no longer required. Decision heuristics are dynamic and updated after every conflict [3]. By recording conflict clauses, CDCL solvers can proof unsatisfiability even in case of ultra rapid restarts.

Nowadays, restarts have become an essential feature of CDCL solvers. Many different strategies have been studied and used [4–11]. State-of-the-art SAT solvers tend to restart more and more frequently. An explanation for this trend is that the heuristics and data-structures have become restart-friendly. Therefore we decided to experiment with strategies that restart radically faster than commonly used in the CDCL solvers. The results show that these strategies are effective on the industrial benchmarks of the SAT 2009 competition.

2 Restarts

Restart strategies have been used in SAT solvers for over a decade. First, we will provide an overview of their use in state-of-the-art solvers. Second, we will discuss two aspects of CDCL solvers, heuristics and data structures, that influenced these strategies. Recent developments in these areas facilitate frequent restarts. Third, we argue that – due to rapid restarts – CDCL solvers have become complete local search solvers.

* The second author is supported by the Dutch Organization for Scientific Research (NWO) under grant 617.023.611

2.1 A history of restart strategies

Although currently all competitive CDCL solvers use restarts, this was not always the case. The solver `grasp` [12], which first introduced clause learning in the context of satisfiability testing, did not use restarts in its original version. Following the work of Gomes et al. [1], which demonstrated the effectiveness of restart for addressing issues arising from the heavy-tailed distribution, developers started equipping their solvers with fixed-size restart strategies. The solvers `zChaff` [3], `BerkMin` [13] use rather frequent fixed restarts with restart sizes of 700 and 550 respectively, while the solver `Siege` [14] uses a larger fixed restart size of 16,000 conflicts.

`MiniSAT` 1.13 [15] was the first to demonstrate the effectiveness of the geometric restart strategy suggested by Walsh [5]. Starting with a small first restart, the size of consecutive restarts grows geometrically. A commonly used restart strategy in the recent years is based on a sequence of restart sizes suggested by Luby et al. [4]. In their work the authors show that the suggested sequence is log optimal when the runtime distribution of the problems is unknown. In this strategy the length of restart i is $u \cdot t_i$ when u is a constant unit run and

$$t_i = \begin{cases} 2^{k-1}, & \text{if } i = 2^k - 1 \\ t_{i-2^{k-1}+1}, & \text{if } 2^{k-1} \leq i < 2^k - 1. \end{cases}$$

Since unit runs are commonly short, solvers using the Luby restart strategy exhibit frequent restarts. The solvers `Rsat` 2.0 [16] and `TiniSat` [17] use a unit run of 512 conflicts, while `MiniSAT` 2.1 [18] and `precoSAT` [8] use a shorter unit run of 100 conflicts. The solver `picoSAT` [7] introduced a frequent restart strategy in which the restart length grows geometrically until it reaches a bound. At this point the restart sequence starts again and the bound grows geometrically. Another approach, which receives much attention lately, combines an underlying uniform restart strategy with a dynamic element which can induce, or suppress, restarts. The dynamic decision can be made according to variable agility [6–8], variety of decision levels in learnt clauses and backtrack sizes [9, 10], or using local search techniques [11].

2.2 Direction heuristics

Direction heuristics select the value for decision variables. In theory, these heuristics can be very powerful: Perfect direction heuristics would result in a solver that never needs to backtrack to find a solution. If such a heuristic exists, which can be computed in polynomial time, then $\mathcal{P} = \mathcal{NP}$. CDCL solvers use a variety of direction heursics.

For instance, `zChaff` maintains two counters for each variable, one for true and one for false. These counters refer to the activity in recent conflicts. The sign of the highest counter is preferred. The direction heuristics in `MiniSAT` are very minimalistic: It uses *negative branching*: i.e. the decision variable is always assigned to false. Although it might seem a bit arbitrary, it is not. Two properties of this heuristic contribute the fast performance. First, it consequently chooses the same sign. Therefore it keeps searching in the same search space. Second, always branching on false is much better than always branching on true. The latter is an artifact of the encoding of most

benchmark instances. A direction heuristics technique called *phase-saving* was introduced in `Rsat`. Phase-saving assigns each decision variable to the value last forced by Boolean constraint propagation (BCP). In essence, this technique was already used in local search solvers. We will further discuss this in Section 2.4.

The changes in direction heuristics can hardly be separated from the trend we observed for restart strategies. Rapid restarts only make sense in case the solver will not end up in a completely different search space again and again. With the direction heuristics used in `zChaff` this could easily happen. If for a high ranked variable both counters are almost equal than that variable can be flipped frequently. While using negative branching it will happen less often. Yet as soon as a decision variable is chosen which BCP mostly assigns to true, the search space becomes different. However, phase-saving is ideal for rapid restarts since this direction heuristic ensures one hardly moves after a restart.

2.3 Boolean constraint propagation

Most of the computational cost of CDCL solvers is spent on BCP. Moskewicz et al. [3] state that in most cases it is greater than 90% of the total cost. This observation has consequences for rapid restart strategies: If a solver would restart very frequently, say after every couple of conflicts, then it often has to go down the search-tree all the way from the root. As a result, much more time will be spent on BCP slowing down the solver.

An important breakthrough in speeding up BCP is the introduction of the *watch literal data structure* in `zChaff` [3]. This data structure is now used in all state-of-the-art CDCL solvers. It has been implemented very efficiently in `MiniSAT` [15]. Recent improvements of this data structure were used in `picoSAT` [7]. Additionally, the relative burden of BCP can be reduced by spending more time on reasoning techniques. For example by making conflict analysis stronger. Two recent improvements in this direction are conflict clause minimization [19] and conflict clause (self-) subsumption [20].

Both developments influence the optimal restart strategy. The cheaper the relative cost of going down the search-tree, the cheaper it is to perform a restart. Therefore, it is expected that future improvements in BCP speed and additional reasoning will ensure that the optimal restart strategy will be more rapid.

2.4 Complete local search

Ten challenges for SAT solving have been posed by Selman et al. [21] in 1997. Although several of these challenges have been faced, hardly any progress has been reported on Challenge 5: Desiging a competitive complete local search solver. Appearently, it is hard to add completeness to local search solvers effectively. On the other hand, CDCL solvers have been slowly begun to mimic local search solvers. This could explain why the performance of current CDCL solvers heavily depends on the seed, even for unsatisfiable benchmarks.

An important step towards local search is the introduction of phase-saving in CDCL solvers in 2007 [16]. Essentially the same technique is used in the local search SAT solver `UnitWalk` [22] since 2001. The UnitWalk algorithm starts by initializing a

random full assignment. In each iteration, this assignment is improved by the following procedure: First, a random order of the variables is created. Second, the most important (based on this random order) free variable is assigned to the value in the full assignment. Third, BCP is applied. Each assignment due to BCP is copied to the full assignment. After BCP is finished, the procedure returns to the second step until all variables are assigned. In other words, both techniques copy the value *from* the full assignment for decision variables and copy the value *to* the full assignment for implied variables.

Due to the combination of phase-saving and rapid restarts, one can argue that CDCL solvers hardly perform search anymore. They merely improve the full assignment, while recording clauses for every encountered conflict. Therefore, modern CDCL solvers could be considered as complete local search solvers. This claim will become stronger if the trend towards ultra rapid restarts will continue.

3 Results

Because we observed several signs in favor of ultra rapid restarts, we decided to experiment with strategies that restart radically faster compared to those used in the current CDCL solvers. The dataset we have used for this experiment includes all industrial instances which were used in the SAT competition of 2009. All the experiments presented in this paper were conducted on a cluster of 14 Dual Intel Xeon CPUs with EM64T (64-bit) extensions, running at 3.2GHz with 4GB of RAM under Debian GNU/Linux 4.0.

The solver we used for the experiments is the award-winning `MiniSAT` 2.0 which we equipped with a phase-saving direction heuristic. We experimented with 12 different unit runs for the Luby sequence and used a timeout of 900 seconds. To provide more stable numbers, we ran all experiments with three different seeds.

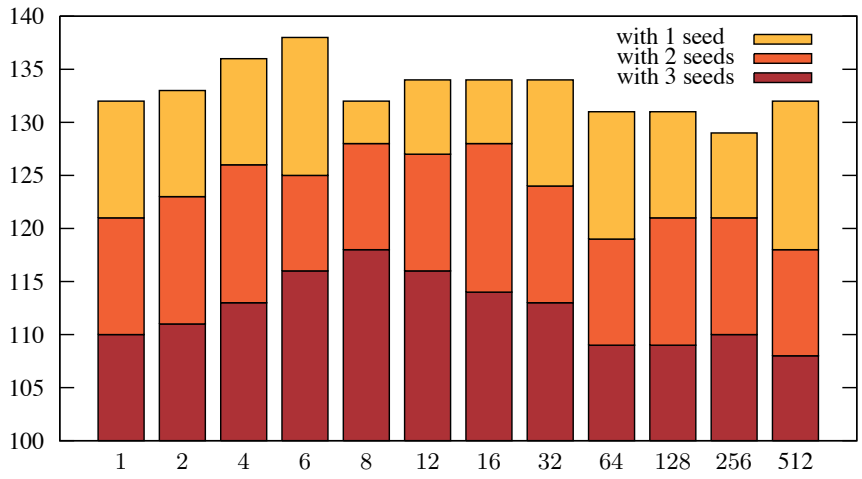

Fig. 1. Histrogram showing the number of solved instances for different unit runs of the Luby sequence. The baseline at 100 instances represents the original version of `MiniSAT` 2.0.

The original version of MiniSAT 2.0, which applies negative branching and uses a geometrical restart strategy, solves 100 instances (44 SAT, 56 UNSAT) within the timeout. Figure 1 shows the results for the adapted solver using phase-saving and Luby sequences. Notice that using any of these Luby sequences solves many more benchmark instances compared to the original version. The optimal restart strategy for this test set seems around a unit run of 6 or 8. Recall that this number is much smaller than what is commonly used in CDCL solvers.

The size of the unit run has a clear impact on the number of conflicts the solver encounters while solving a problem. Table 1 shows the average numbers. The smaller the unit run, the smaller the number of conflicts. Although the results using a unit run of 1 and 512 show a comparable performance on the dataset, the former resolves significantly fewer conflicts. Apparently, smaller unit runs require less search to solve instances. The results using other unit runs hint in this direction as well. Therefore, we expect that – assuming that the (relative) cost of performing a restart will further be reduced – even smaller unit runs will appear optimal in the future.

Table 1. The average number of conflicts for several unit runs of the Luby sequence.

Strategy	SAT	UNSAT	SOLVED	UNSOLVED	ALL
Luby-1	90465	171629	137513	309941	237799
Luby-2	79064	181351	138151	349777	260505
Luby-4	76743	188944	140772	380772	277324
Luby-6	83970	204974	153252	387720	285578
Luby-8	81043	210837	155211	401257	294354
Luby-12	91667	197671	151839	412065	299301
Luby-16	88195	205252	153884	428446	309785
Luby-32	95870	222315	167783	436521	321921
Luby-64	79550	212722	155225	452032	329556
Luby-128	93769	214950	160234	470681	341863
Luby-256	96148	222214	165134	477443	348211
Luby-512	95981	222871	164034	487855	354604

4 Conclusions

We showed that the award winning solver MiniSAT 2.0 can significantly be improved by adding phase-saving and rapid restarts. The optimal strategy on the industrial benchmarks of the SAT 2009 competition restarts far more frequently compared to strategies used by the current state-of-the-art solvers. This result supports our observation that SAT solvers tend towards ultra rapid restarts and become complete local search solvers.

References

1. Gomes, C., Selman, B., Kautz, H.A.: Boosting combinatorial search through randomization. In: AAAI/IAAI. (1998) 431–437
2. Marques-Silva, J.P., Lynce, I., Malik, S.: Chapter 4. Conflict-Driven Clause Learning SAT Solvers. In Handbook of Satisfiability. IOS Press (2009) 131–153
3. Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. Proceedings of the 38th conference on Design automation (2001) 530–535
4. Luby, M., Sinclair, A., Zuckerman, D.: Optimal speedup of Las Vegas algorithms. Inf. Process. Lett. **47**(4) (1993) 173–180
5. Walsh, T.: Search in a small world. In Dean, T., ed.: IJCAI 99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, Morgan Kaufmann (1999) 1172–1177
6. Biere, A.: Adaptive restart strategies for conflict driven SAT solvers. [23] 28–33
7. Biere, A.: PicoSAT essentials. Journal on Satisfiability, Boolean Modeling and Computation **4** (2008) 75–97
8. Biere, A.: P{re,i}coSAT@SC'09. [24] 41–44
9. Audemard, G., Simon, L.: GLUCOSE: A solver that predicts learnt clause quality. [24] 7–8
10. Pipatsrisawat, K., Darwiche, A.: Rsat description for SAT competition 2009. [24] 45–46
11. Ryvchin, V., Strichman, O.: Local restarts. [23] 271–276
12. Marques-Silva, J., Sakallah, K.A.: GRASP - a new search algorithm for satisfiability. In: ICCAD. (1996) 220–227
13. Goldberg, E., Novikov, Y.: BerkMin: a fast and robust sat-solver. (2002) 142–149
14. Ryan, L.: Efficient algorithms for clause learning SAT solvers. PhD thesis, Simon Fraser University, School of Computing Science (2004)
15. Eén, N., Sörensson, N.: An extensible SAT-solver. In Giunchiglia, E., Tacchella, A., eds.: Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. LNCS 2919, Springer (2003) 502–518
16. Pipatsrisawat, K., Darwiche, A.: A lightweight component caching scheme for satisfiability solvers. In Marques-Silva, J., Sakallah, K.A., eds.: SAT 2007: Theory and Applications of Satisfiability Testing, 10th International Conference. LNCS 4501, Springer (2007) 294–299
17. Huang, J.: A case for simple SAT solvers. In Bessiere, C., ed.: CP. LNCS 4741, Springer (2007) 839–846
18. Sörensson, N., Eén, N.: Minisat 2.1 and minisat++ 1.0 sat race 2008 editions. Technical report (2008)
19. Sörensson, N., Eén, N.: Minisat v1.13 a sat solver with conflict-clause minimization. Technical report (2005)
20. Han, H., Somenzi, F.: On-the-fly clause improvement. In: Theory and Applications of Satisfiability Testing - SAT 2009. LNCS 5584, Springer (2009) 209–222
21. Selman, B., Kautz, H., McAllester, D.: Ten challenges in propositional reasoning and search. In: IJCAI'97, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (1997) 50–54
22. Hirsch, E., Kojevnikov, A.: Solving boolean satisfiability using local search guided by unit clause elimination. Principles and Practice of Constraint Programming (2001) 605–609
23. Büning, H.K., Zhao, X., eds.: Theory and Applications of Satisfiability Testing - SAT 2008, 11th International Conference, SAT 2008, Guangzhou, China, May 12-15, 2008. Proceedings. In Büning, H.K., Zhao, X., eds.: SAT. LNCS 4996, Springer (2008)
24. SAT 2009 competitive events booklet: preliminmary version (2009)