GLUEMINISAT 2.2.10 & 2.2.10-5

Hidetomo Nabeshima
University of Yamanashi, JAPAN

Koji Iwanuma
University of Yamanashi, JAPAN

Katsumi Inoue
National Institute of Informatics, JAPAN

***Abstract*—GLUEMINISAT 2.2.10 is a SAT solver based on MINISAT 2.2 and the LBD evaluation criteria of learned clauses. A new feature of 2.2.10 is an adaptive restart strategy based on literal block size. The default restart strategy is same as GLUCOSE, that is, the solver restarts when the average of LBDs for recent learned clauses becomes worse. When the average size of literal block is small, the solver uses other strategies: the conflict generation speed based restart strategy for slow restart instances, and Luby restart strategy for fast restart instances.**

I. INTRODUCTION

GLUEMINISAT is a SAT solver based on MINISAT 2.2 [1] and the LBD evaluation criteria of learned clauses [2]. One of the feature of GLUEMINISAT is the on-the-fly lazy simplification techniques based on binary resolvents [3], which are inprocessing techniques and are executed frequently during the search process of the satisfiability checking. These techniques try to identify the truth value of variables and to detect equivalent literals, and to simplify (learned) clauses by binary self-subsuming resolution. These were introduced from 2.2.7 and some of them are refined and extended in 2.2.10.

The main feature of 2.2.10 is an adaptive restart strategy which is based on literal block size. This is introduced for solving hard instances that can not be solved by the LBD based restart strategy. The other new features are two kinds of preprocessing techniques to eliminate variables, and some minor changes for VSIDS decision heuristics [4].

II. MAIN TECHNIQUES

The main feature of 2.2.10 is *an adaptive restart strategy* based on literal block size. The default restart strategy of GLUEMINISAT is same as GLUCOSE, that is, the solver restarts when the average of LBDs for recent learned clauses becomes worse. Intuitively, the solver restarts to change the search space for finding better learned clauses. This strategy works well for most application instances. From the results of the SAT 2014 Competition, we found that this strategy is weak in the specific instances which have small literal blocks, that often occurs in cryptographic instances. A *literal block* is a set of literals which are propagated at the same decision level, and is a key concept of the LBD evaluation criteria, which evaluates a learned clause by the number of literal blocks in it. The literals in a block would be expected to be propagated at the same time. This means that even if a clause is long, it would generate a propagation from the clause when it consists of large literal blocks. In brief, the worth of LBD can be exhibited for large block sized instances.

For small block sized instances, the strategies based on LBD do not work well. Actually, when the restart speed (that is, the number of conflicts per a restart) of those small block sized instances are slow or fast, then the LBD based restart strategy did not solve them in many cases. This means that the LBD based restart strategy is too sensitive or unresponsive for such instances.

GLUEMINISAT changes the restart strategy when the average size of literal blocks after 200000 conflicts is small (≤ 1.75 literals/block),

- if the restart speed is slow (> 400 conflicts/restart), then the solver uses the conflict generation speed based restart strategy.
- if the restart speed is fast (< 125 conflicts/restart), then the solver uses Luby restart strategy that is the default restart strategy of MINISAT 2.2.

The former strategy restarts when the average number of recent decisions to generate a conflict is larger than the global ones.

In addition to the above, the version 2.2.10-5 changes the LBD evaluation criteria into the activity based one, which is used in MINISAT, for reducing learned clauses. That is 2.2.10-5 does not use the LBD criteria for small block sized instances.

III. PREPROCESSING TECHNIQUES

In 2.2.10, we have introduced two kinds of preprocessing techniques: incremental variable elimination and decision variable elimination.

Variable elimination [5] is one of effective preprocessing techniques, which eliminates a variable x if the number of resolvents with respect to x is less than the number of clauses that contains x or $\bar{x}$. *Incremental variable elimination* iteratively applies the variable elimination by relaxing the condition gradually, that is, the solver permits increasing clauses if the decreasing rate of variables is greater than the increasing rate of clauses. This technique is applied for large instances (≥ 10000 variables) and the ratio of clauses to variables is small (≤ 10).

Decision variable elimination classifies variables into non-decision variables that are not selected by decision heuristics. If each clause has at most one non-decision variable, then the value of non-decision variables are propagated by other literals in the clauses eventually. This means that the mechanism of satisfiability-checking does not need to any modification. GLUEMINISAT classifies a variable x into a non-decision variable when it satisfies the following conditions:

1) each original clause that contains x or $\bar{x}$ has no non-decision variables.

2) the cost of x is small ($\leq$ 25), where the cost is the estimated number of resolvents with respect to x (positive occurrences times negative occurrences).
3) x has the smallest cost in each clause that has x or $\bar{x}$.

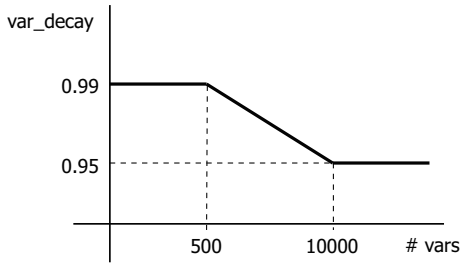
The first one is a sufficient condition for propagating truth value of non-decision variables. The second and third ones are heuristics to select infrequent and insignificance variables. Furthermore, when a learned clause has two or more non-decision variables, then the solver classifies them into decision variables except for one. This is introduced to avoid obstructing the propagation from the clause.

IV. Other techniques

GlueMiniSat uses VSIDS decision heuristics [4] same as MiniSat. The variable decaying factor of VSIDS (`var-decay` in MiniSat) controls the decrement of the activity for every variable relatively. If it is low, the activity decreases rapidly and the decision heuristics selects variables which are related to *recent* conflicts. Conversely, if it is high, the decision heuristics selects variables based on *overall* conflicts.

To explore the search space diversely in an early stage, it is a natural way to increase the decaying factor gradually. Glucose 3.0 has such a mechanism and it was implemented in GlueMiniSat 2.2.8. In 2.2.10, the parameters of this strategy are slightly changed. It starts from 0.6, and is increased up to 0.95 gradually until 100000 conflicts occur.

The final value of the variable decaying factor is 0.95 in default. When a given instance has not many variables ($<$ 10000 variables), the solver changes the final value of the factor according to the following graph. This intends to suppress the change of variable activity in small instances.



The restart blocking [6] which was introduced from 2.2.8 but is disabled in 2.2.10.

V. Availability

GlueMiniSat is developed based on MiniSat 2.2. Permissions and copyrights of GlueMiniSat are exactly the same as MiniSat. GlueMiniSat can be downloaded at http://glueminisat.nabelab.org/.

Acknowledgment

This research is supported in part by Grant-in-Aid for Scientific Research (No. 26330248) from Japan Society for the Promotion of Science and by Kayamori Foundation of Informational Science Advancement.

References

[1] N. Eén and N. Sörensson, "An extensible SAT-solver," in *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT 2003), LNCS 2919*, 2003, pp. 502–518.
[2] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern SAT solvers," in *Proceedings of IJCAI-2009*, 2009, pp. 399–404.
[3] H. Nabeshima, K. Iwanuma, and K. Inoue, "On-the-fly lazy clause simplification based on binary resolvents," in *ICTAI*. IEEE, 2013, pp. 987–995.
[4] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proceedings of the 38th Design Automation Conference (DAC 2001)*, 2001, pp. 530–535.
[5] N. Eén and A. Biere, "Effective preprocessing in SAT through variable and clause elimination," in *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT 2005), LNCS 3569*, 2005, pp. 61–75.
[6] G. Audemard and L. Simon, "Refining restarts strategies for sat and unsat," in *CP*, ser. Lecture Notes in Computer Science, M. Milano, Ed., vol. 7514. Springer, 2012, pp. 118–126.