Nigma 1.2

Chuan Jiang
Department of Computer Science
Iowa State University
Ames, Iowa, USA

Gianfranco Ciardo
Department of Computer Science
Iowa State University
Ames, Iowa, USA

***Abstract*—We describe Nigma 1.2, a CDCL SAT solver with partial backtracking. Nigma 1.2 is an improved version with stochastic local search and failed literal probing as inprocessing.**

I. Introduction

Nigma 1.1 was entered in the SAT competition 2014 [1]. Nigma 1.2 is an improved version with stochastic local search and failed literal probing as inprocessing.

II. Main Techniques

Nigma features partial backtracking [2], a conservative backtracking strategy. This technique was inspired by the observation that CDCL (Conflict Driven Clause Learning) solvers often redo many discarded variable assignments, as the popular branching heuristics [3][4] tend to keep the solver in the same search space. By backtracking to a level between the conflicting level and the assertion level (the second-highest level among the literals in the learned clause, or the top level if the learned clause contains only one literal), the solver discards as little as possible of the assignment trail and retains as many sub-solutions as possible. Then, a generalized unit propagation is invoked to assign the conflicting variable at the assertion level and amend the existing assignment trail. For implementation details of partial backtracking, please refer to [2]. Nigma 1.2 maintains two implication queues, one storing implications at the current level, the other storing implications at lower levels in the order of levels and arrivals. This helps Nigma determine when to invoke the generalized unit propagation and avoids searching for the implication with the lowest level by scanning the queue.

Compared with Nigma 1.1, Nigma 1.2 has been improved by employing stochastic local search and failed literal probing as inprocessing, which are described next.

A. Stochastic Local Search

We observed that some satisfiable instances which are hard for DPLL procedure can be easily solved by stochastic local search. Thus, we implemented the stochastic local search of [5]. For each try, a random assignment is generated (*False* is more likely to be guessed than *True*). Then, the solver selects an unsatisfied clause and flips one of its variables according to a probability distribution based on the number of clauses that become unsatisfied if that variable is flipped, until the instance is satisfied or a maximum number of flips is reached. Since Nigma uses stochastic local search as a complementary technique, limits on the time spent on stochastic local search and on the maximum number of tries are set, and local search is invoked only on small instances (with fewer than 2,000 variables after preprocessing).

B. Failed Literal Probing as Inprocessing

Nigma invokes failed literal probing [6] to simplify the boolean formula periodically during search. Variables which are not eliminated or assigned at the top level are considered in a circular manner. Each probing starts from the variable where the last probing stopped, and stops if all the variables have been probed or the limit on the maximum number of propagated literals is reached. If none or only a few failed literals are detected in one probing, the next probing will be postponed. Failed literal probing is invoked as inprocessing only if fewer than 50,000 variables remain.

III. SAT Race 2015 Specifics

We submit Nigma 1.2 to the main track of SAT Race 2015. It is compiled by GCC in 64-bit with the -O3 flag.

IV. Availability

Nigma is an open-source SAT solver under MIT license. The source code is available for download at `http://sourceforge.net/projects/nigma/`.

V. Acknowledgments

Our work was supported in part by the National Science Foundation under grants CCF-0954132 and CCF-1442586.

References

[1] C. Jiang and G. Ciardo, "Nigma 1.1," in *Proceedings of SAT Competition 2014 Solver and Benchmark Descriptions*, 2014, p. 53.

[2] C. Jiang and T. Zhang, "Partial backtracking in CDCL solvers," in *Proceedings of the 19th international conference on Logic for Programming, Artificial Intelligence, and Automated Reasoning*, 2013, pp. 490–502.

[3] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient sat solver," in *Proceedings of the 38th conference on Design Automation*, New York, New York, USA, 2001, pp. 530–535.

[4] K. Pipatsrisawat and A. Darwiche, "A lightweight component caching scheme for satisfiability solvers," in *Proceedings of the 10th international conference on Theory and Applications of Satisfiability Testing*. Springer-Verlag, 2007, pp. 294–299.

[5] A. Balint and U. Schöning, "Choosing probability distributions for stochastic local search and the role of make versus break," in *Theory and Applications of Satisfiability Testing*, 2012, pp. 16–29.

[6] I. Lynce and J. P. Marques-Silva, "Probing-based preprocessing techniques for propositional satisfiability," in *Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence*, 2003, pp. 105–110.