noSAT-MaxSATv2

Ole Lübke Sibylle Schupp
Institute for Software Systems
Hamburg University of Technology (TUHH)
Hamburg, Germany
{ole.luebke, schupp}@tuhh.de

***Abstract*—Using a SAT solver to solve MaxSAT is a well-established and efficient method. However, in resource-constrained computing environments, e.g., embedded systems, such algorithm designs can be hard to realize. With *noSAT-MaxSAT*, we explore alternatives that do not rely on an external, dedicated SAT solver, by executing an anytime local-search MaxSAT solving algorithm only on the hard clauses of the formula instead.**

In many real-world problems a significant portion of the hard clauses feature not more than two literals. Therefore, in *noSAT-MaxSATv2* we integrated a linear-time algorithm for 2-SAT as a preprocessing step to find a good initial assignment before proceeding to solve the remaining hard clauses with local search. Additionally, compared to the previous version of *noSAT-MaxSAT*, we updated the local-search algorithm from SATLike to NuWLS, the algorithm which won all incomplete tracks of the MaxSAT Evaluation 2022.

I. Introduction

In recent years the solvers in the incomplete/anytime track of the MaxSAT Evaluation (MSE) have converged to employ algorithms that rely on complete SAT solvers. A call to the SAT solver is often executed to obtain a valid initial assignment for the hard clauses in partial MaxSAT problems, or iteratively in linear search-based MaxSAT solvers [1, 2, 3, 4].

Our goal is to develop an efficient MaxSAT solver that is suitable for deployment in resource-constrained computing environments. Here, careful analysis of the resource requirements of the software, especially regarding runtime and memory, is usually required to verify the system against its specification. Each additional software module complicates this process, potentially to the point of infeasibility. Therefore, one of the key requirements for our solver is that it must not rely on a SAT solver.

noSAT-MaxSATv2 is based on the NuWLS local search algorithm, which won all incomplete tracks of the MSE 2022. It entered the competition as NuWLS-c and uses a dedicated SAT solver to solve the hard clauses in partial MaxSAT problems [5]. Instead of employing a SAT solver, *noSAT-MaxSATv2* first executes a linear-time 2-SAT algorithm on all hard clauses that have two or fewer literals. We refer to such clauses as *2-clauses*. For solving the remaining hard clauses, it uses NuWLS to solve the remaining hard clauses.

In the following, we elaborate on the employed 2-SAT solving algorithm and describe the requirements and architecture of *noSAT-MaxSAT*.

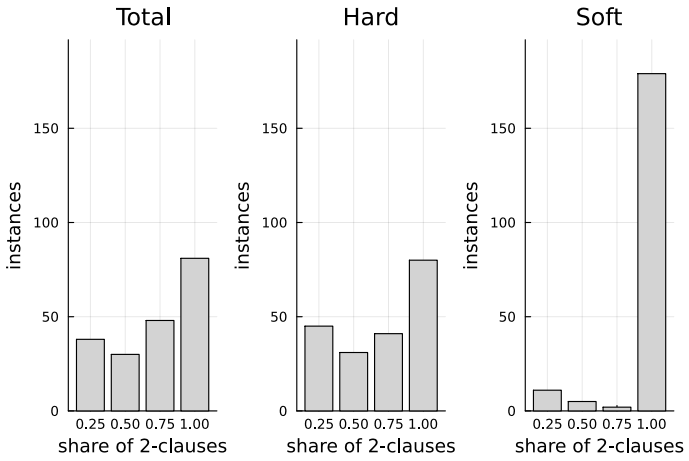

Fig. 1. Histograms showing the share of 2-clauses in the MSE 2022 benchmark instances

II. 2-SAT Solving

Many real-world MaxSAT problems have a large portion of 2-clauses. This is evident in the benchmark instances used in the MSE 2022, as shown by the histograms in Figure 1.

While MaxSAT is NP-complete even when restricted to only 2-clauses [6], 2-SAT can be solved in linear time [7, 8].

Because of its simplicity, we decided to use the linear-time algorithm by Even et al. [7] to solve hard 2-clauses. When at least 50% of the hard clauses are 2-clauses, we apply the algorithm. Otherwise, we preprocess using decimation [9] (as in the previous version of *noSAT-MaxSAT*). The resulting variable assignment is then used as the initial assignment for the local search algorithm that is employed to solve any remaining hard clauses.

III. Requirements & Architecture

noSAT-MaxSAT is developed under the following requirements, derived from common constraints found in programming embedded systems. The software

1) is programmed in C;
2) is self-contained, i.e., it has no external dependencies (other than the C standard library);
3) does not allocate memory dynamically;
4) does not use floating-point operations;
5) does not contain unbounded loops, i.e., it only contains `for` loops where the loop variable `i` is an integer that

is monotonically increased (decreased) until it reaches a certain maximum (minimum) `n`. However, `n` is not required to be a compile-time constant (yet it must be constant upon entering the loop), and the loop condition may be extended by conjunctively adding any number of boolean expressions (i.e., the loop may terminate before reaching `n`).

A solver which fulfills all of these requirements could not be expected to perform well in the MSE, because the sizes of the benchmarks are unknown beforehand, which conflicts with requirements 3) and 5). To circumvent this, *noSAT-MaxSAT* is split into a library that fulfills the requirements, and an application that uses the library but is not bound by the above-mentioned restrictions.

The interface of the library essentially consists of two functions: `nsms_solve` and `nsms_calcMemoryRequirements`. Given the number of variables and clauses of a formula, the latter function computes (an upper bound on) the number of bytes of memory the solver will need. The former function takes the formula, a pointer to a sufficiently-sized memory block, and the algorithm configuration. The application code takes care of parsing the input file and allocating memory to construct the formula that is then passed to the library functions.

In a constrained computing environment this splitting is not an option. For a particular application, however, it can be expected that the problem domain is more homogenous in such environments than in the MSE, so upper bounds on the number of variables and clauses are known a priori and memory can be pre-allocated statically.

References

[1] Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. *MaxSAT Evaluation 2019 : Solver and Benchmark Descriptions*. Department of Computer Science, University of Helsinki, 2019. URL: https://helda.helsinki.fi/handle/10138/308068.

[2] Fahiem Bacchus, Jeremias Berg, Matti Järvisalo, and Ruben Martins. *MaxSAT Evaluation 2020 : Solver and Benchmark Descriptions*. University of Helsinki, Department of Computer Science, 2020. URL: https://helda.helsinki.fi/handle/10138/318451.

[3] Fahiem Bacchus, Jeremias Berg, Matti Järvisalo, and Ruben Martins. *MaxSAT Evaluation 2021 : Solver and Benchmark Descriptions*. Department of Computer Science, University of Helsinki, 2021. URL: https://helda.helsinki.fi/handle/10138/333649.

[4] Fahiem Bacchus, Jeremias Berg, Matti Järvisalo, Ruben Martins, and Andreas Niskanen. *MaxSAT Evaluation 2022 : Solver and Benchmark Descriptions*. Department of Computer Science, University of Helsinki, 2022. URL: https://helda.helsinki.fi/handle/10138/347396.

[5] Yi Chu, Shaowai Cai, Zhendong Lei, and Xiang He. "NuWLS-c: Solver Description". In: *MaxSAT Evaluation 2022 : Solver and Benchmark Descriptions*. Department of Computer Science, University of Helsinki, 2022, p. 28.

[6] M. R. Garey, D. S. Johnson, and L. Stockmeyer. "Some Simplified NP-complete Graph Problems". In: *Theoretical Computer Science* 1.3 (1976), pp. 237–267. DOI: 10.1016/0304-3975(76)90059-1.

[7] S. Even, A. Itai, and A. Shamir. "On the Complexity of Time Table and Multi-Commodity Flow Problems". In: *16th Annual Symposium on Foundations of Computer Science*. 1975, pp. 184–193. DOI: 10.1109/SFCS.1975.21.

[8] Bengt Aspvall, Michael F. Plass, and Robert E. Tarjan. "A Linear-Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas". In: *Information Processing Letters* 8.3 (1979). DOI: 10.1016/0020-0190(79)90002-4.

[9] Shaowei Cai, Chuan Luo, and Haochen Zhang. "From Decimation to Local Search and Back: A New Approach to MaxSAT". In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*. IJCAI-17. Melbourne, Australia, Aug. 2017, pp. 571–577. DOI: 10.24963/ijcai.2017/80.