noSAT-MaxSAT

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

Sibylle Schupp
Institute for Software Systems
Hamburg University of Technology (TUHH)
Hamburg, Germany
schupp@tuhh.de

***Abstract*—Since 2019, all solvers in the incomplete track of the MaxSAT Evaluation (MSE) include a dedicated SAT solver. In resource-constrained computing environments, e.g., embedded systems, such algorithm designs can be hard to realize. Yet, the MSE results show an undeniable increase in efficiency when using a SAT solver. With *noSAT-MaxSAT*, we propose to replace the call to a SAT solver by executing the MaxSAT solver only on the hard clauses of the formula instead. The algorithm is based on SATLike, which was was the only algorithm that did not rely on a SAT solver in the MaxSAT Evaluation 2018. Additionally, our implementation satisfies a set of requirements often found in embedded system software.**

I. Introduction

In recent years the solvers in the incomplete track of the MaxSAT Evaluation (MSE) have converged to employ algorithms that rely on complete SAT solvers. Indeed, since 2019, that is true for all incomplete solvers that entered the MSE. 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].

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 with regards to 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-MaxSAT is based on the SATLike algorithm by Lei and Cai [4], which is the most recently proposed algorithm in the MSE that does not rely on a SAT solver. SATLike entered the MSE 2018 together with its variant SATLike-c, which does employ an external SAT solver to obtain a satisfying assignment for hard clauses and consistently performed better than the former [5]. On the one hand, its performance highlights the undeniable effectiveness of solving MaxSAT problems through SAT. On the other hand, this leads to the core idea of *noSAT-MaxSAT*: Replacing the execution of a SAT solver in SATLike-c by running SATLike itself instead, but only on the hard clauses.

In the following, we introduce the requirements and architecture of *noSAT-MaxSAT*, and briefly describe the SATLike algorithm and the modifications made for *noSAT-MaxSAT*.

II. 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. It applies the SATLike algorithm as described in the following section. 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 much 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.

III. Algorithm

Algorithm 1 noSAT-MaxSAT

Require: partial weighted MaxSAT formula F, unsigned integers $maxFlips$ and $maxTries$, SATLike parameters
Ensure: a feasible assignment for F and the resulting total cost, or no assignment

```
minCost ← ∞
bestAssignment ← random
for t ← 0;
t < maxTries and no. of unsat. clauses > 0;
increment t by 1
 do
  assign bestAssignment to F
  preprocess F
  if no. of soft clauses > 0 then
    execute noSAT-MaxSAT on hard clauses of F
  end if
  for f ← 0;
  f < maxFlips and no. of unsat. clauses > 0;
  increment f by 1
   do
    if current cost < minCost then
      minCost ← current cost
      bestAssignment ← current assignment
      f ← 0
    end if
    v ← select variable according to SATLike
    flip v
  end for
  if current cost < minCost then
    minCost ← current cost
    bestAssignment ← current assignment
  end if
end for
```

A high-level description of *noSAT-MaxSAT* is given in Algorithm 1. It is based on the SATLike algorithm by Lei and Cai that employs an effective clause-reweighting and variable selection mechanism [4]. For preprocessing it is combined with a unit clause propagation-based method that was introduced by Cai et al. [6] and was also present in the original implementation. After preprocessing, SATLike is executed only on the hard clauses of the formula, trying to obtain a satisfying assignment for them. As mentioned earlier, such a step can greatly improve efficiency, but is usually performed by a dedicated SAT solving algorithm. Details on the SATLike algorithm and its parameters are omitted here for brevity; we use the same parameters as reported by Lei and Cai [4]. The parameter $maxTries$ is set to the largest possible value (`UINT64_MAX`) so the solver runs until it is stopped by an external signal, as the MSE rules for incomplete solvers require. $maxFlips$ is initialized with 10 times the number of variables of the input formula to allow for restarts and ensure the feedback mechanism of the preprocessing method is actually used (conflicting unit clauses are resolved by setting the affected variable to its value from $bestAssignment$) [6].

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 (visited on 05/13/2022).

[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 (visited on 05/13/2022).

[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 (visited on 04/25/2022).

[4] Zhendong Lei and Shaowei Cai. "Solving (Weighted) Partial MaxSAT by Dynamic Local Search for SAT". In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. IJCAI-18. Stockholm, Sweden, July 2018, pp. 1346–1352. DOI: 10.24963/ijcai.2018/187.

[5] Ruben Martins, Matti Jarvisalo, and Fahiem Bacchus. "MaxSAT Evaluation 2018". SAT 2018 (Oxford, UK). July 2018. URL: https://maxsat-evaluations.github.io/2018/mse18-talk.pdf (visited on 05/13/2022).

[6] 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.