SharpSAT-TD Participating in Model Counting Competition 2021

Tuukka Korhonen Matti Järvisalo

HIIT, Department of Computer Science, University of Helsinki, Finland*

May 30, 2021

Abstract

We describe SharpSAT-TD, our submission to the unweighted and weighted tracks of Model Counting Competition 2021. SharpSAT-TD is based on SharpSAT, with the primary novel modification being the use of tree decompositions in decision heuristics.

1 Overview

SharpSAT-TD is based on the exact model counter SharpSAT [7], from which it inherits the basic structure of search-based model counter with clause learning, component analysis, and component caching. The main new feature in SharpSAT-TD is that we compute a tree decomposition of the input formula with the FlowCutter algorithm [2, 6], and integrate the tree decomposition to the decision heuristic of the counter. Another significant new feature is a new preprocessor. Further, SharpSAT-TD extends SharpSAT by directly supporting weighted model counting.

2 Integrating Tree Decompositions into the Decision Heuristic

We compute a tree decomposition of the primal graph of the input formula with the anytime FlowCutter algorithm [2, 6], always using 120 seconds in the computation. We select one node of the tree decomposition as a root so that the bag of the root is a 1/2-balanced separator of the primal graph.

In SharpSAT, the variable selection heuristic is based on frequency and activity scores, in particular, for each variable x scores $\texttt{freq}(x)$ and $\texttt{act}(x)$ are maintained, and the variable with highest $\texttt{score}(x) = \texttt{freq}(x) + \texttt{act}(x)$ is selected. We add a tree decomposition based term to this score. In particular, in SharpSAT-TD the variable selection done using

$$\texttt{score}(x) = \texttt{freq}(x) + \texttt{act}(x) - C \cdot d(x),$$

where $d(x)$ denotes the distance in the tree decomposition from the root node to a closest node whose bag contains x (normalized to the interval $[0 \dots 1]$), and C is some positive constant. In particular, this score prefers variables that are closer to the root in the tree decomposition. The constant C is selected as $C = 100 \exp(n/w)/n$, where w is the width of the tree decomposition and n is the number of variables. Note that this makes the tree decomposition based score more significant when w is smaller.

3 Preprocessing

We implement a new preprocessor completely from scratch. The preprocessing is done before computing the tree decomposition, and one of the goals of the preprocessing is to decrease the treewidth of the formula. The preprocessing techniques used, in the order of application, are propagation-based vivification, complete vivification, sparsification, equivalence merging, and B+E [3, 4].

In propagation-based vivification, for each clause c and a literal l in c we check if $\neg(c \setminus \{l\})$ implies UNSAT via unit propagation, and if yes we strenghten c by removing l from it. In complete vivification we do the same, but the UNSAT check is done by a complete SAT-solver. The SAT-solver is also a new implementation. Note that complete vivification also results in backboning the formula. In sparsification we attempt to remove clauses that are implied by other clauses. The redundancy of a clause is checked with a SAT-solver. The goal of sparsification is to reduce treewidth. In equivalence merging we merge two variables if they are equivalent and they are adjacent in the primal graph. The equivalency is checked with a SAT-solver. Note that in the primal graph, merging two adjacent variables corresponds to edge contraction, which does not increase treewidth. Finally, for unweighted formulas we re-implement the B+E algorithm [3]. Our implementation of B+E is done in a way to ensure to not increase the treewidth of the

*Work funded by Adacemy of Finland under grants 322869 and 328718.

formula. On some instances our implemention of B+E seems faster than the original, while eliminating the same number of variables.

4 Further New Features and Modifications

We disabled the "implicit BCP" feature of SharpSAT because it decreased the overall number of public instances solved in preliminary experiments, although we note that on some instances it appears useful.

We changed the SharpSAT learned clause scoring scheme into the LBD scheme [1]. This also fixed a bug of SharpSAT where it does not delete any learned clauses if their median score is 0, which often happened. SharpSAT-TD also changes the desired number of learned clauses stored based on their estimated usefullness, i.e., LBD scores. In preliminary experiments it seems that these LBD score based techniques do not result in very significant effects in the running times, but the bugfix sometimes does.

We implemented the probabilistic component caching scheme introduced in GANAK [5]. We note that even though GANAK is also a SharpSAT-derivant, our implementation is different from the implementation of GANAK. Also, in our implementation instead of having a dynamic hash-length we fix a hash-length of 128 bits, noting that it results in a collision probability of $< 10^{-9}$ on all cases where the counter does at most 10^{14} cache lookups (note that doing more than 10^{14} cache lookups within the 3600 seconds time limit of the competition seems impossible on the competition hardware).

5 Extension to Weighted Model Counting

While in principle it is clear that any model counter whose trace corresponds to a d-DNNF-compilation can be extended to also weighted model counting, efficient implementation is not necessarily straightforward, and in the case of SharpSAT required editing hundreds of lines of code. Our extension to weighted model counting is implemented via template parameters, so changing the data types or the semiring that the counter works on is simple. In particular, on some public instances of the weighted track of the competition there is an issue of underflow with the double-precision floating point data type. In these instances we re-run the counter with a custom datatype that stores the logarithm of the weighted model count.

References

[1] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *IJCAI*, pages 399–404, 2009.

[2] M. Hamann and B. Strasser. Graph bisection with pareto optimization. *ACM J. Exp. Algorithmics*, 23, 2018.

[3] J. Lagniez, E. Lonca, and P. Marquis. Improving model counting by leveraging definability. In *IJCAI*, pages 751–757. IJCAI/AAAI Press, 2016.

[4] Cédric Piette, Youssef Hamadi, and Lakhdar Sais. Vivifying propositional clausal formulae. In *ECAI*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 525–529. IOS Press, 2008.

[5] S. Sharma, S. Roy, M. Soos, and K. S. Meel. GANAK: A scalable probabilistic exact model counter. In *IJCAI*, pages 1169–1176. ijcai.org, 2019.

[6] B. Strasser. Computing tree decompositions with FlowCutter: PACE 2017 submission. *CoRR*, abs/1709.08949, 2017.

[7] M. Thurley. sharpSAT - Counting models with advanced component caching and implicit BCP. In *SAT*, volume 4121 of *LNCS*, pages 424–429. Springer, 2006.