SharpSATTD-CH Participating in Model Counting Competition 2024

Yipei Deng   Junping Zhou

Department of Electronic Information, Northeast Normal University, China*

May 22, 2024

Abstract

We describe SharpSATTD-CH, a model counter competing in the unweighted and weighted tracks of Model Counting Competition 2024. SharpSATTD-CH is based on SharpSAT-TD. Our primary innovation lies in enhancing the existing cache management scheme of SharpSAT-TD.

1 Overview

SharpSATTD-CH is an exact model counter based on SharpSAT-TD [3], which inherits its advanced techniques, incorporating tree decomposition-based preprocessing in decision heuristics, clause learning, and component analysis. The main innovation of SharpSATTD-CH lies in improving the original cache management strategy by introducing a new cache management scheme. This enhancement makes the cache management strategy more effective, thereby further improving the solving speed.

2 Introduction of Caching

The caching technique directly impacts the time complexity and space complexity of model counters. It reduces a lot of repetitive computations. In essence, a caching scheme is a mapping that associates a CNF formula with its corresponding information. If a solver encounters duplicate components (those already calculated and stored in the cache) when performing computations, it can directly access the previously computed results, reducing the time spent on repetitive calculations. Actually, many state-of-the-art solvers have incorporated caching techniques to enhance their computational efficiency, including Cachet [1], SharpSAT [2], and SharpSAT-TD [3], which won the first place in both the model counting category and the weighed model counting category in Model Counting Competition 2023.

3 An Improved Cache Management Strategy

In the existing caching management strategies, the utilization efficiency of the cache tends to sharply decrease as more entries are inserted when solving larger CNF formulas. Therefore, it's crucial to design an effective cache management strategy so that more entries (components given their model counts) in cache can be reused. In SharpSATTD-CH, every entry encountered during the search is a candidate for being inserted into the cache. Whenever the cache size exceeds a certain threshold (a set parameter), the entry deletion is performed. To determine which entries should be removed from the cache, each entry *e* in the cache is associated with an integer value *score*(*e*) and a Boolean value *flag*(*e*). Once an entry is hit during the search, a Boolean value *flag*(*e*) is changed to *true*, whose is initialized to *false*. For each entry *e*, its *score*(*e*) is defined as the follows.

$$score(e) = \lambda_1 score_{td}(e) + \lambda_2 score_{hit}(e) \tag{1}$$

where λ_1 and λ_2 ($0<\lambda_1, \lambda_2 <1$) are two parameters, $score_{td}$(e) is the score value in SharpSAT-TD, whose evaluation is based on the age of the entry, and the $score_{hit}$(e) is the score value calculated by the number of cache hits (represented as HitCount(*e*)) and the hit rate (represented as rate(*e*)), where the rate(*e*) is obtained by dividing the number of cache hits by the number of variables in entry e. At first, $\text{score}_{hit}(\text{e})$ is initialized to 0. Then, $\text{score}_{hit}(\text{e})$ is updated (1) when *rate*(*e*) is higher than a set parameter (here set to 0.3), $\text{score}_{hit}(\text{e})$ will gain a reward; (2) when entry e associates with *flag*(*e*)=$true$, $\text{score}_{hit}(\text{e})$ will also receive a bonus. Before performing the deletion operation, the average score(*e*) of all entries is calculated. Entries with scores below this average are removed, typically clearing approximately half of the space. After each deletion operation, every *score*(*e*) is updated to $\lfloor score(e)/2 \rfloor$. Furthermore, *flag*(*e*) is set to *false* if the current *score*(*e*) becomes 0, even when its previous value is $true$. Indeed, this dynamic adjustment of *scores* ensures that entries which are frequently accessed in the past but are no longer being hit as time can be promptly removed. For efficiency reasons, updates to each *score*(*e*) don't occur constantly. This approach minimizes the time spent on updating entry values.

References

[1] Sang T, Bacchus F, Beame P, et al. Combining Component Caching and Clause Learning for Effective Model Counting[J]. SAT, 2004, 4: 7th.

[2] Thurley M. sharpSAT–counting models with advanced component caching and implicit BCP[C]//International Conference on Theory and Applications of Satisfiability Testing. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006: 424-429.

[3] Korhonen T, Järvisalo M. SharpSAT-TD in Model Counting Competitions 2021-2023[J]. arXiv preprint arXiv:2308.15819, 2023.