2019 2nd International Conference on Mechanical Engineering, Industrial Materials and Industrial Electronics (MEIMIE 2019)

Optimization of Approximate Model Counting Method Based on Bivariate Decomposition

Xuan Zhou[a], Bin Zhang[b], Yankun Tu[c], Shuaiqi Wang[d] and Sihan Jia[e]

College of Computer Science and Technology, Jilin University, Changchun 130012, China

[a]zhouxuan2116@mails.jlu.edu.cn, [b]zhangbin2116@mails.jlu.edu.cn, [c]tuyk2116@mails.jlu.edu.cn, [d]wangsq2116@mails.jlu.edu.cn, [e]jiash2116@mails.jlu.edu.cn

Keywords: Approximate model counting, bivariate decomposition, #SAT

Abstract: This paper proposes an improved method for approximate model counting based on bivariate optimization. With the development of hash functions, scalable approximate model counting algorithms that can solve tens of thousands of variables in recent years have been proposed, improved and applied. However, in practical applications, there are still a large number of laws for the actual input data of different problems, and there is a lot of optimization space. This paper proposes an input variable optimization algorithm based on bivariate optimization for approximate model counting. It optimizes the structure of a large number of bivariate and univariate inputs in practical problems, and successfully improves the calculation time of the most approximate model counting problems without losing accuracy.

1. Introduction

The term "artificial intelligence" was first proposed in 1956, for more than 60 years, with it developed rapidly, it has become a wide-ranging cross-cutting and cutting-edge discipline. The intelligent planning is one of the key research fields of artificial intelligence, revolutionary advances in recent years, it is widely used in important areas of aerospace, industrial and so on. Among them, the propositional satisfiability (SAT) can effectively solve most of the planning problems, which is also the first to be proved as the NP problem. However, there are still many classic problems in the field of intelligent planning, such as Bayesian network reasoning, confidence analysis, etc., which need to be transformed into new problems. Then the model counting (#SAT) is generated, which can calculate the number of all the CNF formulas which can satisfy the assignment. In 1979, Valiant et al. [1] proved that the complexity of the #SAT problem is higher than the SAT problem, which is #P-complete.

On the implementation front, the earliest approaches to #SAT were based on DPLL-style SAT solvers and could compute exact counts. One of the quickest resolution principles prove theorems in propositional logic based on the current method is based on DPLL, Birnbaum and Lozinkii[2] directly extended DPLL to solve model counting problems, thus, the CDP algorithm is proposed, after finding a partial solution, the number of solutions is incrementally calculated by adding appropriate multi-factors. Based on this algorithm, Bayardo and Pehoushek [3] use the connection graph to solve the problem separately by decomposing the formula into multiple sub-formulas by describing the relationship between the variables. Sang et al. also designed the Cachet by combining the DPLL algorithm with component caching, clause learning and forward-looking strategies. There Relsat [4] and sharpSAT [5] also use these optimization improvements. On small and medium-sized issues, the application of accurate model counting is very successful and effective, but when scaling to larger areas, the use of accurate model counting is not enough and the time is expensive - all index-level, so it’s not very suitable in practical applications. And in many areas, the approximate counting is sufficient to deal with these problems, such as probabilistic inference. The approximate model counting method appears in front of people, and ApproxCount[6] is proposed by Selman et al., it can estimate models by sampling method, which belongs to lower (or upper) bounding counters, similar to SearchTreeSampler [7], SE [8] anSampleSearch [9].

In applications such as probabilistic reasoning, according to Karp and Luby's counting model, it is not possible to calculate CNF based on Monte Carlo engine, but DNF can be calculated. The complexity of both is a #P-complete problem. But CNF is more widely used than DNF. At the same time, Karp and Luby [9] also pointed out that the random approximation algorithm of the DNF formula is unlikely to be applicable to the CNF formula.

So, in this paper, we use the ApproxCount, which is a new algorithm based on SampleSat that is almost uniformly sampled from the solution space of the propositional logic formula. We decompose and optimize the bivariate, and remove the redundant variables and disjunction paradigm through the classification method. The calculation speed is greatly improved without loss of accuracy

The structure of this paper is as follows. In the second section, we introduce the related work, and in the third section, we introduce the preliminary concepts and theorems. In section 4, we list the results of the experiment. Finally, the conclusion is drawn in the fifth section.

2. Related Works

2.1 Theories

Approximate model counting which uses the extension rule is an alternative for exact model counting in practice. Based on SampleSat, ApproxCount sampled from the set of solutions of a propositional formula near-uniformly. As for SampleSat, it is a technique for counting models of Boolean satisfiability problems, and it doesn’t need uniform or near-uniform samples, but it can transform unguaranteed search sample into guaranteed one with good bounds. After SampleSat setting some variables, ApproxCount uses sample method to count models of the residual formula. The number of calls ApproxCount makes to SampleSat is determined by the number of variables in the formula under consideration [6]. To solve SAT problem, the extension rule is an available method, and using inclusion–exclusion principle is the core method to solve this kind of problems. On the other hand, ApproxCount is based on a biased random walk strategy, which has been shown to be an effective way to solve various and varied SAT problems.

2.2 Conceptions

Dealing with approximate counting and near-uniform sampling, Sipser [10] initiated and used a method based on hashing. And there are some others just like it using the same family of hashing functions, such as Hybrid-Mbound [11]. In this paper, we use an approximate model counting algorithm which is called ApproxMC. It uses 3-wise independent linear hashing functions from the Hxor (n, m, 3) family. It evolved from UniWit. The main idea of UniWit is to randomly divide the model spaces of a given instance into some small parts by using r-wise independent hashing functions. There is something make UniWit better than other hashing-based sampling algorithms. Back to ApproxMC, its reports are having the specified tolerance of exact count [12]. With packaging the core part as a function called ApproxMCCore, users could just invoke it for multiple times, which is easier to understand and operate. The inputs are a CNF formula and a threshold, and return a ε-approximate estimate of the model count of the initial CNF formula.

And below are parts of the used theory. First, let Σ be an alphabet and $R \subseteq \Sigma* \times \Sigma*$ be a binary relation. R is an N P-relation if R is polynomial-time decidable, and if there exists a polynomial p $(\cdot)$ such that for every $(x, y) \in R$, we have $|y| \leq p\ (|x|)$. Let LR be the language. If R is an N P-relation, we may further assume that for every $x \in LR$, every witness $y \in Rx$ is in {0, 1} n, where n = p (|x|) for some polynomial p $(\cdot)$.

3. Algorithm

For approximate model counting algorithm ApproxMC, we use 3-directional opposite linear hash function Hxor (n, m, 3). For an appropriate m, we randomly divide the model set of the input formula into sufficiently small units. After selection, test whether it is not empty and has only pivotal elements. If the result is negative, then choose a random hash function from Hxor (n, m + 1, 3) and divide the model set into two cells. Repeat the process until the unit meets the requirements. If not, report failure and return.

```
Algorithm $Approxmc\left(F,\varepsilon,\delta\right)$
1: counter $\leftarrow$ 0; C $\leftarrow$ emptylist;
2: $pivot \leftarrow$ 2 X ComputeThreshold ( $\varepsilon$ )
3: t $\leftarrow$ ComputelerCount ( $\delta$ );
4: repeat;
5:         c $\leftarrow$ ApproxMCCore (F, pivot);
6:         counter $\leftarrow$ counter +1;
7:         if (c $\neq \perp$)
8:             Addtolist (C, c);
9: until (counter < t);
10: finalCount $\leftarrow$ FindMedian (C);
11: return finalcount;
Algorithm Compute Threshold ( $\varepsilon$ )
1: return $\left\lceil 3e^{1/2}(1+\frac{1}{\varepsilon})^2 \right\rceil$
Algorithm Computeltercount ( $\delta$ )
1: return $\lceil 35\log_2(3/\delta) \rceil$
```

The bivariate decomposition optimization algorithm is a simultaneous decomposition of two variables to the CNF paradigm. Basic principle: Find a binary disjunction paradigm and ensure that the frequency of occurrence is high enough to assign variables (0, 0), (1, 0), (0, 1), (1, 1). For (0, 0), the situation must not work. For (1, 0) case decomposition: All non-existence paradigms containing variable one and variable two are 1, and can be deleted. All non-and variable two containing variable one in all disjunction paradigms are removed. (0, 1) and (1, 1) is similar with (1, 0). This resulted in three CNF files, each of which became smaller because of fewer variables and disjunction paradigms. The approximate model count is obtained for the new CNF (not necessarily calculating for the three CNF files, for the different data sets, there are suitable rules, with the Bivariate Decomposition Algorithm, it gets three CNF files, just count the available CNF files and multiple the number), and the number of solutions is obtained.

4. Experiment

This part implements the decomposition of bivariate. Compared with the approxMC implemented by Python and the approxMC optimized by the bivariate decomposition algorithm, the experimental results highlight the good performance of the bivariate decomposition optimization algorithm. The experimental environment is: operating system Linux (Ubuntu 18.04), CPU Intel Core i7-7700HQ 2.8GHz, memory 16GB. The ApprocMC code (python implement) is https://github.com/ Uddipaan/ ApproxMC. For several different types of data sets, we performed the following tests (all test results were run averaging ten times):

Table.1. Blasted Data Set (CPU Time/s)

Problem	Pre-Time	New-Time
blasted4.cnf~	0.455	0.402
blasted47.cnf~	271.635	139.9
blasted49.cnf~	0.696	0.525
blasted110.cnf~	0.405	0.394
blasted201.cnf~	0.396	0.383
blasted205.cnf~	0.443	0.431

Table.2. Plan Data Set (CPU Time/s)

Problem	Pre-Time	New-Time
5step.cnf	0.512	0.485
tire-1.cnf	0.809	0.564
tire-3.cnf	1.354	1.347

Table.3. DQMR Data Set (CPU Time/s)

Problem	Pre-Time	New-Time
or-50-10-1.cnf	134.8	95.2
or-50-10-9.cnf	190.5	152.3
or-20-9-10.cnf	6.86	5.21
or-20-1-10.cnf	74.3	54.1
or-20-10-10.cnf	9.15	6.73

5. Conclusion

In this paper, an approximate model counting optimization algorithm for bivariate decomposition is proposed. Combined with the existing approximate model counting method, a fast solution is realized. The experimental results show that the use of this algorithm improves the effectiveness of the existing approximate model counting method. In the future, the optimization algorithm can be used in a more advanced approximate model counting method, which further improves the calculation speed of the current approximate model counting, reduces the calculation time, and makes the solution speed and capability improve faster.

References

[1] LG Valiant. The complexity of computing the permanent [J]. Theoretical Computer Science, 1979, 8 (2): 189-291.

[2] E. Birnbaum and E. L. Lozinskii. The good old Davis-Putnam procedure helps counting models. Journal of Artificial Intelligence Research, 10 (1): 457–477, June 1999.

[3] Bayardo R J, Pehoushek J D. Counting models using connected components[C]// Seventeenth National Conference on Artificial Intelligence & Twelfth Conference on Innovative Applications of Artificial Intelligence. AAAI Press, 2000.

[4] R. J. Bayardo Jr. and R. Schrag. Using CSP look-back techniques to solve real- world SAT instances. In Proc. of AAAI, pages 203–208, 1997.

[5] M. Thurley. sharpSAT: counting models with advanced component caching and implicit bcp. In Proc. of SAT, pages 424–429, 2006.

[6] Wei W and Selman B. A new approach to model counting [C]. In Proc. of 8th International Conference on Theory and Applications of Satisfiability Testing. 2005 226-240.

[7] S. Ermon, C.P. Gomes, and B. Selman. Uniform solution sampling using a constraint solver as an oracle. In Proc. of UAI, 2012.

[8] Gogate and R. Dechter. Samplesearch: Importance sampling in presence of determinism. Artificial Intelligence, 175 (2): 694–729, 2011.

[9] R.M. Karp, M. Luby, and N. Madras. Monte-Carlo approximation algorithms for enumeration problems. Journal of Algorithms, 10 (3): 429–448, 1989.

[10] M. Sipser. A complexity theoretic approach to randomness. In Proc. of STOC, pages 330–335, 1983.

[11] Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Model counting: A new strategy for obtaining good bounds. In Proc. of AAAI, pages 54–61, 2006.

[12] Chakraborty, Supratik, K. S. Meel, and M. Y. Vardi. A Scalable Approximate Model Counter. Principles and Practice of Constraint Programming. Springer Berlin Heidelberg, 2013.