IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 10, 2018, accepted June 13, 2018, date of publication July 13, 2018, date of current version August 15, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2855739

A Parallel Extension Rule-Based Algorithm for #SAT Problem Using Model-Counting Tree

NAIYU TIAN[1], DANTONG OUYANG[1], FENGYU JIA[2], MENG LIU[1], AND LIMING ZHANG[1]

[1]Key Laboratory of Symbolic Computation and Knowledge Engineering, Ministry of Education, College of Computer Science and Technology, Jilin University, Changchun 130012, China

[2]China Asset Management Co., Ltd., Beijing, China

Corresponding author: Liming Zhang (limingzhang@jlu.edu.cn)

This work was supported by the National Natural Science Foundation of China under Grants 61672261, 61502199, 61402196, and 61373052.

ABSTRACT The #SAT problem, which is also called model counting, is one of the most important problems in artificial intelligence and is challenging to the researchers. The model counting based on extension rule (CER) algorithm is an exact algorithm for model counting. The weak point of the algorithm is the high computational complexity which adds to the running time. We introduce parallel CER, an algorithm that parallelizes the model counting algorithm CER. The CER algorithm is based on the extension rule. We propose a notion of MC-Tree for describing the computing procedure. We implemented the algorithm on a quad-core machine using OpenMP to measure the performance. Our experimental results on simulated data show that: 1) with the increase of the number of processors, the running time of our parallel algorithm reduces in inverse proportion, and furthermore, the algorithm is more efficient in case of using each number of processors when the complementary factor is higher and 2) the scalability of the algorithm is linear for all instances, and the efficiency is evident.

INDEX TERMS Extension rule, model counting, model-counting tree, multi-core, OpenMP, parallelization.

I. INTRODUCTION

In recent years, there has been a comprehensive study and a significant step forward in the field of Propositional Satisfiability (SAT) [1]–[5]. SAT is to find if there is an interpretation which makes the propositional formula true. The problem is of great significance and widely used in the field of artificial intelligence. Many real-world problems such as formal verification [6], synthesis [7], classic planning problem [8], and model-based diagnosis problem [9] can be compiled into SAT instances and solved effectively by SAT solvers [10]. In fact, SAT is a Non-deterministic Polynomial (NP) complete problem [11]. It is the first proved NP-complete problem, which has important theoretical and practical significance. And almost all NP-complete problems from a variety of domains can be transformed into SAT problems and many of these problem instances can be effectively solved via satisfiability.

However, sometimes it is not enough to just know if a propositional formula is satisfiable. To solve many important problems such as conformant probabilistic planning [12] and performing inference in Bayesian networks [13], we need to know the number of the true interpretations of corresponding propositional formulas. The complexity of these problems is higher than NP-complete problems, that is, it is #P-complete [14], [15]. #SAT problem is to get the number of the true interpretations of a propositional formula. The true interpretations are called the models of propositional formula. Therefore, #SAT problem is also called model counting. It is an important extension of SAT.

Currently, #SAT problem has drawn more and more attention from researchers. #SAT is a well-studied problem that is of fundamental theoretical importance. In addition, many algorithms for #SAT problem have been proposed. The algorithms of #SAT are divided into approximate algorithms [16]–[19] and exact algorithms [20]–[23]. Usually, approximate algorithms are based on different techniques such as SampleSat, that is, an algorithm that samples from the solution space of a propositional logic formula near-uniformly [24], the additional constraints of the input formula [18]. The advantages of these algorithms are that the running time is faster and the size of problems that can be solved by approximate algorithms is larger than by the exact algorithms. The disadvantage is that the number of models that have been solved is not precise. Most exact algorithms for #SAT are based on the study of Davis and Putnam [25]. Based on the Davis-Putnam procedure, Birnbaum and Lozinskii [20] presented the CDP (Counting by Davis-Putnam) algorithm that counts the number of models of propositional formulas. In CDP,

a propositional formula is split into two subformulas by choosing a variable x, one in which x is true, and the other is false. The model sets of the two subformulas are disjointed. Let $M(\Phi)$ stands for the number of models of the original propositional formula Φ, and $M(\Phi_1)$, $M(\Phi_2)$ separately stands for the number of models of the two subformulas,Φ_1, Φ_2. In addition, n, n_1, and n_2 denote the number of variables that occur in Φ, Φ_1, and Φ_2. Then: $M(\Phi) = M(\Phi_1) * 2^{n-n_1} + M(\Phi_2) * 2^{n-n_2}$. Bayardo and Pehoushek [21] presented a further extension, based on recursively identifying disconnected constraint-graph components, that substantially improves counting performance. The nodes of the connected graph represent the variables that are contained in the clause set. If there is a boundary between nodes, the two variables represented by these two nodes are contained in the same clause. By this approach, all variables of the clause set can be divided into a plurality of disconnected components. Thus the propositional formula can be decomposed into multiple subformulas to be solved separately. Sang *et al.* [22] presented an effective procedure for #SAT on combining component caching and clause learning. In this procedure, the conflicts and clauses that appear are recorded to avoid repeated computation.

Resolution principle is not the only rule of inference at the basis of most procedures for both SAT and #SAT. Yin *et al.* [23] presented a new algorithm CER for #SAT based on the extension rule. It is a rule that is different from the resolution principle. The idea is to deduce the set of all the maximum terms for counting models and to use the inclusion-exclusion principle to circumvent the problem of space complexity. Algorithm CER is an exact algorithm which outputs the precise number of models of the input formula. In addition, this method is more efficient than resolution-based methods in cases where the complementary factor of a clause set is higher. Furthermore, when the complementary factor is lower, resolution-based methods are more efficient.

In this paper, we analyze the algorithm CER for #SAT, and present the notion of Model-Counting-Tree. Based on this, we propose a strategy to parallelize the algorithm. Subsequently, we implement the parallel algorithm on a multi-core system using OpenMP. The maximum number of processors used in the implementation is four. Experimental results show that the efficiency for counting models in a parallelized manner improves considerably.

II. RELATED WORK

First of all, the definitions of SAT and #SAT are clarified as follows:

Definition 1: Given a propositional formula δ, the SAT problem is to find if there is an interpretation which makes the propositional formula true. If there is such an interpretation, it is said that the formula δ

Definition 2: Given a propositional formula δ, the #SAT problem is to get the number of the true interpretations of the propositional formula δ.

In general, a propositional formula is solved in the conjunctive normal form (CNF). We can suppose that the CNF of a propositional formula is simply a conjunction of clauses, where each clause is a disjunction of literals, and each literal is either a positive variable or a negative variable. Therefore, a propositional formula can be considered to be a clause set, and the clauses can be seen as a collection of variables.

We use Φ to denote a set of clauses in CNF, C to denote a single clause, X to denote the set of all variables that appear in Φ, m to denote the number of all variables in X, and n to denote the number of clauses in Φ.

The extension rule is the inverse of the resolution. Because it is the basic rule in CER, it must be realized first before understanding the algorithm CER. The definitions of the extension rule are clarified as follows:

Definition 3 [26]: Given a clause C and a set X, $D = \{C \vee a, C \vee \neg a\}$ where "a" is a variable that do not appear in C and $a \in X$. We call the operation proceeding from C to D the extension rule on C. We call D the result of using the extension rule on C.

Example 1: Given the clause $a \vee b$ and the set $X = \{a, b, c\}$ of variables, the result of the extension rule on $a \vee b$ is $\{a \vee b \vee c, a \vee b \vee \neg c\}$.

Theorem 1 [26]: A clause C is logically equivalent to the result of the extension rule D.

Therefore, by using the extension rule on the clauses in a clause set, a new clause set can be derived that is equivalent to the original clause set.

Definition 4 [26]: A clause is a maximum term on a set X if and only if it contains all variables in X in either positive or negative form.

Example 2: Given a set $X = \{a, b, c\}$, then the clause $a \vee b \vee c$ a maximum term on X, and $a \vee b$ is not. Therefore, given a clause set Φ, we can get a new equivalent clause set Φ' where the clauses are all maximum terms.

Theorem 2 [26]: Given a set of clauses Φ with its set of variables $X(|X| = m)$, if all the clauses in Φ are maximum terms on X, then Φ is unsatisfiable if and only if it contains 2^m clauses.

The above theorem is used to tell whether a set of maximum terms is satisfiable, or whether the number of models of the set is zero.

The following theorems are used to count the numbers of models of a given set of maximum terms.

Theorem 3 [23]: Given a set of clause Φ with its set of variables $X(|X| = m)$, if all the clauses in Φ are maximum terms on X, and Φ contains S distinct clauses, then the number of models of Φ is $2^m - S$. $(S \leqslant 2^m)$

Based on Theorem 3, the process of model counting is transformed into the process of computing the number of elements of the equivalent clause set in which all clauses in it are maximum terms generated by using the extension rule. This is the basic idea of CER.

Theorem 4 [23]: There are no intersections between the sets of the maximum terms that extended from two clauses by

using the rule, if and only if there is a complementary literal(s) between the two clauses.

Given a set of clauses $\Phi = \{C_1, C_2, \cdots, C_n\}$, let X be the set of variables that appear in Φ ($|X| = m$). Let P_i be the set of all the maximum terms we can get from C_i by using the extension rule, and let S be the number of distinct maximum terms that we can get from Φ. Then, according to extension rule, distinctly, $S = |P_1 \cup P_2 \cup \cdots \cup P_n|$.

However, it is obvious that using the Extension Rule directly (generating all the maximum terms) is infeasible for the consideration of space complexity, although we know that it is sufficient to count the number of all maximum terms rather than to list them. Therefore, this problem can be circumvented by using the inclusion-extension principle.

Theorem 5 (Inclusion-Exclusion Principle): For finite sets $A_1, \cdots A_n$, counting elements of the union set becomes more convenient by using the equation as follows:

$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_{i=1}^{n} |A_i| - \sum_{1\le i<j\le n} \left|A_i \cap A_j\right| + \sum_{1\le i<j<l\le n} \left|A_i \cap A_j \cap A_l\right| - \cdots + (-1)^{n+1} \left|A_1 \cap A_2 \cap \cdots \cap A_n\right|. \quad (1)$$

Based on famous inclusion-extension principle, we obtain the formula:

$$S = \sum_{i=1}^{n} |P_i| - \sum_{1\le i<j\le n} \left|P_i \cap P_j\right| + \sum_{1\le i<j<l\le n} \left|P_i \cap P_j \cap P_l\right| - \cdots + (-1)^{n+1} \left|P_1 \cap P_2 \cap \cdots \cap P_n\right| \quad (2)$$

where $|P_i| = 2^{m-|C_i|}$, and the value of $\left|P_i \cap P_j\right|$ is as follow.

$$\left|P_i \cap P_j\right| = \begin{cases} 0, & \textit{there are complementary literals in } C_i \cup C_j \\ 2^{m-|C_i \cup C_j|}, & \textit{otherwise} \end{cases}$$

Using formula (2), we can easily get the number of models of clause sets.

Example 3: Given a clause set $\Phi = \{\neg a \vee b \vee \neg c, a \vee c, \neg a\}$, we can count models of Φ. Based on formula (2), the number of maximum terms that Φ could extend via extension rule is known, that is, $S = 2^0 + 2^1 + 2^2 - 0 - 2^0 - 0 + 0 = 6$, therefore the result is $2^3 - 6 = 2$.

Clearly, the worst-case time complexity of Algorithm CER is exponential. However, there are situations where Algorithm CER is tractable. For example, if there are complementary literal(s) in each pair of clauses in a set, the complexity of Algorithm CER will be linear in the number of clauses. Because only the first n terms in formula (2) are nonzero terms and need to be computed. Intuitively, for the clause sets in which there are complementary literal(s) for more pairs of clauses, Algorithm CER is more efficient, because there are less nonzero terms that have to be computed using formula (2). We can use the complementary factor to estimate the nonzero terms that are really counted in formula (2). A complementary factor is defined as follow.

Definition 5 [26]: Given a set of clauses $\Phi = \{C_1, C_2, \cdots, C_n\}$, the complementary factor of the set is the ratio of the number of pairs that contain complementary literal(s) to the number of all the pairs in the set. That is, $W/(n * (n-1)/2)$, where W stands for the number of pairs that contain complementary literal(s).

Although it is sometimes difficult to calculate the time complexity precisely by using the complementary factor, nevertheless, the higher the complementary factor of clause set is, the more efficient Algorithm CER is.

We note that Algorithm CER counts models by using extension rule that is the inverse of the resolution and employs the inclusion-exclusion principle to circumvent the problem of space complexity. To parallelize the algorithm CER, we can divide the terms in formula (2) into a number of groups. And every group can be allocated to each processor dynamically. In the following section, we propose an algorithm structure on which the parallelization is primarily based.

III. METHOD OF PARALLELIZATION

In this section, a parallelized strategy for #SAT is given based on the extension rule. To describe the process clearly, we also present a notion of Model-Counting-Tree of clause set.

A. MODEL-COUNTING-TREE

Definition 6: Let Q be the intersection of finite sets $A_1, A_2, \cdots, A_n$. Then the set that consists of these sets is called ancestry set of Q, denoted as $AS(Q) = \{A_1, A_2, \cdots, A_n\}$. In particular, if there is only one set A, then $Q = A$, and we denote the ancestry set as $AS(A)$.

Example 4: Given an intersection of maximum term sets $Q = P_1 \cap P_2 \cap P_3$, P_i is the set of all the maximum terms we can get from C_i in a clause set $\{C_1, C_2, \cdots, C_n\}$. Then the ancestry set of Q is $AS(Q) = \{P_1, P_2, P_3\}$.

Definition 7: Given a clause set Φ, $X(|X| = m)$ is the set of variables that appear in Φ. P_i is the set of all the maximum terms generated from clause C_i in Φ. Let P be the set of all the maximum terms on X, clearly $|P| = 2^m$, $P \cap P_i = P_i$, $(i = 1, 2, \cdots, n)$. T is a Model-Counting(MC)-Tree for Φ, iff:

1) The root of T is labeled by the $|P|$.
2) The children of a node labeled $|Q|$ in T are $\left\{|Q \cap P_i| \,\middle|\, P_i \notin AS(Q)\right\}$.

Fig.1 illustrates a MC-Tree of the clause set $\Phi = \{C_1, C_2, C_3, C_4\}$. Generating a node's children based on its AS ensures that every node is uniquely explored within the tree.

The Model-Counting(MC)-Tree is a instrumentality for representing and enumerating calculation items for model counting in a best-first fashion. The complete MC-Tree

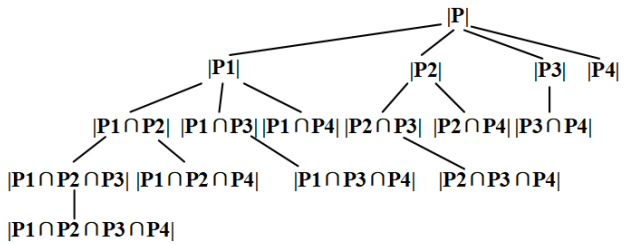


FIGURE 1. The MC-Tree of the clause set: $\{C_1, C_2, C_3, C_4\}$.

enumerates calculation items that are indispensable for model counting using a particular order on them. We can count the number of models on the MC-Tree by traversing the tree and by addition or subtraction of the nodes. Different nodes may lead to different operations, either addition or subtraction. If the number of Levels that the node stays is odd, then the subtraction operation should be made, otherwise the addition operation (the root stays on the Level 0) need to be done. Ultimately, we can get a number from the MC-Tree. This number is the solution of model counting we intend.

Theorem 6: Model counting based on the MC-Tree is sound and complete.

Proof: The calculation process on the MC-Tree is equivalent to the formula $2^m - S$. In addition to the root, the other nodes of the MC-Tree, one-to-one, correspond to the calculation items in formula (2). In addition, the root is $|P| = 2^m$. Therefore, the calculation process traversing the MC-Tree is equivalent to the formula $2^m - S$. However, the latter is sound and complete. Hence model counting based on the MC-Tree is sound and complete.

The MC-Tree actually is a deformation of the formula $2^m - S$, and they are equivalent. We identify the MC-Tree as a recurring data structure, thereby facilitating its use in the framework of model counting. For model counting problem based on the extension rule by which the solution can be obtained by the addition or subtraction between multiple computing items, the MC-Tree induces disparate calculations.

Definition 8: Given a MC-Tree T and it's subtree T', T' is a maximum subtree iff the root of T' is a child of the root of T, denoted as $ms(T)$.

As is shown in Fig.1, in fact, there are as many maximum subtrees in MC-Tree as there are clauses in the clause set.

Theorem 7: The maximum subtrees in MC-Tree is equal to each other, and the processes of calculating their values are independent.

Proof: It is easy to find that there is an equivalence relation of nodes on the MC-Tree. The process of determining the value of nodes on the MC-Tree is not related, that is, they can be completed independently. Based on Definition 7, in addition to the root, the generation of any node is just related to its parent node. That means the nodes that are on the same path from the root to leaf are associated on generation. But calculating the value of each node is independent of any other. In addition, the maximum subtrees are identified as some equivalence classes that contain multiple nodes. Therefore, the maximum subtrees are equal and independent.

The MC-Tree of a clause set Φ may be regarded as the combination of its root and maximum subtrees that may be treated as distinct MC-Trees with smaller size. Let $T, ms_1(T), ms_2(T), \cdots, ms_n(T)$, respectively, denote the MC-Tree of Φ and its maximum subtrees, and the result of calculating MC-Tree T is denoted as $Cal(T)$. Then we have the formula:

$$Cal(T) = 2^m - \sum_{i=1}^{n} Cal(ms_i(T)) \tag{3}$$

Thereby, based on the Theorem 7, we can firstly find all the maximum subtrees of a MC-Tree, and then compute all maximum subtrees with smaller size in parallel. In fact, the maximum subtree T_i is the set of calculation items that contain P_i instead of $P_1, \cdots, P_{i-1}$ in formula (2). Therefore, computing all maximum subtrees with smaller size in parallel is actually computing the corresponding set of calculation items of formula (2) in parallel.

In fact, the size of each maximum subtree is not same, so their computation complexity is asymmetric. We can find that in a complete MC-Tree, the number of nodes in the first maximum subtree (according the order from left to right) is the sum of the nodes of the other maximum subtrees. Similarly, the number of nodes in the second maximum subtree is the sum of the nodes of the maximum subtrees except the first and the second one. The same rule is suitable for others. Therefore, it may be a case that the computation complexity of the initial maximum subtree is one half of the whole MC-Tree. However, any maximum subtree of $MC - Tree(ms(T))$ can be divided into its root R and maximum subtrees of it. We then obtain the following formula, similar to formula (3).

$$Cal(ms(T)) = Cal(R) - \sum_{i=1}^{n} Cal(ms_i(ms(T))) \tag{4}$$

Based on the formula (3) and (4), we can divide the $MC - Tree(T)$ into multiple nodes and a number of smaller subtrees instead of maximum subtrees of T, and compute them in parallelization. In the next section, we propose a parallelized algorithm in which the number of processors is a parameter in it.

B. PCER ALGORITHM

In this subsection, Algorithm parallel CER (PCER) implements the parallelized computation of #SAT in detail.

The number of processors is a parameter in Algorithm PCER. We can divide the algorithm into two main stages. The first stage comprises steps from 2 to 8 in the algorithm. In the first stage, compute the value of nodes on *Level* i ($0 \leq i < k$) and $k = \lceil \log_2 K \rceil$, K is the number of processors. The second stage consists of steps from 9 to 15. In this stage, compute the value of nodes on *Level* i ($k \leq i \leq n$). However, in implementation, we can parallelize

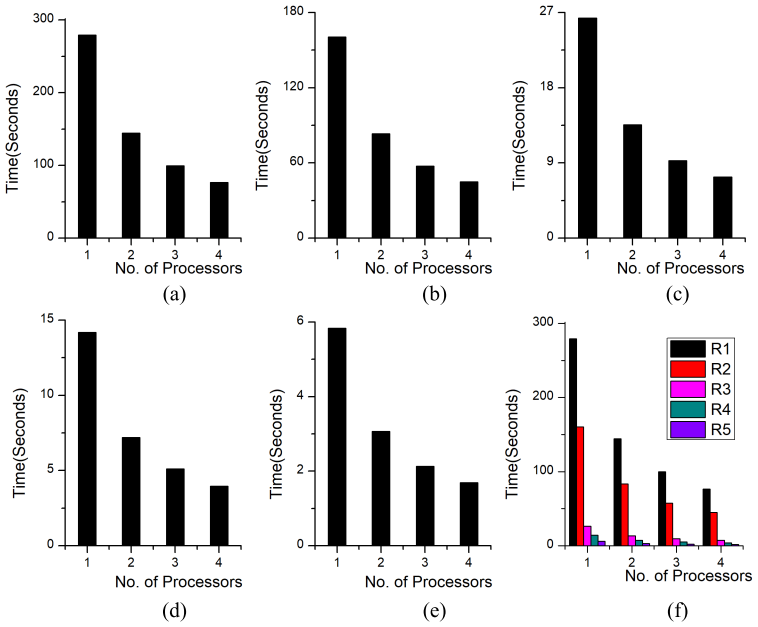


FIGURE 2. Running time of the PCER algorithm using different values of K at instances whose complementary factors are different. (a) R1. (b) R2. (c) R3. (d) R4. (e) R5. (f) R1, R2, R3, R4, R5.

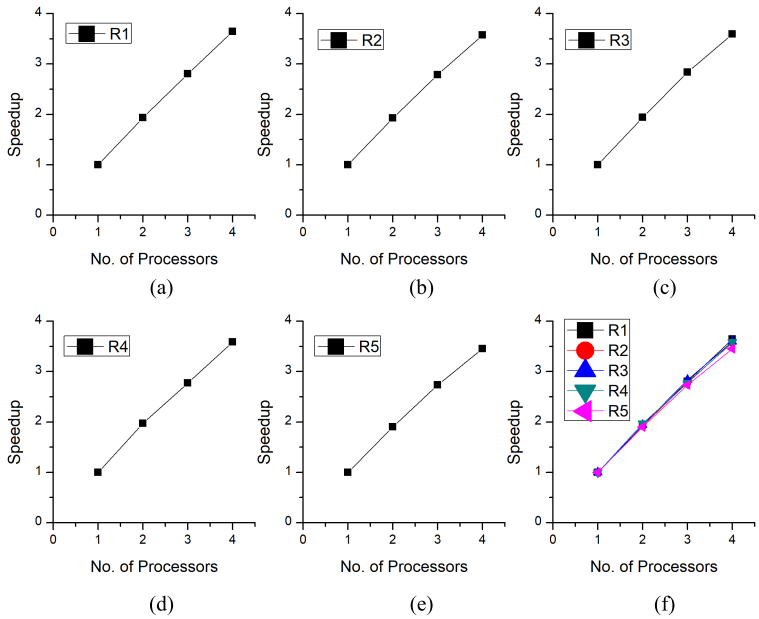


FIGURE 3. Scalability plot of parallel PCER at different instances. (a) R1. (b) R2. (c) R3. (d) R4. (e) R5. (f) R1, R2, R3, R4, R5.

the algorithm on a multi-core system by parallelizing the two stages. We parallelize the first stage by assigning one Level from 0 to $k-1$ to each processor dynamically and compute the value of nodes on each *Level* separately. The second stage can be parallelized by assigning one subtree to each processor dynamically in which its root is the node on the *Level* k.

Note that the variable *sum* denotes the number of all the maximum terms that could not be generated by using the extension rule. The final output is the number of models of the clause set Φ, and we initialize *sum* to be 0 in step 1. In steps 4 and 11, the *node* N is computed according to the formula (2). In addition, the number of $P_i(i = 1, 2 \cdots, n)$ in the node N is odd or is not determined whether the node is on the odd Level or not (step 12).

The algorithm PCER is another form of algorithm CER. In case of one processor, they are equivalent. The parallelization, which is implemented in the next section, is based on the algorithm PCER.

Algorithm 1 PCER (Model Counting in Parallelization)

Input: A clause set: $\Phi = \{C_1, C_2, \cdots, C_n\}$;
the set of variables: $X = \{x_1, x_2, \cdots, x_n\}$;
the number of processors: K;
Output: The number of the models of clause set Φ: *sum*

```
Init: sum ← 0; k ← ⌈log2 K⌉
for each Level i (0 ≤ i < k) do
    for each node N on Level i do
        Compute the value of N ;
        if i is an odd number then
            sum − = the value of N ;
        else
            sum + = the value of N;
for each subtree Ti (the root of Ti is the node on the Level k) do
    for each node N in Ti do
        Compute the value of N ;
        if i is on the odd level then
            sum − = the value of N ;
        else
            sum + = the value of N;
```

IV. EXPERIMENTAL RESULTS

Our experiments address the performance of the parallel version for the algorithm PCER. We implemented the algorithm on a Quad-core processor (3.4 GHz each) machine. The programs are coded in C language with OpenMP directives. In our experimental study, we consider the following.

- The input sets of clauses are randomly generated by a program with parameters m (the number of all variables), n (the number of all clauses), and p (the probability of any variable occurrences in each clause). Each variable occurs with a certain probability p in any clause. Variables appear in the form of positive variable with a certain probability p' and in the form of negative variable with probability $1 - p'$, the interval of p' is (0.1-0.5). We control the complementary factor of each instance by the parameters p and p'.
- The instances in our experiment contain 30 variables and 100 clauses. The CER algorithm is more efficient for the instances that have the higher complementary factor. When the complementary factor is high, CER algorithm shows a relatively good performance and the running time is less. To observe the experiment conveniently and see the improvement of the efficiency clearly, we select the instances whose complementary factors are approximately 0.2, 0.3, 0.4, 0.5 or 0.6.
- For each instance, we run the algorithm PCER using different number of processors, from $K = 1$ to 4.

TABLE 1. The parallel efficiency of PCER algorithm using different numbers of processors.

Instance	Complementary factor	Parallel efficiency		
		Two processors	Three processors	Four processors
R1	0.19	0.967	0.935	0.912
R2	0.32	0.963	0.929	0.893
R3	0.40	0.970	0.944	0.899
R4	0.54	0.986	0.925	0.897
R5	0.63	0.951	0.912	0.863

- For each instance, we get the running time of the parallel algorithm by calculating the average running time for 20 experiments.

Fig.2 shows the experimental study for the PCER algorithm using different values of K at different instances. R_1, R_2, R_3, R_4, and R_5 are the instances whose complementary factors are approximately 0.2, 0.3, 0.4, 0.5, and 0.6, respectively. From the results of the Fig.2, we observe the following remarks:

1) With the increase of the number of processors, the running time of PCER algorithm reduces in inverse proportion.
2) The higher complementary factor of instance is, the more efficient PCER algorithm is, in case of using each number of processors.

Fig.3 shows the scalability of PCER algorithm for instances R1, R2, R3, R4, and R5, respectively. The parallel algorithm reduces the running time of the sequential version, that is, $K = 1$, and the speedup achieved scales are linear with increasing the number of processors for all instances. All these proved that the approach has very good scalability.

We determine the parallel efficiency E_k of the PCER algorithm using K processors by

$$E_k = t_1/(t_k * k) \tag{5}$$

where t_1 represents the running time on a processor and t_k represents the running time using K processors. We can see the results of parallel efficiency from Table 1. There are two observations as follows.

1) As the number of processors increases, there has been a slight decrease in the parallel efficiency.
2) With the increase of complementary factor, the parallel efficiency slightly decreased.

V. CONCLUSION

In this paper, we introduced the exact algorithm CER, which is based on extension rule and proposed the notion of Model-Counting-Tree to parallelize the CER algorithm. We presented an efficient parallelization for exact sequential algorithm CER. The parallel algorithm was implemented on a Quad-Core machine using OpenMP. We compared the running time and speedup according to running the algorithm with different criteria. From the comparisons, we conclude the following:

1) With the increase of the number of processors, the running time of PCER algorithm reduces in inverse proportion.
2) The higher the complementary factor of instance is, the more efficient the PCER algorithm is, in case of using each number of processors.
3) The parallel algorithm reduces the running time of the sequential version, that is, K=1, and the speedup achieved scales are linear with increasing the number of processors for all instances. The parallel algorithm PCER has very good scalability.
4) The parallel efficiency of PCER algorithm performs excellently.

In future work, we will continue to improve the effectiveness of our algorithm. Although the parallelization method presented in this paper has been developed to solve #SAT, it is limited not only to this setting but also can be applied to different procedures that can be described on the data structure similar to the Mode-Counting-Tree. Also, we will try to use these proposed strategies into some other problems in the future.

ACKNOWLEDGMENT

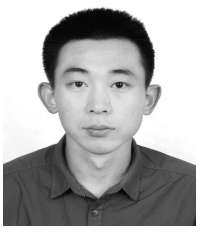
The authors would like to thank the anonymous referees for their valuable comments.

REFERENCES

[1] K. Xu and W. Li, "The SAT phase transition," *Sci. China E, Technol. Sci.*, vol. 42, no. 5, pp. 494–501, Oct. 1999.
[2] K. Xu and W. Li, "On the average similarity degree between solutions of random k-SAT and random CSPs," *Discrete Appl. Math.*, vol. 136, no. 1, pp. 125–149, Jan. 2004.
[3] J. Gao, M. Yin, and K. Xu, "Phase transitions in knowledge compilation: An experimental study," in *Proc. Int. Conf. Theory Appl. Satisfiability Test.*, Ann Arbor, MI, USA, 2011, pp. 364–366.
[4] C. Luo, S. Cai, W. Wu, and K. Su, "Double configuration checking in stochastic local search for satisfiability," in *Proc. 28th AAAI Conf. Artif. Intell.*, Québec City, QC, Canada, 2014, pp. 2703–2709.
[5] S. Cai and K. Su, "Comprehensive score: Towards efficient local search for SAT with long clauses," in *Proc. 23th Int. Joint Conf. Artif. Intell.*, Beijing, China, 2013, pp. 489–495.
[6] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu, "Symbolic model checking using SAT procedures instead of BDDs," in *Proc. 36rd ACM/IEEE Design Automat. Conf.*, San Jose, CA, USA, 1999, pp. 317–320.
[7] W. N. N. Hung, X. Song, G. Yang, J. Yang, and M. Perkowski, "Optimal synthesis of multiple output Boolean functions using a set of quantum gates by symbolic reachability analysis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 9, pp. 1652–1663, Sep. 2006.
[8] H. Kautz and B. Selman, "Unifying SAT-based and graph-based planning," in *Proc. 16th Int. Joint Conf. Artif. Intell.*, Stockholm, Sweden, 1999, pp. 318–325.
[9] J. Zhang, F. Ma, and Z. Zhang, "Faulty interaction identification via constraint solving and optimization," in *Proc. Int. Conf. Theory Appl. Satisfiability Test.*, Trento, Italy, 2012, pp. 186–199.
[10] L. Zhang, C. F. Madigan, M. H. Moskewicz, and S. Malik, "Efficient conflict driven learning in a Boolean satisfiability solver," in *Proc. Int. Conf. Comput. Aided Design*, New Orleans, LA, USA, 2001, pp. 279–285.
[11] S. A. Cook, "The complexity of theorem-proving procedures," in *Proc. 3rd Ann. ACM Symp. Theory Comput.*, Shaker Heights, OH, USA, 1971, pp. 151–158.
[12] C. Domshlak and J. Hoffmann, "Fast probabilistic planning through weighted model counting," in *Proc. 16th ICAPS*, Ambleside, U.K., 2006, pp. 243–252.
[13] T. Sang, P. Beame, and H. Kautz, "Solving Bayesian networks by weighted model counting," in *Proc. 20th Nat. Conf. Artif. Intell.*, Pittsburgh, PA, USA, 2005, pp. 475–482.
[14] F. Bacchus, S. Dalmao, and T. Pitassi, "Algorithms and complexity results for #SAT and Bayesian inference," in *Proc. 44th Annu. IEEE Symp. Found. Comput. Sci.*, Cambridge, MA, USA, Oct. 2003, pp. 340–351.
[15] J. Zhou, M. Yin, and C. Zhou, "New worst-case upper bound for #2-SAT and #3-SAT with the number of clauses as the parameter," in *Proc. 24th Nat. Conf. Artif. Intell.,* Atlanta, GA, USA, 2010, pp. 217–222.
[16] W. Wei and B. Selman, "A new approach to model counting," in *Proc. Int. Conf. Theory Appl. Satisfiability Test.*, Scotland, U.K., 2005, pp. 324–339.
[17] C. P. Gomes, J. Hoffmann, B. Selman, and A. Sabharwal, "From sampling to model counting," in *Proc. 20th Int. Joint Conf. Artif. Intell.*, Hyderabad, India, 2007, pp. 2293–2299.
[18] C. P. Gomes, A. Sabharwal, and B. Selman, "Model counting: A new strategy for obtaining good bounds," in *Proc. 21th Nat. Conf. Artif. Intell.*, Boston, MA, USA, 2006, pp. 54–61.
[19] L. Kroc, A. Sabharwal, and B. Selman, "Leveraging belief propagation, backtrack search, and statistics for model counting," *Ann. Oper. Res.*, vol. 184, no. 1, pp. 209–231, Apr. 2001.
[20] E. Birnbaum and E. L. Lozinskii, "The good old Davis–Putnam procedure helps counting models," *J. Artif. Intell. Res.*, vol. 10, no. 1, pp. 457–477, Jun. 1999.
[21] R. J. Bayardo, Jr., and J. D. Pehoushek, "Counting models using connected components," in *Proc. 17th Nat. Conf. Artif. Intell.*, Austin, TX, USA, 2000, pp. 157–162.
[22] T. Sang, F. Bacchus, P. Beame, H. Kautz, and T. Pitassi, "Combining component caching and clause learning for effective model counting," in *Proc. Int. Conf. Theory Appl. Satisfiability Test.*, Vancouver, BC, Canada, 2004, pp. 20–28.
[23] M. Yin, H. Lin, and J. Sun, "Counting models using extension rules," in *Proc. 22nd Nat. Conf. Artif. Intell.*, Vancouver, BC, Canada, 2007, pp. 1916–1917.
[24] W. Wei, J. Erenrich, and B. Selman, "Towards efficient sampling: Exploiting random walk strategies," in *Proc. 19th Nat. Conf. Artif. Intell.*, San Jose, CA, USA, 2004, pp. 670–676.
[25] M. Davis and H. Putnam, "A computing procedure for quantification theory," *J. ACM*, vol. 7, no. 3, pp. 201–215, Jul. 1960.
[26] L. Hai, S. Jigui, and Z. Yimin, "Theorem proving based on the extension rule," *J. Autom. Reasoning*, vol. 31, no. 1, pp. 11–21, Sep. 2003.

NAIYU TIAN received the B.S. degree from Jilin University in 2016, where she is currently pursuing the Ph.D. degree with the College of Computer Science and Technology. Her research interests include model-based diagnosis, propositional satisfiability (SAT), and #SAT.

DANTONG OUYANG received the M.S. degree and the Ph.D. degree in computer science from Jilin University in 1993 and 1998, respectively. She is currently a Professor with the College of Computer Science and Technology, Jilin University. Her current research interests include model-based diagnosis and automated reasoning.



FENGYU JIA received the M.S. degree from Jilin University in 2016. He is currently with China Asset Management Co., Ltd. His research interests include model-based diagnosis and #SAT.

MENG LIU received the B.S. degree from Jilin University in 2015, where she is currently pursuing the Ph.D. degree with the College of Computer Science and Technology through the successive postgraduate and doctoral program. Her research interests include model-based diagnosis, propositional satisfiability (SAT), and MaxSAT.

LIMING ZHANG received the M.S. degree and the Ph.D. degree in computer software and theory from Jilin University in 2004 and 2012, respectively. Since 2014, he has been a Senior Engineer with the College of Computer Science and Technology, Jilin University. His research interests include hitting set problem, model-based diagnosis, and propositional satisfiability.