J Inf Process Syst, Vol.13, No.5, pp.1149~1157, October 2017
https://doi.org/10.3745/JIPS.02.0071

ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

An Algorithm Solving SAT Problem Based on Splitting Rule and Extension Rule

Youjun Xu*

Abstract

The satisfiability problem is always a core problem in artificial intelligence (AI). And how to improve the efficiency of algorithms solving the satisfiability problem is widely concerned. Algorithm IER (Improved Extension Rule) is based on extension rule. The number of atoms and the number of clauses affect the efficiency of the algorithm IER. DPLL rules are helpful to reduce these numbers. Then a complete algorithm CIER based on splitting rule and extension rule is proposed in this paper in order to improve the efficiency. At first, the algorithm CIER (Complete Improved Extension Rule) reduces the scale of a clause set with DPLL rules. Then, the clause set is split into a group of small clause sets. In the end, the satisfiability of the clause set is got from these small clause sets'. A strategy MOAMD (maximum occurrences and maximum difference) for the algorithm CIER is given. With this strategy, a better arrangement of atoms could be got. This arrangement could make the number of small clause sets fewer and the scale of these sets smaller. So, the algorithm CIER will be more efficient.

Keywords

Extension Rule, IER, MOAMD Strategy, Satisfiability Problem, Splitting Rule

1. Introduction

Satisfiability problem [1] is always a core problem in artificial intelligence (AI). Methods for satisfiability problem are widely used in other fields, such as computer aided design [2,3], machine vision [4], database [5], and data mining [6]. There are many methods for satisfiability problem, such as methods based on tableau [7], methods based on resolution [8,9] and methods based on extension rule [10,11].

Methods based on resolution judge satisfiability of a clause set by resolution of clauses in the set. If an empty clause could be got in the resolution, the set is unsatisfied. Methods based on extension rule extend clauses to maximum terms and number these terms. Then judge the satisfiability of a clause set by the number of maximum terms. When the complementary factor [10] of a clause set is big, methods based on extension rule would be more efficient. Algorithm IER [10] is a method based on extension rule. IER first run an efficient incomplete algorithm to judge the satisfiability of a clause set. If the set is satisfiable, run the complete algorithm ER [10] to judge the satisfiability. A new complete algorithm CIER is proposed in this article. With the strategy maximum occurrences and maximum difference (MOAMD), it would be more efficient.

Manuscript received September 13, 2016; first revision February 8, 2017; accepted April 10, 2017.
Corresponding Author: Youjun Xu (xu_dqsy@163.com)
* College of Computer Science and Information Technology, Daqing Normal University, Daqing, China (xu_dqsy@163.com)

2. ER, IER

Definition 1 [10]: *Given a clause C and an atom set M, D={C ∨a, C ∨¬ a | a ∈M, a and ¬ a don't appear in C}, the deduction from C to D is called extension rule, and elements in D are called the results of extension rule.*

Theorem 1 [10]: *Given a clause set Σ and its atom set M(|M|=m). If all clauses in Σ are maximum terms, Σ is unsatisfiable iff there are 2^m different clauses in the set Σ.*

Comparing with methods based on resolution, methods based on extension rule are contrary derivation. Algorithm ER extends clauses to maximum terms and judge the satisfiability of a clause set by calculating the number of maximum terms extended from the set.

For calculating the number of maximum terms extended from a clause set, Lin et al. [10] proposed Formula 1.

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

$$\left|P_i\right| = 2^{m-|C_i|}$$

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

It could be seen from Formula 1 that efficiency depends on atom number *m* and clause number *n*. Therefore, efficiency could be improved by the reduction of *m* and *n*.

Lin et al. [10] proposed algorithm IER to improve efficiency. The algorithm constructs a clause, and deletes clauses containing complementary literals with the clause. A smaller clause set is got by deleting these clauses. If the answer could be got from the smaller set, IER is more efficiency. However, IER is not a complete algorithm. If the answer could not be got from the smaller set, algorithm ER has to be run.

As can be seen above, if algorithm IER could not find answer from the smaller set, the efficiency would not be improved. However, if answer could be got from another constructed smaller set, algorithm ER need not to be run. Furthermore, if a clause set is divided into several smaller sets, the algorithm would be efficiency and complete at the same time.

3. CIER

3.1 Improve Algorithm ER with DPLL

As mentioned above, the efficiency of algorithm ER depends on atom number m and clause number n. Therefore, the efficiency of algorithm ER could be improved by reducing m and n with DPLL rules [12,13].

Theorem 2 [14]: (tautology rule) *Deleting all tautology in clause set S, we got clause set S'. If S' is an empty set, S is satisfiable. Otherwise, S is unsatisfiable if and only if S' is unsatisfiable.*

With Theorem 2, not only clause number but also atom number could be reduced.

Algorithm TR(S):

Input: clause set S.

Output: clause set S containing no tautology.

1. While S contains tautology
2. Loop
3. Find a tautology C in S and delete it.
4. EndLoop

THEOREM 3 [12]: (single literal rule) *If a clause set S contains a single literal L, delete all clauses containing L and get another clause set S'. (1) If S' is an empty set, S is satisfiable. (2) If S' is not empty, delete literal ¬L in all clauses and get another clause set S''. S is unsatisfiable if and only if S'' is unsatisfiable.*

With theorem 3, clauses containing the single literal L are deleted and L is deleted from atom set. Therefore, clause number and atom number are reduced.

Algorithm SiR(S):

Input: clause set S.

Output: clause set S containing no literal L and ¬L.

1. While S contains single literals.
2. Loop
3. Find a single literal L in S.
4. Delete all clauses containing L in S.
5. Delete literal ¬L in all clauses.
6. EndLoop

DEFINITION 2 [12]: *Literal L is a pure literal, if and only if ¬L doesn't appear in the clause set S.*

THEOREM 4 [12]: (pure literal rule) *If a literal L in the clause set S is a pure literal, delete all clauses containing L and get another clause set S'. (1) If S' is an empty set, S is satisfiable. (2) Otherwise, S is unsatisfiable if and only if S' is unsatisfiable.*

With Theorem 4, all pure literals and clauses containing pure literals are deleted. So, clause number and atom number are reduced.

Algorithm PR(S):

Input: a clause set S.

Output: clause set S containing no pure literals.

1. While S contains pure literals.
2. Loop
3. Find a pure literal L in S.
4. Delete all clauses containing L.
5. EndLoop

THEOREM 5 [13]: (splitting rule) *Delete all clauses containing literal L is a clause set S and delete literal ¬L in all clauses containing ¬L. Then, a new clause set S_1 is got. If S_1 is empty, S is satisfiable. Otherwise, if S_1 contains empty clause, S_1 is unsatisfiable. Delete all clauses containing literal ¬L in a clause set S and delete literal L in all clauses containing L. Then, a new clause set S_2 is got. If S_2 is empty, S is satisfiable. Otherwise, if S_2 contains empty clause, S_2 is unsatisfiable. S is unsatisfiable if and only if S_1 and S_2 are all unsatisfiable.*

With Theorem 5, clause set S is divided into two smaller clause set S_1 and S_2. Comparing with S, S_1 and S_2 have smaller clause number and atom number. Continue using the theorem, more smaller clause sets would be got. If any one of these sets is satisfiable, S is satisfiable. Others don't need to be deduced. The satisfiability of these sets could be judged with ER. Like IER, if any one of these sets is satisfiable, the calculation would be ended. The difference from IER is that the method is complete. In the worst case, if S is unsatisfiable, all small clause sets need to be calculated. The method and ER have same algorithm complexity in the worst case.

THEOREM 6: *Splitting clause set S with split rule, the set of clause sets $\{S_1, S_2, S_3, \ldots, S_n\}$ would be got. If one of these sets is satisfiable, S is satisfiable. If all these sets are unsatisfiable, S is unsatisfiable.*

Proof: Split the clause set S with splitting rule and get two new clause sets S_{left} and S_{right}. Take S as the root of a binary tree, S_{left} as left child and S_{right} as right child. Continue splitting leaf sets with splitting rule and get a set of sets $\{S_1, S_2, S_3, \ldots, S_n\}$. Each S_i is a leaf set of the binary tree.

If one of these sets is satisfiable, his father set is satisfiable according to splitting rule. At last, the root set is satisfiable.

If two brother sets are unsatifiable, their father set is unsatisfiable according to splitting rule. Like this, if all leaf sets are unsatisfiable, their father sets are unsatisfiable, too. And at last, the root set is unsatisfiable.

Algorithm SpR(s,k):

Input: a clause set S, and a constant k, $1<k<2^m$

Output: a queue of clause sets Q, $|Q|=k$

1. Put S in Q
2. i = 1
3. While i<k Loop
4. Get a clause set S' from the head of Q.
5. Select a literal L from S'.
6. Delete literal L from clauses containing L and get a new clause set C_1.
7. Delete literal ¬L from clauses containing ¬L and get a new clause set C_2.
8. Put clauses containing no literal L and ¬L into the clause set C_3.
9. Put $C_1 \cup C_3$ to the tail of the queue Q.
10. Put $C_2 \cup C_3$ to the tail of the queue Q.
11. i=i+1
12. EndLoop

From Theorem 6, the satisfiability of clause set S could be judged by the satisfiability of series of clause sets $\{S_1, S_2, S_3, \ldots, S_n\}$. Therefore, the following algorithm CIER* could be used to judge the satisfiability of S. In the algorithm CIER*, ER(S) represent judging the satisfiability of S by algorithm ER. If S is satisfiability, ER(S) returns true or false.

Algorithm CIER*(S):
Input: a clause set S.
Output: "Satisfiable" if S is satisfiable, or else "UnSatisfiable".

```
1. Q = SpR(S, k)
2. While Q is not empty. Loop
3.    Take a clause set S' from the head of Q.
4.    If ER(S')==true then return Satisfiable
5. EndLoop
6. return UnSatisfiable
```

The number of small sets is decided by the constant k. The constant k is the times that splitting rule is used. Therefore, k needs to be reduced. The depth first search strategy could be used to reduce k. If an empty clause set is got in the construction of a binary tree, the tree is satisfiable. If a set containing empty clause is got in the construction, the subtree is unsatisfiable. If a leaf set is satisfiable, the construction could be stopped. The sequence of atoms which are used to split the tree also decides the efficiency of the method. More subtrees would be pruning if an appropriate sequence is adopted.

3.2 MOAMD

If split a clause set with an atom which has maximum occurrences, two sets as small as possible would be got. It is the MOAMD strategy.

MOAMD Strategy: take the atom with maximum occurrences as priority, and take the literal having more occurrences of the atom as priority.

The following algorithm constructs a literal queue with MOAMD strategy. In the algorithm, f(L) represents the occurrence number of literal L. M is the set of atoms in clause set S and Mq is the set of atoms in literal array QL.

Algorithm MOAMD(S, k):
Input: a clause set S and a constant k.
Output: a literal array QL.

```
1. i=1
2. while i<k  Loop
3.    Find an atom in M-Mq whose f(L)+f(¬L) is largest.
4.    If f(L)>f(¬L) then QL[i]=L
5.    Else QL[i]= ¬L
6.    i = i +1
7. EndLoop
```

Taking the atom with maximum occurrences as priority, more subtrees would be pruning in small depth and get clause sets as small as possible. Therefore, the height of the tree would be reduced and the efficiency of the method would be improved.

Algorithm CIER(S):

Input: a clause set S.

Output: "Satisfiable" or "UnSatisfiable"

1. S=TR(S)
2. If S is an empty set, return "Satisfiable".
3. S=SiR(S)
4. If S is an empty set, return "Satisfiable".
5. If S contains empty clauses, return "UnSatisfiable".
6. S=PR(S)
7. If S is an empty set, return "Satisfiable".
8. QL=MOAMD(S,k)
9. Construct a binary search tree with atoms in QL by the order they are in QL.
10. While there are leaf sets still not being judged. Loop
11. i=1
12. While i<=k Loop
13. L=QL[i]
14. Delete clauses containing literal L in S and get a new set S'.
15. If S' is an empty set, return "Satisfiable"
16. Delete literal ¬L from clauses containing ¬L in S'.
17. If S' contains empty clauses, prune the set.
18. If S' has a right brother, put literals on the path into QL. Goto 10
19. Else return "UnSatisFiable"
20. i=i+1
21. EndLoop
22. If ER(S')==true then return "Satisfiable"
23. EndLoop
24. Return "UnSatisFiable"

THEOREM 7: *Algorithm CIER is correct and complete.*

Proof: From the correctness of algorithm ER, the satisfiability of all small sets could be judged correctly. Based on Theorem 6, clause set S is satisfiable if one of these small sets is satisfiable. And if all small sets are unsatisfiable, S is unsatisfiable. Therefore, the theorem is correct.

When S is satisfiable and the algorithm doesn't stop in step 2, 4, 7, assume that all these small sets are unsatisfiable. From Theorem 6, S is unsatisfiable. Therefore, the assumption is wrong. At least one of these sets is satisfiable. At last, the algorithm would stop in step 15 or 22 and return "Satisfiable". When S is unsatisfiable and the algorithm doesn't stop in step 5, easy to know that all small sets are unsatisfiable. After judging the satisfiability of every small set, the algorithm would stop in step 24 and return "UnSatisfiable". Therefore, the theorem is complete.

4. Experiment

Algorithm CIER with strategy MOAMD is tested and compared with algorithm ER and IER. Clause sets used in the test are generated randomly. And every data in figures is the mean of fifty test data.

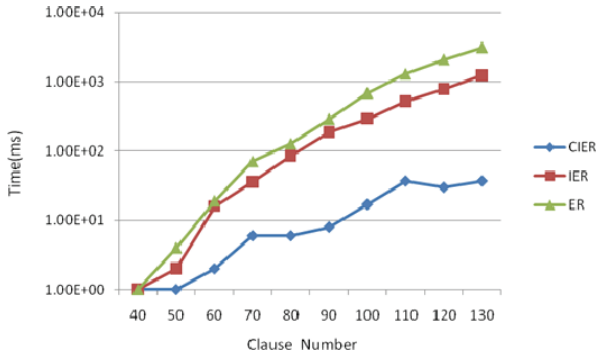

Fig. 1. Experimental results of clause sets whose atom number is 20.

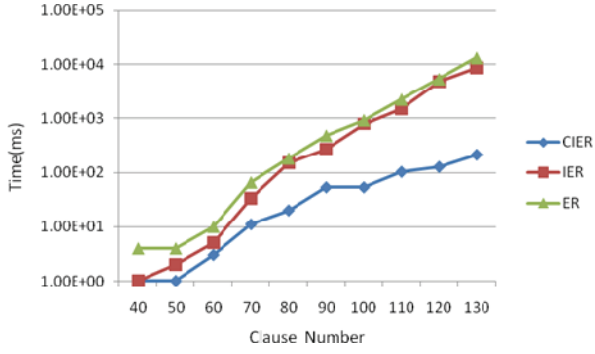

Fig. 2. Experimental results of clause sets whose atom number is 30.

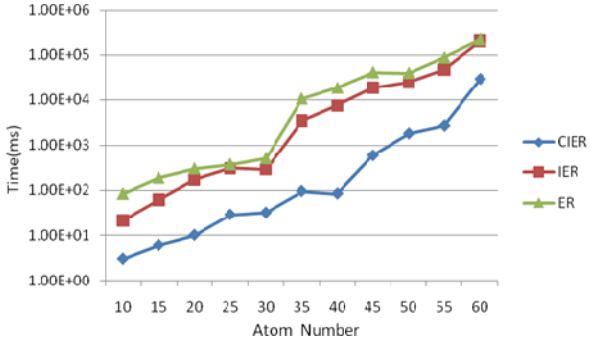

Fig. 3. Experimental results of clause sets whose clause number is 90.

In Figs 1–3, the maximal length of clauses is 10. As shown in Figs. 1–3, since algorithm IER doesn't use heuristic information, the efficiency of algorithm IER has not a good improvement comparing with algorithm ER. However, algorithm CIER uses MOAMD strategy and splitting rule to reduce the

number of small sets and the number of clauses. The efficiency of algorithm CIER has a good improvement comparing with algorithm ER and IER.

5. Conclusion

In conclusion, algorithm CIER uses splitting rule to split a clause set into some small sets. The method keeps the algorithm complete and effective. The use of MOAMD strategy reduces the scale of clause sets need to be calculated and makes pruning happen in shallow depth of the binary search tree. All of the above methods make the algorithm CIER more effective than ER and IER, just as shown in the experiment.

Acknowledgement

This paper is fully supported by Scientific Research Fund (Grant No. 12ZR08) of Daqing Normal University.

References

[1] I. Kanja and S. Szeider, "Parameterized and subexponential-time complexity of satisfiability problems and applications," *Theoretical Computer Science*, vol. 607 (Part 3), pp. 282-295, 2015.

[2] J. Han, Z. Jin, and B. Xia, "Proving inequalities and solving global optimization problems via simplified CAD projection," *Journal of Symbolic Computation*, vol. 72, pp. 206-230, 2016.

[3] D. Kaiss, M. Skaba, Z. Hanna, and Z. Khasidashvili, "Industrial strength SAT-based alignability algorithm for hardware equivalence verification," in *Proceedings of the Formal Methods in Computer Aided Design*, Austin, TX, 2007, pp. 20-26.

[4] J. Rintanen, "Planning as satisfiability: heuristics," *Artificial Intelligence*, vol. 193, pp. 45-86, 2012.

[5] C. Carapelle, A. Kartzow, and M. Lohrey, "Satisfiability of ECTL* with constraints," *Journal of Computer and System Sciences*, vol. 82, no. 5, pp. 826-855, 2016.

[6] W. Wu and M. S. Hsiao, "SAT-based state justification with adaptive mining of invariants," in *Proceedings of the IEEE International Test Conference*, Santa Clara, CA, 2008, pp. 1-10.

[7] H. Kurokawa, "Tableaux and hypersequents for justification logics," *Annals of Pure and Applied Logic*, vol. 163, no. 7, pp. 831-853, 2012.

[8] J. A. Robinson, "A machine-oriented logic based on the resolution principle," *Journal of the ACM*, vol. 12, no. 1, pp. 23-41, 1965.

[9] A. R. KhudaBukhsh, L. Xu, H. H. Hoos, and K. Leyton-Brown, "SATenstein: automatically building local search SAT solvers from components," *Artificial Intelligence*, vol. 232, pp. 20-42, 2016.

[10] H. Lin, J. Sun, and Y. Zhang, "Theorem proving based on the extension rule," *Journal of Automated Reasoning*, vol. 31, no. 1, pp. 11-21, 2003.

[11] Y. Li, J. G. Sun, X. Wu, and X. J. Zhu, "Extension rule algorithms based on IMOM and IBOHM heuristics strategies," *Journal of Software*, vol. 20, no. 6, pp. 1521-1527, 2009.

[12] M. Davis and H. Putnam, "A computing procedure for quantification theory," *Journal of the ACM*, vol. 7, no. 3, pp. 201-215, 1960.

[13] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem-proving," *Communications of the ACM*, vol. 5, no. 7, pp. 394-397, 1962.
[14] X. H. Liu, *Automated Reasoning Based on Resolution Methods*. Beijing: Science Press, 1994.

Youjun Xu

He received the Master degree in computer application from Jilin University in 2005 and the Ph.D. degree in computer software and theory from Jilin University in 2011. Now he is a lecturer at College of Computer Science and Information Technology, Daqing Normal University, China. His main interests include Automated Reasoning, Internet of Things.