An Algorithm Based on Hitting Set for SAT Problem

Youjun Xu
School of Computer Science and Information Technology,
Daqing Normal University
Daqing, China
xu_dqsy@163.com

Yingrui Ma*
School of Computer Science and Information Technology,
Daqing Normal University
Daqing, China
E-mail: mayingrui-1@163.com

Zi Li
School of Computer Science and Information Technology,
Daqing Normal University
Daqing, China
Lizi_daqing@163.com

***Abstract*—The satisfiability problem is always a core problem in artificial intelligence and has been concerned for decades. Achievements in the research of the satisfiability problem have already been used in the research of other areas, such as computer aided design, database system. Now, there are many methods for the satisfiability problem, such as methods based on resolution, methods based on tableau, methods based on extension rule. By the study of extension rule, a conclusion can be drawn that the satisfiability of a clause set can be judged by the existence of a hitting set, which containing no complementary pairs of literals. With this conclusion, the algorithm CBHST (complementary binary hitting set tree) is designed. CBHST judge the existence of a hitting set without complementary pairs of literals in a binary tree. If the hitting set exists, CBHST will give the result that the clause set is satisfiable. Finally, CBHST is compared with an algorithm based on resolution DR and an algorithm based on extension rule IER. The test data shows that CBHST is efficient.**

Keywords—satisfiability problem, hitting set, extension rule, complementary factor, complementary binary hitting set tree

I. INTRODUCTION

Satisfiability(SAT) Problem[1] is always a core problem in artificial intelligence and has been concerned for decades. Achievements in the research of the SAT problem have already been used in the research of other areas, such as computer aided design [2,3], database system[4], computer vision[5], model diagnosis[6]. SAT Problem is a NP-complete problem. It's hard to get an efficient algorithm which can judge the satisfiability of any SAT Problem.

The main methods for SAT problem include methods based on resolution [7,8], methods based on tableau[9], and methods based on extension rule[10,11]. Robinson proposed resolution principle [7]. The resolution principle is known to be a basic tool for theorem proving. Methods based on resolution try to conclude the unsatisfiability of a clause set by deducing an empty clause. Davis and Putnam proposed the algorithm DP[12]. Soon, based on the work, Davis,Logemann and Loveland proposed the algorithm DPLL[13]. It's a brilliant work in the study of methods based on resolution. Most resolution algorithms is based on the study of DPLL. Algorithm DR[14] proposed by Dechter and Rish is one of these algorithms. Algorithm DR sorts atoms in a clause set, and resolves clauses by this order. Many redundant clauses would be added into the clause set in this process.

Methods based on tableau focus on the binary relation in clause sets. They describe binary relation with logic formulas and have advantages in logic without equality [15, 16]. Tableau was proposed by Beth in 1955. After that, Smullyan and Fitting extended his work. Today, these methods are widely accepted as reasoning methods.

Extension rule is proposed by Lin and develop rapidly in recent years [10]. Methods based on extension rule judge a clause set's satisfiability by numbering the maximum terms. Then, Lin proposed algorithm IER witch is efficient when complementary factor of a clause set is large.

In the study of extension rule, we find that SAT problems could be transferred into hitting set problems. Algorithm CBHST proposed in this paper needn't add redundant clauses in clause sets and needn't numbering the maximum terms. Therefore, it is efficient.

II. HITTING SET

DEFINITION 1[17]: Set H is called the hitting set of a set of clause sets C if and only if, and for each. If H is minimal, it is called minimal hitting set

Karp proposed first hitting set problem [18]. After that, Reiter advanced HS-tree algorithm for calculating minimal hitting sets of a conflict set. However, maybe some answers are missed in the calculating. This would not happen in algorithm HS-DAG[19] proposed by Greniner. Ouyang improved the searching process. Less nodes would be searched in algorithm NEWHS-tree. So, it is more efficient.

Calculating all the minimal hitting sets is a NP-complete problem. However, judging the existence of hitting sets is not a NP-hard problem, if hitting sets needn't to be minimal.

III. EXTENSION RULE

Extension rule proposed by Lin[10] is a method for solving SAT problem. It's a method different from methods based on resolution. Methods based on resolution try to conclude the unsatisfiability of a clause set by deducing an empty clause. In this process, clauses contain complementary literals would be resolved and the number of clauses would be reduced. However, methods based on would increase the number of clauses and conclude the unsatisfiability of a clause set by counting the number of maximum terms.

DEFINITION 2[10]: Given a clause C and a set M,

978-1-7281-2598-5/19/$31.00 ©2019 IEEE

C' = {C∨L, C∨¬L | L is an atom, L∈M, and L does not appear in C}.

We call the operation proceeding from C to C' the extension rule on C. We call C' the result of the extension rule.

Given a clause set whose number of atoms is m, any clause in the set could be extended to maximum terms. Atoms in this clause set have two forms, positive literal and negative literal. For each maximum term, there is a truth assignment which makes truth values of all literals in the term false. As the result, the truth assignment makes the truth value of the term false. There are 2^m different truth assignments at most form atoms. Therefore, a clause set is unsatisfiable if 2^m different maximum terms could be extended from the set, since that none of the 2^m truth assignments could satisfy all the maximum terms.

THEOREM 1[10]: Given a set of clauses S with its set of atoms M (|M|=m), if the clauses in S are all maximum terms on M, then S is unsatisfiable iff it contains 2^m clauses.

Extension rule could be used to judge the satisfiability of a clause set. The efficiency of the method depends on the complementary factor of the clause set. When the complementary factor is big, the method is more efficient.

DEFINITION 3[10]: Given a clause set S={C_1, C_2, …,C_n}, complementary factor F is the ratio of the number of pairs of clauses containing complementary literals and the number of all clauses.

$$F = \frac{K}{n*(n-1)/2} \tag{1}$$

K is the number of pairs of clauses containing complementary literals.

Given a pair of clauses containing A and¬A, it could be conclude from definition 2 that maximum terms extended from the clause contain A will not contain ¬A. Like that, maximum terms extended from the clause contain¬A will not contain A. Thus, the two clauses would not extend a same maximum term.

THEOREM 2[10]: The intersection of two maximum term sets come from two clauses is empty if the two clauses containing complementary literals.

From theorem 1, it could be concluded that a clause set is unsatisfiable if 2^m different maximum terms could be extended from the clause set. That is to say, if there is a maximum term could not be extended from any clause in the clause set, the clause set is satisfiable. As shown in theorem 2, a maximum term could not be extended from a clause set if the maximum term containing complementary literals with any clause is the clause set. Therefore, the clause set could not extend all maximum terms. And the clause set is satisfiable.

THEOREM 3: A set of clauses S is satisfiable if there is a maximum term C_M, and C_M contains at least one complementary literal for each clause in S.

Proof:

Let S is a satisfiable set of clauses. According to theorem 1, it is easy to know that there is a maximum term which can't be extended from S. There is no harm to call the term C_M. If there is a clause C in S containing no complementary literals for C_M, each literal in C_M but not appears in C could be extended from C according to definition 2. It conflicts with the conclusion that C_M can't be extended from S. So, C_M contains at least one complementary literal for each clause in S.

If there is a maximum term C_M, C_M contains at least one complementary literal for each clause in S. It's easy to know that C_M can't be extended from any clause in S. That is to say, C_M can't be extended from S. So, according to theorem 1, S is satisfiable.

IV. Converting from the Satisfiability to the Existence of Hitting Set

It could be concluded from theorem 3 that a clause set is satisfiable if there exists a maximum term C_M containing complementary literals with each clause in the clause set. Another maximum term C_M' could be built which is composed with complementary literals of each literal in C_M. For C_M containing complementary literals with each clause in the clause set, C_M' and each clause have same literals. Clause is the set of literals. Therefore, C_M' is a hitting set of the clause set.

COROLLARY 1: A set of clauses S is satisfiable if there is a hitting set $S_{hitting}$ of S and $S_{hitting}$ doesn't contain complementary literals.

Proof:

If clause set S is satisfiable, there must be a maximum term C_M and C_M contains complementary literal for each clause in S. Then we can construct a maximum term C_M' composing with complementary literal of each literal in C_M. Since each clause in S has at least one pair of complementary literals with C_M, and C_M' has the same pair of complementary literals with C_M. Each clause in S has at least one same literal with C_M'. That is to say, C_M' is a hitting set of the clause set S. Because C_M' is a maximum term, it doesn't contain any pair of complementary literals.

If a clause set S has a hitting set $S_{hitting}$ and $S_{hitting}$ doesn't contain complementary literals, we could construct a clause C with each literal in $S_{hitting}$. Then C is extended to a maximum term C_M'. Obviously, C_M' has at least one same literal with each clause in S. From C_M', we could get a new maximum term C_M composing with complementary literal of each literal in C_M'. So, C_M and each clause in S contain at least one pair of complementary literals. From theorem 3, S is satisfiable.

According to corollary 1, the satisfiability of a clause set could be judged by the existence of hitting set containing no complementary literals. That is to say, the satisfiability problem of a clause set could be regarded as the existence of a hitting set of the clause set. Calculating all the minimal hitting sets is a NP-complete problem. However, when a clause set is satisfiable, there must be a hitting set. And the hitting set might be found quickly.

V. Algorithm CBHST

Complementary binary hitting set tree(CBHST) could be used to judging the existence of a hitting set containing no complementary literals of a clause set. CBHST is a binary tree. The root is marked with the clause set S which need to be judged the satisfiability. Then choose an atom A, and mark the edge linking left child with the positive literal A. And mark the other edge linking right child with negative literal ¬A. Delete all clauses containing A in S and delete ¬A in clauses containing ¬A. Then we get the clause set S_{left}. Mark left child with S_{left}. If S_{left} is empty, mark left child with √ instead. Or mark left child with × if there is an empty clause in S_{left}. S_{right} is a set which is constructed like S_{left}. Delete all clauses containing ¬A in S and delete A in clauses containing A. Mark right child with S_{right}. Otherwise, mark right child with √ if S_{right} is empty or mark with × if S_{right} contains empty clause. Continually extend the tree until all leaves are marked with √ or ×.

Obviously, the set of literals which are used to mark edges from root to the leaf marking with √ is a hitting set of S. Truth assignments would lead to the unsatisfiability of one clause in S, which make literals marking from root to the leaf × true. Therefore, S is satisfiable if there is a leaf marking with √ and unsatisfiable if all leaves are marked with ×.

THEOREM 4: A set of clauses S is satisfiable iff there is a leaf marking with √ in CBHST.

Proof:

If there is a leaf marking with √ in CBHST. As is known, √ represents an empty clause set. That is to say, every clause in S containing at least one literal which is used to mark edges from root to leaf √. The set of literals marking from root to leaf √ is a hitting set of S. Because an atom only occur in one layer, the hitting set containing no complementary literals. According to corollary 1, S is satisfiable.

Let S is satisfiable. According to corollary 1, there must be a hitting set. Therefore, the path marking with all literals in the hitting set leads to a leaf marking with √.

EXAMPLE: Judge the satisfiability of clause set S = { {A,C,¬D}, {B,¬C}, {D}, {A,C,D}, {¬A,¬B, ¬D} }.

The CBHST of S is constructed as Figure 1.

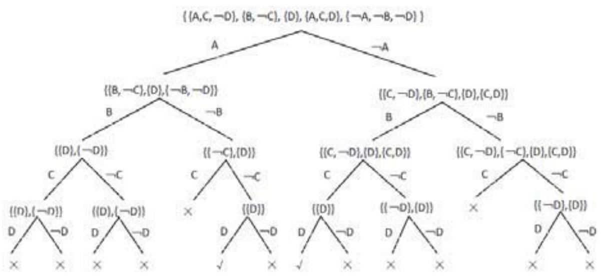

Fig. 1. CBHST of the clause set S

In Figure 1, two leaves are marked with √. Therefore, there are two hitting sets, {A, ¬B, ¬C,D} and {¬A,B,C,D}. According to corollary 1, S is satisfiable.

As show in Figure 1, although the answer is right, too many nodes in CBHST need to be extended. Therefore, strategies need to be adopted. Order atoms by the number of occurrences. Use atom with big number to mark edges in shallow depth. Then, more clauses would be filtered soon. Fewer nodes need to be extended and smaller sets would be used to mark these nodes.

In this example, the number of occurrences of atoms are D(four times), A(three times), C(three times), B(two times). As show in Figure 2, construct CBHST of S by this order.

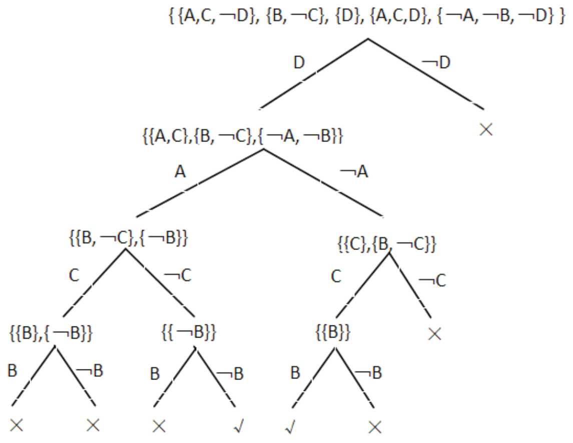

Fig. 2. CBHST of the clause set S after ordering atoms

There are 27 nodes in figure 1 and only 15 nodes in Figure 2. Moreover, clause sets which are used to mark these nodes are smaller.

Constructing CBHST by the way shown in Figure 1 and Figure 2, all hitting sets will be found. However, we only need to judge the satisfiability of S. Therefore, not all nodes need to be extended in CBHST. Leaf marking with √ might be found quickly if CBHST is extended in depth-first order. By this way, only one leaf marking with √ would be found and many nodes are still not extended. When a node is extended, choose the child marking with small set as the next node to be extended. Because the set is small, the path would be ended with √ or × earlier.

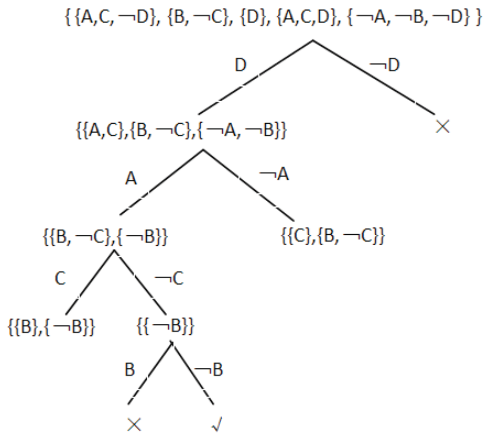

Fig. 3. CBHST of the clause set S taking strategiesof depth first and smaller set first

In Figure 3, CBHST is constructed in depth-first order and choose small set as priority. There are only nine nodes and some of them are not extended. The number of nodes is less than the number in Figure 2.

If a path end with leaf marking with ×, the constructing of CBHST need backtracking. For saving time, nodes needing to be extended are pushed on the stack. Because

CBHST is constructed in depth-first order, only one node need to be pushed on the stack in every layer. Therefore, if a clause set has m atoms, m-1 nodes need to be pushed on the stack at most. The algorithm needn't too much space.

ALGORITHM CBHST:

Input: a clause set S.

Output: "Satisfiable" or "Unsatisfiable".

```
1) Sorts atoms in S by the number of occurrences and arranges them in array.
2) Pushes S on the stack
3) While stack is not empty
4)         $S_{current}$ = pops a clause set from the stack
5)         Select an atom v and v is an atom in $S_{current}$. Occurrence number of v is highest.
6)         Deletes clauses containing literal V in $S_{current}$ and literal ¬V in others clauses. Then we get the clause set $S_1$.
7)         If $S_1$ is empty, output "Satisfiable".
8)         Deletes clauses containing literal ¬V in $S_{current}$ and literal V in others clauses. Then we get the clause set $S_2$.
9)         If $S_2$ is empty, output "Satisfiable".
10)        If $S_1$ and $S_2$ contain no empty clause, and if $S_1$'s clause number is not less than $S_2$'s, pushes $S_2$ on the stack before pushing $S_1$ on the stack. Otherwise, pushes $S_1$ before $S_2$.
11)        If $S_1$ contain empty clause and $S_2$ contain no empty clause, pushes $S_2$ on the stack.
12)        If $S_2$ contain empty clause and $S_1$ contain no empty clause, pushes $S_1$ on the stack.
13) Endwhile
14) output "UnSatisfiable".
```

THEOREM 5: Algorithm CBHST is correct and complete.

Proof:

Nodes need to be extended are pushed on the stack. It is easy to know from theorem 4 that a leaf marking with √ would be found before stack is empty if S is satisfiable. So, the algorithm would be ended at step 7 or step 9 and return "Satisfiable".

If S is unsatisfiable, from theorem 4, leaf marking with √ doesn't exist. All non-leaf nodes would be extended before stack is empty and stack would be empty at last. So, the loop will stop. The algorithm would be ended at step14 and return "Unsatisfiable".

Therefore, the algorithm is correct.

As mentioned above, for any clause set S, algorithm CBHST would stop. The algorithm is complete.

VI. EXPERIMENT

Algorithm CBHST is tested and compared with algorithm DR and algorithm IER. Algorithm DR is a famous algorithm based on resolution. And algorithm IER is an efficient algorithm based on extension rule. All data are random generated. All numbers shown in follow figures and tables are means of fifty results.

When atom number is 20, length of clause is 3, clause number is between 40 and 130, the result is shown is Figure 4.

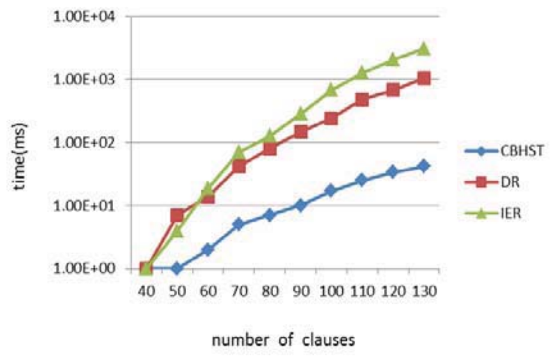

Fig. 4. Experimental results when the clause sets' numberof atoms is 20 and the length of clauses is 3

When atom number is 30, length of clause is between 6 and 10, clause number is between 40 and 130, the result is shown in Figure 5.

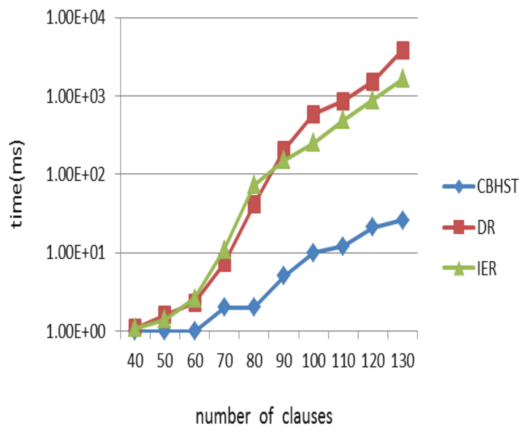

Fig. 5. Experimental results when the clause sets' numberof atoms is 30 and the length of clauses is 6~10

The experiment in Figure 4 tests short clauses. And the experiment in Figure 5 tests long clauses. As shown in Figure 4 and Figure 5, algorithm CBHST is more efficient than DR and IER.

The length of clause in Figure 5 is longer than in Figure 4. So, more clauses are filtered by one literal in Figure 5 than in Figure 4. With same atom number, less nodes are extended in Figure 5 than in Figure 4. Therefore, even though atom number in Figure 5 is bigger than in Figure 4, the time consumed in Figure 5 is shorter than in Figure 4.

The numbers shown in Table I and Table II are numbers of nodes extended by CBHST. Each number shown in these tables is the mean of fifty test results with the same parameters. The length of clause in Table I and Table II is different. The length in Table I is 3. And the length in Table II is between 6 and 10.

TABLE I. NUMBER OF NODES EXTENDED BY CBHST WHEN THE LENGTH OF CLAUSES IS 3

Clause number / Atom number	40	50	60	70	80	90	100	110	120	130
20	21	24	25	27	29	27	29	29	30	29
25	25	28	30	33	33	36	34	36	37	36
30	29	32	33	37	39	40	43	42	43	43
35	31	35	37	42	44	46	47	48	49	49
40	34	39	40	47	48	50	51	54	54	57
45	37	47	48	50	51	56	58	57	60	62
50	40	44	46	53	55	61	62	63	66	67
55	43	51	57	56	63	68	68	69	70	73
60	46	50	52	57	64	72	69	71	75	76
65	47	52	63	60	76	75	73	75	80	87
70	47	55	57	62	77	76	83	79	89	89

TABLE II. NUMBER OF NODES EXTENDED BY CBHST WHEN THE LENGTH OF CLAUSES IS 6~10

Clause number / Atom number	40	50	60	70	80	90	100	110	120	130
20	14	15	16	17	17	18	18	19	19	20
25	17	19	19	21	21	21	22	24	23	23
30	19	21	22	23	27	26	26	26	27	27
35	21	24	25	26	27	28	37	30	30	31
40	24	26	28	29	30	32	32	34	34	35
45	26	28	31	32	34	34	38	37	38	39
50	28	31	33	35	36	38	39	40	41	42
55	29	33	35	38	39	41	42	43	45	48
60	31	35	38	40	42	44	45	49	50	51
65	33	37	41	42	45	47	48	50	51	53
70	34	39	42	45	48	50	51	53	55	56

From Table I and Table II, with the increase of atom number and clause number, nodes needed to be extended increase. With same atom number and clause number, the number of occurrences of a literal is small when clause length is short. And the number of clause filtered by a literal is small, too. Therefore, more nodes needed to be extended.

VII. CONCLUSION

From the study of extension rule, the satisfiability problem of a clause set could be converted to the existence of a hitting set containing no complementary literals of the clause set. And tests show that algorithm CBHST is more efficient than DR and IER.

ACKNOWLEDGEMENT

This work was fully supported by Natural Science Foundation of Heilongjiang Province (F201128), the Natural Science Foundation of Daqing Normal University (18kyz017). *Yingrui Ma is Corresponding author.

REFERENCES

[1] I. Kanja and S. Szeide, “Parameterized and subexponential-time complexity of satisfiability problems and applications”, Theoretical Computer Science, vol.607, (2015), pp. 282-295.

[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, (2016), pp. 206-230.

[3] D. Kaiss, M. Skaba, Z. Hanna and Z. Khasidashvili, “Industrial strength SAT-based alignability algorithm for hardware equivalence verification”, Formal Methods in Computer Aided Design Austin, TX, USA, (2007) November 11 – 14.

[4] C. Carapelle, A. Kartzow and M. Lohrey, “Satisfiability of ECTL with constraints”, Journal of Computer and System Sciences, vol.82, (2016), pp. 826-855.

[5] J. Rintanen, “Planning as satisfiability: Heuristics”, Artificial Intelligence, vol.193, (2012), pp.45-86.

[6] D. Borrego and I. Barba, “Conformance checking and diagnosis for declarative business process models in data-aware scenarios”, Expert Systems with Applications, vol.41, (2014), pp.5340-5352.

[7] J. A. Robinson, “A machine oriented logic based on the resolution principle”, Journal of the ACM, vol.12, (1965), pp.23-41.

[8] 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, (2016), pp.20-42.

[9] H. Kurokawa, “Tableaux and hypersequents for justification logics”, Annals of Pure and Applied Logic, vol.163, (2012), pp.831-853.

[10] H. Lin, J. Sun and Y. Zhang, “Theorem proving based on the extension rule”, Journal of Automated Reasoning, vol.31, (2003), pp.11-21.

[11] L. Ying, S. Jigui and W. Xia, “Extension rule algorithms based on IMOM and IBOHM heuristics strategies”, Journal of Software, vol.20, (2009), pp.1521-1527.

[12] M. Davis and H. Putnam, “A computing procedure for quantification theory”, Journal of the ACM, vol.7, (1960), pp.201-215.

[13] M. Davis, G. Logemann and D. Loveland, “A machine program for theorem proving”, Communications of the ACM, vol.5, (1962), pp.394-397.

[14] R. Dechter and I. Rish, “Directional resolution: The Davis-Putnam procedure, revisited”, Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning, (1994), pp.134-145.

[15] R. M. Smullyan, Editor, “First-order logic”, Springer-Verlag, New York, (1994).

[16] M. Fitting, Editor, “First-order logic and automated theorem proving”, Springer- Verlag, New York, (1996).

[17] R. Reiter, “A theory of diagnosis from first principles”, Artificial Intelligence, vol.32, (1987), pp.57-96.

[18] R. M. Karp, “Complexity of Computer Computations”, Edited M. Thacher, Plenum press, New York, (1972), pp.85-103.

[19] R. Greiner and B. A. Smith, “A correction to the algorithm in Reiter’s theory of diagnosis”, Artificial Intelligence, vol.41, (1989), pp.79-88.

[20] D. Ouyang, J. Ouyang, X. Cheng and J. Liu, “A method of computing hitting set in model-based diagnosis”, Chinese Journal of Scientific Instrument, vol.25, (2004), pp.605-608.

[21] Y. Jiang and L. Lin, “The computation of hitting sets with Boolean formulas”, Chinese Journal of Computers, vol.26, (2003), pp. 919-924.

[22] A. Cincotti, V. Cutello and F. Pappalardo, “An ant-algorithm for the weighted minimum hitting set problem”, Proceedings of the 2003 IEEE Swarm Intelligence Symposium”, New York, USA, (2003) April 24-26.

[23] N. Shi and C. Chu, “Fast parallel molecular solution to the hitting-set problem”, Proceedings of the 2008 Eighth International Conference on Intelligent Systems Design and Applications, Kaohsiung, Taiwan, (2008) November 26-28.

[24] S. Vinterbo and A. Ohrn, “Minimal approximate hitting sets and rule templates”, International Journal of Approximate Reasoning, vol.25, (2000), pp.123-143.