A method of combining SE-tree to compute all minimal hitting sets*

ZHAO Xiangfu and OUYANG Dantong**

(School of Computer Science and Technology, Jilin University, Changchun 130012, China; Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Changchun 130012, China)

Received April 19, 2005; revised September 16, 2005

Abstract In model-based diagnosis, the candidate diagnostic results are generally characterized by all minimal hitting sets for the collection of all conflict sets. In this paper, a new method is proposed to judge a hitting set by the number of conflict sets corresponding to components, and the computing procedure is formalized by combining revised SE-tree (set enumeration tree) with closed nodes to generate all minimal hitting sets. Results show that because closed nodes are added into SE-tree, the search efficiency is highly improved. Furthermore, the proposed method is easy to be understood and implemented. Compared with other effective algorithms with completeness in some experimental tests, the diagnosis efficiency of our proposed method is higher, particularly for single- and double-fault diagnosis.

Keywords: model-based diagnosis, conflict set, minimal hitting set, set enumeration tree.

Model-based diagnosis is one of the active research branches in Artificial Intelligence. Its basic principle is employing the model of a device to judge faults logically, according to the difference between the model's prediction and the observation. Generally, minimal conflict sets are firstly deduced by system description and observation, and then the candidate diagnoses are obtained by computing minimal hitting sets for the collection of all minimal conflict sets. Many methods have been proposed to compute minimal hitting sets, such as HS-TREE[1], HS-DAG[2], HST-TREE[3], BHS-TREE[4], BOOLEAN FORMULAS[5], LOGIC ARRAY[6], GENETIC ALGORITHM[7], etc. The main shortcomings of the above methods include: (1) Some hitting sets may be lost by pruning[1,3]; (2) a tree or a graph needs to be constructed, and the corresponding data structure and algorithm are quite complex[1—3]; (3) computing by recursion is rather inefficient[1—4]; (4) the completeness of the results of some random search algorithms cannot be guaranteed[7]; (5) commonly all hitting sets are firstly stored, and finally they still need to be reduced to minimal ones[4—6]; (6) each component needs to be mapped into a unique numerical value[3].

In order to overcome the shortcomings mentioned above, we propose a novel method of judging a hitting set by the number of conflict sets corresponding to components. Furthermore, the procedure is formalized by combining SE-tree[8] with closed nodes to generate all minimal hitting sets. The advantages of this method include: (1) Any hitting set will not be lost by pruning; (2) although it is described by tree structure, only simple linked list structure but not tree or graph structure needs to be used, and it can be easily implemented; (3) all the hitting sets obtained are minimal, so final reduction is not needed, and all the minimal hitting sets are bound to be obtained; (4) since closed nodes are added to SE-tree, and creation of nodes of non-minimal hitting sets is avoided, the search efficiency is highly improved.

1 Preparation

Firstly, it is necessary to introduce some definitions and theorems corresponding to model-based diagnosis. Then, we will introduce SE-tree briefly.

Definition 1.[1] A system is a triple (*SD*, *COMPS*, *OBS*), where *SD* (the system description) is a set of first order sentences; *COMPS* (the system components) is a finite set of constants; and *OBS* (the system observation) is a finite set of first order sentences.

* Supported by Program for New Century Excellent Talents in University, the Chinese National High Technology Research and Development Plan (Grant No. 2003AA118020), Major Research Program of National Natural Science Foundation of China (Grant No. 60496320 and 60496321), Basic Theory and Core Techniques of Non Canonical Knowledge

** To whom correspondence should be addressed. E-mail: ouyangdantong@163.com; ouyd@mail.jlu.edu.cn

In the following, a unary predicate $AB(\cdot)$ is interpreted to mean "abnormal". $AB(c)$ is true iff c is abnormal, where $c \in COMPS$.

Definition 2.[1] A conflict set (CS) for (SD, $COMPS$, OBS) is a set $\{c_1, c_2, \cdots, c_n\} \subseteq COMPS$, such that $SD \cup OBS \cup \{\sim AB(c_1), \sim AB(c_2), \cdots, \sim AB(c_n)\}$ is inconsistent. A CS for (SD, $COMPS$, OBS) is called a minimal conflict set (MCS), iff no proper subset of it is a conflict set for (SD, $COMPS$, OBS).

Definition 3.[1] Let F be a set cluster. A set H is called a hitting set (HS) for F, if: (1) $H \subseteq \bigcup_{S \in F} S$; (2) for any $S \in F$, $H \cap S \neq \phi$. If no proper subset of a hitting set H is a hitting set for F, then H is called a minimal hitting set (MHS) for F.

We can conclude from Definition 3 that, if a set S_1 is an MHS for F, and S_2 is a proper superset of S_1(i.e. S_1 is a proper subset of S_2), then S_2 is not an MHS for F. This conclusion is a theory basis of adding closed nodes to HSSE-tree algorithm introduced next.

Theorem 1.[1] $D(\Delta, COMPS\text{-}\Delta)$ is a consistency-based diagnosis for (SD, $COMPS$, OBS), iff Δ is an MHS for the collection of conflict sets for (SD, $COMPS$, OBS).

From Theorem 1, we can conclude that, if we know all (minimal) conflict sets, then we can obtain the diagnoses by computing all minimal hitting sets for the (minimal) conflict set cluster.

In the following, we will briefly describe SE-tree proposed by Rymon[8]. It is suitable for enumerating all the possible combinations of all elements of a set. For example, a full SE-tree of $S = \{a, b, c, d\}$ is shown in Fig. 1.

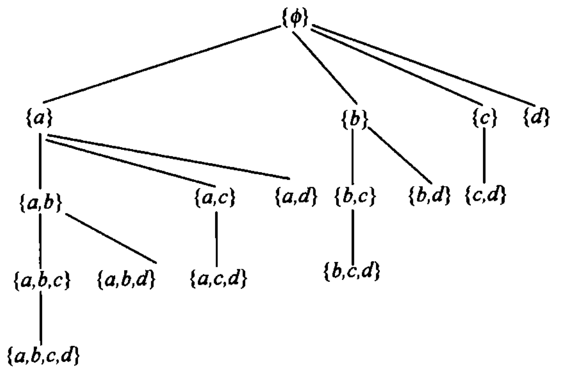


Fig. 1. The SE-tree of the set $\{a, b, c, d\}$.

2 The description of the method

Based on the above basic definitions, some definitions are given below, and several corresponding theorems and corollaries are also outlined to describe the method.

Definition 4. If $e \in S$, it means an element e is relating to a set S. Moreover, $Rcount$ (F, e) is marked as the number of sets in the set cluster F, each of which is related to the element e.

Theorem 2. Let F be a set cluster, a set $H = \{e_1, e_2, \cdots, e_j\} \subseteq \bigcup_{S \in F} S$. If H is a hitting set for F, then $Rcount(F, e_1) + Rcount(F, e_2) + \cdots + Rcount(F, e_j) \geqslant |F|$.

Corollary 1. Let F be a set cluster. A set $H \subseteq \bigcup_{S \in F} S$ is a hitting set for a set cluster F, iff the number of all the different sets, each of which is related to some elements of H, is equal to $|F|$.

According to Corollary 1, we can judge whether a set H is a hitting set for a set cluster F by computing the number of different sets in F, each of which is related to some elements of H. This is the basis of Algorithm 1 to be introduced next.

Theorem 3. Let $F = \{S_1, \cdots, S_n\}$ be a set cluster, m be the number of all the different elements in $S_1, \cdots, S_n$. If $k = \min(n, m)$, then the length l of any minimal hitting set for F is no more than k.

Proof. On the one hand, all elements of an MHS are from all the different elements in $S_1, \cdots, S_n$(the total number of different elements is m), so $l \leqslant m$; on the other hand, any MHS should intersect with any S_i in F, each element in an MHS is at least relating to an S_i, so $l \leqslant n$. Therefore, $l \leqslant \min(n, m)$, i.e. $l \leqslant k$. Q.E.D.

Theorem 3 provides a theory basis for a terminative condition of Algorithm 2 to be introduced next.

Example 1. Let a set cluster $F = \{\{M_1, M_2, A_1\}, \{M_1, M_3, A_1, A_2\}\}$. We can obtain the total number of elements $m = 5$, the length of $Fn = |F| = 2$, so $k = \min(2, 5) = 2$. Therefore, we can easily obtain all the hitting sets for F: $\{M_1\}$, $\{A_1\}$, $\{M_2, M_3\}$, $\{M_2, A_2\}$. Obviously, the length of any hitting set is no more than k ($k = 2$).

Next, we give the basic idea and the basic steps of algorithms of computing all hitting sets for a set cluster. The basic idea: For all the elements (we suppose that the total number is m), combine the SE-tree, enumerate all the necessary sets in the order of Breadth-First, and judge whether the enumerating set H is a hitting set for the set cluster F, i.e. by computing the number of different sets in F, each of which is related to some elements of H. If the number is equal to n, then H is a hitting set for F. At the same time, add some flags (an *MHS* node ("√"), a closed node ("×")) to nodes to improve the search efficiency.

Next, we will describe in detail the algorithm of judging whether a set is a hitting set for a set cluster, and the algorithm of computing all the *MHS*s.

Algorithm 1. The algorithm of judging whether a set H is a hitting set for a set cluster F (IsHS)

Input: a set cluster $F=\{S_1, S_2, \cdots, S_n\}$, $\bigcup_{S\in F} S = S_1 \cup S_2 \cup \cdots \cup S_n = \{c_1, c_2, \cdots, c_m\}$, and a set $H=\{c_1, c_2, \cdots, c_j\} \subseteq \bigcup_{S\in F} S$ (note: j is the length of H). Output: a Boolean value.

The main operation of the algorithm IsHS is as follows:

Step 1: Start from the first element of H, initialize the subscript: $i=1(1\leqslant i\leqslant j)$; introduce the temporal variable $F'=F$; and initialize the number of sets related to the element as $count=0$.

Step 2: If F' is empty, i.e. $|F'|=0$, or all the elements of H are computed, i.e. $i>j$, then judge the total number of sets related to some elements in the set H: if $count=n$, then return TRUE, else return FALSE, stop the algorithm. Otherwise continue.

Step 3: Compute the number of sets related to the element c_i: $Rcount(F', c_i)$, then add to $count$, i.e. $count += Rcount(F', c_i)$.

Step 4: Delete the sets related to the element c_i from F', i.e. $F' \leftarrow F' \setminus \{S \mid c_i \in S\}$.

Step 5: Consider the next element of H, i.e. $i \leftarrow i+1$, then return to Step 2.

Note: In this algorithm, it is not required that all the sets in the set cluster F are minimal (i.e. there may exist some sets where one is a proper subset of another); but if all the sets in F are minimal, the efficiency will be further improved. And the time complexity of this algorithm is $O(nm)$ in the worst situation.

Algorithm 2. The algorithm of combing the revised SE-tree to compute all the *MHS*s (HSSE-tree)

Input: We suppose the inputs are a set *COMPS* and a set cluster F, actually the set *COMPS* can be obtained by $F(COMPS=\bigcup_{S\in F} S = S_1 \cup S_2 \cup \cdots \cup S_n = \{c_1, c_2, \cdots, c_m\})$. Output: all the *MHS*s.

In this algorithm, we do not use the tree structure, but the uniform linked list structure, such as set-enumerating linked list ***enumSetLink***, minimal hitting sets linked list ***pHS***, and need-to-extend linked list ***needExLink***, etc., and the structure of the list is shown in Fig.2. Furthermore, we suppose that the elements in the set *COMPS* are in some regular order (such as in dictionary order), and the elements in any enumerating set are also in the same order as in *COMPS*.

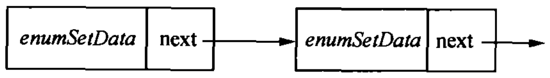


Fig. 2. Nodes of the linked list.

The main operation of the algorithm HSSE-tree is as follows:

Step 1: As the first layer of a SE-tree is an empty set, we start from the second layer, i.e. initialize ***enumSetLink*** as the list of all single-element sets, and let a pointer p point to the first linked node of the ***enumSetLink***. Initialize the need-to-extend linked list ***needExLink*** and the minimal hitting sets linked list ***pHS*** both NULL list, and let a pointer q point to the first node of the ***needExLink***. Suppose $m=|COMPS|$, $n=|F|$, $k=\min(n, m)$; and $i=2$, as the current layer of the SE-tree.

Step 2: If $i>k$, stop.

Step 3: If $p=$NULL, goto Step 9, i.e. to extend the list of needExLink.

Step 4: Call the algorithm IsHS to judge whether the set which p points to (i.e. $p\rightarrow$ ***enumSetData***) is a hitting set. If the algorithm return FALSE, then goto Step 7, else continue.

Step 5: Judge whether there is any non-empty set in *pHS* being a proper subset of the set $p \rightarrow$ *enumSetData*. If not, i.e. it is an *MHS*, then add the set $p \rightarrow$*enumSetData* to the *pHS*, and mark the flag "√" to the corresponding node of the SE-tree. Otherwise, mark the flag "×" to the corresponding node of the SE-tree.

Step 6: Free the node which p points to, and let p point to the next node, goto Step 3.

Step 7: If the set $p \rightarrow$*enumSetData* containing the last element of *COMPS* cannot be extended, then mark the flag "×" to the corresponding node of the SE-tree, otherwise add the set to the need-to-extend list of *needExLink*. Goto Step 6.

Step 8: If $q =$ NULL, i.e. the *needExLink* has been completely extended, then goto Step 10.

Step 9: Let *index* ← the subscript of the element in *COMPS*, which is the last in $q \rightarrow$*enumSetData*. Then for ($j = index + 1$; $j < m$; $j++$), merge all the elements in the set of $q \rightarrow$*enumSetData* and the element *COMPS*[j] into a new sequence set, and add it to the *enumSetLink*; then free the node q points to, and let q point to the next node in the *needExLink*, goto Step 8.

Step 10: Let the pointer p point to the first node of the *enumSetLink*, $i \leftarrow i + 1$, goto Step 2.

Additionally, there is something to be explained as follows:

(1) From Step 3 to Step 7, Algorithm 2 calls IsHS to gradually judge whether the set in the current node of the *enumSetLink* is a hitting set, at the same time marks the flag "√" or "×" to the nodes of *MHS*s or can-not-extend nodes in the SE-tree separately, to close them to improve the search efficiency.

(2) Step 9 is the process of extending all the nodes in the linked list of *needExLink*, and the extended sets are all added to the linked list of *enumSetLink*.

(3) The hitting sets obtained by Algorithm 2 are all the *MHS*s for the set cluster F. Because the process of enumerating subsets by SE-tree is in the order of Breadth-First, once a set is a hitting set and not a proper superset of any generated *MHS* in the *pHS*, then it is bound to be minimal. Furthermore, if the node is an *MHS*, then the node is marked with "√", not to extend it. Therefore, all of its direct subsequent nodes, which are bound to be the proper superset of that *MHS*, can never be generated. If a set is not a hitting set and it has the last element of *COMPS*, then the node is marked with "×", and cannot be extended. In addition, if a set is a proper superset of some generated *MHS*, then the corresponding node is also marked with "×". Otherwise, if a set is not a hitting set, and it does not have the last element of *COMPS*, then it can be extended as more as possible, so all the *MHS*s will be generated finally.

(4) The time complexity of Algorithm 2 is $O(2^m)$ in the worst situation, because the number of nodes generated by SE-tree is at most $\sum_{i=2}^{k} C_m^i$ ($k = \min(n, m)$).

3 An example and the result

Example 2. Compute all *MHS*s of a set cluster $F = \{\{a, b, c, d\}, \{a, b\}, \{b, c\}, \{a, c\}, \{b, d\}, \{b\}\}$.

(1) Initialize: *enumSetLink* $= \{\{a\}, \{b\}, \{c\}, \{d\}\}$, i.e. all the single-element sets; the linked lists of *needExLink* and *pHS* are both NULL. $m = 4$, $n = 6$, $k = \min(n, m) = 4$, *COMPS* is a sequence set $\{a, b, c, d\}$, and "d" is the last element of *COMPS*.

(2) $i = 2$; $i \leqslant k$. Now the list of *enumSetLink* is $\{\{a\}, \{b\}, \{c\}, \{d\}\}$. Call the algorithm IsHS to judge and mark all the enumerating sets as follows: $\{a\}$: $Rcount(F, a) = 3 < 6$, as the set does not contain "d", the last element of *COMPS*, so the node is added to the linked list of *needExLink*. $\{b\}$: $Rcount(F, b) = 5 < 6$, as the set does not contain "d", the node is added to the list of *needExLink*. $\{c\}$: $Rcount(F, c) = 3 < 6$, as the set does not contain "d", the node is added to the list of *needExLink*. $\{d\}$: $Rcount(F, d) = 2 < 6$, as the set contains "d", the node is marked with "×".

After judging and marking all the nodes of *enumSetLink*, the linked list of *enumSetLink* is NULL; and the linked list of *needExLink* becomes $\{\{a\}, \{b\}, \{c\}\}$.

(3) As the list of *needExLink* is $\{\{a\}, \{b\}, \{c\}\}$, not NULL, it needs to be extended as follows: $\{a\}$ is extended as $\{a, \underline{b}\}$, $\{a, \underline{c}\}$, $\{a, \underline{d}\}$, and the

three new nodes are added to the *enumSetLink*, then the node of {a} in the *needExLink* is freed. {b} is extended as {b, $\underline{c}$}, {b, $\underline{d}$}; and both of the new nodes are added to the *enumSetLink*, then {b} in the *needExLink* is freed. {c} is extended as {c, $\underline{d}$}; and the new node is added to the *enumSetLink*, then {c} in the *needExLink* is freed.

After extending all the nodes in the *needExLink*, the *enumSetLink* becomes: {{a, b}, {a, c}, {a, d}, {b, c}, {b, d}, {c, d}}, and the *needExLink* is NULL.

(4) $i = 3$, $i \leqslant 4$. Now, the *enumSetLink* is: {{a, b}, {a, c}, {a, d}, {b, c}, {b, d}, {c, d}}. Call the algorithm IsHS to judge and mark as follows: {a, b}: calls the algorithm IsHS({a, b}, F), $count = 6 = n$, so it is a hitting set; now *pHS* is NULL, so it is an *MHS*. Then it is added to the *pHS*, the node is marked with "√". {a, c}: calls IsHS({a, c}, F), $count = 5 < 6$, not a hitting set. At the same time, there is no "d" in the set, so it is added to the *needExLink*. {a, d}: calls IsHS({a, d}, F), $count = 3 < 6$, not a hitting set, and "d" is in the set, so it is marked with "×". {b, c}: calls IsHS({b, c}, F), $count = 6 = n$, is a hitting set, and not a proper superset of the generated *MHS* of {a, b}, so it is added to the list of *pHS*, and is marked with "√". {b, d}: calls IsHS({b, d}, F), $count = 5 < 6$, not a hitting set, and "d" is in it, so it is marked with "×". {c, d}: calls IsHS({c, d}, F), $count = 4 < 6$, not a hitting set, and "d" is in the set, so it is marked with "×".

After judging and marking all the sets in *enumSetLink*, the list of *enumSetLink* is NULL; and the list of *needExLink* becomes {{a, c}}.

(5) As the list of *needExLink* is {{a, c}}, not NULL, it is extended as follows: {a, c} is extended as {a, c, $\underline{d}$}, and the new node is added to the *enumSetLink*, then {a, c} in the *needExLink* is freed.

After extending *needExLink*, it becomes NULL, and the *enumSetLink* becomes {{a, c, d}}.

(6) $i = 4$, $i \leqslant 4$. Now, *enumSetLink* is {{a, c, d}}, and call IsHS to judge and mark as follows: call IsHS({a, c, d}, F), $count = 5 < 6$, not a hitting set, and "d" is in it, so the node is marked with "×".

After judging and marking all the nodes in the *enumSetLink*, both the *enumSetLink* and the *needExLink* are NULL.

(7) Now, the list of *needExLink* is NULL, so it needs not to be extended.

(8) $i = 5$, $i > k = 4$, so the algorithm is terminated. Finally, all the *MHS*s for F are sets in the linked list of *pHS*: {a, b} and {b, c}.

It is easily validated that the sets of {a, b} and {b, c} are all *MHS*s for the set cluster for F (The process of the method is given in Fig. 3).

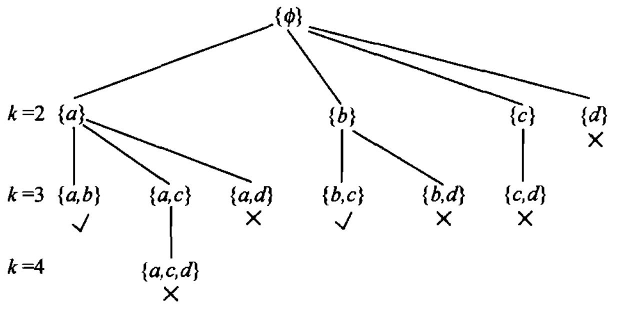


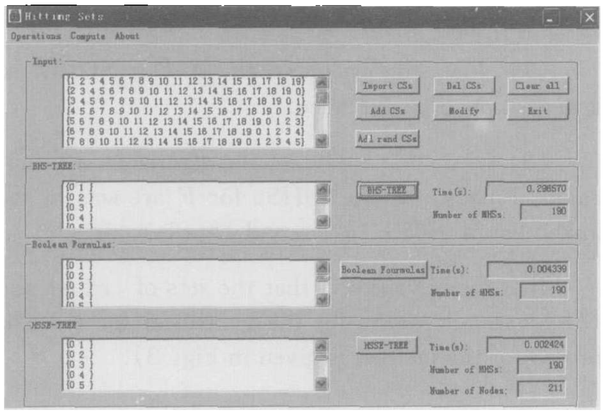
Fig. 3. The process of computing all *MHS*s combining SE-tree.

4 Comparisons

In Ref. [9], we know that the two methods of Boolean Formulas and BHS-tree are more efficient than other algorithms which can obtain all the MHSs for a set cluster.

We have implemented a program in Visual C++ 6.0 to compare the efficiency of HSSE-tree with BHS-tree and Boolean Formulas. The GUI of the program is shown in Fig. 4. And we let all the conflict sets be like {1, 2, ⋯, $m-1$}, {2, 3, ⋯, m}, ⋯, {m, 1, ⋯, $m-2$} (Intel Pentium 4 CPU 2.60 GHz, 512 M RAM, Windows XP). When $m = 3, 4, \cdots, 20$, the experimental result is shown in Table 1 and Fig. 5.

From Table 1 and Fig. 5, we can see clearly that HSSE-tree has the best efficiency among the three better methods when there are more same elements among the conflict sets of the set cluster, i.e. it is quite suitable for generating *MHS*s particularly for single- and double-fault diagnosis, because the depth of revised SE-tree is quite low. Therefore, the number of generated nodes is very small. Results show that with the increasing number of elements, the diagnostic efficiency becomes more and more remarkable.



```
{1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19}
{2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 0}
{3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 0 1}
{4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 0 1 2}
{5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 0 1 2 3}
{6 7 8 9 10 11 12 13 14 15 16 17 18 19 0 1 2 3 4}
{7 8 9 10 11 12 13 14 15 16 17 18 19 0 1 2 3 4 5}
```

BHS-TREE:

```
{0 1 }
{0 2 }
{0 3 }
{0 4 }
```

Time(s): 0.298570
Number of MHSs: 190

Boolean Formulas:

```
{0 1 }
{0 2 }
{0 3 }
{0 4 }
```

Time(s): 0.004339
Number of MHSs: 190

HSSE-TREE:

```
{0 1 }
{0 2 }
{0 3 }
{0 4 }
{0 5 }
```

Time(s): 0.002424
Number of MHSs: 190
Number of Nodes: 211

Fig. 4. The GUI of the program.

Table 1. Running time of the BHS-tree, Boolean Algebra, and HSSE-tree methods

Number of elements (m)	BHS-tree (s)	Boolean Algebra (s)	HSSE-tree (s)
3	0.000160	0.000042	0.000053
4	0.000351	0.000073	0.000077
5	0.000717	0.000115	0.000109
6	0.001338	0.000167	0.000148
7	0.002342	0.000232	0.000196
8	0.003873	0.000341	0.000254
9	0.006285	0.000443	0.000323
10	0.009864	0.000565	0.000408
11	0.015088	0.000716	0.000508
12	0.022243	0.000908	0.000618
13	0.032252	0.001130	0.000747
14	0.046080	0.001450	0.000926
15	0.065168	0.001754	0.001074
16	0.091164	0.002135	0.001279
17	0.122911	0.002611	0.001546
18	0.166507	0.003194	0.001821
19	0.225453	0.003772	0.002087
20	0.285578	0.004334	0.002356

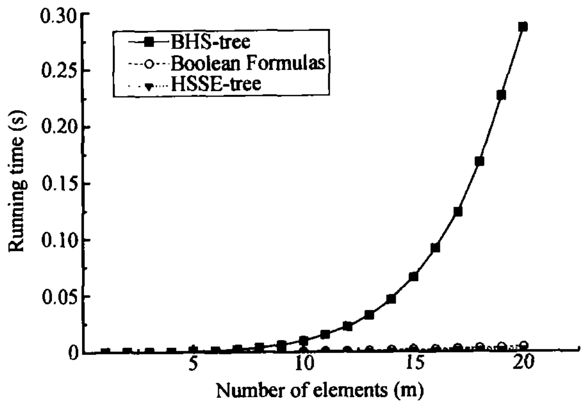


Fig. 5. Running time among BHS-tree, Boolean Formulas and HSSE-tree.

5 Conclusions

A novel method of generating all *MHS*s for a set cluster has been described in this paper. Firstly, an algorithm of judging a hitting set is proposed. Then combing with the revised SE-tree, the algorithm HSSE-tree is proposed to gradually generate all the *MHS*s. At the same time, by adding some closed flags ("√" and "×") into SE-tree, the search efficiency is highly improved.

The method is easy to be understood and implemented. Furthermore, when running the method practically, the space of variables of nodes can be allocated and freed dynamically. Therefore, the space utilization is highly improved, and it can be used to compute *MHS*s even when there are quite a lot of elements in *COMPS*.

Compared with other methods of computing *MHS*s, especially the two methods with better efficiency, our method is more efficient when there are more same elements among the conflict sets of the set cluster, i.e. it is quite suitable for generating *MHS*s as candidate diagnostic results particularly for single- and double-fault diagnosis in model-based diagnosis engineering.

References

1 Reiter R. A theory of diagnosis from first principles. Artificial Intelligence, 1987, 32(1): 57—96.
2 Greiner R., Smith B.A. and Wilkerson R.W. A correction to the algorithm in Reiter's theory of diagnosis. Artificial Intelligence, 1989, 41(1): 79—88.
3 Wotawa F. A variant of Reiter's hitting-set algorithm. Information Processing Letters, 2001, (79): 45—51.
4 Jiang Y. F. and Lin L. Computing the minimal hitting sets with binary HS-tree. Journal of Software (in Chinese), 2002, 13(22): 2267—2274.
5 Jiang Y. F. and Lin L. The computation of hitting sets with Boolean formulas. Chinese Journal of Computers (in Chinese), 2003, 26(8): 919—924.
6 Lin L. Computing minimal hitting sets with logic array in model-based diagnosis. Journal of Jinan University (natural science) (in Chinese), 2002, 23(1): 24—27.
7 Lin L. and Jiang Y. F. Computing minimal hitting sets with genetic algorithm. In: Proceedings of the 13th International Workshop on Principles of Diagnosis, Austria, 2002, 77—80.
8 Rymon R. Search through systematic set enumeration. In: Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning, Cambridge, MA, 1992, 539—550.
9 Lin L. The algorithms of computing minimal hitting sets in model-based diagnosis. Application Research of Computer (in Chinese), 2002, (9): 36—39.