极小碰集求解算法的性能分析与比较

何嫱君[1],赵相福[1],欧阳丹彤[2],张立明[2]
(1. 浙江师范大学计算机系,浙江金华 321000; 2. 吉林大学计算机科学与技术学院,吉林长春 130012)

摘 要: 基于模型的诊断为人工智能领域中一个重要的研究分支,极小碰集即候选诊断的求解过程极大影响最终的诊断效率.本文关注当前主要的极小碰集求解算法,简要介绍了它们的基本思想,从算法描述和实例比较了它们的异同和复杂性,并设计实现了一个统一的实验平台,测试并比较了它们的实际执行效率,为实际选择合适的算法提供了重要参考依据.

关键词: 基于模型的诊断; 碰集; 性能
中图分类号: TP18 **文献标识码:** A **文章编号:** 0372-2112 (2019)05-1101-10
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2019.05.018

Performance Analysis and Comparison of Algorithms for Generating Minimal Hitting Sets

HE Qiang-jun[1], ZHAO Xiang-fu[1], OUYANG Dan-tong[2], ZHANG Li-ming[2]
(1. *Department of Computer, Zhejiang Normal University, Jinhua, Zhejiang* 321000, *China*;
2. *College of Computer Science and Technology, Jilin University, Changchun, Jilin* 130012, *China*)

Abstract: Model-based diagnosis is an important branch of research in the field of artificial intelligence. The efficiency for generating all minimal hitting sets, i.e., candidate diagnoses, considerably affects the final diagnostic process. This paper focuses on the current major algorithms for computing minimal hitting sets. First, the basic ideas of algorithms were briefly introduced. Then, the similarities and differences, and complexity of them were compared by simple algorithm description and examples. An integrated experimental platform was implemented for testing and comparing their time efficiency, which provides an important reference for the actual selection of an appropriate algorithm in practice.

Key words: model-based diagnosis; hitting set; performance

1 引言

基于模型的诊断是人工智能领域的一个重要研究分支.通常,针对待诊断的故障设备,根据其模型和当前的观测信息,先根据一阶逻辑等方法推理求出待诊断系统的极小冲突集合,再根据极小冲突集合求出所有的极小碰集,极小碰集即为系统的候选诊断结果.因此,极小碰集的求解效率将直接影响最终的诊断效率[1~5].

其实,很多求解与已知集合相交的问题,比如集合覆盖问题、本原蕴含问题、教师-课程分配问题等,都可以看作是极小碰集求解的问题.

由于计算极小碰集是一个NP-Hard问题,如何提高求解极小碰集的效率成为专家学者研究的目标.迄今为止已经产生了大量的求解极小碰集的算法.比如,在基于模型的诊断领域中,最早的经典算法是Reiter在1987年提出的HS-Tree算法[6].然而HS-Tree算法在生成HS树的过程中会产生较多的冗余节点,而且可能会因为剪枝策略而丢失有效解.因此,许多研究者对此算法进行了改进,提出了效率更高的HST-Tree算法[7]等.但是,这些算法并不能保证在任何情况下都能保持较高的性能,有些算法的性能也并不是在任何情况下都很糟糕.为了深入研究这些算法,我们实现了极小碰集的多算法集成求解系统,集成了当下主要的10种求解算法,并通过实例比较了这些算法的效率.

2 基本定义

定义1[6] F是一个集合簇,若集合H满足:$H\subseteq$

收稿日期:2018-05-02;修回日期:2018-07-13;责任编辑:蓝红杰
基金项目:浙江省自然科学基金(No. LY16F020004)

$\bigcup_{S\in F} S$,并且对于每一个集合 $S\in F$,都有 $H\cap S\neq\varnothing$,那么 H 称为 F 的一个碰集.

定义 2[6] F 是一个集合簇,如果 F 的一个碰集的任意真子集都不再是 F 的碰集时,那么将该碰集称为极小碰集(Minimal Hitting Set,MHS).

例 1 设冲突集簇 $F=\{\{1,2,3\},\{2,3,4\},\{3,4,5\},\{4,5,6\}\}$,则它的所有极小碰集为:{3,4},{3,5},{3,6},{2,4},{2,5},{1,4}.

3 极小碰集求解算法

本文主要研究以下 10 种求解极小碰集的算法:HS-Tree、HST-Tree、BHS-Tree、CHS-Tree、Boolean 代数、CSSE-Tree、CSISE-Tree、inverse CS-Tree、CS-Tree with Mark Set 及 HSSE-Tree 算法.

3.1 HS-Tree 算法

定义 3[6] 给定集合簇 F,$H(n)$ 表示该集合簇节点 n 产生的极小碰集,以任一集合标记为根节点的一个初始的 HS-Tree T_1 定义及产生如下.

(1)按照节点集合各元素宽度优先生成树 T_1,每次选择 F 中不含当前分支路径上任何元素的一个集合作为当前节点的标签;若已无满足条件的集合,则标记该节点为"√".

(2)重用节点标签:若 n 由集合 S 标记,且 $S\in F$,若 $H(n')\cap S=\{\}$,将节点 n' 标记为 S.

(3)剪枝规则.

①若节点 n 由"√"标记且 $H(n)\subseteq H(n')$,则关闭 n.

②若已生成节点 n 且 $H(n')\subseteq H(n)$,则关闭 n'.

③若节点 n 和 n' 分别用 F 的集合 S 和 S' 标记,且 S' 是 S 的真子集,则对于每条由 n 产生的由 $\alpha(\alpha\in S-S')$ 标记的边都是冗余的,冗余边连同其下方的子树从 HS-Tree 中移除.

最终,所有标记为"√"的 $H(n)$ 即为所有的极小碰集.

HS-Tree 算法随着产生节点的增多,空间复杂度呈指数级增长.

例 2 设集合簇 $F=\{\{2,4,5\},\{1,2,3\},\{1,3,5\},\{2,4,6\},\{2,4\},\{2,3,5\},\{1,6\}\}$,以{2,4,5}作为根节点的 HS-Tree 如图 1 所示.

3.2 HST-Tree 算法

定义 4[7] 给定集合簇 F,设集合 $COMP=\{e\mid e\in S,S\in F\}$,一个初始的 HST-Tree T_2 定义如下.

(1)令 MIN 为 $|COMP|$.

(2)令 v 为树的根. 设 $i(v)=|COMP|+1$,将 v 标记为打开.

(3)宽度优先处理 T_2 的节点.

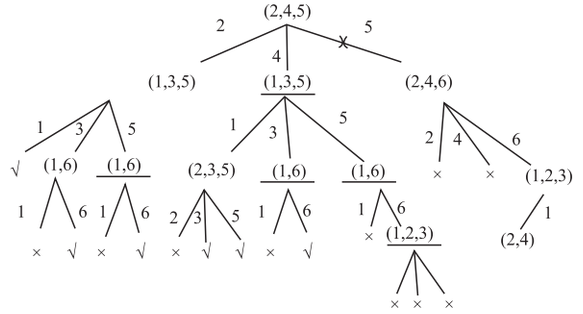

图1 HS-Tree方法产生所有可能的极小碰集

①令 $h(v)=\{c\mid ci(c)=i(v')\}$,$v'$ 是从根到 v 的路径上的节点.

②若对于所有的 $x\in F$,$x\cap h(v)\neq\{\}$,将节点 v 标记为"√",否则,将 v 标记为打开,让 $y\cap h(v)=\{\}$ 的 y 作为 F 中的第一个集合,y 是 F 中的一个子集. 对于每一个先前没有定义过的 y 中的组件 c,令 $ci(c)=$MIN,然后令 MIN = MIN − 1,$\min(v)=$MIN + 1. 若 $i(v)>\min(v)$,创建一个范围为 $\min(v),\cdots,i(v)-1$ 的数组,否则把 v 标记为"×".

③对于 $\min(v),\cdots,i(v)-1$ 中的每一个值 n,创建一个新的节点 v',v' 的父节点为 v,标记 v' 为"□",令 $i(v')=n$. 当 v 的同一代节点都被处理后,处理 v' 这一代的节点,重复执行步骤③.

(4)剪枝规则.

①若与 v 相关联的碰集是已经生成的碰集的超集,将 v 标记为"×".

②删除闭合的节点和该节点与其父节点之间的路径.

HST-Tree 算法的空间复杂度随着产生节点数成指数级增长. 而因为 HST-Tree 只计算极小碰集,实例中创建节点的数量通常远小于理论值.

例 3 给定冲突集簇 $F=\{\{1,2\},\{2,3\},\{1,3\},\{2,4\},\{2\}\}$,它的 HST-Tree 方法生成所有可能的极小碰集的过程如图 2 所示.

3.3 CHS-Tree 算法

定义 5[8] 给定集合簇 F,CHS-Tree T_3 定义如下.

(1)删除冲突集簇 F 中的超集.

(2)将 F 中势最小的集合作为扩展节点. 若势相等,则选择元素出现频率最大的集合.

(3)在去超集后的 F 中,对势最小的集合 $CS=\{a_1,a_2,\cdots,a_m\}$,先删除 F 中包含 a_1 的集合,然后跳转到第 1 步,直到 F 成为空集,依次类推,直到将 F 中包含 a_m 的集合删除.

(4)从根节点开始递归生成所有子节点,然后从根节点出发,每一条路径都产生一个碰集,将它去超集后

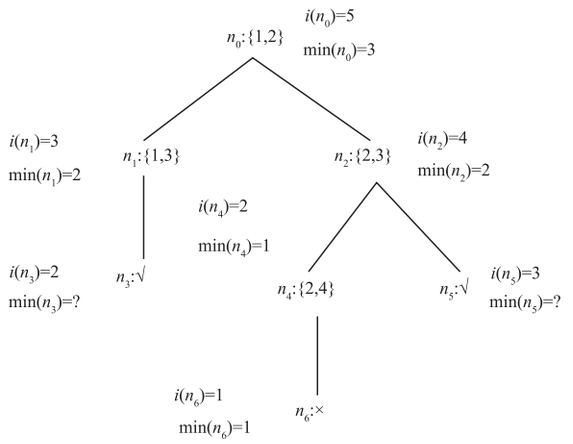

图2　HST-Tree方法产生所有可能的极小碰集

就得到了极小碰集.

CHS-Tree 算法的复杂度与产生的节点数有关,通过删除相关集合和元素使树的深度和宽度不断减小,避免产生冗余节点,使复杂度降低.

例 4　考虑冲突集簇 $F=\{\{2,4,5\},\{1,2,3\},\{1,3,5\},\{2,4,6\},\{2,4\},\{2,3,5\},\{1,6\}\}$,它的 CHS-Tree 方法生成所有可能的极小碰集的过程如图 3 所示.

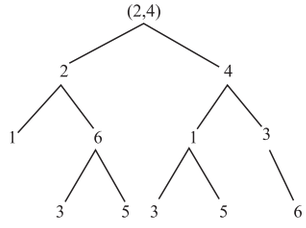

图3　CHS-Tree方法产生所有可能的极小碰集

3.4　BHS-Tree 算法

定义 6[9]　给定集合簇 $F=\{C_1,C_2,\cdots,C_n\}$,BHS-Tree T_4定义如下.

(1)T_4每个节点的标记由集合簇 C 和 H 组成. C 用 < > 标识,H 用[ ]标识. 根节点的 $C=F$,$H=\{\}$. 左右子树根节点记为 C_lH_l、C_rH_r.

(2)若 $C=\{\}$,则 T_4就是它本身.

(3)否则,任取元素 $a\in\cup C_i$,则 $C_l=\{C_i-\{a\}\mid a\in C_i\}$,$H_l=\{a\}$,$C_r=\{C_i\mid a\notin C_i\}$,$H_r=\{\}$. 将初始冲突集簇去超集.

(4)每个节点都建立它的极小冲突集的碰集簇 M,从叶节点开始从底向上进行递归计算.

(5)若 C 是空集,T_4的极小碰集 M 为 H;否则,从底向上递归计算第 6 步和第 7 步直至根节点.

(6)T_4包含了左右子树,$M=\{H,\{m_l\cup m_r\mid m_l\in M_l,m_r\in M_r\}\}$,$H$ 和 M 取自于同一节点,M_l和 M_r分别是是左右子树的碰集.

(7)根节点的 M 去超集后即为 T_4的极小碰集.

BHS-Tree 算法需要自底向上递归计算,且需要去超集后才能得到极小碰集,占用内存较大.

例 5　考虑冲突集簇 $F=\{\{2,4,5\},\{1,2,3\},\{1,3,5\},\{2,4,6\},\{2,4\},\{2,3,5\},\{1,6\}\}$,它的 BHS-Tree 方法生成所有可能的极小碰集的过程如图 4 及图 5 所示.

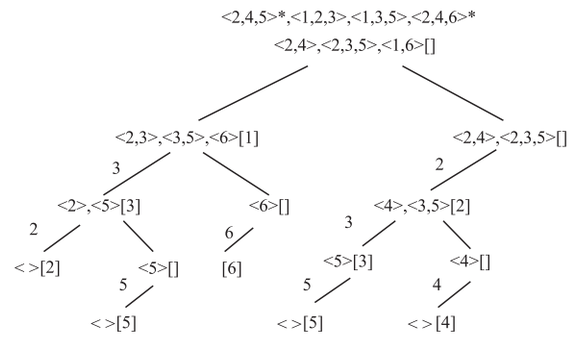

图4　BHS-Tree *T*4的生成

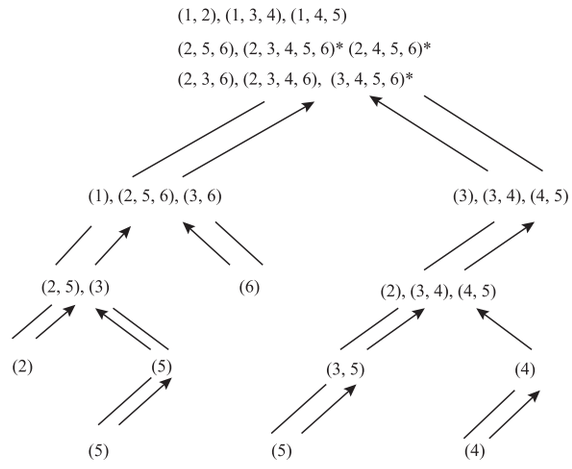

图5　BHS-Tree方法产生所有可能的极小碰集

3.5　布尔(Boolean)代数算法

定义 7[10]　用布尔代数方法计算极小碰集定义如下.

(1)设 $F=\{C_1,C_2,\cdots,C_m\}$ 为一个集合簇,F 的布尔形式定义为:$e_{11}e_{12}\cdots e_{1n1}+e_{21}e_{22}\cdots e_{2n2}+\cdots+e_{m1}e_{m2}\cdots e_{mnm}$,记为 *CSF*.

(2)设 $H=\{h_1,h_2,\cdots,h_n\}$ 为一个集合,$h_1h_2\cdots h_n$为该集合的布尔形式,记为 *HF*.

(3)设集合簇 *HS* 是集合簇 F 的所有极小碰集的集合簇,$HS=\{H_1,H_2,\cdots,H_k\}$,其中 $H_i=\{h_{i1},h_{i2},\cdots,h_{ini}\}$,则 $h_{11}h_{12}\cdots h_{1n1}+h_{21}h_{22}\cdots h_{2n2}+\cdots+h_{k1}h_{k2}\cdots h_{knk}$为碰集簇的布尔形式.

(4)函数 $\Gamma(\Pi)$递归定义如下:

①$\Gamma(0)=1$,$\Gamma(1)=0$;

②$\Gamma(\bar{e})=e$;

③$\Gamma(\bar{e}\Pi)=e+\Gamma(\Pi)$;

④$\Gamma(\bar{e}+\prod)=e*\Gamma(\Pi)$;若 Π 不满足以上 4 条,则用规则⑤计算.

⑤$\Gamma(\Pi)=\bar{e}*\Gamma(\Pi_1)+\Gamma(\Pi_2)$. $\bar{e}$ 是 Π 中任意一个文字,Π_1 是 Π 中不含 $\bar{e}$ 的全部单项式,Π_2 是 Π 中所有项删除 $\bar{e}$ 后剩余的布尔形式.

(5)根据上述定义,有如下结论.

设 $CSF=\Pi$ 是最小集合簇 F 的布尔形式,则 $\Gamma(\Pi)$ 即为 F 的极小碰集簇的布尔形式.

设 F 表示冲突集合簇,Π 是 CS 的布尔形式,算法 BHS 的输入为 Π,输出为极小碰集的布尔形式 $\Gamma(\Pi)$.

①若 Π 由一个单项构成 $\Pi=(\bar{e}_1\bar{e}_2\cdots\bar{e}_n)$,则由上述定义,输出结果为:$e_1+e_2+\cdots e_n$. 此时 BHS 算法终止,否则执行②.

②检查 Π 中是否有单文字单项式,若有,执行③,否则执行④.

③设 $\bar{e}$ 是 Π 中的单文字单项式,根据上述定义提取公因子 e. 若 Π 中所有的单项式都含 $\bar{e}$,则输出 BHS(Π)$=e$,算法终止;否则,BHS(Π)$=e*$BHS(Π_1),并递归计算 BHS(Π_1).

④计算 Π 中各文字出现的频率,设 $\bar{e}$ 出现频率最高.

⑤根据 $\Gamma(\Pi)$ 的定义的规则⑤,执行按 $\bar{e}$ 分解,BHS(Π)$=e*$BHS(Π_1)$+$BHS(Π_2),并分别递归计算 BHS(Π_1)和 BHS(Π_2).

其中 Π_1 是 Π 中不含 $\bar{e}$ 的全部单项式,Π_2 是在 Π 的全部表达式中删除 $\bar{e}$ 之后剩余的布尔表达式.

布尔代数算法不需要建立树,只需一次递归即可得到极小碰集,时间和空间复杂度目前最优.

例 6 设 $F=\{\{2,4,5\},\{1,2,3\},\{1,3,5\},\{2,4,6\},\{2,4\},\{2,3,5\},\{1,6\}\}$,$H=\{1,2\}$.

则 $CSF=(245+123+135+246+24+235+16)$,$HF=(12)$.

根据 $\Gamma(\Pi)$函数定义的规则5对 CSF 进行按2分解.

$\Pi_1=(135+16)$

$\Pi_2=(45+13+46+4+35+135+16)$

则

$$\begin{aligned}\Gamma&=(245+123+135+246+24+235+16)\\&=2*\Gamma(135+16)\\&\quad+\Gamma(45+13+46+4+35+135+16)\end{aligned}$$

又因为

$$\Gamma(135+16)=1+\Gamma(35)+\Gamma(6)\Gamma(35)=3+5$$

所以

$$2*\Gamma(135+16)=12+236+256$$

也即 F 中含 2 的碰集为$\{1,2\},\{2,3,6\},\{2,5,6\}$.

最后,对于不含 2 的碰集部分:

$$\begin{aligned}&\Gamma(45+13+46+4+35+135+16)\\&=13+4+35+16\\&=4*\Gamma(13+35+16)\\&=4*((13+15)+36)\\&=413+415+436\end{aligned}$$

于是,所有的极小碰集为$\{\{1,2\},\{2,3,6\},\{2,5,6\},\{4,1,3\},\{4,1,5\},\{4,3,6\}\}$.

3.6 HSSE-Tree 算法

定义 8[11] HSSE-Tree 算法定义如下.

设枚举集合链为 *enumSetLink*、待扩展的集合链为 *needExLink*,极小碰集链为 *pHS*. *enumSetData* 域表示元素的集合,*next* 域表示指向下一个相邻的节点的指针. 假定系统的元素是有序的.

(1)SE-Tree 的第一层是空集,从第二层开始,将 *enumSetLink* 初始化为所有单元素的集合链. 将指针 p 指向 *enumSetLink* 的第一个集合节点. 将 *needExLink* 和 *pHS* 初始化为空链,将指针 q 指向 *needExLink* 的第一个集合节点. 令 $m=|COMPS|$,$n=|F|$,$k=\min(m,n)$. 设 $i=2$,表示当前 HSSE-Tree 的层次.

(2)若 $i>k$,则算法终止.

(3)若 $p==$NULL,则跳转到第 8 步.

(4)调用算法 IsHS① 来判定 p 所指的集合是否为碰集,若返回假,跳转到第 7 步.

(5)判断极小碰集链 *pHS* 中是否存在指针 p 所指的集合的非空真子集. 若存在,那么在 SE-Tree 中的相应的节点处标记"×",跳转到第 6 步. 否则将指针 p 所指的集合加入到极小碰集链 *pHS* 中,同时在 SE-Tree 中相应的节点处标记"√".

(6)释放指针 p 所指的节点,让 p 指向 *enumSetLink* 的下一个相邻节点,转到第 3 步.

(7)若指针 p 所指的集合中含有 *COMPS* 中最后的元素,那么就在 HSSE-Tree 中的相应的节点处标记"×",跳转到第 6 步. 否则就将该集合加入到 *needExLink* 中,转到第 6 步.

(8)若 $q==$NULL,跳转到第 10 步.

(9)设 $q\rightarrow$ *enumSetData* 中最后一个元素在 *COMPS* 中的标号为 *index*←. 将 *COMPS* 中的各个元素标号为 *index*+1 到 m,分别与 $q\rightarrow$*enumSetData* 中的所有元素结合后形成一个新的集合,将此新集合加入到 *enumSetLink* 中,释放 *needExLink* 中的该节点. q 向后移动到 *needExLink* 中相邻的下一个节点,跳转到第 8 步.

(10)指针 p 指向 *enumSetLink* 的第一个节点,令 $i=$

① IsHS 算法用于判定给定的一个集合 H 是否给定集合簇 F 的碰集,即判定 H 是否和 F 中每一个集合交集是否都非空.

$i+1$,跳转到第 2 步.

HSSE-Tree 算法用链表实现,数据结构相对简单,但产生的节点较多,在最坏情形的时间复杂度依然较差,而对单双故障因扩展层数较少而具有较高的效率.

例 7　设冲突集簇 $F=\{\{1,2,3,4\},\{1,2\},\{2,3\},\{1,3\},\{2,4\},\{2\}\}$. 则产生所有极小碰集的过程如图 6 所示.

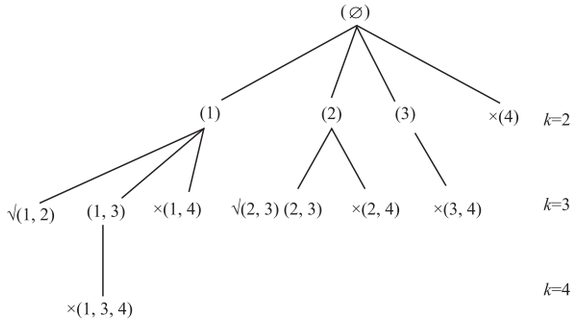

图6　HSSE-Tree算法产生所有可能的极小碰集

3.7　翻转的(inverse)CS-Tree 方法

定义 9[12]　一个有序诊断集 C① 的一棵翻转的 CS-Tree T_5定义如下.

(1)T_5的根标识为∅.

(2)设 n 为 T_5的一个节点,n_p是 n 的父节点,n'是 n 的左兄弟节点. 设 n 的标识为有序集 $S_n=S_{n_p}\cup\{c\}$,其中 c 为 C 中 c'的下一个邻接元素,c'是 n'的标签集 $S_{n'}$中的最后一个元素.

(3)对于 T_5中的每一棵子树的最左边的节点,该节点的 $S_n=S_{n_p}\cup\{c\}$,式中 n_p是 n 的父节点,元素 c 是 C 中紧邻 c_p的下一个元素,元素 c_p是 S_{n_p}的最后一个元素. 若 $S_n==\varnothing$,则 c 为 C 中第一元素.

根据以上的定义,T_5产生过程如下.

(1)深度优先生成 T_5.

(2)剪枝规则.

对每一个节点,调用 IsHS 算法来判断该标签集是否是一个碰集.

①若 n 的标签集 S_n是一个碰集,关闭该节点.

②若 n 的标签集 S_n是某个已经产生的不是碰集的子集,则关闭该节点.

翻转的 CS-Tree 方法因元素的顺序不同可能存在不同样式的树,因此树的节点个数也是不同的,相应的复杂度也是不同的. 该算法可能产生一些冗余节点且节点间存在一些冗余的比较,因此效率受到一定的影响.

例 8　设一个初始的诊断集 $C=\{1,2,3,4\}$,假设所有的极小碰集为{1},{3},以及{2,4}. 则产生所有极小碰集的过程按照 1,2,3,4 的顺序如图 7 所示.

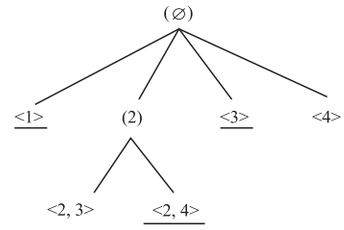

图7　翻转的(inverse)CS-Tree方法产生所有可能的极小碰集

3.8　CSSE-Tree 算法

定义 10[13]　一个有序的初始诊断集 C 的 CSSE-Tree T_6定义如下.

(1)T_6的根标签集为∅.

(2)T_6的每一个节点 n 都由标签集 S_n、父边标签 E_{P_n}以及子边标签集 E_{C_n}组成. 根节点的标识为:$S_T=\{\varnothing\}$,$E_{P_T}=\{\}$,$E_{C_T}=C$.

(3)设每一个节点 n 的子边标签集 $E_{C_n}=\{e_1,e_2,\cdots,e_k\}$,那么对于每一个 $e_i\in E_{C_n}(1\leqslant i\leqslant k)$,都存在一个子节点 n_{e_i},它的标签集为 $S_{n_{e_i}}=S_n\cup\{e_i\}$,父边标签为 e_i,子边标签集为 $E_{c_{n_{e_i}}}=\{e_{i+1},\cdots,e_k\}$.

根据以上的定义,T_6产生过程如下.

(1)宽度优先生成树 T_6.

(2)剪枝规则.

对每一个节点调用 IsHS 来判断该标签集是否是一个碰集.

①若 n 的标签集 S_n是一个碰集,将该节点标识为“√”.

②若 n 的标签集 S_n不是一个碰集,则:

for(每一个在节点 n 的右边标识为“√”的节点 m)

$$\text{if}\ (E_{p_m}\in E_{c_n}\wedge S_n\supseteq S_m-\{E_{p_m}\})$$
$$E_{c_n}=E_{c_n}-\{E_{p_m}\}$$

CSSE-Tree 算法的复杂度与产生的节点数有关,但避免了许多冗余节点的产生,且比较次数也有所减少,因此复杂度比翻转的 CS-Tree 低.

例 9　设一个初始的诊断集 $C=\{1,2,3,4\}$,设所有的极小碰集为{1},{3},以及{2,4}. 则产生所有极小碰集的过程按照 1,2,3,4 的顺序如图 8 所示.

3.9　带标识集的 CS-Tree(CS-Tree with Mark Set)算法

定义 11[12]　一个有序的初始诊断集 C 的 CS-Tree with Mark Set T_7定义如下.

(1)T_7的根节点标识为 C.

① 注:初始情况下,给定冲突集合簇 F 时,$C=\{e\mid e\in S,S\in F\}$,后面提到的 CSSE-Tree 方法、CSISE-Tree 方法、CS-tree with Mark Set 方法初始情况与之相同.

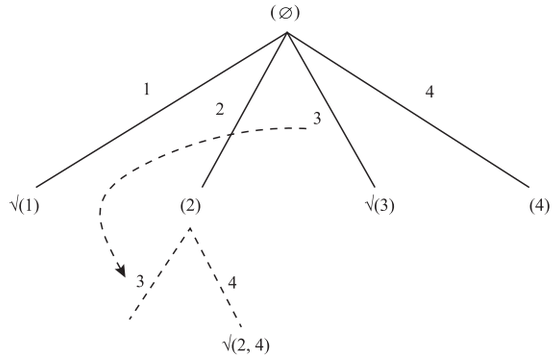

图8 CSSE-Tree 方法产生所有可能的极小碰集

(2)T_7中的任一节点 n 标识为$[S_n, S_{M,n}]$,$S_n \subseteq C$ 是 n 的标签集,$S_{M,n} \subseteq S_n$ 是 n 的标识集. 若 $S_{M,n} = S_n$ 或$|S_n| = 1$,则 n 没有后继节点. 否则,对于每一个元素 $c \in S_n - S_{M,n}$,n_c是 n 的后继节点,$n_{c'}$是 n_c的直接左兄弟节点,n_c 的标识为$[S_{M,n} \cup (S_n - S_{M,n} - \{c\}), S_{M,n'} \cup \{c'\}]$.

(3)T_7 的每一棵子树的最左节点 n,满足 $S_{M,n} = S_{M,n_p}$,n_p为 n 的父节点.

根据以上的定义,T_7产生过程如下.

(1)深度优先生成树 T_7.

(2)剪枝规则.

对每一个节点调用 IsHS 来判断该标签集是否是一个碰集.

①若 n 的标识 $S_{M,n}$是极小碰集,且为已产生的树 T_7 中某节点的标签集,则关闭节点 n.

②若 n 的标签集 S_n 不是碰集,则关闭节点 n.

CS-Tree with Mark Set 算法因元素顺序不同会产生不同的树,因此复杂度也不同,但是本算法因为引入了标识集而变得更为复杂.

例 10 设一个初始的有序诊断集 $C = \{1,2,3,4\}$,假设所有的极小碰集为$\{1,2\}$,$\{2,3\}$,以及$\{2,4\}$. 则产生所有极小碰集的过程按照 1,2,3,4 的顺序如图 9 所示.

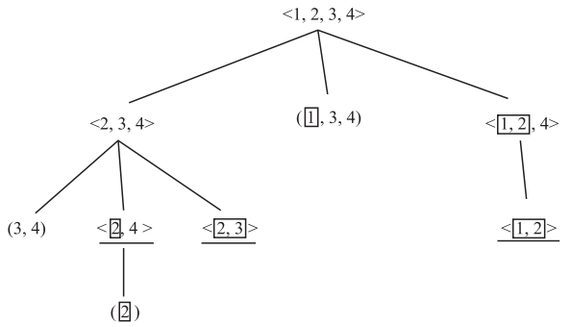

图9 CS-Tree with Mark Set 方法产生所有可能的极小碰集

3.10 CSISE-Tree 算法

定义 12[13] 一个有序的初始诊断集 C 的 CSISE-Tree T_8定义如下.

(1)T_8的每一个节点 n 由标签集 S_n、父边标签 E_{P_n} 以及子边标签集 E_{C_n}三部分组成. 根节点标识为:$S_T = S$,$E_{P_T} = \{\}$,$E_{C_T} = C$.

(2)对于每一个节点 n,若$|S_n| = 1$,则该节点终止;否则假定 $E_{C_n} = \{e_1, e_2, \cdots, e_k\}$,对于每一个 $e_i \in E_{C_n}$($1 \leqslant i \leqslant k$),都存在一个子节点 n_{c_i},它的标签集为 $S_{n_{c_i}} = S_n - \{e_i\}$,父边标签为 e_i,子边标签集为 $E_{c_{n_i}} = \{e_{i+1}, \cdots, e_k\}$.

根据以上的定义,T_8产生过程如下.

(1)宽度优先生成树 T_8.

(2)剪枝规则.

对于同一层的节点,从右至左对每一个节点调用 IsHS 来判断该标签集是否是一个碰集.

①若 n 的标签集 S_n不是碰集,则将该节点标识为“×”.

②若 n 的标签集 S_n是碰集且 $E_{c_n} \neq \{\}$,则:

for (每一个在节点 n 的右边标识为“×”的节点 m)

if $(E_{p_m} \in E_{c_n} \wedge S_n - \{E_{p_m}\} \subseteq S_m\})$

$E_{c_n} = E_{c_n} - \{E_{p_m}\}$

(3)标识规则.

若 n 的标签集 S_n是碰集,将该节点标识为“√”,删除它的父节点标识“√”.

for (每一个在节点 n 的右上边标识为“√”的节点 m)

if $(S_n \subset S_m)$

删除节点 m 的标识“√”

CSISE-Tree 算法的复杂度与产生的节点数有关,相对于翻转的 CS-Tree 方法,CSISE-Tree 方法避免了一些冗余节点的产生.

例 11 设一个初始的有序诊断集 $C = \{1,2,3,4\}$,假设所有的极小碰集为$\{1,2\}$,$\{2,3\}$,以及$\{2,4\}$. 则产生所有极小碰集的过程按照 1,2,3,4 的扩展顺序如图 10 所示.

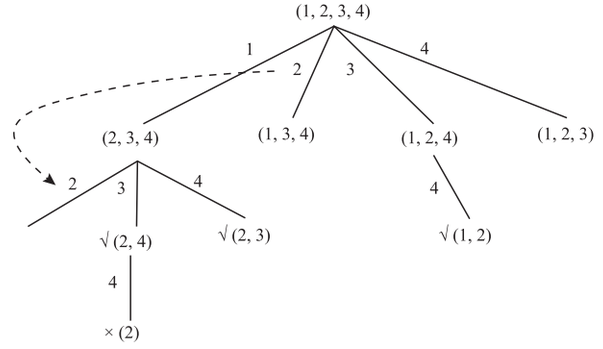

图10 CSISE-Tree方法产生所有可能的极小碰集

4 实验与结果分析

4.1 算法集成实验平台

上述所提到的算法都已实现,实现界面如图 11 所

示. 本实验平台使用C++语言进行实现,平台的界面使用 Visual C++ 6.0 编程实现.

本系统可以求解并显示每个算法所产生的极小碰集的个数、运行时间以及产生的节点个数. 通过比较各算法运行的时间及产生的节点个数,可以直观地比较各算法的执行效率.

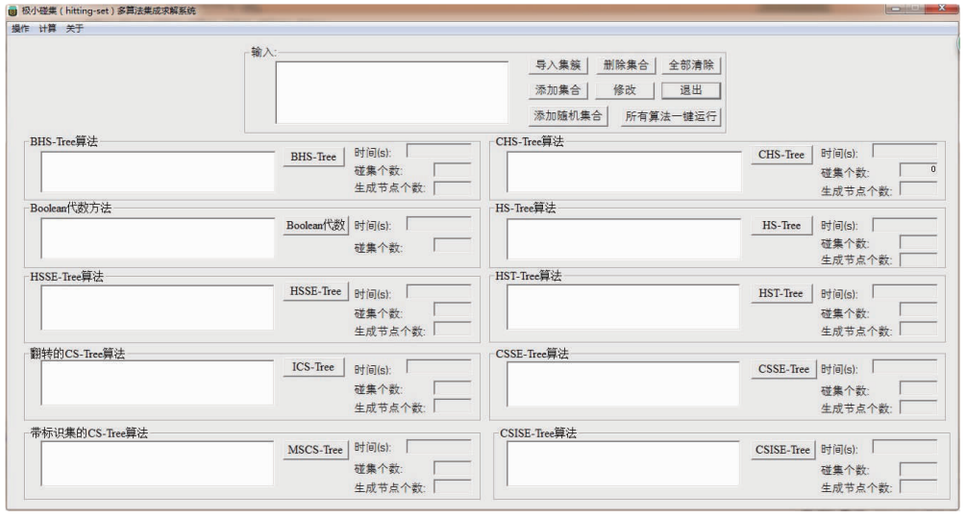

图11　极小碰集多算法集成求解系统主界面

4.2　极小碰集求解算法的比较

算法的性能随系统的部件数量、冲突集簇的集合长度以及冲突集簇中的集合个数的变化而有不同的表现,可以通过运行各算法,比较它们各自产生节点的个数以及所花费的时间. 本文比较的是系统集成的这 10 种算法,其中由于布尔代数方法并不生成树,因此不考虑此算法的生成节点数目.

下面将从三个不同的角度对算法比较:(1)系统部件数不同时算法性能的比较;(2)冲突集簇的集合长度不同时算法性能的比较;(3)冲突集簇的集合个数不同时算法性能的比较.

4.2.1　系统部件数不同时算法性能的比较

设第一组的冲突集簇数据形如:$\{1,2,\cdots,i-1\}$,$\{2,3,\cdots,i\}$,$\cdots$,$\{i,1,\cdots,i-2\}$. 其中 i 表示冲突集簇中包含的系统部件数(即不重复的元素个数),该形式的冲突集簇生成的极小碰集长度较短,这 10 种求解极小碰集的算法各自所产生的节点数目以及计算的耗时如图 12 和图 13 所示.

从图 12 可以看出,当系统部件数较少时,各算法生成的节点数目都较少. 随着系统部件数的增加,带有标识集的 CS-Tree 和 CSISE-Tree 算法所产生的节点数都急剧增多. BHS-Tree、CHS-Tree、HSSE-Tree、HS-Tree、HST-Tree、翻转的 CS-Tree 和 CSSE-Tree 算法所产生的节点数目并没有大幅增加,而是保持在一个比较稳定的数值,展现了较好的性能.

从图 13 可以看出,当系统部件数目较少时,各个算法的计算耗时都较短. 随着系统部件数的增加,带有标

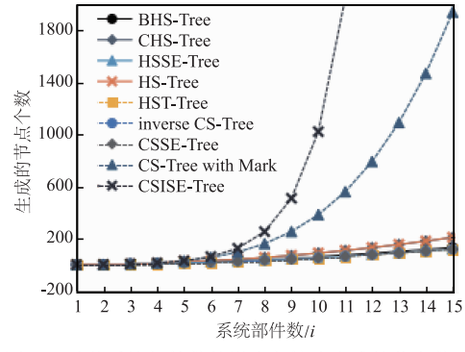

图12　随着系统部件数的增多各算法生成的节点数目

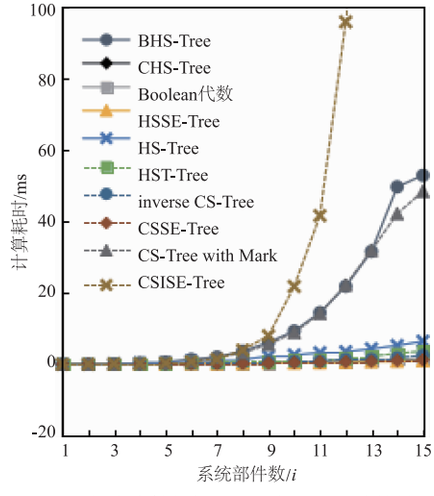

图13　随着系统部件数的增多各算法的计算耗时

识集的 CS-Tree、CSISE-Tree 和 BHS-Tree 算法的计算耗

时都急剧增加. CHS-Tree、Boolean 代数、HSSE-Tree、HS-Tree、HST-Tree、翻转的 CS-Tree 和 CSSE-Tree 算法的计算耗时并没有因此而大幅度的增加,也保持在一个相对稳定的水平.

因此,随着系统部件数目的增多,无论是从产生的节点数目还是从计算耗时方面,CHS-Tree、Boolean 代数、HSSE-Tree、HS-Tree、HST-Tree、翻转的 CS-Tree 和 CSSE-Tree 算法的运行效率都相对较高,性能相对稳定. 反观带有标识集的 CS-Tree 和 CSISE-Tree 算法,当系统部件数大于 9 时,算法生成树中的节点数明显增多,运行耗时大幅增加. 特别是 CSISE-Tree 算法,在系统部件数大于 10 时,运行效率大幅下降. 因此无论是从产生的节点数目还是计算耗时方面,随着系统部件数的增多,它们的运行效率都急速下降. 而对于 BHS-Tree 算法,虽然随着系统部件数的增多,它的生成树中的节点数量并没有明显增多,但是它的运行耗时却大幅增加,因此随着系统部件数的增多它的运行效率也大幅下降.

综上,在计算生成的极小碰集簇的集合长度较短的情况下,当系统的部件数少于 9 时,这 10 种算法的性能发挥都很不错,都可以选用;而当系统部件数多于 9 时,可选用性能仍然较高的 CHS-Tree、Boolean 代数、HSSE-Tree、HS-Tree、HST-Tree、翻转的 CS-Tree 和 CSSE-Tree 算法.

4.2.2 冲突集簇的集合长度不同时算法性能的比较

设第二组的冲突集簇数据形如:$\{1,2,\cdots,j\}$,$\{2,3,\cdots,j+1\}$,$\cdots$,$\{k,k+1\cdots,k+j-1\}$(注:这里取 $k=10$,表示冲突集簇的集合个数). 这 10 种求解极小碰集的算法各自所产生的节点数目以及计算的耗时如图 14 和图 15 所示.

从图 14 可以看出,当冲突集簇的集合长度较短时,各个算法所产生的节点数都较少. 随着冲突集簇的集合长度的增加,带有标识集的 CS-Tree 和 CSISE-Tree 算法所产生的节点数都急剧增多. BHS-Tree、CHS-Tree 和 CSSE-Tree 算法所产生的节点数目并没有因此而大幅度的增加,而是保持在一个比较稳定的数值,展现了较好的性能. HSSE-Tree、HS-Tree、HST-Tree 和翻转的 CS-Tree 算法产生的节点数目随着冲突集簇的集合长度的增加而缓慢增加.

从图 15 可以看出,当冲突集簇的集合长度较短时,各个算法的计算耗时都较短. 随着冲突集簇的集合长度的增加,带有标识集的 CS-Tree 和 CSISE-Tree 算法的计算耗时都急剧增加. BHS-Tree、CHS-Tree、Boolean 代数、HSSE-Tree、HS-Tree、HST-Tree、翻转的 CS-Tree 和 CSSE-Tree 算法的计算耗时并没有大幅度增加,保持在一个相对稳定的水平.

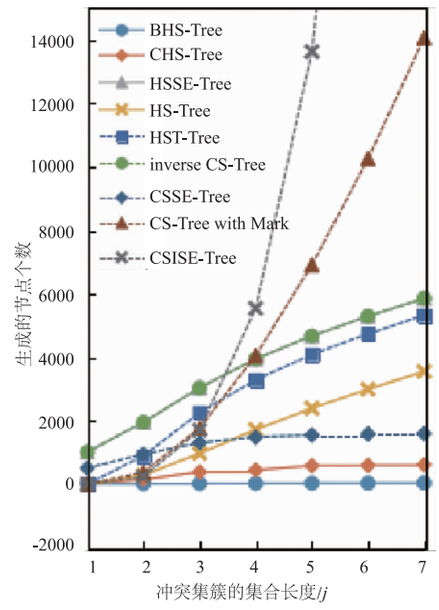

图14 随着冲突集簇的集合长度增多各算法生成的节点数目

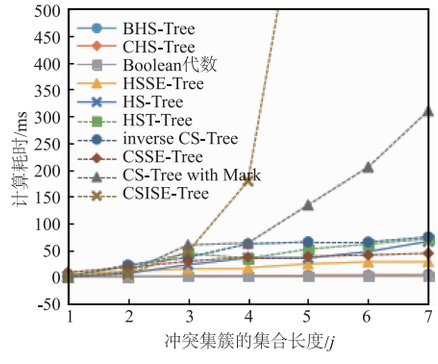

图15 随着冲突集簇的集合长度增多各算法的耗时

因此,随着系统部件数目的增多,在产生的节点数目方面,HSSE-Tree、HS-Tree、HST-Tree 和翻转的 CS-Tree 算法随着冲突集簇的集合长度的增加,产生的节点数也在缓慢地增加. 但是在计算耗时方面,BHS-Tree、CHS-Tree、Boolean 代数、HSSE-Tree、HS-Tree、HST-Tree、翻转的 CS-Tree 和 CSSE-Tree 算法的运行效率都相对较高,性能相对稳定. 其中 Boolean 代数算法的性能表现最好,计算耗时随着冲突集簇长度的增加,上升幅度较小且耗时最短. 反观带有标识集的 CS-Tree 和 CSISE-Tree 算法,当冲突集簇的集合长度大于 4 时,无论是从产生的节点数目还是计算耗时方面,随着冲突集簇的集合长度的增大,它们的算法生成树中的节点数量明显增多,运行耗时大幅增加,运行效率都急速下降.

综上,在计算生成的极小碰集长度较短的情况下,当冲突集簇的集合长度小于 4 时,这 10 种算法都体现了良好的性能,都可以选用;而当冲突集簇的集合长度

大于4时,可选用性能仍然较高的 BHS-Tree、CHS-Tree、Boolean 代数和 CSSE-Tree 算法.

4.2.3　冲突集簇的集合个数不同时算法性能的比较

设第三组的冲突集簇数据形如:$\{1,1+k\}$,$\{2,2+k\}$,…,$\{k,2k\}$,该形式下的冲突集簇计算生成的极小碰集势较大. 这 10 种求解极小碰集的算法各自所产生的节点数及计算耗时如图 16 和图 17 所示.

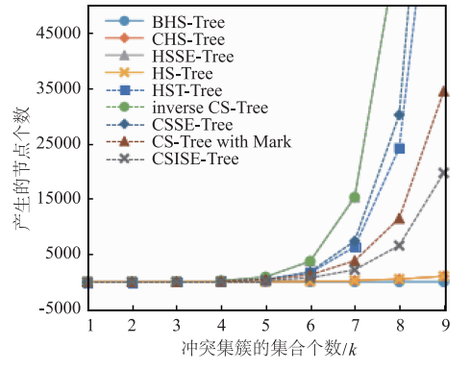

图16　随着冲突集簇的集合个数增多各算法生成的节点数

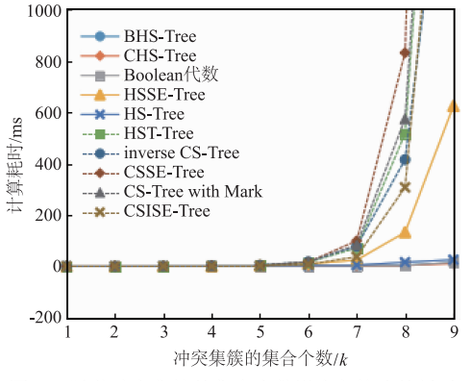

图17　随着冲突集簇的集合个数增多各算法的耗时

从图16可以看出,当冲突集簇的集合个数较少时,各个算法所产生的节点数目都较少. 随着冲突集簇的集合个数的增加,翻转的 CS-Tree、CSSE-Tree、HST-Tree、带有标识集的 CS-Tree 和 CSISE-Tree 算法所产生的节点数都急剧增多. 而 BHS-Tree、CHS-Tree 和 HS-Tree 算法所产生的节点数目并没有大幅度增加,而是保持在一个比较稳定的数值,展现了较好的性能.

从图17可以看出,当冲突集簇的集合个数较少时,各个算法的计算耗时都较短. 随着冲突集簇的集合个数的增加,翻转的 CS-Tree、CSSE-Tree、HST-Tree、带有标识集的 CS-Tree、CSISE-Tree 和 HSSE-Tree 算法的计算耗时都急剧增加. 而 BHS-Tree、CHS-Tree、Boolean 代数和 HS-Tree 算法的计算耗时并没有大幅增加,也保持在一个相对稳定的水平.

随着冲突集簇中集合个数的增加,HSSE-Tree 算法虽然产生的节点数目并没有大幅增加,但计算耗时却有明显增加. 而 BHS-Tree、CHS-Tree、Boolean 代数和 HS-Tree 算法无论从产生的节点数目还是计算耗时方面,运行效率都相对较高,性能相对稳定. 反观翻转的 CS-Tree、CSSE-Tree、HST-Tree、带有标识集的 CS-Tree 和 CSISE-Tree 算法,当冲突集簇的集合个数大于7时,随着冲突集簇中集合长度的增大,生成树中的节点数量明显增多,运行耗时大幅增加,运行效率都急速降低.

可以看出,在计算生成的极小碰集较长的情况下,当冲突集簇的集合个数小于7时,这 10 种算法都具有良好的性能,都可以选用;而当冲突集簇的集合个数多于7时,可选用性能仍较高的 BHS-Tree、CHS-Tree、Boolean 代数和 HS-Tree 算法.

综上,可以得出以下结论:CHS-Tree 和 Boolean 代数算法的综合性能最优. 这也可从它们的算法原理中得以解释:CHS-Tree 算法首先删除了冲突集簇中的超集,可以减少节点数目和循环次数. 然后依据集合的势以及与集合相关联元素的个数来选择扩展节点,生成树的过程较为简单,能产生较少的节点. 而 Boolean 代数算法不需要建立树或者图,不必剪枝,而是利用数组或者动态链表,只要一次递归计算字符串就可以得到碰集,空间复杂度和时间复杂度都较低.

此外,针对极小碰集的势:(1)当计算生成的极小碰集势较小时,可以优先选用 CHS-Tree、Boolean 代数和 CSSE-Tree 算法,因为随着系统部件数的增多和冲突集簇中集合长度的加长,这些算法都有较高的计算效率;(2)当计算生成的极小碰集势较大时,可以优先选用 BHS-Tree、CHS-Tree、Boolean 代数和 HS-Tree 算法,因为随着冲突集集合个数的增加,这些算法在该情况下保持着较高的计算效率. 另外,翻转的 CS-Tree、带有标识集的 CS-Tree、CSISE-Tree、HSSE-Tree 和 HST-Tree 算法在这两种情况下,只在特定情况下有较好的表现,所以在这两个场景下不推荐使用.

5　结束语

本文首先对当下 10 种主要的求解极小碰集的算法进行了简介、实例分析、算法实现,并利用极小碰集多算法集成求解系统对这 10 种求解极小碰集的算法进行了实验比较,具体从系统的部件数量、冲突集簇的集合长度以及冲突集簇中的集合个数三个角度测试了各个算法性能的表现,通过实验结果进一步分析了各算法的适用范围,为以后算法的选择、研究与改进等提供了参考依据.

参考文献

[1] Zhao X, Ouyang D, Zhang L. Computing all minimal hit-

ting sets by subset recombination[J]. Applied Intelligence, 2018,48(2):257 – 270.
[2] Zhao X,Ouyang D. Deriving all minimal hitting sets based on join relation[J]. IEEE Transactions on Systems, Man, and Cybernetics:Systems,2015,45(7):1063 – 1076.
[3] 赵相福,欧阳丹彤. 使用 SAT 求解器产生所有极小冲突部件集[J]. 电子学报,2009,37(4):804 – 810.
ZHAO Xiang-fu, OUYANG Dan-tong. Deriving all minimal conflict sets using satisfiability algorithms[J]. Acta Electronica Sinica,2009,37(4):804 – 810. (in Chinese)
[4] 欧阳丹彤,刘伯文,周建华,张立明. 结合问题特征利用 SE-Tree 反向深度求解冲突集的方法[J]. 电子学报,2017,45(5):1175 – 1181.
OUYANG Dan-tong, LIU Bo-wen, ZHOU Jian-hua, ZHANG Li-ming. A method ofcomputing minimal conflict sets combining the structure property with the anti-depth SE-tree[J]. Acta Electronica Sinica,2017,45(5):1175 – 1181. (in Chinese)
[5] 刘梦,欧阳丹彤,刘伯文,张立明,张永刚. 结合问题特征的分组式诊断方法[J]. 电子学报,2018,46(3):589 – 594.
LIU Meng,OUYANG Dan-tong,LIU Bo-wen,ZHANG Li-ming,ZHANG Yong-gang. Grouped diagnosis approach using the feature of problem[J]. Acta Electronica Sinica, 2018,46(3):589 – 594. (in Chinese)
[6] Reiter R. A theory of diagnosis from first principles[J]. Artificial Intelligence,1987,32(1):57 – 95.
[7] Wotawa F. A variant of Reiter's hitting-set algorithm[J]. Information Processing Letters,2001,(79):45 – 51.
[8] 王肖,赵相福. 用 CHS-tree 基于集合势的方法计算极小碰集[J]. 计算机集成制造系统,2014,20(2):401 – 406.
Wang Xiao,Zhao Xiangfu. Computing minimal hitting sets with CHS-tree method[J]. Computer Integrated Manufacturing Systems. 2014,20(2):401 – 406. (in Chinese)
[9] 姜云飞,林笠. 用对分 HS—树计算最小碰集[J]. 软件学报,2002,13(12):2267 – 2274.
Jiang Yunfei, Lin Li. Computing the minimal hitting sets with binary HS-tree[J]. Journal of Software, 2002, 13(12):2267 – 2274. (in Chinese)
[10] 姜云飞,林笠. 用布尔代数方法计算最小碰集[J]. 计算机学报,2003,26(8):919 – 924.
Jiang Yunfei,Lin Li. The computation of hitting sets with Boolean formulas[J]. Chinese Journal of Computers, 2003,26(8):919 – 924. (in Chinese)
[11] Zhao X F,Ouyang D T. A method of combining SE-tree to compute all minimal hitting sets[J]. Progress in Natural Science,2006,16(2):169 – 174.
[12] Han,B,Lee,S J. Deriving minimal conflict sets by CS-trees with mark set in diagnosis from first principles[J]. IEEE Transactions on Systems, Man, and Cybernetics, 1999,29(2):281 – 286.
[13] Zhao X F,Ouyang D T. Improved algorithms for deriving all minimal conflict sets in model-based diagnosis[A]. Lecture Notes in Computer Science[C]. springer. 2007, 4681:157 – 166.

作者简介

何嫱君 女,1992 年生于河南三门峡. 浙江师范大学软件工程硕士,研究方向为基于模型的故障诊断.
E-mail:1002180265@qq.com

赵相福(通信作者) 男,博士、副教授,1981 年生于山东汶上. 浙江师范大学教师、研究生导师,研究方向为基于模型的故障诊断、智能诊断推理、区块链等.
E-mail:xiangfuzhao@zjnu.cn

欧阳丹彤 女,博士、教授,1968 年生于吉林长春. 吉林大学教师、博士生导师,研究方向为基于模型的故障诊断、自动推理等.
E-mail:ouyd@jlu.edu.cn

张立明 男,博士、高级工程师,1980 年生于吉林长春. 吉林大学教师,研究方向为基于模型的故障诊断等.
E-mail:limingzhang@jlu.edu.cn