计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2018.20160900
55(4):791-801, 2018

基于动态极大元素覆盖值的极小碰集求解算法

邓召勇　欧阳丹彤　耿雪娜　刘　杰
(吉林大学计算机科学与技术学院　长春　130012)
(符号计算与知识工程教育部重点实验室(吉林大学)　长春　130012)
(dengzy19941019@163.com)

Computing the Minimal Hitting Sets with Dynamic Maximum Element Coverage Value

Deng Zhaoyong, Ouyang Dantong, Geng Xuena, and Liu Jie
(*College of Computer Science and Technology, Jilin University, Changchun* 130012)
(*Key Laboratory of Symbolic Computation and Knowledge Engineering (Jilin University), Ministry of Education, Changchun* 130012)

Abstract　The minimal conflict set cluster is constructed by all the minimal conflict sets. The minimal hitting sets of all the minimal conflict sets are the faults of the system to be diagnosed. Therefore, computing all the minimal hitting sets according to the minimal conflict set cluster is an important step in model based diagnosis. In this paper, a new algorithm is proposed to obtain all the minimal hitting sets by using dynamic maximum element coverage value. The algorithm processes elements in turn according to the descending order of the element coverage value. The search space can be greatly reduced by joining the corresponding heuristic strategy and pruning strategy in the process of solving the hitting sets. Adjacency list is used to store the minimal conflict set cluster in the algorithm. Among the basic storage structures, adjacency list has better space cost than matrix. Besides, the adjacent pointer can quickly find the elements in the minimal conflict set cluster which can be covered by an element. When a hitting set is obtained, the minimal hitting set is obtained by using the minimal hitting set decision rule. Therefore, the algorithm can generate and only generate all the minimal hitting sets. Experimental results show that the proposed algorithm has better computational efficiency than other algorithms.

Key words　model-based diagnosis (MBD); minimal conflict set; minimal hitting set; dynamic maximum element coverage value; heuristic strategy; pruning strategy

摘　要　在基于模型诊断(model-based diagnosis, MBD)中,因为所有极小冲突集的极小碰集就是待诊断系统的诊断结果,所以利用所有极小冲突集构造极小冲突集合簇,并基于极小冲突集合簇计算极小碰集是诊断的关键步骤.提出一种基于动态极大元素覆盖值求解极小碰集的新算法.该算法按照元素的元素覆盖值从大到小的顺序依次处理元素,并在求解碰集的过程中加入启发式策略和剪枝策略,使得搜索空间极大减少;利用邻接链表存储输入的极小冲突集合簇,邻接链表相对于用矩阵作为存储结构有较好

收稿日期:2016-11-25;**修回日期**:2017-04-27
基金项目:国家自然科学基金项目(61672261, 61502199, 61402196, 61373052);浙江省自然科学基金项目(LY16F020004)
This work was supported by the National Natural Science Foundation of China (61672261, 61502199, 61402196, 61373052) and the Natural Science Foundation of Zhejiang Province of China (LY16F020004).
通信作者:刘杰(liu_jie@jlu.edu.cn)

的空间开销且通过邻接指向能快速地找到元素可以覆盖的集合簇中的元素;每得到一个碰集便使用极小碰集判定规则进行筛选,因此算法结束时可以产生而且仅产生所有的极小碰集.实验结果表明该算法有较高的计算效率.

关键词 基于模型诊断;极小冲突集;极小碰集;动态极大元素覆盖值;启发式策略;剪枝策略

中图法分类号 TP18

极小碰集的应用领域广泛,很多现实和理论中的问题都可以转化为求解极小碰集问题,例如基于模型诊断(model-based diagnosis, MBD)[1]中极小诊断问题、极小覆盖集问题以及规则冲突检测算法中位向量的碰集问题[2]等.基于模型诊断是人工智能领域里一个非常重要的研究分支,它的主要工作原理是根据系统的描述和观测进行逻辑推理得到冲突部件集合,再通过冲突部件集合计算极小碰集,即得到诊断结果.

迄今为止,国内外学者已对极小碰集求解算法进行了许多研究和优化.最经典的计算极小碰集的算法是1987年由人工智能专家Reiter[1]提出的HS-TREE算法,该算法在求解过程中加入了剪枝策略和关闭策略.HS-TREE算法的缺点是空间复杂度呈指数级增长、产生节点多、效率低,并且会因为剪枝策略导致求得的极小碰集不完备.为了保证极小碰集求解算法的完备性,Greiner等人[3]于1989年提出了HS-DAG算法.HS-DAG算法在HS-TREE算法的基础上优化了剪枝策略,避免了剪掉正确解的情况,但是HS-DAG算法的模型结构比较复杂,空间复杂度高,计算过程较为繁琐.针对这些缺点,姜云飞和林笠[4]于2002年提出了BHS-TREE算法,该算法无需剪枝,求解效率有明显提高,但由于是树形结构,导致该算法在编程实现时存在数据结构复杂、不易实现等缺点.近年来,国内学者陆续提出了一些串行极小碰集求解算法:2003年姜云飞和林笠[5]提出Boolean算法;2006年和2011年欧阳丹彤等人提出HSSE-TREE[6]和DMDSE-TREE[7]算法;2015年王艺源等人[8]提出CSP-MHS算法.此外,近年来也有一些并行及分布式[9-10]极小碰集求解算法被提出,并行及分布式极小碰集求解算法利用多核和多处理器的优势提高极小碰集求解效率.并行及分布式极小碰集求解算法相对串行极小碰集求解算法效率有了较大提高,但是串行算法经过相应修改可以转变为并行及分布式算法,因此本文主要关注串行极小碰集求解算法.串行极小碰集求解算法中效率较为突出的算法是Boolean算法,Boolean算法用布尔代数变量表示待诊断系统的部件,并用布尔代数知识计算极小碰集.目前来看,在布尔代数算法之后提出的串行算法虽然在某些集合簇上有较高的效率,但综合来看,仍是布尔代数算法占优.

布尔代数算法是目前求解极小碰集效率优良的算法,然而当系统很大时,碰集的极小化过程会花费较多时间,因此不适用于规模较大的系统.本文针对布尔代数算法的缺点,引入特定状态下元素的元素覆盖值作为启发式信息,提出了一种基于动态极大元素覆盖值求解极小碰集的新算法MHS-DMECV.本文中元素覆盖值的概念类似于2011年欧阳丹彤等人[7]提出的DMDSE-TREE算法中的度,且MHS-DMECV算法和DMDSE-TREE算法都优先选取覆盖集合簇中元素最多的元素作为碰集的1个元素,但是本文中的元素覆盖值简化了度的定义且MHS-DMECV算法在选取元素时添加了启发式策略和剪枝策略.MHS-DMECV算法的主要优点是:

1) 按照元素的元素覆盖值由大到小的顺序选取元素,选取的元素构成的集合 S 若能构成碰集,则 S 是最快构成碰集的元素集合,否则直接返回对当前元素的处理.

2) 求解过程中添加启发式策略和剪枝策略,使得搜索空间极大减少,且剪枝策略不会丢失极小碰集.

3) 使用邻接链表存储输入的集合簇,邻接链表结构能快速找到元素能够覆盖的集合簇中的元素.

4) MHS-DMECV算法只对邻接链表进行简单遍历,因此程序容易实现.

5) MHS-DMECV算法结束时能保证求得所有的极小碰集.

1 背景知识

本节介绍碰集(hitting set, HS)的相关定义和概念,并给出一个实例说明碰集、极小碰集、容量、势以及频数的具体含义.

定义 1[1]. 碰集. 设 $CS=\{S_i \mid i=1,2,\cdots,n\}$ 是冲突集合簇,称 HS 为 CS 的一个碰集,如果 HS 满足:1) $HS\subseteq \bigcup_{S\in CS} S$;2) 对于任意一个 $S\in CS$,都有 $HS\cap S\neq\varnothing$.

集合簇 CS 的一个碰集是极小碰集(MHS)当且仅当它的任意真子集都不是 CS 的碰集.

定义 2. 容量. 若集合簇 CS 中包含元素的个数为 n,则称 n 为集合簇 CS 的容量,记为 $Cap(CS)=n$.

设 U 是集合簇 CS 中所有元素的并集构成的集合,即 $U=\bigcup S_i=\{e_1,e_2,\cdots,e_m\}$,用 $Car(U)$ 表示集合 U 的势,即集合 U 中元素的个数.

定义 3. 频数. 若 $e\in S$,则称集合 S 含有元素 e,记 $Freq(CS,e)$ 表示集合簇 CS 中含有元素 e 的元素的个数,称为元素 e 在集合簇 CS 中的频数.

例 1. 设集合簇 $CS=\{\{1,2\},\{2,3\},\{3,4\}\}$,则集合簇 CS 的容量 $Cap(CS)=3$;集合 $U=\{1,2,3,4\}$ 的势 $Car(U)=4$;容易看出集合 $\{2,3\}$ 构成集合簇 CS 的一个碰集,并且它还是极小碰集. 若选取元素 $e=2$,则元素 2 在集合簇 CS 中的频数 $Freq(CS,2)=2$,因为元素 2 出现在集合簇 CS 的元素 $\{1,2\}$ 和 $\{2,3\}$ 中.

2 极小碰集求解算法

本节首先给出元素覆盖值、已覆盖数和可覆盖数等概念,然后详细描述算法 JudgeMHS 和 MHS-DMECV.

2.1 相关定义及定理

本节给出元素覆盖值、待处理序列、已覆盖数和可覆盖数的定义,并基于启发式策略、剪枝策略和极小碰集判定规则给出 3 个定理.

定义 4. 元素覆盖值. 若元素 e 在集合簇 CS 的某个元素 $S_i(i=1,2,\cdots,n)$ 中出现,且在当前状态下 S_i 中没有其他元素出现,则称元素 e 覆盖 S_i;若对于集合簇 CS 中的所有元素,元素 e 能覆盖的元素个数为 C,则称 C 为元素 e 的元素覆盖值,记为 $Cover(e)$. 显然 $0\leqslant Cover(e)\leqslant Freq(CS,e)$.

例 2. 对于集合簇 $CS=\{\{1,2\},\{2,3\},\{3,4\}\}$,若首先选择元素 1,则它能覆盖集合 $\{1,2\}$,所以 $Cover(1)=1$;此时再选择元素 2,由于集合 $\{1,2\}$ 已被覆盖,所以元素 2 只能覆盖集合 $\{2,3\}$,因此 $Cover(2)=1$;对于元素 3,由于集合 $\{2,3\}$ 已被覆盖,所以元素 3 只能覆盖集合 $\{3,4\}$,因此 $Cover(3)=1$;对于元素 4,由于集合簇 CS 中的所有元素都已被覆盖,因此 $Cover(4)=0$.

定义 5. 待处理序列. 在当前状态下,待处理的元素集合为 $Pend=\{e_i,e_{i+1},\cdots,e_{j-1},e_j\}$,计算出 $Pend$ 集合中所有元素的元素覆盖值 $Cover(e_k)$ $(k=i,i+1,\cdots,j-1,j)$,并按照元素覆盖值从大到小的顺序排列,此时得到的序列称为待处理序列,记为 Seq.

例 3. 对于集合簇 $CS=\{\{1,2\},\{2,3\},\{3,4\}\}$,初始时没有选中任何元素,则 $Cover(1)=1$,此时恢复到初始状态,即没有选中元素 1 时的状态. 此后每计算出一个元素 e 的元素覆盖值,都将元素 e 造成的影响进行恢复,所以 $Cover(2)=2$,$Cover(3)=2$ 和 $Cover(4)=1$,因此待处理序列 $Seq=[2,3,1,4]$(为了描述方便,假设具有相同元素覆盖值的元素按照元素本身的大小从小到大排序).

定义 6. 已覆盖数和可覆盖数. 在当前状态下,如果集合簇 CS 中已被覆盖的元素个数为 $AlreadyCover$,则称 $AlreadyCover$ 为已覆盖数. 对于待处理序列 Seq,Seq 能覆盖的集合簇 CS 中元素总数为 $CanCover$,则称 $CanCover$ 为可覆盖数.

例 4. 对于集合簇 $CS=\{\{1,2\},\{2,3\},\{3,4\}\}$,假设已经选取元素 2 作为碰集 HS 的一个元素,且 HS 中没有选取其他元素,即 $HS=\{2\}$. 则在当前状态 $state$ 下,集合簇 CS 中的元素 $\{1,2\}$ 和 $\{2,3\}$ 已被元素 2 覆盖,所以已覆盖数 $AlreadyCover=2$;基于当前状态 $state$,计算出待处理元素集合 $Pend=\{1,3,4\}$ 中每个元素的元素覆盖值为 $Cover(1)=0$,$Cover(3)=1$ 和 $Cover(4)=1$(在计算完某一个元素 e 的元素覆盖值后,都将该元素 e 造成的影响恢复,使其回到当前状态 $state$). 注意到在当前状态 $state$ 下集合簇 CS 中只有一个元素 $\{3,4\}$ 能被覆盖,且 $\{3,4\}$ 可以被元素 3 或元素 4 覆盖,所以可覆盖数 $CanCover=1$. 此时新的待处理序列 $Seq'=[3,4,1]$.

定理 1. 对于当前状态 $state$,若已知已覆盖数 $AlreadyCover$ 和可覆盖数 $CanCover$,且 $AlreadyCover+CanCover<Cap(CS)$,则预先选择的元素构成的集合 $Before$ 并上新的待处理序列 Seq' 中任意元素构成的集合 $NewPend$,此时形成的并集 $S=Before\cup NewPend$ 一定不能构成碰集,反之亦然. 显然 $0\leqslant AlreadyCover+CanCover\leqslant Cap(CS)$.

证明.

1) 必要性.

假设 $NewPend$ 表示新的待处理序列 Seq' 中所有元素构成的集合,$S=Before\cup NewPend$. 由于已覆盖数 $AlreadyCover$ 和可覆盖数 $CanCover$ 的和

小于集合簇 CS 的容量，因此 $\exists S' \in CS$，$Before \cap S' = \varnothing$ 且 $NewPend \cap S' = \varnothing$，从而 $S \cap S' = \varnothing$，由碰集定义可知 S 不构成碰集. 又因为 $NewPend$ 表示新的待处理序列 Seq' 中所有元素构成的集合，因此对于 $NewPend$ 的任意子集 $S_u \subseteq NewPend$，$S = Before \cup S_u$ 也一定不能构成碰集，必要性得证.

2) 充分性.

假设 $NewPend$ 表示新的待处理序列 Seq' 中所有元素构成的集合，$S = Before \cup NewPend$. 由于 S 不构成碰集，因此 $\exists S' \in CS$，$S \cap S' = \varnothing$. 将 $S = Before \cup NewPend$ 代入得 $(Before \cap S') \cup (NewPend \cap S') = \varnothing$，即在预先选择的元素集合 $Before$ 以及新的待处理序列 Seq' 中不存在元素 e 使得 e 能够覆盖集合 S'，所以已覆盖数 $AlreadyCover$ 与可覆盖数 $CanCover$ 的和一定小于集合簇 CS 的容量，即 $AlreadyCover + CanCover < Cap(CS)$. 由于 $NewPend$ 表示新的待处理序列 Seq' 中所有元素构成的集合，且 $NewPend$ 中不存在元素 e 能够覆盖集合 S'，因此 $NewPend$ 的任意子集 $S_u \subseteq NewPend$ 中的元素也一定不能覆盖 S'，所以 $AlreadyCover + CanCover < Cap(CS)$ 仍然满足，充分性得证. 证毕.

基于定理 1 和相关定义给出启发式策略和剪枝策略.

1) 启发式策略. 对于当前状态 $state$，若已知已覆盖数 $AlreadyCover$ 和可覆盖数 $CanCover$，且 $AlreadyCover + CanCover < Cap(CS)$，则直接退出对当前状态所在层的处理.

2) 剪枝策略. 对于待处理序列 $Seq = [e_i, e_{i+1}, \cdots, e_{j-1}, e_j]$，从前往后扫描 Seq 时发现位置 k 处元素 e_k 的元素覆盖值 $Cover(e_k) = 0$，则只需考虑位置 k 之前的序列 $Seq_1 = [e_i, e_{i+1}, \cdots, e_{k-1}]$（如果 Seq 中不存在位置 k 使得 $Cover(e_k) = 0$，那么 Seq_1 就指 Seq）.

定理 2. 对于求解集合簇 CS 的极小碰集问题，剪枝策略不会导致丢解.

证明. 从前往后扫描待处理序列 Seq 的过程中，如果发现某个位置 k 使得该位置处元素 e_k 的元素覆盖值 $Cover(e_k) = 0$，由于待处理序列 Seq 是按照元素覆盖值 $Cover(e)$ 从大到小的顺序排列，且 $Cover(e) \geqslant 0$，因此位置 k 及其之后位置对应的元素 $e_t (t = k, k+1, \cdots, j-1, j)$ 的元素覆盖值 $Cover(e_t) = 0$. 由元素覆盖值的定义知，元素 e_t 对覆盖集合簇 CS 中的元素不再有帮助，且极小碰集的定义要求极小碰集 MHS 中的任意 1 个元素 $e' \in MHS$ 都是不可取代的. e' 至少能覆盖集合簇 CS 中的 1 个元素 $S \in CS$，且极小碰集中的其他元素都不能覆盖 S，因此去除 Seq 中位置 k 及其之后位置对应的元素对于计算极小碰集并不会导致丢解，也即可以用 Seq_1 代替 Seq 参与运算. 如果 $Seq_1 = Seq$，显然可以用 Seq_1 代替 Seq. 证毕.

极小碰集判定规则. 假设碰集 $HS = \{e_x, e_{x+1}, \cdots, e_y\}$，若依次去除元素 $e_i \in HS (i = x, x+1, \cdots, y)$ 的过程中，发现得到的集合 HS' 满足碰集定义，则判定 HS 不是极小碰集，否则恢复 e_i 造成的影响，继续处理去除 e_{i+1} 的情况. 若在去除碰集 HS 最后 1 个元素 e_y 时 HS' 仍不满足碰集定义，则判定 HS 是极小碰集.

定理 3. 极小碰集判定规则正确有效.

证明. 假设碰集 $HS = \{e_x, e_{x+1}, \cdots, e_y\}$，若去除 HS 中任意 1 个元素 $e_i (i = x, x+1, \cdots, y)$ 之后得到的集合 HS' 满足碰集定义，则由极小碰集的定义知碰集 HS 一定不是极小碰集，否则 HS 存在是极小碰集的可能性. 若去除碰集 HS 中任意 1 个元素 $e_i (i = x, x+1, \cdots, y)$ 后得到的集合 HS' 都不满足碰集定义，则说明碰集 HS 中每个元素都不可取代，因此判定 HS 是极小碰集. 证毕.

由于本文的算法只能极大限度地缩小搜索的碰集空间，它并不保证求得的碰集一定是极小碰集，因此在给出求解极小碰集的算法 MHS-DMECV 之前，首先给出极小碰集判定算法 JudgeMHS.

2.2 JudgeMHS 算法

本节给出极小碰集判定算法 JudgeMHS 的算法描述.

本文用邻接链表存储输入的集合簇 CS，目的是快速找到集合 U 中特定元素具体能覆盖集合簇 CS 中的哪些元素. 引入 2 个结构 $SetNode$ 和 $ListNode$，其中 $SetNode$ 包含属性 $setNum$（$setNum$ 表示 U 中特定元素能覆盖集合簇 CS 中的第 $setNum$ 个元素）和 $next$（$next$ 指向下一个 $SetNode$ 节点）；$ListNode$ 包含属性 $adjacent$（$adjacent$ 表示 U 中特定元素具体能覆盖的集合簇 CS 中元素的邻接指向）.

例 5. 对于集合簇 $CS = \{\{1,2\}, \{2,3\}, \{3,4\}\}$，由于 $Cap(CS) = 3$，$U = \{1,2,3,4\}$，$Car(U) = 4$，所以需要 4 个 $ListNode$ 节点来表示集合 U 中的 4 个元素，这里用 $Head[4]$ 数组表示（数组索引代表集合 U 中相应元素）. 对于集合簇 CS，用 1 代表第 1 个元素 $\{1,2\}$，2 代表第 2 个元素 $\{2,3\}$，3 代表第 3 个

元素{3,4},因此集合簇 CS 对应的邻接链表结构如图 1 所示:

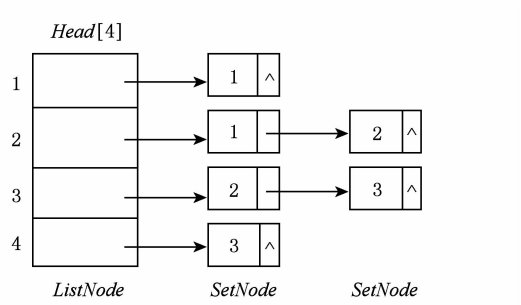

Fig. 1 The adjacency list of the set cluster CS

图 1 集合簇 CS 对应的邻接链表

有了对集合簇 CS 的邻接链表表示形式,下面基于极小碰集判定规则给出极小碰集判定算法 JudgeMHS. $SetFlag$ 数组(大小为集合簇 CS 的容量)维护碰集 HS 能够覆盖的集合簇中的元素,$SetFlag[i](i=1,2,\cdots,Cap(CS))$ 初始为 0 表示未覆盖,大于等于 1 表示覆盖(等于 1 表示 HS 中只有 1 个元素能覆盖 i 对应的集合簇中的元素,大于 1 表示 HS 中存在多个元素能覆盖 i 对应的集合簇中的元素);逻辑变量 $Flag$ 标志 HS 是否是极小碰集,false 表示 HS 不是极小碰集,true 表示 HS 是极小碰集.

算法 1. JudgeMHS.

输入:碰集 HS、集合簇 CS 的邻接链表 $Head$ 数组;

输出:碰集 HS 是否是极小碰集,若是返回 true,否则返回 false.

① 将 $SetFlag$ 数组中每个元素的值初始化为 0,执行步②.

② 循环处理 HS 中的元素 e,循环未结束时利用 $Head[e]$的邻接指向获取元素 e 能覆盖的集合簇中的元素 $setNum$,将 $SetFlag[setNum]+1$,返回步②,否则执行步③.

③ 循环处理 HS 中的元素 e,循环未结束时利用 $Head[e]$的邻接指向获取元素 e 能覆盖的集合簇中的元素 $setNum$,将 $SetFlag[setNum]-1$,即去除元素 e,执行步④,否则返回 true.

④ 置 $Flag=$false,循环处理 $SetFlag$ 数组中的元素 $SetFlag[i]$,循环未结束时如果 $SetFlag[i]=0$,则置 $Flag=$true,退出当前循环并执行步⑤,否则继续执行循环;循环结束时执行步⑤.

⑤ 如果 $Flag=$false,则说明去除碰集中的元素 e 后构成的集合仍然满足碰集定义,则 HS 不是极小碰集,返回 false,否则执行步⑥.

⑥ 利用 $Head[e]$的邻接指向获取元素 e 能覆盖的集合簇中的元素 $setNum$,将 $SetFlag[setNum]+1$,即恢复元素 e 对 $SetFlag$ 数组造成的影响,执行步③.

2.3 MHS-DMECV 算法

本节给出 MHS-DMECV 算法的算法描述并分析其完备性和复杂性.

MHS-DMECV 算法中 $HittingSet$ 用于在算法执行过程中动态生成碰集,$MinHittingSet$ 用于存储所有的极小碰集.

算法 2. MHS-DMECV.

输入:待处理序列 Seq、与 Seq 对应的邻接链表 $Head$ 数组、与 Seq 对应的元素覆盖值 $Cover$ 数组、动态保存碰集的容器 $HittingSet$、集合簇 CS 的元素覆盖标志 $SetFlag$ 数组(初始时每个元素的值都为 0)、已覆盖数 $AlreadyCover$(初始为 0);

输出:集合簇 CS 对应的所有极小碰集容器 $MinHittingSet$.

① 循环处理 Seq 中的元素 e,执行步②,循环结束则返回.

② 如果 e 是 Seq 中最后一个元素,且 $Cover[e]+AlreadyCover<Cap(CS)$,则返回;否则执行步③.

③ 如果 $Cover[e]+AlreadyCover=Cap(CS)$,将 $HittingSet$ 中元素存储到一个新的容器 $container$ 里并将元素 e 也加入 $container$. 以 $container$ 作为输入调用 JudgeMHS,若返回 true,则将 $container$ 加入 $MinHittingSet$,返回步①处理 Seq 中下一个元素. 若 $Cover[e]+AlreadyCover<Cap(CS)$,执行步④.

④ 将元素 e 加入 $HittingSet$,遍历 $Head[e]$的邻接指向,若集合簇 CS 中存在元素 $setNum$ 的覆盖标志 $SetFlag[setNum]=0$ 且 e 能覆盖 $setNum$,则将 $AlreadyCover+1$;将 $Head[e]$能遍历到的集合簇 CS 中对应元素的覆盖标志+1,执行步⑤.

⑤ 构建 2 个数据结构 Seq' 和 $Head'$,其中 Seq' 表示在元素 e 已被选中的状态下 Seq 中在 e 之后的序列里元素覆盖值不为 0 的元素集合,此时 Seq' 对应的元素覆盖值数组为 $Cover'$;$Head'$ 数组构建 1 个对应 Seq' 的邻接链表,即只有在 Seq' 中的元素在 $Head'$ 数组中才会有邻接指向,且 $Head'$ 数组中元素的邻接指向指向的 $SetNode$ 节点集正是 Seq' 中

相应元素能覆盖的集合簇 CS 中的元素集对应的 $SetNode$ 节点集. 构建可覆盖数 $CanCover$(初始为0),$CanCover$ 表示在元素 e 已被选中的状态下,Seq'可以覆盖的集合簇 CS 中的元素数,执行步⑥.

⑥ 如果 $AlreadyCover+CanCover<Cap(CS)$,则从 $HittingSet$ 中移除元素 e,遍历 $Head[e]$的邻接指向,将 $Head[e]$能遍历到的集合簇 CS 中对应元素的覆盖标志减 1,若集合簇 CS 中存在元素 $setNum$ 的覆盖标志 $SetFlag[setNum]=0$ 且 e 能覆盖 $setNum$,则将 $AlreadyCover-1$,返回;若 $AlreadyCover+CanCover=Cap(CS)$,执行步⑦.

⑦ 按照元素覆盖值从大到小对 Seq'进行排序,得到新的待处理序列 Seq';将 Seq'、Seq'对应的邻接链表 $Head'$数组、$Cover'$数组、$HittingSet$、$SetFlag$数组以及 $AlreadyCover$ 作为输入递归调用 MHS-DMECV. 递归返回时从 $HittingSet$ 中移除元素 e,遍历 $Head[e]$的邻接指向,将 $Head[e]$能遍历到的集合簇 CS 中对应元素的覆盖标志减 1,若集合簇 CS 中存在元素 $setNum$ 的覆盖标志 $SetFlag[setNum]=0$ 且 e 能覆盖 $setNum$,则将 $AlreadyCover-1$,返回步①继续处理 Seq 中下一个元素.

MHS-DMECV 算法结束时,$MinHittingSet$ 包含所有极小碰集,下面从理论上对 MHS-DMECV 算法的完备性和复杂性进行分析.

1) 完备性. 在循环处理待处理序列 Seq 中的元素 e 时(不考虑启发式策略和剪枝策略),由于是递归处理,因此它总能枚举出所有可能的集合. 加入启发式策略会得到最先能够构成碰集的碰集集合 HSS,且 HSS 是极小碰集集合的超集,因此能保证在算法结束时得到所有的极小碰集. 又因为剪枝策略并不会导致丢失极小碰集,所以 MHS-DMECV 算法对于求解集合簇 CS 的极小碰集问题是完备的.

2) 复杂性. 假设 MHS-DMECV 算法的总运行时间用 T 表示(此时不考虑启发式策略和剪枝策略),由于初始时需要执行 m 次循环(m 表示集合 U 的势),因此 $T=T(1)+T(2)+\cdots+T(m)$. 在每一次循环中都遍历一遍邻接链表和对相应元素排序,这个子过程的时间复杂度可以用 $T_{sub}=O(mn+m\,\mathrm{lb}\,m)$ 表示,其中 n 表示集合簇 CS 的容量;又 $T(1)=T_{sub}+T(2)+T(3)+\cdots+T(m)$,$T(2)=T_{sub}+T(3)+T(4)+\cdots+T(m)$,$\cdots$,$T(m-1)=T_{sub}+T(m)$,$T(m)=O(1)$,所以 $T=O(T_{sub}2^m)$,即算法最坏时间复杂度是指数级;由于需要存储所有极小碰集,且在最坏情况时极小碰集数是指数级增长,因此算法的空间复杂度在最坏情况时是指数级. 在 MHS-DMECV 算法中,因为每次处理的元素 e_i 的元素覆盖值都是待处理序列 Seq 中最大的,这种策略能规避掉很多不必要的处理. 例如元素 e_1 能覆盖集合簇 CS 中的所有元素,元素 e_2 只能覆盖集合簇 CS 中的 1 个元素. 如果首先处理 e_1,则不需要再递归处理 e_2,否则由于处理到 e_2 时,发现单独由 e_2 构成的集合 $\{e_2\}$并不足以构成碰集,且在 e_2 之后的序列中存在元素构成的集合并上$\{e_2\}$能构成碰集,则会继续递归处理. 考虑启发式策略,如果选择元素 e_i 放入动态碰集容器 $HittingSet$ 后,发现 $AlreadyCover+CanCover<Cap(CS)$,即从位置 i 之后的序列里不存在元素集合并上 $HittingSet$ 能构成碰集,此时可以直接返回,从而 $T(i)=T_{sub}$. 考虑剪枝策略,那么 $T(i)=T_{sub}+T(i+1)+T(i+2)+\cdots+T(j)$,且 $j\leqslant m$. 一般来说,启发式策略和剪枝策略都能很好地发挥作用,因此添加启发式策略和剪枝策略后,本文的 MHS-DMECV 算法有较好的效率.

3 实例分析

本节基于一个实例对 MHS-DMECV 算法的执行流程进行分析.

例 6. 对于集合簇 $CS=\{\{1,2\},\{2,3\},\{3,4\}\}$,利用 MHS-DMECV 算法求 CS 的所有极小碰集.

由例 3 知初始时待处理序列 $Seq=[2,3,1,4]$,邻接链表 $Head$ 数组如图 2 所示,元素覆盖值数组 $Cover=[2,2,1,1]$,$HittingSet$ 和 $MinHittingSet$ 为空,集合簇 CS 的元素覆盖标志 $SetFlag=[0,0,0]$,已覆盖数 $AlreadyCover=0$. 此时的状态如表 1 和图 2 所示,将元素 2 添加到 $HittingSet$ 之后,此时的状态对应于表 2 和图 3.

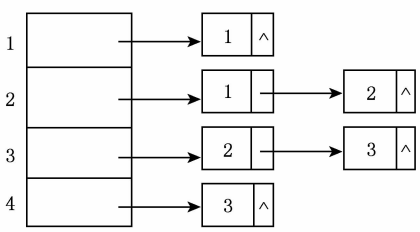

Fig. 2 The corresponding adjacency list about Seq in table 1

图 2 表 1 中 Seq 对应的邻接链表

Table 1 Value of Each Variable at the Beginning

表 1 初始时各变量对应的值

Seq	*Cover*	*SetFlag*	*HittingSet*	*MinHittingSet*	*AlreadyCover*	*CanCover*
2	2	0	{}	{}	0	null
3	2	0				
1	1	0				
4	1					

Table 2 Value of Each Variable When Adding 2 into *HittingSet*

表 2 将元素 2 加入 *HittingSet* 时各变量对应的值

Seq	*Cover*	*SetFlag*	*HittingSet*	*MinHittingSet*	*AlreadyCover*	*CanCover*
3	1	1	{2}	{}	2	1
4	1	1				
		0				

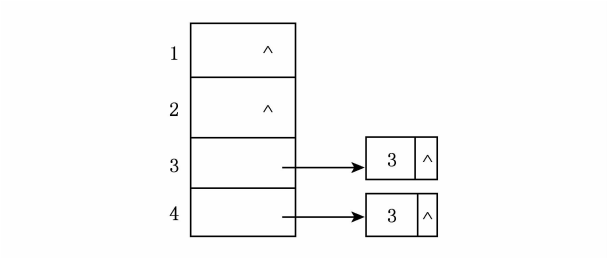

Fig. 3 The corresponding adjacency list about *Seq* in table 2

图 3 表 2 中 *Seq* 对应的邻接链表

处理本层元素 3 时,发现 $AlreadyCover+Cover[3]=3=Cap(CS)$,因此新建 1 个容器 *container* 存储 *HittingSet* 中的元素 2,并将元素 3 也添加到 *container* 中,以 *container* 作为输入调用 JudgeMHS,容易看出 JudgeMHS 返回 true,因此将 *container* 加入 *MinHittingSet*. 元素 4 与元素 3 处理类似,所以当本层循环处理完毕时 *MinHittingSet* 中包含极小碰集{2,3}和{2,4}. 回到上一层处理元素 3,此时的状态对应于表 3 和图 4.

Table 3 Value of Each Variable When Processing 3

表 3 处理元素 3 时各变量对应的值

Seq	*Cover*	*SetFlag*	*HittingSet*	*MinHittingSet*	*AlreadyCover*	*CanCover*
1	1	0	{3}	{2,3},{2,4}	2	1
		1				
		1				

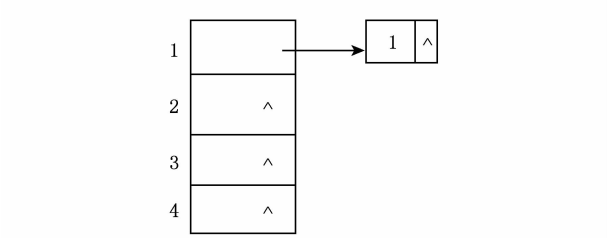

Fig. 4 The corresponding adjacency list about *Seq* in table 3

图 4 表 3 中 *Seq* 对应的邻接链表

处理本层元素 1 与前面处理元素 3 和 4 类似,所以本层循环结束时 *MinHittingSet* 中包含极小碰集{2,3},{2,4}和{3,1}. 回到上一层处理元素 1,此时的状态对应于表 4 和图 5.

处理本层元素 1 时发现 $AlreadyCover+CanCover<Cap(CS)$,因此算法返回上一层,由于本层已是第 1 层,所以 MHS-DMECV 算法完成了对集合簇 *CS* 中所有极小碰集的求解,即 *MinHittingSet* 中包含 3 个极小碰集,它们分别是{2,3},{2,4}和{3,1}.

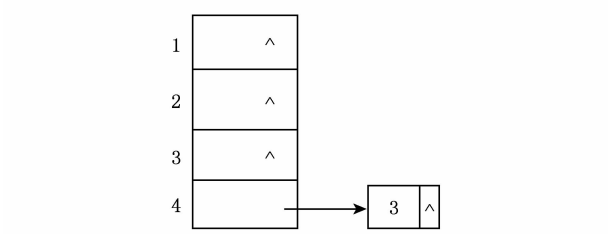

Fig. 5 The corresponding adjacency list about *Seq* in table 4

图 5 表 4 中 *Seq* 对应的邻接链表

Table 4 Value of Each Variable When Processing 1

表 4 处理元素 1 时各变量对应的值

Seq	*Cover*	*SetFlag*	*HittingSet*	*MinHittingSet*	*AlreadyCover*	*CanCover*
4	1	1	{1}	{2,3},{2,4},{3,1}	1	1
		0				
		0				

4 实验分析

本节对 MHS-DMECV 算法的性能进行实验分析. 在实验对比部分,选取目前效率优良的布尔代数算法[5]作为比较对象. 实验平台如下:Windows 7 操作系统,CPU Intel Core i7-3770 3.4 GHz,8.00 GB RAM,Java.

本文测试用例由伪随机集合簇生成器产生,伪随机集合簇生成器输入参数包括元素个数 m(m 表示集合 U 的势)、集合簇容量 n 以及元素在一个集合簇元素中出现的概率 p. 在同一个测试用例中,所有元素的 p 值均相等,因此集合簇中每个元素包含元素的期望个数等于 mp. 对于元素个数为 4、集合簇容量为 3 的测试用例用 M4N3 表示.

本文用伪随机集合簇生成器生成了 4 类测试用例,分别为 M15N200, M20N200, M25N200 以及 M30N200. 其中每类测试用例包含 10×17 个用例,即每类测试用例又细分为 10 组,各小组均包含 p 取值 0.15～0.94 的 17 个测试用例. 所有测试用例中集合簇容量均为 200. 在 10 组测试用例中对取相同概率 p 的 10 个测试用例进行实验,取其平均执行时间作为实验结果.

图 6 描述了在测试用例为 M15N200 时布尔代数算法与 MHS-DMECV 算法的性能对比图,表 5 是与图 6 对应的实验统计结果. 图 6 中横轴表示元素在 1 个集合簇元素中出现的概率 p;右侧纵轴表示对应实例的极小碰集数量. 表 5 中 Number of *MHSs* 表示在 10 组测试用例中对应 p 取值 0.15～0.94 时的平均极小碰集个数;Speedup Ratio 是布尔代数算法与 MHS-DMECV 算法平均运行时间(精确到微秒)的比值,即加速比.

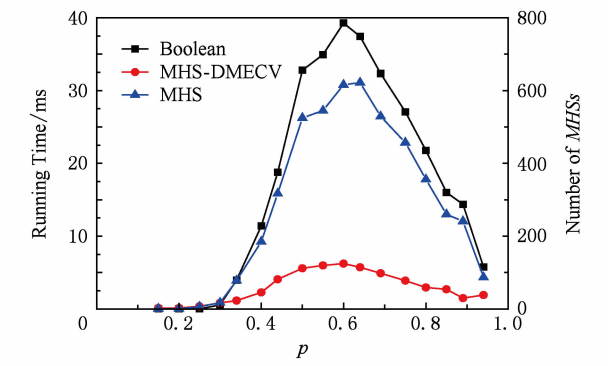

Fig. 6 Performance comparison of Boolean and MHS-DMECV under M15N200

图 6 M15N200 下 Boolean 与 MHS-DMECV 性能对比图

Table 5 Experimental Statistical Results of Boolean and MHS-DMECV Under M15N200

表 5 M15N200 下 Boolean 与 MHS-DMECV 实验统计结果

p	Number of *MHSs*	Boolean-Time/ms	MHS-DMECV-Time/ms	Speedup Ratio
0.15	1	0.003	0.122	0.03
0.20	1	0.003	0.125	0.03
0.25	6	0.014	0.357	0.04
0.30	17	0.556	0.830	0.67
0.34	78	3.950	1.150	3.43
0.40	185	11.391	2.273	5.01
0.44	317	18.762	4.067	4.61
0.50	525	32.798	5.580	5.88
0.55	545	34.929	5.977	5.84
0.60	616	39.270	6.222	6.31
0.64	622	37.435	5.721	6.54
0.69	529	32.333	4.904	6.59
0.75	457	27.083	3.886	6.97
0.80	356	21.774	2.942	7.40
0.85	260	16.009	2.696	5.94
0.89	241	14.372	1.487	9.66
0.94	87	5.750	1.906	3.02

由图 6 和表 5 可知,在极小碰集个数较少时布尔代数算法效率优于 MHS-DMECV 算法,这是因为 MHS-DMECV 算法需要遍历邻接链表以及对待处理序列中的元素排序,这两个操作的时间占比在极小碰集个数较少时会相对突出. 但随着极小碰集个数的增长,MHS-DMECV 算法优势逐渐显现出来,在元素出现概率 $p=0.5$ 时,MHS-DMECV 算法平均比布尔代数算法快 5～6 倍.

图 7 和表 6 对应测试用例为 M20N200 的情况.

由图 7 和表 6 可知,在极小碰集个数达到上千个时,MHS-DMECV 算法执行时间较布尔代数算法明显降低.对比 M15N200 的实验结果可以看出,在随机情况下极小碰集个数与元素数 m 是正相关的.为了对比数据规模较大时 2 种算法的性能差异,图 8 和表 7 给出 M25N200 情况下的性能对比图和实验统计结果.

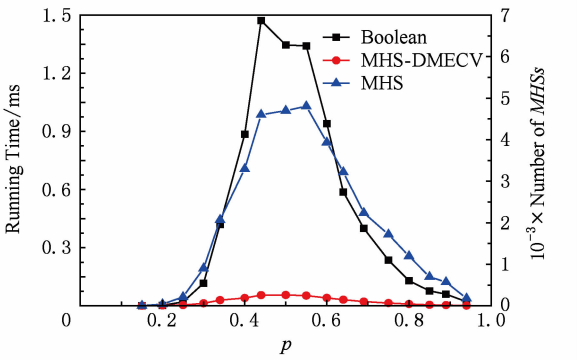

Fig. 7 Performance comparison of Boolean and MHS-DMECV under M20N200

图 7 M20N200 下 Boolean 与 MHS-DMECV 性能对比图

Table 6 Experimental Statistical Results of Boolean and MHS-DMECV Under M20N200

表 6 M20N200 下 Boolean 与 MHS-DMECV 实验统计结果

p	Number of *MHSs*	Boolean-Time/ms	MHS-DMECV-Time/ms	Speedup Ratio
0.15	4	0.054	0.499	0.11
0.20	41	2.452	2.236	1.10
0.25	215	20.096	3.969	5.06
0.30	902	116.596	13.015	8.96
0.34	2071	419.887	29.426	14.27
0.40	3295	884.805	40.337	21.94
0.44	4602	1471.299	55.018	26.74
0.50	4700	1345.692	55.844	24.10
0.55	4806	1340.954	52.468	25.56
0.60	3934	940.625	40.869	23.02
0.64	3221	587.225	31.595	18.59
0.69	2238	399.317	21.790	18.33
0.75	1719	235.683	14.496	16.26
0.80	1198	129.707	9.096	14.26
0.85	698	77.087	5.165	14.92
0.89	573	60.355	3.656	16.51
0.94	179	21.220	1.094	19.40

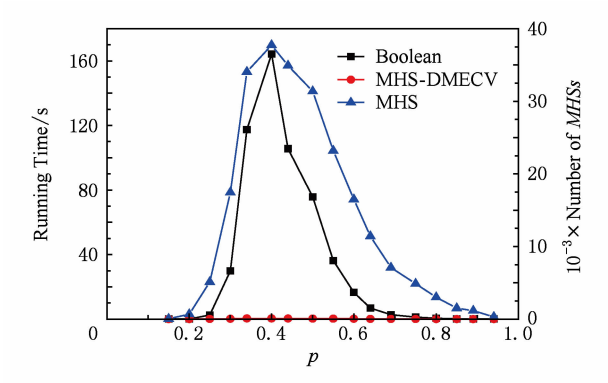

Fig. 8 Performance comparison of Boolean and MHS-DMECV under M25N200

图 8 M25N200 下 Boolean 与 MHS-DMECV 性能对比图

Table 7 Experimental Statistical Results of Boolean and MHS-DMECV Under M25N200

表 7 M25N200 下 Boolean 与 MHS-DMECV 实验统计结果

p	Number of *MHSs*	Boolean-Time/s	MHS-DMECV-Time/s	Speedup Ratio
0.15	46	0.001 766	0.002 114	0.84
0.20	670	0.054 107	0.012 829	4.22
0.25	5 114	2.508 194	0.076 174	32.93
0.30	17 456	29.893 360	0.227 184	131.58
0.34	34 052	117.499 843	0.388 214	302.67
0.40	37 747	164.318 207	0.406 870	403.86
0.44	34 917	105.598 795	0.368 531	286.54
0.50	31 386	75.730 647	0.318 316	237.91
0.55	23 201	36.295 324	0.237 121	153.07
0.60	16 496	16.544 366	0.179 769	92.03
0.64	11 413	6.821 990	0.111 420	61.23
0.69	7 105	2.744 902	0.070 021	39.20
0.75	4 873	1.199 593	0.042 079	28.51
0.80	3 013	0.499 788	0.024 810	20.14
0.85	1 506	0.157 540	0.011 739	13.42
0.89	1 161	0.086 486	0.006 882	12.57
0.94	300	0.016 331	0.002 032	8.04

从图 8 和表 7 可知,在极小碰集个数达到上万级别时,布尔代数算法需要几十甚至上百秒才能计算出所有极小碰集,而 MHS-DMECV 算法在零点几秒内就得到了结果.在概率 $p=0.5$ 时,MHS-DMECV 算法比布尔代数算法快 200 倍左右.最后对比 1 组数据规模更大的数据,即 M30N200 这类测试用例,图 9 和表 8 给出 M30N200 情况下的性能对比图和实验统计结果.

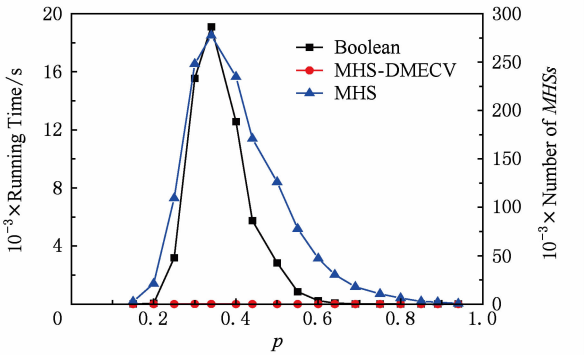

Fig. 9 Performance comparison of Boolean and MHS-DMECV under M30N200

图 9 M30N200 下 Boolean 与 MHS-DMECV 性能对比图

在 M30N200 类数据对比中,容易看出在极小碰集个数达到几十万的规模时,布尔代数算法在可容忍时间内已经失效;反观 MHS-DMECV 算法,即使极小碰集个数达到几十万的规模,MHS-DMECV 算法也能在几秒之内得出结果. 在概率 $p=0.5$ 时,MHS-DMECV 算法比布尔代数算法快 2 000 倍左右.

Table 8 The Experimental Statistical Results of Boolean and MHS-DMECV Under M30N200

表 8 M30N200 下 Boolean 与 MHS-DMECV 实验统计结果

p	Number of *MHSs*	Boolean-Time/s	MHS-DMECV-Time/s	Speedup Ratio
0.15	2 575	1.220 751	0.051 354	23.77
0.20	21 172	50.248 502	0.361 026	139.18
0.25	109 402	3 190.370 604	1.533 166	2 080.90
0.30	248 207	15 551.148 400	3.189 440	4 875.82
0.34	277 390	19 100.517 690	3.375 260	5 658.98
0.40	234 817	12 559.294 920	2.654 243	4 731.78
0.44	170 883	5 739.942 604	1.741 878	3 295.26
0.50	126 061	2 845.279 210	1.227 208	2 318.50
0.55	77 818	853.838 424	0.723 070	1 180.85
0.60	47 205	238.677 822	0.461 778	516.87
0.64	30 358	65.765 575	0.287 653	228.63
0.69	17 894	16.344 207	0.168 364	97.08
0.75	10 507	4.466 864	0.095 731	46.66
0.80	6 208	1.394 066	0.050 244	27.75
0.85	2 787	0.295 333	0.023 831	12.39
0.89	2 049	0.098 390	0.011 976	8.22
0.94	489	0.007 429	0.003 210	2.31

从上面的 4 类数据对比中得出,虽然在某些小数据上布尔代数算法占有优势,但随着数据规模的增大,MHS-DMECV 算法具有非常优良的性能开销. 综合来看,MHS-DMECV 算法较布尔代数算法更具有实用价值. 下面给出近年来的一些极小碰集求解算法与 MHS-DMECV 算法的比较.

由于冲突集合簇的极小碰集求解问题是 NP 难问题,因此目前国内外的算法都是试图避开无用路径的搜索. 理想情况是,对于一个给定的冲突集合簇,算法搜索的每条路径上元素构成的集合直接形成一个极小碰集,即算法不需要做无用功. DMDSE-TREE 算法[7]利用极大度来尽量避免非极小碰集路径的搜索,但是 DMDSE-TREE 算法没有引入其他较好的剪枝策略,因此效率没有较大提高. MHS-DMECV 算法除了使用动态极大元素覆盖值规避掉大量无用路径的搜索之外,还引入了启发式策略和剪枝策略进一步缩小搜索空间. CSP-MHS[8]算法将极小碰集求解问题转换为约束可满足问题,并调用相应的求解器求解极小碰集. 该算法具有较好的创新性,但是在转换的过程中会丢失一些可以加快极小碰集求解效率的信息,因此 CSP-MHS 算法能正确地解决问题,但是 CSP-MHS 算法在效率上有所欠缺.

5 结束语

本文提出基于动态极大元素覆盖值求取所有极小碰集的 MHS-DMECV 算法,求解效率较高. MHS-DMECV 算法引入特定状态下元素的元素覆盖值作为启发式信息,并通过邻接链表存储元素覆盖信息完成极小碰集的求解. MHS-DMECV 算法的主要优点有 4 个方面:

1) 使用邻接链表存储特定状态下的元素覆盖信息,对邻接链表只做简单遍历操作,因此算法效率较高,程序易于实现;

2) 将特定状态下的元素覆盖值从高到低排序,并按照这个顺序选择元素,因此能减少不必要的搜索,提高求解效率;

3) MHS-DMECV 算法添加了启发式策略和剪枝策略,算法效率大幅提高;

4) 产生碰集 *HS* 时使用 JudgeMHS 算法判定 *HS* 是否是极小碰集,因此算法结束时能保证得到所有的极小碰集.

实验结果表明:MHS-DMECV 算法性能优良,对于极小碰集求解问题有较高的求解效率.

本文提出的 MHS-DMECV 算法除了应用在基于模型的诊断领域外,还可以将其应用到极小覆盖

集问题、智能规划和软件调试中.针对具体问题的特点,将本文方法设计为相适应的算法,在特定问题上取得更理想的效果.

参 考 文 献

[1] Reiter R. A theory of diagnosis from first principles [J]. Artificial Intelligence, 1987, 32(1): 57-96

[2] Li Lin, Lu Xianliang. An algorithm for detecting filters conflicts based on the intersection of bit vectors [J]. Journal of Computer Research and Development, 2008, 45(2): 237-245 (in Chinese)
(李林, 卢显良. 一种基于位向量交集运算的规则冲突检测算法[J]. 计算机研究与发展, 2008, 45(2): 237-245)

[3] Greiner R, Smith B A, Wilkerson R W. A correction to the algorithm in Reiter's theory of diagnosis [J]. Artificial Intelligence, 1989, 41(1): 79-88

[4] Jiang Yunfei, Lin Li. Computing the minimal hitting sets with BHS-tree [J]. Journal of Software, 2002, 13(12): 2267-2274 (in Chinese)
(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)

[5] Jiang Yunfei, Lin Li. The computation of minimal hitting sets with Boolean formulas [J]. Chinese Journal of Computers, 2003, 26(8): 919-924 (in Chinese)
(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)

[6] Zhao Xiangfu, Ouyang Dantong. A method of combining SE-tree to compute all minimal hitting sets [J]. Progress in Natural Science, 2006, 16(2): 169-174

[7] Zhang Liming, Ouyang Dantong, Zeng Hailin. Computing the minimal hitting sets based on dynamic maximum degree [J]. Journal of Computer Research and Development, 2011, 48(2): 209-215 (in Chinese)
(张立明, 欧阳丹彤, 曾海林. 基于动态极大度的极小碰集求解方法[J]. 计算机研究与发展, 2011, 48(2): 209-215)

[8] Wang Yiyuan, Ouyang Dantong, Zhang Liming, et al. A method of computing minimal hitting sets using CSP [J]. Journal of Computer Research and Development, 2015, 52(3): 588-595 (in Chinese)
(王艺源, 欧阳丹彤, 张立明, 等. 利用CSP求解极小碰集的方法[J]. 计算机研究与发展, 2015, 52(3): 588-595)

[9] Jannach D, Schmitz T, Shchekotykhin K. Parallelized hitting set computation for model-based diagnosis [C] //Proc of the 29th AAAI Conf on Artificial Intelligence. Menlo Park, CA: AAAI, 2015: 1503-1510

[10] Zhao Xiangfu, Ouyang Dantong. Deriving all minimal hitting sets based on join relation [J]. IEEE Trans on Systems, Man, and Cybernetics: Systems, 2015, 45(7): 1063-1076

Deng Zhaoyong, born in 1994. Master candidate at Jilin University. His main research interest is model-based diagnosis.

Ouyang Dantong, born in 1968. Professor and PhD supervisor of Jilin University. Senior member of CCF. Her main research interests include model-based diagnosis, model checking and automated reasoning (ouyangdantong@163.com).

Geng Xuena, born in 1988. PhD at Jilin University. Her main research interest is model-based diagnosis (183267037@qq.com).

Liu Jie, born in 1973. Associate professor at Jilin University. Her main research interests include model-based diagnosis, model checking and Boolean satisfiability.