软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software [doi: 10.13328/j.cnki.jos.007227]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

BWSS: 结合可疑集合簇计算极小碰集的 Boolean 算法*

赵相福[1], 黄 森[2], 魏 霞[1], 童向荣[1], 欧阳丹彤[3], 张立明[3]

[1](烟台大学 计算机与控制工程学院, 山东 烟台 264005)

[2](浙江师范大学 计算机系, 浙江 金华 321004)

[3](吉林大学 计算机科学与技术学院, 吉林 长春 130012)

通信作者: 赵相福, E-mail: xiangfuzhao@163.com

摘 要: 在基于模型的诊断领域中, 因为极小冲突集的极小碰集即为待诊断设备的候选诊断, 所以计算极小碰集是候选诊断的一个关键步骤. 其中, 极小碰集是一个 NP-hard 约束求解问题, 随着问题规模增大, 求解难度成指数级增长. Boolean 算法是计算极小碰集的经典算法, 然在求解过程中, 解集的极小化却占据运算的绝大部分时间. 为解决该问题并提升计算效率, 本文提出了结合可疑集合簇计算极小碰集的 BWSS 算法, 通过深度分析 Boolean 算法生成树规则, 找到使候选解成为超集的集合, 在向根节点扩展元素时, 如果候选解与可疑集合簇中至少一个集合交集为空, 那么该解为极小候选解, 否则删除该解, 通过递归的策略保证算法结束时产生且仅产生所有极小碰集. 除此之外, 每个候选解在极小化时, 至少存在 $m(m \geqslant 1)$ 个元素甚至整个解无需极小化. 理论上, BWSS 算法的复杂度要远低于 Boolean 算法. 通过随机数据及大量基准电路数据, 实验结果表明本文所提算法与目前最先进的几种算法相比, 运行时间减少了几个数量级.

关键词: 基于模型诊断; 极小碰集; Boolean 算法; 候选解; 冲突集

中图法分类号: TP18

中文引用格式: 赵相福, 黄森, 魏霞, 童向荣, 欧阳丹彤, 张立明. BWSS: 结合可疑集合簇计算极小碰集的Boolean算法. 软件学报. http://www.jos.org.cn/1000-9825/7227.htm

英文引用格式: Zhao XF, Huang S, Wei X, Tong XR, Ouyang DT, Zhang LM. BWSS: Boolean Algorithm with Suspicious Set Clusters for Calculating Minimal Hitting Sets. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/7227.htm

BWSS: Boolean Algorithm with Suspicious Set Clusters for Calculating Minimal Hitting Sets

ZHAO Xiang-Fu[1], HUANG Sen[2], WEI Xia[1], TONG Xiang-Rong[1], OUYANG Dan-Tong[3], ZHANG Li-Ming[3]

[1](School of Computer and Control Engineering, Yantai University, Yantai 264005, China)

[2](Department of Computer, Zhejiang Normal University, Jinhua 321004, China)

[3](College of Computer and Science and Technology, Jilin University, Changchun 130012, China)

Abstract: In the field of model-based diagnosis, all minimal hitting sets(MHS) of minimal conflict sets(MCS) are the candidate diagnoses of the device to be diagnosed, so the calculation of MHS is a key step for generating candidate diagnoses. MHS is a classic NP-hard constraint solving problem. The bigger the problems get, the harder it becomes exponentially to solve them. Boolean algorithm is typical in calculating MHS. However, in the process of solving, most of the runtime is taken up by the minimization of the intermediate solution sets. This study proposes BWSS Algorithm combined with suspicious set clusters for calculating MHS. By analyzing the spanning tree rule of Boolean algorithm in depth, the set that causes the candidate solution to become a superset is found. When extending elements to the root node, the candidate solution, if discovered to share at least one empty set with the suspicious set cluster, shall be minimal. Otherwise, the solution will be removed. The recursive strategy will be employed to ensure that all and only MHS are generated at the end of the algorithm. In addition, each candidate solution has at least m ($m \geqslant 1$) elements or even the entire solution in no need of complex

* 基金项目: 国家自然科学基金 (61972360, 62076108, 62072392)
收稿时间: 2022-05-05; 修改时间: 2024-01-30; 采用时间: 2024-05-15; jos 在线出版时间: 2024-07-03

minimization. Theoretically, BWSS Algorithm is far less complex than Boolean Algorithm. According to random data and mass reference circuit data, Experimental results show that compared with many other state-of-the-art methods, the proposed algorithm reduces several orders of magnitude in runtime.

Key words: model-based diagnosis; minimal hitting set; Boolean algorithm; candidate; conflict set

随着系统规模及复杂程度不断增加, 当新兴人造设备 (核技术、空间探索、智能汽车等) 出现故障时, 由于专家经验有限, 基于经验的诊断技术已不再适用. 为了快速找到故障点, 大量专家利用系统结构、行为、状态、功能等方面的知识, 提出基于模型的诊断 (Model-Based Diagnosis, MBD)[1]. MBD 主要分为两个步骤: 冲突识别和候选诊断. 如图 1, 冲突识别是根据待诊断设备的描述和实际观测产生极小冲突集, 设备的候选诊断是求解极小冲突集的极小碰集. 因为冲突识别可以根据系统已知结构和当前观测的信息离线求出极小冲突集[2−3], 所以整个诊断问题的求解效率将取决于极小碰集求解算法.

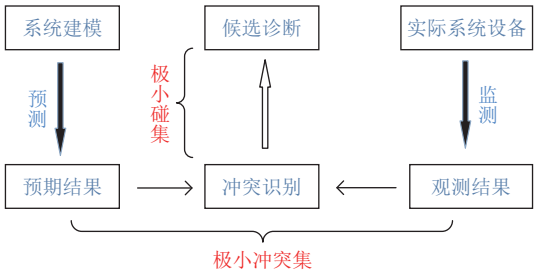


图 1 基于模型诊断流程图

极小碰集求解是人工智能领域的一个重要课题, 最具有代表性的工作是人工智能专家 Reiter 在故障诊断中的应用, 其提出了 HS-Tree 算法[1], 然而可能会因剪枝而丢失解的完备性. 1989 年 Greiner 在 HS-Tree 算法的基础上提出了 HS-DAG 算法, 利用有向无环图结构对相同节点进行重用, 保证了算法的完备性[4]. 2001 年 Wotawa 等人对 HS-DAG 算法进行了优化, 提出了 HST-Tree 算法[5], 通过对元素的唯一标识, 不再需要有向无环图也能保证算法的完备性, 同时减少了大量冗余节点的生成.

2003 年姜云飞、林笠提出了 Boolean 算法[6], 通过布尔规则来产生所有极小碰集, 在很多情况下都优于当前其它集中式的求解算法[7], 其后, 该算法的许多优化算法及相似算法相继被提出[8]. 比如, Abreu 等人提出的 STACCATO 算法其原理和 Boolean 算法类似[9], 该算法通过设置求解极小碰集的数量及时间来中止算法. Cardoso 等人在 STACCATO 算法的基础上提出了 MHS^2 算法[10], 利用 Map-Reduce 环境进行并行计算, 虽然并行计算一般都高于串行计算, 然而在一定情况下, 串行计算可以转化为并行计算[11]. Pill 等人将 Boolean 算法的深度优先搜索替换为宽度优先搜索[12], 改变了算法扩展规则, 减少了空间消耗并提高了计算效率. 黄森等人利用 Boolean 算法与增量算法结合提出了 IBoolean 算法[13], 大幅度减少节点生成的数量. 以上优化算法大多期望减少节点的生成数量来提高求解效率, 近几年, 一些研究人员对解的极小化进行了大量研究, 整体效率提升明显[14]. 比如, 刘思光等人利用 Boolean 算法与独立覆盖思想结合[15], 判断每个碰集的真子集是否仍然满足碰集的定义, 该策略在判断每个解的极小性时, 不必考虑解集的规模, 极大提高了极小化时间. 王荣全等人在优化极小化策略的同时, 又将碰集求解问题转化成 SAT 问题[7], 把所有的冲突集合以子句形式表示成 SAT 的输入 CNF 进行迭代求解. Pill 等人通过对候选解中元素个数设置算法中止条件, 因求出的解集相对完备算法较少, 故极小化时相对耗时较少[11]. 王秋杰、金涛等人在针对配电网问题集合簇特有的拓朴结构特点[16], 对候选解的生成及极小化时设置了特定的目标函数, 使其计算出来的解都为极小解, 提高了求解效率. 赵相福等人针对问题集合簇独有的结构, 提出了一种针对树形结构的算法[17], 求出的解仅为极小解, 极大地提高了求解效率.

极小化运算作为极小碰集算法的最后一步, 其作用是保证算法最终求得问题解的极小性, 所选用方法效率的高低直接影响整体效率[18−19]. 目前常见的极小化算法分为两种, 一种是基于子超集检测极小化算法 (sub-superset detecting minimization, SSDM)[7], 该算法用新得到碰集与极小碰集簇中部分甚至全部解进行子集比较, 随着解集规

模的增加, 极小化时间往往占据整体求解的绝大部分. 另一种是独立覆盖检测去超集算法 (Independent Coverage Checking, ICC)[15], 其策略是判断碰集 (解) 中是否存在独立覆盖度为 0 的元素, 也就是说, 若碰集中去掉某个或某些元素后仍是碰集, 则为超集, 反之为极小碰集. ICC 算法因需要判断每一个解的真子集是否为碰集, 也牺牲了部分空间. 目前去超集算法, 未见考虑碰集算法自身对解集的影响, 只单纯寻找极小化条件, 且在极小化时, 都是考虑与全局的解集或者问题集合簇操作.

针对以上问题, 本文基于 Boolean 算法生成树规则, 提出一种结合局部问题集合簇去超集的策略. 所提算法对候选解进行极小化时, 只与部分集合取交集, 减少了对全局的搜索, 并且对已发现是超集的候选解进行删除, 在一定程度上起到了剪枝作用. 除此之外, 该算法无需额外的空间消耗, 最终结果仅且生成所有的极小碰集.

1 基础知识

1.1 基于模型诊断的相关概念

定义 1[1] 待诊断系统可定义为一个三元组 (D, P, O). 其中, D 为系统描述, 为一阶谓词公式的集合; P 为系统组成部件的集合, 是一个有限常量集; O 为观测集, 是一阶谓词公式的有限集.

如果系统输出的真实值和理论值不同, 我们称为是不一致的, 反之是一致的.

定义 2[1] 称部件集$\{c_1, c_2, \ldots, c_n\} \subseteq P$ 是冲突集, 当且仅当 $D \cup O \cup \{\neg A(c_1), \neg A(c_2), \ldots, \neg A(c_n)\}$是不一致的. 其中, c 正常时 $A(c)=0$, c 故障时 $A(c)=1$, $c \in P$. 若冲突集的任意真子集都不是冲突集, 称冲突集是极小冲突集.

定义 3[1] 设 $F=\{S_1, S_2, \ldots, S_n\}$为集合簇, 称 H 是 F 的碰集, 如果 H 满足以下两个条件:

(1)$H \subseteq \bigcup_{S_i \in F} S_i$;

(2) 对于任意一个 $S_i \in F$, 都有 $H \cap S_i \neq \emptyset$.

如果 H 的任意真子集都不是 F 的碰集, 那么碰集 H 为 F 的一个极小碰集. 其中, 去掉定义 3 中的约束 (2), 则称 H 为 F 的候选解.

定义 4[1] 给定一个 MBD 问题, 一个诊断被定义为一组组件Δ 的集合, 其中$\Delta \in P$, 当 $D \wedge O \wedge \{A(c)|c \in \Delta\} \wedge \{\neg A(c)|c \in P-\Delta\}$是一致的.

由定义 2-4 知, 给定待诊断系统 (D, P, O), $\Delta \subseteq P$ 为该系统的极小诊断, 当且仅当 Δ 为当前极小冲突集簇的极小碰集. 其中求解极小碰集的效率将直接影响极小诊断的效率. 因此, 研究高效的极小碰集求解方法对提升诊断效率具有重要的意义.

定义 5[15] 候选解 $H=\{e_1, e_2, \ldots, e_m\}$, 冲突集 $S_j \in F$, 若 $H \cap S_j=\{e_k\}$, 则称在 H 中, e_k 覆盖了集合 S_j. e_k 覆盖 F 中集合的个数称为 e_k 的覆盖度. 如果 H 中没有其他元素覆盖 e_k 所覆盖的集合, 则称 e_k 独立覆盖这些集合, 独立覆盖集合的个数称为 e_k 的独立覆盖度.

由定义 5 知, 若碰集中存在独立覆盖度为 0 的元素, 则该碰集为超集, 否则为极小碰集. 因为独立覆盖为 0 的元素没有单独覆盖任何集合, 去掉该元素也不影响碰集覆盖集合的个数, 故去掉该元素的碰集也是碰集[15]. 下面用例 1 对上面的定义进行详细描述.

例 1 给定问题集合簇 $F=\{\{1, 2, 3\}, \{3, 4, 5\}, \{5, 6, 7\}, \{1, 4, 7\}\}$.

集合{1, 5}, {1, 3, 6}, {1, 3, 7}及{3, 4, 6}都是碰集, 也是集合簇 F 的候选解. {1, 3, 6}中的元素 1、3、6 分别独立覆盖集合{**1**, 4, 7}, {**3**, 4, 5}, {5, **6**, 7}, 因元素 1、3、6 独立覆盖都大于 0, 故{1, 3, 6}为极小碰集. 对于集合{**1**, 2, **3**}, 因碰集{1, 3, 6}中有两个元素同时覆盖, 故没有任何元素独立覆盖该集合. 换句话说, {1, 3, 6}去掉任何一个元素都不构成碰集 (或者其子集{1, 3}, {1, 6}, {3, 6}都不是碰集), 故{1, 3, 6}为极小碰集. 而碰集{1, 3, 7}, 因存在子集{3, 7}为碰集, 故{1, 3, 7}为超集.

1.2 Boolean 算法简介

Boolean 方法将问题集合簇编码为 Boolean 表达式, 然后应用给定的规则对表达式进行计算得到碰集, 再应用 Boolean 中的吸收律得到所有极小碰集. 具体规则和 Boolean 表达式计算原理见文献 [6], 其算法实现可递归表

示为算法 1.

算法 1 Boolean(*F*, *allMHS*)

输入: 问题集合簇 $F=\{S_1, S_2, \ldots, S_n\}$, 候选解 *allMHS*;
中间变量: *LeftMHS* 和 *RightMHS* 分别存储左右分支两个冲突集的极小碰集.
输出: 极小碰集簇 *allMHS*.

```
Begin
1. IF(F ≠ ∅){
2.    IF (F=={S}) return {{e_i}|e_i ∈ S, S ∈ F};
3.    ELSE IF (∃e({e} ∈ F)){ //存在单元素集合
4.       Boolean({S_i|S_i ∈ F ∧ e ∉ S_i}, allMHS);
5.       allMHS ← {mhs ∪ {e}|mhs ∈ allMHS};
6.    }
7.    ELSE IF (∃e ∀ S_i(S_i ∈ F → e ∈ S_i)){ //所有集合都含有同一种元素
8.       Boolean({S_i−{e}|S_i ∈ F ∧ e ∈ S_i}, RightMHS); //右分支, 左分支为空
9.       allMHS ← {{e}} ⊎ RightMHS;
10.   }
11.   ELSE {
12.      e ← Select_element( ⋃_{S_i∈F} S_i ); //从中间问题集合簇中选取元素
13.      Boolean({S_i|S_i ∈ F ∧ e ∉ S_i}, LeftMHS); //左分支
14.      Boolean({S_i|S_i ∈ F ∧ e ∉ S_i} ∪ {S_i−{e}|S_i ∈ F ∧ e ∈ S_i}, RightMHS); //右分支
15.      allMHS ← {lmhs ∪ {e}|lmhs ∈ LeftMHS} ⊎ RightMHS;
16.   }
17. }
allMHS←Min(allMHS); //极小化处理
End
```

注: 第 9、15 行, 符号 ⊎ 代表两个解集直接合并. 在中间问题集合簇进行左右分支时, 右分支问题集合簇不含有元素 *e*, 在求出的候选解中不会含有元素 *e*. 而左分支解集需要归并元素 *e*, 所以左分支解集与右分支解集中的元素 (候选解) 不相同, 故用直接合并符号, 以降低操作的时间复杂度.

这里我们把解的生成放在算法递归后 (第 2、5、9、15 行), 也就是说, 解的生成是自叶子节点到根节点扩展的过程, 与文献 [15] 中的自根节点到叶子节点生成解在时间及空间复杂度上一致. Boolean 算法在求解的过程中可能会产生超集, 最终求解后需要去超集检测 (算法 1 结束后的阴影部分).

为了描述方便, 在 Boolean 算法运行时, 我们给出如下定义.

定义 5 Boolean 算法生成树中, 称没有子节点的为**叶子节点**, 其中含有 0 个元素的叶子节点为空叶子节点, 含有 $m(m \geqslant 1)$ 个元素的叶子节点为叶子集合节点.

定义 6 在 Boolean 算法求解过程中, 当前所关注的问题集合簇称为**中间问题集合簇**; 称去掉含有元素 *e* 集合的集合簇为**左分支集合簇** (第 13 行); 称只去掉元素 *e* 的集合簇为**右分支集合簇** (第 14 行); 仅在右分支而不在左分支的集合构成的簇称为**可疑集合簇**.

下面通过实例 2 进一步说明 Boolean 算法深度优先生成一棵树的过程以及对上面定义的一些具体描述.

例 2 计算问题集合簇 $F=\{\{1, 2, 3\}, \{3, 4, 5\}, \{5, 6, 7\}, \{1, 7\}\}$的所有极小碰集.

为了清楚分析动态问题集合簇及候选解的生成, 把选取的元素放在路径上, 中间问题集合簇放到了节点上, 在

候选解生成时, 只有路径上的元素和叶子节点为候选解元素, 其余节点不是候选解部分. 用 F_1、F_2 表示由 F 生成的左右节点 (根节点用 F 表示), 以“1”结尾表示左节点, “2”结尾表示右节点, 如: F_{11}、F_{12} 是 F_1 的左、右节点. 可疑集合簇是动态变化的, 对于 F_1 与 F_2 来说, F_2 比 F_1 多的集合簇为可疑集合簇, 由图 2 可以看出可疑集合簇 $F_{2\text{-}1}$={{2, 3}, {7}}.

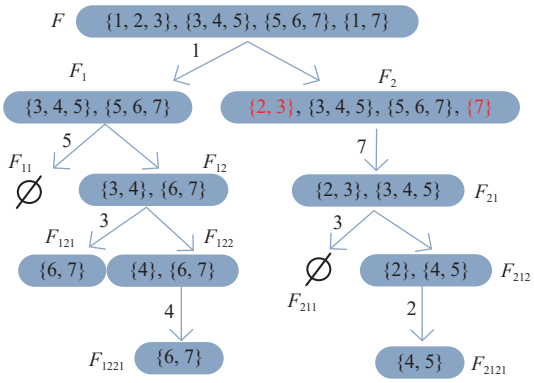


图 2 Boolean 算法计算极小碰集

根节点 F, 首先选取重复度最高 (若有多个则按字典排序) 的元素 1, 生成左分支 F_1={{3, 4, 5}, {5, 6, 7}}, 右分支 F_2={{2, 3}, {3, 4, 5}, {5, 6, 7}, {7}}. 对于 F_1, 因为元素 5 覆盖了 F_1 的所有集合 (7 行), 则左分支 F_{11} 为空, 右分支 F_{12}={{3, 4}, {6, 7}}. 对于 F_{12} 选取元素 3, 则左分支 F_{121}={{6, 7}}, 右分支 F_{122}={{4}, {6, 7}}. 因为左分支 F_{121} 只有一个集合不再进行分支 (2 行), 右分支 F_{122} 存在单元素集合的情况 (3~6 行), 没有右分支只有左分支, 得到 F_{1221}={{6, 7}}, 不再递归分支且在算法第 2 行结束. 同理, 右分支 F_2 和 F_1 处理方式一样, 这里不再赘述.

按照上述描述, 布尔算法根据深度优先的规则生成了一棵树, 接下来从叶子节点求解这棵树. 从空叶子节点 F_{11} 扩展元素 5, {{5}}为 F_1 的左分支候选解; F_{121} 含有两个元素, 候选解为{{6}, {7}}, 扩展元素 3, {{3, 6}, {3, 7}}为 F_{12} 的左分支候选解; F_{1221} 含有两个元素, 候选解为{{6}, {7}}, 扩展元素 4, {4, 6}, {4, 7}为 F_{122} 的候选解. 因为 F_{122} 的候选解是 F_{12} 的右分支候选解, F_{12} 的候选解是 F_1 的右分支候选解, 所以左右两部分解集合并即是 F_1 的所有候选解. F_1 的所有候选解再扩展元素 1, 就得到了 F 的左分支候选解, 同理可以得到 F_2 的右分支候选解.

下面对 Boolean 算法及两种极小化策略的时间复杂度做简要分析:

设冲突集有 n 个集合, 每个集合有 m 个元素, 每次选取元素的重复度为 k. 令 t=(n/k), 可近似看作候选解中元素的个数. 算法每次递归时, 左分支减少 k 个集合, 右分支由上层中间问题集合簇继承并减少一种元素. 计算碰集的时间复杂度为 $O(\sum_{i=0}^{t-1} m^{i+1}(n-ik))=O(k\frac{m^2(m^t-1)}{m-1})\approx O(km^{t+1})$.

共求出 m^t 个碰集解, SSDM 算法需要比较所有解的包含关系, 因每个碰集含有 t 个元素, 取包含关系时间复杂度为 $O(t)$, 去超集的时间复杂度为 $O(tm^{2t})$.

ICC 算法需要判断每个候选解的真子集是否为碰集, 其中每个候选解需要判断 t 个子集, 每个子集判断为碰集的时间复杂度为 $O(nt)$, 去超集的时间复杂度为 $O(nt^2m^t)$.

由此可见, 无论取以上哪种极小化策略, 去超集时间都占据整体求解时间的较大比重.

2 结合可疑集合簇去超集的 Boolean 算法

由例 2 可知, 布尔算法在生成解集时, 每次先得到节点 F_1, F_2 的碰集, F_1 再扩展上层元素, 合并这两部分解就能够得到节点 F 的所有碰集. 如果我们能够保证每次所求局部问题集合簇 F_1, F_2 的碰集都是极小的, 且 F_1 扩展上层元素后得到的解也是极小的, 通过递归的策略, 能够得到且仅得到 F 所有的极小碰集.

接下来, 我们深入分析 Boolean 算法生成树及扩展候选解的策略, 给出一种简洁高效的去超集策略.

碰集 $HS=\{e_1, e_2, \ldots, e_m\}$, HS 的中间候选解 $H=\{e_1, e_2, \ldots, e_k\}$, $H \subseteq HS$, $1 \leqslant k<m$. 当候选解 H 扩展元素 e_{k+1} 时, 若成为超集, 显然, 要么 $e_i(1 \leqslant i \leqslant k)$ 独立覆盖度为 0, 要么 e_{k+1} 的独立覆盖度为 0.

命题 1 当候选解 H 扩展元素 e_{k+1} 时, 元素 e_{k+1} 的独立覆盖度最多为对应可疑问题集合簇中集合的个数.

证明: 因为 Boolean 算法是深度优先生成树, 元素 e_{k+1} 之前覆盖的集合已经被上面路径的元素 $e_j(j=k+2, k+3, \ldots, m)$ 覆盖, 不再独立覆盖这些集合. 显然, 当扩展到元素 e_{k+1} 时, 元素独立覆盖度最多是对应可疑问题集合簇中集合的个数. 证毕

命题 2 若扩展完元素 e_{k+1} 使极小候选解成为超集, 那候选解 H 一定与可疑集合簇中的每个集合交集非空.

证明: 如果中间问题集合簇得到的解 H 为极小候选解, 显然候选解中的元素独立覆盖度都不为 0. 因为布尔算法在选取元素 e_{k+1} 后生成的左分支不会含有元素 e_{k+1} 的集合, 所以元素 e_{k+1} 不会覆盖左分支的任何集合, 也不会影响候选解 H 中元素的独立覆盖度. 当候选解扩展元素 e_{k+1} 时, 只会影响元素 e_{k+1} 的独立覆盖度, 而元素 e_{k+1} 独立覆盖的集合由命题 1 知, 只独立覆盖可疑集合簇, 因此从叶子节点扩展元素 e_{k+1} 且成为超集必然会使元素 e_{k+1} 的独立覆盖度变为 0, 即极小候选解覆盖了可疑集合簇全部集合. 证毕

基于命题 1、2, 我们可以得到如下结论:

当候选解中的元素只有叶子节点元素时, 候选解定为中间问题集合簇的极小候选解 (此时候选解只有一个元素). 从这一点作为算法递归的开始, 候选解每一次向上扩展元素时, 判断候选解与可疑集合簇是否存在交集为空的集合, 若存在, 则候选解扩展上层元素也为极小候选解; 反之, 该候选解为超集需要删除. 按照上面的方式递归向根节点扩展元素, 直到算法结束, 那么求出的解也全为极小碰集, 并且能够求出全部的解. 其中算法的正确性及完备性由 Boolean 算法和独立覆盖策略保证.

在 Boolean 算法计算时, 我们也得到了额外的去超集优化, 具体见命题 3.

命题 3 不含可疑集合簇的问题集合簇向根节点归并元素时无需去超集判断.

证明: 在 Boolean 算法中, 对于不含有可疑集合簇的分支, 候选解在选取元素时有三种情况. 第一, 叶子节点为候选解中第一个元素, 该元素独立覆盖叶子节点集合, 独立覆盖度至少为 1; 第二, 中间问题集合簇中所有集合都含有同一种元素, 当选取该元素时不会产生对应的左分支或者左分支为空, 没有子孙节点向该元素归并解, 该元素即是候选解的第一个元素, 独立覆盖度至少为 1; 第三, 集合中只有一个元素, 在元素的选取时显然独立覆盖本身集合, 独立覆盖度也至少为 1; 对于命题 3 的三种情况分别对应算法 1 的前三种情况, 也就是说, 只需要对 15 行元素的扩展进行极小性判定. 证毕

设 $F_l \subset F$, 可疑集合簇 $F_r=F-F_l$, x 是 F_l 的极小碰集, 元素 e 是 x 要扩展的元素. 我们得到定理 1.

定理 1 如果 x 与 F_r 中的集合都有交集, 该候选解 x 为超集.

证明: 因为 $F_l \subset F$, 若 x 与 F_r 中的集合都有交集, 则 x 也是 F 的极小碰集, 显然, x 再添加元素 e, 则成为 F 超集. 接着证明 x 与 F_r 中的集合至少有一个没有交集, 则 $x \cup \{e\}$ 是 F 的极小碰集. 反正法, 若 $x' \leftarrow x \cup \{e\}$ 是 F 的超集, 则必有 x' 的真子集是 F 的极小碰集. 显然, 元素 e 与 F_l 中的任一集合都没有交集, 且 x' 去掉任一元素 (除了元素 e) 都不是 F_l 和 F 的碰集, 故结论与假设矛盾. 证毕.

根据定理 1, 结合 Boolean 算法与可疑集合簇我们提出 BWSS 算法 (Boolean With Suspicious Sets), 仅需修改 Boolean 算法的第 15 行为如下两行内容, 即为 BWSS 算法的伪代码:

1. $LeftMHS \leftarrow \{lmhs \cup \{e\} \mid \exists S_i \in F \wedge e \in S_i, s.t., lmhs \cap (S_i - \{e\}) == \varnothing, lmhs \in LeftMHS\}$;

2. $allMHS \leftarrow LeftMHS \uplus RightMHS$;

解释; 若左分支解集存在与可疑集合簇不相交的集合, 则扩展元素 e 后并把该左分支解加入极小候选解中, 否则删除该解. 接下来使用例 3 及图 3 进一步说明 BWSS 算法计算极小碰集.

例 3 计算冲突集簇 $F=\{\{1, 2, 3\}, \{3, 4, 5\}, \{5, 6, 7\}, \{1, 4, 7\}\}$的所有极小碰集.

图 3 中字体标红的集合为其对应左分支的可疑集合簇, 其生成解的规则和图 1 类似, 只不过增加了对候选解极小性的判断. 从左边黑色三角号标记的叶子集合节点{6, 7}开始, 当叶子集合节点初始归并到候选解时为{{6}, {7}}, 发现对应的可疑集合簇为{{4}}, 候选解{{6}, {7}}与可疑集合簇交集为空, 则为极小侯选解, 可以归并元素

3. 对于中间集合簇{{3, 4, 5}, {5, 6, 7}}在求解的过程中只有一个可疑集合{4}, 我们已经计算完毕, 其余节点按照原始 Boolean 算法计算即可, 得到的所有极小候选解为{5}, {3, 6}, {3, 7}, {4, 6}, {4, 7}, 对于这些解再归并元素 1 时, 因为对应的可疑集合簇不为空, 需要对解的极小性进行判断. 其中只有极小候选解{3, 7}与可疑集合簇{{2, 3}, {4, 7}}都有非空交集, 因而需要去掉. 其余解归并元素 1, 左分支共得到 4 个极小碰集解{1, 5}, {1, 3, 6}, {1, 4, 6}, {1, 4, 7}. 同理, 右分支没有任何极小候选解全部覆盖对应的可疑集合簇, 最终得到 7 个极小碰集解{3, 7}, {3, 4, 5}, {3, 4, 6}, {2, 4, 5}, {2, 5, 7}, {2, 4, 6}, {2, 4, 7}, 加上左分支的极小碰集解共得到 11 个极小碰集解.

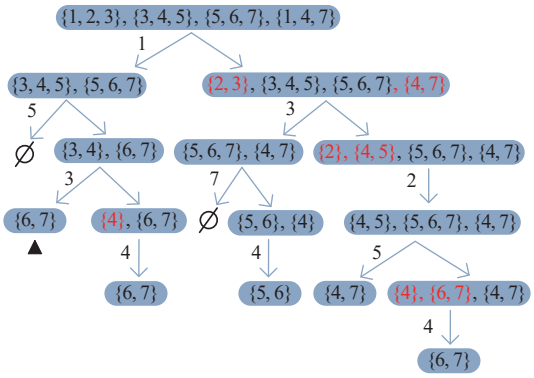


图 3　使用 BWSS 算法对可疑集合簇标记

该算法充分结合了独立覆盖思想及 Boolean 算法计算碰集的规则, 提出了一种更简便高效的计算方法. 由命题 3 我们可以知道, 当 Boolean 算法递归进入到左右分支时我们才考虑候选解的极小性判断, 因为其他情况不含有可疑集合簇, 所以无需极小化判断, 这也为算法去超集减少了大量判断.

时间复杂度: 设 n, m, k, t 变量同 Boolean 算法时间复杂度分析相同. BWSS 算法与 Boolean 算法的不同点在于去超集上, BWSS 算法对每一候选解在极小化过程中发现与可疑集合簇没有交集的集合时直接结束判断, 去超集时取决于候选解与集合取交集的次数.

含有 t 个元素的集合取交集的时间为 $O(t)$, 最坏情况下, 候选解每次扩展元素时都需要与可疑集合簇的所有集合取交集. 从叶子节点扩展元素, 候选解对集合取交集的次数为 $1*k+2*k+\ldots+t*k=\frac{kt(1+t)}{2}\approx\frac{n^2}{2k}$, 所以总体去超集的时间复杂度 $O(m^t\frac{n^2}{2k})$. 布尔算法的时间复杂度为 $O(km^{t+1})$, 若 $\frac{n^2}{2k}<km$, 即 $n^2<2mk^2$, 则极小化时间将低于求解时间. 除此之外, 叶子节点等三种情况的元素不必去超集判断, 因而总体去超集时间远小于 $O(m^t\frac{n^2}{2k})$.

3　实验分析

在本节中我们对 BWSS 算法进行实验分析 (https://github.com/helong0102/hittingset-BWSS), 选取目前效率较快的 MHS-DMECV、BWIICC、Boolean、STACCATO 四种算法作为比较对象. 平台如下: Windows 10 操作系统, CPU Intel Core i7-1065G7 1.30GHz 16GB 4 核 RAM C++, 实验测试用例分为人工随机数据与 ISCAS-85 基准电路数据①.

3.1　随机数据

本节采用文献 [7,14,15] 的伪随机数据, 共有三个输入参数: 问题集合簇内集合的个数 n, 集合簇内元素的最大种类数 m、每一位元素出现的概率 p(在同一组测试数据中, 所有元素的 p 均相等). 图 3 中, 参数设计, n 取值为 200, m 取值为 15、20、25、30, p 取值为 0.15~0.85(间隔 0.05), 每一组数据随机三次, 取其平均执行时间作为实验

① 基准电路冲突集压缩包获取地址: http://www.fe.up.pt/~rma/benchmarks2.zip

结果.

在图 4 中, 各子图中的横坐标表示元素出现的概率 p, 右侧纵坐标表示极小碰集数量. 从图中可以看出, 对于解集数量小于 100 的用例, 各算法运行时间相近. 但随着求解数量增多, BWSS 算法优势逐渐明显. 其中 Boolean、SACCTAO 算法选择的极小化策略是 SSDM 算法, 随着解集增多, 极小化占据了整体求解时间的主导, 故二者算法拟合度较高. 当解集在 3, 000 时, 运行时间花费约是 BWSS 算法的 20 倍, 解集在 35, 000 时, 运行时间花费约是 BWSS 算法的 200 倍, 解集在 300, 000 时, 由于运行时间设置, 二者花费时间已经超过 10, 000 秒, 而 BWSS 算法也可以在 10 秒内得出结果. 可见传统的子集检测极小化策略, 在解决高难度问题时 (解集数量大于 100, 000), 效率远低于 BWSS 算法.

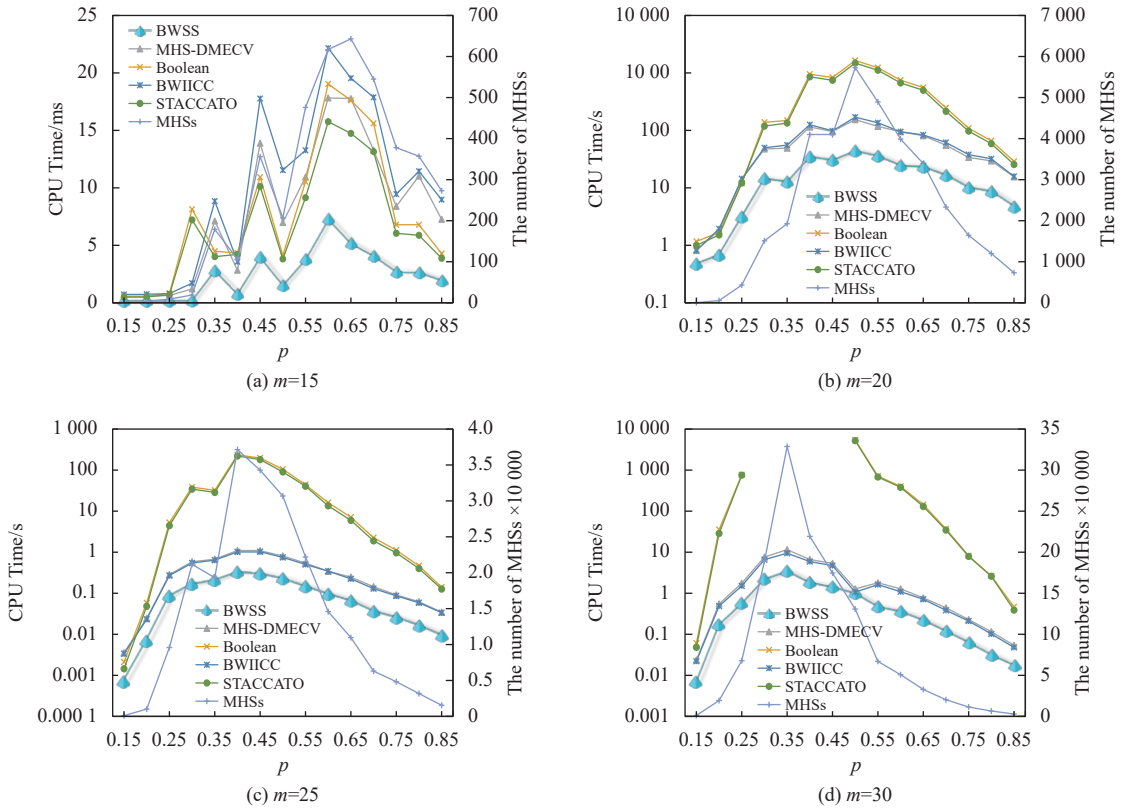
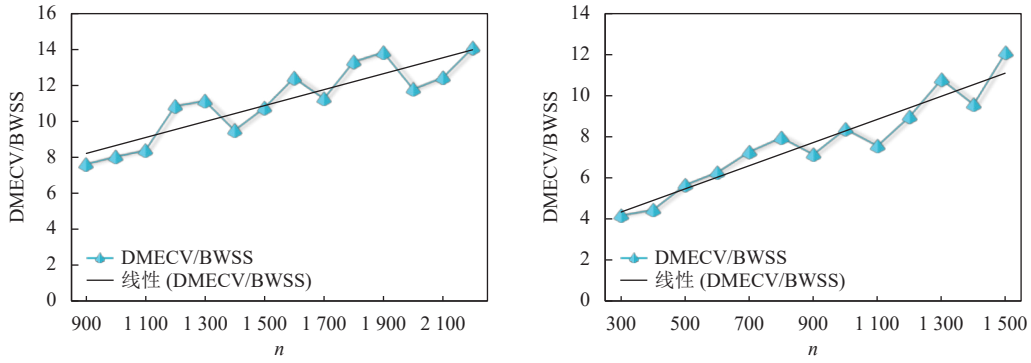


图 4 BWSS 算法与其他几种算法运行时间比较

MHS-DMECV、BWIICC 算法在极小化时, 采用的是独立覆盖思想, 即判断候选解中是否存在独立覆盖度为 0 的元素. 每个候选解极小化时只与全局问题集合簇集合的个数与子集的个数相关, 与解集个数无关, 由图 4(a~d)知, 当全局问题集合簇个数相等时, 时间花费整体都是 BWSS 算法的 5 倍左右, 并不受解集规模的影响.

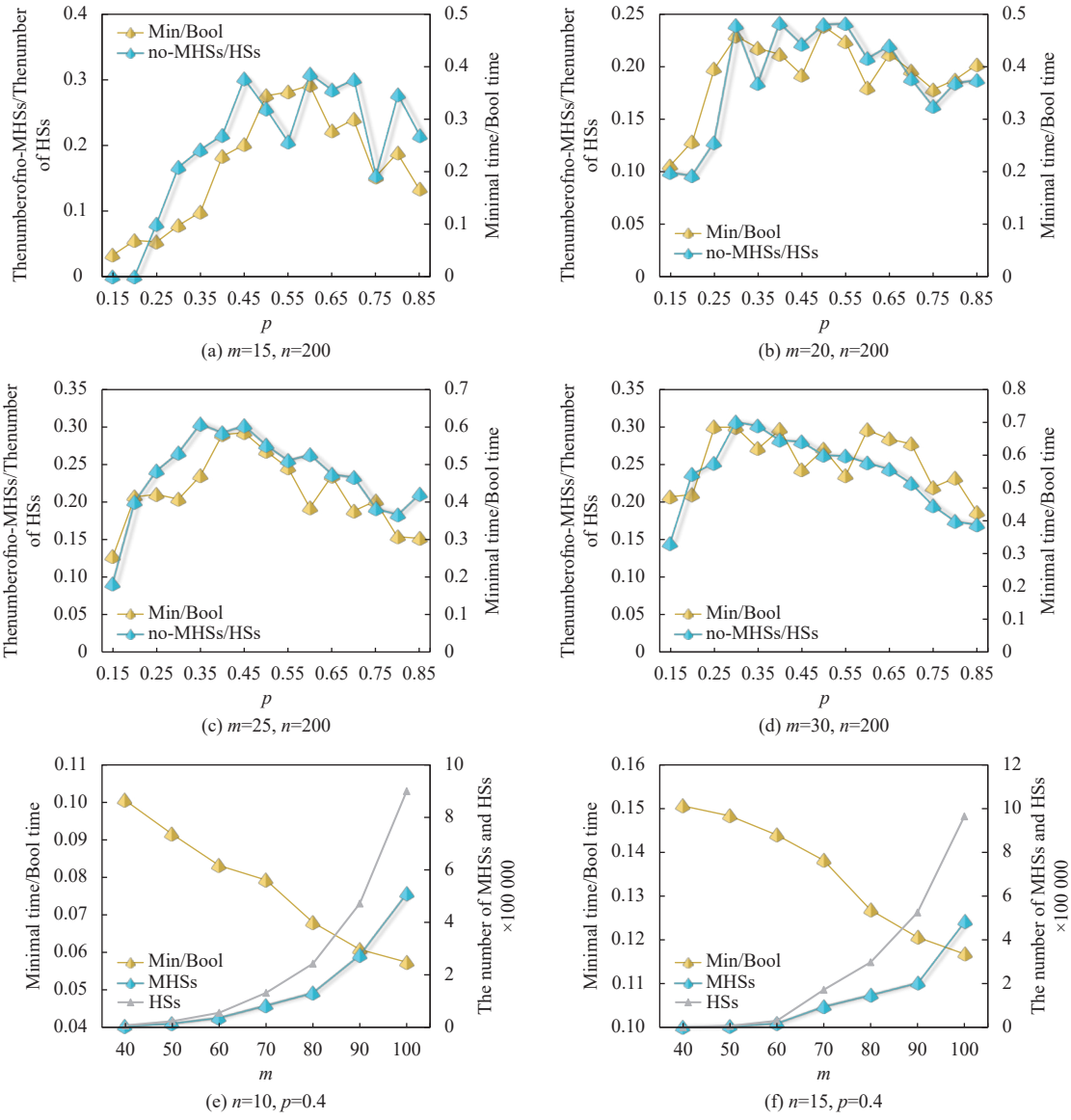
图 5 进一步描述可疑集合簇去超集算法和独立覆盖去超集算法 (选用 MHS-DMECV 算法与 BWSS 算法进行比较), 纵坐标表示 MHS-DMECV 算法与 BWSS 算法的运行时间之比, 横坐标表示问题集合簇内集合的个数. 随着问题集合簇内集合个数 n 的不断增大, MHS-DMECV 与 BWSS 的运行时间的比值逐渐增大, 即 BWSS 算法的相对运行效率越高. 当集合的个数是 600、1 000、1 400、1 800、2 200 时, MHS-DMECV 运行时间花费约是 BWSS 算法的 6、8、10、12、14 倍, 可见随着问题集合簇内集合个数增大, 比值成一定的线性增长.

图 6 中 Bool、Min 分别表示 BWSS 算法计算碰集与极小化的时间, 即 BWSS=Bool+Min. no-MHSs、HSs 表示非极小碰集及总碰集的个数, 即 HSs=MHSs+no-MHSs.Min/Bool 表示 BWSS 算法极小化占计算碰集的比值, no-MHSs/HSs 表示非极小碰集占总碰集的比值, n 为集合的个数, m 为每个集合内元素的最大种类数.



(a) m=35, p=0.20 (b) m=30, p=0.25

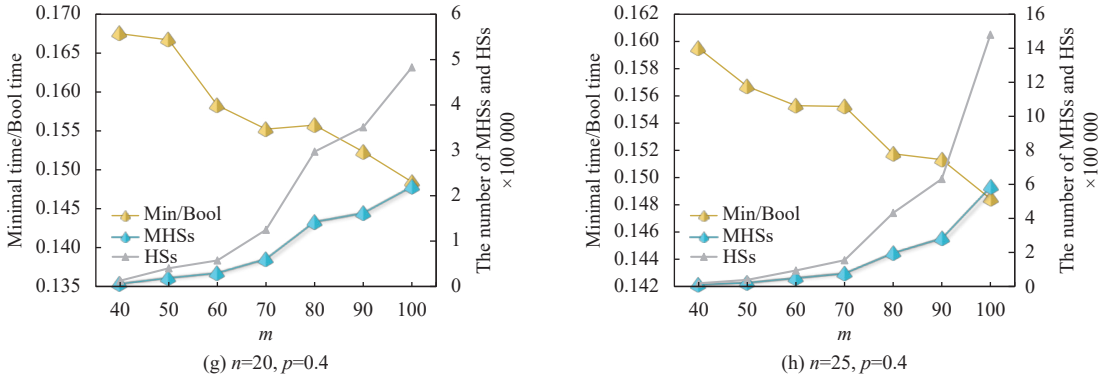
图 5 BWSS、MHS-DMECV 算法运行时间之比



(a) m=15, n=200 (b) m=20, n=200

(c) m=25, n=200 (d) m=30, n=200

(e) n=10, p=0.4 (f) n=15, p=0.4



(g) n=20, p=0.4　　(h) n=25, p=0.4

图 6　BWSS 算法解集极小化时间与计算碰集时间之比

由图 6 可知, BWSS 算法的极小化时间占比大多低于计算碰集时间的 30%, 求解时间占据了主导地位. 其中子图 *a*~*d* 设置了单一变量 p, 当 p 取值低于 0.25(高于 0.7) 时, 由于每个集合内生成的数据较稀疏 (较密集), 在 BWSS 算法求解时, 中间问题集合簇生成单元素集合的概率较高 (中间问题集合簇内的所有集合含有同一种元素的概率较高), 由于命题 3, 候选解内较多元素不需要极小化, 故极小化总体占比较小. 当 p 取值在 0.3~0.7 之间时, 由于集合 n 的个数不变, 极小化时间都介于求解时间的 20%~30%. 其中, no-MHSs/HSs 与 Min/Bool 拟合度较高, 当 no-MHSs/HSs 比值越高, 说明算法求出的冗余解相对较多, 故极小化时间相对较多. 换句话说, BWSS 算法的极小化策略能够很好的对极小解及冗余解进行识别, 对于一些极小候选解, 大量元素不需要极小化操作, 而对于需要极小化的元素, 在与可疑集合簇取交集时, 发现存在交集为空的集合, 立即中止极小化判断. 除此之外, 当局部问题集合簇发现局部超集及时删除, 也起到了一定的剪枝作用.

由 BWSS 算法的时间复杂度分析可知, 极小化时间与计算碰集的时间之比为 $\frac{n^2}{2mk^2}$, 可见, 当比值越小, 极小化占比越小. 子图 *a*~*d* 中, 我们都设置了 n=200, m 为 15~30, 从理论上来看很不利于我们算法的极小化策略, 但是当 n 越大, 元素的重复度 k 又相对较大, 故整体的实验效果并不会太差 (由图 6 也可看出). 在子图 *e*~*h* 中, n, p 为定值, 单一变量为 m, 随着 m 取值增大 (20~100, 间隔 10), 极小化时间占总体求解时间越来越少. 然而, 由于随机数据的原因, 每个候选解内元素重复度 k 并不是一个定值, 故下降趋势有一定的波动.

3.2　ISCAS-85 基准电路冲突集

在 1985 年电路与系统的国际研讨会上, 十个组合电路被确定为 ISCAS-85 基准电路 (包含 c880, c2670, c5315, c7552 等), 主要用于比较测试生成领域的结果. DXC'09 的研究人员考虑了 1400 多个真实环境, 在这些电路中设置了一些故障, 生成了相应的问题集合簇[20]. 表 1 中, 列名 MCS-Family 表示问题集合簇类别, 如 c880:CS020 表示电路 c880 第 20 号的故障场景. BWSS、BWIICC、Boolean 表示三种算法的运行时间 (s), $\frac{BWIICC-BWSS}{BWIICC}$、$\frac{Boolean-BWSS}{Boolean}$ 分别表示 BWSS 算法比 BWIICC、Boolean 算法节省的时间效率, Min/Bool 表示 BWSS 算法的极小化时间与计算碰集时间之比, m, n 表示问题集合簇内元素的种类数及集合的个数, "—"符号表示 BWIICC 算法由于内存不足而无法给出运行结果, MHSs 表示极小碰集的数量.

由表 1 可以看出, BWSS 算法比 BWIICC 算法平均节省时间在一个数量级, 比 Boolean 算法平均节省时间在三个数量级以上. 对于问题集合簇 c880, 基本元件的个数 (基准电路中的门, 对应图 6 中的 m) 上限是 400, 由于机器配置 BWIICC 算法因内存不足, 只可以求出 200, 000 左右的解集, 而 BWSS 算法仍可以在 1 秒内求出 500, 000 以上的解集. 随着问题规模的增大, c2670, c5315, c7552 问题簇中元件的个数上限达到 1, 200, 2, 400, 3, 600, 不仅对算法在运行时间上是一个巨大挑战, 在内存空间方面也有较高的要求.

不仅如此, 我们还统计得出在表 1 的四种基准电路, 共一千多个问题集合簇中, 所有的 m 都大于 n, 且有超过 99% 以上都是 m 远大于 n(由 m, n 列也可以明显看出), 这对 BWSS 算法来说无疑是有利的, 其极小化时间仅占求

解碰集时间的 2% 左右, 比图 6 产生的随机数据比值要低. 可见, 在真实环境的电路中, 新提出来的 BWSS 算法比其他算法更加有效.

表 1 BWSS、BWIICC、Boolean 算法在基准电路数据运行时间比较

MCS-Family	BWSS	BWIICC	$\frac{BWIICC-BWSS}{BWIICC}$	Boolean	$\frac{Boolean-BWSS}{Boolean}$	Min/Bool	m, n	MHSs
c880:CS020	0.240 149	—	—	4 168.565 116	99.99%	2.11%	382, 10	189 826
c880:CS033	0.366 817	2.484 514	85.24%	2 236.916 721	99.98%	2.55%	382, 10	137 670
c880:CS034	0.673 964	—	—	>10 000	—	3.78%	382, 11	569 478
c880:CS061	0.122 208	1.279 238	90.45%	385.183 549	99.97%	7.23%	368, 11	60 912
c880:CS065	0.060 907	0.317 447	80.81%	7.349 233	99.17%	1.61%	376, 10	10 368
c880:CS071	0.300 336	1.841 463	83.69%	1 345.545 487	99.98%	3.14%	382, 12	101 952
c880:CS089	1.131 833	—	—	>10 000	—	2.14%	382, 13	311 040
c2670:CS005	0.028 947	0.769 768	96.24%	237.833 395	99.99%	1.55%	1 188, 16	21 229
c2670:CS015	0.065 955	1.010 154	93.47%	517.859 494	99.99%	1.85%	1 188, 19	42 458
c2670:CS021	0.058 306	1.010 154	94.23%	83.771 619	99.93%	1.24%	1 151, 13	17 671
c2670:CS045	0.120 151	1.281 423	90.62%	217.226 527	99.94%	0.67%	1 151, 16	26 768
c2670:CS049	0.352 656	—	—	1 675.557 412	99.98%	2.51%	1 188, 18	86 325
c2670:CS050	0.984 415	—	—	9 794.000 254	99.99%	3.19%	1 180, 11	231 000
c2670:CS063	0.764 386	—	—	2 264.540 501	99.97%	2.16%	1 151, 13	108 780
c2670:CS091	0.585 599	—	—	9 982.785 942	99.99%	2.57%	1 191, 16	235 102
c2670:CS095	0.457 656	—	—	1 596.367 891	99.97%	1.93%	1 095, 15	78 720
c2670:CS100	1.059 034	—	—	7 106.983 614	99.99%	2.26%	1 095, 15	157 440
c5315:CS013	0.321 187	—	—	6 539.158 936	99.99%	3.44%	2 073, 7	134 360
c5315:CS018	0.068 371	1.068 631	93.60%	90.578 441	99.92%	2.78%	2 067, 6	13 025
c5315:CS019	0.048 431	0.631 721	92.33%	30.691 244	99.84%	3.38%	2 070, 7	8 215
c5315:CS021	0.687 751	—	—	9 125.169 798	99.99%	1.89%	2 305, 9	171 264
c5315:CS024	1.789 075	—	—	6 897.649 878	99.97%	2.03%	2 277, 7	137 664
c5315:CS040	0.074 091	0.617 938	88.01%	12.250 784	99.40%	1.18%	2 101, 10	5 670
c5315:CS071	0.338 365	—	—	528.165 406	99.94%	1.48%	2 065, 11	38 610
c5315:CS090	0.674 368	—	—	1 740.440 622	99.96%	1.87%	2 065, 13	77 220
c7552:CS000	0.096 244	0.808 454	88.10%	17.174 705	99.44%	2.78%	3 503, 29	5 143
c7552:CS004	0.071 693	1.241 514	94.23%	222.555 819	99.97%	1.58%	3 503, 13	17 272
c7552:CS013	0.120 988	1.415 478	91.45%	45.231 901	99.73%	1.06%	3 503, 31	10 286

4 总 结

Boolean 算法是计算极小碰集经典的算法, 因该算法可以和大部分极小碰集算法相互转化, 故对该算法提出的优化策略能够具有一定的通用性. 极小碰集的极小化一直是阻碍算法提升的关键, 其中, 基于独立覆盖度思想的去超集算法, 剥离了碰集极小性判定效率与极小碰集簇大小的相关性, 转化为碰集与原始问题集合簇取交集操作, BWIICC、MHS-DMECV 算法在解决高难度问题时比结合 SSDM 去超集的 Boolean 算法节省了 3 个数量级, 是国内外较快的算法.

本文深入分析 Boolean 算法使候选解成为超集的集合, 提出了可疑集合簇概念及 BWSS 算法, 在候选解极小性判断时, 只需与可疑集合簇进行取交集, 若存在交集为空的集合, 则为极小候选解, 否则删除该解. 由于 BWSS 算法的叶子节点等三种情况的元素无需极小化判断, 求解碰集时间占据算法的主导, 整体时间效率上比 BWIICC、MHS-DMECV 算法节省一个数量级左右. 在解集数量大于 10 万的测试用例中, BWSS 算法仍然可以保持较高的求解效率, 此外, BWSS 算法没有额外的空间消耗, 为求解大规模问题提供了条件.

References:

[1] Reiter R. A theory of diagnosis from first principles. Artificial Intelligence, 1987, 32(1): 57–95. [doi: 10.1016/0004-3702(87)90062-2]

[2] Zhao XF, Ouyang DT. Deriving all minimal conflict sets using satisfiability algorithms. Acta Electronica Sinica, 2009, 37(4): 804–810 (in Chinese with English abstract). [doi: 10.3321/j.issn:0372-2112.2009.04.024]

[3] Metodi A, Stern R, Kalech M, Codish M. A novel SAT-based approach to model based diagnosis. Journal of Artificial Intelligence Research, 2014, 51: 377–411. [doi: 10.1613/jair.4503]

[4] Greiner R, Smith BA, Wilkerson RW. A correction to the algorithm in Reiter's theory of diagnosis. Artificial Intelligence, 1989, 41(1): 79–88. [doi: 10.1016/0004-3702(89)90079-9]

[5] Wotawa F. A variant of Reiter's hitting-set algorithm. Information Processing Letters, 2001, 79(1): 45–51. [doi: 10.1016/s0020-0190(00)00166-6]

[6] Jiang YF, Lin L. The computation of hitting sets with Boolean formulas. Chinese Journal of Computers, 2003, 26(8): 919–924 (in Chinese with English abstract). [doi: 10.3321/j.issn:0254-4164.2003.08.004]

[7] Wang RQ, Ouyang DT, Wang YY, Liu SG, Zhang LM. Solving minimal hitting sets method with SAT based on DOEC minimization. Journal of Computer Research and Development, 2018, 55(6): 1273–1281 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.2018.20160809]

[8] Gainer-Dewar A, Vera-Licona P. The minimal hitting set generation problem: Algorithms and computation. SIAM Journal on Discrete Mathematics, 2017, 31(1): 63–100. [doi: 10.1137/15M1055024]

[9] Abreu R, van Gemund A. A low-cost approximate minimal hitting set algorithm and its application to model-based diagnosis. In: Proc. of the 8th Symp. on Abstraction Reformulation and Approximation. Lake Arrowhead: AAAI, 2009. 2–9.

[10] Cardoso N, Abreu R. MHS2: A map-reduce heuristic-driven minimal hitting set search algorithm. In: Int'l Conf. on Multicore Software Engineering, Performance, and Tools. Saint Petersburg: Springer, 2013. 25–36. [doi: 10.1007/978-3-642-39955-8_3]

[11] Jannach D, Schmitz T, Shchekotykhin K. Parallelized hitting set computation for model-based diagnosis. In: Proc. of the 29th AAAI Conf. on Artificial Intelligence. Austin: AAAI, 2015. 1503–1510. [doi: 10.1609/aaai.v29i1.9389]

[12] Pill I, Quaritsch T. Optimizations for the Boolean approach to computing minimal hitting sets. In: Proc. of the 20th European Conf. on Artificial Intelligence. Montpellier: IOS Press, 2012. 648–653. [doi: 10.3233/978-1-61499-098-7-648]

[13] Huang S, Zhao XF, Tong XR. An incremental Boolean algorithm for computing minimal hitting sets. In: Proc. of the 13th Int'l Congress on Image and Signal Processing, BioMedical Engineering and Informatics. Chengdu: IEEE, 2020. 56–59. [doi: 10.1109/CISP-BMEI51763.2020.9263682]

[14] Deng ZY, Ouyang DT, Geng XN, Liu J. Computing the minimal hitting sets with dynamic maximum element coverage value. Journal of Computer Research and Development, 2018, 55(4): 791–801 (in Chinese with English abstract). [doi: 10.7544/ISSN1000-1239.2018.20160900]

[15] Liu SG, Ouyang DT, Zhang LM. Method of minimality-checking of candidate solution for minimal hitting set algorithm. Journal of Software, 2018, 29(12): 3733–3746 (in Chinese with English abstract). [doi: 10.13328/j.cnki.jos.005311]

[16] Wang QJ, Jin T, Wang MQ. A hierarchical minimum hitting set calculation method for multiple multiphase faults in power distribution networks. IEEE Transactions on Industrial Electronics, 2021, 68(1): 4–14. [doi: 10.1109/tie.2020.2967691]

[17] Zhao XF, Tong XR, Ouyang DT, Zhang LM, Hou YZ. TreeMerge: Efficient generation of minimal hitting-sets for conflict sets in tree structure for model-based fault diagnosis. IEEE Transactions on Reliability, 2021, 70(4): 1596–1610. [doi: 10.1109/TR.2021.3115130]

[18] Chen XM, Meng XF, Qiao RX. Method of computing all minimal hitting set based on BNB-HSSE. Chinese Journal of Scientific Instrument, 2010, 31(1): 61–67 (in Chinese with English abstract). [doi: 10.19650/j.cnki.cjsi.2010.01.011]

[19] Huang J, Chen L, Zou P. A compounded genetic and simulated annealing algorithm for computing minimal diagnosis. Journal of Software, 2004, 15(9): 1345–1350 (in Chinese with English abstract).

[20] Kurtoglu T, Narasimhan S, Poll S, Garcia D, Kuhn L, Dekleer J, van Gemund A, Feldman A. First international diagnosis competition–DXC'09. In: Proc. of the 20th Int'l Workshop on Principles of Diagnosis, 2009. 383–396.

附中文参考文献:

[2] 赵相福, 欧阳丹彤. 使用 SAT 求解器产生所有极小冲突部件集. 电子学报, 2009, 37(4): 804–810. [doi: 10.3321/j.issn:0372-2112.2009.04.024]

[6] 姜云飞, 林笠. 用布尔代数方法计算最小碰集. 计算机学报, 2003, 26(8): 919–924. [doi: 10.3321/j.issn:0254-4164.2003.08.004]

[7] 王荣全, 欧阳丹彤, 王艺源, 刘思光, 张立明. 结合 DOEC 极小化策略的 SAT 求解极小碰集方法. 计算机研究与发展, 2018, 55(6):

1273–1281. [doi: 10.7544/issn1000-1239.2018.20160809]

[14] 邓召勇, 欧阳丹彤, 耿雪娜, 刘杰. 基于动态极大元素覆盖值的极小碰集求解算法. 计算机研究与发展, 2018, 55(4): 791–801. [doi: 10.7544/ISSN1000-1239.2018.20160900]

[15] 刘思光, 欧阳丹彤, 张立明. 极小碰集求解中候选解极小性判定方法. 软件学报, 2018, 29(12): 3733–3746. [doi: 10.13328/j.cnki.jos.005311]

[18] 陈晓梅, 孟晓风, 乔仁晓. 基于 BNB-HSSE 计算全体碰集的方法. 仪器仪表学报, 2010, 31(1): 61–67. [doi: 10.19650/j.cnki.cjsi.2010.01.011]

[21] 黄杰, 陈琳, 邹鹏. 一种求解极小诊断的遗传模拟退火算法. 软件学报, 2004, 15(9): 1345–1350.

赵相福(1981－), 男, 博士, 教授, 研究生导师, CCF 高级会员, 主要研究领域为基于模型的故障诊断, 区块链, 智能诊断推理, SAT.

童向荣(1975－), 男, 教授, 研究生导师, CCF 会员, 主要研究领域为机器学习, 软件安全性.

黄森(1995－), 男, 硕士生, CCF 学生会员, 主要研究领域为基于模型的故障诊断.

欧阳丹彤(1968－), 女, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为基于模型的故障诊断, 自动推理, SAT.

魏霞(1995－), 女, 硕士生, 主要研究领域为基于模型的故障诊断.

张立明(1980－), 男, 博士, 教授, 研究生导师, CCF 专业会员, 主要研究领域基于模型的故障诊断, 自动推理, SAT.