软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(4):1964−1979 [doi: 10.13328/j.cnki.jos.006845]

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

针对 MUS 求解问题的加强剪枝策略*

蒋璐宇[1,3], 欧阳丹彤[1,3], 董博文[2,3], 张立明[1,3]

[1](吉林大学 计算机科学与技术学院, 吉林 长春 130012)

[2](吉林大学 软件学院, 吉林 长春 130012)

[3](符号计算与知识工程教育部重点实验室 (吉林大学), 吉林 长春 130012)

通信作者: 张立明, E-mail: limingzhang@jlu.edu.cn

摘　要: 极小不可满足子集 (minimal unsatisfiable subsets, MUS) 的求解是布尔可满足性问题中的一个重要子问题. 对于一个给定的不可满足问题, 其 MUS 的求解能够反映出问题中导致其不可满足的关键原因. 然而, MUS 的求解是一项极其耗时的任务, 不同的剪枝过程将直接影响到搜索空间的大小、算法的迭代次数, 从而影响算法的求解效率. 提出一种针对 MUS 求解的加强剪枝策略 ABC (accelerating by critical MSS), 依据 MSS、MCS、MUS 这 3 者之间的对偶性和碰集关系特点, 提出 cMSS 和 subMUS 概念, 并总结出 4 条性质, 即每个 MUS 必是 subMUS 的超集, 进而在避免对 MCS 的碰集进行求解的情况下有效利用 MUS 和 MCS 互为碰集的特征, 有效避免求解碰集时的时间开销. 当 subMUS 不可满足时, 则 subMUS 是唯一的 MUS, 算法将提前结束执行; 当 subMUS 可满足时, 则剪枝掉此节点, 进而有效避免对求解空间中的冗余空间进行搜索. 同时, 通过理论证明 ABC 策略的有效性, 并将其应用于目前最高效的单一化模型算法 MARCO 和双模型算法 MARCO-MAM, 在标准测试用例下的实验结果表明, 该策略可以有效地对搜索空间进行进一步剪枝, 从而提高 MUS 的枚举效率.

关键词: 极小不可满足子集; 极大可满足子集; MUS 枚举; 幂集探索; 不可行分析

中图法分类号: TP18

中文引用格式: 蒋璐宇, 欧阳丹彤, 董博文, 张立明. 针对MUS求解问题的加强剪枝策略. 软件学报, 2024, 35(4): 1964–1979. http://www.jos.org.cn/1000-9825/6845.htm

英文引用格式: Jiang LY, Ouyang DT, Dong BW, Zhang LM. Enhanced Pruning Scheme for Enumerating MUS. Ruan Jian Xue Bao/Journal of Software, 2024, 35(4): 1964–1979 (in Chinese). http://www.jos.org.cn/1000-9825/6845.htm

Enhanced Pruning Scheme for Enumerating MUS

JIANG Lu-Yu[1,3], OUYANG Dan-Tong[1,3], DONG Bo-Wen[2,3], ZHANG Li-Ming[1,3]

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

[2](College of Software, Jilin University, Changchun 130012, China)

[3](Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education (Jilin University), Changchun 130012, China)

Abstract: Enumerating minimal unsatisfiable subsets (MUS) is an important subproblem in the Boolean satisfiability problem. For an unsatisfiable problem, the MUS enumeration can reflect the key factors resulting in its unsatisfiability. However, enumerating MUS is extremely time-consuming, and different pruning schemes will directly affect the size of the search space and the total number of iterations, thus affecting the algorithm efficiency. This study proposes a novel enhanced pruning scheme, accelerating by critical MSS (ABC), to accelerate the MUS enumeration. According to the relationship among maximal satisfiable subsets (MSS), minimal correction sets (MCS), and MUS, the concepts of cMSS and subMUS are put forward. Additionally, four properties are summarized, namely that

* 基金项目: 国家自然科学基金 (62076108, 61872159, 61972360)
收稿时间: 2022-06-15; 修改时间: 2022-11-06; 采用时间: 2022-11-23; jos 在线出版时间: 2023-07-28
CNKI 网络首发时间: 2023-07-31

each MUS must be a superset of subMUS, and then the feature that MUS and MCS are mutually hitting sets can be effectively employed to avoid the time cost in solving hitting sets of MCS. When the subMUS is unsatisfiable, it will be the only MUS, and the algorithm will terminate in advance; otherwise, the node representing subMUS will be pruned to effectively avoid searching the non-solution space. Meanwhile, the effectiveness of the proposed ABC scheme is proven by theorem, which has been applied to the state-of-the-art algorithms MARCO and MARCO-MAM, respectively. Experimental results on SAT11 MUS benchmarks show the proposed scheme can effectively prune the search space to improve the enumeration efficiency of MUS.

Key words: minimal unsatisfiable subset (MUS); maximal satisfiable subset (MSS); MUS enumeration; power set exploration; infeasibility analysis

布尔可满足性问题 (Boolean satisfiability problem, SAT) 是第一个被证明的 NP 完全问题[1], 也是人工智能自动推理领域中的关键问题, 在电子设计自动化 (EDA)[2]、基于模型诊断[3−5]、软件验证[6]、组合优化[7−9]、自动定理证明[10]等问题上有着广泛的应用. 对于一个给定的布尔逻辑公式, SAT 求解器判断其可满足性: 若公式可满足, 则输出“可满足”并返回一组使得公式被满足的对布尔变量的完全赋值; 若公式不可满足, 则仅输出“不可满足”. 因此, 对于那些不可满足的情况, 若单纯使用 SAT 求解器仅能得出一个“不可满足”的结论, 无法对于公式中的不可满足性做进一步的分析. 在实际生产应用中, 对于不可满足性的进一步分析是有现实意义和价值的. 例如, 在需求分析中, 每个需求对应着正在开发的系统中的约束, 对需求的一致性 (也称可满足性) 检查即为检查所有的需求是否能同时被满足. 若约束集合是不可满足的, 则需要识别并修复这些需求之间的冲突.

当问题的约束 (即子句) 不能同时被满足时, 就构成了一个不可满足问题. 求解该问题的极小不可满足子集 (minimal unsatisfiable subset, MUS) 可以分析出公式中的具体哪些子句之间存在冲突 (即不能同时被满足); 求解该问题的极大可满足子集 (maximal satisfiable subset, MSS) 可以分析出公式中哪些子句是一致的 (即可以同时被满足); 求解该问题的极小修正集 (minimal correction set, MCS) 可以分析出去除哪些子句可以使得公式被满足. 对于一个不可满足的情况, 它们能够从不同角度对不可满足性的原因、解决方案给出更具体的分析. 作为 SAT 问题的一个重要扩展, MUS 的枚举同样有着广泛的现实应用, 例如本体调试[11]、本体中不一致性检测[12]、形式等效性检查[13]、布尔函数的 2-分解[14,15]、基于模型的诊断问题[16,17]等领域中都有重要应用.

近年来, 国内外众多学者致力于对 MUS 枚举方法进行研究[17−32]. 各式各样的枚举算法相继被提出, 从不同角度提升了枚举性能. 在早期, 算法集中于求解单个 MUS. 然而, 由于 MUS 是极小不可满足子集, 它们对应于一个没有被满足的约束系统中的“冲突”, 识别出越多的冲突, 越有助于我们理解系统没有被满足的原因, 从而更好地分析并修复系统. 因此, 单一 MUS 的提取意义有限, 后续开始有工作倾向于更多 MUS 的枚举, 甚至全部枚举. 然而, 在通常情况下, 由于 MUS 的个数与给定 CNF (conjunctive normal form) 公式的大小指数相关, 对于一些较大的实例来说, 其 MUS 的全部枚举将是不可行的. 因此, 当实例的 MUS 完全枚举不可行时, 对其尽可能多的部分枚举成为最佳选择. 近年来, 相关研究开始集中于 MUS 的部分枚举, 本文的研究也是针对 MUS 的部分枚举. 目前, MUS 的枚举方法主要有两大分支: 基于碰集对偶性关系的算法和基于种子缩减的算法.

基于碰集对偶性关系的算法利用 MCS 与 MUS 互为极小碰集的性质进行求解, 这类方法通常先求出部分或全部的 MSS, 然后根据 MCS 与 MSS 互为补集的性质计算出每个 MSS 所对应的 MCS, 对 MCS 集合求极小碰集从而计算出 MUS 的集合, 代表的算法有 DAA[18]、CAMUS[19]、eMUS[20]等. 2005 年, Bailey 等人[18]提出了 DAA 算法, 通过对当前节点进行扩充操作得到一个 MSS, 从而得到其对应的补集 MCS, 对当前的 MCS 集合求极小碰集并判断其可满足性, 不断迭代上述过程从而计算出 MUS 的集合. 但由于 DAA 算法的枚举不具有完备性, 2008 年, Liffiton 等人[19]在 DAA 的基础上提出了完备的 CAMUS 算法, 与之不同的是, CAMUS 首先计算出全部的 MSS, 再取其补集得到全部的 MCS, 并对 MCS 的集合求极小碰集得到全部 MUS 的集合. 可以看出, CAMUS 算法可以对 MUS 进行完备求解, 但其存在一个明显的短板——在对 MUS 的枚举开始前必须计算出全部的 MCS, 对于存在指数级数目 MCS 的不可满足问题而言, 使用 CAMUS 算法枚举 MUS 就会变得不可行. 2013 年, Previti 等人[20]为了克服 CAMUS 算法的这一显著短板提出了 eMUS 算法, 算法中使用极大化模型加速了求解过程. 在极大化模型下, 若当前节点可满足则必是 MSS, 可以省去扩充操作的时间消耗. 同时, 对于每一个探索的种子节点,

即代表给定公式的子句集的某一个子集的节点, 根据其可满足性必能得到一个 MUS 或 MCS, 若得到的是 MCS 则利用碰集的对偶性求解出 MUS, 因此该算法不必等到全部 MCS 求出后再开始枚举 MUS. 在这类算法中, 由于极小碰集的求解复杂度高且算法要多次调用碰集求解过程, 因此这类算法普遍存在耗时较大的问题.

基于种子缩减的算法则是利用收缩操作遍历当前种子节点的子集, 在找到其对应的 MUS 后, 对其超集进行剪枝, 从而避免了对无解空间的探索, 这类方法的代表算法有 MARCO[21−23]、MARCO-MAM[24,25]、TOME[26]、ReMUS[27] 等. 2013 年, Liffiton 等人 [21] 提出结合哈斯图枚举 MUS/MSS 的 MARCO 方法, 对于在哈斯图中任意一个未探索的种子节点, 根据其可满足性做相应的扩充 (或收缩) 操作, 从而得到一个 MSS (或 MUS). 2016 年, Liffiton 等人[22]针对尽可能减少对扩充和收缩操作的调用次数、让算法更倾向于优先找到不可满足的种子等目标, 将 eMUS 中极大化模型的概念引入用于优化 MARCO 算法, 省去了对可满足种子节点的扩充过程. 同年, Zhao 等人[23]提出了极大化 MARCO 算法的多种子并行求解方法, 通过并行计算加速了算法枚举效率. 2019 年, 欧阳丹彤等人[24]提出了基于双模型的 MARCO-MAM 算法, 在极大化模型的 MARCO 算法基础上增加了中间模型, 用于加速 MUS 或 MSS 的枚举. 文献 [22] 中指出, 在这类算法中扩充和收缩操作一共平均占据总运行时间的约 80%. 因此, 扩充和收缩操作是基于种子缩减算法的主要耗时操作. 要想减少对扩充和收缩操作的调用, 就需要对搜索空间进行更充分的剪枝. 通过剪枝更多节点来减少算法所需的迭代次数, 从而达到减少对扩充和收缩操作调用次数的目的.

这两类算法从不同的角度出发来枚举 MUS, 它们的利弊也各不相同. 在上述第 1 类方法中, 碰集的求解是算法中的主要耗时操作; 而在第 2 类方法中, 由于调用一次收缩 (或扩充) 操作将涉及对其子集 (或超集) 的可满足性逐个判断直到得到一个 MUS (或 MSS), 因此调用收缩和扩充操作的过程则是算法中最为耗时的部分. 本文在对上述求解 MUS 算法的深入分析基础上, 意图设计出一种将上述两类算法的优势结合起来的策略来加强对搜索空间剪枝. 借助第 1 类方法中使用 MUS、MCS 及 MSS 三者之间的关系来枚举 MUS 的思想, 对使用第 2 类方法的基本框架的算法制定剪枝方案, 从而使应用该剪枝策略后的算法拥有了这两类方法各自的优势. 考虑到部分具有特殊结构特征的 MSS 可以帮助算法对搜索空间进行剪枝, 提出关键 MSS 的概念, 利用关键 MSS 的特殊性质对搜索空间中的无解空间进行剪枝, 从而提高枚举的效率.

通过上述分析, 我们将本文提出的加强剪枝策略 ABC (accelerating by critical MSS) 的优势总结为以下 3 点: 首先, 当算法求解出关键 MSS 时, 通过 MSS、MCS、MUS 这 3 者之间的相互关系和碰集的特点, 可以直接构造出一个子句集, 使得该集合中的每一个元素都一定包含在该问题的 MUS 中. 这样保证了整个算法中没有碰集求解过程的情况下, 利用碰集自身的性质和 3 种集合间的关系而获得了碰集求解的效果, 避免计算碰集的大范围时间开销. 其次, 极大化模型使得算法前期在哈斯图的顶部区域搜索, 而关键 MSS 自身特点使得其在哈斯图中的位置一定处于顶部区域. 因此, 本文提出的策略与使用极大化模型的算法框架结合起来将可以保证在算法执行的前期就可以找到尽可能多的关键 MSS, 从而保证其构造出的 MUS 子集尽可能大, 对无解空间的剪枝也随之更加充分. 最后, 本文通过证明得出结论: 当关键 MSS 构造出的子句集不可满足时, 那么它是该问题唯一的 MUS. 因此, 若构造出的子句集判断为不可满足时, 使用本文提出的 ABC 策略可以使算法提前结束执行.

本文第 1 节介绍本文所需的基础知识, 包括 SAT 问题中的相关概念, MUS、MSS 和 MCS 的概念及 3 者之间的关系, 哈斯图结构及相关操作. 第 2 节介绍 MARCO 算法和 MARCO-MAM 算法. 第 3 节介绍本文提出的加强剪枝策略 ABC, 并总结出 4 个性质从理论上证明了策略的正确和有效性. 第 4 节将所提出的方法应用于已有的 MARCO 算法和 MARCO-MAM 算法中, 通过对比实验验证了所提模型的有效性. 最后总结全文.

1 基础知识

本节主要介绍本文所涉及的基本概念, 首先介绍 SAT 问题中的定义, 然后介绍 MUS、MSS 及 MCS 的基本概念及相关性质, 最后介绍哈斯图及对哈斯图的探索操作和剪枝操作.

1.1 SAT 问题相关定义

SAT 问题是理论计算机科学中的重要问题, 同时也是 MUS、MCS 和 MSS 问题的基础. 下面给出 SAT 问题中相关定义.

定义 1. 文字 (literal)[33]. 给定一个布尔变量集合 $X=\{x_1,x_2,\ldots,x_n\}$, 文字 l_i 是变量 x_i 或变量 x_i 的否定 $\neg x_i$. 前者称为正文字, 后者称为负文字.

定义 2. 子句 (clause)[33]. 子句 c_i 是若干个文字的析取 $c_i=l_{i1}\vee l_{i2}\vee\ldots\vee l_{ij}$.

定义 3. 合取范式 (CNF)[33]. 一个 CNF 公式 F 是若干个子句的合取, $F=c_1\wedge c_2\wedge\ldots\wedge c_m$.

定义 4. 命题可满足性问题 (SAT)[33]. 对于给定的布尔变量集合, 若能找到一组对变量的真值指派, 使得给定的命题公式的值为真, 则称该问题是可满足的; 反之, 若不存在这样的真值指派, 则称该问题是不可满足的.

1.2 不可满足问题中的概念及关系

MUS 是 SAT 问题的扩展, 在不可满足问题中, 每个约束描述为一个子句. 通过对不可满足问题中 MUS、MSS 和 MCS 的计算, 可以从繁多的约束中分析出导致问题不可满足的原因. 下面给出 MUS、MSS 和 MCS 的概念以及三者之间的关系.

定义 5. 极小不可满足子集 (MUS)[24]. 称集合 U 是不可满足问题中子句集 F 的 MUS, 当且仅当 $U\subseteq F$, 且 U 不可满足, 且 $\forall c\in U, U\setminus\{c\}$ 可满足.

定义 6. 极大可满足子集 (MSS)[24]. 称集合 S 是不可满足问题中子句集 F 的 MSS, 当且仅当 $S\subseteq F$, 且 S 可满足, 且 $\forall c\in F\setminus S, S\cup\{c\}$ 不可满足.

定义 7. 极小修正集 (MCS)[24]. 称集合 C 是不可满足问题中子句集 F 的 MCS, 当且仅当 $C\subseteq F$, 且 $F\setminus C$ 可满足, 且 $\forall c\in C, F\setminus(C\setminus\{c\})$ 不可满足.

根据定义 6 和定义 7 可以看出, 每一个 MSS 的补集都是一个 MCS, 反之亦然. 文献 [20] 中详细说明了 MUS 与 MCS 之间存在互为极小碰集的对偶关系. 下面给出极小碰集的概念:

定义 8. 碰集 (hitting set)[16]. 称集合 H 是集合簇 Q 的碰集, 当且仅当 $H\subseteq\bigcup_{Q'\in Q}Q'$, $\forall Q'\epsilon Q, H\cap Q'\neq\varnothing$.

定义 9. 极小碰集 (minimal hitting set)[16]. 称集合 H_M 是集合簇 Q 的极小碰集, 当且仅当集合 H_M 是集合簇 Q 的碰集且集合 H_M 的任意真子集都不是集合簇 Q 的碰集.

例 1: 给定一个布尔变量集合 $X=\{a,b\}$ 和在其上的子句集 $C=\{l_1:(a\vee b), l_2:(\neg a), l_3:(a\vee\neg b), l_4:(b)\}$, 则该问题所对应的所有 MUS 为 $\{l_1,l_2,l_3\}$ 和 $\{l_2,l_3,l_4\}$, 所有 MSS 为 $\{l_1,l_3,l_4\}$、$\{l_1,l_2,l_4\}$ 和 $\{l_2,l_3\}$, 所有 MCS 为 $\{l_2\}$、$\{l_3\}$ 和 $\{l_1,l_4\}$. 且根据 MCS 与 MSS 互为补集的关系、MCS 与 MUS 互为极小碰集的关系可知, 只要求出其中一者的全部集合, 其余两者就可以通过这些关系而求解出来.

1.3 哈斯图

在 MUS 枚举问题中, 求解的关键在于对子句集的幂集的探索. 而哈斯图结构能够很好地刻画出整个搜索空间并能清晰反映出集合间的包含关系. 其形状上中间鼓, 两头尖, 中间层的顶点数量最多, 具有对称性. 一个含有 4 个子句的不可满足问题对应的哈斯图如图 1 所示. 哈斯图在横向和纵向上都具有显著的特点.

(1) 横向: 哈斯图的最顶层节点代表子句集 C, 最底层节点代表空集. 同一水平层的节点所代表的集合具有相同的基数, 自上而下基数逐层减一. 假设子句集 C 中共包含 n 个子句, 令最底层为第 1 层, 那么第 i 水平层 (i 从 1 开始编号) 含有的节点个数为 C_n^i.

(2) 纵向: 哈斯图的纵向体现出节点所代表集合的包含关系. 对于哈斯图中的每个节点, 其下一层与之有边相连的节点代表的是该集合的子集, 其上一层与之有边相连的节点代表的是该集合的超集, 相邻层节点间无边相连则说明两个节点所代表的集合之间无包含关系. 由此可见, 从一个节点出发, 在向上的路径可到达的所有节点都是该节点的超集, 在向下的路径可到达的所有节点都是该节点的子集.

公理 1. 一个可满足的子句集的任何一个子集也一定是可满足的, 一个不可满足的子句集的任何一个超集也一定是不可满足的.

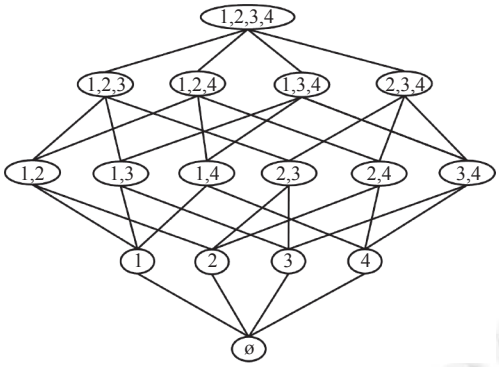


图 1 哈斯图

算法通过在哈斯图上执行相应的探索操作和剪枝操作来完成对 MUS 的枚举过程. 探索操作对当前种子节点执行向上或向下探索, 使其扩充为一个 MSS 或者收缩为一个 MUS; 剪枝操作则是根据公理 1 在哈斯图上进行相应的剪枝, 剪去哈斯图中 MSS 对应节点的全部子集所在节点、MUS 对应节点的全部超集所在节点, 从而缩小算法的搜索空间. 下面给出两种探索操作 (收缩操作和扩充操作) 以及两种剪枝操作 (向上剪枝和向下剪枝操作) 的定义.

定义 10. 收缩操作 (*Shrink*)[21]. 给定一个子句集 C 和当前不可满足的种子 *seed*, 对 $\forall$ 子句 $l \in seed$, 将所有满足执行操作 $seed = seed \backslash \{l\}$ 后不改变 *seed* 不可满足性质的子句 l 从 *seed* 中删除, 从而将 *seed* 收缩成为一个 MUS 的过程称为对该种子的收缩操作. 对 *seed* 的收缩过程可以看作在哈斯图中从 *seed* 节点向下方路径探索的过程.

定义 11. 扩充操作 (*Grow*)[21]. 给定一个子句集 C 和当前可满足的种子 *seed*, 对 $\forall$ 子句 $l \in C \backslash seed$, 将所有满足执行操作 $seed = seed \cup \{l\}$ 后不改变 *seed* 可满足性质的子句 l 加入 *seed* 中, 从而将 *seed* 扩充成为一个 MSS 的过程称为对该种子的扩充操作. 对 *seed* 的扩充过程可以看作在哈斯图中从 *seed* 节点向上方路径探索的过程.

定义 12. 向上剪枝操作 (*BlockUp*)[21]. 称对给定集合的全部超集从哈斯图上剪枝的操作为对该集合的向上剪枝操作.

此操作是针对 MUS 的剪枝操作, 通过在 *map* 中加入以下子句实现. 对于一个 MUS U, 有:

$$BlockUp(U) = \bigvee_{i:C_i \in U} \neg x_i,$$

其中, C_i 为 U 中的一个子句, x_i 是 *map* 中代表子句 C_i 的变量.

定义 13. 向下剪枝操作 (*BlockDown*)[21]. 称对给定集合的全部子集从哈斯图上剪枝的操作为对该集合的向下剪枝操作.

此操作是针对 MSS 的剪枝操作, 通过在 *map* 中加入以下子句实现: 对于一个 MSS S, 有:

$$BlockDown(S) = \bigvee_{i:C_i \notin S} x_i,$$

其中, 将公式的完整子句集命名为 F, C_i 为集合 $F \backslash S$ 中的一个子句, x_i 是 *map* 中代表子句 C_i 的变量.

2 MUS 求解算法回顾

本节将回顾 MARCO 算法, 包括传统版本和优化版本, 即极大化模型的 MARCO 算法, 以及 MARCO-MAM 算法.

2.1 MARCO 算法

MARCO 算法是近年来最受欢迎的 MUS 枚举算法之一, 其框架简单、枚举快速. MARCO 算法的伪代码如算法 1 所示, 其基本流程是不断地重复以下 3 个步骤.

(1) 当哈斯图中存在未探索节点时, 取一个未探索节点 *seed*.

(2) 判断 *seed* 的可满足性: 若可满足, 将 *seed* 扩充成一个 MSS; 否则, 将 *seed* 收缩为一个 MUS.

(3) 将其全部子集或全部超集标记为已探索.

算法 1. MARCO.

输入: 不可满足的子句集 *C*;

输出: 已找到的子句集 *C* 的 MUS 集合和 MSS 集合.

```
1. map ← BooleanFormula(nvars=|C|)
2. while map 是可满足的 do
3.    seed ← GetUnexplored(map)
4.    if seed is SAT then
5.        MSS ← Grow(seed)
6.        output MSS
7.        map ← map ∧ BlockDown(seed)
8.    else
9.        MUS ← Shrink(seed)
10.       output MUS
11.       map ← map ∧ BlockUp(seed)
12.   end if
13. end while
```

在 MARCO 算法的每一次迭代中, 都会将当前的 *seed* 转化为一个 MSS 或 MUS, 因此保证了每轮迭代都会有一个新输出的解. 根据可满足性问题的特点可知, 一个可满足的子句集的任何一个子集也一定是可满足的, 一个不可满足的子句集的任何一个超集也一定是不可满足的. 因此, 算法每输出一个 MSS (或 MUS) 时, 将通过 *BlockDown* (或 *BlockUp*) 操作向公式 *map* 中添加相应子句以阻塞该 MSS 的全部子集 (或该 MUS 的全部超集), 从而缩减了搜索空间, 避免了算法在无解空间中的探索.

使用极大化模型的 MARCO 算法 (算法 2) 是其作者在 2016 年对于 MARCO 算法提出的一个优化版本[22], 它与上文提到的 MARCO 算法有两个主要差别: (1) 在每轮迭代开始时, MARCO 算法随机选择一个未探索的种子节点, 而极大化模型的 MARCO 算法则选择当前哈斯图中的极大未探索种子节点. (2) 当选择的种子节点判定为可满足时, MARCO 算法需要对该种子进行扩充操作, 使其转化为一个 MSS; 而极大化模型的 MARCO 算法则可以省略对种子的扩充操作, 直接将该种子输出为一个 MSS, 这是由于极大化模型保证了种子节点是极大未探索子集, 因此无需对种子进行扩充操作. 极大化模型 MARCO 算法的伪代码描述在算法 2 中, 可以看出算法 1 和算法 2 的区别也体现在第 3 行和第 5 行. 算法 1 中第 3 行通过调用 *GetUnexplored* 函数从 *map* 中选取一个子句集的未探索子集作为 *seed*, 而算法 2 中第 3 行则调用 *GetUnexploredMax* 函数从 *map* 中选取一个子句集的极大未探索子集作为 *seed*; 算法 1 中第 5 行对当前判定为可满足的 *seed* 执行了扩充操作, 得到一个极大可满足子集, 而算法 2 中第 5 行将当前判定为可满足的 *seed* 直接作为一个极大可满足子集. 优化后的 MARCO 在算法初期更倾向于找到更多的 MUS, 同时省去了扩充操作的时间开销, 在一定程度上加速了枚举 MUS 的效率.

算法 2. MARCO+: 使用极大化模型的优化版本.

输入: 不可满足的子句集 *C*;

输出: 已找到的子句集 *C* 的 MUS 集合和 MSS 集合.

```
1. map ← BooleanFormula(nvars=|C|)
2. while map 是可满足的 do
```

```
3.    seed ← GetUnexploredMax(map)
4.    if seed is SAT then
5.        MSS ← seed
6.        output MSS
7.        map ← map ∧ BlockDown(seed)
8.    else
9.        MUS ← Shrink(seed)
10.       output MUS
11.       map ← map ∧ BlockUp(seed)
12.   end if
13. end while
```

2.2 MARCO-MAM 算法

MARCO-MAM 算法是在极大化模型的 MARCO 版本上的进一步改进, 使用极大化模型和中间模型共同求解, 算法的伪代码描述在算法 3 中. 与极大化模型 MARCO 不同的是, 当目前选取的种子节点 *seed* 不可满足时, 算法将通过调用 *GetUnexploredMid* 函数引入一个中间种子 *midseed* 辅助求解. 该中间种子所代表的集合是 *seed* 所代表集合的子集, 且其基数是 *seed* 所代表集合的一半, 该中间种子节点即在哈斯图中从节点 *seed* 处向下路径为其层数的一半的某个子集表示的节点. 再对 *midseed* 做与 *seed* 相同的操作: 先判断其可满足性, 若可满足则将它扩充为一个 MSS, 否则将其收缩为一个 MUS. 相比之下, 极大化模型更倾向于从哈斯图顶部开始逐步向下搜索, 而极大化—中间模型则更倾向于在哈斯图的顶部—中部—顶部—中部…这样往复地向上、下搜索.

算法 3. MARCO-MAM.

输入: 不可满足的子句集 *C*;
输出: 已找到的子句集 *C* 的 MUS 集合和 MSS 集合.

```
1. map ← BooleanFormula(nvars=|C|)
2. while map 是可满足的 do
3.    seed ← GetUnexploredMax(map)
4.    if seed 是可满足的 then
5.        MSS ← seed
6.        output MSS
7.        map ← map ∧ BlockDown(seed)
8.    else
9.        MUS ← Shrink(seed)
10.       output MUS
11.       map ← map ∧ BlockUp(seed)
12.       midseed ← GetUnexploredMid(map, seed)
13.       if midseed 未被探索过 then
14.          if midseed 是可满足的 then
15.             MSS ← Grow(midseed)
16.             output MSS
17.             map ← map ∧ BlockDown(midseed)
18.          else
```

```
19.            MUS ← Shrink(midseed)
20.            output MUS
21.            map ← map ∧ BlockUp(midseed)
22.          end if
23.        end if
24.    end if
25. end while
```

3 利用关键 MSS 的加强剪枝策略

本文提出一种加强对哈斯图的剪枝策略, 并提出关键 MSS 的概念. 根据 MUS、MCS 和 MSS 之间的关系, 我们可以通过关键 MSS 进一步对哈斯图进行剪枝操作. 本节首先给出所提出剪枝策略的理论依据, 之后介绍将其结合到 MARCO 算法和 MARCO-MAM 算法中的整体算法流程.

3.1 算法的理论基础

现有的 MARCO 算法和 MARCO-MAM 算法的剪枝过程均通过已找到的 MUS 和 MSS 来完成的, 在哈斯图中裁剪掉 MUS 的全部超集、MSS 的全部子集, 从而缩减了算法的搜索空间. 然而, 当求解的 MSS 具有某些特定的结构特征时, 可以利用这些 MSS 对搜索空间进行进一步的剪枝操作. 在下述定义 14 中, 我们给出了这种满足“具有特定的结构特征的 MSS”的定义.

定义 14. 关键极大可满足子集 (critical MSS, cMSS). 给定一个子句集 C, 称一个极大可满足子集 M 是 C 下的一个关键极大可满足子集当且仅当集合的基数满足 $|M|=|C|-1$, 即, C 中只有一个子句不在 M 中.

定义 15. 子极小不可满足子集 (subMUS). 称所有关键极大可满足子集的补集中唯一的子句组成的集合为子极小不可满足子集.

在给出关键 MSS 和 subMUS 的定义后, 我们对本文所提出的加强剪枝策略 ABC 表述如下.

算法每求出一个关键 MSS 时, 更新 subMUS 并判断其可满足性: 若可满足, 则对代表集合 subMUS 的节点做 *BlockDown* 操作; 否则, 输出 subMUS 为 MUS, 算法结束.

性质 1. 集合 subMUS 一定包含在每一个 MUS 中.

证明: 假设给定的子句集 C 中有 n 个 cMSS, 每个 cMSS 记为 $cMSS_i$ $(1\leqslant i\leqslant n)$, $cMSS_i$ 中缺少的子句记为 c_i, 则 subMUS $=\{c_1,\ldots,c_n\}$. 根据每个 MSS 的补集都是一个 MCS, 则有每个 $cMSS_i$ 对应的 $MCS_i=\{c_i\}$, 又根据 MUS 和 MCS 互为极小碰集的性质可知, 每个 MUS 中必然包含 $c_1,\ldots,c_n$. 因此集合 subMUS 必然是每个 MUS 的子集, 它一定包含在每一个 MUS 中.

性质 2. 使用 MARCO 类算法框架每搜索到一个关键 MSS, 剪枝后哈斯图中的剩余节点一定是代表 subMUS 的超集的节点.

证明: 哈斯图中的节点除代表 subMUS 和其超集的节点外, 剩余节点或者是代表 subMUS 真子集的节点, 或者是与 subMUS 之间无包含关系的集合的节点. 采用反证法证明.

首先, 假设剩余节点中存在代表 subMUS 真子集中的节点. (1) 该节点代表空集: 由于空集是任何一个关键 MSS 的子集, 因此算法在找到第 1 个关键 MSS 时, 就已经通过 *BlockDown* 操作将代表空集的节点剪枝, 故剩余节点中一定没有代表空集的节点. (2) 该节点代表 subMUS 的一个非空真子集: 令该非空真子集为 Q, 则 Q 中至少存在一个元素 e, 使得 $e\in$ subMUS 且 $e\notin Q$. 由 $e\in$ subMUS 可知, 已求解出的关键 MSS 必有一个为集合 $P=\{i|i\in$ 子句集C且$i\neq e\}$, 即从子句集 C 中去除一个元素 e 的集合. 由于集合 Q 中没有元素 e, 而集合 P 是所有没有元素 e 的集合中基数最大的集合, 故必有 Q 是 P 的子集, 而集合 P 又是一个关键 MSS, 求解出该关键 MSS 时通过 *BlockDown* 操作已经将其全部子集所在节点剪枝, 因此不存在这样的集合 Q, 故假设不成立, 剩余节点中没有代

表 subMUS 真子集中的节点.

接下来, 假设剩余节点存在代表与 subMUS 之间无包含关系的集合的节点, 任取其中一个与 subMUS 无包含关系的集合 S, 那么至少存在一个元素 i 使得 $i \in S$ 且 $i \notin$ subMUS. 由 $i \notin$ subMUS 可知 i 必属于用于构造 subMUS 的每一个关键 MSS, 故当第 1 个关键 MSS 被求解出来时, 通过对此关键 MSS 所在节点做 *BlockDown* 操作已经使得此关键 MSS 的全部子集所在节点被剪枝, 因此不存在这样的集合 S, 故该假设不成立.

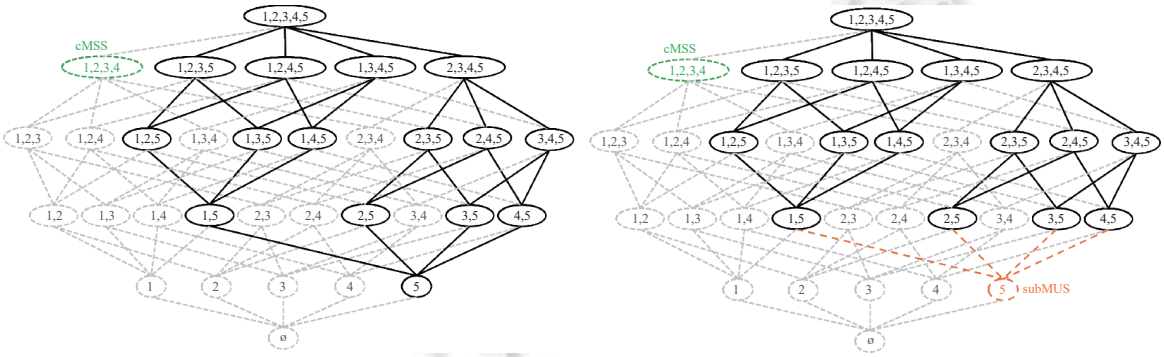
性质 3. 若 subMUS 是不可满足的, 则 subMUS 本身是该问题唯一的 MUS.

证明: 假设 subMUS 是不可满足的, 那么一定存在一个 subMUS 的非空子集, 集合 subMUS′是一个 MUS. 由性质 1 可知, subMUS 一定是 MUS 的子集. 那么, 集合 subMUS 自身是一个 MUS. 又由于 MUS 具有极小性, 因此 subMUS 的严格超集都不会是 MUS, 故 subMUS 是唯一的 MUS. 另外, 也可以通过性质 2 来解释性质 3, 由于经过关键 MSS 的剪枝后, 哈斯图中剩余未探索节点所表示的集合一定是 subMUS 的超集, 当 subMUS 不可满足时, 由于未探索节点中没有代表 subMUS 真子集的节点, 故哈斯图中代表 subMUS 的节点是极小未探索子集, 因此无需收缩操作, 直接将 subMUS 输出为一个 MUS, 同时对其全部超集作 *BlockUp* 操作, 故哈斯图中剩余节点全部被阻塞, 哈斯图中无未探索节点, 算法结束, 因此我们通过性质 2 也证明了性质 3 的正确性.

性质 4. 在仅考虑在关键 MSS 下的剪枝的前提下, 算法每找到一个关键 MSS, 使用 ABC 策略后将会多剪枝一个节点, 该节点为代表 subMUS 集合的节点.

证明: 若要证明此性质, 只需证明对 subMUS 所在的节点做 *BlockDown* 操作只剪枝一个节点即 subMUS 本身, 只需证明在做此 *BlockDown* 操作前 subMUS 的全部真子集所在的节点均已被剪枝. 而根据性质 2 可知, 在当前 subMUS 下最后一个关键 MSS 被求出时, 哈斯图中 subMUS 的真子集所在的节点均已被剪枝掉, 因此得证.

根据性质 1 和性质 2 可知, 通过关键极大可满足子集可以构造出一个子极小不可满足子集, 将整个搜索空间从 C 的幂集缩减为子极小不可满足子集的全部超集. 性质 3 则反映出在一种较特殊的情况下, 即子句集中的 MSS 均为关键 MSS 时, 使用本文所提出的 ABC 策略可以直接求解出子句集中唯一的 MUS 并结束算法的执行, 对比不使用该策略的情况可以提前一轮迭代结束算法, 减少一次对收缩操作的调用, 避免了这一部分的时间损耗. 性质 4 则反映出在绝大多数情况下策略的优化效果, 即每搜索到一个关键 MSS 比不使用该策略的情况下多剪枝一个节点, 如图 2(b) 所示. 该策略结合极大化模型可以使得求解效率更高, 这是由于使用极大化模型可以保证算法优先选择哈斯图中顶层节点开始遍历, 而关键 MSS 具有其基数为子句个数减一的特点, 这一特点使得关键 MSS 一定位于哈斯图的第 2 顶层, 这也意味着关键 MSS 可以尽早被发现, 从而在算法执行的前期加速剪枝过程, 有效缩减算法的搜索空间.



(a) 使用关键 MSS 进一步剪枝策略前的搜索空间　　(b) 使用关键 MSS 进一步剪枝策略后的搜索空间

图 2　给定子句集使用 ABC 剪枝策略前后哈斯图所代表的搜索空间 (虚线节点和边代表被剪枝)

例 2. 假设对子句集 C={1, 2, 3, 4, 5}求解 MUS, 搜索到一个 MSS 为{1, 2, 3, 4}时, 根据定义 15 可知, 该例中的 subMUS 为{5}, 算法的搜索空间可以缩减为仅剩下哈斯图中那些为 subMUS 的超集所对应的节点, 如图 2(a) 所示, 在使用 ABC 策略的情况下将比不使用该策略的情况下多剪枝一个代表集合{5}的节点. 若下一轮迭代中搜

索到集合{1, 2, 3, 5}也是一个 MSS 时, 更新该例中 subMUS 为{4, 5}, 则使用 ABC 策略的情况下又可以比不使用该策略的情况下多剪枝一个代表集合{4, 5}的节点. 因此, 在最优情况下, 使用 ABC 策略所能增加剪枝的节点个数等同于该子句集中的关键 MSS 个数.

3.2 算法的伪代码

算法 4 和算法 5 分别是将本文提出的 ABC 策略应用于 MARCO 算法和 MARCO 算法后得到的 MARCO-ABC 算法和 MARCO-MAM-ABC 算法的伪代码. 其中, ABC 策略在两个伪代码中的位置均位于第 8–16 行. 当算法找到一个 MSS 时, 先通过其集合基数判断是否为关键 MSS. 如果它是关键 MSS, 那么该 MSS 对应的 MCS 为一个单元素集合, 将此 MCS 中的唯一子句, 即关键 MSS 在子句集下缺少的子句加入 subMUS 中 (第 10 行), 判断 subMUS 的可满足性, 若可满足则剪枝 subMUS 集合所对应的节点 (第 11 行), 否则, 输出 subMUS 集合为该问题唯一的 MUS, 算法终止 (第 12–15 行). 由于 subMUS 为不可满足时属于一种较为特殊的情况, 在第 3.1 节中的分析可知, 只有当问题的全部 MSS 都是关键 MSS 时才会导致 subMUS 不可满足, 在这种情况下, 该问题的每一个 MSS 都是关键 MSS, 每一个 MCS 都是单元素集合, 且问题只有一个 MUS, 即 subMUS. 在实际应用中, 它对应于一个过度约束的系统中只有一个冲突的情况. 因此考虑到这种情况的特殊性在使用该策略时也可以选择去掉伪代码中 12–15 行的部分, 这样可以省去第 12 行对 SAT 求解器的调用.

算法 4. MARCO-ABC.

输入: 不可满足的子句集 C;
输出: 已找到的子句集 C 的 MUS 集合和 MSS 集合.

```
1. map ← BooleanFormula(nvars=|C|)
2. while map 是可满足的 do
3.    seed ← GetUnexploredMax(map)
4.    if seed 是可满足的 then
5.       MSS ← seed
6.       output MSS
7.       map ← map ∧ BlockDown(seed)
8.       if seed 是 cMSS then
9.          MCS ← complement(seed)
10.         subMUS ← subMUS ∪ MCS
11.         map ← map ∧ BlockDown(subMUS)
12.         if subMUS 是不可满足的 then
13.            MUS ← subMUS
14.            return
15.         end if
16.      end if
17.   else
18.      MUS ← Shrink(seed)
19.      output MUS
20.      map ← map ∧ BlockUp(seed)
21.   end if
22. end while
```

算法 5. MARCO-MAM-ABC.

输入: 不可满足的子句集 C;
输出: 已找到的子句集 C 的 MUS 集合和 MSS 集合.

```
1. map ← BooleanFormula(nvars=|C|)
2. while map 是可满足的 do
3.    seed ← GetUnexploredMax(map)
4.    if seed 是可满足的 then
5.       MSS ← seed
6.       output MSS
7.       map ← map ∧ BlockDown(seed)
8.       if seed 是 cMSS then
9.          MCS ← complement(seed)
10.         subMUS ← subMUS ∪ MCS
11.         map ← map ∧ BlockDown(subMUS)
12.         if subMUS 是不可满足的 then
13.            MUS ← subMUS
14.            return
15.         end if
16.      end if
17.   else
18.      MUS ← Shrink(seed)
19.      output MUS
20.      map ← map ∧ BlockUp(seed)
21.      midseed ← GetUnexploredMid(map, seed)
22.      if midseed 未被探索过 then
23.         if midseed 是可满足的 then
24.            MSS ← Grow(midseed)
25.            output MSS
26.            map ← map ∧ BlockDown(midseed)
27.         else
28.            MUS ← Shrink(midseed)
29.            output MUS
30.            map ← map ∧ BlockUp(midseed)
31.         end if
32.      end if
33.   end if
34. end while
```

4 实验与分析

本文的实验环境为一台 60 GB RAM、Intel Xeon E5-2690v4 2.6 GHz 的 CPU、64 位 Ubuntu 18.04 版本 Linux

操作系统的主机. 我们将本文提出的加强剪枝策略 ABC 应用于 MARCO 算法和 MARCO-MAM 算法, 分别得到 MARCO-ABC 算法和 MARCO-MAM-ABC 算法并与原算法作对比. 使用的 MARCO (http://www.iwu.edu/~mliffito/marco/) 算法代码为文献 [22] 中的使用极大化模型版本, MARCO-MAM 算法、MARCO-ABC 算法和 MARCO-MAM-ABC 算法均通过在 MARCO 算法的基础上增加相应语句实现 (MARCO-ABC 与 MARCO-MAM-ABC 代码开放下载网址: https://github.com/jiangluyu1998/ABC-scheme-for-MUS). 算法所用的编程语言为 Python 3.7 版本, 并调用 C++语言编写的 MiniSAT 求解器[33]用于种子节点可满足性的判断. 我们采用标准测试用例 SAT11 中的 MUS 实例[34]对算法进行了测试, 并对每组实验设置了 3 种时间限制, 分别为 0.5 h、1 h 和 2 h.

表 1 所示为在相同条件下 MARCO 算法与应用本文提出的 ABC 策略的 MARCO-ABC 算法分别在 0.5 h、1 h 和 2 h 的时间限制下枚举 MUS 个数的比较, 表 2 所示为在相同条件下 MARCO-MAM 算法与应用本文提出的 ABC 策略的 MARCO-MAM-ABC 算法分别在 0.5 h、1 h 和 2 h 的时间限制下枚举 MUS 个数的比较. 各表中加粗的结果表示其枚举个数更多 (对于结果相同的情况, 结果均不加粗), 各表中最后一行的#win 代表在其上方的所有测试用例中, 算法得到的枚举个数超过对方的测试用例个数.

表 1 MARCO 算法和 MARCO-ABC 算法在不同时间限制下枚举 MUS 个数比较

Instances	MARCO			MARCO-ABC		
	0.5 h	1 h	2 h	0.5 h	1 h	2 h
fdmus_b14_134	158	513	1 566	**173**	**584**	**1 664**
fdmus_b14_141	82	199	642	**84**	**203**	642
fdmus_b14_210	80	**205**	**637**	80	204	609
fdmus_b15_291	51	102	231	**53**	**108**	**253**
fdmus_b15_294	47	100	**224**	47	100	218
fdmus_b15_304	**32**	59	143	31	**63**	**144**
fdmus_b15_421	151	294	581	**157**	**327**	**673**
fdmus_b15_463	174	369	764	**189**	**405**	**823**
fdmus_b17_1037	338	338	338	**500**	**1 085**	**2 356**
fdmus_b17_1324	**58**	**117**	228	57	116	**229**
fdmus_b17_1456	**195**	**386**	**766**	186	374	750
fdmus_b17_420	158	318	**738**	**161**	**327**	673
fdmus_b20_141	16	32	67	**18**	**36**	**71**
fdmus_b20_171	20	38	74	**21**	38	74
fdmus_b20_238	44	**93**	222	**45**	92	**236**
fdmus_b20_341	**188**	629	1 562	176	**639**	**1 623**
fdmus_b20_349	33	**66**	141	33	63	**149**
fdmus_b20_381	29	66	**162**	**31**	**67**	158
fdmus_b20_492	38	**88**	**352**	**40**	86	322
fdmus_b20_498	61	179	**552**	**65**	**200**	518
fdmus_b21_111	14	27	55	14	**29**	**60**
fdmus_b21_112	14	26	53	**15**	**27**	53
fdmus_b21_167	**35**	70	**153**	34	70	148
fdmus_b21_481	31	64	**224**	31	**65**	175
fdmus_b21_96	115	300	715	**118**	**311**	**738**
fdmus_b22_102	34	63	**136**	**36**	**70**	133
fdmus_b22_113	12	23	45	12	**25**	**49**
fdmus_b22_133	46	100	226	**54**	**108**	**229**
fdmus_b22_172	21	42	**88**	**23**	42	86
fdmus_b22_241	29	60	**145**	**30**	60	142
fdmus_b22_356	14	**28**	**63**	14	27	62
fdmus_b22_488	51	110	428	51	**122**	**476**
fdmus_b22_600	15	26	62	**20**	**29**	**69**

表 1 MARCO 算法和 MARCO-ABC 算法在不同时间限制下枚举 MUS 个数比较 (续)

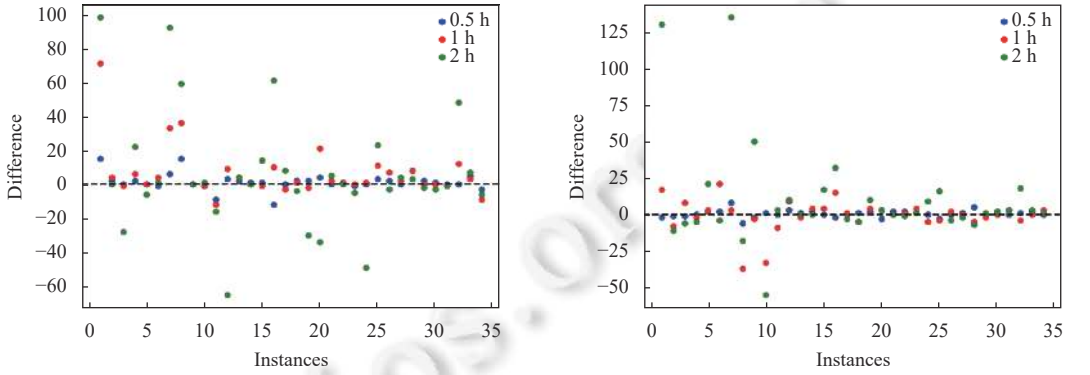
Instances	MARCO			MARCO-ABC		
	0.5 h	1 h	2 h	0.5 h	1 h	2 h
fdmus_b22_662	**23**	**47**	**81**	20	38	75
#win	6	8	14	**20**	**21**	**17**

表 2 MARCO-MAM 算法和 MARCO-MAM-ABC 算法在不同时间限制下枚举 MUS 个数比较

Instances	MARCO-MAM			MARCO-MAM-ABC		
	0.5 h	1 h	2 h	0.5 h	1 h	2 h
fdmus_b14_134	**181**	574	1 486	179	**591**	**1 616**
fdmus_b14_141	**83**	**194**	**598**	82	186	587
fdmus_b14_210	**78**	183	**611**	77	**191**	605
fdmus_b15_291	52	**109**	**252**	52	107	247
fdmus_b15_294	44	93	208	**46**	**96**	**229**
fdmus_b15_304	33	53	**124**	**35**	**74**	120
fdmus_b15_421	142	305	689	**150**	**308**	**824**
fdmus_b15_463	**191**	**420**	**787**	185	383	769
fdmus_b17_1037	**539**	**1 082**	2 362	537	1 079	**2 412**
fdmus_b17_1324	77	**149**	**290**	**78**	116	235
fdmus_b17_1456	178	**383**	746	178	374	**749**
fdmus_b17_420	154	323	664	**157**	**332**	**674**
fdmus_b20_141	17	**35**	70	**18**	33	70
fdmus_b20_171	19	35	74	**21**	**39**	74
fdmus_b20_238	48	92	219	48	**96**	**236**
fdmus_b20_341	**190**	600	1 496	188	**615**	**1 528**
fdmus_b20_349	31	68	**157**	31	**69**	154
fdmus_b20_381	32	**71**	**170**	**33**	66	165
fdmus_b20_492	40	89	375	**42**	**93**	**385**
fdmus_b20_498	**65**	176	510	62	**179**	**513**
fdmus_b21_111	13	29	57	**15**	29	57
fdmus_b21_112	13	26	**52**	**15**	**27**	51
fdmus_b21_167	31	63	151	**32**	**67**	**152**
fdmus_b21_481	28	**63**	152	28	58	**161**
fdmus_b21_96	**118**	**309**	710	115	305	**726**
fdmus_b22_102	36	67	**135**	36	**69**	131
fdmus_b22_113	12	23	**47**	**13**	23	45
fdmus_b22_133	40	**98**	**223**	**45**	93	216
fdmus_b22_172	22	**41**	89	22	39	**90**
fdmus_b22_241	27	60	136	**29**	60	**138**
fdmus_b22_356	14	27	58	14	**30**	**61**
fdmus_b22_488	47	**106**	390	**48**	102	**408**
fdmus_b22_600	14	34	62	**16**	34	**65**
fdmus_b22_662	23	44	76	23	**47**	**77**
#win	8	13	12	**17**	**17**	**19**

从表 1 和表 2 中均可以看出, 无论是 MARCO 算法还是 MARCO-MAM 算法, 在结合了本文提出的加强剪枝策略 ABC 后, 在绝大多数标准测试用例上的结果都有较为明显的提升, 在 3 种时间限制下, 结合本文所提出的剪枝策略 ABC 的算法#win 值均高于未结合 ABC 的算法. 并且在 0.5 h 的时间限制下的优化效果要优于时间限制 2 h 的结果, 这是由于极大化模型导致算法从哈斯图的顶部区域逐渐向下探索, 而由于关键 MSS 自身的集合基数大导致其位于哈斯图的第 2 顶层, 从而会在算法的早期被尽可能多地搜索出来, 而通过性质 4 可知, 每找到一个关

键 MSS, 使用 ABC 就会多剪枝掉一个节点. 因此在早期, ABC 策略的优化效果会更加明显. 为使表 1、表 2 中结果可视化更清晰, 我们在图 3 中绘制横坐标为标准测试用例在表中的序号, 纵坐标为应用 ABC 策略下枚举的 MUS 个数与不应用 ABC 策略下枚举的 MUS 个数的差值的散点图, 如图 3 所示. 图中每一列代表同一个标准测试用例下, 在不同时间限制下枚举个数的差值大小. 需要特殊说明的是, 在图 3(a) 中由于在标准测试用例 fdmus_b17_1037 下 MARCO-ABC 与 MARCO 所枚举的 MUS 个数差异过大, 导致所绘制的散点图中其余数据因与其数据规模差距过大而压缩至纵坐标为 0 的横线附近, 因此在该散点图中我们去掉了标准测试用例 fdmus_b17_1037 中的结果. 从图 3 中可以看出, 在 MARCO 算法和 MARCO-MAM 算法中使用本文提出的 ABC 策略, 均能在大多数标准测试用例下枚举出更多的 MUS, 从而说明了该策略对于提升 MUS 枚举效率是有效的.



(a) MARCO 与 MARCO-ABC 枚举 MUS 个数差值　　(b) MARCO-MAM 与 MARCO-MAM-ABC 枚举 MUS 个数差值

图 3　算法分别应用 ABC 策略与不应用 ABC 策略所枚举的 MUS 个数差值的散点图

通过对表 1 和表 2 之间结果的对比可以发现, ABC 策略对 MARCO 算法的优化效果比对 MARCO-MAM 算法的优化效果明显一些, 这是由于 MARCO-MAM 算法相比 MARCO 算法多了一个中间模型, 在通过对中间模型的探索时可能会将目前还未被探索到的关键 MSS 所能额外剪枝的节点提前剪枝掉, 因此 ABC 策略对 MARCO-MAM 算法的优化效果不如对 MARCO 算法明显. 由于 ABC 策略是利用问题中关键 MSS 来加强剪枝的策略, 因此对于本身具有越多关键 MSS 的问题, 本文所提出的剪枝策略的加强剪枝效果就越好. 通过性质 4 可知, 在最优情况下, 使用 ABC 策略所增加的剪枝节点个数等于问题中关键 MSS 的个数. 考虑到其他 MSS 的剪枝可能会使 ABC 策略额外剪枝的节点被提前剪枝掉, 因此通过使用 ABC 策略而实际增加的剪枝节点个数会略小于问题中关键 MSS 的个数.

5 总　结

为提高 MUS 算法的枚举效率, 本文提出了一种加强剪枝的策略 ABC, 首先提出 cMSS 和 subMUS 概念, 并提出每个 MUS 必是 subMUS 的超集的性质, 进而在避免对 MCS 的碰集进行求解的情况下有效利用 MUS 和 MCS 互为碰集的特征, 有效避免求解碰集时的时间开销. 同时, 归纳总结出 4 条性质为 ABC 策略的正确性和有效性提供了理论依据, 当 subMUS 不可满足时则 subMUS 是唯一的 MUS, 算法将提前结束执行; 当 subMUS 可满足时, 则剪枝掉此节点, 从而有效避免对求解空间中的冗余空间进行搜索. 我们将该策略应用于已有的 MARCO 算法和 MARCO-MAM 算法, 通过在标准测试用例下的实验结果表明, 本文所提出的 ABC 策略对枚举效率具有一定的提升效果. 该策略与使用极大化模型的算法结合起来, 可以在算法的前期得到明显的提升效果, 并且对于子句集中含有越多关键 MSS 的实例, 该策略对算法的优化效果越佳. 后续的研究重点将围绕两个方向展开: 一方面, 寻找更优的策略以加速 MUS 的枚举; 另一方面, 将 MUS 枚举算法应用于具体的实际问题中提升其效率.

References:

[1] Sebastiani R, Vescovi M. Automated reasoning in modal and description logics via SAT encoding: The case study of K(m)/ALC-satisfiability. Journal of Artificial Intelligence Research, 2009, 35: 343–389. [doi: 10.1613/jair.2675]

[2] Nam GJ, Aloul F, Sakallah KA, Rutenbar RA. A comparative study of two Boolean formulations of FPGA detailed routing constraints. IEEE Trans. on Computers, 2004, 53(6): 688–696. [doi: 10.1109/TC.2004.1]

[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] Tian NY, Ouyang DT, Liu M, Zhang LM. A method of minimality-checking of diagnosis based on subset consistency detection. Journal of Computer Research and Development, 2019, 56(7): 1396−1407 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.2019.20180192]

[5] Zhou HS, Ouyang DT, Zhao XF, Zhang LM. Two compacted models for efficient model-based diagnosis. In: Proc. of the 2022 AAAI Conf. on Artificial Intelligence. AAAI, 2022. 3885–3893. [doi: 10.1609/aaai.v36i4.20304]

[6] Shazli SZ, Tahoori MB. Using Boolean satisfiability for computing soft error rates in early design stages. Microelectronics Reliability, 2010, 50(1): 149–159. [doi: 10.1016/j.microrel.2009.08.006]

[7] Lei ZD, Cai SW. Solving set cover and dominating set via maximum satisfiability. In: Proc. of the 2020 AAAI Conf. on Artificial Intelligence. New York: AAAI, 2020. 1569–1576. [doi: 10.1609/aaai.v34i02.5517]

[8] Cai SW, Zhang XD. Pure MaxSAT and its applications to combinatorial optimization via linear local search. In: Proc. of the 26th Int'l Conf. on Principles and Practice of Constraint Programming. Louvain-la-Neuve: Springer, 2020. 90–106. [doi: 10.1007/978-3-030-58475-7_6]

[9] Cai SW, Li YJ, Hou WY, Wang HR. Towards faster local search for minimum weight vertex cover on massive graphs. Information Sciences, 2019, 471: 64–79. [doi: 10.1016/j.ins.2018.08.052]

[10] Brown C E. Reducing higher-order theorem proving to a sequence of SAT problems. Journal of Automated Reasoning, 2013, 51(1): 57–77. [doi: 10.1007/s10817-013-9283-8]

[11] Arif MF, Mencía C, Marques-Silva J. Efficient MUS enumeration of Horn formulae with applications to axiom pinpointing. In: Proc. of the 18th Int'l Conf. on Theory and Applications of Satisfiability Testing. Austin: Springer, 2015. 324–342. [doi: 10.1007/978-3-319-24318-4_24]

[12] Xiao GH, Ma Y. Inconsistency measurement based on variables in minimal unsatisfiable subsets. In: Proc. of the 20th European Conf. on Artificial Intelligence. Montpellier: IOS Press, 2012. 864–869. [doi: 10.3233/978-1-61499-098-7-864]

[13] Cohen O, Gordon M, Lifshits M, Nadel A, Ryvchin V. Designers work less with quality formal equivalence checking. In: Proc. of the 2010 Design and Verification Conf. and Exhibition. 2010.

[14] Chen H, Marques-Silva J. Improvements to satisfiability-based Boolean function Bi-decomposition. In: Proc. of the 19th IFIP WG 10.5 on the Advanced Research for Systems on Chip. Hong Kong: Springer, 2011. 52–72. [doi: 10.1007/978-3-642-32770-4_4]

[15] Lee RR, Jiang JHR, Hung WL. Bi-decomposing large Boolean functions via interpolation and satisfiability solving. In: Proc. of the 45th Annual Design Automation Conf. Anaheim: ACM, 2008. 636–641. [doi: 10.1145/1391469.1391634]

[16] Han B, Lee SJ. Deriving minimal conflict sets by CS-trees with mark set in diagnosis from first principles. IEEE Trans. on Systems, Man, and Cybernetics, Part B (Cybernetics), 1999, 29(2): 281–286. [doi: 10.1109/3477.752801]

[17] Stern RT, Kalech M, Feldman A, Provan GM. Exploring the duality in conflict-directed model-based diagnosis. In: Proc. of the 26th AAAI Conf. on Artificial Intelligence. Toronto: AAAI Press, 2012. 828–834.

[18] Bailey J, Stuckey PJ. Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In: Proc. of the 7th Int'l Symp. on Practical Aspects of Declarative Languages. Long Beach: Springer, 2005. 174–186. [doi: 10.1007/978-3-540-30557-6_14]

[19] Liffiton MH, Sakallah KA. Algorithms for computing minimal unsatisfiable subsets of constraints. Journal of Automated Reasoning, 2008, 40(1): 1–33. [doi: 10.1007/s10817-007-9084-z]

[20] Previti A, Marques-Silva J. Partial MUS enumeration. In: Proc. of the 27th AAAI Conf. on Artificial Intelligence. Bellevue: AAAI Press, 2013. 818–825.

[21] Liffiton MH, Malik A. Enumerating infeasibility: Finding multiple MUSes quickly. In: Proc. of the 10th Int'l Conf. on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Yorktown Heights: Springer, 2013, 160–175. [doi: 10.1007/978-3-642-38171-3_11]

[22] Liffiton M H, Previti A, Malik A, Marques-Silva J. Fast, flexible MUS enumeration. Constraints, 2016, 21(2): 223–250. [doi: 10.1007/s10601-015-9183-0]

[23] Zhao WT, Liffiton MH. Parallelizing partial MUS enumeration. In: Proc. of the 28th IEEE Int'l Conf. on Tools with Artificial Intelligence (ICTAI). San Jose: IEEE, 2016. 464–471. [doi: 10.1109/ICTAI.2016.0077]

[24] Ouyang DT, Gao H, Tian NY, Liu M, Zhang LM. MUS enumeration based on double-model. Journal of Computer Research and Development, 2019, 56(12): 2623–2631 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.2019.20180852]

[25] Gao H. Diagram and tree structure-based MUS solution [MS. Thesis]. Changchun: Jilin University, 2020 (in Chinese with English abstract). [doi: 10.27162/d.cnki.gjlin.2020.004377]

[26] Bendik J, Benes N, Cerna I, Barnat J. Tunable online MUS/MSS enumeration. In: Proc. of the 36th IARCS Annual Conf. on Foundations of Software Technology and Theoretical Computer Science. Chennai: FSTTCS, 2016. 50: 1–50: 13. [doi: 10.4230/LIPIcs.FSTTCS.2016.50]

[27] Bendík J, Černá I, Beneš N. Recursive online enumeration of all minimal unsatisfiable subsets. In: Proc. of the 16th Int'l Symp. on Automated Technology for Verification and Analysis. Los Angeles: Springer, 2018. 143–159. [doi: 10.1007/978-3-030-01090-4_9]

[28] Luo J, Liu SF. Accelerating MUS enumeration by inconsistency graph partitioning. Science China Information Sciences, 2019, 62(11): 212104. [doi: 10.1007/s11432-019-9881-0]

[29] Bendík J, Černá I. Replication-guided enumeration of minimal unsatisfiable subsets. In: Proc. of the 26th Int'l Conf. on Principles and Practice of Constraint Programming. Louvain-la-Neuve: Springer, 2020. 37–54. [doi: 10.1007/978-3-030-58475-7_3]

[30] Narodytska N, Bjørner N, Marinescu MC, Sagiv M. Core-guided minimal correction set and core enumeration. In: Proc. of the 27th Int'l Joint Conf. on Artificial Intelligence. Stockholm: AAAI Press, 2018. 1353–1361.

[31] Bacchus F, Katsirelos G. Finding a collection of MUSes incrementally. In: Proc. of the 13th Int'l Conf. on Integration of AI and OR Techniques in Constraint Programming. Banff: Springer, 2016. 35–44. [doi: 10.1007/978-3-319-33954-2_3]

[32] Bendík J, Černá I. MUST: Minimal unsatisfiable subsets enumeration tool. In: Proc. of the 26th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Dublin: Springer, 2020. 135–152. [doi: 10.1007/978-3-030-45190-5_8]

[33] Eén N, Sörensson N. An extensible SAT-solver. In: Proc. of the 6th Int'l Conf. on Theory and Applications of Satisfiability Testing. Santa Margherita Ligure: Springer, 2004. 502–518. [doi: 10.1007/978-3-540-24605-3_37]

[34] MUS track of the 2011 sat competition. 2022. http://www.cril.univ-artois.fr/SAT11/

附中文参考文献:

[4] 田乃予, 欧阳丹彤, 刘梦, 张立明. 基于子集一致性检测的诊断解极小性判定方法. 计算机研究与发展, 2019, 56(7): 1396–1407. [doi: 10.7544/issn1000-1239.2019.20180192]

[24] 欧阳丹彤, 高菡, 田乃予, 刘梦, 张立明. 基于双模型的MUS求解方法. 计算机研究与发展, 2019, 56(12): 2623–2631. [doi: 10.7544/issn1000-1239.2019.20180852]

[25] 高菡. 基于图及树结构的极小不可满足集求解方法 [硕士学位论文]. 长春: 吉林大学, 2020. [doi: 10.27162/d.cnki.gjlin.2020.004377]

蒋璐宇(1998－), 女, 博士生, 主要研究领域为 SAT 问题, MUS 问题.

董博文(1998－), 男, 硕士, 主要研究领域为 SAT 问题, MUS 问题.

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

张立明(1980－), 男, 博士, CCF 高级会员, 主要研究领域为 SAT 问题, 基于模型的诊断.