软件学报 ISSN 1000-9825, CODEN RUXUEW
2026,37(4):1634−1649 [doi: 10.13328/j.cnki.jos.007509] [CSTR: 32375.14.jos.007509]

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

CDCL 算法的冷重启技术*

张昕荻[1,2], 陈志翰[1,2], 蔡少伟[1,2]

[1](基础软件与系统重点实验室 (中国科学院 软件研究所), 北京 100190)

[2](中国科学院大学 计算机科学与技术学院, 北京 100049)

通信作者: 蔡少伟, E-mail: caisw@ios.ac.cn

摘　要: SAT 求解的 CDCL 算法被广泛应用于软硬件验证领域, 重启策略是其中的核心组件之一. 目前, 主流的 CDCL 求解器采用了“热重启”技术, 保留了变元序、赋值倾向、学习子句等主要搜索信息, 且重启频率极高. 热重启技术会使 CDCL 重启之后更倾向于搜索重启前的搜索空间, 有可能会长期陷于一个不利的局部区域, 缺乏探索性. 首先对现有的 CDCL 算法进行测试, 证实了在不同的初始搜索设置下, 主流 CDCL 求解器的求解时间有巨大的扰动. 为了利用上述观察, 提出一种遗忘搜索信息的“冷重启”技术, 即阶段性的遗忘变元序、赋值倾向、学习子句, 实验证明了该技术可以有效地提高主流 CDCL 算法的性能. 同时, 也进一步拓展了其并行版本, 每个线程探索不同的区域, 提高了并行算法的性能. 此外, 冷重启技术主要改进了串并行求解器可满足实例的求解能力, 为设计可满足导向的 SAT 求解器提供了新的改进思路. 通过引入并行冷重启技术, PaKis 求解器可满足性实例的 PAR2 打分平均改进 41.81%. 基于相关技术设计的并行 SAT 求解器 ParKissat-RS 以领先亚军 24% 的大幅领先优势取得国内首个国际 SAT 竞赛并行组冠军.

关键词: 可满足性问题; 冷重启; 信息遗忘

中图法分类号: TP301

中文引用格式: 张昕荻, 陈志翰, 蔡少伟. CDCL算法的冷重启技术. 软件学报, 2026, 37(4): 1634–1649. http://www.jos.org.cn/1000-9825/7509.htm

英文引用格式: Zhang XD, Chen ZH, Cai SW. Cold Restart Technique for CDCL Algorithms. Ruan Jian Xue Bao/Journal of Software, 2026, 37(4): 1634–1649 (in Chinese). http://www.jos.org.cn/1000-9825/7509.htm

Cold Restart Technique for CDCL Algorithms

ZHANG Xin-Di[1,2], CHEN Zhi-Han[1,2], CAI Shao-Wei[1,2]

[1](Key Laboratory of Systems Software (Institute of Software, Chinese Academy of Sciences), Beijing 100190, China)

[2](School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: The CDCL algorithm for SAT solving is widely applied in the field of hardware and software verification, with restart being one of its core components. Currently, mainstream CDCL solvers often employ the “warm restart” technique, which retains key search information such as variable order, assignment preferences, and learnt clauses, and has a very high restart frequency. The warm restart technique tends to make CDCL solvers more inclined to visit the search space that is explored before restarts, which may lead to being trapped in an unfavorable local search space for a long time, lacking exploration of other regions. This study first tests the existing CDCL algorithms and confirms that under different initial search settings, the runtime for mainstream CDCL solvers exhibits significant fluctuations. To leverage this observation, the proposed “cold restart” technique forgets search information, specifically by periodically forgetting variable order, assignment preferences, and learnt clauses. Experimental results demonstrate that this technique can effectively improve mainstream CDCL algorithms. In addition, this study further extends its parallel version, where each thread explores different search spaces, enhancing the performance of the parallel algorithm. Moreover, the cold restart technique primarily improves the

* 基金项目: 中国科学院战略性先导科技专项 (前瞻战略科技先导专项) (XDA0320000, XDA0320300)
收稿时间: 2024-09-12; 修改时间: 2025-05-28; 采用时间: 2025-08-12; jos 在线出版时间: 2025-09-02
CNKI 网络首发时间: 2025-11-27

performance of sequential and parallel solvers on satisfiable instances, providing new insights for designing satisfiable-oriented solvers. Specifically, the proposed parallel cold restart technique improves the PAR2 score of PaKis on satisfiable instances by 41.81% on average. The parallel SAT solver named ParKissat-RS, which integrates the proposed ideas, wins the parallel track of the SAT competition with a significant margin of 24% over the runner-up.

Key words: satisfiability problem (SAT); cold restart; information forgetting

命题逻辑可满足性问题 (satisfiability problem, SAT) 是判断给定的命题逻辑公式可满足性的判定问题. 作为第 1 个被证明的 NP-完全问题, SAT 是计算机科学和数理逻辑的基础问题, 其重要性也被包括 Knuth 在内的多位图灵奖得主高度评价[1]. 冲突驱动的子句学习技术 (conflict-driven clause learning, CDCL) 是目前求解 SAT 问题的最主要方法, 它是一种深度优先的系统搜索算法, 并因拓展了非时序回溯和子句学习两种剪枝技术而得名[2]. CDCL 在大量的学术[3]和工业问题中得到应用, 如电子设计自动化领域[4]、软硬件测试[5,6]、模型检查[7]、密码分析[8]、智能调度[9]等.

CDCL 维护了一棵二叉搜索树, 树上每一个节点会引出最多两个边, 代表对某一个布尔变元赋值为“真”或者“假”, 树上从根节点出发的任一路径代表着一组变元赋值, 一般当前访问的分支所代表的部分赋值会被存放于一个栈中[1]. 伴随着重启[10−13]、赋值启发式[14,15]、子句管理[16]、化简[17]、混合求解[18]等技术的不断进步, CDCL 求解器的性能相比早期版本, 得到了大幅改进[19].

重启策略是 CDCL 的核心组件之一, 对于 SAT 求解器的高效性贡献巨大. 1997 年, Gomes 等人[20,21]发现一些系统性的回溯算法, 在求解一些随机可满足实例时, 对搜索顺序引入一定的随机性, 会使求解时间发生巨大的扰动, 呈现出一种有趣的重尾分布现象. 因此, 重启技术被引入于系统搜索算法来防止求解器长时间陷入一个没有解的搜索空间[22]. 后来, 重启技术也被证明可以用来提升不可满足实例的求解能力, 相关的解释有两种: 一方面, 重启可以被用来压缩赋值栈的大小, 以改进决策变元的顺序[23]; 另一方面, 有证据表明重启可以帮助求解器得到质量更高的学习子句[24].

目前 CDCL 的重启策略聚焦于判断何时中断当前的搜索, 然后清空当前赋值后重新搜索[10]. 为此, 学者们提出了大量的重启技术. 其中一个比较著名的重启策略叫作 Luby 重启, 它会根据一组固定间隔序列进行重启, 属于静态重启算法[13]. 此外, Luby 重启策略在一类特殊的 Las Vegas 随机算法上, 已经被证明是最优的重启策略[25]. 另一方面, 动态重启算法则会利用求解过程中的信息来判断重启时机. 例如, Glucose-style 重启策略统计了最近学习子句的平均文字块距离 (literal block distance, LBD), 当其明显差于全局的平均 LBD 值时, 则选择重启[11]. Biere 等人[10]也基于该算法提出了一套基于指数加权平均 (EMA) 的重启策略. 目前主流的 CDCL 求解器通常实现了多种重启策略, 并定期切换[16,26]. 在主流的重启策略下, CDCL 求解器的重启颇为频繁——求解器有时会每秒重启数百次.

主流的重启技术可以被视作“热重启”, 因为它们只会在重启时撤销变元的赋值, 并保留搜索过程中的全部信息. 其中主要包含: 分支启发式中变元的打分 (变元序)、相位启发式中变元的期望赋值 (赋值倾向) 和用于局部剪枝的学习子句 (学习子句) 这 3 类重要的搜索信息.

一方面, 热重启的使用减少了之前计算开销的浪费, 但另一方面也会让 CDCL 在重启后快速回到重启前的搜索区域, 有可能会使搜索陷入一个不利的搜索方向, 缺乏对于其他搜索空间的探索性. 因此, 本文对如何合理地遗忘搜索信息来增加算法的探索性, 以进一步提升 CDCL 串并行算法的性能展开了讨论, 并给出了一系列有趣的结果.

首先, 本文通过实验表明, 虽然目前主流的 CDCL 算法执行了大量的重启, 但是在不同的搜索顺序下, 求解同一实例的时间变化巨大. 该现象的存在提示我们要对当前的“热重启”技术进行补充, 合理地利用不同搜索顺序下运行时间的差异性, 通过更加彻底的重启, 来增加算法的探索性, 以提升算法的整体求解能力.

因此, 本文对于“重启时是否需要遗忘掉一些搜索信息?”这一未探索的关键问题开展讨论, 提出“冷重启”的概念, 并研究 3 种不同的冷重启技术及其变种版本. 其中, “冷重启”指的是在重启之后会遗忘掉搜索信息的重启策略. 第 1 种冷重启策略会在重启后重置变元的打分为随机值, 被称为遗忘变元序 (记为 FO (forgetting order)); 第 2 种冷重启策略会在重启后重置变元的相位为随机相位, 被称为遗忘赋值倾向 (记为 FP (forgetting phase)); 第 3 种冷重启会在重启之后遗忘所有子句数据库中学习子句, 被称为遗忘子句 (记为 FC (forgetting clause)).

本文在 3 个代表性的 CDCL 求解器上实现了本文的冷重启技术. 3 个求解器分别为 CaDiCaL-watch-sat[27]、MapleLCMDistChronoBT-DL[28]和 kissat-MAB[29]. 实验结果表明, 结合冷重启和热重启可以帮助改进 CDCL 求解器, 甚至有时可以获得显著的提高. FO 和 FP 策略可以帮助同时改进 CDCL 求解器的可满足性求解能力和不可满足性求解能力, 可使基础求解器分别整体多求解出 6.5 和 5.3 个实例. FC 策略主要可以改进 CDCL 求解器的可满足性求解能力, 但会使不可满足性求解能力显著下降. 考虑了多种冷重启的复合策略在可满足实例上, 可平均多求解 7.3 个实例, 且最优策略几乎均为 FO+FC. 通过对于遗忘子句 LBD 加以约束, 只遗忘 LBD 大于给定阈值的学习子句, 我们发现, 当求解目标为提高可满足性实例的求解能力时, 我们应该将该阈值设置得尽可能小. 通过对 3 种技术的组合测试, 本文发现 FO 有最优的综合提高能力.

进一步, 本文在具有最优性能的串行冷重启技术 FO 上, 实现了其并行版本, 并集成于两个代表性的并行求解器 (P-mcomsps[30]和 PaKis[31]) 中, 实验表明并行 FO 技术可以帮助提高并行 CDCL 求解器的性能. 并行 FO 可以更好地利用 CDCL 在不同搜索顺序下的运行时间差异. 为了更深入探索并行冷重启技术的提升潜力, 本文基于 kissat-MAB 求解器和并行 FO 技术, 形成的并行求解器即可拥有令人吃惊的可满足性求解能力和加速比, 如 PaKis 求解器可满足性实例的 PAR2 打分平均改进了 41.81%. 此外, 基于串行 FC 冷重启相关的分析, 本文设计了一种简单的子句共享机制, 只在线程间共享 LBD≤2 的学习子句, 从而形成了一个可以在可满足性求解能力上超过主流求解器的算法. 本文提出的并行冷重启方法, 相比于主流的多样性并行方法, 有更好的拓展性. 在 64 线程的环境以内, 该方法的加速比效率持续提升.

本文围绕 CDCL 算法的冷重启技术提出了如下 3 个研究问题 (RQ).

RQ1: 对主流的热重启 CDCL 求解器的启发式进行扰动, 其运行时间是否存在剧烈的扰动?

RQ2: 重启之后的搜索信息是否应该被保留?

RQ3: 是否可以借助冷重启的互补优势, 设计一套高效的并行求解器?

本文第 1 节介绍相关基础知识. 第 2 节对现存基于热重启的 CDCL 算法的重尾现象进行实证研究. 第 3 节基于上述实验结论, 提出 3 种不同类型的冷重启技术, 并研究多种冷重启技术的混合冷重启技术以及特性. 第 4 节进一步提出并行冷重启技术并研究相关的信息交互机制. 最后总结全文.

1 基础知识

本节给出了可满足性问题的基础知识.

1.1 可满足性问题与 CDCL 框架

给定一组由布尔变元组成的集合 $V=\{V_1,V_2,\ldots,V_n\}$, 其中每一个变元只能被赋值为真 ($\top$) 或者假 ($\bot$). 变元 v 的正文字由变元本身表示 v, 负文字则由其否定表示 $\neg v$. 子句 (clause) 为文字的析取组成. 合取范式 (conjunctive norm form, CNF) 由子句的合取组成. 可满足性问题 (SAT) 为判断给定的命题逻辑公式可满足性的判定问题. SAT 求解器是求解 SAT 问题的工具, 一般输入为 CNF 公式.

目前主流的 SAT 求解器均基于 CDCL 框架[2]. CDCL 会迭代地执行布尔约束传播 (BCP) 推理, 并根据推理过程中的冲突归结出新的学习子句用于剪枝和回退. 若冲突出现在原始子句则返回不可满足. 若 BCP 无法推出矛盾则会进入分支组件, 选择一个未赋值变元并赋值, 然后再重复 BCP. 若推理无矛盾且所有变元均已赋值则返回可满足. CDCL 是一个回溯算法, 学习子句可以用于剪枝搜索空间, 变元选择顺序和赋值倾向则影响算法的分支搜索顺序. 三者对于 CDCL 算法的性能影响巨大[1].

1.2 可满足性问题求解技术

● 变元排序启发式. CDCL 的变元分支启发式会根据搜索历史信息, 优先选择近期最有可能产生冲突的未赋值变元[32]. SAT 求解器为每一个变元维护一个分数, 并采用堆或者优先队列进行排序[26]. 变元的打分依据主要为变元参与到冲突的频率和时间. 例如经典的 VSIDS 启发式[15,33]会优先选择经常参与最近高质量学习子句中的变元, LRB 启发式[14]会优先选择赋值期间参与冲突分析比率 (学习率) 比较高的变元, VMTF 启发式[34]会贪心地选择参与最近一次冲突的变元.

● 变元赋值倾向启发式. CDCL 的赋值倾向被称为一个变元的相位 (phase). 最常用的技术为相位存储 (phase saving) 技术[35], 即将变元赋值为最近一次的赋值, 若该变元从未赋值, 则赋值为默认值. 近期的 SAT 求解器进一步集成了相位重置 (rephase) 技术[18,26], 会定期将变元的相位重置为较深搜索分支下的部分赋值 (靶相位)[36], 局部搜索的高一致性赋值[18]等.

● 学习子句管理技术. CDCL 的学习子句会帮助剪枝搜索空间, 但是过多的子句会给推理查询带来负担. 目前主流的 SAT 求解器会定期地删除掉一部分质量差的学习子句. 代表性的技术有 Oh 等人[16]提出的 3 层子句管理技术. 该技术根据子句的 LBD 将学习子句分为 3 组, 第 1 组高质量子句不会被清除, 第 3 组质量差的学习子句会被定期减半, 中间一组的学习子句会根据使用频率决定是否需要移动到其他两组.

● 重启技术. 最顶级的 CDCL 求解器实现了多种重启技术, 并会在求解过程中定期在高频重启和低频重启之间切换, 但是即便是低频重启, 频率有时也可以达到每秒钟数百次. 重启时, 为了保证算法的完备性, 求解器只会清空变元赋值, 并保留所有学习子句, 变元序和赋值倾向. 为了提高重启的效率, 一些重启后赋同样值的变元不会清空赋值. 以上所有的技术均会使得重启前后的搜索空间高度相似, 不利于空间的探索. 本文称主流的重启技术为“热重启”, 代表技术有 Luby 重启[13], Glucose-style 重启[11]及其 EMA 的改进版本[10].

● 并行求解技术. 顶尖的并行 SAT 求解算法可以分为分治法和多样性算法. 分治法的代表技术为 cube-and-conquer 技术[37], 它会使用前瞻求解器把原始问题拆分为众多子问题并在不同的线程上分别求解这些子问题. 应用该技术取得了多个数学问题上的突破[38,39], 但是该技术存在严重的负载不均衡问题, 且在国际 SAT 比赛上的表现显著弱于基于多样性的方法. 另一方面, 基于多样性的方法[23,30,31,40]会在不同线程上采用不同的求解器或者参数, 共同求解同一个问题. 该方法利用了不同技术间的互补性, 并在线程间共享部分高质量的学习子句. 历年 SAT 比赛并行组的冠军多为基于该技术研制的求解器. 但是, 多样性方法的求解能力受最优参数限制, 且由于优质参数数量有限且需要提前对参数间互补性进行调研, 故其大规模拓展能力受限.

1.3 实验环境设置

本文所有的实验均在一台高性能的服务器上运行, 该服务器包含 2 块 AMD EPYC 7763 中央处理器, 主频为 2.45 GHz, 总计 128 个实际物理核心. 机器的内存为 1 TB, 操作系统为 Ubuntu 20.04 LTS (64 bit).

实验数据集合为 2020 年与 2021 年国际 SAT 竞赛的标准测试实例集合, 分别简称为 SC20 和 SC21. 每组数据集包含 400 个来自工业和学术背景的实际问题. 求解器的运行设置与国际 SAT 比赛保持一致, 即每个串行求解器或并行求解器的线程独占一个 CPU 物理核心, 每一个实例的运行截止时间为 5 000 s.

对于每一个求解器, 本文汇报其在某一实例集合下的可满足实例求解数目 (#SAT), 不可满足实例求解数目 (#UNS) 和全部求解数目 (#ALL = #SAT + #UNS). 求解器的打分采用平均 PAR2 打分机制, 即求解器求解所有实例的平均求解时间, 其中超时实例的求解时间算作截止时间的 2 倍. 在同一组实例集合下, 平均 PAR2 打分越小, 求解器性能越好. 由于官方没有提供具体可满足实例和不可满足实例, 本文汇总了本文以及 SAT 比赛上所有求解器的求解结果, 形成了对应的子实例集合, 用于分别计算可满足和不可满足实例的 PAR2 打分.

给定一个测试实例, 若一个并行求解器使用 t 线程求解的运行时间为 T_t, 其加速比为 T_1/T_t. 本文在统计一组实例上的平均加速比时, 只考虑了单线程和 t 线程均可以求解的实例, 并去除了两者可以在 1 s 内求解的实例, 这种过于简单的实例会由于操作系统调度开销导致加速比统计出现特别明显的误差. 另外, 为了考虑未求解实例, 本文也汇报了不同线程下求解器的 PAR2, 作为性能改进的参考.

本文的代码和所有实验结果开源于 GitHub 仓库 (https://github.com/shaowei-cai-group/cold-restart).

本文选取了 3 个系列的代表求解器作为基准串行求解器.

● MapleLCMDistChronoBT-DL V3.0 (简记为 Maple-DL)[28]: 该求解器是 Maple 系列求解器的一个改进版本, 该系列求解器由不同团队开发. 自从首个以 Maple 冠名的求解器 MapleCOMSPS 提出以来, 该系列求解器在国际 SAT 竞赛中取得冠军 4 次. Maple-DL 为该系列在 SAT 比赛中夺冠的最新版本.

● CaDiCaL-watch-sat (简记为 CaDiCaLWS)[27]: 该求解器基于 CaDiCaL 系列求解器开发, 拥有更好的性能. 获得了 2021 年和 2022 年 SAT 比赛 CaDiCaL Hack 的冠军.

● kissat-MAB[29]: 最近几年的 SAT 比赛冠军多基于 kissat 求解器[26]进行开发, 本文选择的求解器为 2021 年 SAT 比赛的冠军求解器, 基于该技术的求解器在后面几年的竞赛中也被证明有高效的性能及稳定性.

本文选取了两个具有代表性的并行求解器作为基准并行求解器. 由于本文提及的技术被集成于 2022 年之后的所有 SAT 比赛的冠军求解器之中, 因此, 本文没有选择 ParKissat-RS 系列求解器[41]或其改进版本作为基准求解器.

● PaKis[31]: 该求解器是第 1 个基于 kissat 研制的并行求解器. 它基于多样性技术研制, 其目标是探索具有最优可满足实例求解能力的互补参数.

● P-mcomsps[30]: PaInleSS 系列并行求解器曾经取得国际 SAT 比赛并行组的 4 次冠军, 代表了目前综合性能顶级的并行 SAT 求解器. 本文选取的 SAT 求解器为 2021 年夺冠的版本. 该求解器也是基于多样性并行技术.

2 热重启 CDCL 求解器运行时间的差异性

早期学者对 SAT 等组合优化问题开展研究时, 发现基于 DPLL 框架 (CDCL 的前身) 的回溯搜索算法存在一种很有意思的现象: 求解器的性能会随着部分启发式的轻微扰动, 求解时间发生巨大的变化, 并发现运行时间的分布满足重尾分布. 为了解决该问题, 2000 年左右, Gomes 等人[20−22]提出对 DPLL 求解器定期完全重启, 并在分支顺序上引入一定的扰动, 来消除该现象并提高求解器的运行效率.

随后, 为了不浪费之前的搜索过程, 学者们提出了各式各样的重启算法, 并默认保留所有的搜索信息. 其中主要保存的信息有变元序, 赋值倾向和学习子句. 我们称这类重启技术为“热重启”. 例如 2003 年研制的著名 CDCL 求解器 MiniSat 会在重启后保留上述信息[15].

随着热重启技术普及, CDCL 求解器的重启频率日益提高, 并衍生了很多加速重启的技术. 相关技术虽然提高了 CDCL 求解器的求解性能, 但是代价是会使得求解器对启发式特别敏感, 例如对变元排序和赋值倾向等启发式. 由于其变元搜索顺序和赋值倾向均无变化, 故集成了热重启技术的 CDCL 求解器在重启之后更倾向于进入重启前的搜索空间.

因此, 本节将深入探索以下问题.

RQ1: 对主流的热重启 CDCL 求解器的启发式进行扰动, 其运行时间是否存在剧烈的扰动?

如果 RQ1 成立, 则可以借助该现象, 引入彻底重启技术, 通过增加探索性, 来改进求解器的性能.

2.1 不同初始搜索顺序下的运行时间差异

变元搜索顺序是 CDCL 的核心启发式之一. 为了研究变元序对于运行时间的影响, 本文选择了 kissat-MAB[29]作为研究对象. 默认情况下, kissat-MAB 会按照内部变元标号顺序作为初始变元搜索顺序. 另外, 本文从 SC20 标准测试实例集合中, 选择出 kissat-MAB 可以在 5 000 s (同 SAT 比赛标准一致) 内求解的实例, 作为测试数据集. 其中, 本文称某实例在 kissat-MAB 默认配置下的求解时间为 T_{baseline}.

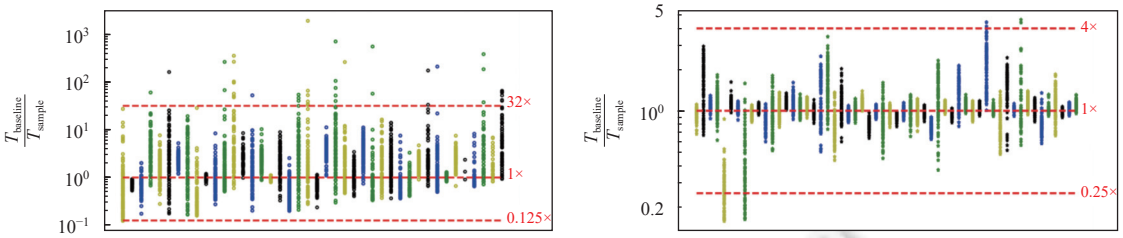
对于选中的每一个实例, 我们均随机生成了 100 组变元的随机序列 $\{o_1, o_2, \ldots, o_{100}\}$, 并作为 kissat-MAB 的初始变元顺序. 然后, 本文统计了在所有采样的变元序 o_i 下的运行时间, 记作 $T_{\text{sample}}(o_i)$. 然后, 对于 $T_{\text{sample}}(o_i)$ 非超时的序列 o_i, 本文计算了改变初始搜索顺序后, 运行时间的加速比 $T_{\text{baseline}}/T_{\text{sample}}(o_i)$, 该指标可以用于衡量 kissat-MAB 运行时间的扰动情况. 该值大于 1 表示有加速, 小于 1 表示有性能衰退.

图 1 给出了 kissat-MAB 在求解所选实例时, 100 个采样顺序的加速比. 图中每一列代表一个实例, 并按照 T_{baseline} 的大小进行排列. 图上的点表示某实例对应的加速比. 为了增加图片的可读性, 图中省略了 10 min 以内可以求解的简单实例的求解情况, 并将纵坐标的加速比数值以对数刻度呈现. 图 1(a) 和 (b) 分别展示了可满足实例和不可满足实例的求解情况.

根据图 1 的数据可以得到以下结论.

对于可满足性实例, CDCL 求解器的运行时间对于初始变元序的变化十分敏感. 在随机采样的变元序的帮助下, 33% 的可满足实例的运行时间, 出现了 32 倍以上的加速. 此外, 通过随机采样的变元初始序, kissat-MAB 可以求解 33 个在默认变元序下无法求解的可满足实例.

对于不可满足实例, CDCL 求解器运行时间的差异性相对较小. 从图 1(b) 可知, 通过扰动 kissat-MAB 的初始变元序, 不可满足实例的求解时间变化通常在 ±4 倍以内.



(a) 可满足实例 (b) 不可满足实例

图 1 不同初始变元搜索顺序下的运行时间扰动图

2.2 不同初始赋值倾向下的运行时间差异

变元赋值倾向 (相位) 是另一个对 CDCL 的搜索顺序有巨大影响的启发式. 我们同样也研究了随机相位对于求解器性能的影响. 实验按照与第 2.1 节相似的方法和设置执行, 并将实验的结果进行了总结. 其中初始相位的结果与图 1 中初始变元序的时间扰动图类似, 故不再过多赘述.

为了更加客观地呈现时间的变化情况, 表 1 中分别给出了在随机初始变元序和初始相位采样下, 时间变化的统计数据. 本文根据可满足性将实例进行了分类, 并统计了每一个实例在 100 次采样下的求解情况. 对于每一个实例, 我们统计了 100 次采样中基求解器可以求解, 但是扰动后无法求解的采样比例, 并将实例集合上的均值汇报于“失败”列. 类似地, 我们也统计了相较于基求解器来说, 加速比超过 2 倍、4 倍、10 倍和 32 倍采样的平均比例 (加速比率列), 以及求解速度变慢 0.5 倍和 0.254 倍的平均比率 (衰退比率列).

表 1 随机初始采样实验结果总结 (%)

实验	实例集合 (实例数)	失败	加速比率				衰退比率	
			2×	4×	10×	32×	0.5×	0.25×
随机初始变元序实验	SAT (151)	3.58	25.72	12.37	4.72	1.14	25.21	15.03
	UNSAT (121)	2.1	1.7	0.06	0.0	0.0	7.78	5.22
	ALL (272)	2.92	15.03	6.89	2.62	0.63	17.46	10.67
随机初始相位实验	SAT (151)	3.71	25.29	13.23	3.87	1.03	21.95	10.91
	UNSAT (121)	1.02	2.95	0.79	0.03	0.0	5.31	2.2
	ALL (272)	2.51	15.35	7.7	2.17	0.57	14.54	7.03

此外, 最近顶尖的 CDCL 求解器普遍采用了一种相位重置技术 (rephase)[18,36], 该技术会定期地更新变元的相位, 并优先将相位更新为 CDCL 搜索的较深的赋值倾向, 或者更新为局部搜索中取得的一致性较高的完全赋值. 尽管相位重置技术会引入一定概率的随机性, 但是采用了该技术的 CDCL 求解器的运行时间仍然存在明显的运行时间扰动.

相位重置技术中定期更改相位为高一致性赋值会大幅提高性能, 于是本文进一步测试是否高一致性的初始赋值会使得 100 次采样中出现求解性能更高的加速比. 我们尝试利用单元传播技术来构建高一致性的初始解, 该技术类似于 Knuth[42]的“warm-up”的想法, 也类似于“ReasonLS”系列求解器[18]中为局部搜索求解器赋值的技术. 本文同样为每一个实例, 随机产生 100 组高一致性赋值, 并统计了其求解的运行时间.

虽然该方法有更好的加速比稳定性, 但是, 该方法并不能产生比随机初始赋值更好的加速比. 该技术失败的原因有可能是高一致性赋值使得搜索方向有更高的趋同性, 缺乏了探索性.

2.3 热重启 CDCL 观察总结

本节对随机初始变元序和随机初始相位展开了细致的实验, 发现可满足实例上的运行时间扰动明显要高于不可满足实例上的运行时间扰动. 对于每一个实例, 我们计算了这 100 次运行时间的变异系数 (coefficient of variation,

CV), 即一组数据的标准差与该组数据平均数的比值. CV 值越高说明该组数据的离散程度越高. 我们进一步计算了实例集合上所有数据 CV 的均值.

对于不同的初始随机变元序的运行时间, 可满足实例集合上的平均 CV 值为 71.5%, 不可满足实例集合上的平均 CV 值为 27.2%. 对于不同的初始随机相位的运行时间, 可满足实例集合上的平均 CV 值为 71.8%, 不可满足实例集合上的平均 CV 值为 36.8%.

于是, 我们可以得到如下的观察结论, 并回答了 RQ1.

观察 1. 目前主流 CDCL 的性能 (运行时间) 对于初始变元序和初始赋值倾向十分敏感, 尤其是对于可满足实例.

对于可满足问题, 分支启发式直接决定了算法搜索的大方向; 对于不可满足问题, 启发式影响了冲突归结产生学习子句的顺序和质量, 影响了对搜索空间的剪枝效率. 热重启倾向于使用重启前的启发式信息, 使 CDCL 处于一个不利于求解的上下文时, 难以跳出当前搜索空间去探索其他区域, 因此会出现求解时间的大幅扰动. 此外, 对于可满足实例, SAT 求解器会在遇到第 1 个解时退出, 因此求解时间对搜索启发式更加敏感.

3 串行冷重启技术

很多组合优化问题的算法, 引入一定的随机噪声都会使得求解时间发生明显的变化[21,43]. 早期学者普遍采用重启技术增加算法的探索性, 以提升算法求解速度. 然而, 顶尖的 CDCL 求解器通常使用了一种高频热重启的策略, 保留了所有搜索信息. 即, 重启后保留用于剪枝的学习子句的同时, 保留基于冲突历史的变元排序打分[44], 保留变元回退前的赋值倾向[35]. 因此, 当前的 CDCL 求解器倾向于重启之后快速地回到之前的搜索区域.

通过第 2 节的实验, 现存的 CDCL 算法仍然存在运行时间严重的不稳定现象, 这极有可能是目前 CDCL 普遍采用热重启策略导致的. 为了减少求解器长时间卡在一个不利的搜索空间的概率, 本节提出了本文研究的核心问题, 并给出了 4 项研究观察.

RQ2: 重启之后的搜索信息是否应该被保留?

本文称主流 CDCL 普遍采用的重启策略为热重启, 并保留了 3 种主要的搜索信息: 变元的搜索顺序 (通过保留变元打分)、变元赋值倾向 (通过保留变元相位) 和学习子句. 对称地, 本文称在重启之后遗忘掉所有搜索信息的重启策略为“完全冷重启”, 且遗忘掉部分搜索信息的重启策略为“部分冷重启”技术. 为了方便区别于主流 CDCL 中的技术, 上述两种策略都统称为“冷重启”技术.

本节, 我们首先研究了 3 种单独的基础冷重启策略 (表 2), 然后进一步研究三者的组合冷重启技术 (见第 3.4 节).

表 2 遗忘变元序、遗忘赋值倾向和遗忘学习子句的冷重启结果汇总

数据集合	求解器	种类	#SAT	PAR2 (SAT)	#UNS	PAR2 (UNS)	#ALL	PAR2 (ALL)	p
SAT比赛2020年数据集合 (400个实例)	Maple-DL	默认	107	5 043.39	113	3 095.50	220	5 078.55	—
		+FO	113 (+6)	4 846.62	111 (−2)	3 175.93	224 (+4)	5 014.36	1E6
		+FP	114 (+7)	4 731.01	111 (−2)	3 179.28	225 (+5)	4 960.39	8E5
		+FC	114 (+7)	4 869.95	105 (−8)	3 620.68	219 (−1)	5 190.05	1E6
	CaDiCaLWS	默认	123	4 247.01	120	2 853.88	243	4 608.88	—
		+FO	126 (+6)	3 998.46	123 (+3)	2 704.77	249 (+6)	4 435.03	1E6
		+FP	130 (+7)	3 891.82	122 (+2)	2 796.81	252 (+9)	4 418.14	3E5
		+FC	125 (+2)	4 110.95	112 (−8)	3 298.22	237 (−5)	4 708.32	6E5
	kissat-MAB	默认	151	2 510.49	121	2 557.91	272	3 670.18	—
		+FO	159 (+8)	2 226.81	122 (+1)	2 515.19	281 (+9)	3 518.92	8E5
		+FP	156 (+5)	2 363.07	122 (+1)	2 591.20	278 (+6)	3 614.02	1E6
		+FC	158 (+7)	2 278.36	109 (−12)	3 421.98	267 (−5)	3 879.05	7E5
SAT比赛2021年数据集合 (400个实例)	Maple-DL	默认	115	3 458.14	143	3 076.04	258	4 163.22	—
		+FO	120 (+5)	3 270.75	146 (+3)	3 067.84	266 (+8)	4 084.84	1E6
		+FP	118 (+3)	3 279.54	144 (+1)	3 054.42	262 (+4)	4 082.84	1E6
		+FC	115	3 520.02	132 (−11)	3 637.65	247 (−11)	4 449.51	1E6

表 2 遗忘变元序、遗忘赋值倾向和遗忘学习子句的冷重启结果汇总 (续)

数据集合	求解器	种类	#SAT	PAR2 (SAT)	#UNS	PAR2 (UNS)	#ALL	PAR2 (ALL)	p
SAT比赛2021年数据集合 (400个实例)	CaDiCaLWS	默认	126	2667.20	144	2999.09	270	3811.46	—
		+FO	132 (+6)	2483.37	144	2994.74	276 (+6)	3735.90	8E5
		+FP	132 (+6)	2501.37	143 (−1)	3049.62	275 (+5)	3768.62	3E5
		+FC	130 (+4)	2557.67	138 (−6)	3251.08	268 (−2)	3884.82	6E5
	kissat-MAB	默认	142	1654.57	147	2714.55	289	3274.09	—
		+FO	143 (+1)	1624.87	152 (+5)	2555.97	295 (+6)	3188.47	4E5
		+FP	142	1655.81	150 (+3)	2634.91	292 (+3)	3237.56	1E6
		+FC	143 (+1)	1677.60	137 (−10)	3339.01	280 (−9)	3573.68	7E5

3 种单独的冷重启策略如下.

- FO 冷重启: 重启之后遗忘所有变元排序启发式中的变元打分.
- FP 冷重启: 重启之后遗忘所有相位保留启发式的存储内容.
- FC 冷重启: 重启之后遗忘所有子句管理数据库中的学习子句.

本文在实现的过程中, 并没有直接将热重启全部替换为冷重启. 预实验表明, 在如此高的重启频率下, 完全采用冷重启技术会使得求解器的性能大幅衰退. 因此, 本文选择在保留主流求解器中高频热重启的前提下, 混合低频冷重启技术以增加算法的探索性.

事实上, 冷热重启的结合技术与搜索算法中的一个热门话题在本质上有一定联系: 如何处理好算法利用 (exploitation) 和探索 (exploration) 两者间的均衡. 其中热重启与利用相关, 冷重启与探索相关.

本文旨在以最简单的形式将冷重启技术插入在热重启算法中, 因此我们在设计混合策略时, 主要考虑如何设置一个合理的冷重启间隔.

冷重启间隔设置: 本文选用了一种常见的线性递增间隔, 属于静态重启间隔的范畴. 令 r 表示算法自从上一次冷重启到现在遇到的所有冲突次数, n 为算法累计执行的冷重启次数. 则当 $r \geqslant p \times n$ 时, 算法就会执行一次冷重启. 其中 p 是唯一需要调节的参数.

本文选择冲突次数作为标准, 是因为 CDCL 求解器普遍使用冲突作为一种运行时间刻度的单位, 且热重启也同样采用该标准. 在统一标准下, 求解器执行冷重启的次数正相关于热重启的执行次数. 在本文的设置下, 一般求解器执行冷重启的次数不会超过 10 次, 例如 kissat-MAB+FO 执行冷重启的次数最多为 3 次; 相比之下, 热重启技术执行次数一般为冷重启执行次数的 3–4 个数量级.

参数调节: 本文提及的技术只包含一个超参 p. 对于每一组实例集合和求解器, 参数 p 的取值范围会从 10^5 – 10^6 之间进行简单的人工调参, 且参数的调节精度为 10^5, 即总计有 10 种选择. 对于每一个求解器及一组实例集合, 每一种冷重启技术均会选择出一个最优的参数, 并汇报于表 2 中的“p”列. 参数的调节目标是达到最优的综合性能, 且所有的参数均以科学计数法表示. 更精细的参数调整有潜力带来更好的实验结果, 例如提高一倍精度的情况下, kissat-MAB+FC 在 6.5E5 及 7.5E5 均可唯一求解出一个其他参数下无法求解的实例. 本文主要聚焦于规律的观察总结, 因此未设置复杂的预测调参技术来进行优化.

3.1 遗忘变元序的冷重启技术 (FO)

变元的分支打分启发式会为每一个变元分配一个浮点数分数或者时间戳, 每次遇到冲突时会对这两个值更新. 每一次 FO 冷重启被执行时, 若启发式维护了浮点打分, 则 FO 会将每一个变元的打分重置为 [0, 1] 之间的随机值, 然后将变元打分增加量设置为 1, 并更新维护变元排序的数据结构 (一般为堆); 若启发式维护了时间戳, 则 FO 会重置时间戳, 产生一组随机赋值顺序并按照上述顺序重新加入数据结构 (一般为优先队列)[26]. 上述两种操作既可以保证变元可以产生一组随机序列, 也可以使得随机扰动的影响被控制在一次冲突以内.

本文在 3 个基础求解器上实现了 FO 冷重启技术, 实验结果被汇总于表 2.

从结果中可以得知, 定期执行 FO 冷重启可以提升求解器的求解性能, 尤其是对于可满足实例. 具体地, 通过引入 FO, 3 个基求解器平均在 SC20 和 SC21 实例集合上, 求解数目分别平均提高 6.3 和 6.7 个实例. 对于可满足实例, 上述数据分别为 5.7 和 4.0 个. 可满足实例的求解性能提升主要归功于不同初始搜索顺序下巨大的运行时间扰动, 在求解时, 定期的切换搜索空间, 可以帮助求解器减少卡死在某不利的搜索空间的概率.

根据实验结果, 我们同样也发现 FO 的引入也有助于改进不可满足实例的求解能力, 这对于 kissat-MAB 求解器尤为明显. 一个合理的解释为, CDCL 的 UNSAT 证明可以看作启发式寻找冲突归结路径的技术, 而冷重启可以帮助 CDCL 找到一个更优的冲突归结路径[23].

观察 2. 遗忘变元序的冷重启技术可以帮助求解器同时提高可满足和不可满足实例的求解能力.

3.2 遗忘赋值倾向的冷重启技术 (FP)

变元赋值倾向和选择顺序都主要作用于算法的搜索方向. 本节的 FP 技术主要作用于相位保存[35]技术. 每次 FP 重启被执行时, 本文均会将 CDCL 变元的相位随机重置为 0 或 1.

本文采用了与第 3.1 节类似的实验设置, 实验结果也同样被汇报于表 2.

实验结果表明, FP 冷重启技术可以帮助 CDCL 提高综合求解能力. 借助 FP 冷重启, Maple-DL、CaDiCaLWS 和 kissat-MAB 平均在两组实例集合上, 各自平均多求解了 4.5、7.0 和 4.5 个实例. 同时, FP 的实验结果与 FO 的实验结果整体上类似, 算法的综合性能改进主要归功于 FP 在可满足实例上的提高.

观察 3. 遗忘赋值倾向的冷重启技术可以帮助求解器大幅提高可满足实例的求解能力, 小幅度提高不可满足实例的求解能力.

3.3 遗忘学习子句的冷重启技术 (FC)

与变元序和相位主要作用于搜索顺序上不同, 学习子句在 CDCL 求解器中主要的作用为帮助剪枝, 过多的学习子句会影响求解器的推理速度.

本文设计了一种简单的学习子句遗忘策略. 我们选中的 CDCL 求解器均为学习子句维护了一个数据库, 并设计了相关的管理算法. 每次执行 FC 冷重启时, 求解器会清空学习子句数据库中所有的学习子句.

实验结果同样被汇报于表 2 中, 从中可以得知, FC 冷重启总是会使得不可满足实例的求解性能衰退. 但是, FC 冷重启技术有时可以帮助求解器提升可满足实例的求解性能. 事实上, FC 可以帮助本文选择的所有 3 个求解器提高求解性能. 例如, 在 SC20 实例集合上, FC 冷重启可以帮助 Maple-DL 和 kissat-MAB 分别多求解 7 个实例. 这个结果让人十分吃惊. 主流的 CDCL 求解器维护了一个子句管理组件, 并会定期地删除一部分质量较差的学习子句, 来减少推理负担. 本文有关于 FC 冷重启的实验表明, 定期删除全部子句数据库的学习子句对于可满足实例有巨大的收益. 这从另一方面说明, 在设计一个可满足性问题导向的 CDCL 求解器时, 目前学习子句起到的作用有待考虑, 且子句管理组件需要进一步的改进.

本文从两个方面对学习子句的作用进行了更深入的研究, 并尝试解释 FC 对于可满足性实例有效的原因.

3.3.1 FC 冷重启中学习子句的 LBD 阈值研究

为了更好地理解学习子句在 CDCL 中的作用, 本节为 FC 添加了一组补充实验, 以探究不同 LBD 质量的学习子句对于求解效率的影响.

对于 FC 冷重启, 我们设置了一个遗忘学习子句的阈值 ρ, 并在冷重启时选择性地遗忘掉学习子句数据库中所有 LBD $>\rho$ 的学习子句. 在 3 层学习子句管理技术的工作中, Oh[16]认为 LBD >5 的学习子句在 CDCL 求解过程中被使用的概率和能起到的作用都极小. 因此, 我们沿用之前的经验, 分别将阈值 ρ 设置为 0, 1,…, 5, 并在冷重启时将高质量的学习子句保留下来. 其中, $\rho=0$ 表示的是默认版本的 FC.

本节选用 kissat-MAB[29]作为基求解器, 并选用 SC20 与 SC21 实例集进行评估. 冷重启的参数 p 始终保持与默认的 FC 版本一致, 即 $p=7\times10^5$. 实验结果被汇总于表 3.

根据实验结果, 可以得到如下的结论.

(1) 遗忘更多的学习子句通常意味着更好的可满足实例求解能力和更差的不可满足实例的求解能力.

(2) 保留 LBD ⩽ 3 的学习子句能帮助求解器取得最优的综合性能. 实际上, 很多主流的学习子句管理算法也会将 LBD = 3 作为学习子句分组的标准[16].

表 3 遗忘变元序、遗忘赋值倾向和遗忘学习子句的冷重启结果汇总

实例集合	遗忘子句	#SAT	PAR2 (SAT)	#UNS	PAR2 (UNS)	#ALL	PAR2 (ALL)
SAT比赛2020年标准测试实例集合 (400个实例)	LBD > 0	**158**	**2278.36**	109	3421.98	267	3879.05
	LBD > 1	155	2385.65	115	3082.32	270	2689.80
	LBD > 2	153	2428.49	120	2622.74	273	2513.30
	LBD > 3	154	2405.37	120	2578.99	274	**2481.17**
	LBD > 4	153	2447.50	**122**	**2494.85**	**275**	2668.17
	LBD > 5	153	2495.89	121	2542.93	274	2516.38
SAT比赛2021年标准测试实例集合 (400个实例)	LBD > 0	143	1677.60	137	3339.01	280	3573.68
	LBD > 1	**144**	**1626.29**	141	3072.62	285	2403.80
	LBD > 2	142	1708.98	142	2924.58	284	2362.45
	LBD > 3	143	1742.83	**151**	**2636.28**	**293**	**2223.13**
	LBD > 4	140	1748.33	149	2669.04	289	2243.28
	LBD > 5	138	1840.09	150	2643.33	288	2271.89

3.3.2 学习子句的使用效率研究

为了回答我们的猜测, 我们对学习子句的使用情况进行了研究. CDCL 算法中子句一般只有推理和参与冲突两种用途. 单元子句一般会直接用于固定一个变元, 不会加入子句管理数据库, 因此, 本节选择 LBD 介于 2–7 的所有学习子句, 并追踪它们参与推理和冲突的次数. 当一条子句在 BCP 中参与推出了一个变元的赋值, 则它被认为参与了一次推理; 当一条子句参与到一次冲突和新学习子句的产生, 则它被认为参与了一次冲突. 搜索过程中一条学习子句的 LBD 会经常发生变动, 为了方便数据统计, 一条学习子句的 LBD 被算作是该子句产生时的 LBD 数值.

本文统计了不同 LBD 的学习子句参与冲突和推理的次数, 并根据 LBD = 2 的数据进行标准化, 然后将数据汇总于表 4. 由表中的数据可知, 算法中学习子句的使用效率在“LBD = 2”和“LBD = 3”之间发生了巨大的衰退, 其他相邻列之间的衰退则明显较小.

表 4 不同 LBD 学习子句使用次数的标准化数据

类型	LBD = 2	LBD = 3	LBD = 4	LBD = 5	LBD = 6	LBD = 7
参与冲突	1.0	0.2641	0.1887	0.1381	0.1004	0.0670
参与推理	1.0	0.1899	0.1289	0.0833	0.0447	0.0215

观察 4. 遗忘学习子句的冷重启技术总是会使得不可满足实例求解性能变差, 但通常会 (不全是) 使得可满足实例求解性能提高. 另一方面, 尽可能多地保留学习子句有利于不可满足的证明, 但不利于可满足实例的证明, 这是由于低质量学习子句虽然可以用于剪枝搜索空间, 但其利用率较差.

3.4 遗忘多种类型信息的冷重启技术

第 3.1–3.3 节分别对 3 种单独的冷重启技术进行了讨论, 本节则进一步对复合冷重启开展研究, 即在重启时遗忘多种搜索信息的冷重启技术. 混合冷重启策略有如下的 4 种复合形式.

- FO+FP: 重启时, 求解器会随机打乱变元序, 并随机重置相位.
- FO+FC: 重启时, 求解器会随机打乱变元序, 并清空学习子句数据库.
- FP+FC: 重启时, 求解器会随机重置相位, 并清空学习子句数据库.
- FO+FP+FC: 重启时, 求解器会随机打乱变元序, 随机重置相位, 并清空学习子句数据库.

表 5 汇报了复合冷重启的实验结果, 给出了每一个求解器的最优复合配置和对应的求解数目变化情况. 在 SC20 和 SC21 实例集合上, 对于每一个求解器, 本文分别在 Δ_{SAT}、Δ_{UNS} 和 Δ_{ALL} 这 3 列中报告了其在 SAT、UNS 和 ALL 上最优的求解效果变化情况, 和对应的最优复合配置 (*C* 列). 表格中仅汇报求解数量最多的一种策略, 若有多种策略取得最优求解数目, 则汇报 PAR2 最小的复合配置.

表 5 复合策略冷重启技术的最优配置及参数

实例集合	求解器	C_{SAT}	Δ_{SAT}	p (SAT)	C_{UNS}	Δ_{UNS}	p (UNS)	C_{ALL}	Δ_{ALL}	p (ALL)
SAT比赛2020年标准测试实例集合 (400个实例)	Maple-DL	+FO+FC	+8	1E6	+FO+FP	−1	4E5	+FP	+5	8E5
	CaDiCaLWS	+FO+FC	+12	8E5	+FO	+3	1E6	+FP	+9	3E5
	kissat-MAB	+FO+FC	+10	4E5	+FO	+1	8E5	+FO	+9	8E5
SAT比赛2021年标准测试实例集合 (400个实例)	Maple-DL	+FO	+5	1E6	+FO	+3	1E6	+FO	+8	1E6
	CaDiCaLWS	+FO+FC	+6	1E6	+FO+FP	+1	1E6	+FO	+6	8E5
	kissat-MAB	+FO+FC	+3	4E5	+FO	+5	4E5	+FO	+6	4E5

实验结果表明, 复合冷重启策略可平均使得求解器多求解 7.3 个可满足实例, 5 个不可满足实例, 整体多求解 7.17 个实例. 且根据 6 组样本的统计信息, FO+FC 最有希望提高可满足实例的求解效率 (5/6), FO 最有希望提高不可满足实例的求解效率 (4/6) 和全部实例的求解效率 (4/6).

实验结果表明, 对于可满足和不可满足实例, 各自存在对应的最优配置, 总结如下.

观察 5. FO+FC 复合冷重启策略有最优的可满足实例求解能力; 单独的 FO 冷重启策略能达到最优的不可满足实例求解和整体求解能力.

3.5 细分实例求解性能分析

根据之前的实验结果, FO 冷重启有最优的综合性能, 这使我们对其在细分实例集合上的表现产生了兴趣. 本文将 SC20 和 SC21 上的实例按照来源做了详细的分类. 对于所有细分实例的实验结果可以于本文的 GitHub 仓库获得. 在此, 本文对实验数据进行了总结, 其中#k 表示某组细分实例集中的实例个数.

(1) 冷重启更适用于“hypertree 分解” (#14), “滑动瓷砖拼图” (#13), 一些如“最小 super-permutation 问题” (#13) 等旅行商问题和一些染色问题. FO 可以帮助基求解器在这些家族中至少多求解出 2 个实例.

(2) 另一方面, FO 在“原像攻击密码学问题”(#11) 用例上表现较差, 会少求解 2 个实例.

(3) 对于一些类型的实例来说, FO 的引入会使得基求解器产生一定的互补性, 例如 kissat-MAB 和 kissat-MAB +FO 在“电路乘法” (#13) 这组实例都可以求解 8 个实例, 但其中 4 个有求解能力的互补性. 这也启示我们, 利用不同冷重启变元顺序之间显著的互补性, 有潜力据此设计一个更进一步的算法. 实际上, 本文后续章节利用了该特点, 研究了并行冷重启方法.

4 并行冷重启技术

冷重启的引入有助于提升求解器的综合求解能力, 但是第 3.5 节中提到冷重启技术引入后, 求解器也会在某些实例上产生互补性或者产生退化现象. 这是因为冷重启会增加求解器的探索性, 但并不能保证重启之后进入的新搜索空间有更好的搜索性质. 例如, 冷重启可能会中断一些求解器时间占据较长的分支, 这些分支虽然难以找到可行解, 但实际上该分支已经是所有分支中距离真正解比较近的一个分支.

由于探索新分支所带来的风险性, 串行冷重启技术无法消除运行时间的变化. 实验表明, kissat-MAB+FO 的 CV 值与 kissat-MAB 的 CV 仅有 5.57% 的差距. 此外, 重尾现象也是 CNF 求解时间难以预测的重要因素, 该问题已困扰 SAT 领域数十年. 既然难以克服, 则可从另一角度考虑, 利用运行时差异来帮助更高效的求解.

除了混合冷热重启之外, 另外一种方法为, 利用不同参数下冷重启的互补优势来设计一个并行冷重启策略. 本节选择综合性能最好的 FO 冷重启技术作为研究案例, 介绍了并行冷重启技术, 并用于改进主流的求解器性能. 此外, FC 冷重启的并行版本本质上等同于子句共享, 本节也对相关技术展开了分析.

RQ3: 是否可以借助冷重启的互补优势, 设计一套高效的并行求解器?

本节主要探讨以上的研究问题, 并给出了一项观察结论.

按照搜索信息对于 CDCL 的作用, 我们可以将第 3 节的冷重启技术分为两类. 第 1 类是可以为搜索引入多样性的 FO 和 FP 冷重启技术, 另一类是负责搜索空间剪枝的 FC 冷重启技术. 本节分别对两类技术展开了其并行版本的讨论.

首先, 第 4.1 节从 FO、FP 以及 FO+FP 的并行拓展版本中, 我们选择 FO 作为代表, 设计了并行 FO 冷重启技术展开讨论. 并行 FO 冷重启技术的实现方法与串行 FO 冷重启的技术实现类似, 区别在于, 并行 FO 在不同线程上使用了不同的随机种子. 该技术可以视为多样性并行求解器中的一种多样性引入策略.

并行 FO 冷重启技术的实现: 对于 n 线程的并行 SAT 求解器, 每一个线程的随机种子依次设置为 0, 1,…, n–1, 并同时执行串行 FO 技术. 若其中任一线程完成求解, 则并行算法立刻停止其他线程的求解并返回该求解结果.

借助类似的方法, 我们可以简单地实现并行 FP 冷重启和并行 FO+FP 复合冷重启技术. 本文选择单独展开对 FO 技术讨论的原因有两个. 一方面 FO 拥有最佳的串行冷重启综合性; 另一方面, 拓展实验表明, 通过 3 种技术均可以得到同样的实验结论, 且并行 FP 和并行 FO+FP 的实验效果没有并行 FO 的实验提升明显.

然后, 从本质上讲, 将完全 FC 冷重启技术并行化, 类似于线程间不共享学习子句, 即无子句共享技术, 于是, 本文的第 4.2 节根据串行的实验结果展开了并行子句共享的讨论.

4.1 并行 FO 实验评估与加速比分析

本文在 PaKis[31]和 P-mcomsps[30]两个具有代表性的并行 SAT 求解器上, 实现了并行 FO 冷重启技术, 形成了 PaKis+FO(p) 和 P-mcomsps+FO(p) 两个包含并行 FO 技术的并行求解器. 本文用尾缀 (p) 表示该求解器为并行求解器, 默认在 32 线程下运行. 为了保证对比的公平性, 本文也将 PaKis 的基础求解器由 kissat-sc20[26]替换为性能更好的 kissat-MAB[29], 实验表明, 两个版本的性能相当.

表 6 给出了相关的实验结果, 其中 PaKis+FO(p) 和 P-mcomsps+FO(p) 的冷重启参数分别设置为 6E5 和 4E5. 实验结果表明, 并行 FO 技术可以帮助并行算法提高综合求解能力, 并显著提升了可满足性实例的求解性能. 具体地, 在可满足实例集合上, P-mcomsps+FO(p) 的 PAR2 相比 P-mcomsps(p) 的打分改进了 13.7%; PaKis+FO(p) 的 PAR2 打分相较于 PaKis(p) 的打分改进了 41.81%.

表 6 并行 FO 冷重启策略在两个主流并行 SAT 求解器上的实验结果

实例集合	求解器	#SAT	PAR2 (SAT)	#UNS	PAR2 (UNS)	#ALL	PAR2 (ALL)
SAT比赛2020年标准测试实例集合 (400个实例)	P-mcomsps(p)	158	2 169.01	140	843.34	298	2 872.74
	P-mcomsps+FO(p)	160 (+2)	2 112.08	141 (+1)	813.09	301 (+3)	2 834.36
	PaKis(p)	176	1 005.26	130	1 801.21	306	2 671.46
	PaKis+FO(p)	181 (+5)	800.08	130	1 776.44	311 (+5)	2 564.32
SAT比赛2021年标准测试实例集合 (400个实例)	P-mcomsps(p)	143	1 332.89	179	725.23	322	2 220.39
	P-mcomsps+FO(p)	149 (+6)	1 002.65	178 (–1)	778.81	327 (+5)	2 113.21
	PaKis(p)	155	472.05	164	1 705.67	319	2 331.96
	PaKis+FO(p)	158 (+3)	173.68	166 (+2)	1 678.34	324 (+5)	2 249.03

加速比是衡量一个并行技术最常见的指标. 为了更好地测试 FO 的性能, 排除其他原因的影响, 本文基于 kissat-MAB 和并行 FO 技术实现了一个多样性并行求解器, 记作 kissat-MAB+FO(p). 本文计算了该求解器在 SC20 实例集合上, 不同核心数目 (t) 下, 求解器可满足实例, 不可满足实例以及全部实例的求解数量, 以及对应的 PAR2 打分和平均加速比.

表 7 中给出了 kissat-MAB+FO(p) 的不同版本在 1–64 线程下的求解情况. 从表 7 中数据可知, 仅利用并行 FO 技术带来的多样性, 即可获得一个不错的可满足实例求解能力, 相比之下, 不可满足实例的求解能力提升不明显. 基于并行 FO 技术的求解器求解结果更加稳定. 本文选用了 SC21 实例集 (包含 400 个数据), 使用 32 线程的 kissat-MAB+FO(p), 以不同时间种子运行 5 次, PAR2 均值的最高最低分差距为 1.6%, 求解个数差距最多为 2.

从表 7 中数据可知, 该技术的并行拓展性较强, 随着核心数量的增多, 实例求解数量和加速比不断提高, PAR2 打分不断降低. 仅依赖 FO(p) 技术, 64 线程版本相比单核心版本, 求解数量整体提升 13.2%, PAR2 打分改进 27%, 平均加速比达到 16.0 倍; 若考虑简单子句共享技术 (第 4.2 节详细讨论), 64 线程版本相比单核心版本, 求解数量整体提升 18.4%, PAR2 打分改进 40.8%, 加速比达 24.8 倍.

并行 FO 与其他多样性生成方法的对比讨论: 正如前文介绍, 并行冷重启技术可以在客观上被看作一种基于多样性的并行方法. 前人的多样性生成方法主要是采用互补的不同求解器或者同一求解器的不同互补性参数配

置[23,30,31,40]. 与之前的多样性方法类似, 并行 FO 技术也会通过在不同线程上沿不同的求解方向搜索, 据此产生具有互补性的配置.

表 7 并行 FO 冷重启的加速比分析

kissat-MAB+FO(p)版本	t	#SAT	PAR2 (SAT)	#UNS	PAR2 (UNS)	#ALL	PAR2 (ALL)	平均加速比		
								SAT	UNS	ALL
kissat-MAB+FO(p)	1	151	2510	121	2558	272	3670	1.0	1.0	1.0
	2	161	1986	123	2440	284	3376	4.2	1.13	2.8
	4	168	1477	123	2404	291	3120	12.5	1.16	7.5
	8	176	1190	125	2317	301	2951	10.0	1.18	6.1
	16	177	1024	124	2320	301	2872	15.0	1.22	8.9
	32	182	766	125	2209	307	2707	26.8	1.32	15.5
	64	183	667	125	2231	308	2668	27.5	1.35	16.0
kissat-MAB+FO(p)+子句共享 (LBD ⩽ 2)	1	151	2510	121	2558	272	3670	1.0	1.0	1.0
	2	162	1958	126	2159	288	3259	3.5	1.3	2.5
	4	167	1618	127	1984	294	3032	7.5	1.9	5.0
	8	171	1329	129	1723	300	2797	9.4	2.9	6.5
	16	178	924	133	1436	311	2498	17.0	4.6	11.5
	32	180	770	131	1493	311	2445	22.1	6.6	15.4
	64	184	583	133	1354	317	2304	29.0	9.4	20.5
kissat-MAB+FO(p)+子句共享 (LBD ⩽ 3)	1	151	2510	121	2558	272	3670	1.0	1.0	1.0
	2	163	1859	128	1987	291	3148	2.6	1.6	2.2
	4	167	1603	130	1785	297	2951	5.8	2.2	4.2
	8	173	1197	131	1556	304	2672	10.2	3.4	7.2
	16	178	911	134	1316	312	2447	20.7	5.2	13.9
	32	177	948	135	1169	312	2410	20.0	8.0	14.7
	64	182	662	140	891	322	2171	35.5	11.0	24.8

(1) 并行 FO 多样性产生方法是一种轻量级, 易于实现, 且可以支持所有 CDCL 算法的策略. 相比之下, 基于参数的多样性策略需要人工选择不同的互补参数, 且不同求解器的参数种类和数量都不同.

(2) 并行 FO 多样性策略的产生不会影响求解器的综合性能. 根据前人的观点, 随机扰乱测试实例会小幅扰动求解器的性能, 但不会影响求解器的排名[45]. 相比之下, 基于参数的方法, 有可能会使得部分求解器的性能变弱, 不利于求解.

(3) 并行 FO 有良好的拓展性和动态适应性. 基于该方法可以很快地将一个串行 CDCL 求解器拓展到任意核心数目的并行求解器上. 当使用的核心数目场景发生变化时, 无需修改代码. 且基于本文方法的求解器, 加速比在到达 64 核心时, 仍然在持续增高, 有进一步增长的潜力. 相比之下, 面对不同线程数, 之前的方法需要人工选择参数并调参, 甚至会面临参数数量不足以分配给所有线程的情形.

4.2 并行 FO 的共享子句选择

目前, 基于多样性的并行 SAT 求解器通常采用子句共享技术在线程之间交互学习子句, 用于剪枝高维度搜索空间, 以减少不同的线程重复搜索同一空间的概率. 此外, 在多个线程之间“共享学习到的子句”的思想类似于当前热重启之间“保留高质量的学习子句”的想法. 即清空所有子句的 FC 技术的冷重启技术类似于不采用子句共享技术; 保留质量在 LBD $\leqslant \rho$ 学习子句的 FC 冷重启技术类似于线程间共享一部分 LBD $\leqslant \rho$ 的学习子句.

这引起了我们对研究“FC 中学习子句的阈值规律是否也适用于并行子句共享”的兴趣.

根据表 3 的结果, LBD $\leqslant 3$ 的学习子句具有最高的综合性能收益. 本文在 PaInleSS 框架[30]的帮助下, 分别在 kissat-MAB+FO(p) 的基础上, 实现了线程间简单的子句共享技术, 分别仅简单共享 LBD $\leqslant 2$ 和 LBD $\leqslant 3$ 学习子句的子句.

子句共享技术的实现: 每一个线程均设置一个读入共享内存和一个输出共享内存. 每个线程会收集 LBD $\leqslant \rho$ 的学习子句, 存储于该线程的输出共享内存. 每隔 0.75 s, 每个线程的输出共享内存会将收集到的学习子句分发给

其他线程的输入共享内存中, 并最多一次共享 1 500 个文字. 每次线程内部重启的时候, 算法会将输入共享内存中的学习子句读入内部学习子句数据库和相关数据结构中.

表 7 汇总了不同核心下的求解性能和对应的加速比, 从中, 我们可以得到如下的实验结果.

(1) 学习子句在串行 FC 冷重启技术中与在并行子句共享中发挥的作用类似.

(2) 子句共享可以进一步提高不可满足实例的求解性能, 但是对于可满足实例几乎没有影响, 甚至在某些实例上会产生阻碍.

(3) P-mcomsps 求解器中实现了一种动态阈值的学习子句共享技术, 会根据文字共享数量对阈值进行动态更新[30]. 根据实验结果, 我们发现该版本与 LBD ⩽ 2 的版本具有类似的性能, 并可以得到类似的结论.

整体来看, 基于并行 FO 和简单的子句共享策略足以开发一个以可满足性实例求解导向的高性能并行求解器. 基于本节的观察结论, 可以得到如下的观察, 并回答了 RQ3.

观察 6. FO 冷重启技术有助于帮助并行 SAT 求解器更高效、便捷地产生互补配置, 这显著地提升了可满足实例的求解性能, 但是对于不可满足实例的改进较弱. 子句共享技术的引入显著地提高了不可满足实例的求解性能, 但对于可满足实例求解性能的改进有限.

5 相关工作

Gomes 等人[21,22]最早发现了组合优化回溯算法运行时间的重尾分布现象, 并引入了随机性和重启技术. 1998 年, 他们为基于 DPLL 的 SAT 求解器引入了一定的随机性平均打破机制, 并引入了线性增长间隔的重启策略. CDCL 求解器 Chaff[33]提出了著名的 VSIDS 分治策略, 并为赋值决策过程添加了一定的瞬时随机性, 但是它会在重启时维持当前的变元选择顺序. 与上述工作不同的是, 本文的冷重启策略会彻底将变元选择顺序洗牌.

自从 CDCL 算法提出以来, 学者们对 GRASP[2]等求解器展开了研究, 发现在重启之后记录学习子句可以减少一定的运行时间[43]. 这种做法迅速地成为 CDCL 求解器的标准配置, 尽管后续衍生了多种学习子句管理启发式, 但是据我们所知, 没有一个串行 CDCL 求解器会和本文的 FC 冷重启一样, 即在重启的过程中遗忘所有学习子句数据库中的学习子句.

除此之外, 大部分有关于重启的技术主要聚焦于何时进行重启[10−13,15,16,44]. 但是所有相关技术均为热重启技术. 与本文最为相关的技术为目前流行的相位重置技术, 其中会以一部分概率将相位重置为随机值, 类似于本文的 FP 冷重启技术. 然而, 本文系统地提出了 3 种冷重启技术及相关复合冷重启技术, 详细针对该问题展开了讨论, 并探讨了相关的并行冷重启拓展.

目前主流的并行 CDCL 求解器普遍利用不同的求解器或参数设计多样性引入技术[23,30,31,40]. 本文提出的并行 FO 技术是一种更便捷高效的多样性引入技术. 事实上, 基于本文中相关技术研制的 SAT 并行求解器 ParKissat-RS[41], 拓展了预处理器技术, 参加了 SAT 比赛 2022 年并行主赛道, 并以领先第 2 名 24% 的大幅优势取得了冠军. 同时, 其改进版本 PRS[46]蝉联了 2023 年并行赛道的冠军.

6 总 结

主流的 CDCL 求解器在重启时普遍保留了所有搜索信息的热重启技术, 这会使得重启之后更倾向于保持之前的搜索方向. 本文通过实验探讨了该现象, 发现主流 CDCL 方法存在明显的运行时间的扰动. 本文提出定期执行冷重启技术, 遗忘一部分搜索信息, 以通过增加算法的探索性来提高 CDCL 的求解效率. 本文冷重启的并行拓展版本, 进一步利用互补性优势, 为并行 SAT 求解器提出了一种简洁、高拓展性的多样性引入机制. 此外, 详细的实验证明了方法的有效性.

未来, 我们将尝试利用更加先进的技术来设计一套动态冷重启交互机制, 面向具体的实例, 研究更智能的冷重启间隔预测方法. 该工作为定制化 CDCL 的发展提出了一个有趣的发展构想. 同时, 本文的方法可以被视为一种特殊的求解器, 它以可满足性求解能力为导向. 并且我们也准备对 CDCL 求解器内部的其他组件进行重新审视, 以进一步提升其专用性.

References

[1] Biere A, Heule M, van Maaren H, Walsh T. Handbook of Satisfiability. 2nd ed., Washington: IOS Press, 2021. 336.

[2] Marques-Silva JP, Sakallah KA. GRASP: A search algorithm for propositional satisfiability. IEEE Trans. on Computers, 1999, 48(5): 506–521. [doi: 10.1109/12.769433]

[3] Wang Q, Liu L, Lü S. #SAT solving algorithms based on extension rule using heuristic strategies. Ruan Jian Xue Bao/Journal of Software, 2018, 29(11): 3517–3527 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5298.htm [doi: 10.13328/j.cnki.jos.005298]

[4] Marques-Silva JP, Sakallah KA. Boolean satisfiability in electronic design automation. In: Proc. of the 37th Annual Design Automation Conf. Los Angeles: ACM, 2000. 675–680. [doi: 10.1145/337292.337611]

[5] D'Silva V, Kroening D, Weissenbacher G. A survey of automated techniques for formal software verification. IEEE Trans. on Computer-aided Design of Integrated Circuits and Systems, 2008, 27(7): 1165–1178. [doi: 10.1109/TCAD.2008.923410]

[6] Xiang Y, Huang H, Luo C, Yang XW. Software product line testing based on diverse sat solvers and novelty search. Ruan Jian Xue Bao/Journal of Software, 2024, 35(6): 2821–2843 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6906.htm [doi: 10.13328/j.cnki.jos.006906]

[7] Vizel Y, Weissenbacher G, Malik S. Boolean satisfiability solvers and their applications in model checking. Proc. of the IEEE, 2015, 103(11): 2021–2035. [doi: 10.1109/JPROC.2015.2455034]

[8] Mironov I, Zhang LT. Applications of SAT solvers to cryptanalysis of hash functions. In: Biere A, Gomes CP, eds. Theory and Applications of Satisfiability Testing. Berlin, Heidelberg: Springer, 2006. 102–115. [doi: 10.1007/11814948_13]

[9] Großmann P, Hölldobler S, Manthey N, Nachtigall K, Opitz J, Steinke P. Solving periodic event scheduling problems with SAT. In: Jiang H, Ding W, Ali M, Wu X, eds. Advanced Research in Applied Artificial Intelligence. Berlin, Heidelberg: Springer, 2012. 166–175. [doi: 10.1007/978-3-642-31087-4_18]

[10] Biere A, Fröhlich A. Evaluating CDCL restart schemes. In: Proc. of the 2015 Int'l Workshop on Pragmatics of SAT. Austin, 2015. 1–17.

[11] Audemard G, Simon L. Refining restarts strategies for SAT and UNSAT. In: Milano M, ed. Principles and Practice of Constraint Programming. Berlin, Heidelberg: Springer, 2012. 118–126. [doi: 10.1007/978-3-642-33558-7_11]

[12] Ryvchin V, Strichman O. Local restarts. In: Büning H K, Zhao XS, eds. Theory and Applications of Satisfiability Testing—SAT 2008. Berlin, Heidelberg: Springer, 2008. 271–276. [doi: 10.1007/978-3-540-79719-7_25]

[13] Huang JB. The effect of restarts on the efficiency of clause learning. In: Proc. of the 20th Int'l Joint Conf. on Artificial Intelligence. Hyderabad: Morgan Kaufmann Publishers Inc., 2007. 2318–2323.

[14] Liang JH, Ganesh V, Poupart P, Czarnecki K. Learning rate based branching heuristic for SAT solvers. In: Creignou N, Le Berre D, eds. Theory and Applications of Satisfiability Testing—SAT 2016. Cham: Springer, 2016. 123–140. [doi: 10.1007/978-3-319-40970-2_9]

[15] Eén N, Sörensson N. An extensible SAT-solver. In: Giunchiglia E, Tacchella A, eds. Theory and Applications of Satisfiability Testing. Berlin, Heidelberg: Springer, 2004. 502–518. [doi: 10.1007/978-3-540-24605-3_37]

[16] Oh C. Between SAT and UNSAT: The fundamental difference in CDCL SAT. In: Heule M, Weaver S, eds. Theory and Applications of Satisfiability Testing—SAT 2015. Cham: Springer, 2015. 307–323. [doi: 10.1007/978-3-319-24318-4_23]

[17] Li CM, Xiao F, Luo M, Manyà F, Lü ZP, Li Y. Clause vivification by unit propagation in CDCL SAT solvers. Artificial Intelligence, 2020, 279: 103197. [doi: 10.1016/j.artint.2019.103197]

[18] Cai SW, Zhang XD. Deep cooperation of CDCL and local search for SAT. In: Li CM, Manyà F, eds. Theory and Applications of Satisfiability Testing—SAT 2021. Cham: Springer, 2021. 64–81. [doi: 10.1007/978-3-030-80223-3_6]

[19] Biere A, Fleury M, Froleyks N, Heule MJH. The SAT museum. In: Proc. of the 14th Int'l Workshop on Pragmatics of SAT. Alghero: CEUR, 2023. 72–87.

[20] Gomes CP, Selman B, Crato N. Heavy-tailed distributions in combinatorial search. In: Smolka G, ed. Principles and Practice of Constraint Programming—CP97. Berlin, Heidelberg: Springer, 1997. 121–135. [doi: 10.1007/BFb0017434]

[21] Gomes CP, Selman B, Kautz H. Boosting combinatorial search through randomization. In: Proc. of the 15th AAAI Conf. on Artificial Intelligence. Madison: AAAI, 1998. 431–437.

[22] Gomes CP, Sabharwal A. Exploiting runtime variation in complete solvers. In: Biere A, Heule M, van Maaren H, Walsh T, eds. Handbook of Satisfiability. 2nd ed., Washington: IOS Press, 2021. 463–480. [doi: 10.3233/FAIA200994]

[23] Hamadi Y, Jabbour S, Sais L. ManySAT: A parallel SAT solver. Journal on Satisfiability, Boolean Modeling and Computation, 2009, 6(4): 245–262. [doi: 10.3233/SAT190070]

[24] Liang JH, Oh C, Mathew M, Thomas C, Li CX, Ganesh V. Machine learning-based restart policy for CDCL SAT solvers. In: Beyersdorff O, Wintersteiger CM, eds. Theory and Applications of Satisfiability Testing—SAT 2018. Cham: Springer, 2018. 94–110. [doi: 10.1007/978-3-319-94144-8_6]

[25] Luby M, Sinclair A, Zuckerman D. Optimal speedup of Las Vegas algorithms. Information Processing Letters, 1993, 47(4): 173–180. [doi: 10.1016/0020-0190(93)90029-9]

[26] Biere A, Fazekas K, Fleury M, Heisinger M. CaDiCaL, kissat, paracooba, plingeling and treengeling entering the SAT competition 2020.

2020. https://kfazekas.github.io/papers/BiereFazekasFleuryHeisinger-SAT-Competition-2020-solvers.pdf
[27] Manthey N. CaDiCaL modification—Watch sat. 2021. https://helda.helsinki.fi/items/bebd00a0-b358-4bf7-b46f-d08a01282385
[28] Kochemazov S, Zaikin O, Semenov A, Kondratiev V. Speeding Up CDCL Inference with Duplicate Learnt Clauses. IOS Press, 2020. 339–346. [doi: 10.3233/FAIA200111]
[29] Cherif MS, Habet D, Terrioux C. Combining VSIDS and CHB using restarts in SAT. In: Proc. of the 27th Int'l Conf. on Principles and Practice of Constraint Programming. Montpellier: Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021. 20: 1–20: 19. [doi: 10.4230/LIPIcs.CP.2021.20]
[30] Le Frioux L, Baarir S, Sopena J, Kordon F. PaInleSS: A framework for parallel sat solving. In: Gaspers S, Walsh T, eds. Theory and Applications of Satisfiability Testing—SAT 2017. Cham: Springer, 2017. 233–250. [doi: 10.1007/978-3-319-66263-3_15]
[31] Tchinda RK, Djamegni CT. Hkis, hcad, PaKis and PaInleSS exmaplelcmdistchronobt in the SC21. 2021. https://helda.helsinki.fi/items/bebd00a0-b358-4bf7-b46f-d08a01282385
[32] Biere A, Fröhlich A. Evaluating CDCL variable scoring schemes. In: Heule M, Weaver S, eds. Theory and Applications of Satisfiability Testing—SAT 2015. Cham: Springer, 2015. 405–422. [doi: 10.1007/978-3-319-24318-4_29]
[33] Moskewicz MW, Madigan CF, Zhao Y, Zhang L, Malik S. Chaff: Engineering an efficient SAT solver. In: Proc. of the 38th Annual Design Automation Conf. Las Vegas: IEEE, 2001. 530–535. [doi: 10.1145/378239.379017]
[34] Ryan L. Efficient algorithms for clause-learning SAT solvers [MS. Thesis]. Burnaby: Simon Fraser University, 2004.
[35] Pipatsrisawat K, Darwiche A. A lightweight component caching scheme for satisfiability solvers. In: Proc. of the 10th Int'l Conf. on Theory and Applications of Satisfiability Testing. Lisbon: Springer, 2007. 294–299. [doi: 10.1007/978-3-540-72788-0_28]
[36] Biere A, Fleury M. Chasing target phases. 2020. https://fmv.jku.at/papers/BiereFleury-POS20.pdf
[37] Heule MJH, Kullmann O, Wieringa S, Biere A. Cube and conquer: Guiding CDCL SAT solvers by lookaheads. In: Eder K, Lourenço J, Shehory O, eds. Hardware and Software: Verification and Testing. Berlin, Heidelberg: Springer, 2012. 50–65. [doi: 10.1007/978-3-642-34188-5_8]
[38] Heule MJH, Kullmann O, Marek VW. Solving and verifying the Boolean Pythagorean triples problem via cube-and-conquer. In: Creignou N, Le Berre D, eds. Theory and Applications of Satisfiability Testing—SAT 2016. Cham: Springer, 2016. 228–245. [doi: 10.1007/978-3-319-40970-2_15]
[39] Heule M. Schur number five. In: Proc. of the 32nd AAAI Conf. on Artificial Intelligence. New Orleans: AAAI, 2018. 6598–6606. [doi: 10.1609/aaai.v32i1.12209]
[40] Schreiber D, Sanders P. MallobSat: Scalable SAT solving by clause sharing. Journal of Artificial Intelligence Research, 2024, 80: 1437–1495. [doi: 10.1613/jair.1.15827]
[41] Zhang X, Chen Z, Cai S. Parkissat: Random shuffle based and pre-processing extended parallel solvers with clause sharing. 2022. https://helda.helsinki.fi/items/dc603b0f-f5cc-4adc-8f82-a0d38360d1c4
[42] Knuth DE. Satisfiablility. The Art of Computer Programming. Boston Columbus Indianapolis: Addison-Wesley, 2018.
[43] Baptista L, Marques-Silva J. Using randomization and learning to solve hard real-world instances of satisfiability. In: Dechter R, ed. Principles and Practice of Constraint Programming—CP 2000. Berlin, Heidelberg: Springer, 2000. 489–494. [doi: 10.1007/3-540-45349-0_36]
[44] Ramos A, van der Tak P, Heule MJH. Between restarts and backjumps. In: Proc. of the 14th Int'l Conf. on Theory and Application of Satisfiability Testing. Ann Arbor: Springer, 2011. 216–229. [doi: 10.1007/978-3-642-21581-0_18]
[45] Biere A, Heule M. The effect of scrambling CNFs. In: Berre D L, Järvisalo M, eds. Proc. of Pragmatics of SAT 2015 and 2018. Oxford: EasyChair, 2018. 111–126. [doi: 10.29007/9dj5]
[46] Chen Z, Zhang X, Qian Y, Cai S. PRS: A new parallel/distributed framework for SAT. 2023. https://researchportal.helsinki.fi/files/269128852/sc2023_proceedings.pdf

附中文参考文献

[3] 王强, 刘磊, 吕帅. 基于扩展规则的启发式#SAT 求解算法. 软件学报, 2018, 29(11): 3517–3527. http://www.jos.org.cn/1000-9825/5298.htm [doi: 10.13328/j.cnki.jos.005298]
[6] 向毅, 黄翰, 罗川, 杨晓伟. 基于多样性 SAT 求解器和新颖性搜索的软件产品线测试. 软件学报, 2024, 35(6): 2821–2843. http://www.jos.org.cn/1000-9825/6906.htm [doi: 10.13328/j.cnki.jos.006906]

作者简介

张昕荻, 博士, 特别研究助理, CCF 专业会员, 主要研究领域为约束求解, EDA 形式化验证.
陈志翰, 博士生, 主要研究领域为约束求解, 电子设计自动化.
蔡少伟, 博士, 研究员, 博士生导师, CCF 杰出会员, 主要研究领域为约束求解, 组合优化, 运筹优化, EDA 形式化验证.