doi: 10.12068/j.issn.1005-3026.2020.01.008

基于重启策略的学习子句优化方法

李 壮, 刘 磊, 张桐搏, 吕 帅
(吉林大学 计算机科学与技术学院, 吉林 长春 130012)

摘 要: 以学习子句数据库优化为背景,在原MiniSAT求解器的基础上提出了一种新的学习子句的优化方法. 该方法基于博弈论的思想,在若干次重启后,根据当前求解器的实时反馈信息改进MiniSAT原有的增长参数,尽可能靠近学习数据库中子句存储量的均衡点,从而使学习库的存储量尽可能达到Pareto最优. 实验表明:所提的优化方法是有效的,并在随机SAT问题上胜过现有优化方法. 该方法既不会因为学习数据库的子句过多而影响单元传播速度,也不会因为学习数据库中的子句过少而破坏学习的整体性.

关 键 词: DPLL;子句学习;学习子句数据库;MiniSAT求解器;Pareto最优
中图分类号: TP 181 **文献标志码**: A **文章编号**: 1005-3026(2020)01-0044-05

Learnt Clauses Optimize Method Based on Restart Strategy

LI Zhuang, LIU Lei, ZHANG Tong-bo, LYU Shuai
(College of Computer Science and Technology, Jilin University, Changchun 130012, China. Corresponding author: LYU Shuai, E-mail: lus@jlu.edu.cn)

Abstract: With the background of optimization of the learnt clauses database, a new optimization method for learnt clauses based on the original MiniSAT solver was proposed. Based on game theory, this method adjusts the growth parameters after several restarts on the basis of the real-time feedback of the current solver, in order to be as close as possible to the balance point of clauses in the learnt clauses database. This makes the storage capacity of the learnt clause database reach the Pareto optimality as much as possible. Experiments showed that the proposed method is effective and outperforms existing optimization methods in random SAT problems. This method neither affects the speed of unit propagation because of too many clauses in the learnt clause database, nor destroys the integrity of learning because of too few clauses in the learnt clause database.

Key words: DPLL (Davis-Putnam-Logemann-Loveland); clause learning; learnt clauses database; MiniSAT solver; Pareto optimality

可满足性问题(SAT)是计算机领域的热点问题,许多学者在该问题上作了大量研究,并在问题求解效率方面取得了较大的突破,当今SAT求解器已能够解决很多现实问题. 随着SAT求解技术的发展,子句学习技术进入了蓬勃发展的阶段[1]. 对于现实问题,子句学习(clause learning)是DPLL求解中最重要的部分[2].

在实践中,SAT的求解效率很大程度上取决于对已学习子句数据库的管理策略. 即发生冲突后,新子句添加到已学习的子句数据库中,其增长规模是呈指数增长的. 保留太多的学习子句会减缓单元传播的速率,而删除太多将会破坏学习的整体性[3]. 所以,优化目标在于清除掉已学习子句数据库中被判定为在以后的搜索中无关的已学习子句. 学习子句管理策略的效率,很大程度上取决于消除频率和每次删除的子句数量[4].

为了优化已学习子句数据库,近几年一些先进的管理策略已被提出. 首次提出质量测量的是VSIDS启发式,该优化策略假设一个过去学习的子句在未来的求解过程中可能是有用的[5]; Audemard等介绍了一个新的静态措施,加入LBD子句这一概念,更加完善了对学习子句的预

收稿日期: 2019-03-19
基金项目: 国家重点研发计划项目(2017YFB1003103); 国家自然科学基金资助项目(61300049,61763003); 吉林省科技发展计划项目(20180101053JC,20190201193JC).
作者简介: 李壮(1988-),男,吉林白城人,吉林大学博士研究生; 刘磊(1960-),男,吉林长春人,吉林大学教授,博士生导师.

处理技术[3]；Audemard 等提出了一个基于动态的冻结与激活学习子句的原则，即激活最相关的学习子句而冻结不相关的学习子句[6]；Hamadi 等提出规约策略 SBR，保留文字数小于等于 k 的子句，同时随机删除大于 k 的子句[7]；Luo 等提出了学习子句最小化的方法，该方法通过 BCP 删除学习子句中的冗余文字，从而达到对学习子句质量的优化[4]；Luo 等基于 Maple_LCM 求解器的子句最小化方法，在 Maple_CM 求解器上扩展到原始子句的预处理[8]. 对学习库的优化主要包括：优化学习库的时机、应删除哪些子句以及每次优化应保留多少子句. 经分析，优化学习库的时机对求解效率的影响并不大，多数研究人员将注意力集中在应删除哪些冗余子句，即注重了学习子句的质量，保留更多有用的学习子句，而对应保留学习库中子句数量的工作甚少.

本文基于经典的 SAT 求解器，提出了新的动态管理学习子句数据库的算法. 在优化的过程中，通过博弈论的思想，在阶段重启的过程中，通过程序中输出的阶段布尔约束传播（Boolean constraint propagation，BCP）速率和平均 BCP 速率的对比，调整学习数据库的增长参数，更大可能地靠近学习数据库中子句存储量的均衡点，从而使学习库的存储量尽可能地达到 Pareto 最优. 实验结果表明：新的优化方法在随机问题上胜过传统的优化方法，在求解出相同的算例中，GTMiniSAT 求解器的求解效率也远超过 MiniSAT 求解器.

1　CDCL 和重启策略

子句学习采用了强大的冲突分析技术，其主要思想是在求解过程中发生冲突时，分析冲突原因并推理出一些简洁地表达冲突原因的子句，这种已学习的子句用于在后面的搜索过程中避免发生同样的冲突且减少搜索空间，更加快了求解的效率.

当单元传播过程中出现了一对互补文字时，求解器判定子句集 F 中出现了一个冲突，这个冲突将驱动子句学习机制，将冲突原因保存为一个新的子句 C，并将它添加到原子句集中，以防止在接下来的搜索过程发生相同的冲突，这便是冲突驱动子句学习（conflict-driven clause learning，CDCL）[2].

重启发生在子句被学习之后，当指派过程中发生了 $\varepsilon(\varepsilon>1)$ 次冲突后，重启策略被触发，求解器终止当前搜索并返回到第 0 层，即重新求解该子句集[9]. 冲突上界会通过参数 ε 来决定.

2　基于博弈论的学习子句优化方法

学习库空间的占有主要体现在对学习库增长参数的控制. 如何调整学习库空间的分配，使之在优化过程中无限接近学习库的纳什均衡点，达到 Pareto 最优状态，将是子句学习的关键.

2.1　博弈模型

求解效率是求解算法好坏的关键衡量指标. 由于 BCP 时间占了总求解时间约 90% 以上[10]，所以本文将求解时间等同于 BCP 时间.

求解效率是无法量化而又客观存在的，但满足：平均求解效率越高，平均 BCP 速率也越高. 所以在本文中用平均 BCP 速率代替求解效率.

在一个 SAT 问题求解过程中，存在着最基本的数学公式：

$$\text{BCP 总数} = \text{BCP 速率} \times \text{BCP 时间}. \quad (1)$$

在子句学习库中，学习子句的多少严重影响了 BCP 速率. 因此，SAT 求解过程对应了博弈模型，其中：参与者对应平均 BCP 速率和阶段 BCP 速率；策略对应学习库的调整参数 ± 0.01；利益对应求解效率提高和 BCP 速率提高；信息对应平均 BCP 速率和阶段 BCP 速率.

综上，博弈模型的支付矩阵如表 1 所示.

表 1　学习子句数据库的支付矩阵

Table 1　Payment matrix of learnt clause database

BCP 速率	求解效率	
	上升	下降
上升	时间短	学习库较少
下降	学习库较多	时间长

2.2　博弈模型的分析

本文令重启总数为 N，每重启 S 次记为一个阶段，第 i 阶段的 BCP 数量为 s_i，相应的第 i 阶段的执行时间为 t_i，则阶段 BCP 速率 $v_i = s_i/t_i$.

截至第 n 阶段，BCP 总量 $S_n = \sum_{i=1}^{n} s_i$，BCP 总传播时间 $t_n = \sum_{i=1}^{n} t_i$，则平均 BCP 速率为 $\tilde{v}_n = \sum_{i=1}^{n} s_i / \sum_{i=1}^{n} t_i$.

记 $p_i \in [-0.1, 0, 0.1]$ 为每个阶段的增减参数. 当 $v_i < \tilde{v}_i$ 时，$p_{i+1} = -0.1$；当 $v_i > \tilde{v}_i$ 时，$p_{i+1} = 0.1$；当 $v_i = \tilde{v}_i$ 时，$p_{i+1} = 0$. 学习库增减参数 $P_n = \max\{0.5, 1.1 + \sum_{i=1}^{n} p_i\}$.

以上分析不难看出:当 v_i 和 $\tilde{v}_i$ 都减小时,可能出现等值,这时达到了 Pareto 最优,即两者的效用同时得到最大化;当 $v_i > \tilde{v}_i$ 时,说明学习库中的学习子句相对较少. 在不影响 BCP 速率的情况下,可以适量增加学习子句的数量,这里采用增减参数设置为 0.1 来实现;同理,当 $v_i < \tilde{v}_i$ 时,相应设置为 -0.1 来控制学习库的增长速率,从而在不影响求解效率的同时增大阶段 BCP 的增长速率;而两者皆增大的情况是不可能出现的. 实际的支付矩阵见表 2.

表 2 学习子句数据库的实际支付矩阵

Table 2 Real payment matrix of learnt clause database

阶段 BCP 速率	平均 BCP 速率	
	减小	增大
减小	纳什均衡	-0.1
增大	+0.1	公地悲剧

这一过程是一个连续的博弈过程,其根本目的是在不断地追求无限接近纳什均衡点,实现已学习子句库的 Pareto 最优分配.

2.3 基于博弈论的学习子句优化算法

算法 1 是 GTMiniSAT 求解器的简要求解过程,其中包括了学习子句优化的函数.

算法 1 DPLL with clause learning

输入:CNF formula F

输出:A solution P of F or UNSAT if F is not satisfiable

```
1. BCP_num + +
2. if P assigns a value to every variable
3.     return success
4. else
5.     P contains a conflict
6.     choose a conflict graph G to find clause C
       under P and add it to F
7.     rollback variable or backjump
8.     conflict_num + +
9.     if conflict_num > ε
10.        restart + +
11.        if restart% S == 0
12.            BCP_total + = BCP_num
13.            speed_average = BCP_total/nowtime
14.            speed_now = BCP_num/(nowtime -
               lasttime)
15.            if speed_now > speed_average
16.                learntsize_inc + = p
17.            else if speed_now < speed_average
18.                if learntsize_inc - p ≥ 0.5
19.                    learntsize_inc - = p
20. restarts
```

在算法 1 中,第 9 行表明当冲突数量达到了临界值 ε 时重启,第 11 行表明重启次数达到 S(实验表明 55 为最优值)次时,触发重启参数调整策略. 第 12 ~ 19 行描述了控制增长参数的过程. 当冲突数达到 55 次后,第 12 行统计从开始求解截止到目前的 BCP 数量. 第 13 ~ 14 行计算平均 BCP 速率和阶段 BCP 速率. 第 15 ~ 19 行通过对速率的比较,来调整增减参数. 当阶段 BCP 速率大于平均 BCP 速率时,增长参数提高;当阶段 BCP 速率小于平均 BCP 速率时,增长参数降低. 第 18 行控制了下界,当参数小于 0.5 时,增减参数不变.

3 实验与结果

本文从 2017 年 SAT 比赛 Agile 组中随机选取了 200 个测试用例,其规模为几百个变量(1 000条子句)至 180 万变量(668 万条子句)不等,CPU 在求解器运行时间至 5 000 s 时停止. 其重启参数默认为 50,增减参数默认为 0.1. 实验环境:Arch Linux 16.04 操作系统,12 核 CPU,8GB RAM.

图 1 横坐标代表 MiniSAT 求解器的运行时间,纵坐标代表 GTMiniSAT 求解器的运行时间. 所以,分布在斜线以下的点说明这些算例在 GTMiniSAT 求解器上的求解效果较好;在斜线以上的点说明这些算例在 MiniSAT 求解器上的求解效果较好. 可以看出,GTMiniSAT 求解器效率较明显地优于 MiniSAT 求解器. 斜线以上的点多半离斜线较近,说明这些算例在 MiniSAT 求解器的求解效果略优于GTMiniSAT求解器;而斜线

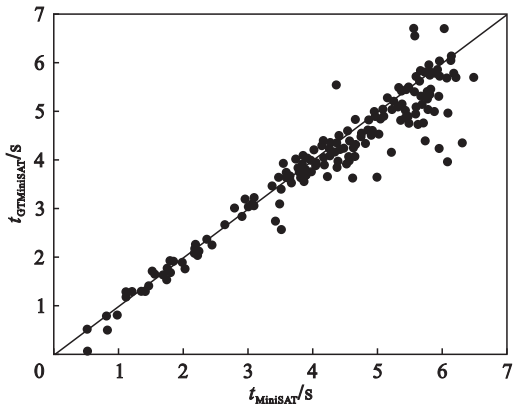

图 1 200 个随机算例的对比图

Fig. 1 Comparison of 200 random benchmarks

以下的点有很大一部分离斜线较远，说明这一部分的算例在 GTMiniSAT 求解器的求解效果明显优于 MiniSAT 求解器.

为了更准确地决定重启的次数对 GTMiniSAT 求解器的影响，在上述 200 个算例中，多次尝试了重启 1 次，5 次…直到重启 65 次，增减参数默认为 0.1，CPU 在求解器运行时间至 5 000 s时停止. 实验结果如表 3 所示.

表 3　GTMiniSAT 求解器重启次数的测试
Table 3　Test of restart times for GTMiniSAT solver

重启次数	求解个数		平均求解时间/s
	可满足	不可满足	
0	89	110	199.81
1	88	100	205.33
5	89	109	212.73
10	89	110	230.19
15	89	110	217.16
20	89	110	218.52
25	89	110	226.89
30	90	110	183.84
35	90	110	203.45
40	90	110	198.74
45	90	110	217.13
50	89	110	212.83
55	90	110	173.24
60	89	111	184.46
65	89	110	200.55

由表 3 可知，重启次数固定在 55 次，平均求解时间最短，而且优于 MiniSAT 求解器. 而其他参数的数据表明求解的平均效率较低，且重启 55 次时求解个数也是最多的. 因此，在重启 55 次后调整增减参数是最合理的.

在固定了重启参数为 55 次之后，接下来需要对增减参数进行测试. 在上述算例中，尝试增减 0.01，0.02…直至 0.1. CPU 在求解器运行时间至 5 000 s 时停止. 实验结果如表 4 所示.

由表 4 可知，增减参数为 0.05 和 0.1 的两组求解效率比较高，而且优于 MiniSAT 求解器. 但两者所处的距离较远，中间出现了一系列的波动. 出现的原因是否会受到重启参数的影响，即重启参数越大，所对应的增减参数则越大，两者是否成正比关系呢？为了证明这个想法，在以下实验中给出了更具体的证明，并且验证本文的 GTMiniSAT 求解器对于一般的工业问题，也是优于 MiniSAT 求解器的.

表 4　GTMiniSAT 求解器增减参数的测试
Table 4　Test of Increasing or decreasing the parameters for GTMiniSAT solver

增减参数	求解个数		平均求解时间/s
	可满足	不可满足	
±0.01	90	110	173.24
±0.02	90	110	224.47
±0.03	90	110	182.01
±0.04	90	110	198.99
±0.05	90	110	171.28
±0.06	90	110	176.24
±0.07	89	110	197.28
±0.08	90	110	201.41
±0.09	90	110	191.56
±0.1	90	110	167.83

GTMiniSAT 求解器主要针对规模较大的工业问题有效，简单问题的优化效果并不明显. 所以，选择了 2017 年 SAT 竞赛中 Random 组的 300 个算例，CPU 在求解器运行时间至 5 000 s 时停止. 实验结果见表 5.

表 5　2017 年 SAT 竞赛随机组 300 算例
Table 5　Test of 300 Benchmark in the random track of the 2017 SAT competition

求解器	求解个数	平均求解时间/s
MiniSAT	124	2 418.37
(±0.05) – GTMiniSAT	114(50)	2 535.89
(±0.1) – GTMiniSAT	114(72)	2 539.37

由表 5 可知，GTMiniSAT 求解器所求出的两组算例是相同的 114 个，且包含在 MiniSAT 求解器所求出的 124 个算例中，平均求解时间也略高于 MiniSAT 求解器. (±0.1) – GTMiniSAT 求解器所解出的 114 个算例中，有 72 个算例比 MiniSAT 求解器快，而(±0.05) – GTMiniSAT 仅有 50 个算例优于 MiniSAT 求解器. 因此，选择增减参数为 0.1 是最合理的.

虽然 GTMiniSAT 求解器平均求解时间并没有胜过 MiniSAT 求解器，但是通过统计实验结果，在这 72 个算例中，MiniSAT 求解器的平均求解时间为 136.56 s，而 GTMiniSAT 求解器的平均求解时间为 44.44 s. 表 6 中统计了所有问题的求解个数和平均求解时间，数据表明：GTMiniSAT 求解器的效率远高于 MiniSAT 求解器，其求解效率提高了 67.45%.

表 6 GTMiniSAT 和 MiniSAT 求解器的对比实验结果
Table 6 Comparative evaluation between GTMiniSAT and MiniSAT solvers

求解器	求解个数	平均求解时间/s
MiniSAT	124(72)	136.56
(±0.1)-GTMiniSAT	114(72)	44.44

为了验证 GTMiniSAT 求解器的有效性,再次对 GTMiniSAT 求解器和 MiniSAT 求解器在共同求解出的 114 个算例中做出数据统计,如表 7 所示. 从表 7 可知,GTMiniSAT 求解器的求解效率也远超过 MiniSAT 求解器,其求解效率提高了 35.76% .

表 7 GTMiniSAT 和 MiniSAT 求解器的对比实验(共同求解算例)
Table 7 Comparative evaluation of between GTMiniSAT and MiniSAT solvers(common instances solved)

求解器	求解个数	平均求解时间/s
MiniSAT	124(114)	109.07
(±0.1)-GTMiniSAT	114(114)	70.06

4 结 语

本文首次将博弈论的思想应用于经典 SAT 求解中,通过重启次数来调整学习库的增长参数,将原有学习数据库的静态等比级数增长改进为动态的变化,尽可能靠近学习数据库中子句存储量的均衡点,从而使学习库的存储量尽可能达到 Pareto 最优.

GTMiniSAT 还存在如下问题有待解决:1)之所以改进 MiniSAT 求解器,因为 MiniSAT 是一款经典的求解器,对于新方法的融入有更大的接受空间,下一步工作将考虑改进 Maple_CM 等当前最先进的求解器. 2)由于分析的角度不同,选择的博弈模型也不同,本文中求解效率和 BCP 速率具有竞争关系,而同时又拥有共同目标——求解 SAT 问题,即它们相互合作却又彼此竞争. 下一步工作将站在两者共同的立场(即合作博弈),对它们的合作做出进一步的博弈分析.

参考文献:

[1] Beame P, Kautz H, Sabharwal A. Towards understanding and harnessing the potential of clause learning[J]. *Journal of Artificial Intelligence Research*, 2004, 22: 319-351.

[2] Silva J, Sakallah K. Grasp—a new search algorithm for satisfiability[C]//IEEE/ACM International Conference on Computer-aided Design (CAD 1996). Ann Arbor, 1996: 220-227.

[3] Audemard G, Simon L. Predicting learnt clauses quality in modern SAT solvers[C]//International Joint Conference on Artificial Intelligence (IJCAI 2009). Pasadena, 2009: 399-404.

[4] Luo M, Li C, Xiao F, et al. An effective learnt clause minimization approach for CDCL SAT solvers[C]// Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI 2017). Melbourne, 2017: 703-711.

[5] Moskewicz M, Madigan C, Zhao Y, et al. Chaff: engineering an efficient SAT solver[C]//Proceedings of Design Automation Conference (DAC 2001). Las Vegas, 2001: 530-535.

[6] Audemard G, Lagniez J M, Mazure B, et al. On freezing and reactivating learnt clauses[C]//Proceedings of the 14th International Conference on Theory and Applications of Satisfiability Testing (SAT 2011). Ann Arbor, 2011: 188-200.

[7] Hamadi Y, Jabbour S, Saïs L. What we can learn from conflicts in propositional satisfiability[J]. *Annals of Operations Research*, 2016, 240(1): 13-37.

[8] Luo M, Xiao F, Li C, et al. Maple_CM, Maple_CM_Dist, Maple_CM_ordUIP and Maple_CM_ordUIP + in the SAT competition 2018[C]//Proceedings of SAT Competition 2018—Solver and Benchmark Descriptions. Oxford, 2018: 44-46.

[9] Atserias A, Fichte J K, Thurley M. Clause-learning algorithms with many restarts and bounded-width resolution[J]. *Journal of Artificial Intelligence Research*, 2011, 40: 353-373.

[10] Dixon H E, Ginsberg M L, Parkes A J. Generalizing boolean satisfiability I: background and existing work[J]. *Journal of Artificial Intelligence Research*, 2004, 21: 193-243.