Journal of Computer Applications ISSN 1001-9081 2017
计算机应用,2017 CODEN JYIIDU http://www.joca.cn

最新录用 doi:

基于动态奖惩的分支策略的 SAT 完备算法

刘燕丽[1,2*]，徐振兴[2]，熊丹[1]

(1.武汉科技大学 理学院，武汉 430068；
2.华中科技大学 计算机学院，武汉 430070)

(*通信作者电子邮箱 yanlil2008@163.com)

摘 要: 针对学习子句数量有限或相似度高，导致历史信息有限、搜索树不平衡的问题，提出了基于动态奖惩的分支策略。首先，对每次单子句传播的变元进行惩罚，依据变元是否产生冲突和产生冲突的间隔，确立不同的惩罚函数。其次，在学习阶段，利用学习子句确定对构造冲突有益的变元，非线性增加它们的活跃度。最后，选择活跃度最大的变元作为新分支变元。在 glucose3.0 算法基础上，完成了改进的动态奖惩算法(AP7)。实验数据显示相比 glucose3.0 算法，AP7 的剪枝率提高了 14.2%~29.3%，少数算例的剪枝提高率可达 51%。改进后的 AP7 算法相比 glucose3.0 运行时间缩短了 7%以上。实验结果表明所提分支策略可以有效降低搜索树层次，使搜索树更加平衡，减少计算时间。

关键词: NP 完全问题；可满足性问题；冲突驱动子句学习；完备算法；分支策略

中图分类号:TP301 一般性问题 **文献标志码**：**A**

Exact SAT algorithm based on dynastic branching strategy of award and punishment

LIU Yanli[1,2*], XU Zhenxing[2], XIONG Dan[1]

(1.College of Science, Wuhan University of Science and Technology, Wuhan Hubei 430068, China;
2.College of *Computer Science*, Huazhong University of Science and Technology, Wuhan Hebei 430070, China)

Abstract: Concerning the problem that less and similar learning clauses caused a very imbalanced search tree, a dynastic branching strategy of award and punishment was proposed. First, these variables occurring in unit propagation were punished. Considering the effect of variables on inducing logic conflicts and the conflicts intervals, different punishment functions were built; then in the learning phase, the positive variables were found for the conflict according to the learning clauses, and their activities were nonlinearly increased; at last, the maximum activity's variable was chosen as the new branch variable. On the basis of the algorithm glucose 3.0, an improved algorithm (AP7) adopted the new dynastic award and punishment strategy. After computing the industrial problems of Satisfiability Problem Competition, the experimental data show that AP7 improved the radios of cutting branches 14.2%~29.3% than glucose3.0, in rare cases, the radios of cutting branches reached 51%. The running time of AP7 is shorter at least 7% than glucose3.0. So, the new branching strategy can efficiently reduce the size of search tree, make the search tree more balanced than ever and shorter the running time.

Keywords: non-deterministic polynomial complete problem; satisfiability problem; conflict driven clause learning; exact algorithm; branching heuristic strategy

0 引言

可满足性问题(SATisfiability problem，SAT)是计算机科学领域经典的 NP 问题，在计算机科学理论中占有非常重要的地位。现实世界的问题大多数都是 NP 问题，即不确定性问题。比如，超大规模集成电路测试、资源配置、网络的搜索、数据挖掘、城市交通等，解决这些工业问题的 SAT 技术推动了人工智能的重要发展。

目前，绝大多数 SAT 完备算法是基于深度遍历二叉树的 DPLL(Davis Putnam Logemann Loveland)[1]算法。2001 年 Moskewicz 等[2]提出的冲突驱动子句学习(Conflict Driven Clause Learning, CDCL)，使得 SAT 求解效率有了标志性的进步，在合理时间内，求解规模从数百个变元，扩大到数万个变元。后续提出的 SAT 完备算法的主要方法和改进[3-4]是基于冲突驱动子句学习技术。比如，非时间序列回溯(Non-chronological Backtracking, NCB)[5]，该策略分析冲突的

收稿日期: 2017-06-07； 修回日期:2017-08-03。基金项目:湖北省教育厅科技项目(B2016015)

作者简介: 刘燕丽(1980-)，女，河南西平人，讲师，硕士，CCF 会员，主要研究方向： NP 问题的算法设计，算法优化；徐振兴(1990-)，男，湖北鄂州人，博士研究生，主要研究方向：NP 问题的近似求解、算法优化；熊丹(1979-)，女，湖北天门人，讲师，硕士，主要研究方向：数理统计、系统辨识。

产生原因，记录优化的学习子句，并在回溯后，优先满足学习子句，以避免再次陷入相同冲突。文字块距离(Literals Blocks Distance, LBD) [6]是统计学习子句中不同分支层的变元数，是一个动态变化值。子句库始终保留 2 元学习子句或 LBD 值小的学习子句。该策略既避免了因学习子句剧增，内存崩溃的危险，又保留了质量高的变元约束条件，是有效的子句删除策略。

变元独立衰减 (Variables State Independent Decaying Sum, VSIDS) [7]分支策略强化了冲突学习对搜索的影响。一旦变元在产生学习子句的过程中出现，那么该变元的活跃度将非线性增加。近几年的 SAT 竞赛数据显示变元独立衰减策略或者其变种是较为高效的分支策略。动态重启(Dynastic Restart, DR) [8]是当算法较长时间不能找到冲突时，程序撤销之前的搜索动作，-重新开始遍历二叉树。因为-新搜索是在含有学习子句的子句库中进行，学习子句代表了之前的搜索信息，所以重启并不是推翻之前的搜索。目前，工业算例的规模可达百万个变元数，千万个子句数。因为问题的复杂度高，量化这些策略在搜索过程中的作用，以及它们之间的相互影响，是非常困难的事情。

本文针对在搜索空间为 2^n(n 是变元数)的二叉树中，由于学习子句数量有限，或学习子句相似度高而导致搜索树层次过深、计算效率低的问题，提出了双阶段评分的新分支策略。即对于搜索阶段传播的变元，若其对构造冲突作用小，则给予适当的惩罚；对于学习阶段中构建冲突的变元给予奖励，通过较为全面地评估变元对搜索的影响，以达到更快地发现冲突，提高剪枝率的目标。计算了 SAT 国际竞赛的工业组算例，实验结果表明新变元选择策略可有效地调整搜索树的高度，缩短运算时间。

1 可满足性问题的求解

文献[9]给出了可满足性问题的相关术语的定义。

1.1 相关术语

定义 1. 布尔变元集合。 符号 V 表示命题变元集合，$V=\{x_1, x_2, x_3, ..., x_{n-1}, x_n\}$，$x_i \in \{true, false\}$。文字 l_i 是布尔变元 x_i，及其否定形式 $\overline{x}_i$。一般称 x_i 为正文字，$\overline{x}_i$ 为负文字。变元 x_i 取 *true* 使得正文字满足；取 *false* 使得负文字满足。

定义 2. 子句集。子句是 V 上若干文字的析取，记为 c。子句集是由若干子句构成的集合，记为 C。子句长度是子句所含文字的个数，记为$|c|$。特别地，长度为 1 的子句称为单子句。长度为 0 的子句称为空子句，记为 δ。

定义 3. 子句满足。若子句满足当且仅当子句中至少有 1 个文字为 *true*。因不含有任何文字，所以空子句永远不满足。若子句不满足当且仅当子句中所有文字均为 *false*，该子句又称为冲突子句，记为 。

定义 4. 合取范式。 合取范式 (Conjunctive Normal Formula，CNF)是由若干 V 上的子句合取构成的命题公式，记为 F。

定义 5. 真值指派。 真值指派是命题变元$V' \to \{1,0\}$的映射，$V' \subseteq V$，记为 A。当$V'=V$ 时，A 是完整真值指派，否则，称之为部分真值指派。

定义 6. 冲突集。 对任意一组 V 上的真值指派，若子句集 C 中至少含有 1 个不满足子句，则称 C 为冲突集，记为F。若两个冲突集的交为空，则称它们是相互独立的。

定义 7. SAT 问题。 对于命题变元集合 V和子句集 C，判定是否存在一组关于 V 的真值指派使得 C 中所有子句满足。

1.2 SAT 问题的完备算法

目前，DPLL 是绝大多数 SAT 完备算法的主流程的控制程序。算法从根节点开始深度遍历二叉树，当遇到冲突的真值指派时，算法回溯。若算法找到一组真值指派，使得每个子句满足，那么程序终止，返回“可满足”。若遍历完完整的二叉树后，未找到一组真值指派使得所有子句满足，那么程序终止，返回“不可满足”。DPLL 算法详见文献[1]。

1）测试冲突的单子句传播。

单子句传播(Unit Propagation)[9-10]可以帮助 SAT 算法推导出真值指派的冲突，减少搜索空间。算法 1 是单子句传播的具体过程。

算法 1. UnitPropagate(F)。

输入：合取范式 F；

输出：若有冲突，返回冲突子句，否则返回 *NoConflict*。

```
1.   F' ← F
2.   do {
3.     取 UnitQue 中单子句 c_j，满足 c_j 的文字 l;
4.     形如 l ∨ l_i ...∨ l_k 的子句已满足，从 F' 中移除;
5.     形如 l̄ ∨ lm ∨ ls ...∨ l_t 的子句 c_n，化简为 lm ∨ ls ...∨ l_t ;
6.       if( lm ∨ ls ...∨ l_t 为单子句)
7.           lm ∨ ls ...∨ l_t 入 UnitQue 队列;
8.       else if( lm ∨ ls ...∨ l_t 为空子句)
9.           return c_n;
10.  }while(UnitQue 不为空)
11.  return NoConflict;
```

单子句传播满足单子句中的文字，不断地简化其他子句，直至单子句队列 *UnitQue* 为空，或冲突子句出现为止。例 1 说明了单子句传播推理冲突子句的过程。因为 $\overline{x}_i \vee x_j$ 等价于 $x_i \to x_j$，所以建立 F 的蕴含图更直观地体现推理过程。如图 1 所示，x_1 是当前第 9 层的分支文字，$\overline{x}_{17}$，x_{15}，x_{23}，x_2 分别是第 2、5、4、2 层的已满足的文字。当 $\overline{x}_{17}$ =1、x_1 =1

时，子句 $x_{17} \vee \overline{x}_1 \vee x_6$ 简化为单子句 x_6，所以 x_6 的决策层亦是 9。同理，未标识括号的变元的决策层均是第 9 层。表示冲突子句 $x_5 \vee \overline{x}_9$。

例1. F={ $x_5 \vee \overline{x}_9$，$x_{17} \vee \overline{x}_1 \vee x_6$，$\overline{x}_1 \vee x_7$，$\overline{x}_1 \vee x_4$，$\overline{x}_6 \vee \overline{x}_7 \vee \overline{x}_4 \vee \overline{x}_3$，$x_3 \vee \overline{x}_{15} \vee x_{11}$，$\overline{x}_{11} \vee \overline{x}_5 \vee \overline{x}_2$，$x_9 \vee \overline{x}_{11} \vee \overline{x}_{23}$, $x_{17} \vee x_2$, …}

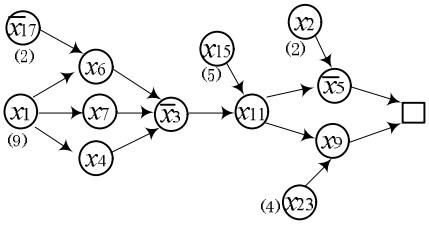

Fig. 1 Logical implication graph of F

图 1　F 的逻辑蕴含理图

2）学习子句的产生。

当搜索到冲突后，SAT 算法进入分析冲突的原因，产生学习子句的学习阶段。图 2 显示了由冲突子句出发，通过消解规则，逐一产生学习子句的过程。消解规则是若两个子句中包含有相反文字，则将这对文字删除，剩余文字通过析取构成新的子句。消解规则不改变合取范式的满足性。

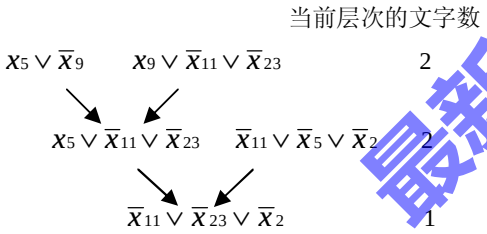

Fig. 2 resolution of learning clause

图 2 学习子句的产生过程

同一个冲突可以学习到多个新子句，如 $x_5 \vee \overline{x}_{11} \vee \overline{x}_{23}$ 和 $\overline{x}_{11} \vee \overline{x}_{23} \vee \overline{x}_2$ 均是学习子句。经实验验证，第一蕴含点 (First unit implication Point, FUIP) [5]的学习方式是更加有效的。具体操作是利用消解规则，按照变元传播的倒序，逐一消解子句中互为相反的变元。记录冲突子句和每个新子句中当前层次的文字数。当该值为 1 时，学习过程结束。如 F 的第一蕴含点的学习子句 L 是 $\overline{x}_{11} \vee \overline{x}_{23} \vee \overline{x}_2$，$F_L$={ $x_5 \vee \overline{x}_9$, $x_9 \vee \overline{x}_{11} \vee \overline{x}_{23}$, $\overline{x}_{11} \vee \overline{x}_5 \vee \overline{x}_2$ }是 L 对应的冲突集。易验证，当学习子句不满足时，在任意真值指派下，冲突集中至少有 1 个子句是不满足的。利用此性质，在后续搜索中，一旦学习子句不满足，即可剪枝。目前，绝大多数 SAT 完备算法依据 FUIP 学习到的子句确定回溯层次。非时间序列回溯是指搜索回溯到学习子句中非当前层次文字的最大层次。如图 2 产生的学习子句 $\overline{x}_{11} \vee \overline{x}_{23} \vee \overline{x}_2$，上层文字集合是{ $\overline{x}_{23}$，$\overline{x}_2$ }，其最大的层次是 4。SAT 算法回溯后，将撤销第 5 层到第 9 层的搜索工作。

2　基于动态奖惩分支策略的 SAT 算法

DPLL+子句学习是 SAT 完备算法的基础。为了降低搜索树的节点数，优化算法的效率，分支策略将选择对搜索树大小影响大的变元。

2.1　基于学习过程的变元策略

如变元独立衰减策略或其变种，这类基于学习过程的分支策略是在产生学习子句的学习阶段，对参与消解操作的所有子句包含的变元，增大其活跃度。算法始终选择当前活跃度最大的变元作为新的分支点。如例 1 会增加变元 x_5、x_9、x_{11}、x_{23}、x_2 的活跃度。变元参与构造冲突越多，其活跃度越大。在后续搜索中选择活跃度大的变元，易找到之前发现过的冲突，有利于减少搜索路径。

我们发现仅基于学习过程的分支策略对减小搜索树大小并不总是有效的。某些情况下，相似的学习过程导致学习子句相似度高。学习子句代表的搜索的历史信息比较集中，回溯后或重启后，基于学习过程的变元策略会仅有少量变元可以优先选择。搜索没有足够的历史信息指引方向，会随机去选择变元作为新的分支节点，这会导致搜索树层次过深，算法陷入困境。目前，绝大多数 SAT 求解器采用动态重启逃脱这种困境。因为子句库保存了表述冲突关系的学习子句，所以重启后的搜索与没有学习信息的原始搜索不同。通过优先选择冲突关系多的变元，新搜索可能会找到冲突。但是动态重启只能缓和算法遇到的问题，并未改变变元的活跃度，程序可能会又陷入到某一子树的过度搜索中。

2.2　基于学习与搜索双阶段的分支策略

为了降低搜索树大小，使搜索树更加平衡，全面地评估变元对搜索的作用，使得越容易构造冲突的变元，越接近树根，增大搜索优先找到冲突的概率是基于动态奖惩分支策略的奖惩算法（Award and Punishment 7，AP7）的主要改进思路。

SAT 完备算法包括 2 个主要过程：一是搜索过程，即单子句传播，发现冲突的操作；二是学习过程，即产生学习子句，增加冲突变元的活跃度，确定回溯层次。AP7 算法对以上两个过程中的变元进行全面地评估。在搜索阶段，降低较长时间内未找到冲突的变元的活跃度，在学习阶段，提高对构造冲突有益的变元的活跃度，并优先选择当前活跃度最大的变元作为分支变元。算法流程如图 3 所示。

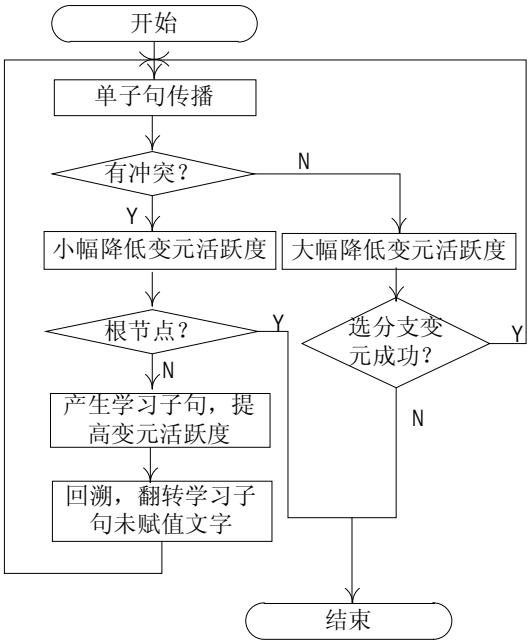

Fig. 3 AP7 algorithm flowchart

图 3 AP7 算法流程图

算法 2 是 AP7 的伪代码。算法中 *penalty* 是让变元活跃度随着传播而衰减的独立参数；*numCC* 是冲突数，初始值为 0；*PenaltyV* 是存储待惩罚变元的向量；*dl* 是搜索树的层数；*lastC*[*x*]是存放变元 *x* 的最近一次出现的冲突；*Act* 是记录变元 *x* 活跃度的向量；F_L 是学习子句 L 对应的冲突集，*Variable*(F_L)是冲突集中出现的所有变元的集合；*V* 是输入算例的变元集合；*BranchV* 是分支变元；max 函数返回活跃度最大的变元；*bl* 是 SAT 算法产生冲突后的回溯层次；AP7 采用非时间序列回溯的方法；*cancel* 函数是算法回溯到指定的层次。

算法 2. AP7(*F*)。

输入：合取范式 *F*；

输出：*F* 是否满足，满足返回“*SAT*”，不满足返回“*UNSAT*”。

```
1.  penalty=0.6;  numCC =0;  penaltyV.clear( );
2.  for( ; ; )
3.    UnitPropagate(F) 且将传播的变元 x 压入 penaltyV 中；
4.    for  x_t ∈ PenaltyV  do          //搜索阶段的惩罚
5.      if  UnitPropagation(F)= =冲突  then
6.        if penalty<0.98 then
7.          penalty= penalty +10^-7;
8.        end if
9.        Act[x_t]=Act[x_t]*penalty+(1-penalty)/(numCC-lastC[x_t]);
10.     else
11.       Act[x_t]=Act[x_t]* penalty;
12.     end if
13.   end for                          //结束惩罚计算
14.   penaltyV.clear( );
15.   if UnitPropagation(F)= =冲突  then   //找到冲突
16.     if dl= =0 then
17.       return UNSAT;
18.     endif
19.     numCC ++;
20.     FUIP 方式产生学习子句 L,并返回回溯层次 bl；
21.     for  x_i ∈ Variable(F_L) do      //学习阶段的奖励
22.       lastC[x_i]=numCC;
23.       Act[x_i] = Act[x_i]+(1/0.9)^numCC
24.       if Act[x_i] >1e100   then
25.         for  x_i ∈ V do
26.           Act[x_i] *=1e-100;        //做平滑
27.         end for
28.       end if
29.     end for                          //学习阶段奖励结束
30.     cancel(bl);
31.     翻转学习子句 L 中未满足的文字；
32.   else                               //未找到冲突
33.     BranchV=max(Act[x_k]);  x_k ∈ V
34.     if  所有变元都有了真值指派  then
35.       return SAT;
36.     end if
37.     dl++;
38.     BranchV 压入 penaltyV;
39.   end if
40. end for
```

AP7 首先做单子句传播，将单子句传播中所有变元压入 *penaltyV* 中，如果这些变元的真值指派之间没有冲突，那么将它们的活跃度降低至 60%-98%。如果这些变元之间有冲突那么除了适当的给予惩罚之外，还给予少量的奖励。以上是传播阶段变元的活跃度计算。

如果单子句传播发现了冲突，且当前搜索层次是第 0 层，那么 SAT 问题无解，返回“*UNSAT*”。如果当前层次不是第 0 层，则 AP7 对冲突进行分析，采用 FUIP 方式产生学习子句，确定回溯层次 *bl*。在学习子句对应的冲突集中出现的变元，其活跃度会得到大小为$(1/0.9)^{numCC}$的奖励。AP7 对构造冲突的变元进行奖励后，开始回溯，撤销第 *bl*+1 层到当前层传播的变元。因为回溯后的学习子句是单子句，所以满足学习子句，进入到新的一轮单子句传播。

如果单子句传播未发现冲突，则 AP7 选择活跃度最大且未赋值的变元作为分支变元。分支层次增加 1，分支变元压入 *penaltyV* 中。AP7 开始新的一轮单子句传播。如果所有变元具有合理的真值指派，没有未赋值的变元，那么 AP7 找到了问题的解，算法返回“*SAT*”。

学习阶段对变元的奖励值不是固定值，该值随着搜索层次的加深，奖励值越大。为了防止活跃度值的溢出，当某个变元的活跃度大于阈值 1E100 时，变元的活跃度整体地平滑下降。搜索阶段实施惩罚时需注意以下几个问题：一、降低活跃度的 *penalty* 不是一个固定值。算法找到越多的冲突，活

跃度降低的越少。这是因为最近找到的冲突很大概率上质量会比之前的冲突要好。二、当搜索找到冲突时，增加了一个变量 $1/(numCC\text{-}lastC[x_t])$。若某个变元一段时间之后，再次找到了冲突，那么它与长时间未有冲突的变元相比，活跃度应该高一些。三、当搜索没有找到冲突时，直接大幅度降低变元的活跃度。虽然这种方式比较野蛮，但是对于复杂的问题，它是有效的。

3 新分支策略的评估

3.1 实验环境和设置

为了准确评估新分支策略在求解工业问题上的作用，实验对比了新算法 AP7 和 glucose3.0。两者仅分支策略不同，AP7 采用的是基于奖惩的新策略，而 glucose3.0 是基于学习过程的变元独立衰减分支策略。在 SAT 竞赛的工业组，glucose 算法及其变种[12]有重要的地位。glucose2.3 算法是 2013 年竞赛的工业组冠军。获得 2014 年~2016 年工业组前三名的算法均是以 glucose3.0 为基础的。同时，glucose 被 SAT 组委会用于确定算例的难度等级。

实验的机器配置采用 Intel E5 2.7 GHz 处理器，32GB 内存，Centos 6.3 服务器版本的操作系统。在相同的机器上，AP7 算法和 glucose3.0 算法求解 2016 年 SAT Competition (http://www.satcompetition.org) 公布的工业算例集。这些算例来自软件测试、调度、硬件电路测试、网络安全、加密算法等实际问题。根据算例的解的满足性，这些算例被分为可满足算例集和不可满足算例集。实验采用 SAT 竞赛的限定时间，每个算例的运算不超过 5000 秒，若超出，该算例结果记为 *unsolved*。

3.2 实验结果

首先给出 glucose3.0 和 AP7 算法在相同机器上计算相同算例的时间结果。表 1 显示了 glucose3.0 和 AP7 算法计算加密 SHA、矩形装配、交通规划等工业问题的运算时间，单位(秒)。表 1 中算例的变元数是 13408~521147，子句数是 308391~13378009。表 1 显示 AP7 算法相比 glucose3.0 计算可满足算例的运行总时间提高了 31.66%。

Table1. Comparison of SAT instances' running time

表 1 可满足算例的运行时间对比

算例名	glucose3.0/s	AP7/s
001-80-12	1783.374	**1745.778**
002-80-12	1652.822	**659.339**
002-80-4	283.076	**198.315**
002-80-8	**621.113**	1009.017
003-80-8	1208.672	**681.527**
004-80-8	450.735	**394.191**
006-80-4	**313.704**	395.040
007-80-8	880.742	**635.395**
008-80-4	**265.648**	449.913
008-80-8	600.966	**487.762**
009-80-4	184.456	**124.684**
009-80-8	935.316	**311.037**
9vliw_m_9stages_iq3_C1_bug3	16.958	**9.713**
9vliw_m_9stages_iq3_C1_bug7	**30.391**	37.258
9vliw_m_9stages_iq3_C1_bug9	75.655	**23.322**
atco_enc1_opt1_03_56	848.849	**17.407**
atco_enc1_opt1_10_15	1308.447	**266.186**
transport-city-sequential-3degree… 2008seed.020...	29.588	**7.290**
transport-city-sequential-3degree… 2008seed.030...	154.362	**35.348**
transport-city-sequential-3degree… 2008seed.050...	191.463	**166.461**
transport-city-sequential-4degree… 2008seed.050...	3481.445	**3149.882**
grid-strips-grid-y-3.055-NOTKNOWN	200.804	**121.526**
gss-18-s100	574.921	**335.892**
gss-19-s100	1970.073	**1081.859**
openstacks-p30_3.085-SAT	1543.116	**35.293**
openstacks-sequencedstrips… p30_3.085-SAT	1541.417	**35.819**
MD5-27-4	**27.064**	58.614

Table2. Comparison of UNSAT instances' running time

表 2 不可满足算例的运行时间对比

算例名	glucose3.0(s)	AP7(s)
smtlib-qfbv-aigs-ext_con_032_008_0256-tseitin	*unsolved*	**0.878**
10pipe_q0_k	**324.461**	331.575
11pipe_k	1891.926	**641.281**
6s165-nonopt	238.564	**78.488**
6s167-opt	524.115	**96.451**
6s168-opt	242.458	**101.478**
6s169-opt	2511.387	**177.528**
7pipe_k	96.272	**40.464**
8pipe_k	200.294	**92.202**
9dlx_vliw_at_b_iq3	156.496	**110.975**
9dlx_vliw_at_b_iq4	324.014	**242.681**
beempgsol2b1	481.567	**355.984**
Beempgsol5b1	406.688	**309.837**
blocks-blocks-36-0.120…	1115.652	**805.624**
blocks-blocks-37-1.130...	874.808	**627.649**
bob12m02-opt	3806.336	**3308.657**
post-cbmc-aes-d-r2-noholes	**925.548**	1008.073
post-cbmc-aes-d-r2	1010.857	**952.85**
rpoc_xits_15_SAT	0.558	**0.131**
q_query_3_L150_coli.sat	**128.853**	186.483
maxxor032	309.920	265.089
velev-vliw-uns-4.0-9	**140.929**	143.284

表 2 显示了 glucose3.0 和 AP7 算法计算硬件电路优化、电子控制等工业问题的运算时间对比。表 2 中算例的变元数是 3295~89315，子句数是 13079~5584002。 glucose3.0 在 5000s 限定时间内，未求解出表 2 的第 1 个算例 smtlib-qfbv-aigs-ext_con_032_008_0256-tseitin，该算例在竞赛网站公布的最好成绩是 Lingeling(druplig)算法的 1.432 秒。对于表 2 中 AP7 和 glcuose3.0 均计算出结果的 21 个算例，AP7 算法的运行总时间比 glucose3.0 缩短了 37.13%。

3.3 实验结果分析

分支数是 SAT 完备算法效率的核心参数。当搜索树分支减少，搜索空间减小，SAT 完备算法的运算时间才会降低。

Table3. Comparison of the number of branch nodes

表 3 算例的分支数的对比

算例名	glucose3.0	AP7	分支数降低比
002-80-12	219859144	**188571304**	14.2%
6s168-opt.cnf	2527635	1492583	40.9%
9vliw_m_9stages_iq3_C1_bug9	4376867	**3091542**	29.3%
atco_enc1_opt1_10_15	6483466	**3040712**	53.1%
transport-city-sequential-3degree…2008seed.030...	1702582	**740187**	56.5%
grid-strips-grid-y-3.055-NOTKNOWN	491662	**413760**	15.8%
openstacks-p30_3.085-SAT	627652	**368956**	41.2%
9dlx_vliw_at_b_iq4	38817464	**53127714**	26.9%
smtlib-qfbv-aigs-ext_con_032_008_0256-tseitin	---	**579400**	---
6s165-nonopt	1963166	**951890**	51.5%

表 3 列出了 glucose3.0 和 AP7 计算的部分算例的分支数以及降低比。降低比=（glucose3.0 分支数-AP7 分支数）/（glucose3.0 分支数）。如果算例在限定时间内未被计算出来，那么其分支数记为---。表中第四列数据显示改进后的 AP7 相比 glucose3.0 可降低分支数 14.2%~56.5%。实验数据显示基于奖惩的分支策略相比仅增加变元活跃度的分支策略可以有效地减少搜索树大小。

为了进一步观察动态奖惩策略对搜索冲突的影响，每增长 1 万个冲突时，算法输出重启的次数。图 4 和图 5 对比了 2 个算法计算相同算例时，相同冲突数下所需的重启次数。图的横轴是冲突数(/万次)，纵轴是重启次数(/次)。glucose3.0 计算 002-80-12.cnf 重启了 103921 次，找到 13841064 次冲突。AP7 则重启了 67927 次，找到冲突 10633140 次。图 1 显示在 0~10633140 次冲突时，2 个算法相应的重启次数。在 0~105 万次冲突搜索中，glucose3.0 比 AP7 重启次数少；在 105 万~1384 万次冲突中，AP7 重启次数少于 glucose3.0。AP7 和 glucose3.0 采用相同的重启策略。实验结果表明综合地评定变元在搜索中的作用，可有效地减少较长时间找不到冲突的情况，从而降低了重启次数。

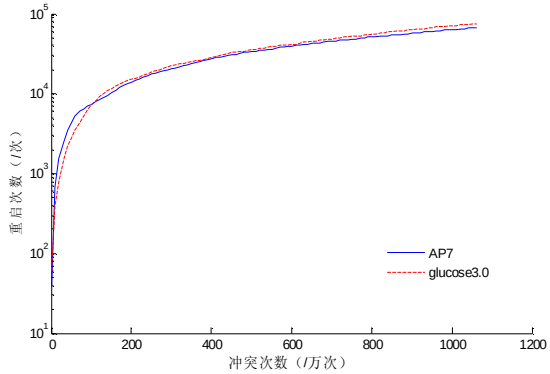

图 4 可满足 002-80-12.cnf 的重启次数和冲突数的对比

Fig.4 Comparison of conflicts and restarts of satisfiable 002-80-12.cnf

glucose3.0 计算不可满足算例 6s168-opt.cnf 的冲突数是 2063558，重启次数是 4758。AP7 计算该算例的冲突数是 1084645，重启次数是 2059。图 5 显示了 6s168-opt.cnf 在 0~120 万次冲突范围内 2 个算法的重启次数的对比，可以看到产生相同个数的冲突时，AP7 的重启次数远低于 glucose3.0 算法的重启次数。因此，基于奖惩的分支策略对于不可满足的工业问题，也可以有效地减小搜索树层次，从而减少重启次数。

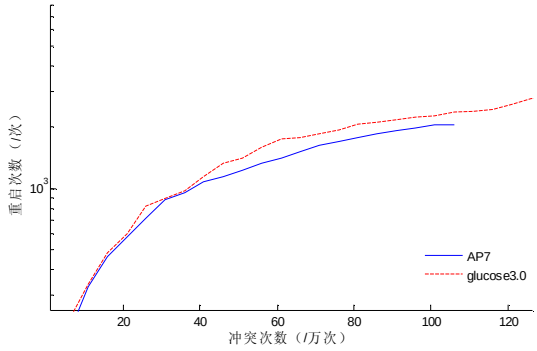

图 5 不可满足 6s168-opt.cnf 的冲突数和重启数的对比

Fig.5 Comparison of conflicts and restarts of unsatisfiable 6s168-opt.cnf

从学习子句质量进一步分析新分支策略的作用。SAT 完备算法每次找到 1 个冲突，产生 1 个学习子句。每个学习子句会计算其相应的 LBD 值，LBD 值为 2 的学习子句是指产生冲突的原因子句分布在搜索树的 2 个层次。因为 LBD 值为 2 的学习子句描述了关系更为紧密的冲突变元之间的关系，所以质量高的这类学习子句一直保留在子句库中，不会被删除。表 4 列出了 glucose3.0 和 AP7 算法正确求解的算例的重启次数，LBD2 学习子句数以及 LBD2 学习子句数与重启次数的比例。若 LBD2 学习子句数与重启次数的比例越大，说明重启后被选择的搜索变元关系更加紧密，导致找到冲突后学习子句的 LBD 值下降。表 4 显示在重启次数较小时（<150）

glucose3.0 算法重启的平均学习质量较高，但是求解复杂的问题（重启次数>150）AP7 算法的平均学习质量较高。这表明综合评定变元可以产生更多高质量学习子句。

算例名	glucose3.0			AP7		
	重启次数	LBD2	LBD2/重启次数	重启次数	LBD2	LBD2/重启次数
002-80-12	103921	10653	0.10	67927	7947	**0.11**
001-80-12	116275	11299	0.09	121041	11423	**0.09**
9vliw_m_9stages_iq3_C1_bug9	315	1566	4.97	43	246	**5.72**
atco_enc1_opt1_10_15	12293	3884	0.31	3730	4579	**1.23**
transport-city-sequential-3degree…2008seed.030...	369	454	**1.23**	115	113	0.98
velev-vliw-uns-4.0-9	3686	5842	1.58	3673	8979	**2.44**
grid-strips-grid-y-3.055-NOTKNOWN	52	791	**15.2**	51	658	12.9
openstacks-p30_3.085-SAT	94	244	**2.59**	60	131	2.18
smtlib-qbfv-aigs-ext_con_032_008_025	---	---	----	60	1140	**19**
6s167-opt	6295	1737	0.27	2283	3271	**1.43**
6s165-nonopt	4186	2093	0.50	2227	3748	**1.68**
9dlx_vliw_at_b_iq4	6288	8511	1.35	5867	14385	**2.45**

Table4. Comparison of the number of restarts and LBD2 clauses

表 4 算例的重启次数、LBD2 子句数的对比

4 结语

工业问题规模大，求解难度高。学习子句数量少或质量不高对搜索的指导意义有限，使搜索易陷入到层次过高，不平衡的困境中。为了缩短求解问题的时间，算法需要尽可能地选择对寻找冲突更有利的变元，以使得搜索树更加平衡。基于动态奖惩的分支策略综合地评估了变元对传播和学习两个主要过程的影响，建立惩罚函数。相比传统、单一的奖励变元评估策略，动态奖惩的分支策略通过调整更关键的变元靠近根节点，从而有效地降低重启次数，提高了学习子句质量，减少搜索树的大小。分支变元策略的改变使得搜索树更加紧凑，将会影响子句删除的标准，后续将对学习子句的删除做新的探索。

参考文献

[1] MARTIN D, GEORGE L M, DONALD L. A machine program for theorem proving [J].Communications of the ACM, 1962, 5(7): 394-397.

[2] MOSKEWICZ M, MADIGAN C, ZHAO Y, et al. Chaff: engineering an efficient sat solver[C]// ACDA 2001: Proceedings of the 38th Design Automation Conference Design Automation. Piscataway, NJ: IEEE, 2001:530 - 535.

[3] CARLOS A, MARIA L, JORDI L. SAT-based MaxSAT algorithms [J]. Artificial Intelligence, 2013, 196:77-105.

[4] KOSHIMURA M, ZHANG T, FUJITA H, et al. QMaxSAT: a partial Max-SAT solver system description [J]. Journal on Satisfiability, Boolean Modeling and Computation,2012, 8: 95-100.

[5] ZHANG L, MADIGAN C F, MOSKEWICZ M H. Efficient conflict driven learning in a boolean satisfiability solver [C]// ICCAD 2001: Proceedings of the 2001 IEEE/ACM International Conference on Computer-Aided Design. Piscataway, NJ: IEEE, 2001:279-285.

[6] LIU Y L,LI C M, HE K, et al. Breaking cycle structure to improve lower bound for Max-SAT[C]//FAW 2016: Proceeding of the 11th International Frontiers of Algorithmic Workshop. Lecture notes in computer science. Berlin：Springer, 2016: 111-124.

[7] AUDEMARD G, SIMON L. Predicting learnt clauses quality in modern SAT solvers[C]// IJCAI 2009: Proceeding of the 2009 International Joint Conference on Artificial Intelligence. Piscataway, NJ: IEEE, 2009:399-404.

[8] GILLES A,JEAN M L,LAURENT S. Improving glucose for incremental SAT solving with assumption[C]//SAT 2013: Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing. Berlin :Springer ,2013: 7962(8):309-317.

[9] DE K E. Aspects of Semidefinite Programming [M]. Berlin: Springer, 2002: 211-228.

[10] 刘燕丽,李初民,何琨. 基于优化冲突集提高下界的 MaxSAT 完备算法[J]. 计算机学报, 2013, 10(36): 2087-2096.(LIU Y L, LI C M, HE K. Improved lower bounds in MAXSAT complete algorithm based optimizing inconsistent set [J]. Chinese Journal of Computers, 2013, 10(36): 2087-2096.)

[11] 刘燕丽,黄飞,张婷.基于环型扩展推理规则的 MaxSAT 完备算法[J]. 南京大学学报,2015,51(4):762-771.(LIU Y L,HUANG F, ZHANG T. MaxSAT complete algorithm based cycle extended inference rules[J]. Journal of Nanjing University (Natural Science),2015, 51(4):762-771.)

[12] GILLES A, LAURENT S. Lazy clause exchange policy for parallel SAT solvers [C]//SAT 2014: Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing. Berlin:Springer, 2014: 8561:197-205

This work is partially supported by the Science and Technology Project of Hubei Provincial Education Department (B2016015).

LIU Yanli, born in 1980, lecturer, M. S. Her research interests include Algorithm design of NP problems, algorithm optimization.
XU Zhenxing, born in 1990, Ph. D. candidate. His research interests include approximate algorithm of NP problems, algorithm optimization.
Xiong Dan, born in 1979, lecturer, M. S. . Her research interests include Mathematical Statistics, System Identification.