CN 43-1258/TP
ISSN 1007-130X

计算机工程与科学
Computer Engineering & Science

第 41 卷第 3 期 2019 年 3 月
Vol. 41, No. 3, Mar. 2019

文章编号:1007-130X(2019)03-0490-08

基于动态奖惩的 CDCL SAT 求解器分支启发式算法*

陈秀兰,刘　婷

(西南交通大学系统可信性自动验证国家地方联合工程实验室,四川 成都 610031)

摘　要:分支启发式算法在 CDCL SAT 求解器中有着非常重要的作用,传统的分支启发式算法在计算变量活性得分时只考虑了冲突次数而并未考虑决策层和冲突决策层所带来的影响。为了提高 SAT 问题的求解效率,受 EVSIDS 和 ACIDS 的启发,提出了基于动态奖惩 DRPB 的分支启发式算法。每当冲突发生时,DRPB 通过综合考虑冲突次数、决策层、冲突决策层和变量冲突频率来更新变量活性得分。用 DRPB 替代 VSIDS 算法改进了 Glucose 3.0,并测试了 SATLIB 基准库、2015 年和 2016 年 SAT 竞赛中的实例。实验结果表明,与传统、单一的奖励变量分支策略相比,所提分支策略可以通过减少搜索树的分支和布尔约束传播次数来减小搜索树的规模并提高 SAT 求解器的性能。

关键词:SAT 问题;分支启发式算法;VSIDS;决策层;冲突决策层;变量冲突频率

中图分类号:TP181　　**文献标志码**:A

doi:10.3969/j.issn.1007-130X.2019.03.015

A dynamic reward and punishment based branching heuristic algorithm for CDCL SAT solvers

CHEN Xiu-lan, LIU Ting

(National-Local Joint Engineering Laboratory of System Credibility Automatic Verification, Southwest Jiaotong University, Chengdu 610031, China)

Abstract: Branching heuristic algorithms play an important role in CDCL SAT solvers, and the conventional branching heuristic algorithms are more concerned with the number of conflicts when calculating the activity scores of variables and do not consider the influence of the decision level or conflict decision level. In order to improve the efficiency of solving SAT problem, we propose a dynamic reward and punishment based branching method (DRPB), which is inspired by the EVSIDS and ACIDS. Whenever a conflict occurs, the DRPB updates the activity score of variables by integrating the numbers of conflicts, decision-making level, conflict decision level, and conflict frequency of variables. We improve the Glucose version 3.0 by replacing the VSIDS algorithm with the DRPB, and conduct an empirical evaluation not only on instances of SATLIB benchmarks, but also on 2015 and 2016 SAT competitions. Experimental results show that compared with the traditional and single variable branching heuristic algorithms, the proposed strategy can reduce the size of the search tree and improve the performance of the SAT solvers by reducing the branches of the search tree and the number of Boolean constraint propagation.

Key words: SAT problem; branching heuristic algorithm; VSIDS; decision level; conflict decision level; conflict frequency of variable

* **收稿日期**:2018-05-24;**修回日期**:2018-07-13
基金项目:国家自然科学基金(61673320)
通信地址:610031 四川省成都市西南交通大学系统可信性自动验证国家地方联合工程实验室
Address: National-Local Joint Engineering Laboratory of System Credibility Automatic Verification, Southwest Jiaotong University, Chengdu 610031, Sichuan, P. R. China

1 引言

布尔可满足性问题 SAT(Boolean SATisfiability problem)是通过将布尔真值赋给公式中的变量来确定以合取范式 CNF(Conjunctive Normal Form)形式给出的命题公式的真值是否为真。1971 年,SAT 问题是首个在时间计算复杂度上被证明的 NP(Non-deterministic Polynomial)完全问题[1]。NP 问题是指用确定性算法可以在多项式时间内被检查或验证的问题。SAT 问题在定理证明、人工智能、模型检验、软件验证、电子设计自动化、计算机辅助设计等许多领域发挥了至关重要的作用。由于 SAT 问题的实际重要性,建立一个快速的 SAT 求解器尤为关键。

在过去的二十多年中,许多主流的 SAT 求解器被开发出来,比如 GRASP[2]、real-sat[3]、SATO[4]、Chaff [5]、Berkmin[6]、MiniSat [7]和 Glucose[8]等,这些求解器主要基于 DPLL(Davis Putnam Logemann Loveland)[9]算法框架。而影响最大的则是在 DPLL 的基础上形成的目前最为流行的冲突驱动子句学习 CDCL (Conflict-Driven Clause Learning)[10]求解框架,并且基于 CDCL 的 SAT 求解器在近年也取得了巨大的发展。

CDCL 求解器包含许多解决问题的关键要素,例如预处理、变量分支启发式算法、布尔约束传播 BCP(Boolean Constraint Propagation)、冲突分析、子句学习、非时序回溯和重启等,其中促使 CDCL 求解器取得成功的一个最关键因素则是变量分支启发式算法。所有分支算法的核心思想都是为每个决策文字(或变量)提供一个评估算子,并且选择具有最大(或最小)得分的文字(或变量)作为新的分支方向。分支启发式算法可以大致分为两类,即静态分支启发式算法和动态分支启发式算法。静态分支启发式算法主要根据子句集本身所固有的结构信息来设计相应算法,例如变量个数、子句长度等,但是却没有与冲突分析过程进行结合。具有代表性的静态分支启发式算法包括 JW(Jeroslow-Wang)[11]、MOM(Maximum Occurrences in Minimization)[12]和 DLIS(Dynamic Literal Individual Sum)[2]等。而动态分支启发式算法在选择分支变量时不仅考虑了子句集本身的固有信息,还与冲突分析过程进行了有效结合,具有代表性的动态分支启发式算法包括 VSIDS(Variable State Independent Decaying Sum)[5]、NVSIDS (Normalized VSIDS)[13]、EVSIDS(Exponential VSIDS)[7]、ACIDS(Average Conflict-Index Decision Score)[14]、CHB(Conflict History-based)[15]和 LRB(Learning Rate Branching)[16]等。但是,这些动态分支启发式算法并不完美,它们的缺点是过多地依赖迄今为止所发生的冲突次数,而忽略了每一次冲突发生时变量所属的决策层。

因此,本文提出了一种基于动态奖惩的分支启发式算法,称为 DRPB(Dynamic Reward and Punishment Based)算法。该算法的主要思想是在每次发生冲突时,对未参与冲突分析的变量给予适当的惩罚,而对参与了冲突分析的变量给予一定的奖励。具体的奖励值是结合冲突次数、变量决策层、冲突决策层和变量冲突频率给出的。对比实验表明,该算法不仅减小了搜索树的规模,还提高了 SAT 求解器的性能。

2 SAT 问题及求解算法

2.1 SAT 问题

布尔公式是由布尔变量与布尔逻辑运算符($\neg$,$\wedge$,$\vee$)连接的逻辑公式。给定一组有限命题变量集合 $X=\{x_1,x_2,\cdots,x_n\}$,$x_i(1\leqslant i\leqslant n)$可被赋值为真(1)或假(0)。文字 l_i 是变量 x_i 或其否定 $\neg x_i$(或$\overline{x_i}$)。子句 C 是若干文字的析取(如 $C=x_1\vee\neg x_2\vee x_3$),子句的长度是它所包含的文字的个数,记为$|C|$,其中子句长度为 1 的被称为单元子句(如 x_4),子句长度为 0 的被称为空子句,记为□。合取范式 CNFφ 是若干子句的合取,例如 $\varphi=(x_1\vee x_2)\wedge(\neg x_2\vee x_3)\wedge(x_1\vee x_3\vee\neg x_4)\wedge(x_2\vee\neg x_3\vee\neg x_4)$,它包含 4 个变量和 4 个子句。

一个子句为真当且仅当其中至少一个文字为真。一个子句为假当且仅当其中所有文字为假。只有当所有的子句同时为真(即可满足)时,CNF 公式才可满足。空子句总是不可满足的,但空公式总是可满足的。因此,SAT 问题是通过为公式中的变量赋布尔真值来确定 CNF 公式是否可满足。

2.2 VSIDS 算法

负责选择分支变量的算法称为分支启发式算法。当前最先进的分支启发式算法是由 Chaff 求解器[5]的作者提出的,称为变量状态独立衰减总和 VSIDS。VSIDS 算法的具体步骤如下:

(1)在求解器运行开始时,每个文字都有一个被称为活性(Activity)的计数器,并且所有变量的

活性分数通常被初始化为0。

(2)当求解器学习到一个子句时,学习子句中的每个文字的活性分数会加1,这个增量1被称为"碰撞"(Bump)。

(3)在每次选择分支变量时求解器会选择活性分数最高的未赋值文字。而当一些文字的活性分数相同时,将会随机选择一个文字进行赋值。

(4)所有文字的活性分数会定期地除以一个大于1的常数α,α被称为求解器运行过程中的衰减因子(Decay Factor)。

2.3 VSIDS算法的改进

由于目前使用最多、影响最大的分支启发式算法是VSIDS算法,许多研究人员在VSIDS的基础上作出了一些高效的改进,取得了许多进展,如NVSIDS[15]、EVSIDS[7]、ACIDS[16]、CHB和LRB等。本文是在EVSIDS和ACIDS的基础上进行改进的,故只介绍这2种算法。

2.3.1 EVSIDS算法

EVSIDS算法是Eén在2008年提出的。在EVSIDS算法中,正文字和负文字拥有相同的计数器,因此减少了计数器的个数。该算法中的活性分数增量不再是1,而是将冲突次数考虑进去,具体的分数改变如下:

$$s' = s + g^n \tag{1}$$

其中,$g = 1/f$,f是阻尼因子(Damping Factor)且$0 < f < 1$,n是冲突次数。与冲突无关(即未参与冲突分析)的其他变量的活性分数保持不变。

2.3.2 ACIDS算法

2015年,Biere等人提出了另一种VSIDS改进算法,称为ACIDS算法。和EVSIDS算法一样,ACIDS算法为每个变量保留一个计数器,但是它的活性分数的改变不同于VSIDS算法和EVSIDS算法。在ACIDS算法中,每当冲突发生时,与冲突相关的变量的分数以如下方式更新:

$$s' = \frac{s+n}{2} \tag{2}$$

其中,n是冲突次数,且未参与冲突分析的变量的活性分数保持不变。ACIDS算法通过在增量上考虑与冲突次数的关系,使得与最近冲突相关的变量具有更大的活性分数。

在上述2种启发式算法中,参与了冲突分析的子句中所包含的变量会在冲突发生时增加其活性分数。但是,无论是EVSIDS算法还是ACIDS算法,在同一层冲突决策层中,与冲突相关的不同变量均会有相同的增量。因此,这2种算法仅仅考虑了冲突次数对变量活性分数增量的影响,还有一些其他的重要因素并未考虑,而这些因素在对分支变量进行选择时也会有重大影响。故本文设计了一种新的分支启发式算法,希望在同一次冲突中(即冲突决策层相同),拥有不同决策层和冲突频率的变量的增量是不同的。

3 基于动态奖惩的分支启发式算法

3.1 DRPB算法

本节提出一种新的分支启发式算法,称为DRPB算法。在DRPB算法中,参与冲突分析的变量会得到相应奖励,而对未参与冲突分析的变量会给予适当惩罚。DRPB算法的伪代码如算法1所示。

算法1 DRPB branching heuristic

Input: A CNF formula F。

Output: The satisfiability of F。

```
1  if UnitPropagation( )==conflict then
2    if decisionlevel==0 then
3      return Unsat
4    end if
5  end if
6  decisionlevel=0; activity score s=0;V←all variables in F;
7  conflict number n=0;p=0.7;/* p is the punishment value */
8  while not AllVariableAssigned( ) do
9    PickBranchingVariable( );
10   decisionlevel ++;
11   if UnitPropagation( )==conflict then
12     backtrack level blevel==ConflictAnalysis( );
13     n++;
14     conflict decision level d=decisionlevel;
15     decisionlevel=blevel;
16     C←variables in conflict analysis;
17     d_v←decision level of variable v when it is assigned
18     n_v←the latest number of conflict of variable v;
19     for each v ∈ C do
20       s' = (s+n_v)/2 + n^((d-d_v)/d);
21     end for
22     for each V\C do
23       s' = ps + (1-p) n_v/n
24       if p<0.99 then
```

```
25            p = p + 10^-7
26          end if
27        end for
28      end if
29      if blevel<0 then
30        return Unsat
31      else BacktrackTo(blevel)
32      end if
33    end while
```

DRPB 算法的细节详述如下:在搜索开始之前,为每个变量设置一个初始化为 0 的浮点数 s,称为活性分数(Activity Score)。算法首先进行单位传播(UnitPropagation),如果出现冲突并且当前决策层为 0,则该 SAT 问题是不可满足的。若未在决策层为 0 时发生冲突,那么在变量决策阶段,将通过函数 *PickBranchingVariable*()来选择具有最高活性得分的未赋值变量作为新的分支变量,并记录该变量所在的决策层。之后,布尔约束传播(BCP)过程将根据新的分支变量来推断其他变量的赋值,并且这些变量的决策层与该分支变量相同。接着,当冲突发生时,函数 *ConflictAnalysis*()将分析冲突并通过唯一蕴涵点 FUIP(First Unit Implication Point)[17]产生学习子句并记录此时产生冲突的冲突决策层(Conflict Decision Level)d。为了体现最新冲突的重要性,EVSIDS 算法和 ACIDS 算法都将到目前为止所发生的冲突次数融入变量活性得分计算中,但并未全面考虑其他的一些重要因素对活性得分的影响,使得与每一次冲突相关的不同变量的得分会加上相同的增量。因此,为了更好地区分此类变量在当前冲突下的重要性,本文在计算活性得分时不仅考虑了冲突次数 n,还考虑了变量冲突频率 n_v、冲突发所生的决策层 d 和变量决策层 d_v。这一改进思路全面评估了与冲突相关的变量所拥有的潜在信息,加大了变量对搜索的作用,并通过增加更易产生冲突和更接近树根的变量被选择的概率使冲突能够提前发生,从而提高求解效率。

无论何时参与冲突分析的变量,其活性得分都会按照式(3)进行更新。

$$s' = \frac{s + n_v}{2} + n^{\frac{d-d_v}{d}} \tag{3}$$

其中,具有较低决策层(即越接近根节点)和较高冲突频率的变量的得分增量会相对大于具有较高决策层和较低冲突频率的变量的得分增量,因此新的分支变量有更多的可能会选择具有较低决策层和较高冲突频率的变量。

在以往的分支启发式算法中,活性得分的增加仅针对参与了冲突分析的变量,对于未参与冲突分析的变量并不改变其活性得分。某些情况下,相似的学习过程会产生相似度较高的学习子句,由于历史信息比较集中,因此仅有少量变量可以优先选择。若在可选的少量变量中有未参与冲突分析的变量,由于其活跃度不变可能会导致算法陷入困境,为逃脱此困境程序会进行动态重启,但是重启只是缓和了问题,并不会改变变量的活性得分。因此,对于那些与冲突无关的变量,需降低其活跃度,从而避免陷入相同困境。

当变量未参与冲突分析时,其当前活性得分将根据式(4)降低。

$$s' = ps + (1-p)\frac{n_v}{n} \tag{4}$$

其中,p 是惩罚值,$0.7 \leqslant p < 0.99$。就是说,如果变量没有参与到当前的冲突分析过程中,它们的活性分数便会降低到原来的 70%~99%。随着冲突次数的增加,变量分数下降会越来越少。另外,冲突频率较高的变量的得分降低程度会相对低于冲突频率较低的变量的。

最后,在冲突分析完之后,回溯层 *blevel* 会被确定,并且 DRPB 算法将撤销从 *blevel*+1 到当前决策层的所有变量赋值。如果回溯层小于 0,则该 SAT 问题是不可满足的。

3.2 示例

我们将通过一个例子来更直观地描述 DRPB 算法,并且给出 EVSIDS、ACIDS 和 DRPB 这 3 种算法的变量活性得分的比较分析。

令 $F = \{x_8 \vee x_2, x_9 \vee \neg x_{11} \vee \neg x_{12}, \neg x_5 \vee \neg x_{11} \vee \neg x_2, x_{11} \vee x_3 \vee \neg x_{10}, x_4 \vee \neg x_1, \neg x_1 \vee x_7, \neg x_6 \vee \neg x_7 \vee \neg x_4 \vee \neg x_3, x_8 \vee \neg x_1 \vee x_6, \neg x_9 \vee x_5, \cdots\}$,并假设 x_1 为第 10 个分支文字,F 的蕴涵图如图 1 所示。

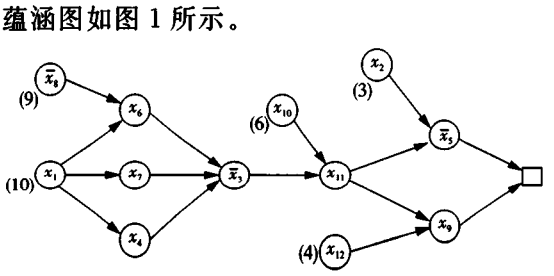

Figure 1 Implication graph of F

图 1 F 的蕴涵图

从图 1 中可以看出,文字 $\neg x_8$、x_2、x_{10} 和 x_{12} 的决策层分别是 9、3、6 和 4,蕴涵图中的其他文字的决策层都和 x_1 相同,均为 10。□表示冲突子句

$\neg x_9 \vee x_5$,故公式 F 的当前冲突发生在第 10 层冲突决策层下。通过冲突分析可得到学习子句 $\neg x_2 \vee x_3 \vee \neg x_{10} \vee \neg x_{12}$。因此,出现在冲突侧和学习子句中的变量(即参与冲突分析的变量)有 x_2、x_3、x_5、x_9、x_{10}、x_{11} 和 x_{12},而未参与冲突分析的变量(即在原因侧的变量)有 x_1、x_4、x_6、x_7 和 x_8。

假设公式 F 到目前为止共发生了 150 次冲突,表 1 中列出了每个变量的最新冲突次数。

假设在第 150 次冲突之前变量 x_2 和 x_7 的活性分数分别为 1 000 和 436,记作 $s_{x_2}=1000, s_{x_7}=436$,令 $p=0.7$,则 x_2 的新得分为 $s'_{x_2}=(s_{x_2}+83)/2+150^{\frac{10-3}{10}}\approx 576.86$,$x_7$ 的新得分为 $s'_{x_7}=0.700015\times s_{x_7}+(1-0.700015)\times(109/150)\approx 305.42$。用 s_i 表示变量 x_i 在第 149 次冲突时的活性得分,用 s_i^{E}、s_i^{A}、s_i^{D} 分别表示分支启发式算法为 EVSIDS($f=0.99$)、ACIDS 和 DRPB 时在第 150 次冲突发生后变量 x_i 的新分数。变量活性得分如表 2 所示。

从表 2 可以看出,对于方框中未参与冲突分析的变量,它们的新得分 s_i^{E}、s_i^{A} 与 s_i 相比不会发生改变。但是,在 DRPB 算法中,s_i^{D} 会因受到惩罚而有所降低。比如,在第 149 次冲突时,变量 x_6 和 x_7 的得分相同,但在第 150 次冲突时,它们的得分由于各自不同的冲突次数而有所不同。对于不在方框内的其他变量(即参与了冲突分析的变量),EVSIDS 算法中的分数增量是相同的,均为 4.52,而 ACIDS 算法和 DRPB 算法的主要区别在于 DRPB 算法不仅根据冲突次数改变变量得分,而且还考虑了每个变量的决策层、冲突决策层和变量最新冲突次数。

4 实验结果

MiniSat 是一个非常著名的 SAT 求解器,其他许多表现优异的求解器,比如 Glucose 等都是基于 MiniSat 的改进版本,但是在最近几年,MiniSat 已经很少在 SAT 竞赛中获奖。Glucose 成为目前最流行的 SAT 求解器,而且近几年的 SAT 竞赛中大部分获奖算法是以 Glucose 为基础的,除此之外,SAT 组委会还使用 Glucose 来确定算例的难度等级。在 Glucose 的许多版本中,Glucose 3.0 又是极为重要的一个版本,因此本文以 Glucose 3.0为基础,嵌入了 DRPB 算法并保持求解器中的其他部分不变,称新版本为 Glucose+DRPB。实验的硬件环境为 Intel i5-2450 2.50 GHz CPU,4 GB 物理内存,Windows 7 操作系统。

4.1 SATLIB 实验结果

表 3 给出 Glucose 3.0 和 Glucose+DRPB 在相同实验环境下求解 SATLIB 基准库中北京竞赛问题集(SAT Competition Beijing)的部分结果。该问题集共有 16 个算例,可大致分为 6 类,各类的算例个数由括号中的数字标出。其中"*Vars*""*Claus*""*Confs*""*Decis*"和"*Props*"分别表示各算例的变量数、子句数、冲突次数、决策次数和布尔约束传播次数。

实验发现对于 2bitadd 系列中的 3 个算例,Glucose+DRPB 的求解效果不如 Glucose 3.0 的,但是 Glucose+DRPB 在求解 SAT Competition Beijing 问题集中剩余的 13 个算例时所发生的冲突次数、决策次数和布尔约束传播次数等于或者小于 Glucose 3.0 的,因此减小了搜索树的规模。通过表 3 可以看出,Glucose+DRPB 求解变量数和子句数较多的算例的效率优于 Glucose 3.0 的,并且从 Glucose 3.0 的求解过程可以发现,布尔约束传播过程往往会花费大量的时间开销,因此降低布尔约束传播次数对于提高求解器的性能非常重要。

Table 1 Quantity of conflicts of variable x_1 to x_{12} at the 10th conflict decision level

表 1 第 10 层冲突决策层变量 x_1 到 x_{12} 的冲突次数

variable	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}
conflicts	150	83	113	85	150	133	109	120	138	125	130	90

Table 2 Variable activity score ($n=150, d=10, p=0.7$)

表 2 变量活性得分($n=150, d=10, p=0.7$)

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}
s_i	330	1 000	387	500	306	436	436	484	306	830	340	950
s_i^{E}	330	1 004.52	391.52	500	310.52	436	436	484	310.52	834.52	344.52	954.52
s_i^{A}	330	575	268.5	500	228	436	436	484	228	490	245	550
s_i^{D}	231.3	576.86	251	350.18	229	305.47	305.42	339.05	223	484.92	236	540.21

Table 3 Results of the two solvers for solving the benchmark—SAT Competition Beijing

表 3 两个求解器求解问题集 SAT Competition Beijing 的结果

Instances	*Vars*	*Claus*	Glucose 3.0			Glucose+DRPB		
			Confs	*Decis*	*Props*	*Confs*	*Decis*	*Props*
2bitadd_11 (3)	649	1 562	**1 045**	**1 950**	**32 806**	1 167	2 046	34 980
2bitcomp_5 (1)	125	310	0	58	125	0	58	125
2bitmax_6 (1)	252	766	0	137	252	0	137	252
3bitadd_32 (2)	8 704	32 316	31 980	58 739	3 363 343	**11 175**	**48 233**	**1 864 555**
4blocksb (3)	410	24 758	475	2 806	23 979	**158**	**934**	**10 236**
e0ddr2-····-5-1 (6)	19 500	103 887	963	6 748	293 690	**270**	**5 971**	**100 663**

表 4 列出了 Glucose 3.0 和 Glucose+DRPB 这 2 个求解器求解 SATLIB 中其他系列的总求解时间,系列名后括号内的数字是该系列的算例个数。

Table 4 Solving time for partial benchmarks from SATLIB

表 4 求解 SATLIB 部分算例的时间

Benchmarks	*Vars*	*Claus*	Glucose 3.0	Glucose+DRPB
			Time/s	
flat50-115 (1 000)	150	545	**5.891**	6.091
RTI_k3···429 (500)	100	429	**3.236**	3.391
CBS_k3···b10 (1 000)	100	403	7.05	**6.562**
bw_large.d	6 325	131 973	0.405	**0.343**
logistics.d	4 713	21 991	0.031	0.031
qg1-08	512	148 957	0.202	**0.156**

如表 4 所示,由于前 3 个系列中不同算例的变量数和子句数是相同的,故共给出了 2 500 个算例的总求解时间。在后 3 个系列中不同算例的变量数和子句数不同,故只选择了一个算例作为代表。由表 4 可以看出,Glucose+DRPB 求解变量数和子句数相对较多的算例的时间少于 Glucose 3.0 的。只有当搜索树的分支减少,搜索的空间减小时,运算时间才会有所降低。

4.2 SAT 竞赛实验结果

接着给出 Glucose 3.0 和 Glucose+DRPB 在相同的实验环境下分别求解 2015 年和 2016 年 SAT 竞赛中 main track 组的相同测试例的实验结果,其中 2015 年和 2016 年的 main track 组的测试例各为 300 个,这些测试例来自网络安全、密码验证、路径规划等领域。对于每个实验测试例,Glucose 3.0 和 Glucose+DRPB 均无预处理且采用国际 SAT 竞赛的限定时间,即每个测试例的运行时间均不超过 5 000 s,若超出,则该测试例的实验结果记为 unsolved,用"—"表示,并且所有的测试例均求解 2 次以进行两两比较。实验结果表明,Glucose 3.0 和 Glucose+DRPB 都可以正常运行,并且得到的结果都是正确的。

表 5 列出了 Glucose 3.0 和 Glucose+DRPB 求解 2015 年和 2016 年 SAT 竞赛的一些可满足(Sat)和不可满足(Unsat)实例的运算时间和布尔约束传播次数的比较,共 24 个实例。

表 5 显示,在 Glucose 3.0 和 Glucose+DRPB 均可求解的 17 个实例中,Glucose+DRPB 有 12 个实例的运算时间比 Glucose 3.0 的更少,总的运算时间缩短了 35.3%。其中,运算时间更少的有 7 个可满足实例和 5 个不可满足实例,时间相比 Glucose 3.0 分别缩短了 40.6%和 25.6%。另外,有 5 个实例的求解时间超过了 Glucose 3.0,可以发现这 5 个实例的变量数和子句数均相对较少,此结果说明 Glucose+DRPB 在求解变量数和子句数相对较少的实例时,其效率低于 Glucose 3.0。除此之外,还存在 Glucose 3.0 没有求解出来的 7 个实例,但 Glucose+DRPB 求解出了结果,且时间较短。

本文所提出的变量活性加权计算相比目前主流求解器的计算方法更为复杂,尽管新方法的计算开销较大,但是它所选择的变量比 VSIDS 算法选择的变量更好。就是说,若忽略计算分支变量的时间,这种贪婪算法能求解出比 VSIDS 算法更多的实例。并且我们知道,在求解过程中布尔约束传播过程会花费大量的时间开销,通过表 5 可以看出 12 个运算时间更短的实例,它们相应的布尔约束传播次数也比 Glucose 3.0 的更少。因此,DRPB 算法可以通过选择更好的分支变量来减少布尔约束传播次数,减小搜索树的规模,进而提升求解效率。

表 6 和表 7 分别列出了 2015 年和 2016 年 SAT 竞赛应用类的测试例中,Glucose 3.0 和 Glucose+DRPB 求解出的实例个数、总运算时间和平均运算时间。

Table 5 Comparison of running time and propagations for Sat and Unsat instances among different solvers

表 5 不同求解器求解可满足和不可满足实例的运算时间和布尔约束传播次数比较

		Instances (*Vars* / *Claus*)	Glucose 3.0		Glucose+DRPB	
			Time/s	*Propagations*	*Time*/s	*Propagations*
SAT 2015	Sat	002-80-4 (13408/308391)	270.63	505 338 357	**186.23**	**478 991 778**
		50bits_14… (834/19141)	—	—	381.09	153 800 442
		gss-19… (31435/94548)	1 551.78	7 546 510 055	**619.52**	**3 016 963 775**
		jgiral…. 188 (2200/9086)	—	—	1 023.6	885 042 589
		mr…8x8#18_20 (9880/8398)	**395.33**	**855 132 242**	467.95	1 051 032 985
		vmpc_33 (1089/177375)	1 993.33	1 453 142 158	**371.56**	**335 345 719**
	Unsat	6s16 (30823/89898)	1 335.62	2 069 533 774	**984.83**	**1 054 178 630**
		46bits_12… (697/17080)	**176.22**	**103 582 182**	377.85	178 847 229
		11pipe_k (89315/5584003)	—	—	992.3	4 353 430 797
		beem…l2b1 (26455/76534)	561.61	1 969 570 543	**445.1**	**1 392 388 123**
		bob12s02 (26294/77920)	—	—	2 773.03	5 674 365 719
		manth…31_6 (4521/15036)	**38.59**	**134 550 014**	55.1	176 579 164
SAT 2016	Sat	snw_13_9_pre (20446/689272)	299.41	403 283 615	**94.21**	**112 212 551**
		ibm-20…90 (222291/928885)	1 142.69	1 943 523 815	**763.69**	**1 089 134 039**
		modgen…11953 (2200/9086)	—	—	42.53	38 319 695
		aes_32_3...2 (708/2664)	**483.04**	**85 594 767**	763.82	95 646 625
		mizh…47-3 (65604/273522)	402.92	1 568 090 702	**206.89**	**415 451 782**
		servers…923 (106856/322046)	2 841.54	5 226 438 752	**1 762.36**	**4 769 666 026**
	Unsat	10pipe_k (67300/3601247)	—	—	863.43	2 206 372 683
		arcfo...6_14 (105743/405350)	894.32	1 676 573 523	**758.47**	**1 397 314 306**
		sok…20…21(105424/921218)	1 617.92	1 917 411 138	**1 493.18**	**1 587 118 187**
		korf-18 (7794/186934)	—	—	2 537.54	1 101 974 573
		uum8.sm…212 (1006/3359)	**40.76**	**43 085 486**	57.79	46 481 952
		cube…sat (455627/1367522)	900.55	1 450 672 862	**266.43**	**420 847 406**

Table 6 Comparison for solving Sat and Unsat instances in SAT 2015 among different solvers

表 6 不同求解器在 2015 年 SAT 竞赛中求解可满足和不可满足实例的情况对比

SAT 2015	Glucose 3.0			Glucose+DRPB		
	Sat	Unsat	Total	Sat	Unsat	Total
Number of Solutions	128	92	220	138(+7.8%)	97(+5.4%)	235(+6.8%)
Total Time/s	89 728.5	61 648.5	151 377.0	85 562.3	54 428.6	136 690.9
Average Time/s	701.0	670.1	688.1	620.0	561.1	581.7

Table 7 Comparison for solving Sat and Unsat instances in SAT 2016 among different solvers

表 7 不同求解器在 2016 年 SAT 竞赛中求解可满足和不可满足实例的情况对比

SAT 2015	Glucose 3.0			Glucose+DRPB		
	Sat	Unsat	Total	Sat	Unsat	Total
Number of Solutions	50	65	115	55(+10%)	70(+7.7%)	125(+8.7%)
Total Time/s	36 672.7	37 269.2	73 941.9	32 507.8	38 522.7	71 030.5
Average Time/s	733.5	573.4	643.0	591.1	550.3	568.2

表 6 和表 7 显示,在 2015 年和 2016 年的 SAT 竞赛中,无论是可满足的、不可满足的还是全部的求解情况,Glucose+DRPB 均比 Glucose 3.0 求解出了更多的实例。在总的求解个数方面,2015 年的 SAT 竞赛中,Glucose+DRPB 比 Glucose 3.0多求解出了 15 个实例,2016 年多求解出了 10 个实例。另外,两个求解器之间的求解个数差异在表格中以百分比的形式给出。比如,与 Glucose

3.0相比,Glucose+DRPB 在 2016 年 SAT 竞赛中多求解出了 8.7%的实例。而且 Glucose+DRPB 在 2015 年的 SAT 竞赛中总共求解出 235 个实例,平均运算时间为 581.7 s,这比 Glucose 3.0 的 688.1 s的平均运算时间更短。

上述实验均说明基于动态奖惩的分支策略和仅仅只增加变量活性分数的分支策略相比有着更少的运算时间,这也直接提高了求解器的性能。

5 结束语

本文首先介绍了 VSIDS 算法以及它的 2 个改进版本 EVSIDS 算法和 ACIDS 算法,传统的分支启发式算法在计算变量活性得分时更关注冲突次数,而忽略了其他一些重要因素。之后提出基于动态奖惩的分支策略 DRPB 算法。DRPB 算法受 EVSIDS 算法和 ACIDS 算法的启发,保留每个变量的活性计数器,但更多地考虑冲突次数、决策层、冲突决策层和变量冲突频率之间的关系。此外,通过一个例子说明 EVSIDS、ACIDS 和 DRPB 3 种算法的变量活性得分差异。最后对 Glucose 3.0 进行了改进,不仅测试了 SATLIB 中的实例,还测试了 2015 年和 2016 年 SAT 竞赛中的实例。实验结果表明,与传统、单一的奖励变量分支策略相比,基于动态奖惩的分支策略可以通过减少搜索树的分支和布尔约束传播次数,从而减小搜索树的规模,减少 CDCL SAT 求解器的运算时间,提高求解器的性能。

参考文献:

[1] Cook S A. The complexity of theorem-proving procedures [C]// Proc of the 3rd ACM Symposium on Theory of Computing, 1971: 151-158.

[2] Marques-Silva J P, Sakallah K A. GRASP: A new search algorithm for satisfiability [C]// Proc of the 1996 IEEE/ACM International Conference on Computer-aided Design, 1996: 220-227.

[3] Bayardo R, Schrag R. Using CSP look-back techniques to solve real-world SAT instances [C]// Proc of the 14th National Conference on Artificial Intelligence, 1997: 203-208.

[4] Zhang H. SATO: An efficient propositional prover [C]// Proc of International Conference on Automated Deduction, 1997: 272-275.

[5] Moskewicz M W, Zhao Y, Madigan C F. Chaff: Engineering an efficient SAT solver [C]// Proc of the 38th Annual Design Automation Conference, 2001: 530-535.

[6] Goldberg E, Novikov Y. BerkMin: A fast and robust sat-solver [C]// Proc of Design, Automation and Test in Europe, 2002: 142-149.

[7] Eén N, Sörensson N. An extensible SAT solver [C]// Proc of International Conference on Theory and Applications of Satisfiability Testing (SAT 2003), 2003: 502-518.

[8] Audemard G, Simon L. Predicting learnt clauses quality in modern SAT solvers [C]// Proc of the 21st International Joint Conference on Artificial Intelligence, 2009: 399-404.

[9] Davis M, Logemann G, Loveland D. A machine program for theorem proving [J]. Communications of the ACM, 1962, 5 (7): 394-397.

[10] Marques-Silva J P, Lynce I, Malik S. Conflict-driven clause learning SAT solvers [M]// Handbook of Satisfiability. Amsterdam: IOS Press, 2009.

[11] Jeroslow R G, Wang J. Solving propositional satisfiability problems [J]. Annals of Mathematics and Artificial Intelligence, 1990, 1(1-4): 167-187.

[12] Freeman J W. Improvements to propositional satisfiability search algorithms [D]. Philadelphia: University of Pennsylvania, 1995.

[13] Biere A. Adaptive restart strategies for conflict driven SAT solvers [C]// Proc of International Conference on Theory and Applications of Satisfiability Testing (SAT 2008), 2008: 28-33.

[14] Biere A, Fröhlich A. Evaluating CDCL variable scoring schemes [C]// Proc of Theory and Applications of Satisfiability Testing (SAT 2015), 2015: 405-422.

[15] Liang J H, Ganesh V, Poupart P, et al. Exponential recency weighted average branching heuristic for SAT solvers [C]// Proc of the 13th AAAI Conference on Artificial Intelligence, 2016: 3434-3440.

[16] Liang J H, Ganesh V, Poupart P, et al. Learning rate based branching heuristic for SAT solvers [C]// Proc of International Conference on Theory and Applications of Satisfiability Testing (SAT 2006), 2016: 123-140.

[17] Zhang L T, Madigan C F, Moskewicz M H, et al. Efficient conflict driven learning in a boolean satisfiability solver [C]// Proc of the 2001 IEEE/ACM International Conference on Computer-Aided Design, 2001: 279-285.

作者简介:

陈秀兰(1994 -),女,四川成都人,硕士生,研究方向为自动推理和机器学习。**E-mail**: 534918591@qq.com

CHEN Xiu-lan, born in 1994, MS candidate, her research interests include automatic reasoning, and machine learning.

刘婷(1993 -),女,四川成都人,硕士生,研究方向为自动推理和机器学习。**E-mail**: 1142880230@qq.com

LIU Ting, born in 1993, MS candidate, her research interests include automatic reasoning, and machine learning.