一种基于识别重复路径的动态决策策略

常文静[1),3)] 徐 扬[2),3)]

[1)](西南交通大学信息科学与技术学院 成都 610036)

[2)](西南交通大学数学学院 成都 610036)

[3)](系统可信性自动验证国家地方联合工程实验室 成都 610036)

摘 要 在现有基于冲突学习子句的求解器中,重启和变量相位存储技术的频繁应用,导致重启之后产生大量重复变量赋值序列,在求解过程中对变量重复赋值会浪费求解资源.本文提出一种基于识别重复路径的动态决策策略.首先,检测搜索过程中产生的重复赋值变量序列,算法中参数依据子句数与变元数的比率而动态变化;其次,更新参与冲突次数最多的变量的活跃值,选择合适的分支决策变量,改变变量赋值序列.本文基于国际 SAT 竞赛中知名求解器 Glucose3.0,MapleCOMSPS,Glucose4.1 以及 Lingeling,分别实现了改进算法——DDIDT.实验结果可得,改进求解器 Glucose_DDIDT 相比 Glucose3.0 降低决策数为 11.2%~61.6%,且 Glucose_DDIDT 求解难度较大实例的个数提高了 63.9%.针对求解 2015 年到 2017 年 SAT 竞赛的应用类型的实例,Glucose_DDIDT 相比 Glucose3.0 的求解个数增长了 6.0%;改进求解器 MapleCOMSPS_DDIDT 相比 MapleCOMSPS 求解个数提高了 2.5%;相比 Glucose4.1,改进求解器 Glucose4.1_DDIDT 的求解个数增长了 3.1%;虽然 Lingeling_DDIDT 求解实例总数相比 Lingeling 只增加 1 个,但求解时间有所减少.实验表明,所提策略可有效识别重复路径,适时选择合适的分支决策变量,改变搜索路径,减少计算时间.

关键词 可满足问题;冲突驱动学习子句;重启;分支决策策略;重复赋值序列

中图法分类号 TP301 **DOI 号** 10.11897/SP.J.1016.2019.02309

A Dynamic Decision Strategy Based on Identification of Duplicate Trail

CHANG Wen-Jing[1),3)] XU Yang[2),3)]

[1)](*School of Information Science and Technology, Southwest Jiaotong University, Chengdu* 610036)

[2)](*School of Mathematics, Southwest Jiaotong University, Chengdu* 610036)

[3)](*National-Local Joint Engineering Laboratory of System Credibility Automatic Verification, Chengdu* 610036)

Abstract It is obvious that frequent restarts were introduced in current conflict driven clause learning (CDCL) solvers. Due to the wide adoption of variable state independent decaying sum (VSIDS) and phase saving, a large proportion of the assignment trails are re-created exactly as the ones before restarts. It empirically exploits the observation that CDCL solvers tend to make the same variables in a similar order. Repeated assignment of variables in the solution process will waste the calculation resources. This paper proposed a new strategy—A dynamic decision strategy based on the identification of duplicate trail (DDIDT). Firstly, the phenomenon of duplicate assignments trails between two succeeding restarts is illustrated by an example and experiment, then designing an algorithm for identifying those duplicate trails feasibly. And the setting of parameters of the threshold of trails is determined by a series of experiments. Secondly, one of the most surprising aspects of the relatively recent practical progress of CDCL solvers is that VSIDS decision selection heuristic. The VSIDS strategy selects decision variable based on a

收稿日期:2018-03-26;在线出版日期:2019-01-08.本课题得到国家自然科学基金(61673320)、中央高校基本科研业务费基金(2682018ZT10)资助.**常文静**,博士研究生,主要研究方向为智能信息处理和自动推理.E-mail:wenjing1021@163.com.**徐 扬**,教授,博士生导师,主要研究领域为逻辑代数、不确定性推理和自动推理.

strong activity-based heuristic. Initially, the score of each variable is the frequency of a literal occurrence in all clauses, and increase all variables additively by 1 which involved in conflicts. These scores are sorted in a trail. More significantly, the size of the real-world instances always being millions of clauses and variables, then the score of every variable which occurs most frequently is also high, so few variables of the front of the trail are still the same even if the trail updates scores periodically (every 256th conflict), because of the increment is one, which is too small that compared to its initial score. As a consequence, when restart for instance, which contains millions of clauses and variables, the front of the trail maybe still holds the same. Changing the assignment trails is essentially turning the order of assigning those variables. Therefore, for changing the activity of variable to a greater degree, the paper proposed a new branching strategy, that is, counting the number of times each variable participates in a conflict and recording the variable that responsible for the most conflict between two restarts, then rewarding this variable with a bonus in order to turn the assignment sequence, accordingly the search path is transformed. We implemented the DDIDT algorithm respectively as a part of these well-known solvers: Glucose3.0, MapleCOMSPS, Glucose4.1 and Lingeling solver, all these solvers got better ranking in international SAT competitions. These empirical results further shed light on that, compared with the Glucose3.0 solver, the rate of reducing decisions of modified Glucose_DDIDT solver is decreased by 11.2%—61.6%, and the number of solved hard instances of modified Glucose_DDIDT is increased by 63.9% compared with Glucose3.0, the above experimental results show that the proposed strategy can work well. To better illustrate the advantages of the proposed method, Application Main-track instances, which originated from the SAT Competitions 2015 to 2017, were also tested by those solvers which integrated DDIDT algorithm: MapleCOMSPS_DDIDT, Glucose4.0_DDIDT and Lingeling_DDIDT. Compared with Glucose3.0, the number of solved instances of the improved Glucose_DDIDT is improved by 6.0%; compared with MapleCOMSPS, the number of solved instances of MapleCOMSPS_DDIDT is added by 2.5%, meanwhile, the number of solved instances of Glucose4.0_DDIDT is improved up to 3.1% compared with Glucose4.0; Although the total number of solved instances of Lingeling_DDIDT is increased by 1 compared with Lingeling, the overall solution performance is improved. Experimental results indicate that the proposed dynamic decision strategy can efficiently identify duplicate assignments trails and deal with it by choosing appropriate decision variable adaptively, reduce the computation time.

Keywords satisfiability problem; conflict driven clause learning; restart; branching decision strategy; duplicate trail

1 引 言

布尔可满足问题(Boolean Satisfiability Problem,简称 SAT 问题)是首个被证明是 NP 完全的问题[1],具有十分重要的理论意义.与此同时,SAT 问题也应用在现实生活的不同领域,包括:(1)计算机科学[2]:约束满足问题、定理证明等;(2)数学[3]:旅行商问题、图着色问题等;(3)计算机辅助设计与制作[4]:软件模型检查、集成电路设计与验证等;(4)人工智能[5]:自动推理、规划问题等.由于 SAT 问题在现实应用中的广泛性和重要性,迫切促使研究人员提高 SAT 问题求解的相关技术.目前,求解 SAT 问题的算法主要分为两类:完备算法和不完备算法.尽管不完备算法可快速求解,却不能证明问题是不可满足的.完备算法不仅能对问题的属性是可满足时,给出问题的解;还在问题无解时,可以给出一个完备的证明,证明此问题是不可满足的.现实生活中许多实际应用问题需要证明问题的无解,因此,本文主要介绍完备算法的相关内容.

1962 年，Davis、Logemann 和 Loveland 提出了完备的 SAT 算法，简称为 DPLL 算法[6]，主要利用单文字规则、纯文字规则和分裂规则，通过深度优先搜索求解子句集，但是由于 SAT 问题的特殊性，导致 DPLL 算法在最坏情况下具有以问题规模为指数的时间复杂性. 尽管如此许多研究学者并没有因此望而却步，自 1996 年举办第一届 SAT 竞赛开始，迄今已成功举办 20 届 SAT 竞赛，第 21 届即将在牛津大学召开. 世界各国研究机构举办的 SAT 竞赛① 促使 SAT 算法及其实现程序的求解效率有了大幅度提高，同时也促使 SAT 问题在实际应用中更加广泛. 当前主流的 SAT 完备求解算法几乎都是基于 DPLL 算法衍生而来，其中最重要的是冲突驱动子句学习(Conflict Driven Clause Learning，简称 CDCL)算法，在 SAT 竞赛中大多数完备的求解器都是基于此算法而设计. CDCL 算法主要基于 DPLL 算法框架，在变元决策、冲突分析与子句学习、非时序回溯、重启、数据结构等方面作了一系列改进. 1996 年，GRASP 求解器[7]引入了非同步回溯和子句学习，很大程度上减小了搜索空间，是 CDCL 算法的雏形求解器；1997 年，SATO 求解器[8]中使用了头尾列表(head/tail list)的数据结构，提高了布尔约束传播的效率；2001 年，Chaff 求解器[9]的作者提出了基于观察文字的数据结构，以及低开销的独立变量状态衰减和决策策略(Variable State Independent Decaying Sum，简称 VSIDS)；2002 年，BerkMin 求解器[10]中添加了子句删除策略，避免出现内存爆炸问题；2004 年，Zchaff 求解器[11]是在 Chaff 求解器的基础上，改进程序的实现方式，提高求解效率；2005 年，Minisat 求解器[12]优化代码结构，使得求解效率有了大幅度提升，之后参加竞赛的多数求解器都是基于 Minisat 改进的；2009 年，Glucose 求解器[13]的作者在 Minisat 求解器的基础上，提出了一种新的学习子句管理策略——文字块距离(Literals Blocks Distance，简称 LBD)和动态重启策略，并且在 2011 年的 SAT 竞赛中获得了 Application 实例组的冠军，之后此求解器已成为 SAT 竞赛中判定实例难度的基础求解器；Biere 自主设计实现的 Lingeling 求解器②，也是基于 CDCL 算法框架，分别获得 2013 年及 2014 年 SAT 竞赛 Application 实例组的冠军；2016 年，Liang 等人提出了一种学习率分支(Learning Rate Branching，简称 LRB)[14-15]决策策略，设计求解器 MapleCOMSPS，获得了 2016 年 SAT 竞赛 Main Track 组的 Application 实例组的冠军以及 2017 年 SAT 竞赛 Main Track 组的 Application 实例组的亚军.

一方面，变量决策策略很大程度上影响着 CDCL SAT 求解器的求解性能. 变量决策策略是从未赋值的变量中选择一个，并对其赋值为 0(False)或 1(True). 对于变量个数为 N 的 SAT 问题，最糟糕情况下的计算复杂度为 $\mathrm{pow}(2,N)$，也就是对于 SAT 问题的搜索二叉树，必须访问到最后一个叶节点才得到可满足的解. 因此，为了避免此种情况的发生，研究学者提出了各种各样的决策策略. 从最基本的随机选择，到 JW(Jeroslow-Wang)算法[16]，Bohm 算法[17]，最短子句出现频率最大(Maximum Occurrence in Minimization，简称 MOM)算法[18]，动态最大变量数(Dynamic Largest Individual Sum，简称 DLIS)算法[7]和 VSIDS 算法. 其中，VSIDS 决策策略具有里程碑意义. 目前主流的决策策略都是基于此策略改进的. 2003 年，Eén 和 Sörensson 在 VSIDS 基础上做了改进，提出了指数的变量状态衰减和决策策略(Exponential VSIDS，简称 EVSIDS)算法[12]，该策略不区分变量的正负赋值(0 或 1)，默认选择一个赋值优先搜索，减少计算和存储开销；2004 年，Ryan 提出的 Variable Move-To-Front (VMTF)算法[19]，内容是将学习子句中的文字移动到活跃度队列前面，该算法对有界模型检验、电路验证问题具有较好的求解效率；2015 年，Biere 等人提出了 Average Conflict-Index Decision Score (ACIDS)[20]算法，增大与最近冲突相关的变量的计数值. 2016 年，文献[14]提出了 LRB 决策策略，分析每个变量在一次冲突间隔内产生学习子句的个数. 不同的决策变量生成不同的搜索路径，不同的搜索路径产生的求解时间千差万别. 如何选择一个更好的决策变量已被证明是 NP-hard 问题. 由于 SAT 问题的特殊性，并没有一个决策策略适用于求解不同类型的问题. 因此，现有的许多求解器融合了不同的变量决策策略，依据设定的相应参数变化，动态地改变决策策略，使得求解器尽可能的搜索不同的路径. 文献[21]通过估计每种决策策略在一段时间内的求解性能来动态改变决策策略，如决策层深度，学习子句的长度；文献[22]也是通过评估每个决策策略的一些求解性能指标(如证明宽度、可满足性评估等)

① The International Conferences on Theory and Applications of Satisfiability Testing (SAT). http://www.satisfiability.org/

② Lingeling, Plingeling and Treengeling. http://fmv.jku.at/lingeling/

来选择不同的决策策略;文献[23]列举了 138 种实例所具有的特征,包括问题的特性、基于图的特性、学习子句特性等,在求解初始和求解过程中评估实例的不同特征,选择较合适的决策策略.

另一方面,重启已经成为 CDCL SAT 的重要特性之一,研究[24]发现,重启可以有效的避免组合搜索中存在重尾分布(Heavy-Tail Distributions)现象.重启是指撤销之前所有的变量赋值,并且在重启后根据决策策略重新选择决策变量赋值,目的是生成不同的搜索路径.重启策略大致可以分类为静态重启和动态重启.静态重启策略主要有:固定间隔序列重启[9]、固定几何增长序列重启[25]、嵌套几何增长序列重启[26]以及 Luby 序列重启[27].这些静态重启策略都是根据重启阈值来触发重启的,但是当求解器接近于得到可满足或不可满足的结论时,如果恰好触发重启,则必须放弃之前的搜索过程,重新开始搜索,浪费不必要的计算资源.因此,许多学者提出了动态重启,现有大多数主流的 SAT 求解器中都采用动态重启策略,根据某些设置参数来触发或延迟重启.动态重启策略的重启次数比静态重启策略的重启次数频繁,因为基于活跃度的变量决策策略和变量相位存储策略[28]的影响,导致频繁的重启有可能在两次重启之间产生重复赋值的变量序列.文献[29-30]说明了此种现象的存在并提出了一种简单易实现的检查重复赋值序列的算法.因为重启是回退的一种特殊形式,因此文献[31]针对回退策略中存在的变量重复赋值,提出了一种部分回退策略,当发生冲突后需要回退时,可不再回退到学习子句中第二大的决策层次,而是直接回退到非重复的变量,减少重复变量传播的时间消耗.但是这些算法并没有本质上改变变量赋值的顺序,依然会产生重复序列,不断产生的重复赋值序列导致求解器一直在相似的搜索空间内搜索,随着变量活跃值的不断累加以及周期性的衰减,变量的赋值顺序发生改变,此时,已耗费过多的计算资源.

综上所述,本文首先分析了重复变量赋值存在的普遍性;基于此,提出一种基于识别重复路径的动态决策策略(Dynamic Decision based on Identification of Duplicate Trails,简称 DDIDT 算法),即通过检测搜索过程中产生的赋值变量序列,若重复序列出现的次数大于某个阈值范围,则增大相应变量的活跃度,利用不断增加的变量活跃度改变变量赋值的排列顺序,进而改变搜索路径.

2 基础知识

2.1 基本概念

首先给出 SAT 问题中一些基本定义.

定义 1. 文字.设 x 为一个布尔变量,该布尔变量可以赋值为 0 或 1.其形式符号 $\neg x$ 和 x 为一个文字的两个取值,其中,$\neg x$ 被称为负文字,x 被称为正文字.

定义 2. 子句.由一个或多个文字的析取组成.例如:$\neg x_1 \vee \neg x_2 \vee x_3$.若子句中至少存在一个文字赋值为 1,则该子句是可满足的.

定义 3. 合取范式(Conjunction Normal Form,简称 CNF).一个或多个子句的合取构成的子句集合.例如:$(\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee \neg x_3) \wedge (x_1 \vee x_3 \vee \neg x_4) \wedge (\neg x_3 \vee x_4 \vee \neg x_5)$.SAT 问题一般用 CNF 表示.

定义 4. 单元子句规则.当一个子句中仅剩一个文字未被赋值(或者该子句仅包含一个文字),且已赋值文字被赋值为 0,那么,若使此子句可满足,则此子句中仅剩的一个未被赋值的文字只能赋值为 1,且此文字称为单元文字.

定义 5. 布尔约束传播(Boolean Constraint Propagation,简称 BCP).对子句集反复应用单元子句规则,直至再也无法得到新的单元子句.

SAT 问题是指给定一个子句集 F,若存在一组变量赋值$\{x_1, x_2, \cdots, x_N\}$($N$ 为子句集 F 中的变量个数),使得 F 中所有的子句都是可满足的,则子句集 F 是可满足的,或者给出证明,对于变量的任何赋值,子句集 F 都是不可满足的.

2.2 CDCL 算法

算法 1. 典型 CDCL 算法.

输入:CNF 公式 F

输出:可满足 SAT 或不可满足 UNSAT

```
1.  clauses←clausesOf(F)
2.  v←∅    //变元赋值集合
3.  level←0  //决策层次
4.  IF UnitPropagation(clauses,v)==CONFLICT
                          //单元传播
5.    THEN RETURN UNSAT
6.  ELSE
7.    WHILE True DO
8.      var=PickDecisionVar(clauses,v)
                          //变量决策
9.      IF no var is selected
                          //所有变量都已被赋值
```

```
10.      THEN RETURN SAT
11.    ELSE
12.      level←level+1
13.      v←v∪{var}
14.    WHILE UnitPropagation(clauses,v)==
                 CONFLICT DO
15.    learned=AnalyzeConflict(clauses,v)
                          //冲突分析
16.      clauses←clauses∪{learned}
17.      dlevel=Computedlevel(clauses,v)
18.      IF dlevel==0
19.        THEN RETURN UNSAT
20.      ELSE
21.        BackTrack(clauses,v,dlevel)
                          //回退
22.        level←dlevel
23.      END WHILE
24.  END WHILE
```

算法 1 中，v 为所有变量的赋值集合，初始值为子句集 F 中所有的单文字，*level* 为决策层次，初始值为 0. *UnitPropagation*() 是单元传播函数，对集合 v 中的赋值进行布尔约束传播，若单元传播过程中发生冲突，则子句集 F 的属性为不可满足的(UNSAT)；否则，通过变量决策分支函数 *PickDecisionVar*() 选择决策变量并赋值，若所有变量都已被赋值，即可判定子句集 F 的属性是可满足的(SAT)，并且终止算法. 一旦确定一个决策变量，就调用 *UnitPropagation*() 函数，直至发生冲突. 发生冲突时，利用 *AnalyzeConflict*() 函数生成学习子句 *learned*，且将学习子句添加到子句集 F，并通过 *ComputeDLevel*() 函数确定回退层次 *dlevel*，如果 *dlevel*=0，则说明子句集 F 为不可满足的，否则，利用 *BackTrack*() 函数，回退到 *dlevel*，从新的决策层次重新开始搜索赋值.

现有 CDCL SAT 求解器中的决策分支函数 *PickDecisionVar*() 主要是指 VSIDS 决策策略及其发展的策略. VSIDS 决策策略为每个变量维护一个活跃度的计数器 $s(l)$ ($s(l)$ 为浮点数)，初始值为每个变量出现在子句集中的次数；当添加学习子句时，奖励学习子句中包含的变量的活跃度，即计数值加 1. 每次决策时，选择未赋值的具有最大 $s(l)$ 值的变量为分支决策变量. 所有变量的计数值 $s(l)$ 周期性的乘以一个常数 q ($0<q<1$)，避免局部最优. 因为 VSIDS 是基于冲突改变变量的计数值，与变量状态无关，计算消耗小，因此，目前大多数求解器都是基于此策略进行改进，如 Glucose 和 Lingeling.

2.3　重启策略

现有的 CDCL SAT 求解器中使用的重启策略大致可分类为静态重启和动态重启.

2.3.1　静态重启策略

(1) 固定间隔序列重启. 即每隔 N 次冲突，重启一次. 在 Chaff[9] 求解器中 $N=700$；BerkMin[10] 求解器中 $N=550$；Siege[19] 求解器中 $N=16\,000$.

(2) 固定几何增长序列重启. 初始时设置重启阈值为 C，每次重启时，重启阈值 $C=C\times q$；q 为增长系数($q>1$). Minisat 1.14 和 2.0 版本中设置 $q=1.1$，$C=100$，即重启序列呈现 100，110，121，…的几何增长序列.

(3) 嵌套几何增长序列重启. 设置两个重启阈值：内部阈值 *inner* 和外部阈值 *outer*，C_0 为初始值，q 为增长系数($q>1$). 每当冲突次数达到当前 *inner* 阈值时，触发重启，并且 $inner=inner\times q$；当 $inner\geq outer$ 时，$outer=outer\times q$，并且 $inner=C_0$. PicoSat[26] 求解器中采用了这种策略.

(4) Luby 序列重启. 文献[27]证明 Luby 序列是未知随机搜索空间的最佳调度方法. 其序列状态如：1，1，2，1，1，2，4，1，1，2，1，1，2，4，8，…，可被形式化表示为

$$t_i=\begin{cases}2^{k-1}, & i=2^k-1\\ t_{i-2^{k-1}+1}, & 2^{k-1}\leq i\leq 2^k-1\end{cases}.$$

由于 Luby 序列的间隔较小，在实际应用中，经过一次冲突就找到解的可能性非常小，因此，Luby 序列通常被乘以一个单位运行(unit run，简称 u)因子. 当 $u=32$ 时，即 Luby 序列为：32，32，64，32，32，64，128，32，32，64，32，32，64，128，256，…. 在 MiniSAT 2.2.2 版本中，$u=100$.

静态重启策略的一个明显缺陷就是一旦达到设置的固定周期时，无论搜索过程是否接近于得出结论，会即刻触发重启，撤销之前所得到的搜索信息. 因此，为了克服此缺陷，学者们提出了动态重启策略，通过评估搜索过程中的状态判断是否重启.

2.3.2　动态重启策略

文献[32]通过计算学习子句相应的 LBD 值当满足以下任一条件时，触发重启：(1) 当短期 LBD 平均值大于长期 LBD 平均值，且冲突次数大于 50 次；(2) 新得到的学习子句的 LBD 值较大.

文献[33]记录每个决策层次的冲突次数 *current* 以及总冲突次数 *global*，当发生冲突并回溯到决策层次 d 时，此时计算总冲突次数 *global* 与当前决

策层次 d 的冲突次数 $current(d)$ 的差值 k，即 $k=global-current(d)$. 当 k 大于设定的阈值时，触发重启.

文献[34]提出通过测试搜索过程的灵敏度来判断是否重启，灵敏度则根据存储的变量相位是否被翻转来显现. 如果大部分已存储的变量相位都已被翻转，则说明大量变量的正负赋值都已尝试过，求解器得到不可满足的可能性增大，此时延迟重启；若只有极少数的已存储的变量相位被翻转，则说明求解器很有可能陷入局部搜索，此时则触发重启.

文献[35]计算区间 LBD 移动平均值 SMA 与累计移动平均值 CMA 的偏离程度，若 $SMA>c\times CMA$ $(c>1)$，触发重启.

文献[36]提出一种 Multi-Armed Bandit 重启策略，动态地改变 4 种静态重启策略.

3 基于识别重复路径的决策策略

3.1 重复赋值序列

以下表示子句集 F.

F：$C_1=-x_8\vee x_2$，

$C_2=-x_1\vee x_4$，

$C_3=-x_1\vee x_7$，

$C_4=x_5\vee -x_9$，

$C_5=-x_1\vee x_6\vee x_8$，

$C_6=-x_3\vee -x_4\vee -x_6\vee -x_7$，

$C_7=-x_3\vee x_{10}\vee -x_{11}$，

$C_8=-x_2\vee -x_5\vee -x_{10}$，

$C_9=x_9\vee -x_{10}\vee -x_{12}$.

表 1 表示各个变量在重启前和重启后的变量活跃值和其相位变化情况.

表 1 重启前后变量的活跃值和相位变化

变量	活跃值（重启前）	活跃值（重启后）	相位（重启前）	相位（重启后）
x_1	73.2	77.8	True	True
x_2	87.5	90.3	True	True
x_3	45.9	47.3	False	False
x_4	66.4	69.5	True	True
x_5	68.2	70.2	False	False
x_6	60.1	63.4	True	True
x_7	54.3	55.2	True	True
x_8	110.8	112.5	False	False
x_9	50.7	51.3	True	True
x_{10}	70.1	71.6	**True**	**False**
x_{11}	75.4	76.3	True	True
x_{12}	98.5	100.2	True	True

表 1 的第 2 列和第 4 列分别表示重启前各个变量的活跃值和相位. 利用 VSIDS 的决策策略，根据未赋值变量的活跃值从大到小的顺序依次确定决策变量. 表 2 表示重启前后变量的赋值序列.

表 2 重启前后变量的赋值序列

决策层次	赋值序列（重启前）	赋值序列（重启后）
1	$-x_8$	$-x_8$
2	x_{12}	x_{12}
3	x_2	x_2
4	x_{11}	$x_1\rightarrow x_4, x_7$
5	$x_1\rightarrow x_4, x_7, x_6, -x_3, x_{10}, x_9$	x_{11}
6		$-x_{10}\rightarrow -x_3$

表 2 的第 1 列表示决策层次，从 1 开始计数，每确定一个决策变量，决策层次随之加 1. 表 2 的第 2 列表示重启前的变量赋值序列. “$-x_8$”表示文字 $-x_8$ 被赋值为 1；“x_{12}”表示文字 x_{12} 被赋值为 1；“$x_i\rightarrow x_j$”表示根据布尔约束传播，变量 x_i 的赋值蕴涵出变量 x_j 的赋值. 决策层次为 5 时，变量 x_5 的赋值存在冲突，此时相应增加学习子句中包含的变量的活跃值. 当达到重启条件时，求解器撤销之前所有的变量赋值，表 1 的第 3 列和第 5 列分别表示重启后各个变量的活跃值和相位，此时重新选择变量活跃值最大的变量进行赋值传播. 表 2 的第 3 列表示重启后的变量赋值序列. 比较表 1 各个变量的活跃值，发现重启前后其变化不大，并且根据变量相位存储机制（即存储每个变量的赋值相位，若此变量在之前已经被赋值了，此时赋值保持不变，若未赋值，则一般默认赋值为 False），各个变量的相位变化也不大. 因此，从表 2 的第 2 列和第 3 列可以发现，重启前后第 1 层到第 3 层的决策变量是相同，即变量的赋值序列是相同的. 分析其原因如下：(1) 子句集规模大，其子句数和变元数甚至能达到百万级别，相应的变量的活跃值也较大，而每次冲突时，学习子句中包含变量的活跃值仅增加 1；(2) 频繁的重启导致产生的学习子句逐渐减少，且学习子句之间的相似度较高，学习子句中包含的变量不断地被奖励，有可能排序在前端的变量的活跃值越来越大，而那些活跃值小的但对求解过程起重要作用的变量将会被忽略. 因此，频繁的重启产生重复变量赋值序列的可能性较大. 从表 2 可以看出，子句集 F 在一次重启间隔之内的重复赋值序列为 $\{-x_8, x_{12}, x_2\}$，决策层次分别为第 1 层、第 2 层和第 3 层. 位于二叉树上层的决策变量很大程度上决定了搜索路径，因此不断产生的重复赋值序列导致求解器一直在相似的搜索空间内搜索，随着变量活跃值的不断累加以及周期性的衰减，变量的赋值顺序有可能会发生改变，此时，已

耗费过多的计算资源. 为了更直观地观察求解过程中此现象存在的普遍性，随机测试 2015 年 SAT 竞赛 Application 类型① Main track 组的实例 aaai10-planning-ipc5-pathways-13-step17.cnf. 图 1 和图 2 分别表示使用不同的重启策略求解的过程中，每次重启时所产生的重复赋值序列. X 轴表示重启次数，Y 轴表示产生的重复赋值序列的大小. 图 1 表示 Luby 重启策略产生的重复序列；图 2 表示 Glucose3.0 求解器中使用的动态重启策略所产生的重复赋值序列. 不同的重启策略针对求解同一实例时，重启次数是不同的，Luby 重启策略的重启次数为 2639，动态重启策略的重启次数为 10 564，由图 1 和图 2 可以看出，重启次数越频繁，产生的重复序列越多，并且其重复序列的值分布在一定的范围之内.

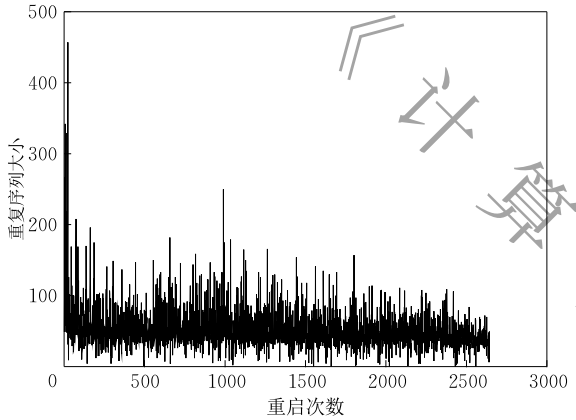

图 1　Luby 重启策略产生的重复序列

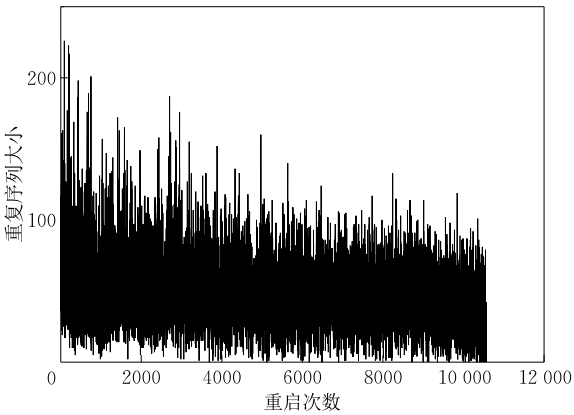

图 2　动态重启策略产生的重复序列

3.2　识别重复赋值序列

根据 3.1 节可知，频繁的重启导致产生大量的重复序列，此时需要通过改变相应变量的活跃度，进而改变变量赋值的排列顺序，下面给出 DDIDT 算法的实现过程.

算法 2.　DDIDT 算法.

输入：触发重启 $restart()$=True

输出：调用函数 $changeTrailOrder()$ 或 $PickDecisionVar()$

1. 初始化 $count \leftarrow 0$
2. $count = checkTrails()$;
3. IF $count > threshold$
4.　　THEN $changeTrailOrder()$;
5. ELSE
6.　　$PickDecisionVar()$;

每次当基于 CDCL 的求解器中重启函数 $restart()$ 被触发时，则调用 DDIDT 算法，算法 2 中 $checkTrails()$ 函数表示识别重复赋值序列算法. 当 $checkTrails()$ 函数的返回值 $count$ 大于设定的阈值 $threshold$ 时，说明变量的赋值序列一直都是重复的，此时需调用 $changeTrailOrder()$ 函数改变变量的活跃值排序，目的是下一次从不同的变量开始赋值搜索；否则，继续调用算法 1 中的 $PickDecisionVar()$ 函数确定赋值变量.

我们定义 S 表示重复赋值序列，$S.size$ 表示重复赋值序列的大小，则 3.1 节中的重复赋值序列为 $S=\{-x_8, x_{12}, x_2\}$，$S.size=3$. $S.size$ 的值越大，说明相同的赋值变量越多，如果在某一段时间内，$S.size$ 的值都保持在某一区间范围内，则应改变变量的赋值序列，使得求解器搜索不同的空间. 下面给出 $checkTrails()$ 算法的实现过程.

算法 3.　识别重复赋值序列算法 $checkTrails()$.

输入：重启前和重启后的变量活跃度数组

输出：计数值 $count$

1. 初始化 $count \leftarrow 0$
2. FOR $dl \leftarrow 1$ TO $decisionLevel$
3.　　IF $activity[$trail_order$[dl]] < activity[x_{next}]$
4.　　THEN $S.size++$;
5. END FOR
6. IF $min \leq S.size \leq max$
7.　　THEN $count++$;
8. RETURN $count$;

算法 3 中，假设 x_{next} 是即将被赋值的决策变量，dl 为决策层次，$decisionLevel$ 为重启前的决策层次，trail_order[] 表示重启前的变量赋值序列，trail_order[dl] 表示 dl 决策层次的决策变量. 每次重启后，检查新产生的赋值序列是否与重启前的赋值序列相同，即检查重启前的赋值序列中位于 x_{next} 之前的决策变量是否与重启后位于 x_{next} 之前的决策变量相同，而变量是按照活跃度 $activity$ 从大到小选择的，因此，只需比较 x_{next} 的活跃值是否大于重

① SAT 2015. https://baldur.iti.kit.edu/sat-race-2015/index.php?cat=downloads

启前与之相同决策层次的决策变量的活跃值，即 $activity[\text{trail\_order}[dl]]<activity[x_{\text{next}}]$，如果大于，则增加重复赋值序列的大小. 如果 $S.size$ 的值符合条件，即 $min\leq S.size\leq max$，此时增加计数器 $count$ 的值. 显然，参数 min 和 max 的值不同，对求解性能影响较大. 若 min 和 max 的值设置较大，则重复赋值序列的大小符合的区间范围较大，不能适时地改变赋值序列；若 min 和 max 的值设置较小，则会频繁地改变赋值序列，使求解器的搜索深度较短，不利于得到全局赋值. 为了较精确的设置参数 min 和 max，我们测试了 2001 年到 2013 年 SAT 竞赛的 Application 类型的不同种类的实例. 采用 Glucose3.0 版本的求解器. 实验中每个实例的运行时间不超过 3600 s. 这些实例来自于不同的实际问题，例如：软件测试、硬件电路测试、图着色问题、网络安全等. 本文大致选取了 21 种类型的实例，并且这 21 种类型的实例大多数在 3600 s 内并未得到结果，因此这些实例的求解难度较大. 根据测试可知，来源相同的实例的重复赋值序列的值所覆盖的区间范围大致相同. 统计不同类型的实例所对应的 $S.size$ 的区间范围，如表 3 所示.

表 3 不同类型实例的重复序列的区间范围

实例名称	实例个数	比率	$S.size()$的区间
homer(2002)	6	7.2～9.5	20～40
par32(2002)	9	3.2～4.0	10～30
li-test(2003)	8	3.8	500～1000
partial(2007)	16	4.4～4.7	300～900
countbits(2009)	10	3.1	50～150
gus-md5(2009)	9	3.2	250～300
ndhf-xits(2009)	10	93.2～110.5	10～20
rbcl-xits(2009)	13	55.2～62.4	10～25
gss(2009)	39	3.1	45～55
hwmcc10(2011)	7	3.0	随机
aes(2011)	12	3.5～12.0	50～80
openstacks(2011)	6	5.0	20～40
traffic(2011)	9	9.2～32.8	随机
slp-synthesis(2011)	23	3.2～3.4	25～60
sokoban-sequential(2011)	5	16.1	20～40
arcfour(2013)	8	37.9～38.5	30～50
bivium(2013)	9	5.7	15～25
ctl(2013)	23	8.0～10.2	随机
hitag(2013)	21	13.0～13.8	15～30
IBM-FV-2004(2002—2012)	25	3.6	50～150
SAT-dat-k(2010—2013)	17	4.5	40～70

表 3 中“随机”表示此类实例中每个实例的 $S.size$ 的值的覆盖范围都是不同的，无法总结出具体的数值. 表 3 的第 3 列表示每种类型实例的子句数与变元数比率的分布范围，比率($ratio$)＝子句数/变元数. 从表 3 可以看出，比率范围为 3～5 的实例类型有 13 种，对应的 $S.size()$的区间范围大致可分类两类：10～80 和 150～1000；比率范围为 7～15 的实例类型有 5 种，对应的 $S.size()$的区间范围为：15～40；其余 3 种实例类型的比率范围均较大，对应的 $S.size()$的区间范围：10～50. 这里，我们设置参数值 min 和 max 是随着不同类型实例的比率而动态变化的，根据上述统计结果，如表 4 所示.

表 4 随着比率变化的参数值

$ratio$	min	max
3～5	10	80
3～5	150	1000
7～15	15	40
其他	10	50

当求解某一实例时，首先计算此实例的比率，然后根据表 4 中相对应的比率范围，相应的设置参数值 min 和 max.

同样的，若算法 2 中阈值 $threshold$ 设置过大，也不利于较早的改变搜索路径；若较小，则导致搜索树层次较浅，也难得到解. 我们同样采用实验的方法，来确定阈值 $threshold$. $threshold$ 值即为图 2 中 x 轴的某一区间范围内所对应的 y 轴的值大于 min 以及小于 max 的“点”的个数. 根据观察测试实例中数据的变化情况，这里设置 $threshold=10,20,30,40$. 我们仍然使用 Glucose3.0 版本求解器，其中 min 和 max 随着比率而动态变化. 测试实例来源于 2015 年 SAT 竞赛 Application 类型 Main Track 组，共 300 个. 限定测试时间为 3600 s. 表 5 表示求解器中使用不同 $threshold$ 时，求解器所求解的实例个数. 由表 5 可知，当 $threshold=10$ 时，求解个数最多.

表 5 不同阈值的求解结果

阈值	求解个数
10	**245**
20	240
30	242
40	238

3.3 动态改变赋值序列

在算法 2 中 $changeTrailOrder()$ 函数的作用是改变变量的赋值序列，而变量是按照活跃值的大小被选择的，因此我们通过额外增加变量的计数值来改变变量的赋值顺序，尽可能的选择不同的决策变量搜索不同的路径.

再次以 3.1 节中子句集 F 为例. 图 3 表示子句集 F 重启前的一次变量赋值的逻辑蕴涵图，以有向非循环图表示.

图 3 中的每个顶点表示一个变量当前的赋值及

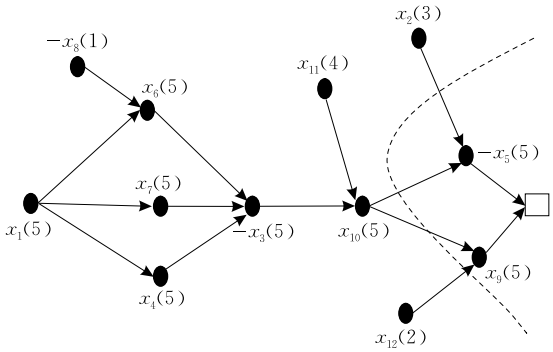

图 3　逻辑蕴涵图

其决策层次. 如果此顶点没有入边，只有出边，则此顶点代表的是决策变量. 从顶点 x_i 到顶点 x_j 的有向边表示变量 x_i 的赋值蕴涵出变量 x_j 的赋值. 例如，$x_1(5)$ 表示变量 x_1 是决策变量，赋值为 1，且其决策层次为 5. 由图 3 可以看出，变量 x_1 蕴涵变量 x_4，因此变量 x_4 的决策层次也是 5. “□”表示冲突子句 “$x_5 \vee -x_9$”. 冲突发生时，根据 1-UIP(First Unique Implication Point)学习机制[9]，即如图 3 中虚线所示，虚线右边的变量为冲突变量，虚线左边为导致冲突的变量)，可得学习子句 $-x_2 \vee -x_{10} \vee -x_{12}$. 学习子句中的变量为 x_{10}，x_2，x_{12} 的决策层次分别是 5，2，3，根据非时序回退，回退到学习子句中第二大的决策层次，即回退到决策层次 3，并且撤销决策层次 3 到决策层次 5 之间所有的变量赋值.

基于学习过程的分支策略 VSIDS 及其发展的一些策略，奖励与冲突有关的所有变量的活跃值(加 1)，图 4 表示学习子句的产生过程.

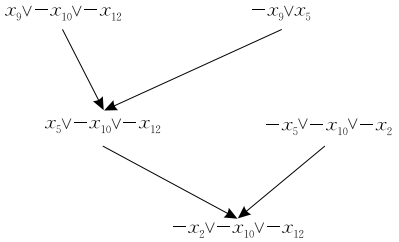

图 4　学习子句的产生过程

由图 4 可知集合 $L=\{x_2, x_{10}, x_{12}, x_9, x_5\}$ 中都是与冲突有关的变量. 变量参与冲突的次数越多，其活跃值越大. 因此，在后续过程中优先选择活跃值大的变量，认为越易避免之前已发生过的冲突，有利于减少搜索路径. 但是，现有的求解器中频繁地重启导致两次重启之前产生较少的学习子句，相应的对变量活跃值的影响也较小，从而进一步导致重复赋值序列的产生. 因此，为了较大程度的改变变量的活跃值，依据变量参与冲突的次数，额外的增加变量的奖励值. 若变量参与冲突的次数越多，则说明较早的选择此变元进行赋值传播，越易发生冲突. 在两次重启间隔之间，统计每个变量参与冲突的次数，记录参与冲突次数最多的变量. 算法 4 给出其具体实现过程.

算法 4.　计算变量参与冲突的次数.

输入：变量的赋值序列 *trail*[]
输出：参与冲突次数最大的变量 *var*
1. 初始化 *max_num*←0
2. 　　*num*[*nVar*]←{0}
3. FOR *i*←*trail.size*()−1 TO 0
4. 　*var*←*trail*[*i*]
5. 　*clause*←*reason*(*var*)
6. 　FOR *j*←0 TO *clause.size*()
7. 　　*var_clause*←*clause* [*j*]
8. 　　IF *level*(*var_clause*)==*level*(*var*) THEN
9. 　　CONTINUE
10. 　　ELSE
11. 　　*num*[*var*]++
12. 　　END FOR
13. 　　IF *max_num* <*num*[*var*] THEN
14. 　　*max_num*=*num*[*var*]
15. END FOR
16. RETURN *var*.

当需要改变变量排序时，即调用 *changeTrailOrder*()函数时，根据算法 4 得到的参与冲突次数最大的变量 *var*，更新变量 *var* 的活跃值 *activity*，即额外增加奖励值 100. 为了避免活跃值的溢出，当变量的活跃值大于阈值 1E100 时，我们平滑下降变量的活跃值. 算法 5 给出其具体实现过程.

算法 5.　更新变量活跃值.

输入：参与冲突次数最大的变量活跃值
输出：更新之后的变量活跃值
1. *activity*[*var*]=*activity*[*var*]+100
2. IF *activity*[*var*]>1e100 THEN
3. 　*activity*[*var*]*=1e-100

4　实验与结果分析

通过实验对比来说明新算法 DDIDT 的优势. 实验环境：Intel Core i3-3240 CPU 3.40 GHz、8 GB 内存，运行系统为 Windows7＋Cygwin2.8.1. 实验中每个实例的求解时间不超过 3600 s. 本文主要选取求解器 Glucose3.0 版本作为基础版本比较分析. 求解器 Glucose3.0_DDIDT 在是 Glucose3.0 的基础上实现了新算法 DDIDT. 实验主要分为三部分. 首先，求解器 Glucose3.0_DDIDT 重新求解 3.1 节中的实例 aaai10-planning-ipc5-pathways-13-step17.cnf，

观察其生成的重复序列;其次,对比了求解器 Glucose3.0 和求解器 Glucose3.0_DDIDT 分别求解 3.2 节中的 21 种不同类型实例的个数;最后,选取 2015 到 2017 年的 SAT 竞赛的 Application 类型的 Main Track 组[①]实例进行测试,进一步评估新算法在求解工业问题的性能优势.

4.1 求解单个实例

图 5 表示求解器 Glucose3.0_DDIDT 中采用 Luby 重启策略时,产生的重复赋值序列.与图 1 比较可得,算法 DDIDT 改变了重复赋值序列的值的大小,从图 5 可以看出,当重启次数大于 500 次时,重复序列的值越来越小.

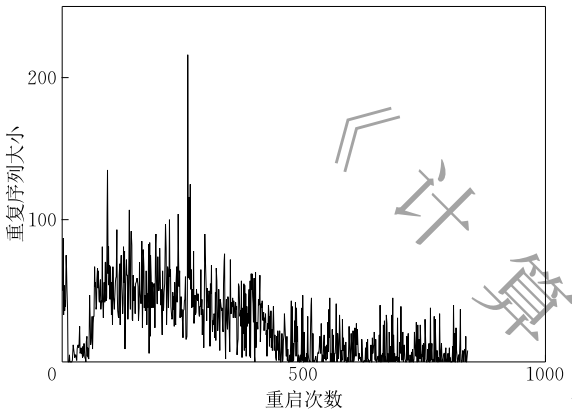

图 5 DDIDT 算法作用于 Luby 重启产生的重复序列

图 6 表示求解器 Glucose3.0_DDIDT 中采用动态重启策略时,产生的重复赋值序列.比较图 2 和图 6 可得,Glucose3.0 中使用的动态重启策略产生的重复赋值序列值的分布范围大致在 25～75,而 Glucose3.0_DDIDT 中使用相同的动态重启策略而产生的重复赋值序列值的分布范围大致在 5～25,重复序列的值减少.

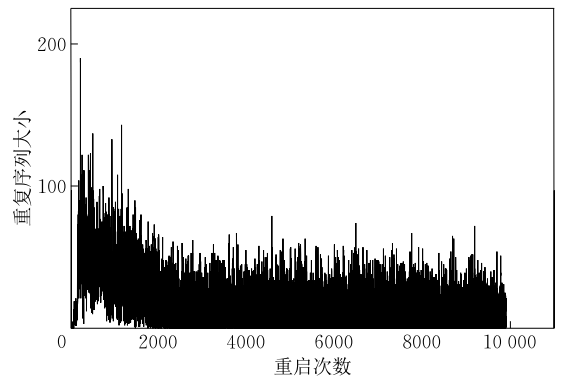

图 6 DDIDT 算法作用于动态重启产生的重复序列

表 6 中对比了求解器 Glucose3.0_DDIDT 和求解器 Glucose3.0 中分别使用 Luby 重启策略和动态重启策略得到的不同参数情况.

表 6 不同参数的对比

参数	重启次数	冲突次数	决策次数	求解时间/s
Luby 重启	2639	2882165	4093937	1012.99
Luby 重启-DDIDT	720	1489543	2025535	870.3
动态重启	10564	1930221	3425985	825.245
动态重启-DDIDT	9021	1722410	2856228	750.36

从表 6 可以看出,对于重启次数、冲突次数、决策次数和求解时间,使用算法 DDIDT 的求解器均有所减少.尤其决策次数对于 SAT 问题的求解效率起着核心作用,只有当搜索树的分支减小,进而搜索空间减少,才会降低运算时间.因此,综上所得,算法 DDIDT 对于求解实例 aaai10-planning-ipc5-pathways-13-step17.cnf 有一定的优势,说明算法 DDIDT 能较好地避免重启之后大量出现重复赋值序列的情况,并且选择越易构造冲突的变量,降低决策次数,自适应地改变搜索路径,减少重启次数,缩短求解时间.

4.2 求解不同类型实例

为了说明算法 DDIDT 中参数 *min*,*max* 和 *threshold* 设置的有效性,本文分别使用求解器 Glucose3.0 和 Glucose3.0_DDIDT 求解 3.2 节中 21 种类型的实例.表 7 为求解各个实例的个数.

表 7 不同算法求解不同类型实例的个数

实例名称	实例个数	Glucose3.0	Glucose3.0_DDIDT
homer(2002)	6	2	3
par32(2002)	9	0	2
li-test(2003)	8	0	3
partial(2007)	16	4	6
countbits(2009)	10	0	1
gus-md5(2009)	9	2	3
ndhf-xits(2009)	10	0	2
rbcl-xits(2009)	13	1	5
gss(2009)	39	3	7
hwmcc10(2011)	7	3	5
aes(2011)	12	2	6
openstacks(2011)	6	0	**0**
traffic(2011)	9	5	5
slp-synthesis(2011)	23	3	8
sokoban-sequential(2011)	5	1	**1**
arcfour(2013)	8	3	4
bivium(2013)	9	3	4
ctl(2013)	23	15	16
hitag(2013)	21	0	2
IBM-FV-2004(2002—2012)	25	9	10
SAT-dat-k(2010—2013)	17	5	7
总数	285	61	**100**

① SAT 2016. https://baldur.iti.kit.edu/sat-competition-2016/index.php?cat=downloads
SAT 2017. https://baldur.iti.kit.edu/sat-competition-2017/benchmarks/

21 种实例总数为 285，Glucose3.0 求解实例的总个数为 61 个，一些类型的实例求解个数甚至为 0，说明了这 21 种实例的求解难度较大，而 Glucose3.0_DDIDT 求解实例的个数为 100 个，整体增加了 39 个，除了实例类型“openstacks(2011)”和“sokoban-sequential(2011)”，其余实例的求解个数均有所增加.

为了验证参数设置的合理性，我们在求解器 Glucose3.0_DDIDT 中设置不同的 *min*，*max* 和 *threshold* 参数值进行测试. 测试实例来源于 2015 年 SAT 竞赛 Application 类型 Main Track 组，共 300 个. 限定测试时间为 3600s. 测试结果如表 8 所示.

表 8　不同参数求解实例的个数

min	*max*	*threshold*	求解个数
80	140	10	234
80	140	20	232
80	140	30	235
80	140	40	230
1000	2000	10	231
1000	2000	20	230
1000	2000	30	228
1000	2000	40	226

表 8 中 *min* 和 *max* 的值为固定不变的，即为表 5 中未包含到的值，*threshold* 仍分别设置为 10，20，30 和 40. 比较表 8 和表 5 可得，当 *threshold*=10，*min* 和 *max* 依据表 4 动态变化时，其求解个数最多(245 个).

4.3　实验结果

为了进一步观察算法 DDIDT 算法对决策次数的影响，表9列举了Glucose3.0和Glucose3.0_DDIDT 在相同环境下求解实例的决策次数及其下降比. 下降比=(Glucose3.0 决策数-Glucose3.0_DDIDT 决策数)/(Glucose3.0 决策数). 实例来自于 2015 年 SAT 竞赛 Application 类型 Main Track 组的部分实例. Glucose3.0 和 Glucose3.0_DDIDT 采用相同的动态重启策略. 表 9 中，“—”表示在实验测试的规定求解时间 3600s 内未被求解.

表 9　不同算法求解不同类型实例的决策数

实例名称	类型	Glucose3.0	Glucose3.0_DDIDT	下降比/%
6s167-opt	unsat	3662141	2368758	35.3
008-80-4	sat	87813902	46885690	46.6
50bits_14.dimacs	sat	—	7060300	—
atco_encl_opt1_04_32	sat	8621392	4982387	42.2
ACG-20-5p0	unsat	1034570	1106898	−7.0
beempgso15b1	unsat	2763664	2814855	−1.9
countbitssrl032	unsat	4618624	2156275	53.3
jgiraldezlevy.2200.9086.08.40.22	sat	4920124	3185887	35.2
manthey_DimacsSorter_36_7	sat	—	1453277	—
manthey_DimacsSorter_28_4	sat	1253098	1044857	16.6
mrpp_8x8#8_9	unsat	119761	74558	37.7
E02F22	sat	2546067	1563897	38.6
manthey_single-ordered-initialized-w46-b9	unsat	4447510	3362114	24.4
post-cbmc-aes-d-r2	unsat	20775095	7966854	61.6
UCG-20-5p1	sat	1226521	1088636	11.2
UTI-20-10p1	sat	7604097	5533263	27.2

从表 9 可以看出，除 2 个实例的决策数增加了 7.0%和 1.9%外，不论实例属性是可满足的或不可满足的，新算法 DDIDT 相比 Glucose3.0 可降低决策数为 11.2%～61.6%. 通过表 9 的实验数据可知新算法可有效减少搜索树规模.

为了更好的评估算法 DDIDT 对求解过程的综合作用，我们在不同的求解器中实现了算法 DDIDT，其中，实例来自于 2015 到 2017 年的 SAT 竞赛的 Application 类型 Main Track 组，共 950 个，测试时间限定为 3600s. 首先在求解器 Glucose3.0 中实现算法 DDIDT，形成 Glucose3.0_DDIDT 求解器. Glucose3.0 中完全基于 CDCL 算法框架，因未集成较多的优化方法而广泛被用于算法改进的比较. 在近几年的 SAT 竞赛中，专设一组基于 Glucose3.0 改进版本的求解器竞赛. 表 10 列举了两种求解器求解实例的个数.

表 10　Glucose3.0 和 Glucose3.0_DDIDT 求解实例个数

Benchmarks	Status	Glucose3.0	Glucose3.0_DDIDT
Sat2015(300)	sat	137	155
	unsat	93	93
	sum	230	248
Sat2016(300)	sat	56	62
	unsat	76	79
	sum	132	141
Sat2017(350)	sat	72	80
	unsat	69	64
	sum	141	144
Total(950)	sat	265	297
	unsat	238	236
	sum	503	**533**

从表 10 的实验结果可以看出，改进版本求解器 Glucose3.0_DDIDT 求解实例总数相较于 Glucose3.0 求解器增长了 6.0%，其中主要体现在可满足实例求解个数的增长. Glucose 求解器的设计者 Audemard 和 Simon 教授在文献[13]中曾提出：当 SAT 求解器的求解个数至少增长 10 个时，说明此求解器的改进是成功的. 因此，由表 9 可以得出，新算法 DDIDT 对于提高求解器的求解性能是有效的，说明算法 DDIDT 具有一定的求解优势.

为了说明此算法对其他求解器亦是有效的,我们选取 SAT 竞赛中表现优秀的求解器:Glucose4.1,MapleCOMSPS 和 Lingeling.其中,Glucose4.1 是 Glucose 系列求解器的最新串行版本,在求解过程中自适应的改变策略,在 SAT2017 竞赛中取得 Agile-Track 组的第三名;MapleCOMSPS 求解器是在 Minisat 求解器的基础上集成了许多的优化方法(inprocessing 和新的变量决策策略),在 SAT2017 竞赛中取得 Main-Track 组的第二名;Lingeling 求解器也是在 CDCL 算法框架上自主实现的求解器,获得了 2013 年及 2014 年 SAT 竞赛 Application 实例组的第一名,在 SAT2016 竞赛中取得 Main-Track 组的第三名.我们在这三种求解器中分别实现了 DDIDT 算法,相应的形成求解器 Glucose4.1_DDIDT,MapleCOMSPS_DDIDT 和 Lingeling_DDIDT.表 11 列举了 6 种求解器的求解实例个数.

表 11 不同求解器的求解实例个数

Benchmarks	Status	Glucose4.1	Glucose4.1_DDIDT	MapleCOMSPS	MapleCOMSPS_DDIDT	Lingeling	Lingeling_DDIDT
Sat2015(300)	sat	134	145	154	156	143	144
	unsat	101	104	102	103	109	109
	sum	235	249	256	259	252	253
Sat2016(300)	sat	63	64	65	77	59	62
	unsat	81	81	78	68	95	94
	sum	144	145	143	145	154	156
Sat2017(350)	sat	77	76	90	90	79	90
	unsat	88	91	68	77	93	80
	sum	165	167	158	167	172	170
Total(950)	sat	274	285	309	323	281	296
	unsat	270	276	248	248	297	283
	sum	544	**561**	557	**571**	578	**579**

从表 11 的实验结果可以看出,改进版本求解器的求解实例个数均有所增长,Glucose4.1_DDIDT 求解总数相较于 Glucose4.1 提高了 3.1%;MapleCOMSPS_DDIDT 求解总数相较于 MapleCOMSPS 求解器增长了 2.5%;相较于 Lingeling,Lingeling_DDIDT 的求解个数仅多了一个.其中,改进求解器性能的提高主要体现在求解可满足实例.

图 7 表示求解器 Glucose3.0 和 Glucose3.0_DDIDT 求解 950 个实例的运行时间对比.

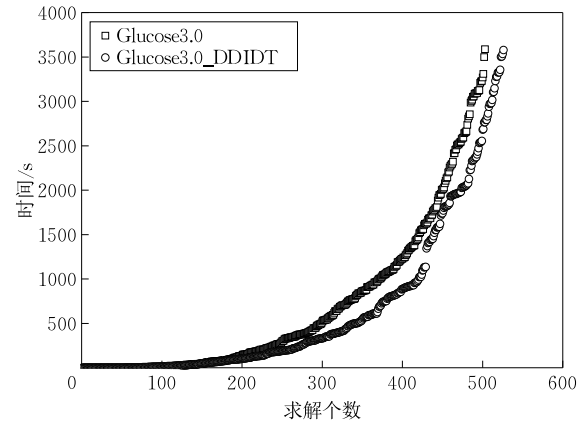

图 7 Glucose3.0 和 Glucose_DDIDT 求解时间对比

图 8 表示求解器 MapleCOMSPS,Glucose4.1,Lingeling 以及改进版 MapleCOMSPS_DDIDT,Glucose4.1_DDIDT,Lingeling_DDIDT 求解 950 个实例的运行时间对比.图 7 和图 8 中的波点曲线越靠近右边以及越靠近 x 轴时,说明此曲线表示的求解时间越小和求解个数越多.因此,由图 7 可以看出,Glucose3.0 和 Glucose3.0_DDIDT 的求解时间相差较大,Glucose3.0 求解实例的时间整体上多于 Glucose3.0_DDIDT 求解器;同理,从图 8 可以看出,当分别比较改进版本与原始版本的求解性能时,有 MapleCOMSPS_DDIDT > MapleCOMSPS,Glucose4.1_DDIDT > Glucose4.1,Lingeling_DDIDT > Lingeling.其中 Lingeling_DDIDT 的求解时间波点曲线仍处于最下方和最右侧,说明 Lingeling_DDIDT 求解器的求解性能最优.

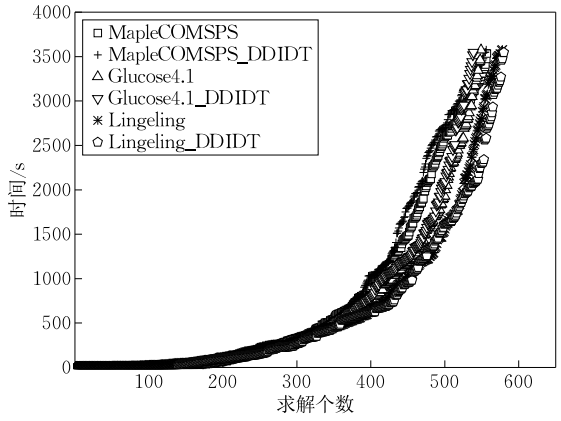

图 8 不同求解器的求解时间对比

5 总　结

针对频繁重启之间存在重复变量赋值序列的现象，本文设计并实现了一种识别重复路径的算法，并结合改进的变量决策策略，提出了一种基于识别重复路径的动态决策策略 DDIDT. 实验结果表明，新算法 DDIDT 可适应性的改变变量赋值的顺序，在一定程度上减少了重复路径的生成. 同时，实验结果表明新算法的求解性能也优于国际 SAT 竞赛中知名的求解器.

本文中一些参数的设置是带有实验性质的，因此，之后的研究方向可以结合机器学习算法，根据每个实例所具有的不同特性相应地确定不同的参数，更好地提升求解器的求解能力.

参　考　文　献

[1] Cook S A. The complexity of theorem-proving procedures//Proceedings of the ACM Symposium on Theory of Computing. Shaker Heights, USA, 1971: 151-158

[2] Khurshid S, Marinov D. TestEra: Specification-based testing of Java programs using SAT. Automated Software Engineering, 2004, 11(4): 403-434

[3] Darwiche A. New advances in compiling CNF into decomposable negation normal form//Proceedings of the Eureopean Conference on Artificial Intelligence. Valencia, Spain, 2004: 328-332

[4] Chen P, Keutzer K. Towards true crosstalk noise analysis//Proceedings of the IEEE/ACM International Conference on Computer-Aided Design. San Jose, USA, 1999: 132-138

[5] Rintanen J, Heljanko K. Planning as satisfiability: Parallel plans and algorithms for plan search. Artificial Intelligence, 2006, 170(12-13): 1031-1080

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

[7] Silva J P M, Sakallah K A. GRASP: A new search algorithm for satisfiability//Proceedings of the IEEE/ACM International Conference on Computer-Aided Design. Los Alamitos, USA, 2002: 220-227

[8] Zhang H. SATO: An efficient propositional prover//Proceedings of the 14th International Conference on Automated Deduction. Townsville, Australia, 1997: 272-275

[9] Malik S, Zhao Y, Madigan C F. Chaff: An efficient SAT solver//Proceedings of the Design Automation Conference. Las Vegas, USA, 2001: 530-535

[10] Goldberg E, Novikov Y. BerkMin: A fast and robust SAT-solver//Proceedings of the Design Automation and Test. Acropolis, Grace, 2002: 142

[11] Zhang L, Mdaigna C, Moskewiez M. Efficient conflict driven learning in a Boolean satisfiability solver//Proceedings of the International Conference on Computer-Aided Design. Braunschweig, Germany, 2001: 279-285

[12] Eén N, Sörensson N. An extensible SAT solver//Proceedings of the International Conference on Theory and Applications of Satisfiability Testing. Santa Margherita Ligure, Italy, 2003: 502-518

[13] Audemard G, Simon L. Predicting learnt clauses quality in modern SAT solvers//Proceedings of the International Joint Conference on Artificial Intelligence. Pasadena, USA, 2009: 399-404

[14] Liang J H, Ganesh V, Poupart P, Czarnecki K. Learning rate based branching heuristic for SAT solvers//Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing. Bordeaux, France, 2016: 123-140

[15] Liang J H, Ganesh V, Poupart P, Czarnecki K. An empirical study of branching heuristics through the lens of global learning rate//Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing. Melbourne. Australia, 2017: 119-135

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

[17] Buro M, Kleine-Büning H. Report on a SAT competition. Bulletin of the European Association for Theoretical Computer Science, 1993, 5(2): 49

[18] Selman B, Kautz H, McAllester D. Ten challenges in propositional reasoning and search//Proceedings of the 15th International Conference on Artificial Intelligence. Aichi, Japan, 1997: 50-54

[19] Ryan L. Efficient Algorithms for Clause-Learning SAT Solvers [M. S. dissertation]. Simon Fraser University, Canada, 2004

[20] Biere A, Fröhlich A. Evaluating CDCL variable scoring schemes//Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing. Austin, USA, 2015: 405-422

[21] Shacham O, Yorav K. Adaptive application of SAT solving techniques. Electronic Notes in Theoretical Computer Science, 2006, 144(1): 35-50

[22] Siddqi S, Huang J. Multi-SAT: An adaptive SAT solver//Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing. Austin, USA, 2015: 24-25

[23] Xu L, Hutter F, Hoos H H. SATzilla2009: An automatic algorithm portfolio for SAT//Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing. Edinburgh, Scotland, UK, 2010: 43-45

[24] Gomes C P, Selman B, Crato N. Heavy-tailed distributions in combinatorial search//Proceedings of the International Conference on Principles and Practice of Constraint Programming. Linz, Austria, 1997: 121-135

[25] Eén N, Sörensson N. Minisat v1.13-a SAT solver with conflict-clause minimization//Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing. St. Andrews, UK, 2005: 121-123

[26] Biere A. PicoSAT essentials. Journal on Satisfiability Boolean Modeling & Computation, 2008, 4(2-4): 75-97

[27] Luby M, Sinclair A, Zuckerman D. Optimal speedup of Las Vegas algorithms. Symposium Theory and Computing Systems, 1993, 47(4): 128-133

[28] Pipatsrisawat K, Darwiche A. A lightweight component caching scheme for satisfiability solvers//Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing. Lisbon, Portugal, 2007: 294-299

[29] Ramos A, Tak P V D, Heule M J H. Between restarts and backjumps//Proceedings of the 14th International Conference on Theory and Application of Satisfiability Testing. Ann Arbor, USA, 2011: 216-229

[30] Ramos A, Tak P V D, Heule M J H. Reusing the assignment trail in CDCL solvers. Journal on Satisfiability Boolean Modeling and Computation, 2011, 5(7): 133-138

[31] Jiang C, Zhang T. Partial backtracking in CDCL solvers//Proceedings of the 19th International Conference on Logic for Programming, Artificial Intelligence and Automated Reasoning. Stellenbosch, South Africa, 2013: 490-502

[32] Audemard G, Simon L. Refining restarts strategies for SAT and UNSAT//Proceedings of the International Conference in Principles and Practice of Constraint Programming. Québec, Canada, 2012: 118-126

[33] Ryvchin V, Strichman O. Local restarts//Proceedings of the 11th International Conference on Theory and Applications of Satisfiability Testing. Guangzhou, China, 2008: 271-276

[34] Biere A. Adaptive restart strategies for conflict driven SAT solvers//Proceedings of the 11th International Conference on Theory and Applications of Satisfiability Testing. Guangzhou, China, 2008: 28-33

[35] Biere A, Fröhlich A. Evaluating CDCL restarts schemes//Proceedings of the 6th Pragmatics of SAT Workshop. Austin, USA, 2015: 15-30

[36] Nejati S, Liang J H, Gebotys C, et al. Adaptive restart and CEGAR-based solver for inverting cryptographic hash functions //Proceedings of the 9th Working Conference on Verified Software: Theories, Tools, and Experiments. Heidelberg, Germany, 2017: 120-131

CHANG Wen-Jing, Ph. D. candidate. Her research interests include intelligent information processing and automated reasoning.

XU Yang, professor, Ph. D. supervisor. His main research interests include logical algebra, uncertainty reasoning, and automated reasoning.

Background

The research area of this paper is Boolean satisfiability (SAT) problem, which is a core problem in Artificial Intelligence, had been proved to be NP-complete problem. State-of-the-art complete SAT solvers are predominantly based on conflict driven clause learning (CDCL) algorithm, which extends the Davis Putnam Logemann Loveland (DPLL) algorithm by adding effective search techniques. Restart is one of the most surprising aspects of the relatively recent practical progress of CDCL solvers. Because of frequent restarts and phase saving in modern CDCL solvers, there exists a large proportion of duplicate assignment trails before and after restarts. Van Der Tak et al. showed that often a large part of the trail can be reused after restarts; Jiang et al. observed that backtracking also exhibits a similar phenomenon and presented a partial backtracking strategy. But these algorithms don't essentially change the order in which these variables are assigned. It still produces repetitive sequences in the process of solving between two restarts. That is to say, the search path of the solver is similar and more computing resources may be consumed.

This paper first analyzes the universality of the existence of repeated variable assignments, in view of this phenomenon, an algorithm is specially designed to identify those duplicate trails feasibly. Next, in order to deal with the duplicate trails, this paper proposes a decision strategy by rewarding variables with a large value, in order to change assignment sequence and converting the search path further.

This work is supported by the National Natural Science Foundation of China under Grant No. 61673320; the Fundamental Research Funds for the Central Universities under Grant No. 2682018ZT10.

SAT problem is the most surprising aspect in the field of automated reasoning, improving the efficiency of SAT solver is often a cruel world. On one hand, local search algorithms had been used for solving large random instances of SAT; on the other hand, a complete CDCL algorithm, which is a systematic backtrack search algorithm, a significant fraction of proving unsatisfiability. Although hundreds of techniques and heuristics have been proposed over the last five decades to improve CDCL solvers, decision branching strategy and restarts are still the two most important. The DDIDT algorithm proposes a new decision strategy based on the question of the restart strategy. The experimental results show that algorithm DDIDT has the advantage of solving Application benchmark.