Software Engineering and Applications 软件工程与应用, 2016, 5(1), 47-55
Published Online February 2016 in Hans. http://www.hanspub.org/journal/sea
http://dx.doi.org/10.12677/sea.2016.51006

Hans汉斯

Heuristic Strategies for Deciding Variables in DPLL Algorithm

Wenxiu Liu, Jianguo Jiang, Qianhui Li

School of Mathematics, Liaoning Normal University, Dalian Liaoning
Email: 1070957463@qq.com, jjgbox@sina.com, lqh274243982@sina.com

Received: Jan. 29th, 2016; accepted: Feb. 12th, 2016; published: Feb. 19th, 2016

Abstract

Aiming at the problem of the resolution efficiency of propositional satisfiability problem for propositional formula φ (CNF Form), based on the study of the DPLL complete algorithm and the research of the word order processing under different forms, a new algorithm for solving SAT problem is proposed. The algorithm groups propositional formula φ according to the length of clause, and sorts the words according to the number of polarity occurrence frequency in the initial sort. Then it considers the number of occurrences of the plus or minus words in variables and gets the value of high numbers. Algorithm instance shows that the new algorithm can reduce the number of rules in the process of using, reduce the solving steps, cut off the unsatisfy space of solution as soon as possible, so as to improve the solution efficiency.

Keywords

DPLL Algorithm, Heuristic Strategy, Propositional Satisfiability Problem

DPLL算法中的变量决策启发式策略

刘文秀，江建国，李千卉

辽宁师范大学数学学院，辽宁 大连
Email: 1070957463@qq.com, jjgbox@sina.com, lqh274243982@sina.com

收稿日期：2016年1月29日；录用日期：2016年2月12日；发布日期：2016年2月19日

文章引用: 刘文秀, 江建国, 李千卉. DPLL 算法中的变量决策启发式策略[J]. 软件工程与应用, 2016, 5(1): 47-55.
http://dx.doi.org/10.12677/sea.2016.51006

摘　要

针对命题公式φ(CNF形式)的可满足性问题的求解效率问题，基于对DPLL完全算法的学习和在不同形式下对子句文字排序处理的研究，提出了一种新的求解SAT问题的算法。新算法根据子句长度对命题公式φ进行分组，并根据变量出现次数进行初始排序，然后考虑变量中正负文字出现的次数，并对次数高的进行赋值。算法实例表明：新算法能减少求解过程中规则使用次数，减少求解步骤，尽早剪除不满足解空间，从而有效提高求解效率。

关键词

DPLL算法，启发式策略，可满足性问题

1. 引言

1960年，Davis和Putnam提出了一种判定可满足性问题(propositional satisfiability problem，简称SAT)的完备算法(DP算法[1])。后来，Logemann和Loveland等人对DP算法进行了改进，提出了一种更高效的SAT判定算法——DPLL算法[2]。目前，DPLL算法已成为一种判定SAT问题的最主流的方法，在物理电子学[3]，自动推理[4]-[6]，电力电子与电力传动[7]，计算机软件与理论[8] [9]等领域得到了广泛应用。

然而，DPLL算法在应用过程中有时会出现求解所需时间过长的问题，这使得整个判定过程不再高效化。针对该问题，研究者们提出了一些算法，主要包括：函数变换算法[10]，SAT邻域快速搜索法[11]，随机算法[12]，变量分解消除算法[13]，变换多项式求解[14]等，这些算法通过对决策层中子句进行处理，有效降低了子句集的规模，但对子句中判定文字的选取比较随机，这就可能导致判定过程中无效判定文字的出现，从而扩大求解空间，降低求解效率。

针对上述问题，本文提出了一种有效选取判定文字的算法。该算法继承了DPLL算法的先进求解思想，并能根据子句长度及变量出现次数进行判定文字的选取。在算法中首先计算子句长度，并根据不同的长度进行分组；其次，计算子句中每个变量的出现次数，进行初始排序；再次，考虑排序后次数最高变量中正负文字的出现次数，并对次数高的文字赋值；最后，根据选取的判定文字证明子句的可满足性。根据子句长度分组的方法能够优先对简单的子句进行判定，从而选取出有效的判定文字，缩减子句集的求解空间。同时，优先对正负文字出现次数较高的变量进行赋值，可以尽量避免冲突的出现，尽早剪除不满足解空间，尽快求得可满足解。将这两种方法结合在一起，就能达到快速选取出有效的判定文字及判定子句可满足性的目的，更能提高求解效率。实例表明，新算法可以高效地判定出子句的可满足性，减少求解步骤，提高求解速度。

2. DPLL 算法

现实生活中，很多难以解决且重要的组合、验证问题均可转化为SAT问题进行求解。通常，SAT问题中的任意命题逻辑公式都可以变为合取范式(conjunctive normal form，简称CNF)进行求解，即求解时只需要找到一组真值赋值，使CNF中的每个子句都是可满足的，从而该CNF就是可满足的。因此在这里只考虑CNF。

CNF通常是由一个或多个子句$C_i\left(i=1,\cdots,n\right)$通过合取运算符$\left(\wedge\right)$或$\left(,\right)$连接起来的，形如$C_1\wedge C_2\wedge\cdots\wedge C_n$或集合$\left\{C_1,C_2,\cdots,C_n\right\}$。子句是指$l_i\left(i=1,\cdots,n\right)$通过析取运算符$\left(\vee\right)$连接，形如$l_1\vee l_2\vee\cdots\vee l_n$，也可简记为

$l_1 l_2 \cdots l_n$，其中 $l_i\,(i=1,\cdots,n)$ 为文字(literal)，l_i 的取值可以为 0 或 1。将子句中所含全部文字的个数称为子句的长度，用符号“$|\ |$”表示。如果子句 C_i 中含有 n 个文字，那么 $|C_i|=n$。若子句中至少存在一组赋值使得其中包含的文字取值为真，那么称该子句是可满足的(取值为 1)，否则称子句是不可满足的(取值为 0)。文字是指变量 l_i 本身及其否定，即变量本身称为正文字(l_i)；其否定称为负文字($\overline{l_i}$)。正文字和负文字统称为文字。

DPLL 算法[1]是一种解决可满足性问题的深度优先搜索的完备算法。经典的 DPLL 系统[15]是一个转移系统，由下面七条基本规则构成：

R1 单文字繁衍规则 $M\|F, C\vee l \Rightarrow Ml\|F, C\vee l$，若 $\begin{cases} M\vDash\neg C \\ l\text{在}M\text{中未定义}\end{cases}$；

R2 纯文字规则 $M\|F \Rightarrow Ml\|F$，若 $\begin{cases} l\text{出现在}F\text{的某些子句中} \\ \neg l\text{不出现在}F\text{的子句中} \\ l\text{在}M\text{中未定义}\end{cases}$；

R3 判定规则 $M\|F \Rightarrow Ml^d\|F$，若 $\begin{cases} l\text{或}\neg l\text{出现在}F\text{的某个子句中} \\ l\text{在}M\text{中未定义}\end{cases}$，这里 l^d 为 l 通过规则得到；

R4 失败规则 $M\|F, C \Rightarrow FailState$，若 $\begin{cases} M\vDash\neg C \\ M\text{中不包含判定文字}\end{cases}$，这里 C 为 $M\|F, C$ 状态下的冲突子句；

R5 回溯规则 $Ml^d N\|F, C \Rightarrow M\neg l\|F, C$，若 $\begin{cases} Ml^d N\vDash\neg C \\ N\text{中不含有判定文字}\end{cases}$，这里 C 为 $Ml^d N\|F, C$ 状态下的冲突子句；

R6 学习规则 $M\|F \Rightarrow M\|F, C$，若 $\begin{cases} C\text{中的每一个原子出现在}F\text{或}M\text{中} \\ F\vDash C\end{cases}$；

R7 忘记规则 $M\|F, C \Rightarrow M\|F$，若 $F\vDash C$。

上述规则中 M 是由规则推导得到的赋值表(即文字的序列表)，F 为 CNF 的有限子句集，其中 R1 至 R5 中 F 保持不变，R6 和 R7 中 F 是变化的。

现简单介绍每条规则：

R1 单文字繁衍规则：采用的是 CHAFF [16]中提出的两观察变量法，即为了使 CNF 命题是可满足的，要求子句集 F 中的每一个公式都为真。因此，若 F 的一个子句包含文字 l，它在当前的赋值表 M 中没有赋值，但已知该子句中的其他文字赋值为假，那么文字 l 一定被繁衍为真。

R2 纯文字规则：如果在 F 中文字仅有正文字 l 或负文字 $\overline{l}$ 时，那么当且仅当文字为真时，F 是可满足的，这时将这个文字称为纯文字。若赋值表 M 中该文字没有定义，那么它一定被繁衍为真。

R3 判定规则：从 F 中选取一个未定义文字 l 添加到赋值表 M 中，将 l 标记为判定文字 l^d，并由 l^d 繁衍，如果繁衍后的赋值表不能使 F 为可满足的，那么由 R5 规则知，需要将 $\overline{l}$ 加入到赋值表中继续繁衍。

R4 失败规则：如果 M 中不包含判定文字，且由赋值表 M 推出冲突子句 C，那么这个赋值过程就是一个失败过程。

R5 回溯规则：如果在没有失败之前发现一个冲突子句 C，那么回溯到当前判定文字处，用 $\overline{l}$ 代替 l^d，并将由 l^d 繁衍得到的文字删掉。此时文字 $\overline{l}$ 不再是判定文字，因为文字 l 繁衍的情况已经讨论完。

R6 学习规则：将子句 C 添加到 F 中的过程称为学习，这里子句 C 中的所有原子出现在 F 或 M 中，但它原来并不在 F 中。

R7 忘记规则：从 F 中移除子句 C 的过程称为忘记，这里子句 C 由 F 推出。

在解决实际 CNF 命题的可满足性时，这些规则的使用顺序可根据实际需要视具体情况而定，不是固

定不变的。下面举一个简单的例子，介绍 DPLL 算法规则的使用。

例 1：判断 CNF 命题 $\{\bar{1}2,\bar{3}4,\overline{56},\overline{256}\}$ 的可满足性。

s1. $\varnothing \| \bar{1}2,\bar{3}4,\overline{56},\overline{256}$ (R3)

s2. $1^d \| \bar{1}2,\bar{3}4,\overline{56},\overline{256}$ (R1)

s3. $1^d 2 \| \bar{1}2,\bar{3}4,\overline{56},\overline{256}$ (R3)

s4. $1^d 23^d \| \bar{1}2,\bar{3}4,\overline{56},\overline{256}$ (R1)

s5. $1^d 23^d 4 \| \bar{1}2,\bar{3}4,\overline{56},\overline{256}$ (R3)

s6. $1^d 23^d 45^d \| \bar{1}2,\bar{3}4,\overline{56},\overline{256}$ (R1)

s7. $1^d 23^d 45^d \bar{6} \| \bar{1}2,\bar{3}4,\overline{56},\overline{256}$ (R5)

s8. $1^d 23^d 4\bar{5} \| \bar{1}2,\bar{3}4,\overline{56},\overline{256}$ (R1)

s9. $1^d 23^d 4\overline{56}(\bar{6}) \| \bar{1}2,\bar{3}4,\overline{56},\overline{256}$

上述命题是可满足的，可满足解为 $\{123456\}=\{111101\}$ 或 $\{123456\}=\{111100\}$。DPLL 算法由空的赋值表开始，首先应用 R3 规则任意选取一个文字进行判定，此处选择文字 1，并将其赋值为 1，在第一个子句中，再应用 R1 规则将文字 2 繁衍为真。当 $\{12\}=\{11\}$ 时，无法进行下一步的赋值推导，所以在第 2 个子句中应用 R3 规则将文字 3 赋值为 1，再应用 R1 规则将文字 4 繁衍为真。依次下去在第 6 步推导后得到 $\{123456\}=\{111110\}$，但出现冲突子句 $\overline{256}$，应用 R5 规则取消 5^d 及由它繁衍得到的 $\bar{6}$，并将 $\bar{5}$ 加入到赋值表中。由于 CNF 中子句的特点，只要每个子句中有一个文字为真，那么该子句就为真，因此，在子句 $\overline{256}$ 中由于 $\bar{5}$ 为真，那么该子句一定为真，所以文字 6 繁衍为 0 或 1 均可。综上，CNF 的每个子句都为真，故该命题是可满足的，最后得到的可满足解为 $\{123456\}=\{111101\}$ 或 $\{123456\}=\{111100\}$。

3. 算法流程

由 CHAFF [16]估计可知，虽然变量决策所占的时间比较少，但却决定着整个搜索空间和搜索时间的大小，所以对它的改进是必不可少的。目前存在的策略可分为以下两种：

1) 随机选取变量的策略。因为它简单易操作，无额外计算等优点备受研究者们青睐，但由于它对变量选取的不确定性，所以在解决某些规模相同的实例时，所耗费的步骤和时间会有很大的差异，并不能总是高效地找到可满足解。

2) 排序选取变量的策略。它是由某种策略计算出变量或子句的某一特征，然后按照计算出的结果进行初始排序，再根据这种策略进行变量的选取。虽然这种方法在选取变量之前增加了一定的计算量，但对后序变量的正确选择具有一定的指向作用，可以在很大的程度上减少推导所用的时间，可以降低整个搜索空间和搜索时间的大小，因此在求解推导中占有一定的优势。

文中采用的是排序选取变量的策略，即在选取判定变量之前先根据子句的长度对 CNF 实例进行分组，然后计算出每个变量的出现次数，挑选出出现次数最高的变量，再计算次数最高的变量中正负文字的出现次数，并对次数最高的进行判定，该排序方法在一定程度上保证了决策变量选取的准确性。

根据在排序选取变量的过程中变量的顺序是否发生改变，可将排序过程分为静态和动态两种。静态排序是指在推导过程中，按照命题给出的变量顺序排序，在推导过程中始终保持这个顺序不变。但随着 CNF 实例规模的增大以及推导过程中学习规则和忘记规则的使用，使得静态排序已经无法满足快速求解的需求，因此变量的动态排序是一种必然的趋势。动态排序是指变量在推导过程中根据某种决策按照变量的特征进行排序，它可以有效解决静态排序的缺点，及时地反应变量的特征，且在推导过程中可以降低求解时间，提高求解效率。

文中采用就是一种动态排序思想，详述如下：

(1) 记录每个子句的长度，根据子句的长度对命题公式 φ 进行分组：

第一组：当 $|c_i|=1$ 时；

第二组：当 $|c_i|=2$ 时；

第三组：剩余的子句。

(2) 记录子句中每个变量的出现次数，记录的初值为 0，变量每出现一次，数值相应地加上 1。

求解时，对第一组直接赋值：若变量为正出现，直接赋值为 1；反之，赋值为 0；在第二、三组中根据记录的每个变量的出现次数进行排序，先选定出现频率较高的变量进行判断，再由变量正负文字出现次数高低进行赋值：若正文字出现次数高，赋值为 1；若负文字出现次数高，赋值为 0；若两个出现次数一样多，随机选取一个进行赋值，然后再运用 DPLL 算法中的基本规则进行赋值推导。

(3) 分组原则：$F=\{C_1,C_2,\cdots,C_n\}$，其中 $C_1,C_2,\cdots,C_n$ 为子句，子句长度记为 $|C_i|$。

$S_1=\cup_i C_i$，$|C_i|=1$，长度为 1 的子句的集合；

$S_2=\cup_j C_j$，$|C_j|=2$，长度为 2 的子句的集合；

$S_3=S-S_1-S_2$，长度大于 2 的子句的集合。

分组后满足：

① $S_i\cap S_j=\varnothing$，$i,j=1,2,3$。分组后各子句集的交集为空集。

② $S_1\cup S_2\cup S_3=\varphi$，分组后各子句集的析取仍为命题公式 φ。

通过引入上述动态排序的思想，并在 DPLL 算法的基础上，提出了一种变量决策启发策略算法，其具体过程见算法 1。

变量决策启发策略算法较之原算法更加高效。原算法从运算步骤上来说，当命题公式 φ 中含有 n 个变量时，运算时隐含地要经历 2^n 步，求解时耗费的空间和运算量都相当大；而新算法在分组赋值后的第一组就已经减少了一些步骤，因为当文字个数为 1 时，要使 CNF 为可满足的，必须将其赋值为真，这样就排除了原算法中将该文字的相反出现赋值为真的情况，从而避免了不必要的计算，减少了求解步骤，同样按照新算法在第二、三组中又剪除了一部分求解步骤，所以新算法的求解步骤最多为 $2^{n_1}+2^{n_2}+2^{n_3}$ 步，其中 n_1 为第一组中所含变量的个数，n_2 为第二组中不同于第一组中的变量的个数，n_3 为第三组中不同于第一、二组中的变量的个数，$n_1+n_2+n_3=n$，这远远小于原来的 2^n 步，从而新算法能极大地提高求解速度。

4. 算法实例

使用变量决策启发策略算法可有效缩减求解步骤，提高求解效率。下面以具体实例进行说明。

4.1. DPLL 算法与新算法的比较

例 2：判断 CNF 命题 $\varphi_1=\{\overline{12},23,\overline{13}4,2\overline{34},14\}$ 的可满足性。

(一) DPLL 算法:

s1. $\varnothing \,||\, \overline{12},23,\overline{13}4,2\overline{34},14$ (R3)

s2. $1^d \,||\, \overline{12},23,\overline{13}4,2\overline{34},14$ (R1)

s3. $1^d\overline{2} \,||\, \overline{12},23,\overline{13}4,2\overline{34},14$ (R1)

s4. $1^d\overline{2}3 \,||\, \overline{12},23,\overline{13}4,2\overline{34},14$ (R1)

s5. $1^d\overline{2}34 \,||\, \overline{12},23,\overline{13}4,2\overline{34},14$ (R5)

s6. $\overline{1} \,||\, \overline{12},23,\overline{13}4,2\overline{34},14$ (R1)

s7. $\overline{14} \,||\, \overline{12},23,\overline{13}4,2\overline{34},14$ (R3)

s8. $\overline{14}\overline{3}^d \,||\, \overline{12},23,\overline{13}4,2\overline{34},14$ (R1)

s9. $\overline{14}\overline{3}^d2 \,||\, \overline{12},23,\overline{13}4,2\overline{34},14$

Algorithm 1. The algorithm of heuristic strategies for deciding variables
算法 1. 变量决策启发策略算法

输入：CNF 命题 φ。

输出：输出一组可满足解或证明命题不可满足。

1. 根据子句长度对其进行分组：若长度为 1，为一组；若长度为 2，为一组；余下的为一组。
2. 计算 φ 中每个变量的出现次数，并进行初始排序。
3. 对长度为 1 的子句优先赋值：若变量为正出现，赋值为 1；反之，赋值为 0。
4. 对长度为 2 的一组进行赋值推导：若上一步的赋值结果可以运用 DPLL 算法中的基本规则进行推导，那么先进行推导，若无法进行下去时，再挑选出出现次数最多的变量(若最多的有好几个，则随机选出一个输出)作为判定文字，并判断正负文字的出现次数：若正出现 > 负出现，文字赋值为 1；反之，赋值为 0(若正负出现一样多，赋值为 0 或 1 均可)。然后再运用 DPLL 算法中的基本规则进行推导，当推导无法进行下去时，观察余下的一组子句。
5. 对余下的一组子句按照第 4 步的方法进行推导，直到完成所有变量的赋值。
6. 若所有变量的赋值结果使得 CNF 命题可满足，那么得到的赋值表即为所求的一组可满足解。若所有变量的赋值结果应用 R4 规则得到冲突子句，那么这个命题 φ 为不可满足。

上述命题 φ_1 是可满足的，可满足解为 $\{1234\}=\{0101\}$。

（二）新算法：

s1. $\varnothing \parallel \overline{12},23,\overline{134},2\overline{34},14$ (R3)

s2. $\overline{1}^{d} \parallel \overline{12},23,\overline{134},2\overline{34},14$ (R1)

s3. $\overline{1}^{d}4 \parallel \overline{12},23,\overline{134},2\overline{34},14$ (R3)

s4. $\overline{1}^{d}4\overline{3}^{d} \parallel \overline{12},23,\overline{134},2\overline{34},14$ (R1)

s5. $\overline{1}^{d}4\overline{3}^{d}2 \parallel \overline{12},23,\overline{134},2\overline{34},14$

所以 φ_1 是可满足的，可满足解为 $\{1234\}=\{0101\}$。

新算法具体推导过程如下：

1 依据子句长度对其进行分组，将 $\overline{12},23,14$ 分为第一组，$\overline{134},2\overline{34}$ 分为第二组。

2 计算命题中每个变量的出现次数，得到 $n(1)=3$，$n(2)=3$，$n(3)=3$，$n(4)=3$。

3 在第一组，依据新算法先挑选出判定文字。由于文字 1,2,3,4 出现次数一样多，故任意选取一个变量即可，此处选择观察变量 1，因为 $n(\overline{1})>n(1)$，所以文字 $\overline{1}$ 赋值为真，然后通过 R1 规则推导得到文字 4 为真。类似地，用同样方法将文字 $\overline{3}$ 赋值为真，并通过 R1 规则繁衍出文字 2 为真。

4 所以命题 φ_1 是可满足的，得到的可满足解为 $\{1234\}=\{0101\}$。

例 3：判断 CNF 命题 $\varphi_2=\{6,\overline{7},1\overline{23},\overline{4}52,\overline{69}\overline{78},87\overline{5},\overline{6}84,\overline{14},57\overline{3}\}$ 的可满足性。

(一) DPLL 算法：

s1. $\varnothing \parallel 6,\overline{7},1\overline{23},\overline{4}52,\overline{69}\overline{78},87\overline{5},\overline{6}84,\overline{14},57\overline{3}$ (R3)

s2. $6^{d} \parallel 6,\overline{7},1\overline{23},\overline{4}52,\overline{69}\overline{78},87\overline{5},\overline{6}84,\overline{14},57\overline{3}$ (R3)

s3. $6^{d}\overline{7}^{d} \parallel 6,\overline{7},1\overline{23},\overline{4}52,\overline{69}\overline{78},87\overline{5},\overline{6}84,\overline{14},57\overline{3}$ (R3)

s4. $6^{d}\overline{7}^{d}9^{d} \parallel 6,\overline{7},1\overline{23},\overline{4}52,\overline{69}\overline{78},87\overline{5},\overline{6}84,\overline{14},57\overline{3}$ (R1)

s5. $6^{d}\overline{7}^{d}9^{d}\overline{8} \parallel 6,\overline{7},1\overline{23},\overline{4}52,\overline{69}\overline{78},87\overline{5},\overline{6}84,\overline{14},57\overline{3}$ (R1)

s6. $6^{d}\overline{7}^{d}9^{d}\overline{85} \parallel 6,\overline{7},1\overline{23},\overline{4}52,\overline{69}\overline{78},87\overline{5},\overline{6}84,\overline{14},57\overline{3}$ (R1)

s7. $6^{d}\overline{7}^{d}9^{d}\overline{85}4 \parallel 6,\overline{7},1\overline{23},\overline{4}52,\overline{69}\overline{78},87\overline{5},\overline{6}84,\overline{14},57\overline{3}$ (R1)

s8. $6^{d}\overline{7}^{d}9^{d}\overline{85}4\overline{1} \parallel 6,\overline{7},1\overline{23},\overline{4}52,\overline{69}\overline{78},87\overline{5},\overline{6}84,\overline{14},57\overline{3}$ (R1)

s9. $6^{d}\overline{7}^{d}9^{d}\overline{85}4\overline{12} \parallel 6,\overline{7},1\overline{23},\overline{4}52,\overline{69}\overline{78},87\overline{5},\overline{6}84,\overline{14},57\overline{3}$ (R1)

s10. $6^{d}\overline{7}^{d}9^{d}\overline{85}4\overline{12}\overline{3} \parallel 6,\overline{7},1\overline{23},\overline{4}52,\overline{69}\overline{78},87\overline{5},\overline{6}84,\overline{14},57\overline{3}$ (R6)

s11. $6^{d}\overline{7}^{d}9^{d}\overline{85}4\overline{12}\overline{3} \parallel 6,\overline{7},1\overline{23},\overline{4}52,\overline{69}\overline{78},87\overline{5},\overline{6}84,\overline{14},57\overline{3},6\overline{78}$ (R5)

s12. $6^{d}\overline{7}^{d}\overline{8} \parallel 6,\overline{7},1\overline{23},\overline{4}52,\overline{69}\overline{78},87\overline{5},\overline{6}84,\overline{14},57\overline{3},6\overline{78}$ (R1)

s13. $6^d\overline{7}^d\overline{89}(9)\|6,\overline{7},1\overline{23},\overline{4}52,\overline{69}\overline{78},87\overline{5},\overline{6}84,\overline{14},57\overline{3},6\overline{78}$ (R1)

s14. $6^d\overline{7}^d\overline{89}(9)\overline{5}\|6,\overline{7},1\overline{23},\overline{4}52,\overline{69}\overline{78},87\overline{5},\overline{6}84,\overline{14},57\overline{3},6\overline{78}$ (R1)

s15. $6^d\overline{7}^d\overline{89}(9)\overline{53}\|6,\overline{7},1\overline{23},\overline{4}52,\overline{69}\overline{78},87\overline{5},\overline{6}84,\overline{14},57\overline{3},6\overline{78}$ (R1)

s16. $6^d\overline{7}^d\overline{89}(9)\overline{534}\|6,\overline{7},1\overline{23},\overline{4}52,\overline{69}\overline{78},87\overline{5},\overline{6}84,\overline{14},57\overline{3},6\overline{78}$ (R1)

s17. $6^d\overline{7}^d\overline{89}(9)\overline{534}\,\overline{1}\|6,\overline{7},1\overline{23},\overline{4}52,\overline{69}\overline{78},87\overline{5},\overline{6}84,\overline{14},57\overline{3},6\overline{78}$ (R1)

s18. $6^d\overline{7}^d\overline{89}(9)\overline{534}\,\overline{12}\|6,\overline{7},1\overline{23},\overline{4}52,\overline{69}\overline{78},87\overline{5},\overline{6}84,\overline{14},57\overline{3},6\overline{78}$

上述命题 φ_2 是可满足的，可满足解是 $\{123456789\}=\{000101000\}$ 或 $\{123456789\}=\{000101001\}$ 。

(二) 新算法：

s1. $\varnothing\|6,\overline{7},1\overline{23},\overline{4}52,\overline{69}\overline{78},87\overline{5},\overline{6}84,\overline{14},57\overline{3}$ (R3)

s2. $\overline{7}^d\|6,\overline{7},1\overline{23},\overline{4}52,\overline{69}\overline{78},87\overline{5},\overline{6}84,\overline{14},57\overline{3}$ (R3)

s3. $\overline{7}^d6^d\|6,\overline{7},1\overline{23},\overline{4}52,\overline{69}\overline{78},87\overline{5},\overline{6}84,\overline{14},57\overline{3}$ (R3)

s4. $\overline{7}^d6^d\overline{4}^d\|6,\overline{7},1\overline{23},\overline{4}52,\overline{69}\overline{78},87\overline{5},\overline{6}84,\overline{14},57\overline{3}$ (R1)

s5. $\overline{7}^d6^d\overline{4}^d8\|6,\overline{7},1\overline{23},\overline{4}52,\overline{69}\overline{78},87\overline{5},\overline{6}84,\overline{14},57\overline{3}$ (R1)

s6. $\overline{7}^d6^d\overline{4}^d8\overline{9}\|6,\overline{7},1\overline{23},\overline{4}52,\overline{69}\overline{78},87\overline{5},\overline{6}84,\overline{14},57\overline{3}$ (R3)

s7. $\overline{7}^d6^d\overline{4}^d8\overline{9}5^d\|6,\overline{7},1\overline{23},\overline{4}52,\overline{69}\overline{78},87\overline{5},\overline{6}84,\overline{14},57\overline{3}$ (R3)

s8. $\overline{7}^d6^d\overline{4}^d8\overline{9}5^d1^d\|6,\overline{7},1\overline{23},\overline{4}52,\overline{69}\overline{78},87\overline{5},\overline{6}84,\overline{14},57\overline{3}$ (R1)

s9. $\overline{7}^d6^d\overline{4}^d8\overline{9}5^d1^d2(\overline{2})\|6,\overline{7},1\overline{23},\overline{4}52,\overline{69}\overline{78},87\overline{5},\overline{6}84,\overline{14},57\overline{3}$ (R1)

s10. $\overline{7}^d6^d\overline{4}^d8\overline{9}5^d1^d2(\overline{2})3(\overline{3})\|6,\overline{7},1\overline{23},\overline{4}52,\overline{69}\overline{78},87\overline{5},\overline{6}84,\overline{14},57\overline{3}$

所以 φ_2 是可满足的，可满足解为 $\{123456789\}=\{100011010\}$ 、$\{123456789\}=\{101011010\}$ 、$\{123456789\}=\{110011010\}$ 或 $\{123456789\}=\{111011010\}$ 。

新算法详细推导过程如下：

1 依据子句长度对其进行分组，将 6 和 $\overline{7}$ 分为第一组； $\overline{14}$ 分为第二组；余下的分为第三组。

2 计算命题中每个变量的出现次数，得到 $n(7)=4$，$n(4)=3$，$n(5)=3$，$n(6)=3$，$n(8)=3$，$n(1)=2$，$n(2)=2$， $n(3)=2$， $n(9)=1$。

3 在第一组中对变量 6 和 $\overline{7}$ 进行赋值，这里变量 6 为正出现，赋值为 1；变量 7 为负出现，赋值为 0。

4 由上一步的赋值结果可知，推导无法进行下去，所以在第二组中选取判定文字，由于 $n(4)>n(1)$，并依据新算法，将变量 4 选取为判定文字，又因为 $n(\overline{4})>n(4)$，所以文字 $\overline{4}$ 赋值为真，但此时变量 1 无法应用 R1 规则进行繁衍，这就需要继续观察余下的一组来进行下一步推导。

5 在前两组中已经得到 $\{674\}=\{100\}$，应用 DPLL 算法中 R1 规则，依次得到 $\{89\}=\{10\}$，此时的赋值结果无法推出剩余变量的确切赋值，所以重新选取判定文字。因为 $n(5)>n(1)=n(2)=n(3)$，故选取变量 5 作为判定文字，又因为 $n(5)>n(\overline{5})$，那么将文字 5 赋值为真。但现在的赋值结果仍无法进行下一步的推导，所以仍需选择新的判定文字。因为 $n(1)=n(2)=n(3)$，故可在这三者中任意选取一个变量作为判定文字，此处选择变量 1，由于 $n(1)=n(\overline{1})$，所以将文字 1 赋值为 1。根据 CNF 中子句的特点，只要每个子句中有一个变量为真，那么该子句就为真，根据上面的描述，CNF 中所有的子句都为真，因此将变量 2 和 3 赋值为 0 或 1 均可使 CNF 命题 φ_2 是可满足的。

6 命题 φ_2 是可满足的，按照新算法求得的可满足解为 $\{123456789\}=\{100011010\}$ 、$\{123456789\}=\{101011010\}$ 、$\{123456789\}=\{110011010\}$ 或 $\{123456789\}=\{111011010\}$ 。

4.2. 实例分析

对比分析例 2 和例 3 在新算法与 DPLL 算法下的赋值推导过程，可以看到，新算法所用步骤明显少

Table 1. Comparison of the results of DPLL algorithm and the new algorithm
表 1. DPLL 算法与新算法的结果对比

实例	R1 (次)		R3 (次)		R5 (次)		R6 (次)	
	DPLL	new	DPLL	new	DPLL	new	DPLL	new
例 2	5	2	2	2	1	0	0	0
例 3	12	4	3	5	1	0	1	0

于原算法所用步骤。在例 2、例 3 中，DPLL 算法分别使用了 8 步和 17 步，而新算法仅仅用了 4 步和 9 步，所用步数几乎是原算法的一半，更重要的是改进后的算法在推导时均比原过程少了一次回溯，这就使得在搜索过程中可以有效地缩减求解步骤，提高求解速度，优化求解效率。通过观察求解结果可知在例 2 中两次赋值结果一样，而在例 3 中两次赋值结果不一样，这是因为在求解 SAT 问题的可满足解时只需找到一组使 CNF 命题为真的赋值即可，所以赋值结果可以不相同。

现将新算法和 DPLL 算法在推导中规则的使用次数进行比较，具体数据见表 1。

由表 1 分析可知：R3 规则的使用次数基本保持不变，且新算法在极大地保证了判定文字准确性的基础上，在一定程度上减少了 R5 和 R6 规则的使用，从而减少了 R1 规则的使用，也避免了同样冲突子句的再次出现，判定文字的准确选取可以有效地减少总的求解步骤，缩短求解时间。因此，变量的决策启发式算法具有可行性。

5. 结语

本文在 DPLL 算法及原有搜索决策策略的基础上，剖析了当前决策层上搜索决策策略存在的问题，提出了一种在决策层中动态排序变量的算法。新算法对子句中判定文字的选取顺序进行启发式的改进，在一定程度上减少了原算法在求解过程中回溯规则和学习规则的使用，降低了规则的使用次数，有效缩减了推导步骤，提高了求解速度，从而使其在解决 SAT 问题时更加高效。然而，这种方法还没有应用到 SAT 问题的求解器中，并不能大规模使用，下一步的工作是要用计算机去实现智能化求解，从而使得变量的决策启发策略算法更加高效和实用。

参考文献 (References)

[1] Davis, M. and Putnam, H. (1960) A Computing Procedure for Quantification Theory. *Journal of the ACM*, **7**, 201-215. http://dx.doi.org/10.1145/321033.321034

[2] Davis, M., Logemann, G. and Loveland, D. (1962) A Machine Program for Theorem Proving. *Communications of the ACM*, **5**, 394-397. http://dx.doi.org/10.1145/368273.368557

[3] 郭庆贺. 基于 DPLL 和 GPS 的同步信号合成及授时系统研究[D]: [硕士学位论文]. 南京: 南京理工大学, 2013.

[4] 李壮. 基于细胞膜演算的伴随子句学习的 DPLL 算法描述[D]: [硕士学位论文]. 长春: 吉林大学, 2015.

[5] Bonacina, M.P. and Johansson, M. (2011) On Interpolation in Decision Procedures. *Automated Reasoning with Analytic Tableaux and Related Methods*: *Lecture Notes in Computer Science*, **6793**, 1-16. http://dx.doi.org/10.1007/978-3-642-22119-4_1

[6] 金继伟, 马菲菲, 张建. SMT 求解技术简述[J]. 计算机科学与探索, 2015(7): 769-780.

[7] 姜波. DPLL 在异形板材预热电源中的应用研究[D]: [硕士学位论文]. 阜新: 辽宁工程技术大学, 2013.

[8] 陈稳. 基于 DPLL 的 SAT 算法的研究及应用[D]: [硕士学位论文]. 成都: 电子科技大学, 2011.

[9] 王洪琳. 基于 DPLL 的#SAT 子类算法的研究[D]: [硕士学位论文]. 长春: 东北师范大学, 2013.

[10] 胡显伟, 任世军. 基于函数变换的求解 SAT 问题的新算法[J]. 智能计算机与应用, 2012, 2(3): 33-36.

[11] 余晓星. 一种 SAT 邻域的快速搜索算法[J]. 现代计算机, 2012(4): 9-11.

[12] 徐云, 陈国良, 张国义. 一种求解难 SAT 问题的改进 DP 算法[J]. 中国科学技术大学学报, 2002, 32(3): 358-362.

[13] 邓晓瑶, 冯志勇, 饶国政, 王鑫. 基于子句文字长度动态约束的变量消除算法[J]. 计算机科学与探索, 2014, 8(11): 1314-1323.

[14] 毕忠勤, 陈光喜, 单美静. 可满足性问题全部解的求解算法[J]. 计算机工程与应用, 2009, 45(3): 35-37.

[15] Nieuwenhuis, R., Oliveras, A. and Tinelli, C. (2006) Solving SAT and SAT Modulo Theories: From an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). *Journal of the ACM*, **53**, 937-977. http://dx.doi.org/10.1145/1217856.1217859

[16] Moskewicz, M.W., Madigan, C.F., Zhao, Y., *et al*. (2001) CHAFF: Engineering an Efficient SAT Solver. *Proceedings of the* 38*th Design Automation Conference*, **17**, 530-535.