计算机科学 1999Vol. 26№. 3

命题逻辑可满足性问题的算法分析*)

The Analysis of Algorithms for the Propositional Logic Satisfiability Problem

李 未 黄 雄
(北京航空航天大学计算机系 北京 100083)

Abstract In this paper we introduce some complete algorithms for the propositional satisfiability problem, including the enumrating algorithm, the Davis-Putnam algorithm, the relaxation algorithm, the counting algorithm and the resolution algorithm. The theoretical analysis of the performances of these algorithms, both the worst-case and the average-case analysis, is also surveyed.

Keywords Satisfiability problem, Algorithms, Design, Analysis, Propositional logic, Computational complexity.

1 引言

可满足性问题(以下简称 SAT)是问:对于一个命题逻辑公式,是否存在对其变元的一个真值赋值使之成立?这个问题在许多领域都有非常重要的意义,其快速求解算法的研究成为计算机科学的中心课题之一。例如在机器定理证明领域,某命题是否是一个和谐的公理集合的推论,这个问题归结为该命题的反面与该公理集合一起是否是不可满足的。通过量词消去技术和 Herbrand 定理的作用,谓词逻辑公式的不可满足性可以归结为命题逻辑公式的不可满足性[1]。在知识库维护中,当知识以逻辑公式的形式表达时,知识库的一致性检查可以归结为命题逻辑公式的可满足性。在开放逻辑中[2~4],新事实是否与已有的知识矛盾,当遇到事实反驳时如何求得最大和谐的知识集,这些问题最后都要归结为命题逻辑公式的可满足性。1971 年 Cook 首次证明了 SAT 是 NP-完全的[5],从而大量的计算问题都可以归约到 SAT。正是由于 SAT 的重要地位,各国学者对它进行了广泛而深入的研究。

但是在 P≠NP 的假设下,SAT 没有多项式时间的算法[6]。目前 SAT 的算法主要分为两类:完全性算法和不完全性算法。完全性算法能够保证正确判定公式的可满足性,但需要耗费指数长的时间。一些完全性算法采用精巧的技术来减小搜索空间,提高时间效率,这类算法是本文的重点。不完全性算法不能保证正确判定可满足性,在某些实例上可能得不到解,但这类算法采用启发式策略指导搜索,普遍比完全性算法算得快。对这类算法的研究是近年来的一股热潮,但本文不介绍这类算法,另有文章专门论述。

2 基本定义与性质

由命题逻辑的常识知道:任何命题逻辑公式都逻辑等价于一个合取范式(*CNF*),因此本文只考虑合取范式的 SAT 问题。下面给出 SAT 的有关定义。

定义 2.1 变元是指形式符号 $x_i, i=0,1,2,\cdots$;文字是指形式符号 $x_i, \overline{x_i}, i=0,1,2,\cdots$;其中 x_i 称为正文字,$\overline{x_i}$ 称为负文字。若 L 是文字,则 L 的反面记为 $\neg L$,定义为:

$$\neg L=\begin{cases}\overline{x_i} & \text{若 } L=x_i,\\ x_i & \text{若 } L=\overline{x_i},\end{cases}$$

子句是文字的有穷集合。若 L 和 $\neg L$ 同时出现在某子句中,称该子句为重言子句。子句中文字的个数称为子句的长度。合取范式(*CNF*)是指子句的有穷集合、所有合取范式的集合记为 *CNF*。k-*CNF*(k=1,2,…)是所有这样的合取范式的集合:其中每个子句的长度不超过 k。

例 2.1 $\{\{x_1,\overline{x_2}\},\{x_4,x_5,\overline{x_1}\},\{x_3,x_4,\overline{x_3}\}\}$ 是一个 3-*CNF* 范式,其中 $\{x_3,x_4,\overline{x_3}\}$ 是重言子句、

*)受国家自然科学基金、博士点基金的资助

子句 $C=\{L_1,L_2,\cdots,L_n\}$ 也记为

$$C=L_1\vee L_2\vee\cdots\vee L_n=\bigvee_{i=1}^{n}L_i$$

即子句中的文字关系解释为析取。CNF 范式 $F=\{C_1,C_2,\cdots,C_m\}$ 也记为

$$F=C_1\wedge C_2\cdots\wedge C_m=\bigwedge_{i=1}^{m}C_i$$

即公式中的子句关系解释为合取。所以上例中的公式也可以写为：

$$(x_1\vee\overline{x_2})\wedge(x_4\vee x_5\vee\overline{x_1})\wedge(x_3\vee x_4\vee\overline{x_3})$$

注意：在上面的定义中，一个文字不能在一个子句中出现多次，一个子句不能在一个公式中出现多次。

定义 2.2 一个真值赋值是一个函数 $A:\{x_i,i=0,1,2\cdots\}\mapsto\{0,1\}$，这里把"1"等同于"真"，"0"等同于"假"，并规定 $\neg 0=1,\neg 1=0$。

定义 2.3 若 A 是一个赋值，L 是一个文字，则定义 A(L) 为：

$$A(L)=\begin{cases}A(x_i) & \text{若 } L=x_i\\ \neg A(x_i) & \text{若 } L=\neg x_i\end{cases}$$

定义 2.4 设子句 $C=\{L_1,L_2,\cdots,L_m\}$，称赋值 A 满足 C 当且仅当 $A(L_1),A(L_2),\cdots,A(L_m)$ 中至少有一个是 1；设公式 $F=\{C_1,C_2,\cdots,C_m\}$，称 A 满足 F 当且仅当 A 满足所有 $C_1,C_2,\cdots,C_m$。特别规定：任意赋值都不满足空子句，任意赋值都满足空公式。

SAT 问题可以定义为：

实例：合取范式 F。

问：是否存在一个赋值满足 F？

若限定 $F\in$ k-CNF，则称为 k-SAT 问题。满足 F 的赋值称为 F 的一个解。若 S 是有穷集，我们以 $|S|$ 表示 S 中元素的个数。

定义 2.5 设 $F\in CNF$，我们以 $V(F)$ 表示 F 中出现的变元的集合，则 $|V(F)|$ 是 F 中变元的个数。$|F|$ 是 F 中所含子句的个数，而以 $L(F)=\sum_{i=1}^{|F|}|C_i|$ 表示 F 的长度，其中 $C_i\in F$。

若有穷集 $S\subseteq\{x_i,i=0,1,\cdots\}$ 包含 $V(F)$，则 S 可以称为 F 的基变元集，每一个公式都有自己的基变元集。同一个公式若关联不同的基变元集，其概率性质也会有所不同。一般地，公式 F 的基变元集假定为 $V(F)$。

显然，赋值 A 对基变元集 S 以外的变元赋什么值对 F 没有影响，因此可以限定 A 只在 S 上有定义，此时不同的真值赋值最多有 $2^{|S|}$ 个。SAT 问题又可以表述为：给定 $F\in CNF$，满足 F 的真值赋值个数是否大于 0？如不特别声明，我们总是假定每一个公式 F 有一个自然的基变元集 $V(F)$，并且只考虑定义在基变元集上的赋值。

定义 2.6 设 $F\in CNF$，L 是一个文字，定义 $F[L/1]$ 为：从 F 中删去包含 L 的子句、从剩余的子句中删去文字 $\neg L$；定义 $F[L/0]$ 为：从 F 中删去包含 $\neg L$ 的子句，从剩余的子句中删去文字 L。

显然，F 是可满足的当且仅当 $F[L/1]$ 或 $F[L/0]$ 是可满足的。若 $L_1,L_2,\cdots,L_k$ 是一组文字且所关联的变元两两不同，$a_i\in\{0,1\},i=1,2,\cdots,k$，则定义

$$F[L_1/a_1,L_2/a_2,\cdots,L_k/a_k]=F[L_1/a_1][L_2/a_2]\cdots[L_k/a_k]$$

定义 2.7 设 $F\in CNF$，L_1,L_2 是两个文字，定义 $F[L_1/L_2]$ 为：把 L_1 同时替换为 L_2，同时把 $\neg L_1$ 替换为 $\neg L_2$。

在 SAT 算法中，经常需要把长公式化简为等价的短公式，有下面两种等价形式：

定义 2.8 设 F_1,F_2 是两个公式，称它们为逻辑等价的，当且仅当 $\forall$ 赋值 A，A 满足 $F_1\Leftrightarrow A$ 满足 F_2，记为 $F_1\equiv F_2$；称它们为可满足性等价的，当且仅当 F_1 是可满足的 $\Leftrightarrow F_2$ 是可满足的，记为 $F_1\sim F_2$。

显然逻辑等价比可满足性等价更强。下面是 SAT 算法经常采用的一些化简策略（设 $F\in CNF$）：

1. 单元子句规则：若 $C=\{L\}$ 是 F 的子句，则 F 变为 $F'=F[L/1]$。

2. 纯文字规则：若文字 L 出现在 F 中，而 $\neg L$ 不出现在 F 中，则 L 称为 F 的纯文字。F 变为 $F'=F[L/1]$。

3. 重言子句规则：若 $C\in F$ 且 C 是重言子句，则从 F 中删去 C 得 $F'=F-C$。

4. 两次出现规则：若 $L\in C_1,\neg L\in C_2$，且 $L,\neg L$ 都不在其它子句中出现，则将 C_1,C_2 合并为 $C'=(C_1-\{L\})\cup(C_2-\{\neg L\})$，$F$ 变为 $F'=F-\{C_1,C_2\}\cup\{C'\}$。

5. 包含规则：若 C_1,C_2 是 F 的子句，且 $C_1\subseteq C_2$，则 F 中删去 C_2，得 $F'=F-\{C_2\}$。

6. 提因子规则：若 C_1,C_2 是 F 的两个子句，且 $C_1=\{L\}\cup B,C_2=\{\neg L\}\cup B$，则以 B 代替 C_1,C_2 得 $F'=F-\{C_1,C_2\}\cup\{B\}$。

7. 异或规则：若 $\{x_1,x_2\},\{\overline{x_1},\overline{x_2}\}$ 是 F 的两个子句，则 F 变为：

$$F'=(F-\{\{x_1,x_2\},\{\overline{x_1},\overline{x_2}\}\})[x_1/\overline{x_2}]$$

8. 同或规则：若 $\{x_1,\overline{x_2}\},\{\overline{x_1},x_2\}$ 是 F 的两个子句，则 F 变为：

$$F'=(F-\{\{x_1,x_2\},\{\overline{x_1},\overline{x_2}\}\})[x_1/x_2]$$

上面这些规则具有以下性质：1）F 与 F' 至少是可满足性等价的，其中重言子句规则、包含规则和提因子规则还保证逻辑等价性；2）每条规则的施用条件是可以在多项式时间内判定的；3）由 F 到 F' 的变换可以在 $L(F)$ 的线性时间内完成；4）$L(F')$ 比 $L(F)$ 严格地小。因此在算法中可以应用这些规则来缩短公式的长度，但并不显著地降低时间效率。

3 穷举法

这是最平凡的一类算法。它考察每个赋值，直到找到满足的真值赋值回答"Yes"，或所有赋值都考察完毕回答"No"。

```
Enum(F∈CNF)
begin
1. for each 赋值 A do
2. if A 满足 F then return(Yes);
3. return(No);
end.
```

显然算法的时间复杂性是 $O(L(F)\cdot 2^{V(F)})$，而且在不可满足的实例上总能达到最坏情况。对上面算法的改进是把所有赋值看成是一棵二叉树，以深度优先法搜索这棵树。

```
Search(F∈CNF)
begin
1. if F 是空公式 then return(Yes);
2. if F 包含空子句 then return(No);
3. 从 V(F)中选择变量 x;
4. if Search(F[x/1])=Yes
5. then return(Yes);
6. if Search(F[x/0])=Yes
7. then return(Yes);
8. return(No);
end.
```

算法 Search 的优点是：一旦在部分赋值上发现公式是可满足的或不可满足的，则该部分赋值以下的整棵子树都不必搜索了。虽然最坏性能与算法 Enum 一样，但平均性能有了很大改进。

4 Davis-Putnam 算法

此算法由 M. Davis 和 H. Putanam 于 1960 年首次提出[5]，现已成为最著名的 SAT 算法。算法如下：

```
DP(F∈CNF)
begin
1. if F=∅ then return(Yes);
2. if F 包含空子句 then return(No);
3. (单元子句规则)if F 包含子句{L}
4. then return DP(F[L/1]);
5. (纯文字规则)if L 是 F 的纯文字
6. then return DP(F[L/1]);
7. 在 F 中任选一个文字 L;
8. if DP(F[L/1])=Yes
9. then return(Yes);
10. else return DP(F[L/0]);
end.
```

算法的正确性不难验证。关于最坏性能，考虑下面的公式：

$$F_0=\{\{x_0,x_1\},\{x_0,\overline{x_1}\},\{\overline{x_0},x_1\},\{\overline{x_0},\overline{x_1}\}\}$$

$$F_n=\{\{x_{2n},\overline{x_{2n+1}}\},\{\overline{x_{2n}},x_{2n+1}\}\}\cup F_{n-1}.$$

显然 F_n 是不可满足的（$n\geq 0$）。算法 DP 在 F_n 上运行，递归调用次数不小于 2^n，而 $L(F_n)=O(n)$，所以 DP 的最坏性能是指数的。就我们目前所知，除了平凡的时间复杂性上界 $O(L(F)\cdot 2^{V(F)})$ 以外，还未能分析出更小的时间复杂性上界。

对算法做平均性能分析一般要做两件事：一是假定 CNF 上的一个概率分布，不同的概率分布将导致不同的分析结果；二是要对算法 DP 做某种简化，直接分析困难很大，简化后的 DP 算法的时间效率一般比原 DP 算法的低。若简化后的 DP 算法的平均时间复杂性是多项式的，则原 DP 算法的平均性能也是多项式的；但是若前者是指数的，则并不能证明后者不是多项式的。

通常一个概率分布由三个参数指定：t—子句数，v—变元数，p—(0,1)区间的一个常数。基变元集 S 假定为 $\{x_1,x_2,\cdots,x_v\}$。最常用的概率分布有以下三种：

1. 对于每个子句 $C_i(i=1,2,\cdots,t)$，每个变元 x_j $(j=1,2,\cdots,v)$正出现在 C_i 中的概率是 p，负出现的概率是 p，不出现的概率是 $1-2p$。这里要求 $p\in(0,\frac{1}{2})$。这种分布排除了重言子句出现的可能性。文[8、9]分析了 DP 算法在这种分布下的性能，他们对 DP 算法做了简化：去掉单元子句规则，并固定变元的选择顺序为 $x_1,x_2,\cdots,x_v$。结果表明：简化后的 DP 算法的平均时间复杂性是多项式的。但如果同时去掉纯文字规则，则简化后的 DP 算法（实际上变为 Search 算法）的平均性能是指数的。

文[10]中证明这种概率分布在某种意义上是不合理的，因为由它产生的随机实例具有这样的性质：在所有 2^v 个赋值中，绝大部分都是满足的赋值！为此他们定义了第二种概率分布。

2. 这种分布是针对 k-CNF 的。首先限定 t 与 v 线性相关，即 $v=[\lambda t]$，λ 是某个正实数。每个子句如下选取：从 S 中随机选取 k 个变元（共有 (k^v) 种可能，每种可能是等概率的），每个变元以 1/2 的概率正出现，以 1/2 的概率负出现。文[10]的结果表明：如果去掉纯文字规则，DP 算法是指数的。

3. 最后这种概率分布定义如下：t 个子句独立产生，每个子句的产生方式为：对于 $2v$ 个文字（包括正文字和负文字），每个文字以概率 p 出现，以概率 $(1-p)$ 不出现。这种分布允许公式中包含重言子句。一般地，t 与 p 都作为 v 的某个函数，因而随着 t，v 和 λ 定义的不同，就定义出 CNF 上的不同概率分布。

对 DP 算法的简化为：去掉单元子句规则，并固定变元的选择顺序为：$x_1, x_2, \cdots, x_v$。对这种简化的 DP 算法在上面分布下的平均性能分析表明：随着上面定义的分布的不同，算法的平均时间复杂性也不同，对某些分布，算法的平均性能是多项式的。对另一些分布则是指数的[11~13]。

对 DP 算法的概率分析可以为不完全的启发式算法的分析提供有用的技巧[14,15]。另一个好处是有助于辨识出那些难解的情况和易解的情况，从而为改进算法提供有价值的信息。例如文[16]中提出一种改进算法平均性能的策略：删除出现次数少的变元。具体做法是：除了用单元子句规则和纯文字规则外，还加上重言子句规则和两次出现规则。经过这四条规则的作用，实例中的每个变元都至少出现三次。此时即使不再采用这些规则，直接采用穷举法也能获得较好的平均性能[16]。DP 算法其实是 Search 算法的精化，第 2 节中的那些规则都可以应用在算法中，虽然不能改善最坏性能，却可以极大地改善平均性能。

5 松驰法

松驰法的简单形式如下：

```
SRelax(F∈CNF)
begin
1. if f=∅ then return(Yes);
2. 从 F 中找到最短的子句 C={L1,…,Lk};
3. if C=∅ then return(No);
4. (分支)for i=1 to k do begin
5. E:=F[L1/0,…,Li-1/0,Li/1];
6. if SRelax(E)=Yes then return(Yes);
7. end;
8. return(No);
end.
```

由于当 F 被满足时，每个子句中至少有一个文字赋值为 1，所以算法的正确性不难验证。当 $F\in k$-CNF 时，循环 4 最多执行 k 次。执行第 i 次时 $(1\leqslant i\leqslant k)$，$E$ 中变元数为 $|V(F)|-i$。若令 T_n 表示 SRelax 在变元数不超过 n 的 k-CNF 公式上运行所进行的最大递归次数，则有 $T_n=1+T_{n-1}+T_{n-2}+\cdots+T_{n-k}(n>k)$，这是推广的 k 阶 Fibonacci 序列，其增长速度可以界定为不超过 ξ_1^n，这里 ξ_k 是方程 $x^{k+1}=2x^k-1$ 的最大实根[17]。读者不难计算 $\xi_2\approx 1.62$，$\xi_3\approx 1.84$，$\xi_4\approx 1.93$，$\cdots$，$\xi_9\approx 1.99$。因为除了递归调用以外，SRelax 的其它语句只耗费线性时间，所以 SRelax 在 k-CNF 公式上运行的时间复杂性为 $O(L(F)\cdot\xi_1^{|V(F)|})$[17]。

对 SRelax 的改进基于自足赋值的概念，它是对纯文字规则的一种推广。基变元集 S 上的一个部分赋值 A（只对 S 中的部分变元有定义的赋值）称为对 F 自足的，当且仅当 F 中每个子句，若含有在 A 下有定义的变元，则均已被 A 满足。显然，若 A 是对 F 自足的赋值，则删除那些被满足的子句，保留那些未满足的子句，得到 F'，有：$F\sim F'$。因此当遇到自足赋值时，分支是不必要的。这样我们可以先判断是否是自足赋值（这是可以在线性时间内完成的），若不是，再分支，得到下面的算法：

```
Relax(F∈CNF)
begin
1. if F=∅ then return(Yes);
2. 从 F 中找到最短的子句 C={L1,…,Lk};
3. if C=∅ then return(No);
4. for i=1 to k do begin
5. if[L1/0,…,Li/1]诱导出自足赋值
6. then begin
7.    E:=F[L1/0,…,Li/1];
8.    return Relax(E);
9.    end;
10. end;
11. for i=1 to k do begin
12. E:=F[L1/0,…,Li/1];
13. if Relax(E)=Yes then return(Yes);
14. end;
15. return(No);
end.
```

循环 4 寻找自足赋值，若找到则不用分支，否则循环 11 负责分支。注意循环 11 执行时必有：$[L_1/0,\cdots L_i/1]$ $(1\leqslant i\leqslant k)$ 诱导出的赋值都不是自足赋值，由自足赋值的定义可知：对于任意 $i=1,2,\cdots k$，$F[L_1/0,\cdots,L_i/1]$ 中必含有一个子句 C_i，是 F 中相应的子句 C_i 的真子集（C_i-C_i 中的文字都已赋值为 0）。因此当 Relax 运行在 k-CNF 公式上时，语句 13 的公式 E 中至少有一个子句的长度不超过 $k-1$，由于语句 2 的作用，在 E 上的调用最多将 E 分裂为 $k-1$ 个子问题。若以 T_n 表示 Relax 在变元数不超过 n 的 k-CNF 公式上运行所进行的最大递归次数，则有：$T'_n=1+T'_{n-1}+\cdots+T'_{n-k+1}(n>k)$。这是推广的 $(k-1)$ 阶 Fibonacci 序列，其增长速度不超过 ξ_{k-1}。这样 Relax 在 k-CNF 公式上运行的时间复杂性为 $O(L(F)\cdot\xi_1^{|V(F)|})$[17]。

比较 SRelax 和 Relax 在 3-CNF 上运行的时间复杂性，前者为 $O(L(F)\cdot 1.84^{|V(F)|})$，后者为 $O(L(F)\cdot 1.62^{|V(F)|})$，有了显著的改进。随后在算法上有了一系列改进，使得底数不断下降：文[18]使底数降为 1.597，文[19]得到 1.571，文[20]得到 1.5045，文[21]得到 1.497，据说有希望达到 $\sqrt{2}=1.414\cdots$。虽然时间复杂性降低了，但算法越来越复杂，算法分析也越来越复杂，文[20]的原始分析长达 53 页。

注意到 ξ_k 随着 k 的增大很快地逼近 2，因此 Relax 在一般的 CNF 公式上运行的时间复杂性为 $O(L(F)\cdot 2^{|V(F)|})$。目前还未能找到在一般的 CNF 公式上运行时间复杂性底数严格小于 2 的算法。

和前面的算法一样，SRelax 和 Relax 在返回 Yes 时，成功地递归调用路径上给出满足输入公式的(部分)赋值。

6 计数法

这类算法的基本思想是计算实例的解的个数，若大于 0 则可满足，否则不可满足。由于重言子句可以在线性时间内识别出来，所以本节我们假定不出现重言子句。

定义 6.1 两个子句 C_1、C_2 称为相交的，当且仅当存在文字 L，使得 $L\in C_1$ 且 $\neg L\in C_2$。不相交的子句称为独立的。

定义 6.2 子句集合 $F=\{C_1,\cdots,C_r\}$ 称为团，当且仅当每个 C_i、C_j 相交($i\neq j$)。称为独立集当且仅当每个 C_i、C_j 独立($i\neq j$)。特别地，一个子句的集合既是团也是独立集，(注意：这里的术语和文[22,23]一致，与文[24]恰好相反)。

定义 6.3 赋值 A 称为被公式 F 抑制，当且仅当 A 不满足 F。

性质：设 A_i 是子句 C_i 抑制的赋值的集合($i=1,2$)，则 $A_1\cap A_2=\varnothing$ 当且仅当 C_1、C_2 相交。

设基变元集为 S，$|S|=v$，$F=\{C_1,\cdots,C_t\}$。令 $A_i(1\leqslant i\leqslant t)$ 表示被 C_i 抑制的赋值的集合，则被 F 抑制的赋值的集合的大小为：

$$|\bigcup_{i=1}^{t}A_i|=\sum_{i=1}^{t}[(-1)^{i-1}\cdot\sum_{1\leqslant i_1<i_2<\cdots<i_i\leqslant t}|A_{i_1}\cap A_{i_2}\cdots A_{i_i}|] \quad (1)$$

其中 $A_{i_1}\cap A_{i_2}\cdots\cap A_{i_i}$ 是被每个 $C_{i_k}(1\leqslant k\leqslant i)$ 抑制的赋值的集合。当 $\{C_{i_1},\cdots,C_{i_i}\}$ 不是独立集时，即存在 C_{i_l}、$C_{i_m}(1\leqslant l\neq m\leqslant i)$ 相交，则 $A_{i_l}\cap A_{i_m}=\varnothing$，即 $A_{i_1}\cap A_{i_2}\cdots\cap A_{i_i}=\varnothing$，所以(1)式只需对独立集求和即可。令 $IND(i)$ 表示所有 i 阶独立集的集合($1\leqslant i\leqslant t$)，对于每个独立集 $X\subseteq F$，不满足 X 中所有子句的赋值个数为 $2^{v-|V(X)|}$，因此(1)式变为：

$$|\bigcup_{i=1}^{t}A_i|=\sum_{i=1}^{t}[(-1)^{i-1}\sum_{X\in IND(i)}2^{v-|V(X)|}] \quad (2)$$

算法就是对 $|\bigcup_{i=1}^{t}A_i|$ 进行计算[22,23]：

```
Iwama(F∈CNF)
begin
1. v:=|V(F)|;
2. sum:=0;
3 IND(1):={{C}|C∈F 且 C 不是重言子句};
4. for i=1 to |F| do begin
5.    sum:=sum+(-1)^{i-1}·Σ_{X∈IND(i)} 2^{v-|V(X)|};
6.    if i 是奇数且 sum<2^v
7.    then return(Yes);
/* 计算 IND(i+1) */
8. IND(i+1):=∅;
9. for each C∈F do
10 for each X∈IND(i)do
11.    if X∪{C}是独立集
12.    then 把 X∪{C}加入 IND(i+1);
13 if IND(i+1)=∅ then break;
14. end;
15. if sum<2^v then return(Yes)
16. else return(No);
end.
```

说明：令 $I_k=\sum_{i=1}^{k}[(-1)^{i-1}\sum_{X\in IND(i)}2^{v-|V(X)|}]$，则 $I_t=|\bigcup_{i=1}^{t}A_i|$。当 k 是奇数时，$I_k\geqslant I_t$，所以第 6 句一旦发现奇数项已小于 2^v，就可以立即返回 Yes。8-12 句计算 IND($i+1$)，基于这样的事实：$i+1$ 阶独立集的 i 阶子集一定是 i 阶独立集，13 句基于这个事实：如果没有 i 阶独立集，那么一定没有($i+1$)阶独立集。

令 LIT(X)表示独立集 X 中的文字的集合，则 LIT(X)是非重言子句，$X\cup\{C\}$ 是独立集当且仅当 LIT(X)与 C 独立。这样 Iwama 中对每个独立集只需保存 LIT(X)即可。第 11 句耗费 $O(|C|)$ 长时间。采用合适的数据结构，算法的最坏时间复杂性为 $O(L(F)\cdot\sum_{i=1}^{|F|}|IND(i)|)$。若 F 是独立集，则时间复杂性为 $O(L(F)\cdot 2^{|F|})$，看起来很耗时，但是当子句彼此不独立时，Iwama 能获得较好的平均性能。极端地，当子句两两相交时，循环 4 只执行 1 次，循环 9 执行 $|F|$ 次，循环 10 执行 $|F|$ 次，所以时间耗费为 $O(L(F)\cdot|F|)$。实际上，因为只有 1 阶独立集，所以，$|\bigcup_{i=1}^{|F|}A_i|=\sum_{i=1}^{|F|}|A_i|=\sum_{i=1}^{|F|}2^{v-|C_i|}$，因此只需判定是否 $\sum_{i=1}^{|F|}2^{-|C_i|}<1$ 即可。

看起来 Iwama 能在其它算法很难算的实例上表现出优越的性能，因此不失为对其它算法的一个补充。对 Iwama 的概率分析表明[25]：Iwama 的确能

在其它算法很困难的概率分布上算得很快。Iwama 的另一个特点是：它在可满足的实例上并不能找到解。

上文提到当 F 是团时，F 抑制的赋值个数为

$$\sum_{i=1}^{t} 2^{n-|C_i|} \tag{3}$$

而对一般的 $F\in CNF$ 必须按照(2)式计算。另一种计算 $|\bigcup_{i=1}^{t} A_i|$ 的办法是：把一般的 $F\in CNF$ 变换为逻辑等价的团 F'，然后按照(3)式计算被抑制的赋值个数[24]。变换的思想如下：设 C_1, C_2 是两个独立的子句，$C_1-C_2=\{L_1,\cdots,L_p\}$，有：

$$\begin{aligned}
C_1\wedge C_2 &\equiv C_1\wedge(C_2\vee\neg L_1)\wedge(C_2\vee L_1)\\
&\equiv C_1\wedge(C_2\vee\neg L_1)\wedge\\
&\quad(C_2\vee L_1\vee\neg L_2)\wedge\\
&\quad(C_2\vee L_1\vee L_2)\\
&\quad\vdots\\
&\equiv C_1\wedge(C_2\vee\neg L_1)\wedge\cdots\\
&\quad\wedge(C_2\vee L_1\cdots\vee\neg L_p)\wedge\\
&\quad(C_2\vee L_1\vee L_2\cdots\vee L_p)\\
&\equiv C_1\wedge(C_2\vee\neg L_1)\wedge\\
&\quad(C_2\vee L_1\vee\neg L_2)\cdots\wedge\\
&\quad(C_2\vee L_2\vee L_2\cdots\vee\neg L_p)
\end{aligned}\tag{4}$$

最后一行是因为 $C_1\subseteq C_2\cup\{L_1,L_2,\cdots,L_p\}$。4 式是一个团，子句集合 $\Theta=\{C_2\vee\neg L_1, C_2\vee L_1\vee\neg L_2,\cdots, C_2\vee L_1\vee L_2\cdots\vee\neg L_p\}$ 恰好抑制赋值 A_2-A_1，这里 A_i 是 C_i 抑制的赋值集合($i=1,2$)。因此 C_2 可以被 Θ 代替。由 C_1, C_2 产生 Θ 的算法如下：

```
I2(C1,C2,Θ)
begin
1. if C1,C2 相交 then return Θ={C2};
2. φ:=C1-C2;
3. if φ=∅ then return Θ:=∅;
/* 设 φ={L1,L2,…,Lp} */
4. Θ:=∅;W:=C2;
5. for i=1 to p do begin
6.    U:=W∪{¬Li};
7.    Θ:=Θ∪{U};
8.    W:=W∪{Li};
9. end;
10. retunr Θ;
end.
```

采用合适的数据结构，12 执行时间为 $O(max\{|C_1|,|C_2|\})$。注意：Θ 中每个子句都包含 C_2。为了把这一方法推广到一般的子句集合 $F=\{C_1,C_2,\cdots,C_t\}$，我们分两个层次：第一层次用一个团 Θ_t 代替 C_t，使得每个 $C_i(1\leqslant i\leqslant t-1)$ 与 Θ_t 中每个子句都相交；在第二层次，逐次向前推进 i，用团 Θ_{i-1} 代替 C_{i-1} 使得每个 $C_i(1\leqslant i\leqslant t-2)$ 与 Θ_{i-1} 中每个子句都相交。注意到，由于 Θ_{i-1} 中每个子句都包含 C_{i-1}，而 C_{i-1} 与 Θ_i 中每个子句相交，所以 Θ_{i-1} 中每个子句与 Θ_i 中每个子句相交，即 $\Theta_{i-1}\cup\Theta_i$ 是一个团。依此类推，直到 C_2 被替换。C_1 是不必替换的。这样问题归结为如何求出 Θ_i 来代替 C_i，使得每个 $C_i(1\leqslant i\leqslant t-1)$ 与 Θ_i 中每个子句相交，利用 12 求 Θ_i 的算法如下：

```
/* 求团 Θi 使得：1){C1,…,Ci}≡{C1,…,Ci-1}∪Θi;
2)每个 Ci(1≤i≤t-1)与 Θi 中每个子句相交。*/
PROC({C1,…,Ci-1},Ci,Θi)
begin
1. if i=1 then return Θi:={Ci};
2. I2(Ci-1,Ci,Ωi);
3. if Ωi=∅ then return Θi:=∅;
4. if i=2 then return Θi:=Ωi;
5. Θi:=∅;
6. for each C∈Ωi do begin
7.    PROC({C1,…,Ci-2},C,Ωi);
8.    Θi:=Θi∪Ωi;
9. end;
10. return Θi;
end.
```

不难对 i 归纳证明 Θ_i 满足条件 1)，2)且 Θ_i 中每个子句都包含 C_i。设最长子句长度为 $k(k\geqslant 2)$，基变元集 S 的大小为 $|S|=n$。PROC 的最坏时间复杂性为 $O(t\cdot k\cdot 2^k\cdot\xi_1')$，其中 ξ_1' 是 $x^{k+1}-2x^k+1=0$ 的最大实根[24]，与第 5 节的 ξ_1 相同。

把一般子句集合变换为团的算法如下：

```
CLIQUE({C1,…,Ct},Θ)
begin
1. Θ:=∅;
2. for i=t to 2 do begin
3.    PROC({C1,…,Ci-1},Ci,Ωi);
4.    Θ:=Θ∪Ωi;
5. end.
6. Θ:=Θ∪{C1};
7. return Θ;
end.
```

CLIQUE 的时间复杂性为 $O(\sum_{i=2}^{t}\cdot k\cdot 2^k\cdot\xi_1')=O(t^2\cdot k\cdot 2^k\cdot\xi_1')$。求出 Θ 后，根据 3 式即可算出解的个数。注意到 k 是常数时，此算法耗时 $O(t^2\cdot\xi_1')$。另一种更巧妙的办法是直接利用 PROC($\{C_1,\cdots,C_{i-1}\},C_i,\Theta_i$)。令 A_i 是 C_i 抑制的赋值集合($1\leqslant i\leqslant t$)，则 Θ_i 抑制的赋值集合为 $A=A_i-\bigcup_{i=1}^{t}A_i$。当 C_i 是空子句时，A_i 是所有赋值的集合，因此 A 恰好是 $\{C_1,\cdots,C_t\}$ 的解的集合。这样得到另一个算法：

```
COUNTING(F,Θ)
begin
1. PROC(F,∅,Θ);
2. return;Θ
end.
```

F 是可满足的当且仅当 Θ 非空，根据(3)式可以算出 F 的解的个数。进一步，因为 Θ 是一个团，我们可以很容易列举出 F 的所有解，即 Θ 抑制的赋值集合，这是 Iwama 和 CLIQUE 所没有的优点。COUNTING 的时间复杂性为 $O(t\cdot k\cdot 2^k\cdot\xi_1')$。

COUNTING 的缺点是空间耗费太多。若只需要判定 F 的可满足性，则不必存储整个 Θ。注意到 PROC$(F,\varnothing,\Theta)$执行过程中，Θ 单调扩张，因此一旦发现 Θ 非空，即可返回 Yes。经过这样的简化，再把 PROC 中的 I2 展开，就得到下面的算法：

```
DEC({C1,…,Ci-1},Ci)
begin
1. if i=1 then return(Yes);
2. if Ci-1⊆Ci then return(No);
3. if i=2 then return(Yes);
4. if Ci-1 与 Ci 相交
5. then return DEC({C1,…,Ci-2},Cr);
/* 设 Ci-1-Ci={L1,…,Lp} */
6. for t=1 to p do begin
7.    C:=Ci∪{L1,L2,…,¬Lt};
8.    if DEC({C1,…,Ci-2},C)=Yes
9.    then return(Yes);
10. end
11. return(No);
end.
```

对照 PROC 可以看出：

$$DEC(\{C_1,\cdots,C_{i-1}\},C_i)=Yes$$

当且仅当

$$PROC(\{C_1,\cdots,C_{i-1}\},C_i,\Theta_i)$$

执行完毕时 Θ_i 非空。因此得到下面的算法：

```
Dubois(F∈CNF)
begin
return DEC(F,∅);
end.
```

我们将此算法命名为 Dubois 是因为它出现在文[24]中。类似的算法出现在文[19]中。读者不难对 i 归纳证明：$DEC(\{C_1,\cdots,C_{i-1}\},C_i)=Yes$ 当且仅当在不满足 C_i 的条件下$\{C_1,\cdots C_{i-1}\}$是可满足的。仔细分析 Dubois 不难看出：Dubois 与 SRelax 本质上是相同的。

7 归结法

归结(resolutions，亦消解)法是研究得最广泛的一类算法[25]，其一般形式针对谓词逻辑公式，包含合一算法在内，我们只讨论它在命题逻辑中的形式。归结法的基本思想是：设 C_1,C_2 是两个子句，$x\in C_1,\bar{x}\in C_2$，则 $C=(C_1-\{x\})\cup(C_2-\{\bar{x}\})$满足：$C_1\wedge C_2\models C$。由 C_1,C_2 得到 C 的这一过程称为归结，C 称为归结式，C_1,C_2 称为父子句，记为 $C_1\wedge C_2\overset{R}{\vdash}C$。显然，若还有 $x_i(\neq x)$使得 $x_i\in C_1,\bar{x_i}\in C_2$，则 C 将是一个重言子句。因此一般要求做归结的父子句有且只有一对互补的文字。

设 $F\in CNF$，记 $R(F)=F\cup\{C\mid\exists\ C_1,C_2\in F,C_1\wedge C_2\overset{R}{\vdash}C\}$，显然 $F\equiv R(F)$。重复这一过程得 $F\equiv R(F)\equiv R^2(F)\equiv\cdots\equiv R^n(F)\cdots$，同时 $F\subseteq R(F)\subseteq R^2(F)\cdots\subseteq R^n(F)\cdots$。但由于基变元集 S 上的子句最多有 2^{2S} 个，因此 CNF 公式最多有 $2^{2^{2S}}$ 个。所以上面的序列必是有穷的。设 $R^n(F)=R^{n+1}(F)$是最大的集合。若 $R^n(F)$中含有空子句，由 $F\equiv R^n(F)$知 F 必不可满足(这是归结法的正确性)；若 $R^n(F)$中不含有空子句，由可以构造出 S 的一个赋值满足 $R^n(F)$[26](这称为归结的反驳完全性)。因此 F 不可满足当且仅当 $R^n(F)$含有空子句。这就是下面的算法：

```
Resolution(F∈CNF)
begin
1. R:=F;
2. repeat
3.    δ:={C|∃ C1,C2∈R,C1∧C2⊢R C};
4.    R:=R∪δ;
5.    if R 含有空子句 then return(NO);
6. until δ=∅;
7. return(Yes);
end.
```

算法的执行时间显然是指数的。Resolution 要产生所有的归结式，其中许多归结式是"无用的"，因此可以采用多种启发式策略来减少无用归结式。但是可以证明：无论用什么样的启发式策略，基于归结原理的算法必定是指数时间复杂性的。这里涉及到"证明长度"的概念。我们称子句序列 $C_1,C_2,\cdots,C_n$ 是一个归结证明，当且仅当每个 $C_i(1\leqslant i\leqslant n)$要么属于 F，要么存在 $C_j,C_k,1\leqslant j,k<i$，使得 $C_1\wedge C_k\overset{R}{\vdash}C_i$。$C_n$ 是此证明的结论，n 是它的长度。对于不可满足的公式 F，必定存在一个推理出空子句的最短的归结证明，其长度是任何基于归结原理的算法耗费时间的下界。文[27]首次证明一类不可满足的 CNF 公式具有这样的性质：它们的最短的推理出空子句的归结证明长度是指数的。这类公式刻画的就是鸽巢原理。对于每个 $n\geqslant 2$，考虑公式 F_n，其包含的子句如下：

$$\begin{cases}\{(x_{i1},x_{i2},\cdots,x_{in-1}) & 1\leqslant i\leqslant n\\ \{\neg x_{ik},\neg x_{jk}\} & 1\leqslant i<j\leqslant n\\ & 1\leqslant k\leqslant n-1\end{cases}$$

直观上，x_{ij}表示第 i 个鸽子放在第 j 个巢内$(1\leqslant i\leqslant n,1\leqslant j\leqslant n-1)$。第一类子句是说：$n$ 个鸽子都落在 $n-1$ 个巢内，第二类子句是说：每个巢内不能放两个不同的鸽子。根据鸽巢原理，F_n 是不可满足的。文[27]证明 F_n 的每个归结证明的长度都是 n 的指数。文[28]～[31]对这些结果进行了改进，表明 3-CNF 公式中也存在一类公式使得最短归结证明的长度是公式长度的指数。文[31]的结果甚至表明：绝

大部分不可满足的 CNF 公式的归结证明长度都是公式长度的指数！文[31]用的概率分布是4节中的第二种概率分布，设 k 是最长子句的长度，v 是变元数，t 是子句数，在这种分布下，每个赋值满足随机选择的子句的概率是 $(1-1/2^k)$，所以它满足整个公式的概率是 $(1-1/2^k)^t$，整个公式可满足的概率不超过 $2^v(1-1/2^k)^t$，设 $t=c\cdot v$，则此概率不超过 $[2(1-1/2^k)^c]^v$，当 $k\geqslant 3, c\geqslant 0.7\cdot 2^k$ 时，此概率不超过 $[2(1-1/2^k)^{0.7\cdot 2^k}]^v\leqslant(2e^{-0.7})^v<0.999^v$。因此当 $v\to\infty$ 时，随机选择的公式不可满足的概率趋于1，文[31]证明在此条件下，最短归结证明的长度是 v 的指数的概率也趋于1。这就是说随着 v 的增大，大部分随机选择的公式的最短归结证明长度是指数的。

总的来说，归结法用于求解SAT问题是低效的。尽管如此，人们还是提出了大量的启发式策略来减少无用归结式的生成，以提高归结法的效率，我们只简单叙述几个策略。

1. 支持集策略：把输入公式 F 分成两个子公式 F_S 和 $(F-F_S)$，要求 $(F-F_S)$ 是可满足的，F_S 称为支持集，每次归结时要求两个子句不能同时来自 $(F-F_S)$。这样由 $(F-F_S)$ 中的子句得到的归结式都被排除了。其直观思想是：由于 $(F-F_S)$ 是可满足的，由 $(F-F_S)$ 中的子句归结出的子句对于最后归结出空子句没有什么帮助。文[32]证明支持集策略是反驳完全的，即当 F 不可满足时，此策略总能推理出空子句。有多种办法确定支持集。例如选择一个赋值 A，F 中不被 A 满足的子句归于 F_S，满足的子句归于 $(F-F_S)$。文[32]的结果表明此策略能显著减少归结式的数目。

2. 语义归结：这是对支持集策略的推广。首先选择一个赋值 A，每次归结时必须是一个父子句被 A 满足，另一个父子句不被 A 满足。可以证明：语义归结也是反驳完全的。

3 单元归结：此策略要求每次归结时两个父子句中必须有单元子句。由于归结式一定是两个父子句中较长的那一个的真子集，因此可以保留归结式而删去较长的父子句，这使得输入公式缩短了，所以单元归结过程总可以在多项式时间内完成。单元归结不是反驳完全的。例如：$F=\{\{x_1,x_2\},\{x_1,\overline{x_2}\},\{\overline{x_1},x_2\},\{\overline{x_1},\overline{x_2}\}\}$ 是不可满足的，而单元归结不可能得出空子句。但是对于一类称为 Horn 公式的 CNF 公式，即限定每个子句中最多出现一个正文字的公式，单元归结是反驳完全的，而且可以在线性时间内判断 Horn 公式的可满足性[33]。

4. 扩展归结：其基本思想是在推理过程中允许引进缩写。设 x 是一个不在公式中出现的变量，L_1，L_2，…L_k 是公式中的文字，$\beta(L_1,\cdots,L_k)$ 是由 L_1，…，L_k 组成的某个公式，则 $x\equiv\beta(L_1,\cdots,L_k)$ 表示 x 是公式 $\beta(L_1,\cdots,L_k)$ 的缩写，将 $x\equiv\beta(L_1,\cdots,L_k)$ 变成 CNF 公式以后加入原来的公式中一起进行推理，这相当于在数学证明过程中引入新的定义，借助新概念来完成证明，直观上这有助于缩短证明的长度。文[34]证明：对于刻画鸽巢原理的公式 F_n，在扩展归结中存在多项式长度的归结证明。

启发式策略一般可以减少无用归结式的生成，但是对这些策略的理论分析一般难度较大。

参考文献

1 石纯一等．人工智能原理．北京：清华大学出版社，1993

2 李未 一个开放的逻辑系统，中国科学(A辑)，1992(10)：1103～1113

3 Li Wei. The inductive process: A logical framework for inductive inference. Science in China (Series A), Supplement 1995, 38: 12～27

4 Zhang Yuping, Li Wei. An operational approach to belief revision. J. Computer Science and Technology, 1996, 11(2): 97～107

5 Cook S A. The Complexity of theorem-proving procedures. In: Proc. 3rd Ann. ACM Symp. on Theory of Computing. 1971. 151～158

6 Garey M R, Johnson D S. Computers and Intractability, A Guide to the theory of NP-Completeness. W. H. Freeman and Company, San Francisco, 1979

7 Davis M, Putnam H. A Computing Procedure for Quantification Theory. Journal of the ACM, 1960, 7: 201～215

8 Goldberg A, et al Average Time Analyses of Simplified Davis-Putnam Procedures. Information Processing Letters, 1982, 15(2): 72～75

9 Goldberg A, et al Corrigendum Avergae Time Analysis of simplified Davis-Putnam Procedures. Information Processing Letters, 1983, 16: 213

10 Franco J. Probabilistic Analysis of the Davis Putnam Procedure for Solving the Satisfiability Problem Discrete Applied Mathematics, 1983, 5: 77～87

11 Purdom P, Brown C. The pure literal rule and polynomial average time. SIAM Jouranal on Computing, 1985, 14(4): 943～953

12 Bugrara K, Purdom P. An exponential lower bound for

the pure literal rule Information Processing Letters, 1988,27:215～219
13 Bugrara,K,et al. Exponential average time for the pure literal rule SIAM Journal on Computing,1989,18(2):409～418
14 Franco J,Ho Yuan Chuan Probabilistic performance of a heuristic for the satisfiability problem. Discrete Applied Mathmatics,1988/1989,22:35～51
15 Frieze A,Suen S Analysis of two simple heuristics on a random instance of k-SAT Journal of Algorithms, 1996,20:312～355
16 Franco J. Elimination of infrequent variables improves average case performance of satisfiability algorithms. SIAM Journal on Computing,1991,20(6),1119～1127
17 Monien B, Speckenmeyer E Solving satisfiability in less than 2^n steps, Discrete Applied Mathematics, 1985,10:287～295
18 Schiermeyer I. Solving 3-satisfiability in less than 1.579^n steps. In. Proc. 6th Workshop Computer Science Logic,LNCS 702,1993:379～394
19 Zhang Wenhui . Number of models and satisfiability of sets of clauses Theoretical Computer Science, 1996, 155:277～288
20 Kullmann O. New methods for 3-SAT decision and worst-case analysis. Theoretical Computer Science, 1998
21 Schiermeyer I. Pure literal look ahead:an $O(1.497^n)$3-satisfiability algorithm (extended abstract), Feb. 27, 1996,unpublished manuscript
22 Iwama K. Complementary approaches to CNF Boolean equations,in Discrete Algorithms and Complexity, Academic Press Inc. ,New York,1987. 223～236
23 Iwama K. CNF Satisfiability Test by counting and polynomial average time. SIAM Journal on Computing, 1989,18(2) 385～391
24 Dubois O. Counting the number of solutions for instances of satisfiability Theoretical Computer Science, 1991,81 49～64
25 刘叙华 基于归结方法的自动推理. 北京:科学出版社,1994
26 Robinson J A A machine-oriented logic based on the resolution principle. Journal of the ACM,1965,12(1), 23～41
27 Haken A. The intractability of resolution Theoretical Computer Science,1985,39:297～308
28 Urquhart A. Hard Examples for Resolution. Journal of the ACM,1987,34(1),209～219
29 Buss S R, Turán G. Resolution proofs of generalized pigeonhole principles. Theoretical Computer Science, 1988,62:311～317
30 Cook S A, Pitassi T. A feasibly constructive lower bound for resolution proofs. Information Processing Letters,1990,34:81～85
31 Chvátal V,Szemerédi E. Many hard examples for resolution proofs. Information Processing Letters, 1990, 34:81～85
32 Wos L,et al. Efficiency and completeness of the set of support strategy in theorem proving. Journal of the ACM,1965,12:536～541
33 Henschen L, Wos L. Unit refutations and Horn sets. Journal of the ACM,1974,21(4):590～605
34 Cook S A. A short proof of the pigeon hole principle using extended resolution. SIGACT News,1976,8:28～32

(上接第 62 页)

化系统应用的一个重要意义同时也在于培养一批应用信息化技术的管理经营人才和工程技术人才。

结论 21 世纪计算机网络将全球化,制造业也将发展到从获取信息到产品分析设计、选购原辅材料和零部件、加工制造,直至营销服务,整个生产过程的全球化。因此,建立基于 Intranet 技术的先进制造技术信息网络系统已势在必行。从上述分析可得出的结论是:现代制造业的发展是与全球范围内的信息化发展结合在一起的,Intranet 技术已经成为制造企业建设信息化系统的重要技术因素之一,也将是企业走向国际化、实施并行工程和科学化管理的基础设施条件。

参 考 文 献

1 Hannam R Computer Integrated Manufacturing:form concepts to realisation. Addison-wesley,1996
2 郭自新. 正在来临的信息网络化时代. 世界制造技术与装备市场,1996(3):11～13
3 左元. Intranet 下管理信息系统维护的优势. 计算机世界,1997-11-17(F7)
4 毛伟,张文辉,袁小春. WorldWideWeb 编程渐进. 北京:清华大学出版社,1997
5 Mcdonald J. CAD/CAM/CAE on the World Wide Web. Mechanical Engineering, 1997,119(9):12～19