基于寻找可满足 2-SAT 子问题的 SAT 算法 *

傅阳春, 周育人
(华南理工大学 计算机科学与工程学院, 广州 510006)

摘　要: 可满足问题(SAT)是一个 NP-Hard 问题。提出了一种求解 SAT 的新算法(FFSAT)。该算法将 SAT 问题转换为寻找一个可满足的 2-SAT 子问题。SAT 问题虽然是 NP 完全问题,但是当所有子句长度不大于 2 时,SAT 问题可以在线性时间求解。使用 2-SAT 算法-BinSat 求解 2-SAT 子问题,当它不满足时,根据赋值选择新的 2-SAT 子问题。实验结果表明,采用本算法的结果优于 UnitWalk。

关键词: SAT 问题; 2-SAT 子问题; 2-SAT 算法
中图分类号: TP301.6　　**文献标志码**: A　　**文章编号**: 1001-3695(2010)02-0462-03
doi:10.3969/j.issn.1001-3695.2010.02.015

New SAT solver based on finding satisfiable 2-SAT sub problem

FU Yang-chun, ZHOU Yu-ren
(*School of Computer Science & Engineering, South China University of Technology, Guangzhou* 510006, *China*)

Abstract: Satisfiability (SAT) problem is one of the NP-Hard problems. This paper introduced a new SAT solver called FS-SAT. This SAT solver solved the problem by searching a satisfiable 2-SAT sub problem. SAT was NP-complete, but it can be solved in linear time when the given formula contains only binary clauses (2-SAT). BinSat(2-SAT solver) was used to solve the 2-SAT sub problem and improved the 2-SAT sub problem according to the truth assignment. The experimental results show that the solver outperforms UnitWalk.

Key words: SAT problem; 2-SAT sub problem; 2-SAT solver

0　引言

命题逻辑合取范式(CNF)的可满足性问题(SAT)是当代理论计算机科学的核心问题。SAT 问题是典型的 NP 完全问题[1],其快速求解方法不仅具有重要的理论意义,还有着直接应用;它的研究成果广泛应用于电子设计自动化、人工智能等领域 ,可解决自动测试向量生成、时序分析、逻辑验证、等价性检查、智能规划等问题。

目前 SAT 问题的算法有完全算法和不完全算法两大类[2]。前者主要是以 Davis-Putnam[3]算法为原型的一类 DP 算法,后者主要基于局部搜索算法。完全算法虽然能保证判定 SAT 问题的可满足性,但是在最坏情况下将达到指数级的时间复杂度,不适合求解大规模的 SAT 问题。局部搜索算法虽然不能判定一个 SAT 问题不满足,但在 SAT 问题可满足时可快速得到它的一个解,因此近年来获得了广泛的研究,其成果丰富,如 Selman 等人的 GSAT[4] 和 WALKSAT[5] 、Edward 的 UnitWalk[6]等。

虽然 SAT 问题是 NP 完全问题,当子句长度大于等于 3 时寻找其解需要花费指数级的时间,但在每个子句的长度小于等于 2(2-SAT)的情况下是可以在线性时间判定其可满足性的。基于单元归结的 BinSat[7] 算法是目前解决 2-SAT 问题的最好算法。Zhang 等人[8]将其应用到 DPLL 算法中,有效地减少了搜索空间。UnitWalk 也通过使用 2-SAT 算法减少了变量翻转的次数。可满足的 2-SAT 子问题的解也是原 SAT 问题的一个解。基于这个思想,本文提出一种新的基于寻找可满足 2-SAT 子问题的局部搜索算法,并与 WALKSAT、UnitWalk 进行实验对比。实验结果表明该算法优于 UnitWalk 算法。

1　SAT 问题描述和转换

1.1　SAT 问题描述

下面给出 SAT 问题的一般性描述和本文用到的符号。

定义 1　变元集合

用符号 X 表示命题变元的集合,若 X 由 n 个变元 $x_1, x_2, x_3, \cdots, x_n$组成,那么 $X = \{x_1, x_2, x_3, \cdots, x_n\}$,用 $n = |X|$表示变元集合的大小。

定义 2　真值指派

X 的一个真值指派 A 是映射 $X \rightarrow \{0,1\}^n$,变元 x_i在 A 下为真,可以表示为 $A(x_i) = 1$,否则 $A(x_i) = 0$。因此在 X 上存在 2^n 个不同的真值指派。

定义 3　文字

对任意变元 x_i,符号 x_i和$\neg x_i$是其文字,称 x_i是正文字,$\neg x_i$是反文字。用 L 表示文字集。文字 x_i在真值指派 A 下取真值当且仅当 $A(x_i) = 1$;文字$\neg x_i$真值指派 A 下取真值当且仅当 $A(x_i) = 0$。

定义 4　子句

X 上的子句是 X 上的一些文字的析取,用 c 表示,$c = \bigvee_{i=1}^{k} l_i$。

收稿日期: 2009-05-29; **修回日期**: 2009-06-29　**基金项目**: 国家自然科学基金资助项目(60673062,60873078)
作者简介: 傅阳春(1984-),男,硕士研究生,主要研究方向为演化计算、粒子群算法、NP 难问题等(dickenf@163.com);周育人,男,副教授,硕导,博士,主要研究方向为演化计算、数据挖掘、智能计算等.

子句在指派 A 下取真值（或称在指派 A 下是可满足的），当且仅当子句包含的文字中至少有一个在指派 A 下取真值。$k=|c|$ 表示子句 c 中的文字数，称为子句长度。一个长度为 k 的子句是一个 k-子句。

定义 5　子句集

子句集 C 是由 X 上的子句组成的集合，$C=\{c_1,c_2,\cdots,c_m\}$。$m=|C|$ 表示子句集中子句的个数。子句集 C 在指派 A 下是可满足的，当且仅当 C 中所有的子句 c 在指派 A 下都是取真值的。

定义 6　合取范式（conjectured normal formula，CNF）

X 上的合取范式 F 是 X 上的一些子句的合取，$F=\bigwedge_{j=1}^{m} c_j$。合取范式 F 在指派 A 下取真值（或者称在指派 A 下是可满足的），当且仅当 F 中包含的所有子句 c 在指派 A 下都是取真值的。

定义 7　SAT 问题

给定一个命题变元的集合 $X=\{x_1,x_2,x_3,\cdots,x_n\}$ 和一个 X 上的合取范式 $F=\bigwedge_{j=1}^{m} c_j$，问是否存在一个关于 X 的真值指派 A，使得 F 是可满足的。

定义 8　子句的文字集

子句 c 中包含的文字构成的集合记为 $V(c)$，即 $V(c)=\{l_1,l_2,\cdots,l_k\}$，若 $c=l_1 \vee l_2 \vee \cdots \vee l_k$。

定义 9　2-SAT 子问题

给定 SAT 问题 T，其命题变元的集合为 $X=\{x_1,x_2,x_3,\cdots,x_n\}$，子句集为 $C=\{c_1,c_2,\cdots,c_m\}$，称 T' 是其 2-SAT 子问题。如果其命题变元的集合是 X，子句集为 $C'=\{c_1',c_2',\cdots,c_m'\}$，且 $V(c_j')\subset V(c_j)\wedge|c_j'|=2,\forall j\in\{1,2,3,\cdots,m\}$。

1.2　SAT 问题的转换

根据 2-SAT 子问题 T' 的定义可以得出，如果 T' 是可满足的，那么它的解也是原 SAT 问题 T 的一个解。于是求解 SAT 问题可以转换为：寻找一个 2-SAT 子问题 T'，然后使用 BinSat 判断其可满足性，若 T' 是可满足的，那么它的解为原问题的一个解；若 T' 是不满足的，使用本文后面提到的调整策略，替换部分文字形成改进的 T'。重复以上过程，直到寻找到可满足的 2-SAT 子问题 T' 为止。

2　FSSAT 算法概述

FSSAT 包含了两个主要的步骤，即使用 2-SAT 算法求解 2-SAT 子问题和选择 2-SAT 子问题。下面将详细介绍这两个过程。

2.1　线性时间的 2-SAT 算法

经典的基于猜测和推导的 2-SAT 算法在 1976 年由 Even 等人[9]提出，它是一个时间复杂度为 $O(|X||C|)$ 的算法。Alvaro[7]对这个算法进行了改进，提出了一个新的 2-SAT 算法——BinSat，其求解时间仅为 $O(|C|)$。算法描述如下：

```
procedure   PropUnit(T)
      while(∃x and ∃x̄y∈T)
         T:=(T-{x̄y})∪{y};
      return T
end
procedure Temp PropUnit(x)
   if tempval(x)=false
   then T:=PropUnit(T∪{x}) return;
   tempval(x):=true;
   tempval(¬x):=false;
   foreach ȳx∈T do:
      if tempval(y)≠true then Temp PropUnit(y);
procedure BinSat(T)
foreach variable p of T do:
   tempval(p):=tempval(p):=NIL;
   permval(p):=permval(p):=NIL;
T:=PropUnit(T);
while(∃x permval(x)=tempval(x)=NIL)do:
   Temp PropUnit(x);
if □∈T
then return Unsatisfiable;
else return Satisfiable;
```

BinSat 是一种基于单元归结（或称为布尔约束推导）的算法。单元归结作为一种优化策略，能够根据赋值对子句进行化简。例如，对于子句 $x_1 \vee \neg x_2 \vee x_3$，如果将变元 x_2 赋值为 1，那么子句将化简为 $x_1 \vee x_3$，而对于含有文字 x_2 的子句则已满足，可以从问题中删去，使原问题子句数相应减少。随着变元不断被赋值，子句中未赋值变元的减少，会出现单元子句，即子句中除了一个文字外，其余所有文字都赋为 0。例如，当 $x_1=0$，$x_2=1$ 时，子句 $x_1 \vee \neg x_2 \vee x_3$ 就成为单元子句 x_3。显然，子句 x_3 要满足，文字 x_3 必须赋值（隐含赋值）为 1，这称为单元子句规则。单元归结就是不断应用单元子句规则，直到问题中不再出现单元子句。在进行单元归结中会使某个子句不满足，即该子句所有文字都被赋值为 0，这时称为出现冲突。

BinSat 中变元的赋值有两种方式，即临时赋值和永久赋值，分别保存在数组 tempval 和 permval 中。BinSat 包含了两个子过程 Prop Unit 和 TempPropUnit。PropUnit 和 TempPropUnit 都是进行单元归结，不同之处是 TempPropUnit 只作临时赋值，而 Pr opUnit 只作永久赋值。首先，选择一个未赋值的文字，在 TempPropUnit 中对该文字进行临时赋值，然后使用单元归结推导该赋值。推导该赋值带来隐含赋值以及该隐含赋值引起的其他隐含赋值，这些隐含赋值都保存在数组 tempval 中。如果在推导过程中产生冲突，例如，当出现要把文字 x_i 赋值为 1 时，而 tempval$[\neg x_i]=1$，这时可以推断出要使 2-SAT 问题满足 x_i 必须赋值为 1，因此调用过程 PropUnit$(T\cup\{x_i\})$，使得 permval$[x_i]=1$。若在 Pr opUnit 进行单元归结时也发生冲突，就可以得出原问题不满足。

当 2-SAT 问题不满足时，BinSat 会在发生冲突时立即返回结果，这样可能存在尚未赋值的变元，得不到足够的信息。因此，本文对 BinSat 进行了修改，即使发生冲突算法也不停止。虽然这时所得到的真值指派 A 不是一个可满足的解，但是对后面选择一个可满足的 2-SAT 子问题是有用的。

BinSat 返回后，真值指派 A 可以这样得到：对于任意变元 $x_i\in X$，若 permval$[x_i]\neq$NIL，则 x_i 赋值为 permval$[x_i]$；若 permval$[x_i]=$NIL，则 x_i 赋值为 tempval$[x_i]$。

2.2　选择 2-SAT 子问题

当 BinSat 返回不满足时，需要重新选择 2-SAT 子问题。本文的选择策略是根据得到的真值指派 A，替换掉 2-SAT 子问题中不为真的文字，使 2-SAT 子问题的子句满足数增加。

首先定义映射 $S:C'\to\{0,1,2\}^m$ 为 2-SAT 问题在真值指派

A 下子句中文字取真的个数,即 $S(c') = A(l_1') + A(l_2')$,$c' = l_1' \vee l_2'$。本文只需考虑 2-SAT 问题中两种类型子句:子句在当前真值指派 A 下不满足($S(c')=0$)和只有一个文字取真值($S(c')=1$)。

对任意 $c_i' \in C'$,$c_i \in C(i=1,2,3,\cdots,m)$($C'$ 为 2-SAT 子问题子句集,C 为原 SAT 问题子句集),有:

a)若 $S(c_i')=0$,对于任意 $l \in V(c_i) - V(c_i')$,若存在 $A(l)=1$,随机选择文字 $l' \in V(c_i')$ 替换为 l,即 $V(c_i') = V(c_i') - \{l'\} \cup \{l\}$。

b)若 $S(c_i')=1$,对于任意 $l \in V(c_i) - V(c_i')$,若存在 $A(l)=1$,把子句 c_i' 中不为真的文字 l' 替换为 l,即 $V(c_i') = V(c_i') - \{l'\} \cup \{l\}$,$l' \in V(c_i') \wedge A(l')=0$;若不存在 $A(l)=1$,随机选择一个文字 $l \in V(c_i) - V(c_i')$,以一定概率 P(本文使用 $P=n/m$)替换子句 c_i' 中不为真的文字 l'。

通过以上两种替换策略,得到新的 2-SAT 子问题 T',使得不满足的子句有更大的概率变得满足。

2.3 算法基本流程

下面给出 FSSAT 基本流程:

a)初始化参数,设置最大循环次数 MAXTRIES。

b)令循环次数 $t=0$,随机选择每个子句中的两个文字生成初始 2-SAT 子问题 T'。

c)调用 BinSat(T')求解 2-SAT 子问题 T',若 BinSat(T')返回满足,循环结束跳转到 e);否则更新真值指派 A。

d)令 $t=t+1$,若 $t \leq$ MAXTRIES,根据 2.2 节的选择策略选择新的 2-SAT 子问题 T',跳转到 c);否则循环结束,未能找到可满足的解。

e)结束,输出结果。

3 实验结果

WALKSAT 和 UnitWalk 是当前国际具有代表性的局部搜索算法。与它们进行对比测试,可以有效反映本文算法的性能。本文测试算法成功运行时间。本文算法采用 C 语言实现,WALKSAT 和 UnitWalk 均从网络下载源码在本机编译运行。本文测试平台为微机 Pentium M 1.7 GHz, 768 MB 内存,操作系统为 Ubuntu 8.04。

本文测试所用的 SAT 问题均是随机生成的 3-SAT 问题,这些实例均来之 SATLIB[10] 网站。根据 Mitchell 等人[11] 的研究,随机生成的难求解 3-SAT 测试样本,当子句数数目与变元数目之比是 4.3 时,SAT 问题的约束即不会过少,也不会过多,它们属于可满足和不满足的概率是相等的,因而这些问题是比较难解的。因此采用这些 3-SAT 测试样本可以有效评价算法的优劣性。本文选用的实例子句数数目与变元数目之比都是 4.3,每个实例测试 100 次,取平均运行时间。

表 1 给出了三种算法的运行时间,其中 n 表示变元数,m 表示子句数。从表 1 中可以看出,表现最好的还是 WALKSAT,在所有问题的解决时间都是最少的。UnitWalk 是 2003 年 SAT 竞赛中随机生成可满足问题方面的获胜者[6],而本文算法明显优于 UnitWalk 算法,在所选择的测试问题都获得了不同程度的提高,而与 WALKSAT 的差距不大,表明本文算法 FSSAT 求解 SAT 问题是有效可行的。

表 1 FSSAT 与 UnitWalk、WALKSAT 的比较结果

合取范式		平均运行时间/s		
n	m	FSSAT	UnitWalk	WALKSAT
100	430	0.005	0.006	0.004
150	645	0.004	0.007	0.003
200	860	0.017	0.041	0.015
250	1 075	0.420	0.839	0.208
300	1 290	0.253	0.566	0.232
350	1 505	0.581	1.045	0.420
400	1 720	0.040	0.065	0.020
450	1 935	0.157	0.302	0.094
500	2 150	0.386	0.839	0.268
600	2 580	0.415	0.567	0.208
700	3 010	0.450	1.256	0.303

4 结束语

本文结合了 2-SAT 算法——BinSat,提出一种新的基于寻找可满足 2-SAT 子问题的新 SAT 算法。通过实验结果表明,本文算法 FSSAT 明显优于著名的 UnitWalk 算法,解决问题花费的时间平均减少了 50%,为解决 SAT 问题提供了一个新的有效途径。如何选择 2-SAT 子问题是影响算法收敛的关键,因此使用更好的启发策略选择 2-SAT 子问题将是未来本文算法的改进方向。

参考文献:

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

[2] GOMES C P, KAUTZ H, SABHARWAL A, *et al*. Satisfiability solvers[M]//Handbook of Knowledge Representation Amsterdam:2007:89-134.

[3] DAVIS M, PUTNAM H. A computing procedure for quantification theory[J]. Communications of the Association for Computing Machinery, 1960, 7(3):201-215.

[4] SELMAN B, LEVESQUE H J, MITCHELL D G. A new method for solving hard satisfiability problems[C]//Proc of the 12th National Conference on Artificial Intelligence. Cambridge:MIT Press, 1992:440-446.

[5] SELMAN B, KAUTZ H, COHEN B. Noise strategies for local search[C]//Proc of the 13th National Conference on Artificial Intelligence. Cambridge:MIT Press, 1994:337-343.

[6] HIRSCH E A, KOJEVNIKOV A. UnitWalk:a new SAT solver that uses local search guided by unit clause elimination[J]. Annals of Mathematics and Artificial Intelligence, 2005, 43(1/4):91-111.

[7] ALVARO D V. On 2-SAT and renamable horn[C]//Proc of the 17th National Conference on Artificial Intelligence. Cambridge:MIT Press, 2000:343-348.

[8] ZHENG Lei, STUCKEY P J. Improving SAT using 2-SAT[J]. Australasian Computer Science Communications, 2002, 24(1):331-340.

[9] EVEN S, ITAI A, SHAMIR A. On the complexity of timetable and multi-commodity flow problems[J]. SIAM Journal of Computing, 1976, 5(4):691-703.

[10] HOOS H H, STÜTZLE T. SATLIB:an online resource for research on SAT[M]. Alabama:IOS Press, 2000:283-292.

[11] MITCHELL D, SELMAN B, LEVESQUE H J. Hard and easy distributions of SAT problems[C]//Proc of the 12th National Conference on Artificial Intelligence. Cambridge:MIT Press, 1992:459-465.