约束满足问题求解的符号OBDD桶消元算法

徐周波[1,2] 古天龙[2] 常 亮[2] 李凤英[1,2]
(西安电子科技大学电子工程学院 西安 710071)[1]
(桂林电子科技大学计算机科学与工程学院 桂林 541004)[2]

摘 要 桶消元算法是求解约束满足问题的一种典型推理方法。针对桶消元算法面临的状态空间爆炸问题,将有序二叉决策图(OBDD)技术与该算法结合起来,给出了约束满足问题的一种求解算法。通过对约束满足问题中变量和域值的编码,将CSP问题转化为命题可满足性问题,给出了约束满足问题的OBDD表示方法;基于桶消元的算法思想,在约束满足问题的OBDD表示的基础上,利用OBDD的"与"操作和"量化"操作等,避免了传统算法中状态的显式枚举,隐式地实现了对CSP的求解。对大量随机生成的测试用例进行了实验分析,结果表明提出的符号算法明显优于桶消元法和符号直接求解法。

关键词 约束满足问题,符号算法,桶消元,有序二叉决策图(OBDD)
中图法分类号 TP301.6 **文献标识码** A

OBDD-based Bucket Elimination Algorithm for Constraint Satisfaction Problem

XU Zhou-bo[1,2] GU Tian-long[2] CHANG Liang[2] LI Feng-ying[1,2]
(School of Electronic Engineering, Xidian University, Xi'an 710071, China)[1]
(School of Computer Science and Engineering, Guilin University of Electronic Technology, Guilin 541004, China)[2]

Abstract Bucket-elimination algorithm is a typical reasoning method for the constraint satisfaction problem(CSP). Aiming at the state explosion problem of bucket-elimination algorithm, ordered binary decision diagram(OBDD) technique was combined with bucket-elimination algorithm, and a symbolic OBDD-based algorithm for CSP was proposed. By encoding each variable and each value in the domain as binary variables, CSP was encoded as a propositional satisfiability (SAT) problem, and then CSP was formulated symbolically by OBDD. Based on the ideas of bucket-elimination algorithm and the symbolic OBDD representation of CSP, the CSP was solved implicitly by the AND operator and the EXIST operator of OBDD, so that the explicit enumeration of states in traditional algorithms was avoided. The simulation results show that the symbolic algorithm is more efficient than both the bucket-elimination algorithm and the direct algorithm based on OBDD.

Keywords Constraint satisfaction problem, Symbolic algorithm, Bucket elimination, Ordered binary decision diagram (OBDD)

1 引言

约束满足问题(CSP, Constraint Satisfaction Problem)是人工智能中多年来研究的一个重要分支,在生产调度[1]、产品配置[2]、时序推理[3]等领域都有着广泛的应用。然而,约束满足问题通常都是NP难问题。CSP的求解算法可以大致分为两类:基于树的搜索方法和基于推理的方法。基于树的搜索算法[4]主要有回溯算法、回跳算法、向后标记、冲突记录、动态反向回溯和带有预处理技术的搜索算法。基于树的搜索方法是通过对可能的赋值空间进行搜索来求解问题,但算法的时间复杂度往往随着搜索树深度的增加呈指数级增长。基于推理的主要方法是桶消元法(BE, Bucket Elimination)[5]。其实质是通过集合的投影操作对变量进行消元;对于一个含有n个变量的CSP只需通过$n-1$次变量消元便可无回溯的求出CSP的所有解。与基于树的搜索方法相比,桶消元法更为简单、直观,且具有更为一般性的算法框架,但是,其时间和空间复杂度是树宽的指数级。

有序二叉决策图[6](OBDD, Ordered Binary Decision Diagram)作为表示布尔函数的一种新型数据结构,在一定程度上对缓减状态爆炸问题具有明显的效果。如文献[7]利用OBDD解决了显式表示不能处理的大规模问题,使得可处理的系统状态数达到10^{20},而显式表示所能处理的系统状态为

到稿日期:2010-09-25 返修日期:2010-12-23 本文受国家自然科学基金(60963010,60903079,61063002),广西自然科学基金重点项目(0832006Z)资助。

徐周波(1976—),女,博士生,讲师,主要研究方向为符号计算、智能规划,E-mail:xzbli_11@guet.edu.cn;古天龙(1964—),男,教授,博士生导师,主要研究方向为形式化方法、符号计算、协议工程;常 亮(1980—),男,博士,副教授,主要研究方向为知识表示与推理、语义Web、智能主体;李凤英(1974—),女,博士生,讲师,主要研究方向为符号模型检验、Petri网、符号调度技术。

$10^3 \sim 10^6$；文献[8]利用 OBDD 解决了传统算法不能解决的大规模的 0-1 最大网络流问题。在求解命题可满足性问题(SAT,Propositional Satisfiability problem)方面，应用 OBDD 及其扩展结构也取得了一定的研究成果[9]。在 CSP 问题的求解方面，一种直观的方法就是通过对 CSP 中变量和值域的编码将 CSP 转换成 SAT[10]，然后通过 OBDD 的“与”操作重复地将约束组合在一起便可求解到 CSP 的所有解(在下文中将其称为符号直接求解法)；但这种符号直接求解法在计算中可能会使生成的中间积的 OBDD 过于庞大。文献[11]针对装配序列规划问题给出了 CSP 的符号 OBDD 求解技术，其基本思想是将 CSP 的回溯算法与 OBDD 技术相结合，通过 OBDD 的“与”、“或”等操作来验证当前变量的赋值是否满足装配几何可行性约束，但算法的时间复杂度较高。

鉴于此，一方面考虑到 OBDD 的高效存储特性；另一方面，因桶消元法的本质是通过集合的投影操作对变量进行消元，而对集合的投影操作可简单地通过 OBDD 的量化操作来实现。为此，本文利用 OBDD 易操作和高紧凑性的优点，从 CSP 的 OBDD 的符号表示出发，结合桶消元法的思想，给出了 CSP 的符号 OBDD 桶消元算法。通过对大量随机生成的测试用例进行实验分析表明，本文的算法明显优于桶消元法和符号直接求解法。

2 预备知识

2.1 约束满足问题(CSP)

在有限域上的约束满足问题(CSP)可定义为一个三元组 $\langle V,D,C\rangle$，其中：

(1) $V=\{v_1,v_2,\cdots,v_n\}$ 为 n 个变量的有限集；

(2) $D=\{D_1,D_2,\cdots,D_n\}$ 为 n 个变量的值域集合，D_i 为变量 v_i 的值域($i=1,2,\cdots,n$)；

(3) $C=\{c_1,c_2,\cdots,c_m\}$ 为约束关系集，$c_i(v_{i1},v_{i2},\cdots,v_{ij}) \subseteq D_{i1}\times D_{i2}\times\cdots\times D_{ij}$ ($i=1,2,\cdots,m,1\leqslant j\leqslant n$)，$v_{il}\in V(l=1,2,\cdots,j)$，$D_{il}$ 为变量 v_{il} 的域，称 c_i 为定义在变量集 $\{v_{i1},v_{i2},\cdots,v_{ij}\}\subseteq V$ 上的 j 元约束。若 C 中所有的约束均为一元或二元约束，则称该 CSP 为二元 CSP。

本文仅讨论二元 CSP。对于二元 CSP 中一组变量 $\{v_1,v_2,\cdots,v_i\}$ 的一个赋值 $t=\langle a_1,a_2,\cdots,a_i\rangle\in D_1\times D_2\times\cdots\times D_i$ ($1\leqslant i\leqslant n$)，其中 a_i 为变量 v_i 的取值，当 $i=n$ 时，称赋值 t 为 CSP 的一个完全实例化，否则称赋值 t 为 CSP 的一个部分实例化。CSP 求解的目标就是找到变量集 V 的一种或所有完全实例化 t，使得约束关系集 C 中的所有约束均得到满足。

2.2 有序二叉决策图(OBDD)

OBDD 是布尔函数的一种新型图形数据结构。一个 OBDD 就是一个有向无环图，图中的所有结点被分为终结点和非终结点两类。终结点仅有 2 个，即终结点 0 和终结点 1，分别表示布尔常量 0 和 1。一个非终结点 u 对应唯一一个布尔函数 $f(u)$，且有一个标记变量 $var(u)$ 和两条输出边。这两条输出边指向的结点 $low(u)$ 和 $high(u)$ 分别代表函数 $f(u)$ 在 $var(u)$ 取 0 和 1 时的值，即 $f(u)=var(u)\cdot high(u)+var(u)'\cdot low(u)$。在图形表示中，一般用圆圈表示非终结点，用方框表示终结点。指向 $low(u)$ 的边用虚线表示，指向 $high(u)$ 的边用实线表示。通常假设连接边的方向向下，并从图中略去边的方向。

例如，布尔函数 $f=x_1+x_2x_3$ 在变量序 $\pi:x_1<x_2<x_3$ 下所对应的 OBDD 和完全二叉树如图 1 所示。由此可见，用 OBDD 表示布尔函数 $f=x_1+x_2x_3$ 较二叉树具有更高的表示效率。

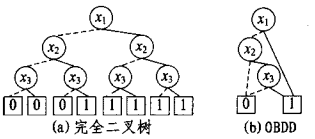

图 1 布尔函数 $f=x_1+x_2x_3$ 的表示

3 约束满足问题的符号 OBDD 描述

因 OBDD 是用于表示布尔函数的一种图形结构，故在 OBDD 描述 CSP 之前，首要解决的问题是将 CSP 用布尔函数来描述，这可通过对 CSP 中的变量和变量域进行布尔编码来实现。设 n 为 CSP$\langle V,D,C\rangle$ 中的变量个数，则对变量 v_i 的值域 D_i 中的每一个值用长度 $l=\lceil\log|D_i|\rceil$ ($i=1,2,\cdots,n$) 的二进制串表示，从而使得 D_i 中的每一个值均可由 $x_i=(x_{i1},\cdots,x_{il})$ 表示，其中 $x_{ik}\in\{0,1\}$ ($k=1,\cdots,l$)。

例如，对于 CSP $\langle V,D,C\rangle$，$V=\{v_1,v_2,v_3\}$，$D_1=\{0,1\}$，$D_2=\{1,2,3\}$，$D_3=\{0,1,2,3\}$，$C=\{c_1,c_2\}$，其中 c_1 为 $(v_1=1)\rightarrow(v_3=3)$，$c_2$ 为 $(v_2=1)\rightarrow(v_3=2)$。则变量 v_1 的值域 D_1 中的每一个值可以用长度为 1 的二进制串表示，从而使得 D_i 中的每一个值均可由 $x_1=(x_{11})$ 表示。若域 D_1 中的值 0 用二进制串 0 表示，1 用二进制串 1 表示，则值 0 可以表示为 x'_{11}，值 1 可以表示为 x_{11}。同理变量 v_2 的值域 D_2 中的每一个值可以由长度为 2 的二进制串表示，设值域 D_2 的值 1,2,3 的编码分别为 00,01 和 10，则值域 D_2 的值 1,2,3 分别可以表示为 $x'_{21}\cdot x'_{22}$，$x'_{21}\cdot x_{22}$，$x_{21}\cdot x'_{22}$；类似地，变量 v_3 的值域 D_3 的值 0,1,2,3 可分别表示为 $x'_{31}\cdot x'_{32}$，$x'_{31}\cdot x_{32}$，$x_{31}\cdot x'_{32}$，$x_{31}\cdot x_{32}$。基于此，约束 c_1 可以表示为布尔函数 $x'_{11}+x_{31}\cdot x_{32}$，约束 c_2 可以表示为布尔函数 $x_{21}+x_{22}+x_{31}\cdot x'_{32}$，其中“$\cdot$”、“$'$”和“$+$”分别表示布尔“与”、“非”和“或”运算。由此可得约束 c_1 和 c_2 的 OBDD 表示，分别如图 2(a)和图 2(b)所示。但在约束的符号表示中，用长度 $l=\lceil\log|D_i|\rceil$ 的二进制串表示值域 D_i 中的值时，总共可表示 2^l 个值，为此对于值个数不是 2 的幂的值域 D_i，还需去除剩余的 $2^l-|D_i|$ 种编码。这可以通过增加下列约束来实现：

$$c_t=\sum_{i=1}^{|D_i|}d_i(x_{i1},x_{i2},\cdots,x_{il}),i=1,\cdots,|D_i| \quad (1)$$

式中，$d_i(x_{i1},\cdots,x_{il})$ 表示值域 D_i 中值 d_i 的布尔函数。

故对于上例，除了对约束 c_1 和 c_2 的符号表示外，需对值域 D_2 增加新的约束 c_3，即

$$c_3=x'_{21}\cdot x'_{22}+x'_{21}\cdot x_{22}+x_{21}\cdot x'_{22}$$

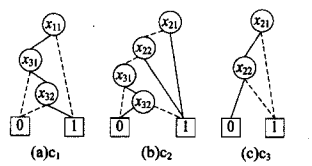

图 2 约束集的符号 OBDD 表示

4 约束满足问题求解的符号 OBDD 技术

基于上述对 CSP 的符号 OBDD 的描述，本文在桶消元算

法思想的基础上，通过 OBDD 的符号操作来求解 CSP，算法的伪代码如图 3 所示。

```
SolveCSP( CSP P)
{
  π=OrderVar(P);//对 P 中的变量按其在约束图中的度从小到大进行排序，得到变量序 π
  for(i=1;i<=n;i++) {//n 为 CSP 中的变量数
      bucket[i]. bdd=BDD(one);//用于存放每一类约束，初始化为常量 1 的 OBDD
      bucket[i]. varscope={i};//用于存放该类约束中所涉及的变量，初始时为变量 i
      if(|Di|不是 2 的幂) bucket[i]. bdd=nc;//nc 为由式(1)生成的约束
  }
  for(i=1;i<=num_constr;i++) {//num_constr 为 CSP 中的约束数
      scope=ConstraintScope(ci) //求约束 ci 所涉及的变量
      v=FirstVarInOrder(scope,π);//根据变量序 π，求在 scope 中的变量序最小的变量
      bucket[v]. bdd=bucket[v]. bdd(BDD(ci);
      bucket[v]. varscope=bucket[v]. varscope∪scope;
      if(bucket[v]==zero) return NULL;//当 bucket[v]为空集时，CSP 无解
  }
  for(i=1;i<n;i++) {
      j=π[i];//从变量序 π 中取出第 i 个变量
      g=∃xj1…∃xjlbucket[j];//其中 l=⌈log|Di|⌉
      scope=bucket[j]. varscope-{j};
      v=FirstVarInOrder(scope,π);
      bucket[v]. bdd=bucket[v]. bdd · g;
      if(bucket[v]. bdd==zero) return NULL;//CSP 无解
      bucket[v]. varscope=bucket[v]. varscope∪scope;
  }
  result=BDD(one);//将 CSP 解初始化为常量 1 的 OBDD
  for(i=n;i>0;i--) {
      result=result · bucket[i]. bdd;
      if(result==zero) return NULL;
  }
  return result;//返回 CSP 的所有解
}
```

图 3　CSP 的符号 OBDD 求解算法

对 CSP 的求解由以下 3 个步骤来实现：

① 对 CSP 中的所有变量根据其在约束图中的度的大小进行递增排序，其中变量的度是指该变量与其他变量之间的约束关系的个数。假设按度的大小递增排序后得到的变量序为 $\pi: v_1 < v_2 < \cdots < v_n$，然后根据此变量序对 CSP 中的各约束进行分类，对于一个含有 n 个变量的 CSP 最多可以将所有约束分为 n 类。假设 v 为约束 c_i 的约束范围中变量序最小的变量，则将约束 c_i 合并到 OBDD 变量 $bucket[v].bdd$ 中，并将约束 c_i 所涉及的变量加入到整形数组 $bucket[v].varscope$ 中。可见，在 $bucket[v].varscope$ 中不包含变量序比 v 小的变量。将约束 c_i 合并到 $bucket[v].bdd$ 中是通过 OBDD 的“与”操作进行实现的。若“与”运算的结果为 OBDD 的终结点 0，则表示不存在满足所有约束的 CSP 解，算法结束。

② 基于变量序 π，对 CSP 中的变量进行消元。先根据以下 OBDD 的量化操作从 $bucket[v_1]$ 中消去变量 v_1，并得到新的约束 g：

$$g=\exists x_{11}\cdots\exists x_{1l}bucket[v_1].bdd$$

式中，$x_{11},\cdots,x_{1l}$ 为变量 x_1 的值域 D_1 的二进制编码对应的布尔变量，$l=\lceil \log|D_1| \rceil$。

设 v 为 g 中的变量序最小的变量，则将新的约束 g 加入到 $bucket[v]$ 中，即：

$$bucket[v].bdd=bucket[v].bdd \cdot g$$

$$bucket[v].varscope=bucket[v].varscope \cup (bucket[v_1].varscope-\{v_1\})$$

在消去变量 v_1 后，接着从 $bucket[v_2]$ 中消去变量 v_2，直到消去变量 v_{n-1}。最后 $bucket[v_n].bdd$ 即为满足所有约束的变量 v_n 的取值。

③ 从变量 v_n 起，根据 π 的逆序将 $bucket[v_i].bdd(i=n, n-1,\cdots,1)$ 逐个进行合取，最后 $bucket[v_1].bdd$ 即为 CSP 的所有解的 OBDD 表示。

例如，对于第 3 节给出的实例，基于上述算法得到 CSP 的所有解的 OBDD 如图 4 所示。图 4 中最右边的路径代表的 CSP 解为：$v_1=1, v_2=2, v_3=3$。

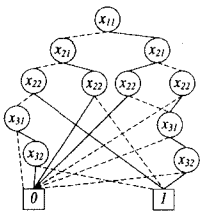

图 4　CSP 的所有解的 OBDD 表示

5　实验结果及分析

为检验本文算法的性能，本文在 Windows XP，P4 3GHz，512MB 内存环境下，以 Visual C++作为开发工具，利用 Colorado 大学开发的 CUDD 软件包[1]，用 C 语言实现了本文的算法。并以随机 CSP 模型随机生成的大量 CSP 作为测试实例，将本文的算法与传统桶消元算法[2] 和符号直接求解法进行了实验对比。随机 CSP 模型有 4 个主要参数$\langle n,m,c,t\rangle$，其中，n 表示 CSP 中的变量个数；m 表示每个变量值域的大小；c 表示约束图中约束的条数；t 表示约束强度，即每条约束中存在冲突的值对所占的比率。在第一组实验中，选取参数为$\langle n, 2m=5, c=0.35n(n-1)/2, t=0.2\rangle$的测试用例，其中 $n=5,10,15,20$ 和 25，对 n 的每个取值测试 10 个实例并取其平均值。针对 n 的不同取值，3 种算法的求解时间对比如表 1 所列。

表 1　3 种算法在测试用例$\langle n,5,0.35n(n-1)/2,0.2\rangle$上的执行时间对比结果

n	本文的算法(秒)	符号直接求解法(秒)	桶消元算法(秒)
5	0.009	0.0105	0.0001
10	0.0261	0.0698	0.0449
15	0.445	2.1515	4.5949
20	8.5075	60.9605	溢出错
25	29.9137	214.6246	溢出错

(下转第 219 页)

[1] CUDD: CU decision diagram package release 2.3.1. http://vlsi.Colorado. edu /fabio/CUDD/cuddIntro. html

[2] http://carlit. toulouse. inra. fr/cgi-bin/awki. cgi/ ToolBarIntro

geometric embedding of minimum multiway cut[J]. Mathematics of Operations Research,2004,29(3):436-461

[6] Naor J, Zosin L. A 2-approximation algorithm for the directed multiway cut problem[C]// Proceedings of the 38th Annual Symposium on Foundations of Computer Science. New York: IEEE Computer Society,1997:548-553

[7] Ford L R, Fulkerson D R. Flows in networks[M]. New Jersey: Princeton University Press,1962

[8] Papadimitriou C H, Steiglitz K. Combinatorial optimization: algorithms and complexity[M]. New Jersey: Prentice-Hall,1982

[9] 张鹏. 树上推广的 Multicut 问题的近似算法[J]. 计算机研究与发展,2008,45(7):1195-1202

[10] 李曙光,辛晓. 参数为 k 的几乎树中的染色多路割[J]. 计算机科学,2010,37(2):246-249

[11] 孙统风,丁世飞,任子晖. 一种基于 Multiway Cut 的多对象图像分割[J]. 计算机应用研究,2010,27(8):3138-3141

[12] Bodlaender H L, Koster A M C A. Combinatorial optimization on graphs of bounded treewidth[J]. The Computer Journal, 2008,51(3):255-269

[13] Kleinberg J, Tardos E. Algorithm design[M]. Boston: Addison-Wesley,2005

[14] Graham R L. Bounds on multiprocessing timing anomalies[J]. SIAM Journal on Applied Mathematics,1969,17(2):416-429

[15] 李强,闫洗文,梅耀元. 基于 VB 的最小生成树 KRUSKAL 算法的实现[J]. 重庆理工大学学报:自然科学版,2010,24(4):101-104

(上接第 202 页)

由表 1 可见,在保持约束强度不变的情况下,随着问题规模的增大,传统的桶消元算法产生溢出错时,本文的算法仍能在较短的时间内找到问题的所有解。这主要是因为 OBDD 的高紧凑性的特点,加上 OBDD 的操作是以集合方式进行,而不像传统算法那样逐个的进行,为此本文算法在时间和空间上都优于传统的桶消元算法。另外,相比符号直接求解法而言,本文的算法在执行时间上也明显占有优势。这主要是因为本文算法在求解 CSP 的所有解之前,对各 *bucket*[*i*]中约束作了更强的约束,即剔除了一些不可能的变量赋值。为此,在计算中生成的中间积的 OBDD 可能要比符号直接求解法的小。而基于 OBDD 的各种操作,其计算时间主要取决于参与操作的 OBDD 的大小,为此本文的算法要优于符号直接求解算法。

在第二组实验中,选取参数为⟨15,5,37,*t*⟩的测试用例,其中约束强度 $t=0.12+0.04z, z=0,1,\cdots,10$。针对 t 的不同取值,3 种算法的求解时间对比曲线图如图 5 所示,其中图 5 中的纵坐标采用对数坐标。

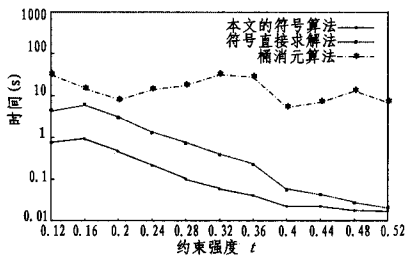

图 5 3 种算法在测试用例⟨15,5,37,t⟩上的实验对比结果

由图 5 可见,在保持问题规模不变的情况下,随着约束强度的增大,两种符号算法都优于传统的桶消元算法。这主要是因为在问题规模不变的情况下,符号算法中的布尔变量数将保持不变,并且随着约束强度的增大,每条约束中不存在冲突的值对逐渐减少,而在表示每条约束的 OBDD 中从根结点到终结点 1 的每一条路径对应的正是不冲突的值对。为此随着 t 的增大,表示每条约束的 OBDD 的大小将逐渐减小,而符号算法的执行时间主要取决于参与操作的 OBDD 的大小,故随着 t 的增大,符号的执行时间逐渐减少。此外,由图 5 可知,本文的符号算法要优于符号直接求解法。其原因同样是因为本文的算法在求解 CSP 的所有解之前,对各 *bucket*[*i*]中约束作了更强的约束,即剔除了一些不可能的变量赋值。

结束语 约束满足问题是实际应用中常见的一类问题。通过对 CSP 问题的符号化描述,将 CSP 问题的求解转化为对 OBDD 图的操作。为了避免在求解过程中生成过大的 OBDD 图,结合了桶消元算法的思想,给出了 CSP 的符号 OBDD 求解算法。以随机 CSP 模型随机生成的大量 CSP 作为测试实例,将本文的算法与桶消元算法和符号直接求解法进行了实验对比,结果表明,本文算法具有较好的时间和空间复杂度。

参 考 文 献

[1] Sadeh N, Sycara K, Xiong Y. Bachtracking techniques for the job shop scheduling constraint satisfaction problem [J]. Artificial Intelligence,1995,76(1/2):455-480

[2] 伍丽华,陈嵩洋,姜云飞. 规划问题编码为约束可满足问题的研究[J]. 计算机科学,2006,33(8):187-292

[3] Pham D N, Thornton J, Sattar A. Modelling and Solving temporal reasoning as propositional satisfiability [J]. Artificial Intelligence,2008,172(15):1725-1782

[4] Miguel I, Shen Q. Solution Techniques for Constraint Satisfaction Problems: Foundations [J]. Artificial Intelligence Review, 2001,15(4):243-267

[5] Dechter R. Bucket Elimination: A Unifying Framework for Reasoning [J]. Artificial Intelligence,1999,113:41-85

[6] Bryant R E. Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams [J]. ACM Computer Survey, 1992, 24(3): 293-318

[7] Burch J, Clarke E, Mcmillan K, et al. Symbolic Model Checking: 10^{20} States and Beyond[J]. Information and Computation,1998, 98(2):142-170

[8] Hachtel G D, Somenzi F. A Symbolic Algorithm for Maximum Flow in 0-1 Networks [J]. Formal Methods in System Design, 1997,10(2/3):207-219

[9] Pan G, Vardi M Y. Symbolic Techniques in Satisfiability Solving [J]. Journal of Automated Reasoning,2005,35(1-3):25-50

[10] Walsh T. SAT v CSP [C] // Proceeding of the 6th international Conference on Principles and Practice of Constraint Programming. 2000,1894:441-456

[11] 徐周波,古天龙. 装配序列规划问题的 CSP 模型及其符号 OBDD 求解技术[J]. 计算机辅助设计与图形学学报,2010,22(5): 803-810