第26卷 第12期
2003年12月

计 算 机 学 报
CHINESE JOURNAL OF COMPUTERS

Vol. 26 No. 12
Dec. 2003

非二元约束满足问题求解

孙吉贵 景沈艳

(吉林大学计算机科学与技术学院 长春 130012)
(吉林大学符号计算与知识工程教育部重点实验室 长春 130012)
(复旦大学智能信息处理开放实验室 上海 200433)

摘 要 在约束满足问题(CSP)的研究中,大部分工作集中在二元约束,但处理实际问题时,常常会遇到非二元约束的情况.该文在概要地讨论了两类求解非二元约束问题方法的基础上,研究了一种将约束传播技术和一般弧相容回溯算法相结合的非二元约束求解方法,并在设计开发的约束求解工具"明月 SOLVER1.0"中实现了该方法,以典型例子给出了实现系统的运行结果.

关键词 约束满足问题;非二元 CSP 求解;约束传播;弧相容;正向检查
中图法分类号 TP301

Solving Non-Binary Constraint Satisfaction Problem

SUN Ji-Gui Jing Shen-Yan

(*College of Computer Science and Technology, Jilin University, Changchun* 130012)
(*Key Laboratory for Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun* 130012)
(*Open Laboratory for Intelligent Information Processing, Fudan University, Shanghai* 200433)

Abstract Most of the research on constraint satisfaction problems (CSPs) concentrates on binary constraints, however, non-binary constraints appear quite frequently when modeling real problems. In this paper, on the basis of summarizing two kinds of approaches on solving non-binary CSPs, we study a non-binary CSP method which combines constraint propagation with GAC backtracking algorithm, and implement it in our Constraint Solving Platform "MingYue (1.0 version)". For some typical examples, we also give experimental results.

Keywords constraint satisfaction problems; non-binary CSPs solving; constraint propagation; arc consistency; forward checking

1 引 言

到目前为止,约束满足中的大部分研究都是假设只考虑二元约束,即约束只包含两个变量.虽然一些学术问题(图着色、斑马等)满足这种条件,但是在许多现实问题中,非二元约束出现得相当频繁.这就使得研究求解非二元约束满足问题越来越引起人们的重视.

约束满足问题由三部分组成:有限变量集 $X=\{x_1,x_2,\cdots,x_n\}$;每个变量 x_i 的有限论域 $D(x_i)$;有限约束集 $C=\{C_1,C_2,\cdots,C_m\}$.每条约束 C_i 是某个变量集 $Vars(C_i)$ 上的约束,这个变量集的大小称为该约束 C_i 的数量.非二元 CSP 是指约束集 C 中存在约束 C_i,其约束数量大于 2 的 CSP.约束 C_i 可看作是 $Vars(C_i)$ 中变量论域笛卡尔乘积的子集(即 C_i 是

收稿日期:2001-07-09;修改稿收到日期:2003-01-23.本课题得到国家自然科学基金(60073039,60273080)、吉林省科技发展计划(20020306)和吉林大学创新基金资助.**孙吉贵**,男,1962年生,博士,教授,博士生导师,主要研究方向为人工智能、约束程序设计、智能决策支持系统.E-mail:jgsun@public.cc.jl.cn.**景沈艳**,女,1976年生,硕士,主要研究方向为约束程序设计、自动推理.

满足该约束的元组的集合).

变量的赋值集合 $A=\{x_1\leftarrow v_1, x_2\leftarrow v_2,\cdots,x_k\leftarrow v_k\}$与约束 C_j是相容的,如果(1)它已经为约束 C_j中所有的变量都赋了值(即 $Vars(C_j)\subseteq\{x_1,x_2,\cdots,x_k\}$);(2)$A$ 限制在 C_j中变量的值的元组是 C_j的成员(即赋值满足约束 C_j). CSP 的解是所有变量的赋值集合$\{x_1\leftarrow v_1,x_2\leftarrow v_2,\cdots,x_n\leftarrow v_n\}$,它与所有约束都相容.

例 1.　考虑 3-SAT 问题,$(X_1\vee X_2\vee X_6)\wedge(\neg X_1\vee X_3\vee X_4)\wedge(\neg X_4\vee\neg X_5\vee X_6)\wedge(X_2\vee X_5\vee\neg X_6)$. 在 3-SAT 问题的非二元 CSP 表示中,对于每个布尔变量 $X_1,X_2,\cdots,X_6$,都有论域$\{0,1\}$,在公式中每个子句是 3 元约束,且应确保每个子句值为 1. 例如,在第一个子句上的约束包含元组 $R\{X_1,X_2,X_6\}=\{(0,0,1),(0,1,0),(0,1,1),(1,0,0),(1,0,1),(1,1,0),(1,1,1)\}$,其中元组(0,0,0)不出现在约束中,因为赋值 $X_1\leftarrow 0,X_2\leftarrow 0,X_6\leftarrow 0$ 不满足该子句.

处理非二元 CSP 时至少有两种选择:一种是先利用对偶图法或隐藏变量法将它转换成二元 CSP,然后使用经典的二元 CSP 求解算法[1]. 这种方法的优点是可以利用大量现成的启发式搜索算法以及当前仅适用于二元 CSP 的算法(如最小正向检查法[2]). 但在实际应用中,它对空间和时间的要求使得它不太实用. 这种二元化方法在转换过程中生成新的变量,这些变量可能有很大的论域,这会导致额外的内存要求. 强制二元化也会产生不自然的陈述,造成约束求解器用户接口额外的困难. 另一种是直接应用非二元 CSP 求解技术,如正向检查(FC)[3,4]或一般弧相容技术(GAC)[5]. 这种方法能够在原始陈述基础上求解非二元约束问题,免除了转换过程及其造成的缺陷,但有效地求解技术和方法还很不成熟,高效的非二元 CSP 求解技术正处于探索阶段.

本文中,我们首先概要地综述了两类解决非二元约束满足问题的方法:一类是转换为二元约束满足问题,再使用经典的二元约束求解算法;另一类是直接把二元约束求解算法扩展到非二元约束的情况. 之后,研究了一种将约束传播技术和一般弧相容回溯算法相结合的非二元约束求解方法,其中约束传播技术通过节点相容、弧相容和边界相容等相容性技术来归约变量论域,大大地修剪了问题的搜索空间;而采用启发式变量排序的一般弧相容回溯方法,提高了搜索效率和整体性能. 我们在自行研发的约束求解工具"明月 SOLVER1.0"中实现并验证了该方法. 最后,对于一些典型的例子,我们给出了实验结果及讨论.

2　非二元 CSP 转换成二元 CSP

二元化的优点在于已有大量现成的启发式搜索算法以及当前仅适用于二元 CSP 的算法(如最小正向检查法). 这里简述两种转化的方法:对偶图法(dual graph)[6]和隐藏变量法(hidden variable)[7].

在对偶图转换中,原始问题中的约束变为新的表示中的变量. 我们把这些表示约束的变量称为 *c*-variable,而原始的变量仍称为变量. 每个 *c*-variable 的论域恰好是满足原始约束的元组的集合. 两个 *c*-variable 间存在二元约束,当且仅当原始约束共享一些变量. 该二元约束要求那些共享变量取相同的值.

例 2.　例 1 中 CSP 的对偶图表示中,有 4 个 *c*-variable $Y_1,Y_2,\cdots,Y_4$,每个代表原始问题中的一条 3 元约束(即子句),见图 1. 例如,*c*-variable Y_1对应非二元约束 $R\{X_1,X_2,X_6\}$,Y_1的论域是约束元组集合$\{(0,0,1),\cdots,(1,1,1)\}$. 二元约束强制出现在多于一个 *c*-variable 中的变量有相同的值.

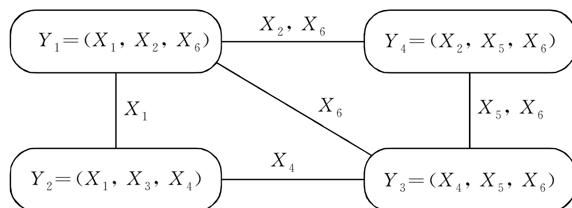

图 1　由对偶图法产生的二元 CSP

在隐藏变量表示中,变量集包括原始问题中所有的变量(它们的域没有改变)加上一个新的隐藏变量 *h*-variable 集. 对于原始问题中的每条约束 C_i,增加一个 *h*-variable H_i. H_i的论域由 C_i中每个元组的唯一标识符组成. 新的表示只包含二元约束,构造如下:对于每个 *h*-variable H_i,在 H_i和 $Vars(C_i)$中每个变量间施加一条二元约束. H_i和 x_k是这样约束的,H_i中每个值对应于 $Vars(C_i)$中变量的值元组,则定义了 x_k的唯一值. 因此,H_i和 x_k间的二元约束由 x_k的唯一值与 H_i中每个值组成(注意约束在另一方向不起作用,x_k的一个值可能与 H_i的许多值相容).

例 3.　在例 1 的 CSP 隐藏变量表示中有 10 个变量:6 个初始变量 $X_1,X_2,\cdots,X_6$和 4 个隐藏变量,对应原始问题中的每条约束(见图 2). 例如,约束 $R\{X_1,X_2,X_6\}$由一个相应的 *h*-variable H_1,它的论域为集合$\{1,2,\cdots,7\}$(约束中 7 个元组的每个的标

识符).可定义 H_1 的值和 $R\{X_1, X_2, X_6\}$ 中的元组之间对应关系如下:1→(0,0,1),2→(0,1,0),3→(0,1,1),4→(1,0,0),5→(1,0,1),6→(1,1,0),7→(1,1,1).然后在变量对 $\{X_1, H_1\}$,$\{X_2, H_1\}$,和 $\{X_6, H_1\}$ 之间施加约束,给出二元约束,

$$C\{X_1,H_1\}=\{(0,1),(0,2),(0,3),(1,4),(1,5),(1,6),(1,7)\},$$

$$C\{X_2,H_1\}=\{(0,1),(1,2),(1,3),(0,4),(0,5),(1,6),(1,7)\},$$

$$C\{X_6,H_1\}=\{(1,1),(0,2),(1,3),(0,4),(1,5),(0,6),(1,7)\}.$$

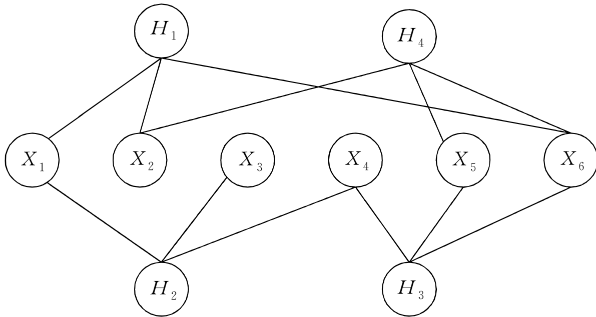

图 2 由隐藏变量法产生的二元 CSP

例如对于 $C\{X_1, H_1\}$,H_1 的值 3 对应元组(0,1,1),其中 $X_1=0$.因此 $H_1=3$ 只与 $X_1=0$ 相容.

FC+[1]是运行在隐藏变量表示上的回溯算法.它与 FC 非常类似,只是在正向检查修剪了任意一个 h-variable 后,额外再修剪由那个 h-variable 约束的所有未实例化变量的论域,以便删除那些已经失去支持的值.若检测到未来变量的论域为空,则进行回溯.

3 非二元 CSP 的求解方法

虽然任何非二元的离散 CSP 都能转换成等价的二元 CSP[8],理论上解决了非二元约束的求解,但在实际中由于其空间和时间的要求,使得它不太实用.在转换过程中生成若干新的变量,这些变量可能有很大的论域,导致算法有额外的内存要求[1].强制二元化也会生成不自然的陈述,造成约束求解器接口和用户额外的困难.

另一个方法就是直接建立非二元 CSP 求解技术,它能够以初始陈述来求解非二元问题.这个方法除去了转换过程及其缺点,但是又出现了新的问题,如二元算法怎样被扩展?对于一些算法,例如回溯或最小弧相容(MAC),这种扩展没有概念上的困难:它们的二元定义只允许一种可能的非二元推广.对于其它的算法,如正向检查(FC),可能有几种推广[4].所有推广和新建立的非二元 CSP 技术的关键都是系统的效率问题.

3.1 正向检查(Forward Checking-FC)

文献[3]中定义的 FC,用于二元约束网络(以下称为 bFC).它是在赋值了当前变量后,在 $C^b_{c,f}$ 的每条约束上应用弧相容一次.如果成功(即没有检测到空域),用新变量继续,否则回溯.其中 $C^b_{c,f}$ 表示包含当前变量和未来变量的二元约束.

FC 策略怎样才能扩展到非二元约束上呢?在包含过去变量和未来变量的约束集上获得弧相容(与 bFC 的同级相容性)似乎是合理的.在二元情况中,这样的集合只有一种选择:约束连接一个过去变量(当前变量)和一个未来变量.在非二元的情况下,则有不同的选择:

(1)约束包含至少一个过去变量和至少一个未来变量.

(2)约束或约束投影包含至少一个过去变量和恰好一个未来变量.

考虑选项(1),定义 $C^n_{p,f}$ 为包含至少一个过去变量和至少一个未来变量的约束集合,$C^n_{c,f}$ 为包含当前变量和至少一个未来变量的约束集合.与二元情况不同的是,在这些集合中,我们必须处理部分实例化的约束,它包含多于一个未实例化的变量.依赖于考虑的约束集($C^n_{p,f}$ 或 $C^n_{c,f}$)以及弧相容是在整个集合上获得,还是依次应用在每条约束上,可以用不同的方法.下面是这些方法的总结[4]:

nFC5:赋值了当前变量之后,使集合 $C^n_{p,f}$ 弧相容.若成功(即没有检测到空域),用新变量继续,否则回溯.

nFC4:赋值了当前变量之后,根据约束次序在 $C^n_{p,f}$ 的每条约束上应用一次弧相容.若成功,用新变量继续,否则回溯.

nFC3:赋值了当前变量之后,使集合 $C^n_{c,f}$ 弧相容.若成功,用新变量继续,否则回溯.

nFC2:赋值了当前变量之后,根据约束次序在 $C^n_{c,f}$ 的每条约束上应用一次弧相容.若成功,用新变量继续,否则回溯.

考虑选项(2),定义 $C^n_{c,1}$ 为包含当前变量和恰好一个未来变量的约束集合,$CP^n_{c,1}$ 为包含当前变量和恰好一个未来变量的约束投影集合.对于选项(2)中的每一个,只存在一个选择.

nFC1[9]:赋值了当前变量之后,根据约束次序

在 $C^n_{c,1}\cup CP^n_{c,1}$ 的每条约束上应用一次弧相容. 若成功,用新变量继续,否则回溯.

nFC0[10]:赋值了当前变量之后,根据约束次序在 $C^n_{c,1}$ 的每条约束上应用一次弧相容. 若成功,用新变量继续,否则回溯.

令 $C=\{C_1,C_2,\cdots,C_k\}$ 是约束集. 我们将用 $AC(C)$ 表示在集合 C 上实施弧相容的过程. 给定约束 $C_1,C_2,\cdots,C_k$ 的任意顺序,用 $AC\{C_1\},AC\{C_2\},\cdots,AC\{C_k\}$ 表示 AC 按照约束顺序在每条约束上执行一次.

例 4. 变量集 $X=\{x,y,z,u,v,w\}$,每个变量论域都是$\{a,b,c\}$,约束如表 1 所示.

表 1 约束表示

C_1	C_2	C_3
x y z	u v w	x y w
a a a	a a a	a a a
a b c	a b b	a b c
a c b	c c c	

为了阐述 6 种算法之间的不同,表 2 展示了一个简单的例子. 它由 6 个变量组成,$\{x,y,z,u,v,w\}$,分享相同的域$\{a,b,c\}$,服从 3 个三元约束:$C_1(x,y,z)$,$C_2(u,v,w)$,$C_3(x,y,w)$. 在赋值了(x,a)之后,没有约束含有两个已实例化的变量. 因此,nFC0 不过滤. nFC1 在子集$\{x,y\}$,$\{x,z\}$,$\{x,w\}$上 C_1 和 C_3 的约束投影上应用弧相容,从 $D(y)$ 中删除 c,从 $D(w)$ 中删除 b. nFC2 先在 C_1,后在 C_3 上应用弧相容,修剪与 nFC1 同样的值. 值得注意的是,如果我们以不同的次序考虑这些约束,过滤将会不同. nFC3 在子集$\{C_1,C_3\}$上应用弧相容,导致 nFC2 的过滤,并从 $D(z)$ 中删除 b. 假定 x 是第一个实例化的变量,nFC4 与 nFC2 在相同的约束上应用弧相容,同理,nFC5 与 nFC3 执行同样的过滤.

在赋值了(u,a)之后,没有约束有两个实例化的变量. 所以,nFC0 不过滤. nFC1 在子集$\{u,v\}$,$\{u,w\}$上 C_2 的约束投影上应用弧相容,从 $D(v)$ 中删除 c,从 $D(w)$ 中删除 c. nFC2 在 C_2 上应用弧相容,从 $D(v)$ 中删除 b 和 c,从 $D(w)$ 中删除 c. nFC3 在子集$\{C_2\}$上应用弧相容,导致与 nFC2 相同的过滤(由于先前的赋值,使得 $D(z)$ 域不同). nFC4 在约束 C_1,C_2,C_3 上应用弧相容,从 $D(y)$ 和 $D(z)$ 中删除 b,从 $D(v)$ 中删除 b 和 c,从 $D(w)$ 中删除 c. nFC5 在整个约束集上应用弧相容,从 $D(y)$ 中删除 b,从 $D(z)$ 中删除 c,从 $D(v)$ 中删除 b 和 c,从 $D(w)$ 中删除 c.

3.2 一般弧相容

(Generalized Arc Consistency,GAC)

Mackworth 于 1977 年提出算法 CN[11,12],它是 AC-3 推广到非二元约束情况. 算法有一个糟糕的最坏情况时间复杂度($O(er^2d^{r+1})$,其中 e 是网络中的约束数,r 是约束所含的最大变量数,d 是最大域的大小). GAC4 由 Mohr 和 Massini 在 1988 年提出[5],思想类似于 AC-4:对每个变量域的每个值计算支持数,删除那些支持数等于 0 的值. 虽然它去掉了 CN 最坏情况下的时间复杂度(GAC4 是 $O(ed^r)$),但是与 AC-4 有同样的缺点:空间复杂度(因为支持值的列表)和平均时间复杂度难以控制. CN 和 GAC4 各自的缺点比它们的二元化版本更重要,使得 CN 只应用于三元约束和非常小的值域,而 GAC4 应用在非常紧密的约束,其中允许的值元组数很小.

表 2 一个简单的问题,在赋值了(x,a)和(u,a)后,6 种算法导致的过滤

赋值	算法	动 作	$D(x)$	$D(y)$	$D(z)$	$D(u)$	$D(v)$	$D(w)$
(x,a)	nFC0	无	a	a,b,c	a,b,c	a,b,c	a,b,c	a,b,c
	nFC1	$AC(\{C_1[x,y]\}),AC(\{C_1[x,z]\}),AC(\{C_3[x,y]\}),AC(\{C_3[x,w]\})$	a	a,b	a,b,c	a,b,c	a,b,c	a,c
	nFC2	$AC(\{C_1\}),AC(\{C_3\})$	a	a,b	a,b,c	a,b,c	a,b,c	a,c
	nFC3	$AC(\{C_1,C_3\})$	a	a,b	a,c	a,b,c	a,b,c	a,c
	nFC4	$AC(\{C_1\}),AC(\{C_3\})$	a	a,b	a,b,c	a,b,c	a,b,c	a,c
	nFC5	$AC(\{C_1,C_3\})$	a	a,b	a,c	a,b,c	a,b,c	a,c
(u,a)	nFC0	无	a	a,b,c	a,b,c	a	a,b,c	a,b,c
	nFC1	$AC(\{C_2[u,v]\}),AC(\{C_2[u,w]\})$	a	a,b	a,b,c	a	a,b	a
	nFC2	$AC(\{C_2\})$	a	a,b	a,b,c	a	a	a
	nFC3	$AC(\{C_2\})$	a	a,b	a,c	a	a	a
	nFC4	$AC(\{C_1\}),AC(\{C_2\}),AC(\{C_3\})$	a	a	a,c	a	a	a
	nFC5	$AC(\{C_1,C_2C_3\})$	a	a	a	a	a	a

4 约束求解平台"明月 SOLVER1.0"中的非二元约束求解

约束求解平台"明月 SOLVER1.0"[13]是基于约束程序设计的思想,用来求解约束满足问题;"明月 SCHEDULER"在 SOLVER 基础上,用于求解组合优化和调度问题. 该平台通过把约束集成到面向对象的程序设计语言(C++)中,将约束、变量、论域和求解器都作为实体,使用预定义的约束集合和

通用的求解器来有效地解决组合优化问题.系统采用问题建模与问题求解分离的设计原则,强有力的建模和通用的求解器是系统的优点.我们令布尔变量对应形如 $Exp1$ op $Exp2$ 的数值约束,这里 $Exp1$ 和 $Exp2$ 是实型(或整型)论域下的表达式(例如 $5x-3y$),op 是关系符号(例如 $=,\geqslant,\neq$).约束之间通过与、或、非逻辑算子结合成复杂约束.当用户声明约束后,系统把它自动转换为内部表示,应用约束传播技术归约变量论域,应用一般弧相容算法求解 CSP.

4.1 约束传播技术

系统首先采用约束传播技术,通过节点相容、弧相容和边界相容等相容性技术来归约变量论域,大大地修剪了问题的搜索空间.针对各个具体的约束类(如+,-,*,/,==,!=,<,>等),系统设计特殊的传播规则,这样当用户"告诉"系统约束条件后,系统会自动地找到相应的约束类,进行约束传播.

算法 1. 约束传播算法.

触发传播事件:变量实例化、边界归约(最小/最大值改变)、论域归约(论域中有值删除).

输入:约束网络 constraintNetwork(包括变量表、约束表等)

输出:如果所有变量的论域都与约束相容,返回 1,否则任何变量论域变为空时,返回 0,证明不相容.

使用的数据结构及参数:队列 thisQueue

算法:如图 3 所示.

```
int constraint_propagate()
  构造约束传播队列 thisQueue(用于存放改变的变量);
  while(变量表中有变量改变)
      恢复变量状态(未改变);
      变量入队列;
  while (队列 thisQueue 非空)
      var=出列变量;
      检查变量 var 所在的每条约束 cst;
      调用约束 cst 的传播函数;
  if (约束 cst 的传播函数返回 0)
     return 0;
  while (约束 cst(传播后)所含的变量 var 改变)
      恢复变量 var 状态(未改变);
      thisQueue->enqueue(var);
  return 1;
```

图 3 约束传播算法

4.2 正向检查和一般弧相容相结合的回溯方法

在搜索过程中优先选择论域最小、参与约束最多的变量,对于当前变量论域中的每个值,检查约束相容性.如果约束只包含一个未实例化的变量,则对该约束应用正向检查(FC)技术,如果约束包含两个以上未实例化的变量,则应用一般弧相容(GAC)技术.当检查完当前变量所在的全部约束而相容,则重复上面操作,选择下一个变量,搜索树深度加 1.否则(检测到空域),就恢复那些由于当前变量的实例化而删除的其它变量的值,顺序回溯.

int FC_GAC(int *curDepth*)

(1) 如果搜索树深度大于变量数,则返回 1.

(2) 按最小论域、最多约束优先的变量启发式选择变量.

(3) 对于当前变量论域中的每个值:

(i) 如果该值可用,则赋给当前变量.

(ii) 如果约束相容(用 Consistent 函数判断,返回 1 表示相容)则递归调用 GAC 函数(深度加 1),否则恢复那些由于当前变量的实例化而删除的其它变量的值;

(4) 如果已经试遍当前变量中的所有值而约束不相容,则返回 0.

int Consistent(int *curDepth*,int *curVar*)

(1) 对于当前变量所在的每条约束,如果约束可以正向检查(只有一个变量未实例化),则正向检查(check_forward 函数),如果正向检查失败(即检测到空域),则从堆栈弹出一个变量,返回 0.

(2) 当前变量入栈.

(3) 当堆栈非空时:

(i) 从堆栈中弹出一变量 v.

(ii) 对于变量 v 所在的每条约束,若约束可弧相容检查(有两个以上的变量未实例化),则对每个未实例化变量进行修正(revise 函数).若修正失败,弹出一个变量,返回 0.

int forward_ checking(int *cstId*,int *curDepth*,int *curVar*)

(1) 在可作正向检查的约束中,找到未实例化的变量(只有一个),称为未来变量.

(2) 对于未来变量中的每个值

(i) 若该值可用,则赋给未来变量.

(ii)检查约束是否相容,若违反了约束,置改变标志,该值置为不可用,未来变量论域大小减 1.

(3) 若改变标志为真,则未来变量入栈.返回未来变量论域大小.

int arc_ consistency_ checking (int *cstId*, int *var*, int *curDepth*,int *curVar*)

(1) 变量 v 状态置为已实例化.

(2) 对于变量 v 中每个值,

(i) 若该值可用,则赋给变量 v.

(ii) 检查余下变量中是否存在使约束满足的值(exist 函数),若不存在,置改变标志,该值置为不可用,变量 v 论域大小减 1.

(3) 变量 v 状态置为未实例化.

(4) 若改变标志为真,则变量 v 入栈.返回变量 v 论域大小.

int satisfaction(int *cstId*)

(1) 对于约束中的每个变量,如果该变量已实例化,则直接把值赋给元组;否则,从变量论域中找到第一个可用的值赋给元组.若没有这样的值,返回 0.

(2) 对于约束中的所有变量,用元组给它们赋值.

(3) 检查约束是否相容,若相容,则返回 1;否则得到下一个元组,重复上述过程.如果已经穷举了所有的元组,则返回 0.

5 实验结果

这里以典型的非二元约束 N-皇后问题测试了“明月 SOLVER1.0”,下面是调用 SOLVER1.0 的 C++程序.程序本身说明系统如何对 N-皇后问题进行建模和求解.

```
void main()
{
 SchInit();
 Int i,nqueen=8;
 SchIntVar** x =new SchIntVar*[nqueen];
 SchIntVar** x1 =new SchIntVar*[nqueen];
 SchIntVar** x2 =new SchIntVar*[nqueen];
 for( i=0;i<nqueen;i++)
 {
  x[i]=new SchIntVar(0,nqueen-1);
  x1[i]=&(*x[i]+i);
  x1[i]=&(*x[i]-i);
 }
 SchTell(SchAllDiff(nqueen,x));
 SchTell(SchAllDiff(nqueen,x1));
 SchTell(SchAllDiff(nqueen,x2));
 SchSolve (generate(nqueen,x));
 for (i=0;i<nqueen;i++)
   cout << *x[i] <<" ";
 SchEnd();
}
```

函数 *SchInit*()用来初始化系统的状态,*SchEnd*()用来释放内存.创建大小为 *nqueen* 的约束整型变量数组 x,$x1$,$x2$.数组 x 中第 i 个变量表示在棋盘中第 i 行里的皇后所在的列.初始时它们的论域均为 0..*nqueen*-1,表示每一行中皇后可能放置在每一列.数组 x 用来保证任意两个皇后不在同一列中,数组 $x1$ 和 $x2$ 用来保证任意两个皇后不在相同的对角线上.调用 *SchAllDiff* 创建约束,指定数组中的元素各不相同.*SchTell* 函数用来声明约束,并把它添加到约束网络中,执行约束传播算法.使用 *generate* 函数创建目标,*SchSolve* 函数在此基础上搜索问题的解.

表 4　N-皇后问题的求解时间

皇后的数目	运行时间(s)
8	0.01
16	0.02
18	0.02
19	0.04
20	0.05
30	0.09
50	2.85
60	3.16
80	0.65
90	6.41
100	1.24

实验结果:系统对皇后数从 8～100 进行了测试,最长运行时间为 7.28s.表 4 给出了系统求解 8～100 的部分 N-皇后问题的运行时间.在没有加入任何特殊的领域知识只使用通用经典方法求解的条件下,系统的效率还是相当高的.图 4 具体给出了 $N=30$ 时的一个解.

当然,在仔细地分析了 N-皇后问题后,人们可以得到一个解决它非常有效的方法[14,15],但是这样的算法只能用来解决 N-皇后问题,而不能解决其它的 CSP 问题.我们的系统提供的是一个通用的求解器,在求解 N-皇后问题时效率可能不如专用的算法,但是它可以有效地求解大多数的 CSP 问题.

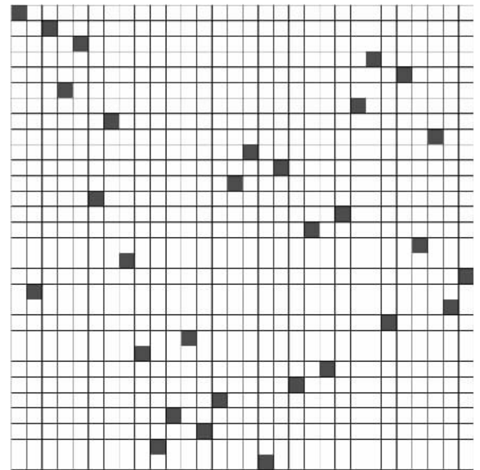

图 4　$N=30$

本文的工作主要是针对非二元 CSP 问题,我们的实现系统自然能够处理二元 CSP 问题,如中国地图 34 个省的着色问题,问题建模为约束 76 条,运行时间为 0.19s,约束检查数为 283.对于约束优化调度问题[16,17],SOLVER 求解器并不见长,而在 SOLVER 基础上开发设计的通用调度器 SCHEDULER 正是为了求解此类问题,目前该系统正在研发中.

参 考 文 献

1 Bacchus F, van B P. On the conversion between non-binary and binary constraint satisfaction problems. In: Proceedings of AAAI′98, Madison WI, 1998. 311～318

2 Dent M J, Mercer R E. Minimal forward checking. In: Proceedings of the 6th IEEE International Conference on Tools with Artificial Intelligence, New Orleans, LA, 1994. 432～438

3 Haralick R M, Elliot G L. Increasing tree seach efficiency for constraint satisfaction problems. Artificial Intelligence, 1980, 14(3): 263～313

4 Bessiere C, Meseguer P, Freuder E C, Larrosa J. On forward checking for non-binary constraint satisfaction. Artificial Intelligence, 2002, 141(1/2): 205～224

5 Mohr R, Masini G. Good old discrete relaxation. In: Proceedings of ECAI′88, Munchen, FRG, 1988. 651～656

6 Dechter R, Pearl J. Tree clustering for constraint networks. Artificial Intelligence, 1989, 38(3): 353～366

7 Dechter R. On the expressiveness of networks with hidden variables. In: Proceedings of the 8th National Conference on Artificial Intelligence, Boston, Mass, 1990. 556～562

8 Rossi F, Petrie C, Dhar V. On the equivalence of constraint satisfaction problems. In: Proceedings of ECAI′90, Stockholm, Sweden, 1990. 550～556

9 Larrosa J, Meseguer P. Adding constraint projections in n-ary csp. In: Proceedings of the ECAI′98 workshop on non-binary constraints, Brighton, UK, 1998. 41～48

10 van H P. Constraint satisfaction in logic programming. Cambridge, MA: MIT Press, 1989

11 Bessière C, Régin J C. Arc consistency for general constraint networks: Pre-liminary results. In: Proceedings of IJCAI′97, Nagoya, Japan, 1997. 398～404

12 Mackworth A K. On reading sketch maps. In: Proceedings of IJCAI′77, Cambridge MA, 1977. 598～606

13 Sun Ji-Gui, Jing Shen-Yan, Jiang Ying-Xin. Detailed design for platform of constraint solving “Mingyue”(Version 1.0). Technical Report, Decision Support System, Research & Development Centre, Jilin University, 2000, (2): 1～87(in Chinese)
(孙吉贵,景沈艳,姜英新. 约束求解平台“明月 1.0 版”详细设计. 吉林大学决策支持系统研发中心技术报告, 2000, (2): 1～87)

14 Marriot K, Stuckey P J. Programming with Constraints. Cambridge, MA: MIT Press, 1998

15 Bernhardsson B. Explicit solutions to the N-queens problem for all N. ACM SIGART Bulletin, 1991, 2(2): 7

16 Jing Shen-Yan, Sun Ji-Gui, Zhang Yong-Gang. Solving Scheduling problems with genetic algorithm. Journal of Jilin University(Science Edition), 2002, 40(3): 263～267(in Chinese)
(景沈艳,孙吉贵,张永刚. 用遗传算法求解调度问题. 吉林大学学报(理学版), 2002, 40(3): 263～267)

17 Baptiste P, Le Pape C, Nuijten W. Constraint-based scheduling: Applying constraint programming to scheduling problems. Boston: Kluwer Academic Publishers, 2001

SUN Ji-Gui, born in 1962, Ph.D., professor, Ph.D. supervisor. His research interests include artificial Intelligence, constraint programming, intelligent decision support systems.

JING Shen-Yan, born in 1976, M.S. Her research interests include constraint programming, automated reasoning.