求解过程中约束一致性维护的多层次策略研究*

韩 靖 陈恩红 蔡庆生

(中国科学技术大学计算机科学系 合肥 230027)

E-mail: hanjing@dawn1.cs.ustc.edu.cn

摘要 约束满足问题广泛出现于人工智能领域.在问题求解过程中保持局部约束一致性以缩小问题搜索空间是十分必要的.过去研究者对约束一致性算法的研究仅着眼于改进单个约束关系的维护.该文立足于更高点,提出从求解层次、维护层次和约束层次优化约束一致性维护的原则及其相应策略,算法MAC-H和AC-I$^+$进一步减少了约束一致性维护的总代价,并克服了原有算法空间复杂度大的缺点.文中以两个典型的约束满足问题:N-皇后问题和斑马难题为分析和测试的例子,证实了这些原则和策略的有效性.

关键词 约束一致性维护,约束检测,多层次原则.

中图法分类号 TP18

约束满足问题CSPs(constraints satisfaction problems)广泛出现于人工智能领域中.它要求为问题中每个变量寻找一个满足相互间的约束关系的赋值.为了简化问题模型,本文以二元约束满足问题(即约束关系为二元关系)为讨论对象.

CSPs问题可以定义为$P=(X,D,R)$.其中n个变量的集合$X=\{X_1,\ldots,X_n\}$,各变量的值域集合$D=\{D_1,\ldots,D_n\}$,所有二元约束关系的集合$R=\{R_{i_1j_1},\ldots,R_{i_mj_m}\}$,$R_{i_1j_1}$是$(X_{i_1},X_{j_1})$上的约束关系,取$D_{i_1}\times D_{j_1}$的笛卡尔乘积的子集.一个解答是指对$\{X_1,\ldots,X_n\}$的一组赋值$\{a_1,\ldots,a_n\}$,$a_i\in D_i$,且满足所有的约束关系$R$.

两个受约束的变量V_i和V_j之间存在着二元约束关系R_{ij},如果对$\forall a\in D_i$(D_i是V_i的当前值域),$\exists b\in D_j$(D_j是V_j的当前值域),有$(a,b)\in R_{ij}$(简写为$R_{ij}(a,b)$成立),则称(V_i,V_j)满足弧约束一致性,其中b称为a在约束关系R_{ij}中的支持值.这种判定一对赋值(a,b)是否属于R_{ij}的过程,称为"约束检测"(Constraint Check).如果$a\in D_i$,对某个与V_i有关的约束R_{ij},a在D_j中找不到支持值,则称a为D_i中的"死值"(Dead Value).

在传统回溯算法的搜索中,由于点不一致性或弧不一致性往往会导致无意义的"反覆"(Thrashing).[1]为避免或减少反覆,在搜索过程中就要维护变量之间的局部约束一致性.Bessiere和Regin[2]指出,完全的约束维护MAC在实际问题和高难度随机问题中都表现出优越性,故本文主要讨论运用于MAC的策略.为了减少约束一致性维护的代价,人们在过去10几年中不断改进约束一致性算法,从AC-1到AC-6[3],AC-7[4]和AC-Inference[4],用尽可能少的约束检测寻找支持值.但是,这种改进仅仅是在对一个约束关系的检测次数上.直到最近,Wallace等人[5]在启发式约束维护次序(Constraint Ordering Heuristics)上才取得了一些成果,但仍有一定的局限性.

本文从总体出发,总结出3个层次的原则策略,包括(1)求解层次,是否在求解过程中的每一步都做约束一致性维护;(2)维护层次,如果必须做,则是否每一个约束都要进行维护,分别在何种情况下进行维护,以及以何种次序进行约束关系的维护;(3)约束层次,对于每一约束的维护,如何用最少的约束检测次数找到支持值.这3个层次和相应策略如图1所示.

1 约束一致性维护中的层次原则

1.1 求解层次——有效性原则

我们首先从约束一致性的有效性入手,讨论如何减少求解过程中的维护次数,以降低维护总代价.就目的而言,只

---

* 本文研究得到国家自然科学基金和中国科学技术大学青年基金资助.作者韩靖,女,1974年生,硕博连读生,主要研究领域为约束满足问题,KDD.陈恩红,1968年生,博士,讲师,主要研究领域为约束满足问题,机器学习,知识发现.蔡庆生,1938年生,教授,博士导师,主要研究领域为人工智能,机器学习,知识发现.

本文通讯联系人:韩靖,合肥 230027,中国科学技术大学计算机科学系

本文1997-04-16收到原稿,1997-07-18收到修改稿

© 中国科学院软件研究所 http://www.jos.org.cn

有达到以下两种效果之一的约束一致性维护才是有效的.① 证实当前的赋值是失败的,即维护过程中发现某个未赋值变量的值域为空;② 删除未赋值变量值域中的死值.有效性原则只进行有效的约束一致性维护.

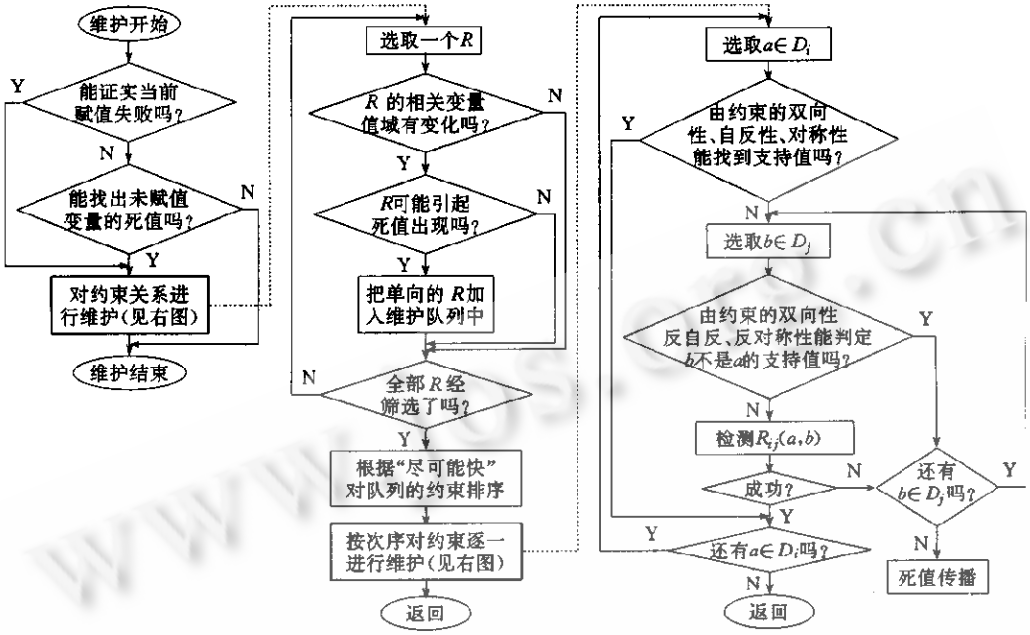


图1　约束一致性维护的层次策略

图 2 所示 4-皇后问题中,X_1 赋值为 1 后进行约束一致性维护.灰方格是与 $X_1=1$ 不一致的死值,$X_2=3$(图中带"×"的格)在 D_3 中找不到支持值,删除;$X_3=2$ 在 D_4 中找不到支持值,删除,从而 $X_2=4$ 失去唯一的支持值,删除.此时,D_2 为空,表示 $X_1=1$,失败,再取 $X_1=2$,重复上述过程.由此可见,该次约束一致性维护是有效的,它达到上述第 1 种效果.

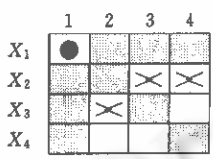


图 2

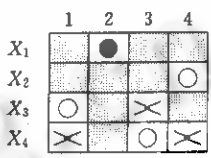


图 3

当 X_1 赋值 2 后,进行约束一致性维护,如图 3 所示.维护结果是 $D_2=\{4\}$,$D_3=\{1\}$,$D_4=\{3\}$,对未赋值变量的值域进行了大量的删减,搜索空间变小.又如斑马难题,如果在求解之前使用约束一致性维护进行初始化,实验表明,将删除 39 个原始值域中的值.这两者都达到上述第 2 种效果.

但是,有些约束一致性维护是无效的.如果问题初始时,变量之间本身就满足约束一致性,则不必在问题求解之前的初始化中进行约束一致性维护.如 N-皇后问题初始化时就没有进行约束一致性维护的必要.另外,实验表明,不少问题在搜索到某一深度后,剩下的未赋值变量的值域大小为 1,此时不应做约束一致性维护,而是直接根据约束关系检测是否解答.如 4-皇后问题 $X_1=2$ 后(见图 3),$|D_2|=|D_3|=|D_4|=1$,经检测,这是解答.再如斑马问题,当 $X_{\text{English}}=3$ 和 $X_{\text{Spaniard}}=4$ 后,约束一致性维护使剩下未赋值变量的值域大小都为 1,经直接检测是解答.

从上面的分析可知,针对具体问题,我们应该首先考虑,在各种情况下(如初始化,每一次赋值后),约束一致性维护是否有效.为减少求解的总代价,应尽可能不进行无效的约束一致性维护.

1.2　维护层次——筛选和排序原则

如果已判定某次约束一致性维护是有效的,而必须进行维护时,应考虑如何减少一次约束一致性维护的代价.为此,本文提出了用于维护层次的两条原则:筛选原则和排序原则以及基于这两条原则的若干策略.

筛选原则　维护尽可能少的约束关系以达到约束一致性.

(1) 相关变量值域没有变化时不进行该约束的维护(初始化除外).变量赋值后,只改变了赋值变量的值域,因此,

© 中国科学院软件研究所 http://www.jos.org.cn

维护初期只对与当前赋值变量有关的约束关系进行维护，而在以后的死值传播中，再对与死值有关的约束进行维护.

(II) 对与当前赋值变量有关的约束进行单向维护. 例如，V_j 赋值为 b 后，对关系 R_{ij} 进行维护，只需对 D_i 中每个值在 $D_j=\{b\}$ 中找支持值，根据约束的双向性，不需要对 $D_j=\{b\}$ 中 b 在 D_i 中找支持值，并且这种维护是可以简化的，因为赋值变量的值域 D_j 中只有 1 个元素.

(III) 肯定不能引起死值出现的约束关系不进行维护. 如不等值约束关系 R_{ij}. 若 V_i 和 V_j 的值域大小都大于 1，则一变量值域中的每个值必能在对方的值域中找到支持值，故不必进行维护；只有当一方的值域大小为 1，才会引起另一方值域中死值的出现，这时，必须进行维护. 再如 N-皇后问题中，对于约束关系 R_{ij}，如果 $|D_j|>3$，则 D_i 中任何一个值 a 都可以在 D_j 中找到支持值，因为在第 j 行与 a 相攻击的位置个数最多为 3 个. 故这样的约束关系也不需要维护.

排序原则　改变约束的维护次序以减少约束检测的次数.

(IV) "失败先传播". 每当有死值出现时，马上对其进行传播，即为与该死值的变量相关的约束关系进行维护(重新寻找支持值). 对这些约束关系的维护优先于其他要进行维护的约束关系. 这一策略适用于约束图为稀疏图的问题，如斑马难题，不适于约束图为完全图的 N-皇后问题.

(V) "尽可能早". Wallace 和 Freuder 首先提出一种直觉规则——"尽可能早"(as Soon as Possible)[5]，在维护过程中，先选择最紧密的约束或变量值域最小的相关约束进行维护，使死值尽早被找出并删除. 最近，Gent 等人[6]进一步证实了这一规则的明显效果，并且利用约束度(Constrainedness)作为启发信息来排列维护约束的次序，在相变(Phase Transition)中优于其他启发信息.

由此可见，即使我们确定了某次约束一致性维护是有效的，还需对要进行维护的约束关系进行筛选和排序，以减少维护代价.

1.3 约束层次——推理原则

当我们确定要对某个约束进行约束一致性维护后，仍然希望通过尽量少的约束检测次数去实现对这个约束的维护. 在过去的 10 多年里，人们围绕着这一目的，不断地对约束一致性算法进行改进，从 AC-1，AC-2，AC-3 到 AC-6，AC-7和 AC-Inference，用尽可能减少约束检测的次数寻找支持值. 其中 AC-Inference 运用约束关系元知识进行推理，比其他算法的约束检测次数少，特别适用于约束检测代价比较大的问题. 但是它也有严重的缺陷，空间复杂度为 $O(ed^2)$，而 AC-6 和 AC-7 的空间复杂度仅为 $O(ed)$.

为了减少 AC-Inference 的空间开销和综合各算法的优点，本文先把二元约束本身的元知识推理规则归纳成以下 3 条：

①如果 $R_{ij}(a,b)$ 或 $R_{ji}(b,a)$ 已经被检测过，则不检测 $R_{ij}(a,b)$.

②如果 $\exists\, b'\in D_j$，且 $R_{ij}(a,b')$ 或 $R_{ji}(b',a)$ 已经被检测为成立，则不检测 $R_{ij}(a,b)$.

③如果能利用约束关系的特殊性质进行推理，为 R_{ij} 中 a 找到一个支持值或者证明 $R_{ij}(a,b)$ 不成立，则不检测 $R_{ij}(a,b)$. 二元关系中有几种特殊性质：自反、反自反、对称、反对称、等价. 由这些性质可以得到

I. R_{ij} 是自反的，且 $a\in D_j\Rightarrow R_{ij}(a,a)$ 成立.

II. R_{ij} 是反自反的，且 $a\in D_j\Rightarrow R_{ij}(a,a)$ 不成立.

III. R_{ij} 是对称的，且 $b\in D_j\wedge R_{ji}(a,b)$ 成立(不成立)$\Rightarrow R_{ij}(a,b)$ 成立(不成立)；

或 R_{ij} 是对称的，且 $b\in D_i\wedge a\in D_j\wedge R_{ij}(b,a)$ 成立(不成立)$\Rightarrow R_{ij}(a,b)$ 成立(不成立).

IV. R_{ij} 是反对称的，且 $b\in D_j\wedge R_{ji}(a,b)$ 成立$\Rightarrow R_{ij}(a,b)$ 不成立；

或 R_{ij} 是反对称的，且 $b\in D_i\wedge a\in D_j\wedge R_{ij}(b,a)$ 成立$\Rightarrow R_{ij}(a,b)$ 不成立.

V. R_{ij} 是等价关系的，它同时具有自反和对称性，兼有上述 I，III 的性质.

例如，在斑马问题中，约束关系有 4 种类型. 其中 In The Same House 具有自反性、对称性；In The Next House 具有反自反性、对称性；On The Right/Left 具有反自反性、反对称性；Not The Same House 具有反自反性、对称性. 运用这些性质，可以减少约束检测的次数.

由上述 3 条规则，可以总结出该层次的**推理原则**：如果能利用约束元知识推理规则找到一个支持值或证实约束检测必然失败，则不进行该次约束检测.

根据这一原则，本文设计了一个找支持值的算法 AC-I$^+$：当要为某个约束关系 R_{ij} 中 $X_i=a$ 在 D_j 中找支持值时，先利用①～③的 I，III 找 a 的支持值(见函数 SeekInferableSupport)，如果找不到，再对 D_j 中的值 b 逐一检测. 而每次检测前，先利用①～③的 I，III 和 IV 判定 b 是否支持 a，若还不能下判定，才用约束检测去检查 $R_{ij}(a,b)$ 是否成立(见函数 NextSupport). 它与 AC-Inference 相比，减少了同样多的约束检测次数，而只需要 $O(ed)$ 的空间开销. 下面算法是找支

© 中国科学院软件研究所 http://www.jos.org.cn

持值算法的具体描述.它采用类似 AC-7 的数据结构.

(1) M 数组.记录每个变量的每个值是否存在于当前值域,$M(i,a)=\text{True}\Leftrightarrow a\in D_i$.其中 first($D_i$)返回 D_i 中最小值;last(D_i)返回 D_i 中最大值;next(a,D_i)返回 D_i 中比 a 大的最小值 a';

(2) $S[j,b]=\{(i,a)/$在 R_{ij} 中 $V_j=b$ 是 $V_i=a$ 在 D_i 中的支持值$\}$.函数 in($S[j,b]$,i,a),如果 $S[j,b]$ 中含值对 (j,b),则返回 True,否则,返回 False;函数 find($S[i,a]$,j,b),如果$\exists\ b'\in D_j$,$S[i,a]$ 中含(j,b'),则 $b\leftarrow b'$,且返回 True,否则,返回 False;

(3) inf_Support$[(i,j),a]=\{b/V_i=a$ 在 D_j 中的支持值不小于 $b\}$.用以确保算法符合元知识规则①;

(4) DeleteStream 栈结构.记录被删除而未进行处理的"死值"($V_i=a$).

```
Procedure ACI+(i,j,a;out b)
{/* 对关系 Rij,为 Vi=a 在 Dj 中找一个支持值 b,
    如果找不到,返回 NULL */
if SeekInferableSupport(i,j,a,b) then return b;
else if Dj=∅ then nosupport←True;
else b←inf_support[(i,j),a];
     NextSupport(i,j,a,b,nosupport);
     if nosupport then return NULL;
     else return b;
}
Function SeekInferableSupport(in i,j,a;
            out b):Boolean;
{/* 尝试利用约束的元知识找支持值,找到则赋
    值于 b,返回 True,否则返回 False */
if (Rij具有自反性) and M[j,a]
then b←a;return True;
if (Rij具有对称性)
then
  for b∈Dj do if M[i,b] and in(S[i,b],j,a)
                then return True;
  if M[j,b] and find(S[j,a],i,b)
  then return True;
if find(S[i,a],j,b) then return True;
     return False;}
Procedure NextSupport(in i,j,a;in out b;
                      out nosupport)
{/* 为 Vi=a 在 Dj 中找一个支持值 b,如果找到,
    nosupport=False,否则返回 nosupport=True */
While not M[j,b] and b<last(Dj)
do b←next(b,Dj);
nosupport←not M[j,b];
do
{if ((Rij是等价关系) and (b≠a))
    or ((Rij具有反自反性) and (b=a))
    or ((Rij具有反对称性) and (in(S[i,b],j,a)
        or find(S[j,a],i,b)))
    or ((Rij具有对称性)
        and (inf_surport[(j,i),a]>b
            or inf_support[(i,j),b]>a))
then goto Skip;
if inf_support[(j,i),b]≤a and M[j,b]
then Check Rij(a,b);如果成立,则
    nosupport← False 否则 nosupport← True;
Skip:b←next(b,Dj);
}While (nosupport and b<last(Dj));
}
```

2　约束一致性维护算法 MAC-H 描述

下面给出包含了上述层次原则的 MAC-H 算法描述.其中 Maintain_Constraint[R_{ij}]用于记录第 n 个约束是否被维护过.如果我们运用 1.2 节中的策略Ⅰ,Ⅱ,则在死值的约束传播中,那些因为Ⅰ,Ⅲ而未经维护的约束必须进行维护.

```
Function MAC-H: Boolean
{/* 如果证实当前赋值失败,则返回 False,否则返回
    True */
if (不可能证实当前的赋值失败)
   and (不可能删除关赋值变量值域中的死值)
then return True;
else
    initialization;
    for Rij∈arc(G) do
     if (j——当前赋值变量) and (Rij不是肯定不
        能引起死值出现的约束)
      then put Rij in ConstraintsStream;
    order (ConstraintsStream);
    for Rij∈ConstraitnsStream do
        Maintain_Constraint[Rij]←True;
        for a∈Di do
            SeekOneSupport(i,j,a,b);
            if b∈NULL then Append(S[j,b],(i,a));
                           inf_support[(i,j),a]←b;
            else remove a from Di;
                if Di=∅ then return False;
            else if (Di 变小将引起维护某些未维护的
                约束 Rij) then 维护 Rij;
            Push(DeleteStream,(i,a));
            #iif(运用 1.2 节(Ⅳ)"失败先传播")
                DeletionPropagation();
    #iif(不运用 1.2 节(Ⅳ)"失败先传播")
        DeletionPropagation();
return True;}
Procedure DeletionPropagation();
{/* 死值所引起的影响的约束传播 */
While DeletionStream≠∅ do
  pop (DeletionStream,(j,b));
  for Rij∈arc(G) do
   if not Maintain_Constraint[Rij] and
    not (Rij肯定不能引起死值出现的约束)
   then Maintain_Constraint[Rij]←True;
     for a∈Dj do SeekOneSupport(i,j,a);
  for (i,a)∈S[j,b] do SeekOneSupport(i,j,a);
}
```

© 中国科学院软件研究所 http://www.jos.org.cn

3 实验结果分析

我们以经典的CSPs问题：N-皇后问题和斑马难题(详见附录)作为例子，对AC-I$^+$和MAC-H中各种策略分别进行测试，以求出第1个解所需要的约束检测的次数#cck作为衡量算法性能的标准，并比较各种策略在同样的运行环境(486DX66)及实现细节下所需的运行时间CPU-Time. 上文在讲述各种层次原则和策略时已经对这两个问题进行了部分分析. 下面对其详细描述和分析.

斑马难题(ftp. cs. city. ac. uk/pub/constraints/cspbenchmarks)可以定义为：$X=\{X_{\text{English}}, X_{\text{Spaniard}}, X_{\text{Japanese}}, X_{\text{Italian}}, X_{\text{Norwegian}}, X_{\text{Red}}, \ldots, X_{\text{Blue}}, X_{\text{Painter}}, \ldots, X_{\text{Doctor}}, X_{\text{Dog}}, \ldots, X_{\text{Zebra}}, X_{\text{Tea}}, \ldots, X_{\text{Water}}\}$共25个变量；每个变量的值域$D_i$都为$\{1,2,3,4,5\}$，每个数码代表所在房屋的位置，如$X_{\text{English}}=2$，表示英国人住在第2座房屋. $R=\{R_1,\ldots,R_{19}\}$，其中R_1：English住在Red屋里；R_2：Spaniard养Dog；…；R_{19}：$X_{\text{Tea}}\neq X_{\text{Coffee}}\neq X_{\text{Milk}}\neq X_{\text{Juice}}\neq X_{\text{Water}}$. 在该问题中，各原则策略的具体运用如下.

E算法：初始化进行约束一致性维护，并且未赋值变量值域大小都为1时，直接进行解答的检测.

R算法：值域大小发生变化的约束才进行维护.

V算法：当值域大小不为1，不对约束"不在同一房屋"进行维护.

D算法：失败先传播.

O算法：考虑约束维护的次序，以约束关系从强到弱的静态序为维护序列. 即先维护两个一元约束，最后是最弱的"不在同一房屋"约束.

ALL算法：运用了所有策略，包括E，V，D和O.

实验结果如图4所示. 从图中可得，各种策略都对求解效率有所改善，#cck和运行时间都少于没有运用任何策略的ACI算法，其中E和R算法对斑马问题维护效率改进最大. ALL算法比ACI算法减少了89.5%的约束检测次数和88.0%的运行时间，可见，所有策略的使用能大大提高求解效率.

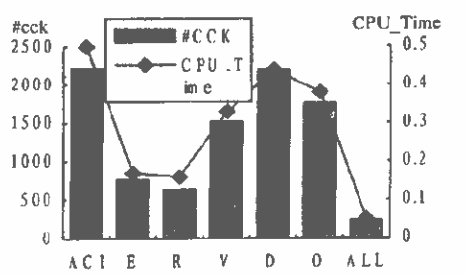


图4 斑马难题各种算法——#cck与CPU_Time(s)

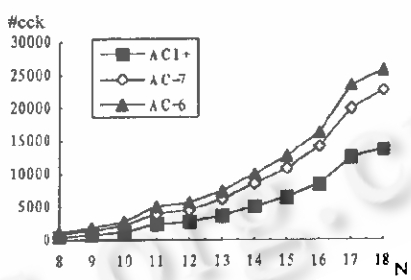


图5 N-皇后问题N-#cck

N-皇后问题可以表示为：$X=\{X_1,\ldots,X_N\}$，X_i表示第i行上皇后放的位置. $D_i=\{1,\ldots,N\}$，两两变量X_i，X_j之间都有约束R_{ij}：(1) $X_i\neq X_j$；(2) $X_i+i\neq X_j+j$(不在同一左斜下线上)；(3) $i-X_i\neq j-X_j$(不在同一右斜下线上). 在这一问题中，由于每次赋值的维护必然改变未赋值变量值域的大小，故维护时所有约束都要考虑，没必要运用R算法；又由于其约束图是完全图，故也不运用D算法. 我们对ACI，E和V算法进行了比较，其中

E算法：不在初始化中进行约束维护，因为初始状态变量之间本来就满足局部约束一致性，并且当未赋值变量值域大小都为1时直接进行解答的检测；

V算法：$|D_j|$大于3时不对R_{ij}进行维护.

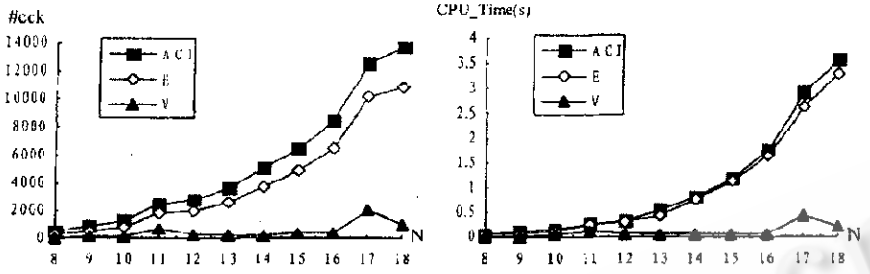
实验结果如图6所示. 从图中可得，E和V算法都提高了效率，而V算法大大降低了维护代价，平均比ACI算法减少了89.6%的约束检测次数和81.8%的CPU运行时间.

我们再比较AC-I$^+$和目前最优的约束一致性算法AC-Inference，AC-7和AC-6. 在相同维护策略下，AC-I$^+$与AC-Inference有相同的约束一致性检测次数，但AC-I$^+$只需要$O(ed)$的附加空间开销，而后者需要$O(ed^2)$的附加空间开销. AC-I$^+$与AC-6和AC-7在约束检测次数上的比较如图5所示.

4 结束语

约束满足问题是一类NP难题，作为提高问题求解效率的一种重要手段，约束一致性维护方法在许多实际应用

© 中国科学院软件研究所 http://www.jos.org.cn



(a) N-皇后问题 N-#cck　　(b) N-皇后问题 N-CPU_Time(s)

图 6

中被广泛采用.因而,如何减少约束一致性维护的代价,一直是人们的研究热点.但过去的研究仅着眼于降低维护单个约束关系的代价.而本文突破这一框架的限制,从求解、维护、约束 3 个层次对其进行了深入研究,提出了相应层次的若干原则和策略,包括有效性原则、筛选原则、排序原则和推理原则.实验结果表明:有效性原则减少了求解过程中维护的总次数;筛选原则减少了一次维护中被维护的约束总个数;排序和推理原则减少了单个约束维护的约束检测次数,从而显著地提高了求解的总效率.这一方法将为研究者提供约束一致性维护的一种新思路.沿着这一思路,我们还将结合其他 Benchmark 问题进行更深入的研究.

参考文献

1 Vipin Kumar. Algorithms for constraint-satisfaction problems: a survey. Artificial Intelligence, 1992,13(1):32～44

2 Christian Bessiere. MAC and combined heuristics: two reasons to forsake fC (and CBJ) on hard problems. In: Freuder Eugere C ed. Proceedings of the 2nd International Conference on Principles and Practice of Constraint Programming. Cambridge: Springer-Verlag, 1996

3 Christian Bessiere, Marie-Odile Cordier. Arc-Consistency and arc-consistency again. In: Proceedings of the 11th National Conference on Artificial Intelligence. Massachusetts: The AAAI Press/The MIT Press, 1993. 108～113

4 Christian Bessiere, Eugene C Freuder, Jean-Charles Regin. Using inference to reduce arc consistency computation. In: Mellish C S ed. Proceedings of the International Joint Conference'95 on Artificial Intelligence. San Francisco: Morgan Kaufmann Publishers, 1995. 592～598

5 Wallace R J. Ordering heuristics for arc consistency algorithm. In: Alto Palo ed. Proceedings of the 9th Canada Conference on Artificial Intelligence. CA: Morgan Kaufmann Publishers, 1992. 163～169

6 Ian P Gent. The constrainedness of arc consistency. In: Smolka Gert ed. Proceedings of the 3rd International Conference on Priciples and Practice of Constraint Programming. Cambridge: Springer-Verlag, 1997. 327～340

Multi-level Strategy for Maintaining Arc Consistency in Problem Solving and Its Implementation

HAN Jing　CHEN En-hong　CAI Qing-sheng

(*Department of Computer Science　University of Science and Technology of China　Hefei　230027*)

Abstract　Constraint satisfaction problems occur widely in artificial intelligence. Hence, arc consistency techniques have been widely studied to simplify constraint networks before or during the search for solutions. To reduce the cost of maintenance, the researchers have focused their work on the improvement of maintaining a single arc consistency. In this paper, from a higher point of view, the authors try to propose some principles and the corresponding strategies of three levels, which are search level, maintenance level and arc level. In this way, MAC-Dynamic and AC-I^+ are presented. The effectiveness of this approach is demonstrated experimentally on two typical benchmarks of CSPs: Zebra Puzzles and N-Queen Problem.

Key words　Arc consistency, constraint check, multi-level principle.

© 中国科学院软件研究所 http://www.jos.org.cn