

一种基于预处理技术的约束满足问题求解算法

孙吉贵^{1),2)} 朱兴军^{1),2)} 张永刚^{1),2)} 李莹^{1),2)}

¹⁾(吉林大学计算机科学与技术学院 长春 130012)

²⁾(吉林大学符号计算与知识工程教育部重点实验室 长春 130012)

摘 要 相容性技术作为约束满足问题的一种有效求解技术,不论是在求解前的预处理过程中,还是在搜索过程中,都扮演着极为重要的角色.文中对预处理阶段的相容性技术进行改进和信息抽取,提出两种应用于搜索过程中的新算法 Pre-AC 和 Pre-AC*,并嵌入到 BT 框架中,形成新的搜索算法 BT+MPAC 和 BT+MPAC*,给出了其正确性证明,通过复杂性分析得到 Pre-AC 和 Pre-AC* 的时间复杂度分别是 $O(nd)$ 和 $O(ed^2)$,明显低于目前最流行的弧相容技术的时间复杂度 $O(ed^3)$.实验测试结果表明:对于不同类别的用例,新算法的执行效率是弧相容维护算法的 2~50 倍.

关键词 约束满足问题;弧相容技术;singleton 弧相容;pre-弧相容
中图法分类号 TP18

An Approach of Solving Constraint Satisfaction Problem Based on Preprocessing

SUN Ji-Gui^{1),2)} ZHU Xing-Jun^{1),2)} ZHANG Yong-Gang^{1),2)} LI Ying^{1),2)}

¹⁾(College of Computer Science and Technology, Jilin University, Changchun 130012)

²⁾(Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun 130012)

Abstract As an effective technology for solving constraint satisfaction problem, consistency technology plays an important role not only in preprocessing, but also in searching. Firstly, this paper proposes two new consistency algorithms called Pre-AC and Pre-AC* applied during searching, which are based on the performance on an improvement of consistency during preprocessing and information extraction. Secondly this paper presents two new searching algorithms called BT+MPAC and BT+MPAC* by embedding those two consistency algorithms into the BT framework respectively. Thirdly, after proving the correctness of Pre-AC and Pre-AC*, this paper analyzes their time complexity. It is evidently that the complexities of Pre-AC and Pre-AC* are $O(nd)$ and $O(ed^2)$ respectively, which are apparently lower than $O(ed^3)$, the complexity of arc consistency algorithm. In the experiments on several kinds of instances, efficiency of the proposed algorithms is 2~50 times higher of that of maintaining arc consistency.

Keywords constraint satisfaction problem; arc consistency; singleton arc consistency; pre_arc consistency

1 引 言

约束满足问题(Constraint Satisfaction Problem,

CSP)是人工智能研究领域的一个重要分支,现实生活中的很多问题,都可以用约束满足问题来建模,如视觉(vision)、调度中的资源分配(resource allocation)、时序推理(temporal reasoning)等.约束满足

收稿日期:2007-07-12;最终修改稿收到日期:2008-04-11.本课题得到国家自然科学基金项目“扩展规则推理方法研究”(60773097)资助.

孙吉贵,男,1962年生,教授,博士生导师,主要从事人工智能、约束程序设计、决策支持系统研究.朱兴军,男,1983年生,硕士研究生,主要从事约束程序研究. E-mail: zhu_xing_jun@163.com. 张永刚,男,1975年生,博士,讲师,主要从事约束程序研究.李莹,女,1985年生,硕士研究生,主要从事自动推理研究.

问题通常都是 NP 难问题,在其众多求解算法中,基于回溯的搜索算法(Backtracking algorithm, BT)是一个完备的核心算法.该算法在选择实例化变量时,采用深度优先策略,若相容性检查失败则启动回溯机制,并通过引入展望(look-ahead)和回顾(look-back)两种模式,显著地提高了搜索效率^[1].基于冲突的向后跳转^[2]和动态的回溯算法^[3]等是回顾模式类的搜索算法;基于相容性技术的算法是展望模式类的搜索算法.而弧相容(Arc Consistency, AC)则是众多相容性算法中一个高效的相容性技术^[4],如算法 AC3^[5]、AC4^[6]、AC2001^[7]等.由于弧相容维护(Maintaining Arc Consistency, MAC)具有高效的求解效率和低额空间代价的特点,所以 MAC 是目前求解约束满足问题的一个主流搜索技术.

弧相容技术的研究工作主要有两个方面:一方面是在原有算法上的改进,如文献^[5-7];另一方面,在问题求解之前,进行一定的预处理,这对问题以后的求解是十分有意义的.因此,弧相容技术也广泛地应用在预处理步骤中^[8-11].在近期的研究工作中, singleton 弧相容算法(Singleton Arc Consistency, SAC)是时间和空间都比较理想的一种相容性技术.在 Debruyne 等人于 1997 年提出的 SAC1 算法^[9]基础上,2004 年~2005 年, Barták 和 Bessière 相继提出基于 AC4 和 AC2001 的 SAC 算法^[10-11],这一系列算法是从执行效率的角度上对原有算法的一种改进.而文献^[12-13]是对 SAC 理论和技术方面的一些研究.虽然 SAC 技术在预处理阶段中起到的作用十分显著,但如果要在搜索过程中维持 SAC 这一相容性,所需要的时间和空间仍然是不可忽视的.尤其是对规模和难度都比较大的问题,时间和空间已经成为它们广泛应用的瓶颈^[10,14],主要问题在于先前的研究工作中,预处理过程和搜索过程中的相容性技术都是彼此独立的,且是无信息交流的.

基于此,本文首次在预处理阶段和搜索过程中的相容性技术之间,建立起信息交流的关系,打破以往彼此独立的模式.在研究预处理阶段的相容性技术的同时,通过对预处理阶段的 SAC 算法进行改进,有效地利用其中的信息,提出了两种新相容性算法 Pre-AC 和 Pre-AC*,并将其嵌入到 BT 算法框架中,形成新的搜索求解算法 BT+MPAC 和 BT+MPAC*.在搜索过程中,Pre-AC 和 Pre-AC* 对问题压缩并不需要约束传播,因此,效率上要高于目前应用的弧相容技术.我们给出了算法的正确性证明和复杂性分析,分析结果表明:本文提出的 Pre-AC

和 Pre-AC* 的时间复杂度分别是 $O(nd)$ 和 $O(ed^2)$,明显低于弧相容技术的复杂度(其时间复杂度是 $O(ed^3)$)^[15].实验测试结果也表明:本文提出的求解算法在搜索效率上是目前流行搜索算法 MAC 的 2~50 倍.

2 约束满足问题和相容性技术

约束满足问题(Constraint Satisfaction Problem, CSP)通常表示成一个三元组 $P=(X, D, C)$,其中 X 是一个变量的有限集合,表示成 $X=\{i | (1 \leq i \leq n)\}$; D 是一个函数,它将每一个变量映射到一个有限的论域上.用 D_i 来表示变量 i 的论域; C 是一个有限约束集合,而且在约束中出现的变量只能是 X 中的变量.其解是为变量集中的每个变量从有限论域中寻找一个值,使得所有的约束都被满足.通常为了提高效率,在求解过程中积极地使用约束,删除变量中的一些肯定不参与解的值.这个过程通常称为约束传播、域过滤或者相容性技术.它在搜索过程中有效地删除许多不相容的值,压缩问题的搜索空间,从而达到提高效率的目的.而这一技术最为有效的是弧相容技术,对于二元 CSP 的约束图中的某一个弧 (i, j) ,称它是弧相容的(Arc Consistency, AC),当且仅当对于 i 论域中能满足 i 上一元约束的每一个值 a ,都在 j 的论域中存在一个值 b ,使得 b 满足 j 上一元约束,并且 (a, b) 满足 i 和 j 上的二元约束 $C(i, j)$.一个 CSP 是弧相容的,当且仅当它的约束图中的每一条弧是弧相容的.文献^[16]给出了其详细概念和相关术语.

以著名的地图着色问题为例.图 1 是该问题的约束图,图中数字表示约束满足问题中的变量,字母表示变量的论域.变量之间的连线代表二元约束关系,其含义是这两个变量不能取相同的值.图 2 和图 3 中的 F 和 S 分别代表问题求解失败和求解成功.图 3 所示的是没有相容性检查的搜索树,扩展分支数目是 14.图 2 所示的是带有弧相容性检查的搜

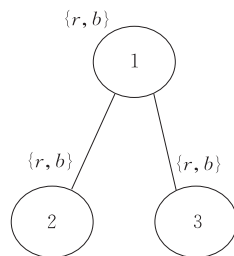


图 1 二元约束问题例子

索树,扩展分支数仅是6个,而展开的节点数目也相应减少,进而提高了搜索效率.对于规模大的问题,相容性技术的优点将更为明显.

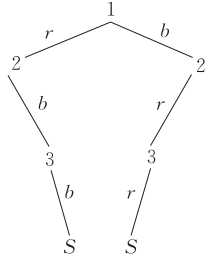


图2 有相容性检查的搜索树

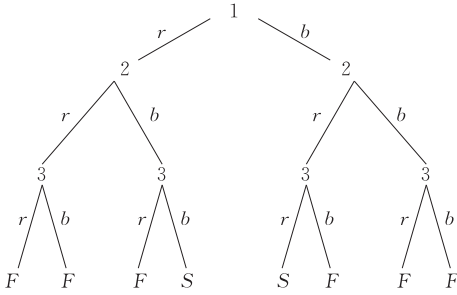


图3 没有相容性检查的搜索树

在弧相容技术的基础上,文献[9-11]引入 singleton 弧相容的概念.问题 P 是 singleton 弧相容的,当且仅当对于 $\forall i \in X, \forall a \in D_i, P|_{i=a}$ 是弧相容的.其中 $P|_{i=a}$ 是将原问题 P 中的变量 i 的论域 D_i 用单独的 $\{a\}$ 进行替换.此后满足 SAC 的一系列算法也相应地出现.

SAC1 算法^[9]并没有利用专门的数据结构来为每个值存储原 CSP 问题中所有与它弧相容的值,这就迫使在 SAC1 执行过程中,每当某个值被删除时,算法都要为所有值重新进行一次相容性传播.这样,大量的重复工作导致算法的效率很低.因此,即使空间复杂度很理想,但其高额的时间代价,使得它在实际应用中使用得较少.

SAC2^[10]是一个基于 AC4 思想的算法,它使用大量的数据结构来避免重复的相容性传播.首先将原问题的论域复制 nd 次(其中, n 是 CSP 问题中变量的个数, d 是变量论域的大小),每个值对 (i, a) 都对应一个原问题的论域.通过维持这些数据结构可以在相容性传播过程中降低时间复杂度,避免重复的传播处理,进而提高算法 SAC 的效率.

SAC-SDS^[11]是一个基于 AC2001 的算法,它和 AC2001 都是以牺牲最优时间复杂度来节省空间,但也维持一些必要的结构来避免重复的检查操作,这样能在算法的时间和空间上都达到一个比较理

想的情况.文献[11]对该算法进行了描述,其思想是为每一个 (i, a) 存储一个局部约束传播等待队列 $Q[i, a]$ 和一个弧相容的论域,算法中用数据结构 $support[i, a]$ 表示.利用这一结构,能知道当某个值被移走时,哪些值不再是 singleton 弧相容的,哪些论域需要做约束传播处理,同时它也能记录最后一次传播过程的结果.相应的局部约束传播等待队列 $Q[i, a]$ 记录了导致这次传播发生的那些变量,利用这一结构,能高效地实现弧相容算法的约束传播过程.此外,算法用结构队列 *PendingList* 来维持 singleton 弧相容性的传播过程,当算法达到一个稳定点时,该队列为空.队列 *PendingList* 中的值对 (i, a) ,表示此时与其对应的 $support[i, a]$ 论域不是 singleton 弧相容的,需要进行约束传播处理.因此,放到这个队列中等待传播.对于子程序 *propagateAC*,执行了一个原始的弧相容约束传播过程,它可以根据不同的弧相容性机制,选用不同的约束传播过程,如 AC3、AC4、AC2001 等.而另外一个子程序 *propagatesubAC* 的执行过程与 *propagateAC* 相同,只是它不需要调用 *updatesubproblems* 子过程.*updatesubproblems* 的作用是避免对所有的子问题都做约束传播处理,只是对那些由于某些值被移走后,导致不再是 singleton 弧相容的子问题进行传播.算法 SAC-SDS 的时间复杂度是 $O(end^4)$,空间复杂度是 $O(n^2 d^2)$ ^[11].

3 Pre-AC 和 Pre-AC* 算法

本节中,我们先给出一个改进的 BT 算法框架 BT-framework;之后对预处理阶段的 SAC 算法修改,提出改进的 SAC 算法;最后,在此基础上提出两个新的相容性算法 Pre-AC 和 Pre-AC*. BT 算法框架是在经典回溯算法的基础上,添加了一个预处理过程(pre-process)和一个局部相容性处理过程(local consistency).这两个过程都能加快 BT 算法的搜索效率.改进的 BT 算法框架 BT-framework 如下:

BT-framework 算法.

BT-framework(CSP: P)

```

 $P' \leftarrow \text{pre\_process}(P);$  /* pre-process phase */
if (isnosolution( $P'$ )) then return nosolution;
 $freevariables \leftarrow P.X;$ 
while (not empty( $freevariables$ )) do
    /* bt solve phase */

```

```

select a variable  $x$  and a value  $v$  for  $x$ ;
 $x \leftarrow v$ ;
 $freevariables \leftarrow freevariables - \{x\}$ ;
push( $P'$ );
 $P' \leftarrow localconsistency(P')$ ; /* local consistency */
if (isnosolution( $P'$ )) then
     $P' \leftarrow pop(P')$ ;
 $freevariables \leftarrow freevariables \cup \{x\}$ ;
return the assignments;

```

BT 算法的执行流程如图 4 所示, 其中 lc 代表局部相容性技术(local consistency), 目前效率比较高的是弧相容技术. 搜索过程大体如下: 首先对于初始问题 P_0 进行预处理操作, 得到新的问题 P_1 , 对问题 P_1 将变量 X_1 实例化为 a , 得到新的子问题 P'_1 , 再调用 lc 使之满足某种相容性, 得到一个新的子问题 P_2 , 而后对这个子问题进行求解, 依此类推, 如果提前确定该子问题没有解, 则立即回溯.

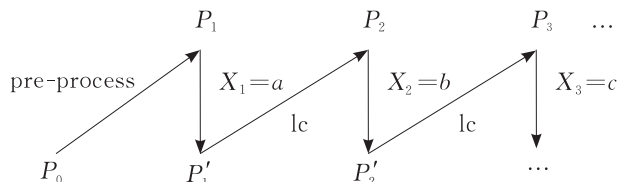


图 4 回溯算法执行过程

利用预处理阶段中 SAC 的机制, 在预处理时保留一些必要的信息. 在给出算法 Pre-AC 之前, 我们先对 SAC 算法进行修改, 称为改进的 SAC, 首先引入一个新的结构: $deleteSAC[i, a]$, 它与原数据结构 $support[i, a]$ 相对应, 在预处理过程中, 对于 $\forall (i, a) \in D$, 都有 $D = deleteSAC[i, a] \cup support[i, a]$. 即在算法 SAC 执行的过程中, 每当从 $support[i, a]$ 中移出一个值, 都要把这个值添加到集合 $deleteSAC[i, a]$ 中. 假设对于原问题 P_0 做预处理后得到的新问题 $P_1 = SAC(P_0)$, 对于 $\forall (i, a) \in D'$ (D' 是问题 P_1 的论域), 集合 $deleteSAC[i, a]$ 所存储信息的含义是: 对问题 P_1 , 变量 i 实例化 a 后所得到的新问题为 P'_1 , 这时需将集合 $deleteSAC[i, a]$ 中的值依次从 P'_1 中移走, 这才能保证问题 P'_1 是弧相容的. 因此我们对原 SAC 算法中的过程修改, 如下所示. 其中集合 $deleteSAC[i, a]$ 初始化为空 (这里只给出 SAC 算法的主要函数).

Revised-SAC 算法.

```

function propagatesubAC(in ( $X; D; C$ ), in  $Q, k, c$ ):
boolean
while pop  $j$  from  $Q$  do
    for each  $(i, a) \in D$  such that  $\exists C_{ij} \in C$  do

```

```

        if not  $\exists b \in D_j$  and  $C_{ij}(a, b)$  then
             $D_i \leftarrow D_i \setminus \{a\}$ ;
             $deleteSAC[k, c] \leftarrow deleteSAC[k, c] \cup \{(i, a)\}$ ;
             $Q \leftarrow Q \cup \{i\}$ ;
        if  $D_i = \emptyset$  then return false;
return true;

procedure updatesubproblems(in  $(i, a)$ : Value)
for each  $(j, b) \in D$  such that  $(i, a) \in support[j, b]$  do
     $support[j, b] \leftarrow support[j, b] \setminus \{(i, a)\}$ ;
     $deleteSAC[j, b] \leftarrow deleteSAC[j, b] \cup \{(i, a)\}$ ;
     $Q[j, b] \leftarrow Q[j, b] \cup \{i\}$ ;
     $PendingList \leftarrow PendingList \cup \{(j, b)\}$ ;

```

上述算法中的函数 propagatesubAC, 添加新参数 (k, c) 作为索引, 用来确定与当前函数使用的论域 D 相对应的 $deleteSAC$. 在改进的 SAC 算法中, 每当从 $support[i, a]$ 中移出值时, 就将该值加入到 $deleteSAC[i, a]$ 中. 算法的第 6、13 行实现这个过程.

通过利用 $deleteSAC$ 存储的信息, 我们给出的新局部相容性算法 Pre-AC 的思想为: 每当变量 i 实例化为值 a 时, 都要将 $deleteSAC[i, a]$ 中的值依次从问题中移出, 这样就达到了对搜索树剪枝的目的, 从而提高求解效率. 如果在移出值的过程中发现某个变量的论域变成空, 则返回假, 迫使 BT 算法启动回溯机制进行回溯处理, 重新进行求解. 伪码如下所示.

Pre-AC 算法.

```

function Pre-AC(in  $P$ : Problem,  $(i, a)$ : Value): boolean
for each  $(j, b) \in D$  such that  $(j, b) \in deleteSAC[i, a]$  do
     $D_j \leftarrow D_j \setminus \{b\}$ ;
    if  $D_j = \emptyset$  then return false;
return true;

```

Pre-AC 相容性技术是利用预处理阶段的信息, 将待求解问题的规模进行压缩, 即在搜索过程中起到高效地剪枝作用, 这样就能减少问题的搜索空间, 进而提高搜索效率. 虽然当今许多相容性技术都能实现这一功能, 如弧相容技术、路径相容技术等, 但它们没有充分地利用预处理阶段的信息, 更重要的是它们需要做大量的、高代价的约束传播, 而我们的 Pre-AC 算法并不需要进行约束传播, 这样就能避免高额的时间开销. 将 Pre-AC 嵌入到前述改进的 BT 框架中得到的新搜索求解算法我们称之为 BT+MPAC.

有时候维持 Pre-AC 不能够像维持弧相容性那样, 移走更多的冗余值, 如图 5 为变量 X 和 Y 之间

的二元约束关系。

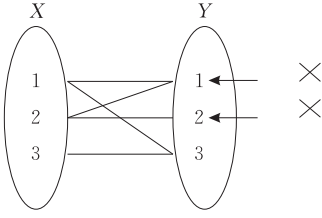


图 5 X, Y 之间的约束关系

图 5 中的数字代表变量的论域,它们之间的连线代表变量 X 和 Y 满足约束关系的值对集合. 现在设某一变量 Z_1 被实例化,这时按 Pre-AC 算法,假设 Y 中的值 1 被移走,而此时 X 中的所有值在 Y 中仍然都能够得到支持,所以 X 的论域不会变化. 之后变量 Z_2 又被实例化,执行 Pre-AC 算法,假设 Y 中的值 2 也被移走,由于 Pre-AC 所维持的是原问题的弧相容性,因此仍可认为 X 中的值 2 在 Y 中存在一个支持(Y 中的值 1),所以 X 的论域没有变化. 但事实上,变量 X 中的值 2 在 Y 中已经不存在支持(因为 Y 中的 1 在上一次 Pre-AC 算法执行时已经被删除). 由此,我们可以在 Pre-AC 算法上对相容性做进一步的检查,若某个变量的论域在 Pre-AC 处理之后发生变化,则就像传统的弧相容算法那样,将所有和这个变量存在约束关系的其它变量都作相应的检查,但是并不做更深一层的传播处理. 这样可以在搜索时间和相容性算法的执行时间上达到一个比较理想的效果. 我们把这种检查方法称为 Pre-AC* 算法,算法如下.

Pre-AC* 算法.

```
function Pre-AC* (in  $P, (i, a)$ ): boolean
  for each  $(j, b) \in D$  such that  $(j, b) \in deleteSAC[i, a]$  do
     $D_j \leftarrow D_j \setminus \{b\}$ ;
    if  $D_j = \emptyset$  then return false;
     $Q \leftarrow Q \cup \{j\}$ ;
  while pop  $j$  from  $Q$  do
    for each  $i \in X$  such that  $\exists C_{ij} \in C$  do
      for each  $a \in D_i$  do
        if not  $\exists b \in D_j$  and  $C_{ij}(a, b)$  then
           $D_i \leftarrow D_i \setminus \{a\}$ ;
        if  $D_i = \emptyset$  then return false;
  return true;
```

4 复杂性和正确性分析

为了更好地衡量上节所提出的相容性算法 Pre-AC 和 Pre-AC* 的性能,本节对这两个算法的

空间复杂度和时间复杂度进行分析,同时对它们的正确性也给出证明.

定理 1. 算法 Pre-AC 和 Pre-AC* 的最坏空间复杂度是 $O(n^2 d^2)$ (n 代表变量的个数, d 代表变量论域的大小,下同).

证明. 对于算法 Pre-AC 本身只用到一个数据结构 $deleteSAC$, 每个变量在最坏情况下,至多只存储其它所有变量的整个论域,所以它所消耗的空间是 $O(nd)$, 而这样的结构一共存在 nd 个,所以算法的最坏空间复杂度是 $O(n^2 d^2)$. 而算法 Pre-AC* 只比原算法 Pre-AC 多使用一个等待队列 Q , 它的空间大小是 $O(n)$, 故此 Pre-AC* 的最坏空间复杂度也是 $O(n^2 d^2)$. 证毕.

定理 2. 算法 Pre-AC 的最坏时间复杂度是 $O(nd)$, 算法 Pre-AC* 的最坏时间复杂度是 $O(ed^2)$ (其中的 e 代表约束网络中的边的个数).

证明. 在算法 Pre-AC 的一次执行中,可以看出每次都要从 $deleteSAC[i, a]$ 中读取一个值对,每一个变量的每个值在这个集合中至多出现一次,并只能读取一次,所以它的最坏时间复杂度是每个 $deleteSAC[i, a]$ 的空间大小 $O(nd)$. 而算法 Pre-AC* 要比算法 Pre-AC 做更进一步地约束传播处理,它从等待队列 Q 中依次取变量进行检查,对于每一个有向弧,至多进行一次约束检查,在这一次约束检查过程中,对任意一个变量的一个值都要从另一个变量中寻找支持,这一过程所用的时间是 $O(d)$. 而对于每个弧,要为一个变量的 d 个值寻找支持,所以它耗用的时间是 $d \times O(d)$, 即 $O(d^2)$. 原问题中一共有 $2e$ 条有向弧,因此算法 Pre-AC* 的最坏时间复杂度是 $2e \times O(d^2)$, 即 $O(ed^2)$. 证毕.

下面我们对算法 Pre-AC 和 Pre-AC* 的正确性进行证明,证明过程用到了文献[9-11, 15]的结果: 算法 SAC 和算法 AC 的执行过程都是正确的.

定理 3. 算法 Pre-AC 和 Pre-AC* 是正确的.

证明. 相容性算法的正确性是指在相容性检查过程中,对于任何一个被移走的值,它都一定不会在原问题的解中出现,即被移走的值将来不可能被扩展成原问题的解. 因此对于算法 Pre-AC 的正确性证明,只需证任意阶段 P'_i 到 P_{i+1} 这一相容性处理过程中的正确性即可(P'_i 是对变量 X_i 实例化为 a_i 后得到新的子问题. P_{i+1} 是对 P'_i 进行相容性处理得到的问题). 现在假设在这个阶段进行相容性检查时,存在某个变量的某个值被删除,并且这个值在将来能够扩展成原来问题的一个解. 不失一般性,设这个

变量和值分别是 X_j 和 a_j . 由于这个值在子问题 P'_i 中能够扩展成一个解, 即当 X_i 实例化为 a_i 且 X_j 实例化为 a_j 的时候, 这个子问题至少存在一个解, 而这个子问题和原问题的约束关系是不变的, 所以这个解也是原问题的一个解. 这样由 SAC 算法和 AC 算法的正确性可以保证 $(X_j, a_j) \in \text{support}[i, a_i]$. 现在由于这个值被算法 Pre-AC 所删除, 因此 $(X_j, a_j) \in \text{deleteSAC}[i, a_i]$, 进而可以得出 $(X_j, a_j) \notin \text{support}[i, a_i]$, 产生矛盾. 由此得出算法 Pre-AC 是正确的. 算法 Pre-AC* 的正确性同理可证. 证毕.

5 实验结果

这里使用了三大类测试用例, 分别是随机的约束满足问题、经典图着色问题, 还有一些标准的测试用例 (benchmark). 通过对这些例子的测试, 可以看出新的相容性算法比原有 MAC 在效率上有着不同程度的提高. 在下面的图表中: BT+MAC 代表在经典的回溯算法中弧相容维护技术, BT+MPAC 和 BT+MPAC* 分别表示在搜索过程中维持 Pre-AC 相容性和 Pre-AC* 相容性. 这三种搜索算法在测试的程序中都使用最小论域启发式策略. 使用 C++ 语言在“明月”^[17] 平台上编写程序, 测试环境为: 硬件 DELL Intel PIV 3.0GHz CPU/512MB RAM; 软件 Windows XP Professional SP2/Visual C++6.0. 其中每个用例测试 10 次取平均值作为最后结果.

(1) 随机约束满足用例

随机问题产生器随机地生成二元约束满足问题 (<http://www.lirmm.fr/bessiere/generator.html>), 问题的难度和规模可以根据参数的设置来控制, 由于问题具有随机性, 所以绝大部分通用约束求解算法和通用启发式策略都采用这种测试方法^[18]. 二元随机约束满足问题有 4 个描述参数 $\langle n, m, p_1, p_2 \rangle$, 其中 n 代表变量的个数, m 代表变量论域的大小 (这里假设所有变量的论域大小相同), p_1 代表此约束满足问题的密度, 即约束图中边的个数和 $n \times (n-1)/2$ (完全图中边的个数) 的比值, p_2 为每个二元约束的松紧度. 本文对问题规模为 $n=m=30$, 约束密度为 0.40, p_2 介于 0.40~0.61 之间的用例进行测试, 实验结果如图 6 和图 7. 从图 6 (p_2 介于 0.40~0.50 之间) 可以看出, 我们的 BT+MPAC 和 BT+MPAC* 在算法的执行效率上是算法 BT+MAC 的约 2~4 倍. 而对于图 7 (p_2 介于 0.51~0.61 之间), BT+MPAC

和 BT+MPAC* 算法的执行效率是算法 BT+MAC 的 4~9 倍.

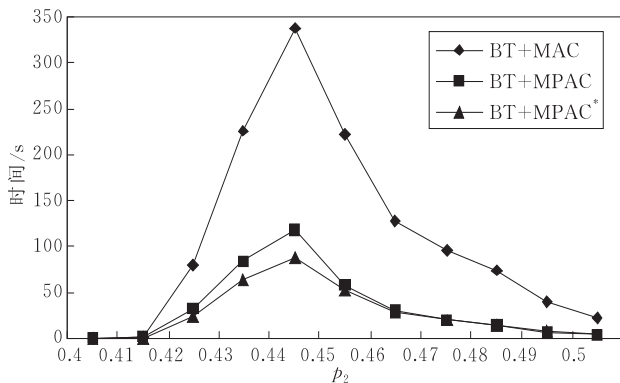


图 6 问题 $\langle 30, 30, 0.40, p_2 \rangle$, p_2 介于 0.40~0.50 之间

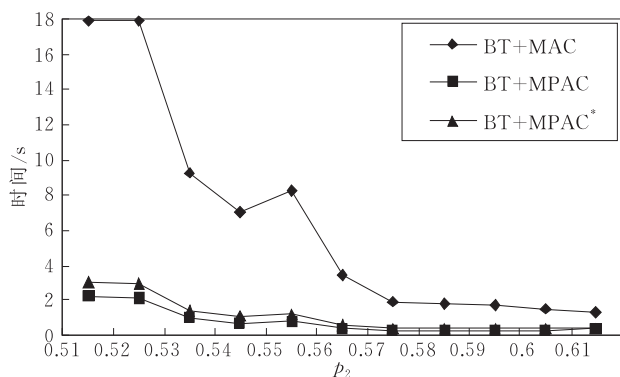


图 7 问题 $\langle 30, 30, 0.40, p_2 \rangle$, p_2 介于 0.51~0.61 之间

(2) 经典图着色问题

图着色问题是一个非常经典的图论问题, 也是约束满足问题中极其具有代表性的问题. 通过随机生成图着色测试样例 (<http://ai.uwaterloo.ca/~vanbeek/software/csplib.tar.gz>), 分别测试四着色和五着色两类问题, 其中每类问题都含有 200 个顶点、4975 条边. 并把每一类问题都分为有解的和无解的两种情况 (由于随机生成的图有可能是一个非平面图, 所以存在没有解的可能性), 分别进行测试和比较. 如表 1 中的数据所示, 每类问题前列是没有解的情况, 后列是有解的情况. 从表 1 可以清晰地看出, 对于没有解的情况, #4color 问题, 我们的 BT+MPAC 算法和 BT+MPAC* 算法的执行效率是原始算法 BT+MAC 的 10 倍; 而对 #5color 问题而言, BT+MPAC 算法和 BT+MPAC* 算法的执行效率是 BT+MAC 算法的 50 倍. 而对有解的例子, 无论是 #4color 问题还是 #5color 问题, 算法 BT+MPAC 和 BT+MPAC* 的执行效率都是算法 BT+MAC 的 2 倍.

表 1 #4color 和 #5color 图着色问题

算法	#4 color 问题		#5 color 问题	
	问题无解	问题有解	问题无解	问题有解
	时间/s	时间/s	时间/s	时间/s
BT+MAC	2.3910	0.2708	24.658	0.5015
BT+MPAC	0.2097	0.1923	0.5173	0.3014
BT+MPAC*	0.2114	0.1920	0.5251	0.3045

(3) 标准库的测试用例(benchmark)

此外我们还测试了一些标准库中的测试样例(<http://cpai.ucc.ie/05/Benchmarks.html>),从中随机选取了两类问题,一类是 frb 问题,对这类问题进行求解一般来说比较困难.文献[19-20]对这类问题进行了详细的研究.另一类是著名的“鸽巢问题(pigeons)”,在测试这组例子的过程中,我们使用二元约束关系,没用使用全局的 alldifferent 约束,并且没有考虑问题本身的对称性和问题固有特点.测试结果见表 2.

表 2 标准库中的测试用例

算法	Frb30-15	Frb35-17	Frb40-19	pigeons
	时间/s	时间/s	时间/s	时间/s
BT+MAC	10.203	19.64	349.51	10872
BT+MPAC	3.234	5.50	167.67	1620
BT+MPAC*	3.766	6.44	126.03	4716

从表 2 中可以看出对于不同规模的 frb 问题,新的 BT+MPAC 算法和 BT+MPAC* 算法在效率上是 BT+MAC 算法的 3 倍.对于 pigeons 问题,BT+MPAC 算法的执行效率大约是 BT+MAC 算法的 7 倍,BT+MPAC* 算法的执行效率是 BT+MAC 算法的 2 倍.

6 结 语

相容性技术是求解约束满足问题的一个重要技术,无论是应用在求解问题前的预处理阶段还是在搜索过程中保持相容性都已经被证明是一个十分有效的手段.但是在先前的研究工作中,这两个阶段的相容性技术都是彼此独立的,即它们之间没有信息交流.本文对预处理阶段中的 SAC 算法进行了改进,提出了一种利用预处理阶段信息的新相容性算法 Pre-AC 和 Pre-AC*,它们打破以往与预处理阶段的相容性技术独立的模式,在搜索中能够积极地利用预处理中的信息来提高求解效率.并将它们嵌入到 BT 算法框架中,形成新的搜索算法 BT+MPAC 和 BT+MPAC*.尽管 Pre-AC 和 Pre-AC* 削减冗余值的能力要比 AC 算法弱,实验结果证实

BT+MPAC 大多数情况下明显比 BT+MAC 展开的节点数多,BT+MPAC* 展开的节点数略多于 BT+MAC 展开的节点数,但文献[16]中已经明确指出,削减冗余值能力的强弱和搜索算法的执行效率并不是一个简单的线性关系.因此,我们的两种算法都是在取得最好的时间效率条件下的相容性检查与搜索空间的折衷.理论分析表明 Pre-AC 和 Pre-AC* 的时间复杂度分别是 $O(nd)$ 和 $O(ed^2)$,明显低于弧相容技术(其时间复杂度是 $O(ed^3)$),实验结果也表明我们的算法在性能上明显优于目前主流的求解技术 MAC.

参 考 文 献

[1] Rina Dechter. Constraint Processing. Morgan Kaufmann, 2003

[2] Patrick Prosser. Hybrid algorithms for the constraint satisfaction problems. Computational Intelligence, 1993, 9(3): 268-299

[3] Ginsberg M L. Dynamic backtracking. Artificial Intelligence, 1993, 1: 25-46

[4] Sabin Daniel, Freuder E C. Contradicting conventional wisdom in constraint satisfaction//Borning Alan. Proceedings of the 2nd International Workshop on Principles and Practice of Constraint Programming. USA, 1994: 10-20

[5] Mackworth A K. Consistency in networks of relations. Artificial Intelligence, 1977, 8(1): 99-118

[6] Mohr R, Henderson T C. Arc and path consistency revised. Artificial Intelligence, 1986, 28(2): 225-233

[7] Bessière Christian, Régin Jean Charles. Refining the basic constraint propagation algorithm//Proceedings of the 17th International Joint Conference on Artificial Intelligence. Seattle WA, Morgan Kaufmann, 2001: 309-315

[8] Lecoutre Christophe, Cardon Stéphane, Vion Julien. Conservative dual consistency//Proceedings of the 22nd Conference on Artificial Intelligence. Vancouver, Canada, 2007: 237-242

[9] Debruyne R, Bessière C. Some practical filtering techniques for the constraint satisfaction problem//Proceedings of the IJCAI'97. Japan, 1997: 412-417

[10] Barták Roman, Erben Radek. A new algorithm for singleton arc consistency//Proceedings of the FLAIRS. Florida, USA, 2004: 257-262

[11] Bessière Christian, Debruyne Romuald. Optimal and suboptimal singleton arc consistency algorithms//Proceedings of the IJCAI. Edinburgh, Scotland, Professional Book Center, 2005: 54-59

[12] Bessière Christian, Debruyne Romuald. Theoretical analysis of singleton arc consistency//Proceedings of the ECAI'04 Workshop on Modeling and Solving Problems with Constraints. Valencia, Spain, 2004: 20-29

- [13] van Dongen M R C. Beyond singleton arc consistency//Proceedings of the 17th European Conference on Artificial Intelligence. Garda, Italy, 2006: 163-167
- [14] Lecoutre Christophe, Prosser Patrick. Maintaining singleton arc consistency//Proceedings of the 3rd International Workshop on Constraint Propagation and Implementation Held with CP'2006. Nantes, France, 2006: 47-61
- [15] Bessi re Christian, R gin Jean Charles, Yap Roland H C, Zhang Yuanlin. An optimal coarse-grained arc consistency algorithm. Artificial Intelligence, 2005, 165(2): 65-185
- [16] Tsang Edward. Foundations of Constraint Satisfaction. London and San Diego: Academic Press Limited, 1993
- [17] Sun Ji-Gui, Jing Shen-Yan. Solving non-binary constraint satisfaction problem. Chinese Journal of Computers, 2003, 26(12): 1746-1752(in Chinese)
(孙吉贵, 景沈艳. 非二元约束满足问题求解. 计算机学报, 2003, 26(12): 1746-1752)
- [18] Gent Ian P, Macintyre Ewan, Prosser Patrick, Smith Barbara M, Walsh Toby. Random constraint satisfaction flaws and structure. Journal of Constraints, 2001, 6(4): 345-372
- [19] Xu Ke, Li Wei. Exact phase transitions in random constraint satisfaction problems. Journal of Artificial Intelligence Research, 2000, 12: 93-103
- [20] Xu Ke, Boussemart Frederic, Hemery Fred, Lecoutre Christophe. A simple model to generate hard satisfiable instances//Proceedings of the IJCAI. Edinburgh, Scotland, 2005: 337-342



SUN Ji-Gui, born in 1962, professor, Ph.D. supervisor. His research interests include artificial intelligence, constraint programming and intelligence information processing.

ZHU Xing-Jun, master candidate. His main research interest is constraint programming.

ZHANG Yong-Gang, Ph.D., lecturer. His main research interest is constraint programming.

LI Ying, master candidate. Her main research interest is automated reasoning.

Background

CSP is an important researching area in artificial intelligence. However, it is the NP-complete task of determining whether a given CSP instance has a solution. As an effective technology for solving CSP instance, consistency technology plays an important role not only in preprocessing, but also in searching. However these two phases are separated as far as we know. So the efficiency of solving problems is not satisfied. This paper proposes two new consistency algorithms applied during searching based on preprocessing and embedded into BT framework to form two searching algorithms. The authors analyze the time and space complexity and prove

correctness of them theoretically. Besides, experiments show that the proposed algorithms have a better performance. All our algorithms mentioned in this paper were implemented in C++ and embedded into the constraint solving platform "Ming Yue 1.0" designed by the authors. The idea of this paper is inspiring since it does not follow the traditional mode. Moreover it exploits the information enough in preprocessing to reduce the search space and speed up solving process by removing the inconsistent values. This paper was supported by the National Natural Science Foundation of China (grant No. 60773097).