最大受限路径相容约束传播算法的研究进展

张永刚 程竹元

(吉林大学计算机科学与技术学院 长春 130012)

(符号计算与知识工程教育部重点实验室(吉林大学) 长春 130012)

摘 要 约束传播技术对于约束满足问题的求解性能至关重要。约束传播技术在一个预处理过程中能彻底地移除一些局部不相容值,或者在搜索期间高效地剪枝搜索树。最大受限路径相容算法(max Restricted Path Consistency,maxRPC)是最近提出的一种强相容性约束传播算法,它能够删除更多不相容值,在解决复杂问题中取得了很好的效果。文中对弧相容算法 AC 和最大受限路径相容算法 maxRPC 的相关算法 AC3,AC3rm,maxRPC1,maxRPC2,maxRPCrm,maxRPC3 等及其相关变体分别进行介绍和比较。在 Mistral 求解器上的实验测试结果验证了各种算法的性能。

关键词 最大受限路径相容算法,约束求解,优化算法,相容性技术

中图法分类号 TP181 **文献标识码** A

Research Progress on Max Restricted Path Consistency Constraint Propagation Algorithms

ZHANG Yong-gang CHENG Zhu-yuan

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

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

Abstract Constrained propagation technology is critical to the performance of constraint satisfaction problems. Constrained propagation technology completely removes some local inconsistency values during a preprocessing process,or efficiently pruning the search tree during the search. The max restricted path consistency algorithm (maxRPC) is a strong consistency constraint propagation algorithm proposed recently,which can remove more inconsistent values,and achieves good results in solving complex problems. In this paper,some algorithms,such as AC3,AC3rm,maxRPC1,maxRPC2,maxRPCrm,maxRPC3 and other related algorithms which are related to the arc consistency algorithm AC and max restricted path consistency algorithm maxRPC,were introduced respectively and compared with each other. The performance of the proposed algorithm is verified by the experimental results on the mistral solver.

Keywords Max restricted path consistency algorithm,Constraint solving,Optimization algorithm,Consistency technology

1 引言

约束满足问题(Constraint Satisfaction Problems,CSP)在编码多变的组合问题实例中发挥着重要作用。约束是限制能够同时分配给变量的值的局部信息。一个约束满足问题(CSP)涉及到对一个约束网络寻找解,也就是满足所有约束的赋值指派。约束确定了给定参数子集允许取的参数值组合。为解决 NP-难的任务,研究者们提出了很多指数级的搜索算法。为避免组合爆炸,必须对搜索树尽可能多地剪枝。因此,传播技术用来搜索之前或搜索过程中移除的一些局部不相容值[1]。

弧相容(Arc Consistencies,AC)和路径相容(Path Consistencies,PC)是研究得最多的局部相容性。执行 AC 的开销很小;而 PC 移除一些不相容的值对时,复杂度很高,并且执行 PC 会对约束图的连接带来一些改变[2],这些缺点限制了 PC 的应用。

约束网络越难,传播技术越有用。因此,找到一个比 AC 局部相容性强并且没有 PC 的缺点的局部相容性算法很重要。于是,受限路径相容(Restricted path consistency-RPC,RPC1)应运而生。RPC1 是一个很有前景的局部相容算法,在移除大多数路径相反不相容(Path Inverse Cnconsistent,PIC)值时,它消耗的时间只比 AC 稍多。另外,RPC1 不删除任何值对,因此不增加约束和修改网络。后来,人们又扩展了 RPC1 到新的局部相容算法——最大受限路径相容算法 maxRPC,分别提出了 RPC2,RPC3,maxRPC1,maxRPC2,maxRPCrm,maxRPC3 等算法及其相关变体。

本文分别对 AC3[3],AC3rm,maxRPC1,maxRPC2,maxRPCrm,maxRPC3 及其相关变体算法的基本思想进行概述,并简单进行对比分析。

文中第 2 节介绍一些基本概念和定义;第 3—5 节分别介绍弧相容算法、最大受限路径相容算法及相关算法的对比实验;最后得出结论,并给出未来的扩展和研究工作。

2 约束满足问题的相关概念

定义 1 一个约束满足问题(CSP)定义为一个三元组 (X,D,C),其中 $X=\{x_1,x_2,\cdots,x_n\}$ 是一个由 n 个变量组成的

本文受国家自然科学基金项目(61170314,61373052),吉林省科技发展计划项目(20170414004GH)资助。

张永刚(1975—),男,博士,副教授,CCF 会员,主要研究方向为约束求解与优化,E-mail:410759804@qq.com;程竹元(1993—),女,硕士生,CCF 会员,主要研究方向为约束求解与优化,E-mail:qing_luo_xiao_shan@163.com(通信作者)。

集合；$D=\{D(x_1),\cdots,D(x_n)\}$是一个域的集合；每一个变量都有一个域，$d$ 是最大域的大小；$C=\{c_1,c_2,\cdots,c_e\}$是一个由 e 个约束组成的集合。本文主要关注二元 CSPs，一个二元约束 c_{ij} 包括变量 x_i 和 x_j。

定义 2(弧相容(AC)) 给定一个约束网络 $N=(X,D,C)$、一个约束 $c\in C$ 和一个参数 $x_i\in X(c)$，一个值 $v_i\in D(x_i)$ 是与 D 中 c 相容的，当且仅当存在一个值元组 τ 满足 c，使得 $v_i=\tau[\{x_i\}]$。这样的值元组称为对 c 上(x_i,v_i)的支持。

论域 D 是 c 上 x_i(广义)弧相容的，当且仅当 $D(x_i)$中所有的值与 D 中的 c 是相容的，即 $D(x_i)\subseteq\pi\{x_i\}(c\bigcap\pi X(c)(D))$。

网络 N 是弧相容的，当且仅当 ϕ 是唯一的比对所有约束上所有参数是弧相容的 D 更紧致的域。

定义 3(路径相容(PC)) 令 $N=(X,D,C)$是一个标准约束网络。

给定 X 中两个参数 x_i 和 x_j，$(v_i,v_j)\in D(x_i)\times D(x_j)$的值对是路径相容的，当且仅当对于参数的任意序列 $Y=(x_i=x_{k_1},x_{k_2},\cdots,x_{k_p}=x_j)$使得对所有的 $q\in[1,\cdots,p-1]$，$c_{k_p,k_{p-1}}\in C$，存在一个值元组$(v_i=v_{k_1},v_{k_2},\cdots,v_{k_p}=v_j)\in\pi Y(D)$使得对所有的 $q\in[1,\cdots,p-1]$，$(v_{k_p},v_{kp+1})\in c_{k_q,k_{q+1}}$。

网络 N 是路径相容(PC)的，当且仅当对所有的参数对$(x_i,x_j)(i\neq j)$，任意(x_i,x_j)上的局部相容值对是路径相容的[1]。

定义 4(受限制路径相容(RPC)) 一个值 $a_i\in D(x_i)$是 RPC，当且仅当它是 AC 的，并且对于每个约束 c_{ij} 满足当 a_i 有唯一的支持 $a_j\in D(x_j)$时，值对(a_i,a_j)是 PC 的。

定义 5(最大受限制路径相容(MaxRPC)) 一个值 $a_i\in D(x_i)$是 maxRPC，当且仅当它是 AC 的，并且对于每个约束 c_{ij}，在 $D(x_j)$中存在 a_i 的支持 a_j 时，满足值对(a_i,a_j)是 PC 的[4]。

一个变量是 RPC(或者 maxRPC)，当且仅当它的所有值都是 RPC(或者 maxRPC)。一个问题是 RPC(或者 max-RPC)，当且仅当变量域没有空域并且所有变量是 RPC(或者 maxRPC)。

定义 6 一个二元约束网络是单弧相容(SAC)，当且仅当 $\forall i\in X,D_i\neq\emptyset,\forall(i,a)\in D$ 时，$P|_{D_i=\{a\}}$ 有一个弧相容子域。

图 1 给出几种实用的传播技术之间的关系。$A\rightarrow B$ 表示 A 的局部相容性强于 B，A 和 B 之间的非通路直线表示 A，B 之间没有相容关系是否更强的可比性。

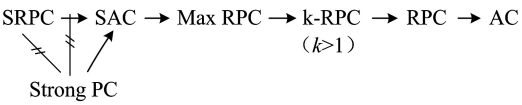

图 1 几种局部相容性算法之间的关系

3 弧相容算法

AC 算法被广泛使用，而且提出时间较早，并产生了很多变体算法，如 AC1，AC2，AC3，AC4，AC5，AC6，AC7，AC2000，AC2001，AC3.1，AC3.2，AC3.3，AC3r 和 AC3rm 等，比较流行的是 AC3，AC2001 和 AC3rm。其中，AC3 算法在相当长的时间内被广泛应用；AC2001，AC3r 和 AC3rm 是 AC3 的变体，由于实现效果好，也被广泛使用。

3.1 AC3 算法

3.1.1 AC3 算法

算法 1 给出 AC3 算法的框架[3]。其使用一个传播集合 Q 来记录所有需要修改的弧。对弧(C,X)修改的目的在于要删除 $dom(X)$中对于 C 已变得不相容的值。如果 revise3(C,X)(见算法 2)删除了 X 中的值，那么需要检查包含变量 X 的约束 C'中的其他变量域。seekACSupport(见算法 3)用于寻找 AC 支持。

算法 1 AC-3.X

```
1. 把约束集合 C 中的所有弧(C,X)加入等待队列 Q
2. 执行初始化部分 init3.X()
3. 当 Q≠Ø 时
4.    从 Q 中取出一个弧(C,X)
5.    如果 revise3.X(C,X)返回真,则
6.        如果 dom(X)=Ø,则返回 FAILURE
7.        否则,将与 X 相关的所有约束 C'(除了 C 以外)的所有弧加入到 Q 中
8. 返回 SUCCESS
```

算法 2 revise3(in C,X):boolean

```
1. 将|dom(X)|的值赋给 nbElements
2. 对于 dom(X)中的每个值 a
3.    如果通过 seekASupport(C,X,a)找不到支持,则将 a 从 dom(X)中删除
4. 如果 nbElements≠|dom(X)|,返回真
```

算法 3 seekACSupport(in C,X,a):boolean

```
1. 将关于(C,X,a)的第一个有效元组赋给 t
2. 当有 t 存在时
3.    进行约束检查 C(t),如果成功,则转到步骤 5
4.    调用 seekNextTuple(C,X,a,t),将 t 更新为 C 中 t<t'并且 t'[pos(X,C)]=a 的最小有效元组 t',如果没有则更新为 nil
5. 如果 t≠nil,返回真
```

3.2 AC3rm 算法

3.2.1 AC3rm 算法

AC3rm 算法是 AC3 算法的变体，它开发了多向残差(multi-directional residues)，具有很好的性能。算法 4－算法 6 是 AC3rm 算法[5]。当一个支持 t 被发现时，这个支持可以同时记录到互为支持的值的支持结构中。但是，不能保证上一个发现的支持是最小支持。最后，一个支持利用 supp(C,X,a)数据结构存储。supp 数据结构初始化为空，revise 函数测试上一个发现的支持是否有效。若失败，则从域中第一个变量开始搜索一个新的支持。算法 4 给出 ACrm 算法的传播框架，算法 5 revise 用以修复删值变量域，算法 6 seekSupport 用以搜索 AC 支持。

算法 4 doAC:boolean

```
1. Q←{(C,X)|C∈1∧X∈scp(C)}
2. while Q≠Ø do
3.    pick and delete(C,X) from Q
4.    if revise(C,X) then
5.       if dom(X)=Ø then return false
6.       Q←Q∪{(C',Y)|C'≠C,Y≠X,{X,Y}⊆scp(C')}
7. return true
```

算法 5 revise(C,X):boolean

```
1. nbElements←|dom(X)|
```

```
2. for each a∈dom(X) do
3.    if supp[C,X,a] is valid then continue
4.    t←seekSupport(C,X,a)
5.    if t=⊤ then remove a from dom(X)
6.    else for each Y∈scp(C) do supp[C,Y,t[Y]]←t
7. return nbElements≠|dom(X)|
```

算法 6　seekSupport(C,X,a):Tuple

```
1. t←⊥
2. while t≠⊤ do
3.    if C(t) then return t
4.    t←setNextTuple(C,X,a,t)
5. return ⊤
```

4　MaxRPC 算法

4.1　MaxRPC 算法简介

MaxRPC 算法主要有 maxRPC1，maxRPC2，maxRPCrm 和 maxRPC3rm 等。MaxRPC 必须保证所有值在每个约束上至少有一个路径相容支持。MaxRPC1 中，AC6 的思想被用了两次。第一次是为了证明一个值(i,a)的 maxRPC1，maxRPC1 在每个与 i 相连的变量域中寻找这个值的最小 PC 支持。为了确定与(i,a)相容的一个值(j,b)是一个 PC 支持，maxRPC1 在每个与 i,j 都相连的变量 k 的域中寻找(i,a)和(j,b)的最小共同支持[11]。MaxRPC1 的局部相容性比 RPC 的剪枝效果更强，但避免了 PC 的缺点。

MaxRPC2 算法利用数据结构存储最近发现的 PC 支持[12]，当判断一个值是否存在 PC 支持时，在搜索支持之前先判断这个最近发现的 PC 支持是否有效，从而提升效率；并且，相比于 maxRPC1，MaxRPC2 在数据结构方面的改变降低了空间复杂度[13]。算法用了两种数据结构，一个数据结构是记录删除的值的集合 DelSet，另一个记录一个值最近找到的一个 PC 支持。算法主要由两步组成，即初始化阶段和传播阶段。在初始化阶段，我们考虑每个值，并且检查它是否是 maxRPC 的。如果不是，从中移除，并且将其添加到 DelSet 中；否则我们存储 PC 支持。

MaxRPCrm 算法是一个粗粒度算法，利用了类似 AC3rm 的支持残差的思想。它的变体算法 LmaxRPCrm 算法是着较简单的 maxRPC 的轻量级版本，近似实现了 maxRPC，同时有着更好的性能和较低的空间复杂度[14]。MaxRPCrm 算法使用支持残差数据结构，这个支持在 AC3rm 中被用到。rm 表示多向残差，一个残差是程序执行期间存储的支持，用于证明一个给定的值是 AC。需要检查时，程序首先检查这个支持残差是否仍然有效，若无效，则从变量域中进行检索。数据结构在回溯中是稳定的，即不需要重新初始化或者修复，因此数据的管理开销很小。

MaxRPC3rm 是粗粒度算法。粗粒度算法的传播队列空间消耗较小，能被非常高效地实现；同时不需要管理额外的数据结构来记录值的支持，同时，它的传播框架能够非常高效地修复有序的启发式。MaxRPC3rm 算法中，当一个变量从约束队列中取出时，处理这个变量。算法使用 firstSup 方法和 nextSup 方法，可以以用户定义的方式来迭代给定值的支持。firstSup 有两个参数 C 和 X_a，并且返回在 $rel(C)$中找到的第一个支持。nextSup 有一个额外的参数：只要找到最后一个发现的支持，则存储最后一个发现的支持。

下面重点介绍 MaxRPC3 系列算法。

4.2　MaxRPC3 算法及其系列算法

4.2.1　MaxRPC3 算法

MaxRPC3 算法是粗粒度算法。LastPC 数据结构和 LastAC 数据结构的使用，避免了很多冗余的约束检查，提高了效率。以增量的方式更新 LastPC 和 LastAC 数据结构。利用额外的数据结构存储支持，避免了如 maxRPC2 和 maxRPCrm 中的大量冗余约束检查，节约了时间，并且比同样消除冗余约束检查的 maxRPC1 的空间复杂度低。算法 7 给出 maxRPC3 传播算法的框架，首先调用 initialization(算法 8)进行初始化，传播过程中，通过算法 9－算法 12 来检查 PC 支持，或者搜索 AC 和 PC 证人。LastPC 数据结构和 LastAC 数据结构用来存储最近发现的 PC 和 AC 支持，避免了很多冗余约束检查，大大节省了约束检查的时间。

算法 7　maxRPC3

```
1. if ¬initialization(Q,LastPC,LastAC) then
2.    return FAILURE;
3. while Q≠∅ do
4.    Q←Q−{x_j};
5.    for each x_i∈X s.t. c_{i,j}∈C do
6.       for each a_i∈D(x_i) do
7.          if LastPC_{x_i,a_i,x_j}∈D(x_j) and ¬checkPCsupLoss(a_i,x_j)
            then
8.             remove a_i;
9.             Q←Q∪{x_i};
10.         else
11.            if ¬checkPCwitLoss(x_i,a_i,x_j) then
12.               remove a_i;
13.               Q←Q∪{x_i};
14.      if D(x_i)=∅ then
15.         return FAILURE;
16. return SUCCESS.
```

算法 8　initialization(Q,LastPC,LastAC):boolean

```
1. for each x_i∈X do
2.    for each a_i∈D(x_i) do
3.       for each x_j∈X s.t. c_{i,j}∈C do
4.          maxRPCsupport←FALSE;
5.          for each a_j∈D(x_j) do
6.             if isConsisteny(a_i,a_j) then
7.                LastAC_{x_i,a_i,x_j}←a_j;
8.                if searchPCwit(a_i,a_j) then
9.                   maxRPCsupport←TRUE;
10.                  LastAC_{x_i,a_i,x_j}←a_j
11.                  if (rm) then LastPC_{x_i,a_i,x_j}←a_i;
12.                  break;
13.            if ¬maxRPCsupport then
14.               remove a_i;
15.               Q←Q∪{x_i};
16.               break;
17.   if D(x_i)=∅ then
18.      return FAILURE;
19. return TRUE.
```

算法 9 checkPCsupLoss(a_i,x_j):boolean

1. if $LastAC_{x_i,a_i,x_j} \in D(x_j)$ then
2. $b_j \leftarrow \max(LastPC_{x_i,a_i,x_j}+1, LastAC_{x_i,a_i,x_j})$;
3. else
4. $b_j \leftarrow \max(LastPC_{x_i,a_i,x_j}+1, LastAC_{x_i,a_i,x_j}+1)$;
5. for each $a_j \in D(x_j), a_j \geqslant b_j$ do
6. if isConsistent(a_i,a_j) then
7. if $LastAC_{x_i,a_i,x_j} \notin D(x_j)$ and $LastAC_{x_i,a_i,x_j} > LastPC_{x_i,a_i,x_j}$ then
8. $LastAC_{x_i,a_i,x_j} \leftarrow a_j$;
9. if searchPCwit(a_i,a_j) then
10. $LastPC_{x_i,a_i,x_j} \leftarrow a_j$;
11. return TRUE;
12. return FALSE.

算法 10 searchPCwit(a_i,a_j):boolean

1. for each $x_k \in X$ s.t. $c_{i,k}$ and $c_{j,k} \in C$ do
2. maxRPCsupport←FALSE;
3. if (($LastAC_{x_i,a_i,x_k} \in D(x_k)$) and ($LastAC_{x_i,a_i,x_k} = LastAC_{x_j,a_j,x_k}$)) or(($LastAC_{x_i,a_i,x_k} \in D(x_k)$) and (isConsistent($LastAC_{x_i,a_i,x_k}$,$a_j$))) or (($LastAC_{x_j,a_j,x_k} \in D(x_k)$) and (isConsistent($LastAC_{x_j,a_j,x_k}$,$a_i$)))
3. then
4. continue;
5. if ¬searchACsup(a_i,x_k) OR ¬searchACsupp(a_j,x_k) then
6. return FALSE;
7. for each $a_k \in D(x_k), a_k \geqslant \max(LastAC_{x_i,a_i,x_k}, LastAC_{x_j,a_j,x_k})$ do
8. if isConsistent(a_i,x_k) and isConsistent(a_j,x_k) then
9. maxRPCsupport←TRUE;
10. break;
11. if ¬maxRPCsupport then
12. return FALSE;
13. return TRUE.

算法 11 searchACsup(a_i,x_j):boolean

1. if $LastAC_{x_i,a_i,x_j} \in D(x_j)$ then
2. return TRUE;
3. else
4. for each $a_j \in D(x_j)$, $a_j > LastAC_{x_i,a_i,x_j}$ do
5. if isConsistent(a_i,x_j) then
6. $LastAC_{x_i,a_i,x_j} \leftarrow a_j$;
7. return TRUE;
8. return FALSE.

算法 12 checkPCwitLoss(x_i,a_i,x_j):boolean

1. for each $x_k \in X$ s.t. $C_{i,k}$ and $C_{j,k} \in C$ do
2. witness←FALSE;
3. if $a_k \leftarrow LastAC_{x_i,a_i,x_k} \in D(x_k)$ then
4. if (($LastAC_{x_i,a_i,x_k} \in D(x_j)$) and ($LastAC_{x_i,a_i,x_j} = LastAC_{x_k,a_k,x_j}$)) or (($LastAC_{x_i,a_i,x_j} \in D(x_j)$) and (isConsistent($LastAC_{x_i,a_i,x_k}$,$a_k$))) or (($LastAC_{x_k,a_k,x_j} \in D(x_j)$) and (isConsistent($LastAC_{x_k,a_k,x_j}$,$a_i$))) then
5. witness←TRUE;
6. continue;
7. if searchACsup(x_i,a_i,x_j) and searchACsupp(x_k,a_k,x_j) then
8. for each $a_j \in D(x_j), a_j \geqslant \max(LastAC_{x_i,a_i,x_j}, LastAC_{x_k,a_k,x_j}$ do
9. if isConsistent(a_i,a_j) and isConsistent(a_j,a_k) then
9. witness←TRUE;
10. break;
11. if¬witness and checkPCsupLoss(a_i,x_k) then
12. return FALSE;
13. return TRUE.

4.2.2 MaxRPC3 系列其他算法

1)MaxRPC3rm 算法

MaxRPC3rm 是粗粒度算法,数据结构受 AC3rm 启发(rm 表示多向残差),采用 LastAC 和 LastPC 数据结构存储最近发现的 AC 和 PC 支持。由于 MaxRPC3rm 中 LastAC 和 LastPC 数据结构仅仅用来存储支持,因此时空开销较小。算法存储最近发现的 AC 支持或者 PC 支持,但是不保证字典值排序[15]较小的值不是 AC 或者 PC 支持,即存储的支持不维持有序性。与 maxRPC3 的另一个不同点是它能利用支持双向性[16],例如,一个值是另一个值的 AC 支持,那么同时另一个值也是这个值的 AC 支持;并且,当一个 PC 支持被发现时,LastAC 也被更新,因为它也是最近被发现的 AC 支持。

2)LightmaxRPC3 算法

LmaxRPC(light-maxRPC)[17]实现了 maxRPC 相容性的近似,其它实现了 maxRPC 相容性的近似,相容性弱于 maxRPC,强于 AC。它仅传播 AC 支持的丢失,而不传播 PC 证人的丢失。当移除变量 x_j 时,对于每个 $a_i \in D(x_i)$,其中 x_i 和 x_j 相互约束,lmaxRPC 仅仅检查在 $D(x_j)$中是否有一个 a_i 的 PC 支持。lmaxRPC3 和 lmaxRPC3rm 分别为 MaxRPC3 和 maxRPC3rm 的轻量级版本,它们不调用 checkPCwitLoss 和 checkPCwitLossrm 函数,同时去掉 maxRPC3 和 maxRPC3rm 主函数的第 10－13 行。利用相同的方法,也能得到 lmaxRPC2 和 lmaxRPCrm。

4.3 MaxRPC 算法的对比分析

表 1 列出了几种 maxRPC 算法的求解性能对比结果,主要包括 maxRPC1,maxRPC2,maxRPCrm,l-maxRPCrm,O-maxRPCrm,maxRPC-EN1,maxRPC3,maxRPC3rm,lmaxRPC3 和 lmaxRPC3rm 算法。对于这些 maxRPC 算法,主要比较了其数据结构上的算法思想、时间花费和空间花费方面的异同。可以看出,lmaxRPC3rm 在时间消耗和空间消耗上都具有一定优势,而且数据结构不需要占用太多空间。

表 1 各种 maxRPC 算法的求解性能对比结果

算法名称	粒度	基于的思想	时间复杂度	空间复杂度	维持数据结构特点
maxRPC1	细粒度	AC6	$O(end^3)$	$O(end)$	维持
maxRPC2	粗粒度	AC2001/3	$O(end^3)$	$O(end)$	维持
maxRPCrm	粗粒度	AC3rm	$O(en^2d^4)$	$O(end)$	LastAC 和 LastPC 仅用来存储存储残差,不增量维持
l-maxRPCrm	粗粒度	maxRPCrm	$O(end^4)$	$O(ed)$	LastAC 和 LastPC 仅用来存储存储残差,不增量维持
O-maxRPCrm	粗粒度	maxRPCrm	$O(ed^2+cd^3)$	$O(c+ed)$	LastAC 和 LastPC 仅用来存储存储残差,不增量维持
maxRPC-EN1	细粒度	AC7	$O(en+ed^2+cd^3)$	$O(ed+cd)$	维持
maxRPC3	粗粒度	AC2001	$O(end^3)$	$O(end)$	Last 结构记录最小支持
maxRPC3rm	粗粒度	maxRPCrm,AC3rm	$O(en^2d^4)$	$O(ed)$	Last 结构记录最近发现的支持,不增量维持
lmaxRPC3	粗粒度	maxRPC3	$O(end^3)$	$O(end)$	Last 结构记录最小支持
lmaxRPC3rm	粗粒度	maxRPC3rm	$O(end^4)$	$O(ed)$	Last 结构记录最近发现的支持,不增量维持

5　测试结果

实验的测试环境为 ubuntu 14.04 LTS 64 位操作系统，Intel i7-3770 处理器，8 GB 内存。选取几组二元的结构化和随机的测试实例[18]，共 287 个。关于二元随机实例，我们选取随机 CSP 模型 Model RB 进行测试，总共选取 60 个二元随机实例进行测试。关于二元的结构化问题，选取问题类 BQWH，Driver 和 Job-Shop 进行测试[19]。

表 2 列出 AC3，maxRPC3，maxRPC3rm，lmaxRPC3 和 lmaxRPC3rm 算法在 60 个随机实例上和 227 个结构化实例上的测试结果，其中 t 是求解时间，n 是节点个数。从表 2 中可以看出，在 frb30-15 上，所有算法都在规定的时间内求出了所有实例的解，其中 lmaxRPC3rm 算法拥有最少的求解时间，并且节点数和约束检查次数都是最少的；在 frb40-19 上，同样是 lmaxRPC3rm 算法拥有最短的求解时间、最少的节点数和最少的约束检查次数；在 frb45-21 上，lmaxRPC3 算法表现最好。在 frb50-23 上表现最好的算法是 maxRPC3rm；frb53-24 上表现最好的算法是 maxRPC3；在 frb56-25 上表现最好的算法是 lmaxRPC3。从整体上来看，拥有最少总求解时间的算法是 lmaxRPC3，与其他算法相比，其拥有最少的节点数和约束检查次数。因此，在 60 个随机问题上表现最好的算法是 lmaxRPC3。

表 2　AC3，maxRPC3 及其变体算法在 287 个实例上的测试结果

问题	AC3(t)	AC3(n)	max RPC3(t)	max RPC3(n)	max RPC3rm(t)	max RPC3rm(n)	lmax RPC3(t)	lmax RPC3(n)	lmax RPC3rm(t)	lmax RPC3rm(n)
frb30-15(10)	1.29	5664.00	1.26	110.30	1.70	168.60	1.08	114.70	**0.85**	**108.11**
frb40-19(10)	104.50	355309.60	8.85	591.10	7.74	606.80	4.53	363.00	**3.49**	**360.89**
frb45-21(10)	1273.39	3697771.30	21.54	1311.90	12.80	837.60	**9.42**	**686.50**	17.31	1556.50
frb50-23(10)	1197.97	2935217.00	114.86	6694.80	**87.01**	**5341.20**	10.57	754.00	24.59	2014.50
frb53-24(10)	—	—	**26.15**	**1395.20**	63.78	3687.70	39.18	2620.50	38.30	3116.50
frb56-25(10)	—	—	105.55	5418.90	137.91	7989.60	**39.76**	**2475.30**	44.07	3397.20
bqwh-15-106(100)	2.99	4.06E+04	9.42	1.41E+03	10.80	2.13E+03	**9.31**	**1.55E+03**	11.42	2.50E+03
bqwh-18-141(100)	**88.99**	**1.01E+06**	310.37	2.85E+04	268.17	3.32E+04	206.87	2.07E+04	224.38	2.98E+04
driver(7)	**17.86**	**2.44E+04**	87.83	1.38E+03	80.58	1.77E+03	102.10	1.75E+03	540.43	2.62E+03
jobShop-e0ddr1(10)	49.12	8.17E+04	382.72	6.31E+03	5.38	9.39E+01	**6.46**	**8.63E+01**	11.22	2.24E+02
jobShop-e0ddr2(10)	0.55	1.23E+03	4.98	4.13E+01	5.64	7.61E+01	6.68	7.70E+01	**4.85**	**7.67E+01**

227 个结构化实例中，在 bqwh-15-106 实例集上，lmaxRPC3 与 maxRPC3 都在规定的时间内求出解，但是 lmaxRPC3 算法的求解时间最少，因此其表现最好。在 bqwh-18-141 上，AC3 算法在规定的时间内求解出了 86 个实例，比其他算法表现都好。在 Driver 上，同样是 AC3 算法表现最好。以上说明，虽然 lmaxRPC3rm 算法的剪枝能力强于 AC3，但是其在查找 PC 和 PC 证人时用了过多的约束检查，从而导致了算法的过度消耗，求解时间相对较长。在实例集 jobShop-e0ddr1 上，lmaxRPC3 求出了 10 个实例，并且拥有最短的求解时间、最少的节点个数及最少的约束检查次数，因此 lmaxRPC3 算法表现最好。在 jobShop-e0ddr2 上，maxRPC3 及其变体算法都求出了比 AC3 多 9 个的实例，其中 lmaxRPC3rm 所用的求解时间最短。在 227 个结构化实例上的整体表现最好的算法是 maxRPC3，因为其求出了最多实例的解。

结束语　约束求解与优化是人工智能中的一个重要领域，约束对于求解 NP 难问题有着很好的效果。本文对约束中的重要传播技术——相容性技术进行了探讨，总结了弧相容和最大受限路径相容的发展及各算法的特点，并通过对比实验来说明算法的改进效果。

到目前为止，AC 和 maxRPC3，lmaxRPC3 都是求解实例中效果较好的算法，值得引起广泛关注并被实际使用。受限路径相容是比最大受限路径相容的相容性稍弱的相容性算法，在应用中同样有着很好的效果。位操作的提出进一步提高了弧相容算法和最大受限路径相容算法的效率，且最新、最好的受限路径相容算法也在约束求解中取得了非常好的效果。下一步打算利用位操作来改进受限路径相容算法，以提高算法性能。

参 考 文 献

[1] FRANCESCA R. Handbook of constraint programming [M]. Netherlands: Elsevier, 2006: 15-18.

[2] GENT I, WALSH T. CSPlib: A Benchmark Library for Constraints[C]// Proceedings of CP-99. 1999.

[3] BESSIERE C, REGIN J C, YAP R H C, et al. An Optimal Coarse-grained Arc Consistency Algorithm [J]. Artificial Intelligence, 2005, 165(2): 165-185.

[4] MACKWORTH A. Consistency in Networks of Relations [J]. Artificial Intelligence, 1981, 8(1): 69-78.

[5] LICOUTRE C, HEMERY F. A study of residual supports in arc consistency[C]// International Joint Conference on Artifical Intelligence. Morgan Kaufmann Publishers Inc., 2007: 125-130.

[6] COHEN D, JEAVONS P, KOUBARAKIS M. Tractable disjunctive constraints[C]// International Conference on Principles and Practice of Constraint Programming. Springer Berlin Heidelberg, 1997: 478-490.

[7] 刘春晖. 基于约束传播的约束求解方法研究[D]. 长春：吉林大学，2008.

[8] BERLANDIER P. Improving domain filtering using restricted path consistency[C]// Artificial Intelligence for Applications. Los Angeles: CA, 1995: 32.

[9] DEBRUYNE R, BESSIERE C. From restricted path consistency to max-restricted path consistency[M]// Principles and Practice of Constraint Programming—CP97. Springer Berlin Heidelberg, 1997: 312-326.

[10] STERGIOU K. Restricted Path Consistency Revisited[M]// Principles and Practice of Constraint Programming. Springer International Publishing, 2015.

（下转第 62 页）