软件学报 ISSN 1000-9825, CODEN RUXUEW
*Journal of Software*,2016,27(11):2701−2711 [doi: 10.13328/j.cnki.jos.004874]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

# 负表约束的简单表缩减广泛弧相容算法*

李宏博[1,2], 梁艳春[1,2], 李占山[1,2]

[1](吉林大学 计算机科学与技术学院,吉林 长春 130012)

[2](符号计算与知识工程教育部重点实验室(吉林大学),吉林 长春 130012)

通讯作者: 李占山, E-mail: zslizsli@163.com

**摘 要**: 广泛弧相容算法(generalized arc consistency,简称 GAC),是求解约束满足问题的核心方法.表约束理论上可以表示所有约束关系,在过去 10 年中,有很多应用于表约束的广泛弧相容算法被提出来.在这些算法中,表缩减算法的效率非常高.但是目前的表缩减算法只能应用于正表约束,无法直接应用于负表约束.首先,提出一种表缩减算法 STR-N,可以直接应用于负表约束;然后,给出了 STR-N 的两个改进版本 STR-N2 和 STR-NIC.实验结果显示,STR-N 算法在负表约束上的求解效率具有明显的优势.

**关键词**: 约束满足问题;广泛弧相容;简单表缩减;负表约束

**中图法分类号**: TP181

中文引用格式: 李宏博,梁艳春,李占山.负表约束的简单表缩减广泛弧相容算法.软件学报,2016,27(11):2701−2711. http://www.jos.org.cn/1000-9825/4874.htm

英文引用格式: Li HB, Liang YC, Li ZS. Simple tabular reduction for generalized arc consistency on negative table constraints. Ruan Jian Xue Bao/Journal of Software, 2016,27(11):2701−2711 (in Chinese). http://www.jos.org.cn/1000-9825/4874.htm

## Simple Tabular Reduction for Generalized Arc Consistency on Negative Table Constraints

LI Hong-Bo[1,2], LIANG Yan-Chun[1,2], LI Zhan-Shan[1,2]

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

[2](Key Laboratory of Symbolic Computation and Knowledge Engineering for the Ministry of Education (Jilin University), Changchun 130012, China)

**Abstract**: Generalized arc consistency (GAC) plays a central role in solving the constraint satisfaction problem. Table constraints can theoretically represent all kinds of constraint relations, and many algorithms have been proposed to establish GAC on table constraints in the past decade. Among these methods, the simple tabular reduction algorithms (STR) are very efficient. However, the existing STR algorithms are suitable for only positive table constraints. They can not directly work on negative table constraints. In this paper, a STR algorithm, called STR-N, is first proposed to directly work on negative table constraints. Then, two improved versions of STR-N, STR-N2 and STR-NIC are presented. Experimental results show that the STR-N algorithms bring improvements over CPU time while solving the instances with negative table constraints.

**Key words**: constraint satisfaction problem; generalized arc consistency; simple tabular reduction; negative table constraint

约束满足问题[1]在人工智能领域有着广泛的应用,许多实际问题可以应用约束满足问题建模,如配置问题[2]、路径规划问题、任务调度问题等.一个约束满足问题(constraint satisfaction problem,简称 CSP)$P$ 由 3 部分组成:$P=\langle X,D,C\rangle$,其中,$X=\{x_1,x_2,\ldots,x_n\}$是一个有限的变量集合;$D=\{dom(x_1),dom(x_2),\ldots,dom(x_n)\}$,其中,$dom(x_i)$是变

* 基金项目: 国家自然科学基金(61472158, 61272207); 吉林省科技计划(20140101200JC)

Foundation item: National Natural Science Foundation of China (61472158, 61272207); Science & Technology Plan of Jilin Province (20140101200JC)

收稿时间: 2014-09-25; 修改时间: 2015-01-07; 采用时间: 2015-07-27

© 中国科学院软件研究所 http://www.jos.org.cn

量 $x_i$ 的离散有限值域;$C=\{c_1,c_2,\ldots,c_k\}$是一个有限约束集合,其中,任意 $c_j\in C$ 表示变量取值之间的制约关系.一个约束 $c$ 包含两个部分:一个有序的变量集合 $scp(c)=\{x_{i1},x_{i2},\ldots,x_{ir}\}$和这些变量有限值域的笛卡尔积 $dom(x_{i1})\times dom(x_{i2})\times\ldots\times dom(x_{ir})$的一个子集,这个子集中的元组表示满足(或者违反)这条约束的变量取值组合.约束有多种知识表示方式,例如,基于断言(predicate)的约束 $x_1+x_2<5$,当 $x_1=2,x_2=3$ 时不满足约束,而当 $x_1=2,x_2=1$ 时满足约束;基于特殊语义的,如表达“全部都不相等”关系的 $alldifferent(x,y,z)$约束[3],只有当 $x,y,z$ 三者中任意两个都不相等时才满足约束.除了这些隐式定义的约束以外,还有显式定义的约束,即,表约束.表约束[4]是显式地列出所有满足(或者违反)约束的元组,当其列出的元组满足约束时,我们称其为正表约束;当其列出的元组违反约束时,我们称其为负表约束.我们以著名的 $n$-皇后问题为例介绍表约束.用 CSP 为 $n$-皇后问题建模时,每个皇后是一个变量,每个皇后可放的 $n$ 个位置是每个变量的值域,任意两个皇后之间不能攻击的条件就是其中的约束关系.对于 4 皇后问题,表示第 1 个和第 2 个皇后之间不能互相攻击的约束 $c_{12}$ 如果用正表约束表示,则包含 6 个满足 $c_{12}$ 的元组 {(1,3),(1,4),(2,4),(3,1),(4,1),(4,2)};如果使用负表约束表示,则包含 10 个违反 $c_{12}$ 的元组 {(1,1),(1,2),(2,1),(2,2), (2,3),(3,2),(3,3),(3,4),(4,3),(4,4)},类似地,第 1 个和第 4 个皇后之间的约束 $c_{14}$ 如果用正表约束表示,则包含 10 个元组{(1,2),(1,3),(2,1),(2,3),(2,4),(3,1),(3,2),(3,4),(4,2),(4,3)};负表约束则包含 6 个元组 {(1,1),(1,4),(2,2),(3,3),(4,1),(4,4)}.显然,出于对空间的考虑,我们应该使用正表约束表示 $c_{12}$、负表约束表示 $c_{14}$.此外,在表约束上对某个元组进行约束检查时,要在所有元组上查找该元组:如果在正表约束中找到该元组,则该元组满足约束;否则不满足约束.在负表约束上结果则恰好相反.显然,元组个数直接影响约束检查的效率,因此,我们应该恰当地选择正表约束和负表约束.

寻找一个约束满足问题的解是 NP-难问题[1],一个高效的完备求解策略是在回溯搜索过程中使用局部相容技术[5]进行剪枝.应用于二元约束满足问题的弧相容算法[6]是在求解中应用最广的局部相容技术,在非二元约束满足问题上[4,7]的弧相容技术称为广泛弧相容.通用的广泛弧相容算法 GAC-valid[4]可以利用约束检查在所有类型的约束上实现广泛弧相容.利用了表约束的特殊知识表示形式后,一些正表约束上专用的广泛弧相容算法相继被提出来.GAC-allowed[4]算法遍历所有满足约束的元组,直到找到一个有效的元组.利用这种方式,为变量值寻找支持.为了更快地寻找支持,文献[8,9]分别提出了两种基于索引的技术加速遍历过程.文献[10]提出了一种方法,在寻找支持时交替访问有效元组和满足元组.基于压缩的方法首先将表约束转化成压缩表[11]或多元决策树[12],然后将专用的算法应用在其各自的数据结构上.除了以上方法以外,2007 年提出了一种表缩减算法 simple tabular reduction(STR)[13],它通过动态地维持约束表中有效的元组来识别那些有支持的值,在松紧度较紧的正表约束上使用时,效率明显高于其他算法.由于 STR 算法执行效率高,近几年一些基于 STR 算法的改进工作[14−19]也相继发表于高水平国际会议与期刊上.但是由于目前的 STR 算法是基于正表约束中记录的元组是满足约束的元组这一特征,对于负表约束中记录的违反约束的元组,STR 算法无法应用.如前面 4-皇后问题例子,正表约束和负表约束可以相互转化,但是 STR 算法高效是由于其应用于记录的元组个数较少的正表约束,而将负表约束转化为正表约束后,其记录的元组个数通常较多,除了对空间因素的考虑以外,这也将直接影响 STR 算法的效率.因此,将负表约束转化成正表约束后使用 STR 算法的做法是不明智的.我国学者近几年在约束程序相关领域的研究工作主要是生成难解问题模型[20,21]的相关理论研究、快速求解可满足性问题[22,23]以及最大可满足问题 max-CSP[24].

本文首先提出一种表缩减算法,称为 STR-Negative[25],简称 STR-N.它通过将约束上所有元组进行分类计数的方式,可以直接在负表约束上实现广泛弧相容.接着,我们提出 STR-N 算法的两种改进版本——STR-N2 和 STR-NIC.STR-N2 可以减少 STR-N 在进行有效性检查时的一些冗余操作;STR-NIC 使用一种遍历条件,可以避免一些冗余的遍历表缩减过程.我们的实验首先将正表约束和负表约束相互转化,然后分别使用 STR 算法和 STR-N 算法进行求解.结果发现,二者求解效率的拐点发生于约束松紧度在 0.466 附近,而不是预期的 0.5.然后,我们比较了 STR-N 算法和其他可应用于负表约束的广泛弧相容算法,包括通用算法 GAC-valid、通用算法在负表约束上的改进版 GAC-valid-forbidden 以及基于多元决策树压缩的方法在应用于负表约束时的变种 mddc.实验结果显示,STR-N 在一些随机问题上求解效率比其他方法更高.在一些标准库测试用例上,STR-NIC 的效率

© 中国科学院软件研究所 http://www.jos.org.cn

最高.

## 1 背景知识

给定一条约束 $c$,$scp(c)=\{x_{i1},x_{i2},\ldots,x_{ir}\}$表示 $c$ 中包含的变量集合,$|scp(c)|$称为 $c$ 的元数.对于 $c$ 上的一个元组 $\tau$,$\tau[x]$表示变量 $x$ 在$\tau$中的取值.我们使用 $t$ 表示约束 $c$ 中包含的违反 $c$ 的元组个数,则 $t/|dom(x_{i1})\times dom(x_{i2})\times\ldots\times dom(x_{ir})|$称为约束 $c$ 的约束松紧度(constraint tightness),即 $dom(x_{i1})\times dom(x_{i2})\times\ldots\times dom(x_{ir})$中违反约束 $c$ 的元组的比例,记为 $t(c)$.通常,当 $t(c)$大于 0.5 时,我们使用正表约束表示 $c$,因为此时满足 $c$ 的元组个数大于违反 $c$ 的元组个数;当 $t(c)$小于 0.5 时,我们使用负表约束表示 $c$,因为此时违反 $c$ 的元组个数大于满足 $c$ 的元组个数.我们称一个元组是有效的,当且仅当这个元组中的所有变量值都没有被从对应的变量值域中删除.判断一个元组是否有效的过程称为有效性检查,而判断一个元组是否满足一条约束的过程称为约束检查.给定一个约束满足问题 $P=\langle X,D,C\rangle$,如果 $C$ 中所有约束元数都小于等于 2,则称 $P$ 为二元约束满足问题;否则,称为非二元约束满足问题.一个约束满足问题的解是为每个变量在其值域中选择一个值构成集合 $S$,使得 $S$ 中的所有变量取值满足 $C$ 中所有约束.在本文中,$(x_i,a)$表示变量 $x_i$ 的取值为 $a$,$c_{ij}$ 表示变量 $x_i$ 和 $x_j$ 之间的约束.

**定义 1(广泛弧相容(generalized arc consistency(GAC)))**[4]. 给定一个约束满足问题 $P=\langle X,D,C\rangle$、一个约束 $c$ 和一个变量 $x\in scp(c)$:

- 一个值$(x,a)$与约束 $c$ 是相容的,当且仅当 $c$ 中存在一个有效的元组$\tau$,使得$\tau[x]=a$,并且$\tau$满足 $c$,此时,$\tau$称为$(x,a)$在约束 $c$ 上的支持.
- 一条约束 $c$ 是广泛弧相容的,当且仅当对于任意 $x\in scp(c)$,$dom(x)$不为空,并且对于任意 $a\in dom(x)$,$(x,a)$与约束 $c$ 是相容的.
- $P$ 是广泛弧相容的,当且仅当 $C$ 中所有约束都是广泛弧相容的.

从广泛弧相容定义可知,一个元组$\tau$称为一个值在一条约束上的支持要满足两个条件:(1) $\tau$是有效的;(2) $\tau$满足这条约束.为了实现广泛弧相容,广泛弧相容算法通常为每个变量的每个值在包含这个变量的所有约束上寻找一个支持,并将那些没有找到支持的值从对应的值域中删除.如果算法执行过程中某个变量的值域中的所有值都被删除,则广泛弧相容失败,问题无解.维持弧相容算法(maintaining arc consistency,简称 MAC)[26]是目前最流行的求解约束满足问题的方法,它在回溯搜索过程中构造一棵搜索树,并使用广泛弧相容算法进行剪枝.在搜索树的每个节点上,一个变量和这个变量值域中的一个值被选中,然后,MAC 使用 GAC 算法进行剪枝,将那些已经不再相容的值删除.如果 GAC 执行失败,则发生回溯.

## 2 正表约束上的表缩减算法

表缩减算法 STR[13]由 Ullmann 于 2007 年提出,其主要思想是:它遍历每条约束表中记录的所有元组,同时检查这些元组是否有效,如果发现一个元组是有效的,那么这个元组中包含的所有变量值在这条约束上就一定有支持.这是由于 STR 是应用于正表约束的,而正表约束中记录的元组都是满足约束的,因此,这个元组同时满足了成为支持的两个条件.使用 STR 实现广泛弧相容的算法流程如算法 1 所示,其中,$STR(c)$是在单个约束 $c$ 上实现广泛弧相容的算法,其伪代码如算法 2 所示,$STR(c)$返回的是在 $c$ 上进行表缩减时发现值域有改变的变量,如果 $STR(c)$发现有变量值域为空,则直接返回 $FAIL$.

**算法 1**. GAC-STR 算法.

```
1:  initialize the propagation queue Q;
2:  while Q is not empty do
3:      select and remove a constraint c from Q;
4:      reduced(X)←STR(c);
5:      if reduced(X)=FAIL then
6:          return FAIL;
```

© 中国科学院软件研究所 http://www.jos.org.cn

```
7:          for each x∈reduced(X) do
8:              for each constraint c′ involving x do
9:                  if c′≠c then
10:                     Q←Q∪{c′}
11:   return success
```

**算法 2**. 正表约束的 STR 算法.

Input: A positive table constraint $c$.

Output: The set of variables whose domain have been reduced.

```
1:    for each x∈scp(c) and x is not assigned do
2:        gacValues(x)←∅;
3:    index←1;
4:    while index≤lastIndex(c) do
5:        τ←table(c)[position(c)[index]];
6:        if τ is valid then
7:            for each (x,a)∈τ and x is not assigned do
8:                gacValues(x)←gacValues(x)∪{x};
9:            index←index+1;
10:       else
11:           removeTuple(τ,index,c);
12:   reduced(X)←∅;
13:   for each x∈scp(c) and x is not assigned do
14:       if |gacValues(x)|<|dom(x)| then
15:           dom(x)←gacValues(x);
16:           if dom(x)=∅ then
17:               return FAIL;
18:           reduced(X)←reduced(X)∪{x};
19:   return reduced(X)
```

由于 STR 算法要遍历表中记录的所有元组,为了配合回溯搜索使用,STR 算法还给出了一种机制,使得在上一层搜索节点就已经失效的元组在下一层不会被重新检查.当前还未被检查失效的元组称为当前元组,STR 在遍历表的过程中只遍历当前元组.请注意,一个当前元组并不一定是有效的,它只是还未被检查是有效的还是无效的.这一机制主要避免了检查那些已经确定失效了的元组.这一机制使用如下数据结构[14]来实现.

- *table*(*c*):*table*(*c*)一个静态数组,其中,静态记录了满足 *c* 的所有元组,数组下标从 1 到元组个数 *table*(*c*).*length*,*table*(*c*)生成之后,其中的元组位置不再变化.
- *position*(*c*):*position*(*c*)是一个长度为 *table*(*c*).*length* 的整数数组,初始时,其元素取值分别是 1 到 *table*(*c*).*length*.它提供的是对 *table*(*c*)中元组的间接访问.使用 *position*(*c*)后,*c* 的表中的第 *i* 个元组是 *table*(*c*)[*position*(*c*)[*i*]].
- *lastIndex*(*c*):*lastIndex*(*c*)是一个整数,它指向 *position*(*c*)的某个下标,这个下标是 *position*(*c*)中最后一个有效的元素.也就是说,*position*(*c*)中所有 *lastIndex*(*c*)之前的元素所对应的 *table*(*c*)中的元组是当前元组,*lastIndex*(*c*)之后所对应的所有元组都是无效的.

这 3 个数据结构的关系及其如何表示当前元组如图 1 所示.

© 中国科学院软件研究所 http://www.jos.org.cn

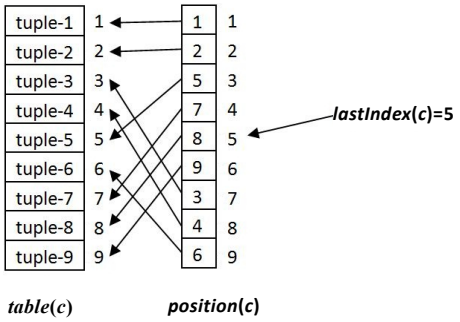


Fig.1 Example of current tuples

图 1 当前元组举例

图 1 中,*table*(*c*)总共记录了 9 个元组,*lastIndex*(*c*)为 5,则 *position*(*c*)中第 6 个元素之前都对应 *table*(*c*)中一个当前元组,此时的当前元组分别为 tuple-1,tuple-2,tuple-5,tuple-7,tuple-8.

算法 2 中,*removeTuple*(*τ*,*index*,*c*)的作用是将 *τ* 标示成已经无效,该函数是将 *position*(*c*)[*index*]和 *position*(*c*)[*lastindex*(*c*)]互换,然后将 *lastIndex*(*c*)减 1,之后,原本 *position*(*c*)[*index*]指向的 *table*(*c*)中的元组就被标示成已经无效,因为只有 *lastIndex*(*c*)之前的元组才是当前元组.在回溯搜索过程中,STR 要在每层的搜索树节点上为每条约束记录 *lastIndex*(*c*),当发生回溯时,要根据在对应层的记录恢复 *lastIndex*(*c*).

## 3 负表约束上的表缩减算法

在实现广泛弧相容时,我们并不一定要找到每个变量值的支持到底是哪个元组,只需证明每个值都存在一个支持就可以了.给定一条约束 *c*、变量 *x*∈*scp*(*c*)、值 *a*∈*dom*(*x*),对于 *c* 上包含(*x*,*a*)的所有元组(即,除 *x* 外,*scp*(*c*)中其他所有变量值域的笛卡尔积),我们根据需要,将这些元组分为 3 类.首先,我们根据有效性将这些元组分为“有效的”和“无效的”;然后,对于所有“有效的”元组,我们可以根据是否满足 *c* 将其分为“有效并且满足 *c* 的”和“有效但是违反 *c* 的”.显然,任意一个“有效并且满足 *c* 的”的元组都是(*x*,*a*)在 *c* 上的支持,因此,我们要做的就是证明“有效并且满足 *c* 的”元组个数大于 0.我们通过如下 3 步证明“有效并且满足 *c* 的”元组存在.

- 第 1 步,我们要计算 *c* 上包含(*x*,*a*)的所有“有效的”元组的个数,记为 *valid*(*x*,*a*,*c*).这个个数就是除 *x* 外,*scp*(*c*)中其他所有变量当前值域的笛卡尔积结果集的大小,即,除 *x* 外,*scp*(*c*)中其他所有变量当前值域大小的乘积.我们可以在 *O*(*r*)时间内计算得到这个数字,其中,*r* 是 *c* 的元数.对于 *dom*(*x*)中的任意值 *a*,*valid*(*x*,*a*,*c*)都是相同的,因此,我们将 *valid*(*x*,*a*,*c*)记为 *valid*(*x*,*c*).
- 第 2 步,我们要计算 *c* 上包含(*x*,*a*)的所有“有效但是违反 *c* 的”元组的个数,记为 *valid-forbidden*(*x*,*a*,*c*).对于负表约束,其记录的所有元组都是违反 *c* 的,因此,*valid-forbidden*(*x*,*a*,*c*)就是(*x*,*a*)出现在 *table*(*c*)中有效元组的次数.因此,我们可以通过遍历 *table*(*c*)中当前元组并检查其有效性这一过程来记录(*x*,*a*)在 *table*(*c*)的有效元组中出现的次数.这一遍历过程可以同时得到 *scp*(*c*)中所有变量的所有值在 *table*(*c*)的有效元组中出现的次数.
- 第 3 步,“有效并且满足 *c* 的”元组个数 *valid-allowed*(*x*,*a*,*c*)就是“有效的”元组个数减去“有效但是违反 *c* 的”元组个数.如果 *valid-allowed*(*x*,*a*,*c*)大于 0,那么(*x*,*a*)在 *c* 上一定存在支持.

基于以上 3 个步骤,我们在算法 3 中给出 STR-Negative 算法的伪代码.

**算法 3**. 负表约束的 STR 算法.

Input: A negative table constraint *c*.

Output: The set of variables whose domain have been reduced.

```
1:    if lastIndex(c)=0 then
2:          return ∅;
```

© 中国科学院软件研究所 http://www.jos.org.cn

```
3:    for each x∈scp(c) and x is not assigned do
4:        compute valid(x,c) for x;
5:        for each value a∈dom(x) do
6:            valid-disallowed(x,a,c)←0;
7:    index←1;
8:    while index≤lastIndex(c) do
9:        τ←table(c)[position(c)[index]];
10:       if τ is valid then
11:           for each (x,a)∈τ and x is not assigned do
12:               valid-disallowed(x,a,c)←valid-disallowed(x,a,c)+1;
13:           index←index+1;
14:       else
15:           removeTuple(τ,index,c);
16:   reduced(X)←∅;
17:   for each x∈scp(c) and x is not assigned do
18:       for each value a∈dom(x) do
19:           if valid(x,c)−valid-disallowed(x,a,c)=0 then
20:               remove (x,a) from dom(x);
21:               if dom(x)=∅ then
22:                   return FAIL;
23:               reduced(X)←reduced(X)∪{x};
34:   return reduced(X);
```

算法 3 中第 1 行、第 2 行,当 *lastIndex*(*c*)为 0 时,说明约束 *c* 中当前有效的元组都是满足 *c* 的元组,所以避免了后面的遍历过程.

**命题 1**. 算法 *STR-Negative*(*c*)的最坏时间复杂度为 *O*(*rd*+*rt*),最坏空间复杂度为 *O*(*rd*+*rt*),其中,*r* 是 *c* 的元数,*d* 是 *scp*(*c*)中变量的最大值域,*t* 是 *table*(*c*)中记录的元组个数.

证明:算法 3 中,第 3 行~第 6 行的循环耗时 *O*(*rd*);第 8 行~第 15 行的循环耗时 *O*(*rt*),因为这一循环最多可能访问 *t* 个元组,检查每个元组是否有效耗时为 *O*(*r*);第 17 行~第 23 行的循环耗时为 *O*(*rd*).因此,算法 *STR-Negative*(*c*)的最坏时间复杂度为 *O*(*rd*+*rt*).算法 *STR-Negative*(*c*)使用的数据结构中,*table*(*c*)占用 *rt* 的空间,因为共有 *t* 个元组,每个元组占用空间为 *r*.*position*(*c*)占用空间为 *t*.*r* 个变量的 *d* 个值的 *valid*(*x*,*c*),*valid-allowed*(*x*,*a*,*c*),*valid-forbidden*(*x*,*a*,*c*)总共占用空间 *O*(*rd*).因此,S*TR-Negative*(*c*)的最坏空间复杂度为 *O*(*rd*+*rt*). □

## 4 优化 STR-Negative 算法

这一节我们将描述 STR-N 算法的两种改进版本——STR-Negative2(STR-N2)和 STR-N-Iteration-Condition (STR-NIC).

### 4.1 STR-Negative2:避免冗余的有效性检查操作

给定一条约束 *c*,当前我们要在 *c* 上执行 *STR-N*(*c*),变量 *x*∈*scp*(*c*),如果当前 *dom*(*x*)和上次在约束 *c* 上调用 *STR-N*(*c*)时的 *dom*(*x*)相同,那么 *dom*(*x*)中的所有值在这中间都没有被删除,因此在进行有效性检查时,可以避免检查 *x* 的值.基于此,我们提出 STR-Negative2 算法.下面是 STR-N2 使用的两种额外的数据结构[14].

- *need-check*(*c*):*need-check*(*c*)是 *scp*(*c*)的一个子集,记录那些在进行有效性检查时需要检查的变量.
- *lastSize*(*c*):*lastSize*(*c*)是一个长度为|*scp*(*c*)|的整数数组,*lastSize*(*c*)[*x*]记录了上次调用 *STR-N*2(*c*)时,变量 *x* 的值域大小,如果本次调用开始前 *x* 的值域大小和 *lastSize*(*c*)[*x*]不相同,那么将 *x* 加入 *need-check*(*c*).

© 中国科学院软件研究所 http://www.jos.org.cn

所有的 *lastSize*(*c*)[*x*]初始时为−1.

与 STR-N 算法相比,STR-N2 算法只需在遍历当前元组前根据上次记录的 *lastSize*(*c*)计算 *need-check*(*c*),然后更新 *lastSize*(*c*),之后,在检查元组有效性时,只检查 *need-check*(*c*)中变量.在回溯搜索中使用时,STR-N2 要在每层的搜索树节点上为每条约束记录 *lastSize*(*c*),当发生回溯时,要根据在对应层的记录恢复 *lastSize*(*c*).

### 4.2 STR-N-Iteration-Condition:避免冗余的遍历元组

显然,对于 STR-N 算法,最耗时的部分是在第 8 行~第 15 行的循环中遍历访问当前元组.我们提出一种遍历条件 iteration condition(IC),来避免那些冗余的遍历元组过程.满足 IC 可以在不遍历访问元组的条件下保证每个变量值都有支持.

**性质 1(遍历条件(iteration condition(IC)))**. 给定一个负表约束 *c*,*valid*-min 是 *scp*(*c*)中所有变量的 *valid*(*x*,*c*)的最小值.当 *valid*-min 大于 *lastIndex*(*c*)时,*scp*(*c*)中所有变量的所有值在 *c* 上一定存在支持.

IC 是正确的,因为 *lastIndex*(*c*)是当前 *c* 中所有“有效但是违反 *c* 的”元组个数(注意:不是某个值的“有效但是违反 *c* 的”,而是所有变量所有值的“有效但是违反 *c* 的”),而 *valid*-min 是最小的 *valid*(*x*,*c*),因此当 *valid*-min 大于 *lastIndex*(*c*)时,所有变量的所有值的 *valid*(*x*,*c*)都一定大于它们的 *valid-forbidden*(*x*,*a*,*c*).

IC 可以在 STR-N 算法开始遍历元组前使用(算法 3 的第 7 行以前),使用了 IC 的 STR-N 算法我们称为 STR-NIC 算法.如果满足 IC,则 STR-NIC 算法直接返回空集,即,没有变量的值域改变了.

## 5 实验结果

我们的测试环境为 Intel(R) Core(TM) i7-3770 CPU@ 3.40GHz,4.00GB RAM,Windows 7,JDK1.7.本文使用的所有 benchmark 测试用例以及随机问题模型生成器 RBGenerator 均可从 http://www.cril.univ-artois.fr/~lecoutre/下载.我们测试了在 MAC 中使用不同的广泛弧相容算法的求解效率,其中,变量启发式使用了著名的 Dom/Ddeg[27,28].

### 5.1 STR对比STR-N

STR 算法和 STR-N 算法分别应用于正表约束和负表约束,而两种约束可以相互转化,我们选择两种算法的依据通常是根据约束松紧度,只有当约束松紧度为 0.5 时,正表约束和负表约束记录的元组个数相同.因此,我们需要了解两种算法在不同约束松紧度时求解效率.对于同一条约束,无论约束松紧度是多少,我们强制将其转化成 STR 或者 STR-N 可以应用的形式,之后比较二者的求解效率.我们采用了随机问题模型 Model RB[20],一个 Model RB 可以根据以下 5 个参数⟨*r*,*n*,*d*,*e*,*t*⟩生成,其中,*r* 是约束元数,*n* 是变量个数,*d* 是变量值域大小,*e* 是约束个数,*t* 是约束松紧度.对于任意一组模型,我们测试了 30 个随机用例并取其平均值进行比较.我们使用 RBGenerator 来生成不同约束松紧度的测试用例.对于正表约束,我们分别实现了 STR[13],STR2[14],STR3[15]这 3 种算法;对于负表约束,我们分别实现了 STR-N,STR-N2,STR-NIC 这 3 种算法.之后,我们分别选择两组中求解效率最高的进行了对比.图 2 展示了在不同的约束松紧度下,最优的 STR 算法求解耗时除以最优的 STR-N 算法的求解耗时.我们的原始数据显示,正表约束上效率最高的是 STR2 算法,而负表约束上效率最高的是 STR-N2 算法.当 *y* 轴值大于 1 时,使用负表约束的 STR-N 的效率更高;当 *y* 轴值小于 1 时,使用正表约束的 STR 算法效率更高.

图 2 中的数据显示,二者效率比较的拐点在约束松紧度为 0.466 左右时发生,而不是我们预期的 0.5 左右.这是由于 STR-N 算法在最后识别那些失去支持的值的时间复杂度为 $O(rd)$,而 STR 算法在这部分时间复杂度为 $O(r)$.对于同样约束松紧度的参数,我们还比较了约束元数对求解效率的影响.从图中可以看出,在相同的约束松紧度条件下,随着元数的增长,有优势的一方优势在逐渐增大.例如在约束松紧度 0.216 时,随着元数从 3 增到 10,STR 的耗时从 STR-N 的 1.69 倍增长到了 2.32 倍.

© 中国科学院软件研究所 http://www.jos.org.cn

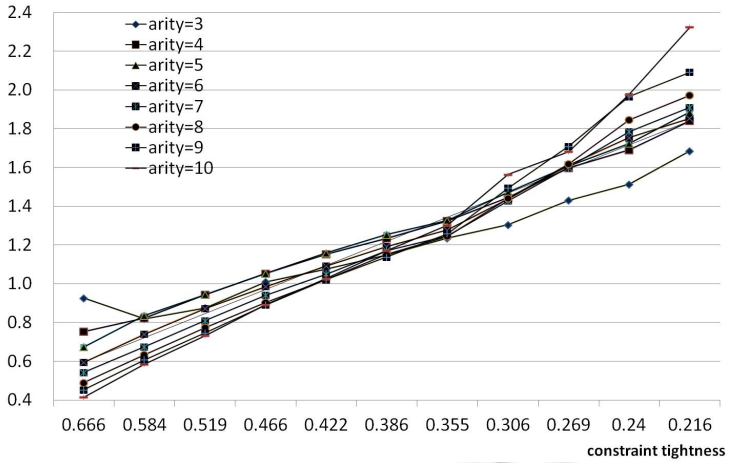


Fig.2 Result of STR vs. STR-N

图 2 STR 算法与 STR-N 算法比较结果

### 5.2 STR-N对比其他广泛弧相容算法

这里,我们比较了本文提出的 3 种 STR-N 算法和其他可应用于负表约束的广泛弧相容算法,包括通用算法 GAC-valid[4]、通用算法在负表约束上的变种 GAC-valid-forbidden[29](GAC-vf)和基于多元决策树压缩算的方法在负表约束上的变种 mddc[12].其中,所有的 GAC-valid 算法使用了“多向残余支持[30]”技术和避免冗余修正检查技术[31].

图 3 是在 15 组不同参数的随机问题的上的测试结果.为了更为直观,我们将每组数据以 GAC-valid 的求解耗时为基准,其他算法的耗时分别除以基准耗时,所得比值展示在图 3 中.

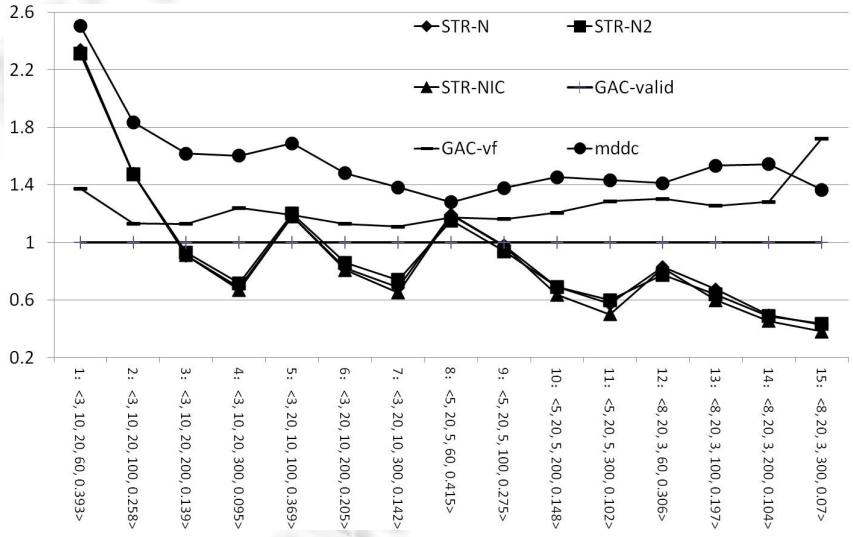


Fig.3 Results of STR-N vs. other GAC algorithms on random instances

图 3 STR-N 与其他广泛弧相容算法随机问题比较结果

从第 12 组~第 15 组可知,当元数较大并且约束个数较少时,STR-N2 效率最高;当约束松紧度特别低时,所有的 STR-N 算法效率都高于其他 3 种算法,其中,STR-NIC 的效率最高.从第 1 组 1、第 2 组、第 5 组、第 8 组可知:当约束松紧度相对较高时,GAC-valid 的效率也相对较高.但是从第 12 组可以看出,当元数较大时,仍然是

© 中国科学院软件研究所 http://www.jos.org.cn

STR-N 算法效率较高.因此,在约束松紧度较低并且元数较大的随机问题上,应用 STR-N 算法的效果最好.

接着,我们测试了可以使用负表约束的 3 组标准库测试用例,包括 ChessboardColoration(cc),Ramsey 和 Golomb Ruler.

- ChessboardColoration 问题是在一个包含 *r* 行 *c* 列的棋盘上使用 *n* 个颜色为每个方格着色,要求任意一个矩形的 4 个角落的方格颜色不能完全相同,每个用例使用 *cc-r-c-n* 表示.
- Ramsey 问题是使用 *k* 个颜色为一个完全图的边着色,使得图中没有单色的三角形.即,任意一个三角形的 3 个边颜色不能相同,每个用例使用 *ramsey-n-k* 表示.
- Golomb Ruler 问题是在长度为 *m* 的尺子上放置 *n* 个标记,使得任意两个标记之间的距离都不相同,每个用例使用 *ruler-m-n-a*4 表示.

图 4 中展示了这些测试用例上求解耗时的比较结果.

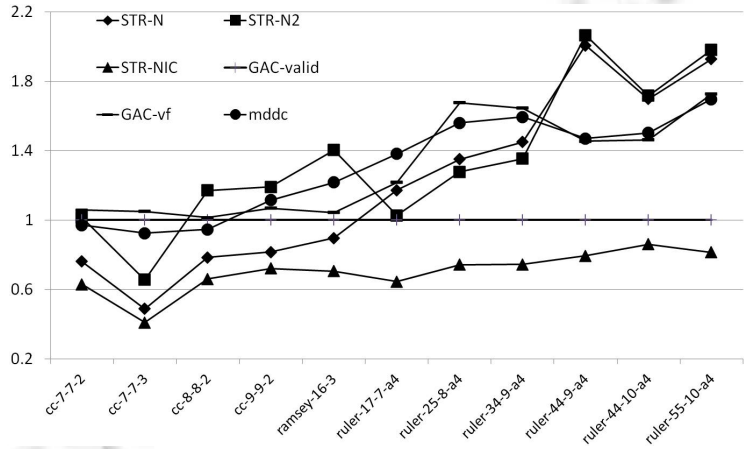


Fig.4 Results of time cost on benchmark instances

图 4 标准库测试用例耗时比较结果

这 3 类标准库测试用例,其中一些测试用例无法在 1 200s 内求解.由于使用了相同的变量启发式 Dom/Ddeg,所有的求解算法都将构造一颗相同的搜索树,也就是效率较高的算法在相同时间内会访问更多的搜索树节点(node),因此对于这些用例,我们比较了它们在 1 200s 内的搜索树节点个数,并将结果展示在图 5 中.

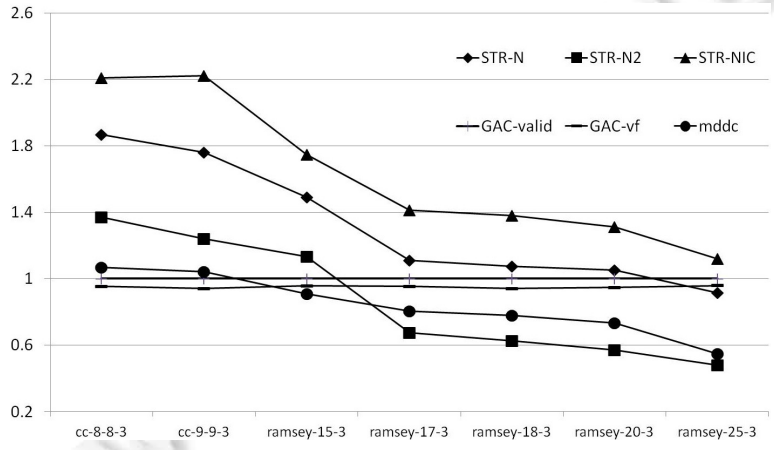


Fig.5 Results of search tree nodes on benchmark instances

图 5 标准库测试用例耗时搜索树节点比较结果

© 中国科学院软件研究所 http://www.jos.org.cn

图 4、图 5 中的数据与图 3 中的数据类似,都是以 GAC-valid 的结果为基准,各种算法在图中展示的结果都是和基准结果的比值.由图中可以看出,在 ChessboardColoration 问题和 Ramsey 问题上,STR-N 算法效率高于其他算法;而在 GolombRuler 问题上,只有 STR-NIC 算法的效率高于其他算法.值得一提的是,STR-NIC 的效率在所有这 3 组标准库测试用例上都高于其他 3 种算法.

## 6 结论与展望

表缩减算法 STR 在正表约束上求解效率非常高,但是目前已有的 STR 算法只能应用于正表约束.本文提出一种可应用于负表约束的 STR-N 算法,同时给出 STR-N 算法的两种改进版本——STR-N2 和 STR-NIC.我们的实验结果显示,当约束松紧度小于 0.466 时,STR-N 算法求解效率高于 STR;当约束松紧度大于 0.466 时, STR 的求解效率较高.与其他应用于负表约束的算法相比,在那些约束松紧度较松的约束上,STR-N 算法的效率较高.

目前,在正表约束 STR 算法的基础上,已经有国外研究学者提出了对应的高阶相容算法[16,17],另外还有基于压缩表的改进工作[19]和基于短支持的改进工作[18].正负表约束上的 STR 算法是这些改进工作的基础,而在负表约束上的高阶相容技术目前仍是空白.因此,我们下一步的工作打算在负表 STR 算法基础上,设计对应的高阶相容算法,并使用压缩表和短支持的技术改进负表约束 STR 算法.

**References**:

[1] Freuder EC, Mackworth AK. Constraint satisfaction: An emerging paradigm. In: Rossi F, van Beek P, Walsh T, eds. Handbook of Constraint Programming. Amsterdam: Elsevier, 2006. 13−28. [doi: 10.1016/S1574-6526(06)80006-4]

[2] Li H, Shen H, Li Z, Guo J. Reducing consistency checks in generating corrective explanations for interactive constraint satisfaction. Knowledge-Based Systems, 2013,43(5):103−111. [doi: 10.1016/j.knosys.2013.01.024]

[3] Régin JC. A filtering algorithm for constraints of difference in CSPs. In: Barbara H, Richard E, eds. Proc. of the AAAI'94. Seattle: AAAI Press, 1994. 362−367.

[4] Bessière C, Régin JC. Arc consistency for general constraint networks: Preliminary results. In: Pollack E, ed. Proc. of the IJCAI'97. Nagoya: Morgan Kaufmann Publishers, 1997. 398−404. [doi: 10.1109/WI.2007.95]

[5] Bessière C. Constraint Propagation. Rossi F, van Beek P, Walsh T, eds. Handbook of Constraint Programming. Amsterdam: Elsevier, 2006. 29−84. [doi: 10.1016/S1574-6526(06)80007-6]

[6] Mackworth AK. Consistency in networks of relations. Artificial Intelligence, 1977,8(1):99−118. [doi: 10.1016/0004-3702(77)90007-8]

[7] Sun JG, Jing SY. Solving non-binary constraint satisfaction problem. Chinese Journal of Computers, 2003,26(12):1746−1752 (in Chinese with English abstract). [doi: 10.3321/j.issn:0254-4164.2003.12.020]

[8] Lhomme O, Régin JC. A fast arc consistency algorithm for *n*-ary constraints. In: Manuela M, Subbarao K, eds. Proc. of the AAAI 2005. Pittsburgh: AAAI Press, 2005. 405−410.

[9] Gent IP, Jefferson C, Miguel I, Nightingale P. Data structures for generalized arc consistency for extensional constraints. In: Proc. of the AAAI 2007. Vancouver: AAAI Press, 2007. 191−197.

[10] Lecoutre C, Szymanek R. Generalized arc consistency for positive table constraints. In: Benhamou F, ed. Proc. of the CP 2006. Nantes: Springer-Verlag, 2006. 284−298. [doi: 10.1007/11889205_22]

[11] Katsirelos G, Walsh T. A compression algorithm for large arity extensional constraints. In: Bessière C, ed. Proc. of the CP 2007. Providence, 2007. 379−393.

[12] Cheng KC, Yap RHC. An MDD-based generalized arc consistency algorithm for positive and negative table constraints and some global constraints. Constraints, 2010,15(2):265−304. [doi: 10.1007/s10601-009-9087-y]

[13] Ullmann JR. Partition search for non-binary constraint satisfaction. Information Science, 2007,177(18):3639−3678. [doi: 10.1016/j.ins.2007.03.030]

[14] Lecoutre C. STR2: Optimized simple tabular reduction for table constraints. Constraints, 2011,16(4):341−371. [doi: 10.1007/s10601-011-9107-6]

[15] Lecoutre C, Likitvivatanavong C, Yap C. A path-optimal GAC algorithm for table constraints. In: De Raedt L, *et al*., eds. Proc. of the ECAI 2012. Montpellier: IOS Press, 2012. 510−515. [doi: 10.3233/978-1-61499-098-7-510]

[16] Paparrizou A, Stergiou K. An efficient higher-order consistency algorithm for table constraints. In: Hoffmann J, Selman B, eds. Proc. of the AAAI 2012. Toronto: AAAI Press, 2012. 535−541.

© 中国科学院软件研究所 http://www.jos.org.cn

[17] Lecoutre C, Paparrizou A, Stergiou K. Extending STR to a higher-order consistency. In: Jardins M, Littman M, eds. Proc. of the AAAI 2013. Bellevue: AAAI Press, 2013. 576−582.

[18] Jefferson C, Nightingale P. Extending simple tabular reduction with short supports. In: Rossi F, ed. Proc. of the IJCAI 2013. Beijing: AAAI Press, 2013. 573−579.

[19] Xia W, Yap RHC. Optimizing STR algorithms with tuple compression. In: Schulte C, ed. Proc. of the CP 2013. Uppsala: Springer-Verlag, 2013. 724−732. [doi: 10.1007/978-3-642-40627-0_53]

[20] Xu K, Li W. Exact phase transitions in random constraint satisfaction problems. Journal of Artificial Intelligence Research, 2000, 12(1):93−103.

[21] Gao J, Wang J, Yin M. Experimental analyses on phase transitions in compiling satisfiability problems. Science China Information Sciences, 2014,58(3):1−11. [doi: 10.1007/s11432-014-5154-0]

[22] Cai S, Su K. Comprehensive score: Towards efficient local search for SAT with long clauses. In: Proc. of the IJCAI 2013. Beijing: AAAI Press, 2013. 489−495.

[23] Cai S, Su K. Local search for boolean satisfiability with configuration checking and subscore. Artificial Intelligence, 2013,204(9): 75−98. [doi: 10.1016/j.artint.2013.09.001]

[24] Huang P, Yin M. An upper (lower) bound for max (min) CSP. Science China Information Sciences, 2014,57(7):1−9. [doi: 10.1007/s11432-013-5052-x]

[25] Li H, Liang Y, Guo J, Li Z. Making simple tabular reduction works on negative table constraints. In: Jardins M, Littman M, eds. Proc. of the AAAI 2013. Bellevue: AAAI Press, 2013. 1629−1630.

[26] Sabin D, Freuder EC. Contradicting conventional wisdom in constraint satisfaction. In: Cohn AG, ed. Proc. of the ECAI'94. London: John Wiley and Sons, 1994. 125−129. [doi: 10.1007/3-540-58601-6_86]

[27] Bessière C, Régin JC. MAC and combined heuristics: Two reasons to forsake FC (and CBJ?) on hard problems. In: Freuder EC, ed. Proc. of the CP'96. Cambridge: Springer-Verlag, 1996. 61−75. [doi: 10.1007/3-540-61551-2_66]

[28] Smith BM, Grant SA. Trying harder to fail first. In: Prade H, ed. Proc. of the ECAI'98. Brighton: John Wiley and Sons, 1998. 249−253.

[29] Lecoutre C. Generalized arc consistency for table constraints. In: Constraint Network: Techniques and Algorithms. ISTE/Wiley, 2009. [doi: 10.1002/9780470611821.ch5]

[30] Lecoutre C, Hemery F. A study of residual supports in arc consistency. In: Sangal R, Mehta H, Bagga RK, eds. Proc. of the IJCAI 2007. San Francisco: AAAI Press, 2007. 125−130.

[31] Li H., Li Z, Wang T. Improving coarse-grained arc consistency algorithms in solving constraint satisfaction problems. Ruan Jian Xue Bao/Journal of Software, 2012,23(7):1816−1823 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4129.htm [doi: 10.3724/SP.J.1001.2012.04129]

**附中文参考文献**:

[7] 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. [doi: 10.3321/j.issn:0254-4164.2003.12.020]

[31] 李宏博,李占山,王涛.改进求解约束满足问题粗粒度弧相容算法.软件学报,2012,23(7):1816−1823. http://www.jos.org.cn/1000-9825/4129.htm [doi: 10.3724/SP.J.1001.2012.04129]

**李宏博**(1985－),男,吉林公主岭人,博士,主要研究领域为约束求解与约束传播.

**李占山**(1966－),男,博士,教授,博士生导师,CCF 专业会员,主要研究领域为约束程序,自动推理,智能规划,基于模型的诊断.

**梁艳春**(1953－),男,博士,教授,博士生导师,CCF 专业会员,主要研究领域为机器学习,智能计算.

© 中国科学院软件研究所 http://www.jos.org.cn