软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(11):3530−3540 [doi: 10.13328/j.cnki.jos.006094]
©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

优化简单表缩减算法求解因子分解编码实例*

王 震[1,2], 李 哲[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: 表约束在约束程序(constraint programming,简称 CP)中被广泛研究.目前,求解表约束问题效率最高的算法是 CT(compact-table)和 STRbit(simple tabular reduction bit).它们在搜索过程中维持广义弧相容(generalized arc consistency,简称 GAC).完全成对相容(full pairwise consistency,简称 fPWC)是一种强于 GAC 的相容性关系,目前,实现 fPWC 效率最高的算法是 PW-CT,但是它无法直接在通用的求解器上实现.因子分解编码(factor-decomposition encoding,简称 FDE)是实现 fPWC 的一种编码方式,通常和简单表缩减(STR)算法一起来使用.当前效率最高的 STR 算法使用了 bitset 的数据结构,用这些算法来求解 FDE 实例可能会造成内存溢出.提出了 STRFDE 算法——一种使用 bitset 结构来求解 FDE 实例的方法.它结合了 CT 和 STRbit 的优势,在保证求解效率的同时,使占用的内存尽可能小.实验结果表明,在许多存在非平凡相交的实例上,该算法是有竞争力的.

关键词: 约束程序;弧相容;表约束;简单表缩减算法;因子分解编码

中图法分类号: TP18

中文引用格式: 王震,李哲,李占山.优化简单表缩减算法求解因子分解编码实例.软件学报,2021,32(11):3530−3540. http://www.jos.org.cn/1000-9825/6094.htm

英文引用格式: Wang Z, Li Z, Li ZS. Optimizing simple tabular reduction algorithm for factor-decomposition encoding instances. Ruan Jian Xue Bao/Journal of Software, 2021,32(11):3530−3540 (in Chinese). http://www.jos.org.cn/1000-9825/6094.htm

Optimizing Simple Tabular Reduction Algorithm for Factor-decomposition Encoding Instances

WANG Zhen[1,2], LI Zhe[1,2], LI Zhan-Shan[1,2]

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

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

Abstract: Table constraints are widely studied in constraint programming (CP). At present, the most efficient algorithms for solving table constraint problems are compact-table (CT) and simple tabular reduction bit (STRbit), which maintain generalized arc consistency (GAC) during the search process. Full pairwise consistency (fPWC) is a consistency that is stronger than GAC. The most efficient algorithm for maintaining fPWC is PW-CT which is difficult to implement in general solver. Factor-decomposition encoding (FDE) is an encoding method that implements fPWC and is usually used together with STR. The current STR algorithms that use bitset may cause memory overflow issues to solve FDE instances. This study proposes STRFDE, a new algorithm using bitset for solving FDE instances. It

* 基金项目: 国家自然科学基金(61802056); 吉林省自然科学基金(20180101043JC); 吉林省发展和改革委员会产业技术研究与开发项目(2019C053-9); 中国科学院太空应用重点实验室开放基金(LSU-KFJJ-2019-08)

Foundation item: National Natural Science Foundation of China (61802056); Natural Science Foundation of Jilin Province (20180101043JC); Industrial Technology Research and Development Project of Jilin Development and Reform Commission (2019C053-9); Open Research Fund of Key Laboratory of Space Utilization, Chinese Academy of Sciences (LSU-KFJJ-2019-08)

收稿时间: 2020-01-04; 修改时间: 2020-05-01; 采用时间: 2020-06-01

© 中国科学院软件研究所 http://www.jos.org.cn

combines the advantages of CT and STRbit that makes the memory as little as possible while ensuring the efficiency of solving. Experimental results show that the proposed algorithm is competitive over a variety of instances with non-trivial intersections.

Key words: constraint programming; arc consistency; table constraint; simple tabular reduction; factor-decomposition encoding

约束程序(constraint programming,简称 CP)是人工智能领域的研究方向之一,主要用于解决组合搜索问题.随着研究的深入,CP 被成功地应用在很多领域,例如时间规划、调度、配置等.表约束是 CP 中一种被广泛研究的约束类型,它将约束用元组明确地定义出来.广义弧相容(generalized arc consistency,简称 GAC)是求解表约束时使用的主要技术,GAC 算法通过识别并删除无效值来加快求解速度.回溯搜索算法是求解表约束的完备算法之一,在回溯搜索树的每一个结点执行 GAC 算法来判断当前状态是否满足 GAC.当前,维持表约束 GAC 的经典算法有简单表缩减(simple tabular reduction,简称 STR)算法[1]:STR2[2]、STR3[3]、STRbit[4]、CT[5]、STR-N[6]、AdaptiveSTR[7]、STR2*[8]和多值决策图(multi-valued decision diagrams,简称 MDD)算法:MDDc[9]、MDD4[10]、Compact-MDD[11].

在 GAC 被广泛使用的同时,高阶相容性也引起了人们的重视,例如关系相容、最大受限成对相容、成对相容(pairwise consistency,简称 PWC)[12]和完全成对相容(full pairwise consistency,简称 fPWC).fPWC 是一种既满足 GAC 又满足 PWC 的相容性,也就是说:如果一个问题满足 GAC,那么它不一定满足 fPWC.当前,有两种方式可以用来实现 fPWC.一种方式为原始数据编码,比如 k-交叉编码(k-interleaved encoding)[13]、因子编码(factor encoding,简称 FE)[14]和因子分解编码(factor-decomposition encoding,简称 FDE)[15]等.FDE 是目前效率最高的编码方式,通常与 STR 算法一起使用.另一种方式为在传播过程中维持高阶相容性,主要算法包括 maxRPWC[16]、maxRPWC+[17]、eSTR[18]、PW-AC[19]和 PW-CT[20]等.这些算法在维持 GAC 的基础上进行额外的检查,当前效率最高的算法是 PW-CT.

通过对维持 GAC 与 fPWC 的主要算法分析,当前效率最高的算法都使用了 bitset 的数据结构,例如 CT,STRbit,PW-CT 等.FDE 是一种将一个约束网络进行编译,形成一个新的约束网络的编码方法.在这个新的约束网络上,维持 GAC 等价于在原始约束网络上维持 fPWC.我们发现,当前使用 bitset 的算法在求解这个新的约束网络时有较高的空间复杂度,可能造成内存溢出问题.这是因为通过 FDE 形成的约束网络的规模要大于原始的约束网络,甚至是它的好几倍.因此在本文中,我们提出了 STRFDE 算法——一种使用 bitset 结构来求解 FDE 实例的方法.它结合了 CT 和 STRbit 的优势,在保证求解效率的同时,使占用的内存尽可能小.

在本文中,STRFDE 根据 FDE 的特性将整个约束网络分解为两个部分,并对这两个部分使用不同的算法.我们说明了针对不同的约束使用不同的算法是合理的.PW-CT,eSTR 和其他 PWC 算法不能直接在主流求解器上实现,这是因为它们的约束不能独立传播.但是,STRFDE 可以直接在主流求解器上实现.实验结果表明了,STRFDE 在大多数实例上的效率要比 PW-CT 和 STR2+FDE 高并且在一些实例上要高于 CT.

1 背景知识

约束满足问题(constraint satisfaction problem,简称 CSP)是判断一个约束网络 N 是否有解.约束网络可以表示为 $N=(X,C)$,其中,X 是由 n 个变量组成的变量集合:$X=\{x_1,x_2,\dots,x_n\}$,C 是由 e 个约束组成的约束集合:$C=\{c_1,c_2,\dots,c_e\}$,$D(x)$表示变量 x 对应的论域.为了简便,我们称(x,a)为一个值,若满足 $x\in X$ 且 $a\in D(x)$.例如,$(x,3)$是一个变量值.

每个约束 $c\in C$ 都包含一个变量集合 $scp(c)$和一个元组集合 $rel(c)$.$scp(c)=\{x_1,x_2,\dots,x_r\}$是变量集合 X 的一个子集,表示约束所涉及的变量.$rel(c)$是变量论域笛卡尔积$\{D(x_1)\times D(x_2)\times\dots\times D(x_r)\}$的子集,表示约束的有效元组.$t=(a_1,a_2,\dots,a_r)$是约束 c 的一个元组,$t\in rel(c)$.对于 $x_i\in scp(c)$,对应的值表示为 $t(x_i)$.元组 t 是有效的当且仅当对于任意的 $x_i\in scp(c)$,满足 $t(x_i)\in D(x_i)$;否则 t 是无效的.如果元组 t 是有效的,则称 t 为约束 c 的一个支持.若 t 为约束 c 的一个支持且 $t(x_i)=a$,则称 t 为(x_i,a)的一个支持.若元组集 $rel(c_i)$中的元组 t_i 与元组集 $rel(c_j)$中的元组 t_j 满足 $t_i[scp(c_i)\cap scp(c_j)]=t_j[scp(c_i)\cap scp(c_j)]$,且 t_i 和 t_j 是有效的,那么我们称 t_i 是 t_j 的 PW 支持.我们称 c_i 和 c_j 为非平凡相

© 中国科学院软件研究所 http://www.jos.org.cn

交,当且仅当 c_i 和 c_j 满足$|scp(c_i)\cap scp(c_j)|>1$.

定义 1. GAC.

- 变量值(x,a)在约束 c 上是 GAC 的当且仅当任意包含 x 的约束中存在对(x,a)有效的支持;
- 变量 x 是 GAC 的当且仅当 $D(x)$中所有变量值是 GAC 的;
- 约束 c 是 GAC 的当且仅当 $scp(c)$中所有变量是 GAC 的;
- 约束网络是 GAC 的当且仅当所有约束是 GAC 的.

定义 2. PWC.

- 约束 c_i 中的元组 t 是 PWC 的当且仅当任意 $c_j\in C$ 中存在对 t 的 PW 支持;
- 约束 c_i 是 PWC 的当且仅当约束 c_i 的所有元组是 PWC 的;
- 约束网络是 PWC 的当且仅当所有约束是 PWC 的.

定义 3. fPWC.

若一个约束网络既满足 PWC 又满足 GAC,那么此约束网络是 fPWC 的.

如图 1(a)所示,对偶图的顶点表示约束网络的约束,边表示两个约束之间涉及相同的变量,边的权值表示涉及的相同变量.Janssen 等人[12]和 Dechter[21]发现,当两个顶点之间存在至少一条长度大于 1 的路径,且这条路径上的每个顶点都包含这两个顶点相同的变量,那么这两个顶点之间的边是多余的.例如,R_1和R_4之间的边是多余的,它们存在路径 $R_1R_2R_4$ 且 R_2 也包含变量 A.将一个对偶图所有的冗余边删除得到的图,称为最小对偶图,如图 1(b)所示.

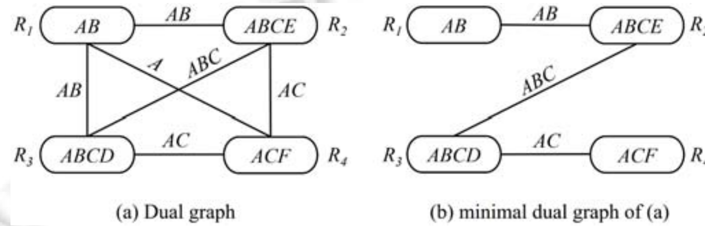

Fig.1 Dual graph and its minimal dual graph

图 1 Dual 图和它的最小对偶图

最小对偶图是对偶图的一个简化,一个对偶图可能有多个最小对偶图.最小对偶图和对偶图拥有相同的表达能力,不会影响算法的正确性.前人的研究工作已经证明,使用最小对偶图比使用对偶图效率更高[20].因此在本文中,我们实现的所有 fPWC 算法都是基于最小对偶图的.

2 STRFDE

在本节中,我们首先描述算法的基本思想并给出伪代码,然后对算法的复杂度进行证明.FDE 是一种将一个约束网络进行编译形成一个新的约束网络的编码方法,在这个新的约束网络上维持 GAC,等价于在原始约束网络上维持 fPWC.这个新的约束网络中的约束具有不同的特征,我们根据这些特征将约束分为两部分:附加约束和原始约束.

针对这两种约束,我们提出了不同的算法模型,因此,STRFDE 包含两种算法:STRFDE^{add} 和 STRFDE^{ori}.

使用不同的算法来处理不同的约束是合理的.每个约束在搜索过程中是相对独立的,它们拥有私有的属性,并根据相同的变量连接起来,我们只需要将变量的改变传播给相应的约束,就能判断当前问题是否相容.在解决一个实例时,根据约束的特征选择合适的算法会提升求解效率.

STRFDE^{add} 和 STRFDE^{ori} 是相对独立的,可以在主流求解器中直接实现.为了更好地介绍它们,我们给出一些定义.

定义 4(平凡变量). 经 FDE 处理之后,变量论域未改变的变量是平凡变量.

© 中国科学院软件研究所 http://www.jos.org.cn

定义 5(**因子变量**). 根据两个约束之间的共同变量生成的变量是因子变量.

定义 6(**原始约束**). 约束中的平凡变量被替换成相应的因子变量,并且对应的元组发生改变,这样的约束是原始约束.

定义 7(**附加约束**). 由一个因子变量和它对应的平凡变量所组成的约束是附加约束.

例 1:假设一个约束网络 $N=(X,C)$如图 2(a)所示,其中,$X=\{x,y,u,v,w\}$,$D(x)=D(y)=\{0,1\}$,$D(u)=D(v)=D(w)=\{0\}$,$C=\{c_1,c_2,c_3\}$.此约束网络经 FDE 编码后形成新的约束网络 $N'=(X',C')$如图 2(b)所示,其中,$X'=\{x,y,u,v,w,f\}$,$D(x)=D(y)=\{0,1\}$,$D(u)=D(v)=D(w)=\{0\}$,$D(f)=\{0,1,2,3\}$,且 $C'=\{c_1',c_2',c_3',c_f\}$.在约束网络 N'中,x,y,u,v 和 w 是平凡变量,f 是由平凡变量 x 和 y 生成的因子变量,c_1',c_2' 和 c_3' 是原始约束,c_f 是附加约束.在实际的问题中,经 FDE 编码形成的约束网络的规模会更加庞大.

c_1

x	y	u
0	0	0
0	1	0
1	1	0

c_2

x	y	v
0	0	0
0	1	0
1	0	0

c_3

x	y	w
0	0	0
1	0	0
1	1	0

(a) Original

c_1'

u	f
0	0
0	1
0	2

c_2'

v	f
0	0
0	1
0	3

c_3'

w	f
0	0
0	3
0	2

c_f

x	y	f
0	0	0
0	1	1
1	1	2
1	0	3

(b) FDE of (a)

Fig.2 Original constraint network and its FDE

图 2 原始约束网络和它的 FDE

2.1 附加约束

附加约束涉及到的变量集合 *scp* 是由两个或两个以上平凡变量和一个因子变量所组成,那么我们可以对平凡变量和因子变量进行不同的处理.在给出具体算法之前,我们首先介绍算法中涉及的数据结构.

- *RSparseBitSet* 是 CT 算法使用的数据结构.*RSparseBitSet* 存储 4 个成员:*words* 是一个 bit 数组,*limit* 是一个存储 *words* 长度的整型变量,*index* 是一个存储 *words* 中非 0 位置的整型数组,*mask* 是一个用于修改 *words* 的 bit 数组.
- *support*[*x*,*a*]是一个静态 bit 数组,长度与 *words* 相同,用来存储当前约束对变量值(*x*,*a*)的 bit 支持.若当前约束的元组中变量 *x* 所对应的值为 *a*,那么 *support*[*x*,*a*]对应的元组 bit 位是 1,否则是 0.
- *last* 是一个整型数组,*last*[*x*]存储变量 *x* 的论域大小.
- *residues* 是一个整型数组,*residues*[*x*,*a*]存储对变量值(*x*,*a*)最近一次找到的支持索引.
- S^{val} 和 S^{sup} 是临时变量集合,S^{val} 中存储论域改变的变量,S^{sup} 中存储论域大于 1 的变量.这两个集合用于提升求解的效率.

算法 1 介绍了为实现 $\text{STRFDE}^{\text{add}}$ 对 *RSparseBitSet*[5]进行扩展新增加的两个方法:intersectWord 和 getWord. intersectWord 方法使用因子变量来修改 *RSparseBitSet* 中的 *words*,参数为因子变量的论域.getWord 方法用来获取当前 *RSparseBitSet* 中的 *words*,返回值是一个 bit 数组.

算法 1. RSparseBitSet.

```
1.  Data: word: Temporary array of long
2.  Method intersectWord(value: Array of long):
3.    for i←limit down to 0 do
4.      offset←index[i]
5.      w←words[offset] & value[offset]    //bitwise AND
6.      if w!=words[offset] then
7.        words[offset]←w
8.        if w=0 then
```

© 中国科学院软件研究所 http://www.jos.org.cn

```
9.         index[i]←index[limit]
10.        index[limit]←offset
11.        limit←limit−1
12. Method getWord(·):
13.   for i←limit down to 0 do
14.     offset←index[i]
15.     word[offset]←words[offset]
16.   return word
```

算法 2 介绍了 $STRFDE^{add}$ 的伪代码,省略部分与 CT 算法相同.$STRFDE^{add}$ 是基于 CT 算法的改进算法,由于附加约束所涉及的最后一个变量为此约束所对应的因子变量 *x*,且因子变量的论域和约束的元组是一一对应的,所以因子变量的论域大小$|D(x)|$等于当前约束的元组数量$|rel(c)|$,若对每一个 $a\in D(x)$都构造一个 *support*[*x*,*a*]而 *support*[*x*,*a*]的长度与 *RSparseBitSet* 中 *words* 的长度相等,这样就可能造成内存溢出问题.为了解决这一问题,我们使用 bitset 来存储因子变量的论域,当因子变量发生改变时,直接使用因子变量的论域来修改约束的元组.同样地,当约束的元组发生改变时,直接使用 *RSparseBitSet* 中的 *words* 来修改因子变量的论域.如此,我们只需要为每个平凡变量构造 *support*[*x*,*a*].因此,*last*,*support* 和 *residues* 的大小都为$|scp|-1$.在算法 2 中的第 4 行和第 21 行所使用的 *getFactorVariable* 方法是用来获取与当前附加约束所对应的因子变量.

算法 2. $STRFDE^{add}$.

```
1.  Data: scp: array of variables
2.  currTable: RSparseBitSet
3.  Method updateTable(·):
4.    x←getFactorVariable(this)
5.    if x.change(·) then
6.      currTab.intersectWord(x.getBitDom(·))
7.    foreach x∈S^val do
8.      …
9.  Method filterDomains(·):
10.   …
11. Method enforceGAC(·):
12.   S^val←{x∈scp:|D(x)|!=lastSize[x]}
13.   foreach x∈S^val do
14.     last[x]←|D(x)|
15.   S^sup←{x∈scp:|D(x)|>1}
16.   updateTable(·)
17.   if currTable.isEmpty(·) then
18.     return Backtrack
19.   filterDomains(·)
20.   if currTable.change(·) then
21.     x←getFactorVariable(this)
22.     x.removeValues(currTab.getWord(·))
```

$STRFDE^{add}$ 由 3 种方法构成.

- *updateTable* 方法利用约束所涉及的变量来更新当前的有效元组:首先,获取当前附加约束所对应的因子变量;然后判断这个变量是否被其他约束修改过,如果被修改过,那么我们需要用它来更新当前的有

© 中国科学院软件研究所 http://www.jos.org.cn

效元组,这通过在第 6 行调用 *intersectWord* 方法来实现;最后,通过被修改过的平凡变量来更新当前的有效元组.

- *filterDomains* 方法利用 *RSparseBitSet* 来过滤平凡变量的论域,若使某个变量的论域为空,则发生回溯.
- *enforceGAC* 方法为 $STRFDE^{add}$ 的入口,在调用 *filterDomains* 之后,只有因子变量 *x* 的论域没有被过滤,若有效元组在调用 *updateTable* 时被平凡变量修改,那么在第 22 行,利用变量的 *removeValues* 方法根据当前有效元组来更新因子变量的论域.

2.2 原始约束

原始约束涉及的变量集合 *scp* 由 *n* 个平凡变量和 *m* 个因子变量组成,其中,*n* 和 *m* 满足 $n\geqslant 0,m\geqslant 0$ 且 $n+m\geqslant 1$. 原始约束中涉及到的因子变量与附加约束中的因子变量不同,这些因子变量的论域与原始约束的元组不是一一对应的,因此不能使用元组直接更新变量的论域.如前所述,因子变量的论域大小等于附加约束的元组个数,因此,在原始约束中对因子变量的每个值都构造一个 *support*[*x*,*a*]同样可能造成内存溢出问题.为解决这个问题,我们根据 STRbit 提出了一种新的算法.STRbit 也是当前主流的 GAC 算法,它的效率和 CT 不相上下.在给出具体算法之前,我们首先介绍算法中涉及的数据结构.

- *bitsup*[*x*,*a*]是一个数组,用来表示当前约束对变量值(*x*,*a*)的 bit 支持集合,它只存储不为 0 的 bit 支持. *bitsup*[*x*,*a*].*size* 表示当前约束对变量值(*x*,*a*)的 bit 支持数量,*bitsup*[*x*,*a*][*i*]表示第 *i* 个 bit 支持, *bitsup*[*x*,*a*][*i*].*ts* 表示第 *i* 个 bit 支持中的索引,*bitsup*[*x*,*a*][*i*].*mask* 表示第 *i* 个 bit 支持对应的 *mask* 向量.
- *val* 是一个存储元组 bit 向量的数组.
- *del*[*x*]是一个变量值的集合,存储所有已经从变量论域 *D*(*x*)中删除但还没有更新当前约束中元组的变量值.
- *residues* 是一个整型数组,*residues*[*x*,*a*]存储对变量值(*x*,*a*)最近一次找到的支持索引.

算法 3 介绍了 $STRFDE^{ori}$ 的伪代码,STRbit 中的 *LAST* 和 *restoreL* 数据结构有较高的复杂度,可能造成内存溢出问题,因此 $STRFDE^{ori}$ 没有使用它们.STRbit 中的 *bitsup* 只存储非 0 的 bit 支持,所以每个 *bitsup*[*x*,*a*]的长度可能是不同的,它的长度在算法初始化时被确定.因子变量的变量值越多,相对应的每个变量值的 bit 支持数量则越少,因此,使用 *bitsup* 会大幅度地减少内存占用.在算法中,我们使用 *residues* 去存储变量值最近一次找到的 bit 支持索引,若此支持仍然有效,就不必去遍历 *bitsup*.

算法 3. $STRFDE^{ori}$.

```
1.  Data: scp: array of variables
2.  Method deleteInvalidTuple(·):
3.    foreach x∈scp do
4.      foreach a∈del[x] do
5.        for i←bitsup[x,a].size−1 down to 0 do
6.          index←bitsup[x,a][i].ts
7.          u←bitsup[x,a][i].mask & val[index]
8.          if u!=0 then
9.            val[index]←(¬u) & val[index]
10.     del[x].clear(·):
11. Method searchSupport(·):
12.   foreach x∈scp do
13.     foreach a∈D(x) do
14.       now←residues[x,a]
15.       index←bitsup[x,a][now].ts
16.       if bitsup[x,a][i].mask & val[index]=0 then
```

```
17.          i←bitsup[x,a].size−1
18.          index←bitsup[x,a][i].ts
19.          while bitsup[x,a][i].mask & val[index]=0 do
20.            i←i−1
21.            if i=−1 then
22.              D(x)←D(x)\{a}
23.              if D(x).empty(·) then
24.                return BackTrack
25.              break
26.            index←bitsup[x,a][i].ts
27.          if i!=−1 then
28.            residues[x,a]←i
29. Method enforceGAC(·):
30.   deleteInvalidTuple(·)
31.   searchSupport(·)
```

$STRFDE^{ori}$由 3 种方法构成.*deleteInvalidTuple* 方法使用已经被删除的值来更新当前约束的元组(第 2 行~第 10 行),也就是使用 *del* 和 *bitsup* 来更新 *val*.在第 8 行,如果 *u* 不等于 0,这意味着当前约束的元组中存在对变量值(x,a)的支持.由于(x,a)已经被删除,因此在第 9 行将支持(x,a)的元组删除.*searchSupport* 方法用于为变量的每一个值来寻找支持(第 11 行~第 28 行).如果这个变量值不存在支持,那么就需要将其从相应的论域中移除.若论域删空,则发生回溯.*enforceGAC* 方法是 $STRFDE^{ori}$ 的入口.实际上,只用 $STRFDE^{ori}$ 就可以求解 FDE 实例,实验结果会在下一节中展示.

2.3 复杂性

下面我们证明 STRFDE 的空间复杂度和时间复杂度,并分析说明 STRFDE 针对 FDE 实例的实际空间复杂度小于 CT 和 STRbit,证明我们的方法是有意义的.为了计算算法的复杂度,我们假定约束网络 N 中约束个数为 e,约束涉及变量个数为 $r=r_o+r_f$(r_o 和 r_f 分别为平凡变量和因子变量的个数),平凡变量论域大小为 d_o,因子变量论域大小为 d_f,约束包含元组个数为 t,bit 位数为 w(w=64).

性质 1. $STRFDE^{add}$ 的最坏空间复杂度为 $O(r_od_ot/w)$.

证明:$STRFDE^{add}$ 采用的数据结构有 *RSparseBitSet*,*support*,*last* 和 *residues*,它们的空间复杂度分别是 $O(3t/w)$,$O(r_od_ot/w)$,$O(r_o)$,$O(r_od_o)$.S^{val} 和 S^{sup} 空间复杂度都为 $O(r_o)$,因此总的空间复杂度为 $O(3t/w+r_od_ot/w+r_o+r_od_o+r_o+r_o)=O(r_od_ot/w)$.证毕. □

对于附加约束,CT 算法中 *support* 结构的空间复杂度为 $O(r_od_ot/w+d_ft/w)$,因此,CT 算法的空间复杂度为 $O(r_od_ot/w+d_ft/w)$.大部分实例的因子变量论域大小 d_f 要远远大于平凡变量的论域大小 d_o,因此直接使用 CT 来求解 FDE 实例可能造成内存溢出问题.

性质 2. $STRFDE^{add}$ 的最坏时间复杂度为 $O(r_od_ot/w)$.

证明:$STRFDE^{add}$ 算法根据 CT 算法改进而来,CT 算法的时间复杂度为 $O(rdt/w)$[22],$STRFDE^{add}$ 算法中第 6 行和第 22 行的时间复杂度为 $O(t/w)$,其余的算法复杂度为 $O(r_od_ot/w)$.因此,附加约束算法的时间复杂度为 $O(r_od_ot/w)$.证毕. □

性质 **3**. $STRFDE^{ori}$ 的最坏空间复杂度为 $O(r_od_ot/w+r_fd_ft/w)$.

证明:$STRFDE^{ori}$ 采用的数据结构有 *bitsup*,*val*,*del* 和 *residues*.对于 *val*,*del* 和 *residues*,它们的空间复杂度分别是 $O(t/w)$,$O(r_od_o+r_fd_f)$和 $O(r_od_o+r_fd_f)$;对于因子变量,*bitsup* 的最坏空间复杂度为 $O(r_fd_ft/w)$;对于平凡变量,*bitsup* 的最坏空间复杂度为 $O(r_od_ot/w)$.因此,$STRFDE^{ori}$ 的最坏空间复杂度为 $O(r_od_ot/w+r_fd_ft/w)$.证毕. □

我们发现,$STRFDE^{ori}$ 和 CT 对于附加约束的最坏空间复杂度相同,但由于 $STRFDE^{ori}$ 的 *bitsup* 只存储不为

0 的 bit 支持,而 CT 的 *support* 存储所有的 bit 元组,所以 STRFDE^{ori} 的实际空间复杂度要小于 CT.对于 STRbit 来说,STRbit 有两个额外的数据结构 *LAST* 和 *restoreL*.*LAST* 的空间复杂度为 $O(r_o d_o + r_f d_f)$,但它在搜索树的每一层会存储到 *restoreL* 中,因此,STRFDE^{ori} 的实际空间复杂度也小于 STRbit.

性质 4. STRFDE^{ori} 的最坏时间复杂度为 $O(r_o d_o t/w + r_f d_f t/w)$.

证明:在不考虑平凡变量和因子变量的情况下,STRFDE^{ori} 算法中 *deleteInvalidTuple* 和 *searchSupport* 的时间复杂度都是 $O(rdt/w)$,因此,STRFDE^{ori} 的最坏时间复杂度为 $O(r_o d_o t/w + r_f d_f t/w)$.证毕. □

和对性质 3 的分析一样,STRFDE^{ori} 的实际时间复杂度也是相对较小的.

3 实 验

我们在 http://www.cril.univ-artois.fr/~lecoutre/benchmarks.html 上选取了 13 组存在非平凡相交的实例集,共有 403 个实例.我们实现并比较了 7 种算法:CT、PW-CT、CT+FDE、STRbit+FDE、STR2+FDE、STRFDE+FDE 和 STRFDE^{ori}+FDE.其中,CT 是当前广泛使用的 GAC 算法,其他算法为 fPWC 算法.STRFDE^{ori}+FDE 只用 STRFDE^{ori} 求解 FDE 实例,而 STRFDE+FDE 则用 STRFDE^{ori} 和 STRFDE^{add} 分别处理原始约束和附加约束.为了简化,下文中使用 STRFDE 代替 STRFDE+FDE.我们采用了 dom/ddeg 变量启发式和 min 值启发式.所有算法的实现基于使用 scala2.12 和 java11 编写的 CP 求解器,测试环境为 Intel Core i7 3.20GHz 处理器、8GB RAM 内存和 Windows 10 操作系统.我们限定算法的超时时间为 1 800s.

表 1 展示了算法求解每个实例集的平均时间 *t*(单位为 s)和平均结点个数 *n*,#表示实例个数;MO 表示算法在此实例集上发生内存溢出,对应的百分比表示发生内存溢出的实例个数占总实例个数的比例(例如,100%意味着算法在此实例集的所有实例上都发生内存溢出).表 1 中有背景的字体表示所有算法中最短的求解时间,加粗字体表示所有 fPWC 算法中最短的求解时间,最后一行给出了每种算法在给定时间内可求解的实例个数.

Table 1 Mean times (s) and nodes for each algorithm

表 1 每种算法的平均时间(秒)和结点数

Name		CT	PW-CT	FDE				
				CT	STRbit	STR2	STRFDE	STRFDE^{ori}
aim-50	*t*	0.040	**0.014**	0.069	0.077	0.068	0.069	0.062
#=24	*n*	3 316	98	3 391	3 391	3 391	3 391	3 391
aim-100	*t*	163.083	**5.899**	168.612	172.007	170.610	168.028	166.848
#=24	*n*	9.47M	78 045	4.59M	408M	4.58M	4.90M	4.88M
aim-200	*t*	1 151.755	**541.227**	816.262	816.768	814.895	815.118	814.392
#=20	*n*	31.03M	4.53M	11.16M	10.40M	12.56M	12.84M	14.21M
dubois	*t*	995.286	820.407	740.444	764.879	727.449	720.033	**716.390**
#=13	*n*	96.37M	50.38M	62.97M	57.93M	66.35M	68.82M	69.18M
renault	*t*	**0.015**	4.001	1.072	0.072	0.072	**0.061**	0.065
#=2	*n*	101	101	101	101	101	101	101
modifiedRenault	*t*	725.629	147.464	92.209	108.066	100.403	**90.781**	93.152
#=50	*n*	22.10M	316 732	945 615	328 602	871 574	954 641	832 858
rand-10-20-10	*t*	0.031	0.662	0.132	0.017	0.024	**0.010**	0.017
#=20	*n*	830	0	0	0	0	0	0
rand-10-60-20	*t*	17.437	MO	MO	MO	0.414	**0.151**	MO
#=50	*n*	27 624	100%	100%	100%	0	0	50%
rand-3-20-20f	*t*	**12.167**	469.325	35.546	37.216	67.944	**29.708**	38.039
#=50	*n*	115 064	329 61	43 695	43 695	43 695	43 695	43 695
rand-3-20-20	*t*	**24.722**	786.030	73.965	75.418	142.138	**59.276**	75.465
#=50	*n*	225 814	53 160	87 943	87 943	87 943	87 943	87 943
rand-5-12-12	*t*	1.000	24.168	MO	MO	0.812	**0.047**	MO
#=50	*n*	1 043	0	100%	100%	0	0	100%
rand-8-20-5	*t*	**3.860**	1 731.243	MO	17.099	17.128	**16.614**	20.082
#=20	*n*	86 903	1 912	70%	6 646	6 646	6 646	6 646
travellingSalesman	*t*	**25.419**	MO	**26.478**	36.135	41.696	31.165	27.939
#=30	*n*	52 933	93%	52 933	52 933	52 933	52 933	52 933
Solve		368	283	272	285	**386**	**386**	311

如表 1 所示,CT 在 5 组实例上是最快的,但是在求解某些 sat 实例时会消耗较多的时间,例如 aim-200 和 dubois;STRFDE 在 4 组实例上是最快的,并且可以求解出的实例数最多;STRFDE 与其他 fPWC 算法相比,在 8 组实例上是最快的.通过分析数据可以发现,CT 检查一个点是否满足 GAC 所用的时间较少,而 fPWC 算法检查一个点所用的时间较长.一般来说,fPWC 算法的效率低于 GAC 算法,这是因为对某些实例,维持 fPWC 会在 GAC 的基础上进行额外的检查,从而删去更多的元组.

比较 PW-CT 与 STRFDE,PW-CT 在一些 sat 实例上具有优势.这是因为当某些约束涉及的元组较少时,使用 PW-CT 有较高的效率.但是当约束涉及大量元组时,使用 PW-CT 的效率就比较低.因此,除了一些元组较少的实例外,STRFDE 的效率是优于 PW-CT 的.与 CT+FDE 和 STRbit+FDE 相比,STRFDE 在大部分实例上的效率有提升,并且它能求解一些 CT 和 STRbit 造成内存溢出的实例.$STRFDE^{ori}$+FDE 在一些实例上也有较高的效率,但是会在求解附加约束的数量远多于原始约束的实例时占用更多的内存,例如 rand-5-12-12,这是因为它为附加约束中的因子变量存储了支持.STRFDE 与 STR2+FDE 相比有巨大的提升,就像 CT 和 STRbit 比 STR2 快一样.综上所述,STRFDE 的效率在 fPWC 算法中是有竞争力的.

在该表中,PW-CT 的结点数比其他算法少.这是因为 PW-CT 使用原始的约束网络来维持 fPWC,其执行额外的 PWC 检查来删除比 CT 更多的元组.如果执行 PWC 检查的速度很慢,并且只删除很少的元组,那么 PW-CT 是低效的.这就解释了为什么 PW-CT 求解随机实例时需要花费更长的时间.后 5 种算法的结点数在一些实例上是相同的,这意味着它们具有相同的搜索树.实际上,STRFDE 在单位时间(s)内搜索的结点比 STR2+FDE 多.

如果一个实例没有解并且没有 PW 支持,那么只需在 fPWC 初始化检查时就可以确定这个实例的结果.这种情况的结点数为 0,例如 rand-10-20-10 和 rand-5-12-12.因此在求解这类实例时,fPWC 算法比 GAC 算法更有优势.对于至少有一个解的实例,例如 rand-8-20-5,它的每个约束都包含大量的元组,在其上维持 fPWC 将花费一定的时间来检查每个元组是否满足 PWC.因此,在求解这类实例时,维持 GAC 比维持 fPWC 更有效率.

为了使结果更加直观,我们通过散点图(如图 3 所示)展示了 STRFDE 与不同算法求解实例的对比情况.如第一个散点图所示,STRFDE 虽然不能优于 CT,但在求解部分实例时比 CT 的效率更高.其他散点图的大部分点位于线 $y=x$ 下,这表明 STRFDE 在求解大多数实例时快于对比算法.

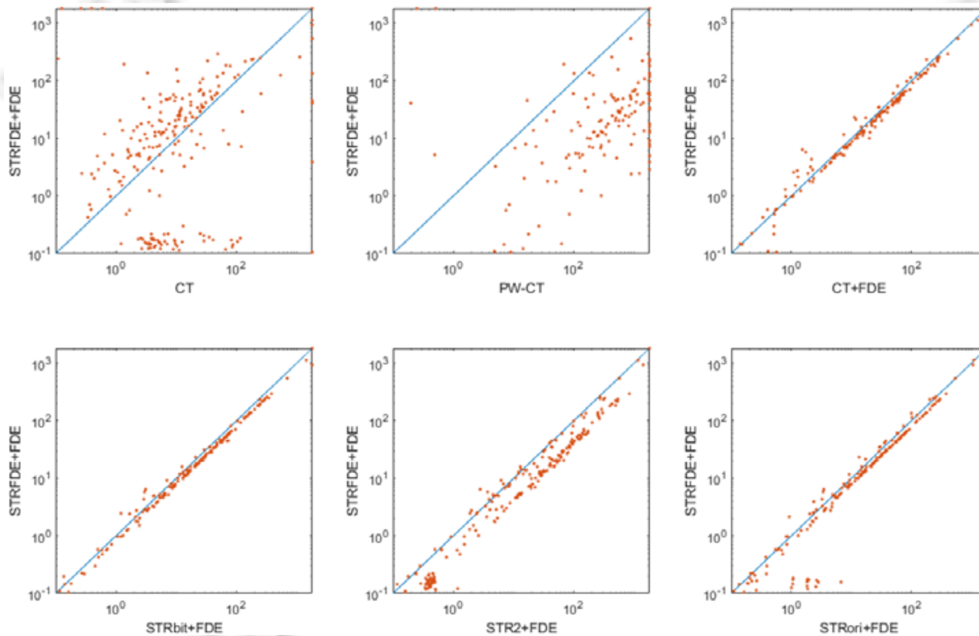

Fig.3 Runtime comparison of STRFDE and other algorithms on all instances

图 3 STRFDE 与其他算法在所有实例上的运行时间比较

表2给出了算法在求解实例集时所占用的平均内存(以Mb为单位),百分比表示内存溢出的比例.通过分析,我们可以得出,在所有的实例上,CT+FDE与STRbit+FDE均比STRFDE占用更多的内存,有的甚至是STRFDE的数十倍;与STR2+FDE相比,STRFDE内存增加的原因是使用了bitset的数据结构.这表明,STRFDE在保持效率的同时尽可能地减少了内存占用.STRbit在求解一些实例时占用内存高于其他算法的原因是,其使用了*LAST*和*restoreL*数据结构.$STRFDE^{ori}$在约束涉及的元组较少时占用内存略优于STRFDE,但整体上还是STRFDE效果更好.

Table 2 Mean memory for each algorithm (Mb)

表 2 每种算法的平均内存 (Mb)

name	#	CT	PW-CT	FDE				
				CT	STRbit	STR2	STRFDE	$STRFDE^{ori}$
aim-50	24	0.44	5.24	1.98	5.50	0.83	1.17	1.16
aim-100	24	1.41	19.32	6.40	18.42	2.64	3.64	3.61
aim-200	20	4.11	51.72	12.69	35.95	5.53	8.40	8.29
dubois	13	0.74	5.31	0.77	1.70	0.46	0.73	0.73
renault	2	5.17	234.93	1 450.96	307.66	4.67	26.67	35.72
modifiedRenault	50	5.75	266.06	1 455.73	378.53	5.71	29.17	38.65
rand-10-20-10	20	0.77	16.75	648.98	75.82	1.34	14.90	26.81
rand-10-60-20	50	48.42	100%	100%	100%	79.22	612.41	50%
rand-3-20-20f	50	1.84	38.35	19.53	25.37	1.05	6.88	8.20
rand-3-20-20	50	1.84	38.35	19.53	25.37	1.05	6.88	8.20
rand-5-12-12	50	22.76	346.20	100%	100%	261.54	908.14	100%
rand-8-20-5	20	10.56	184.47	70%	280.94	7.50	128.02	136.77
travellingSalesman	30	59.73	93%	59.73	42.90	2.20	24.24	24.24

4 总 结

本文提出了一种结合FDE方法一起维持fPWC的表约束求解算法STRFDE,它采用不同的方法来处理不同的约束.STRFDE结合了CT和STRbit的优点,在保证求解效率的同时,使占用的内存尽可能小.实验表明,STRFDE+FDE在大多数情况下是维持fPWC最快的算法.当然,STRFDE也有一些缺点:它只适用于经过FDE处理后的约束网络,不能应用在原约束网络上.下一步,我们准备继续完善STRFDE算法,并将它在主流求解器上实现.

References:

[1] Ullmann JR. Partition search for non-binary constraint satisfaction. Information Sciences, 2007,177(18):3639–3678.

[2] Lecoutre C. STR2: Optimized simple tabular reduction for table constraints. Constraints, 2011,16(4):341–371.

[3] Lecoutre C, Likitvivatanavong C, Yap RHC. STR3: A path-optimal filtering algorithm for table constraints. Artificial Intelligence, 2015,220:1–27.

[4] Wang R, Xia W, Yap RHC, Li Z. Optimizing simple tabular reduction with a bitwise representation. In: Proc. of the IJCAI. 2016. 787–795.

[5] Demeulenaere J, Hartert R, Lecoutre C, Perez G, Perron L, Régin JC, Schaus P. Compact-table: Efficiently filtering table constraints with reversible sparse bit-sets. In: Proc. of the Int'l Conf. on Principles and Practice of Constraint Programming. Cham: Springer-Verlag, 2016. 207–223.

[6] 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 with English abstract). http://www.jos.org.cn/1000-9825/4847.htm [doi: 10.13328/j.cnki.jos.004874]

[7] Yang MQ, Li ZS, Li Z. Optimizing MDDc and STR3 for solving constraint satisfaction problem. Ruan Jian Xue Bao/Journal of Software, 2017,28(12):3156–3166 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5242.htm [doi: 10.13328/j.cnki.jos.005242]

[8] Yang MQ, Li ZS, Zhang JC. Simple tabular reduction algorithm based on time-stamp mechanism. Ruan Jian Xue Bao/Journal of Software, 2019,30(11):3355–3363 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5559.htm [doi: 10.13328/j.cnki.jos.005559]

© 中国科学院软件研究所 http://www.jos.org.cn

[9] Cheng KCK, 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.

[10] Perez G, Régin JC. Improving GAC-4 for table and MDD constraints. In: Proc. of the Int'l Conf. on Principles and Practice of Constraint Programming. Cham: Springer-Verlag, 2014. 606−621.

[11] Verhaeghe H, Lecoutre C, Schaus P. Compact-MDD: Efficiently filtering (s) MDD constraints with reversible sparse bit-sets. In: Proc. of the IJCAI. 2018. 1383−1389.

[12] Janssen P, Jégou P, Nouguier B, Vilarem M. A filtering process for general constraint-satisfaction problems: Achieving pairwise-consistency using an associated binary representation. In: Proc. of the IEEE Int'l Workshop on Tools for Artificial Intelligence. IEEE, 1989. 420−427.

[13] Mairy JB, Deville Y, Lecoutre C. Domain *k*-wise consistency made as simple as generalized arc consistency. In: Proc. of the Int'l Conf. on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems. Cham: Springer-Verlag, 2014. 235−250.

[14] Likitvivatanavong C, Xia W, Yap RHC. Higher-order consistencies through GAC on factor variables. In: Proc. of the Int'l Conf. on Principles and Practice of Constraint Programming. Cham: Springer-Verlag, 2014. 497−513.

[15] Likitvivatanavong C, Xia W, Yap RHC. Decomposition of the factor encoding for CSPs. In: Proc. of the 24th Int'l Joint Conf. on Artificial Intelligence. 2015. 353−359.

[16] Bessiere C, Stergiou K, Walsh T. Domain filtering consistencies for non-binary constraints. Artificial Intelligence, 2008,172(6-7): 800−822.

[17] Paparrizou A, Stergiou K. An efficient higher-order consistency algorithm for table constraints. In: Proc. of the 26th AAAI Conf. on Artificial Intelligence. 2012. 535−541.

[18] Lecoutre C, Paparrizou A, Stergiou K. Extending STR to a higher-order consistency. In: Proc. of the 27th AAAI Conf. on Artificial Intelligence. 2013. 576−682.

[19] Samaras N, Stergiou K. Binary encodings of non-binary constraint satisfaction problems: Algorithms and experimental results. Journal of Artificial Intelligence Research, 2005,24:641−684.

[20] Schneider A, Choueiry BY. PW-CT: Extending compact-table to enforce pairwise consistency on table constraints. In: Proc. of the Int'l Conf. on Principles and Practice of Constraint Programming. Cham: Springer-Verlag, 2018. 345−361.

[21] Dechter R. Constraint Processing. Morgan Kaufmann Publishers, 2003. 211−269.

[22] Verhaeghe H, Lecoutre C, Schaus P. Extending compact-table to negative and short tables. In: Proc. of the 31st AAAI Conf. on Artificial Intelligence. 2017. 3951−3957.

附中文参考文献:

[6] 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/1000-9825/4847.htm [doi: 10.13328/j.cnki. jos.004874]

[7] 杨明奇,李占山,李哲.优化求解约束满足问题的 MDDc 和 STR3 算法.软件学报,2017,28(12):3156−3166. http://www.jos.org.cn/1000-9825/5242.htm [doi: 10.13328/j.cnki.jos.005242]

[8] 杨明奇,李占山,张家晨.一种基于时间戳的简单表缩减算法.软件学报,2019,30(11):3355−3363. http://www.jos.org.cn/1000-9825/5559.htm [doi: 10.13328/j.cnki.jos.005559]

王震(1998－),男,硕士,主要研究领域为约束规划.

李占山(1966－),男,博士,教授,博士生导师,CCF 专业会员,主要研究领域为机器学习,约束推理.

李哲(1990－),男,博士,主要研究领域为约束规划,并行计算.

© 中国科学院软件研究所 http://www.jos.org.cn