软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(11):3355−3363 [doi: 10.13328/j.cnki.jos.005559]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

一种基于时间戳的简单表缩减算法*

杨明奇[1,2], 李占山[1,2], 张家晨[1]

[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: 张家晨, E-mail: zhangjc@jlu.edu.cn

摘 要: 表约束是一种外延的知识表示方法,每个约束在对应的变量集上列举出所有支持或禁止的元组.广义弧相容(generalized arc consistency,简称 GAC)是求解约束满足问题应用最广泛的相容性.Simple Tabular Reduction(STR)是一类高效的维持 GAC 的算法.在回溯搜索中,STR 动态地删除无效元组,降低了查找支持的开销,并拥有单位时间的回溯代价,在高元表约束上获得了广泛运用,并有大量基于 STR 的改进算法被提出,其中,元组集的压缩表示是目前研究较多的方法.同样基于动态维持元组集有效部分的思想,为 STR 提出一种检测并删除无效元组和为变量更新支持的算法,作用于原始表约束并拥有单位时间的回溯代价.实验结果表明,该算法在表约束上维持 GAC 的效率普遍高于现有的非基于压缩表示的 STR 算法,并且在一些实例上的效率高于最新的基于元组集压缩表示的 STR 算法.

关键词: 约束满足问题;简单表缩减;表约束;广义弧相容

中图法分类号: TP18

中文引用格式: 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

英文引用格式: 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). http://www.jos.org.cn/1000-9825/5559.htm

Simple Tabular Reduction Algorithm Based on Time-stamp Mechanism

YANG Ming-Qi[1,2], LI Zhan-Shan[1,2], ZHANG Jia-Chen[1]

[1](College 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 define an arbitrary constraint explicitly as a set of solutions or non-solutions. Generalized arc consistency (GAC) is the most widely used consistency for solving non-binary constraint satisfaction problems (CSPs). Simple tabular reduction (STR), which dynamically deletes invalid tuples during search, speeds up the process of updating supports for variables and can restore in constant time when backtrack occurs. STR achieves dynamically maintaining valid parts of tables during search, which has been shown to be efficient for enforcing GAC. Recent research on improving STR mainly focuses on the compressed representation of tables. In this study, a new algorithm is proposed based on dynamically maintaining valid parts of tables, which deletes invalid tuples and updates supports when enforcing GAC on table constraints. The proposed algorithm is applied to original table constraints and can also restore in constant time. Experimental evaluations show that the proposed algorithm outperforms existing STR algorithms without table

* 基金项目: 国家自然科学基金(61272208, 61373052); 吉林省科技计划(20180101043JC)

Foundation item: National Natural Science Foundation of China (61272208, 61373052); Science and Technology Plan of Jilin Province (20180101043JC)

收稿时间: 2017-07-25; 修改时间: 2017-09-28; 采用时间: 2018-01-22; jos 在线出版时间: 2018-04-27

CNKI 网络优先出版: 2018-04-27 14:58:03, http://kns.cnki.net/kcms/detail/11.2560.TP.20180427.1457.004.html

compression. In some classes of problems, the proposed algorithm is even more efficient than state-of-the-art compression based STR algorithms.

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

约束传播(或局部相容性)作为一种有限的推理方法[1],被广泛应用于求解约束满足问题.目前比较成功的一种方法是将回溯搜索与约束传播相结合,在预处理以及回溯搜索的每一阶段删除不满足相容性的赋值来降低搜索空间.局部相容性根据用于推理的约束子网的规模可以分为不同强度的相容性,通常,更强的相容性意味着更强的剪枝能力和更高的实现复杂度,而广义弧相容(generalized arc consistency,简称 GAC)是研究最多且应用最广泛的局部相容性方法.

近年来,在配置、数据库、偏好建模等领域有着重要应用的表约束求解方法受到了大量学者关注.表约束上的弧相容算法研究诞生了许多成功的技术,其中,simple tubular reduction(STR)[2]是一种可以动态维持元组集有效部分的算法.STR 使用一种简单的数据结构 sparse set[3,4]表示元组的序号,具有增量维持元组集的有效部分以及单位时间的回溯代价的性质.STR2[5]对 STR 提出两点改进:(1) 只对相邻两次调用中元组对应变量论域发生改变的位置检测有效性,实现了增量检测元组有效性;(2) 当变量中所有值均找到支持时,停止为该变量查找支持,避免了无用的支持查找.STR3[6]类似 GAC4 是路径最优的,通过查询 dual table,找到并删除无效值对应的无效元组,再通过被删除的无效元组为其他变量值更新支持.这避免了 STR2 中可能存在的对约束表同一区域的重复查找,但 dual table 中需要处理的元组数目是原始表中约束元数的倍数,当约束元数较高时,同样会存在较多重复的查找,路径最优方法对于能否提升维持 GAC 的效率并不明确.例如,在多数实例上,STR2 的效率优于 STR3[7,8].AdaptiveSTR[9]给出了一种自适应的表缩减方法,可以根据当前约束表中有效部分的规模,自适应地选择代价较低的表缩减策略.文献[10]改善了 STR2 在有效元组集缩减较慢时的性能.

将元组集压缩表示可以降低待处理的元组集的规模,也是一种有效的提高维持 GAC 效率的方法,并受到了深入的研究.其中,MDD 算法[11]将每个约束的元组集建成一个多值决策图,当多值决策图中同构的子图较多时,压缩率较高.在基于 STR 的压缩方法中,STR2-C 和 STR3-C[12]将多个元组表示为一个 c-tuples[13];ShortSTR[14]将多个元组表示为一个短元组;STR-slice[15]将多个元组表示为关联模式的子表;STRbit[16]用和 Compact-Table[17]均采用比特位表示元组序号,一次比特操作可以同时处理多个元组.此类元组集压缩方法维持 GAC 的效率受限于压缩率,当压缩率较高时加速显著;当压缩率降低时,加速效果减弱,部分方法会发生退化,例如,MDDc、STR2-C、STR3-C 在压缩率较低的问题上效率低于 STR2,STR3.

表约束上的高阶相容性算法[18−21]也受到广泛研究,目前应用较为成功的高阶相容性有 pairwise consistency (PWC)[22]以及弱化版 PWC,从降低维护代价的角度出发,最新的维持高阶相容性算法,例如 FE[8]、FDE[23],均采用对约束问题重构的策略,并证明了采用特定的重构方法,在重构后的问题上维持 GAC 等价于对原问题维持 PWC 或更高阶段相容性.值得注意的是,这些方法均通过实验证实,使用 STR2 对重构后的问题维持 GAC 的效率高于 MDDc,STR3 在重构问题上维持 GAC 的效率.

本文提出的 STR2*是一种非基于表压缩的 STR 算法,具有直接处理原始表、实现简单等特点.STR2*采用了一种全新的动态维持元组集有效部分的方法,包含效率更高的检测元组有效性以及为变量值更新支持的方法.实验结果表明,STR2*拥有比 STR2 和 STR3 更高的维持 GAC 的效率,在元组集数目极小时,STR2*趋近于 STR2;在元组集数目增多时,STR2*相对于 STR2 和 STR3 提升显著.对于元组集规模缩减较快的问题,STR2*优于现有的基于表压缩的算法.

1 背景知识

一个约束满足问题 $P=(X,C)$,其中,X 是变量集合$\{x_1,...,x_n\}$,C 是约束集合$\{c_1,...,c_e\}$.$dom(x)$是 $x\in X$ 的当前有效论域.我们使用(x,a)表示 x 的一个值 a(在不引起混淆的情况下,也可直接表示为 a),如果 $a\in dom(x)$,我们称 a 是有效的;否则,a 是无效的.每个 $c\in C$ 包括两个部分:$scp(c)$是 X 中有序的变量子集,表示 c 的约束范围,c 的元数是

|*scp*(*c*)|;*rel*(*c*)是与 *scp*(*c*)对应的元组的集合.给定 $scp(c)=\{x_1,...,x_n\}$,$rel(c)\subseteq\prod_{j=1}^{r}dom(x_{ij})$ 表示 *scp*(*c*)包含的变量中所有可满足的值的组合的集合.我们规定,所有元组集都是有序的.给定一个有序变量集合 $S\subseteq scp(c)$和一个元组 $t\in rel(c)$,*t* 关于 *S* 的投影 *t*[*S*]表示 *t* 中与 *S* 中变量对应的部分.*t* 是(*x*,*a*)在 *c* 中支持 iff *t*[*x*]=*a*.*t* 关于 *x* 是有效的 iff $t[x]\in dom[x]$;*t* 是有效的 iff 对于任意 $x\in scp(c)$,*t* 关于 *x* 是有效的;*c* 关于 $x\in scp(c)$是有效的 iff 对于任意 $t\in c$,*t* 关于 *x* 是有效的;*c* 是有效的 iff 对于任意 $x\in scp(c)$,*c* 关于 *x* 是有效的.显然,*c* 是有效的 iff *c* 中所有元组都是有效的.

定义 1(**广义弧相容 generalized arc consistency**(**GAC**)). 对于一个约束满足问题 *P*,(*x*,*a*)关于 *c* 是 GAC 的 iff *c* 中存在一个(*x*,*a*)的有效支持;*x* 关于 *c* 是 GAC 的 iff 对于任意 $a\in dom(x)$,(*x*,*a*)关于 *c* 是 GAC 的;*c* 是 GAC 的 iff 对于任意 $x\in scp(c)$,*x* 关于 *c* 是 GAC 的;*P* 是 GAC 的 iff 对于任意 $c\in C$,*c* 是 GAC 的.

P 的一个解是一个约束范围为 *X* 的有效元组,使得所有约束都被满足,*P* 是可满足的 iff 存在至少一个解.确定一个约束满足问题是否是可满足的是 NP-hard.显然,当一个值(*x*,*a*)不是 GAC 时,该值不会出现在任何解中,维持 GAC 就是通过在回溯搜索的每一个阶段删除所有不满足 GAC 的值,产生对搜索树剪枝的效果.

2 STR2*

STR2*基于回溯搜索中动态维持元组集的有效部分的思想,即在搜索的每一阶段变量论域发生删减时,删除所有与被删除的值关联的无效元组;同时,在回溯时恢复因回溯而重新变为有效的元组.STR2*采用如下数据结构.

- *table*(*c*)是存储了正表约束 *c* 中所有元组的数组,每个元组按在 *table*(*c*)数组中的位置唯一标识,*table*(*c*).*length* 表示约束 *c* 中元组的总个数.
- *position*(*c*)是存储了约束 *c* 中每个元组在 *table*(*c*)中的位置的数组,*c* 中第 *i* 个元组为 *table*(*c*)[*position*(*c*)[*i*]].在任意时刻,*position*(*c*)中的值是{0,1,2,...,*table*(*c*).*length*−1}的排列.
- *currentLimit*(*c*)是一个记录 *position*(*c*)中最后一个有效元组位置的整数,$-1\leqslant currentLimit(c)\leqslant position(c).length-1$,当 *currentLimit*(*c*)=−1 时,*c* 中当前有效元组数目为 0.
- *levelLimit*(*c*)记录回溯搜索中每一次赋值前 *position*(*c*)中最后一个有效元素的位置,用于对 *currentLimit*(*c*)的备份和恢复.

table(*c*)是一个仅用于查询的表,在 STR2*中,一个约束 *c* 中第 *n* 个元组对应变量 *x* 位置的取值为 *a*=*table*(*c*,*x*)[*t*].由于 *c* 中元组的个数远大于约束的元数,采用这种表示方法的 *table*(*c*)可以最大限度地利用连续的存储空间,加速查询速度.*position*(*c*)中包含了回溯搜索中 *c* 的当前有效元组和无效元组两个部分,其中,*position*(*c*)[0]到 *position*(*c*)[*currentLimit*(*c*)]是有效部分,*position*(*c*)[*currentLimit*(*c*)+1]到 *position*(*c*)[*position*(*c*).*length*−1]是无效部分.当一个元组 *position*(*c*)[*i*]因无效被删除时,我们只需将 *position*(*c*)[*i*]与 *position*(*c*)[*currentLimit*(*c*)]交换,然后将 *currentLimit*(*c*)减 1.而回溯时只需恢复 *currentLimit*(*c*)的值,便实现了对已删元组的恢复.显然,回溯发生后,元组在 *position*(*c*)中的顺序发生了改变,但这不影响算法的正确性.借助于上述数据结构,STR2*可以动态地维持元组集的有效部分,并且只需要单位时间的恢复代价.STR2*基本沿用了 STR2 的数据结构,只是 *table*(*c*)采用了一种与 STR2 中 *table*(*c*)[5]互补的表示方法.

STR2*包含两个部分.第 1 部分对应于算法 1 的第 1 行~第 11 行,算法检测约束 *c* 中元组有效性并删除无效元组.一个元组是无效的当且仅当存在至少一个$(x,a)\in t$ 是无效的.对于任一变量 $x\in scp(c)$,如果 *dom*(*x*)在上次调用 STR2*后没有发生改变,则 *c* 中所有元组关于 *x* 仍然是有效的,无需再次检测.因此,我们仅对 *c* 中元组在变量论域发生改变的位置上检测有效性.我们通过变量的时间戳和其所在约束的时间戳大小关系,识别变量论域在上一次调用 STR2*后是否发生改变,每一个约束 *c* 和每一个变量 *x* 在回溯搜索中被赋予一个全局唯一的时间戳.*time* 是一个全局计数器,用于标识不同事件发生的先后次序和更新变量和约束的时间戳,变量 *x* 论域发生改变时,*stamp*[*x*]被更新为当前的 *time*;在约束 *c* 上 STR2*时,*stamp*[*c*]被更新为当前的 *time*.当 *stamp*[*x*]<*stamp*[*c*]时,说明在最近一次对 *c* 调用 STR2*后,*dom*(*x*)没有发生改变,算法 1 跳过对该变量的有效性检测,对应于算法 1 的

第 3 行、第 4 行.根据变量时间戳和约束时间戳记录的信息可知,初始阶段,所有变量的时间戳均应大于该变量所在的所有约束的时间戳;当约束网络满足 GAC 时,所有变量的时间戳均应小于该变量所在的所有约束的时间戳.STR2*的第 1 部分中,外层循环是对约束包含的变量的迭代,内层循环是对当前有效元组的迭代.每检测完 c 关于一个变量 x 的有效性后,会有一部分关于 x 无效的元组被删除,$currentLimit(c)$随之减小.因此,随着外层循环对变量的迭代,内层循环的迭代次数逐渐减小.

图 1 是 STR2*在检测阶段动态维持当前有效表的一个例子,$scp(c)=\{x,y,z\}$.

- 图 1(a)是约束 c 的 $table(c)$,假设初始阶段约束 c 的状态为 $currentLimit(c)=10$,约束 c 中满足 $stamp[x]>stamp[c]$的变量集合为$\{x,y,z\}$,$dom(x)=\{a,b\}$,$dom(y)=\{b\}$,$dom(z)=\{a,c\}$.
- 图 1(b)是检测 c 关于 x 的有效性,即检测 $position(c)[0]$到 $position(c)[10]$中,元组关于 x 的有效性.8-th~10-th 元组被删除,$currentLimit(c)=7$.
- 图 1(c)是检测 c 关于 y 的有效性,即检测 $position(c)[0]$到 $position(c)[7]$中,元组关于 y 的有效性.2-th~4-th、7-th 元组被删除,$currentLimit(c)=3$.
- 图 1(d)是检测 c 关于 z 的有效性,即检测 $position(c)[0]$到 $position(c)[3]$中元组关于 z 的有效性.5-th、0-th 元组被删除,$currentLimit(c)=3$.

最终,c 的当前有效元组为 6-th、1-th 元组.

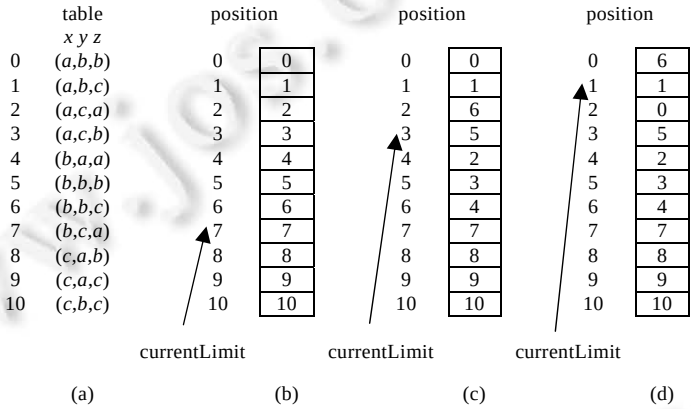

Fig.1 Demonstration of executing STR2* on the constraint c

图 1 STR2*在表约束 c 上的执行演示

STR2*的第 2 部分对应于算法 1 的第 12 行~第 26 行,算法分别为未赋值变量更新支持,并删除没有支持的变量值.外层是对变量的迭代,内层循环是对当前有效元组的迭代.当 $size=|dom(x)|$时,$dom(x)$中所有值均找到支持,算法跳出当前迭代,对应于算法 1 的第 21 行、第 22 行.所有论域发生删减的变量被更新时间戳,并放入集合 $Xevt.c$ 中所有变量均被处理后,更新 c 的时间戳.

算法 1. STR2*(c:constraint):set of variables.

```
1  Save currentLimit(c) to levelLimit(c)
2  foreach x∈scp(c) do
3    if stamp[x]<stamp[c] then
4      continue
5    for i=currentLimit(c) to 0 do
6      index=position(c)[i]
7      if table(c,x)[index]∉dom(x) then
8        Swap position(c)[i] and position(c)[currentLimit(c)]
```

```
9            currentLimit(c)=currentLimit(c)−1
10 if currentLimit(c)=−1 then
11    throw inconsistency
12 Xevt←∅
13 foreach x∈scp(c) do
14    if x is assigned then
15       continue
16    size=|dom(x)|
17    dom(x)←∅
18    for i=0 to currentLimit(c) do
19       index=position(c)[i]
20       if table(c,x)[index]∉dom(x) then
21          dom(x)=dom(x)∪a
22          if |dom(x)|=size then
23             break
24    if |dom(x)|≠size then
25       setStamp(x)
26       Xevt=Xevt∪x
27 setStamp(c)
28 return Xevt
```

算法 2. *setStamp*(*m*:index of variable or constraint).

```
1  time=time+1
2  stamp[m]=time
```

2.1 复杂度分析

性质 1. 对于包含 n 个元组的 r 元约束 c,STR2*的时间复杂度为 $O(r+(Sval+Ssup)\times n)$,其中,$Sval$ 表示 c 中满足 $stamp[x]>stamp[c]$的个数,$Ssup$ 表示 c 中未赋值变量的个数.

证明:算法 1 的第 2 行和第 12 行的时间复杂度为 $O(r)$,检测元组集关于一个变量有效性的复杂度为 $O(n)$,因此,第 1 部分的时间复杂度为 $O(r+Sval\times n)$.在剩余的有效元组集上,为一个变量查找支持的复杂度为 $O(n)$,因此,第 2 部分的时间复杂度为 $O(r+Ssup\times n)$.STR2*的总时间复杂度为 $O(r+(Sval+Ssup)\times n)$. □

3 实验结果及分析

我们在非二元正表约束上测试 STR2*与现有 STR 算法的性能.测试所用实例均来自 CSP 求解大赛,可在 http://www.cril.univ-artois.fr/~lecoutre/#下载.我们测试了正表实例,并将部分包含负表的实例转换成正表测试.程序采用 java 编写,运行环境为 Intel i5@3.2Ghz 处理器、4GB 内存、windows10 64 位操作系统.回溯搜索采用 *dom*/*ddeg* 的变量启发式、字典序的值启发式和 dom^v 的 *revise* 启发式[24].每个实例的求解时间上限为 600s,T/O 表示求解时间超过 600s,所有算法均在找到第 1 组解或判定无解后结束.

表 1 分别给出了 STR2*、STR2 和 STR3 在多组实例集上的平均运行结果.#表示每组实例集测试的实例的个数;*STR*2*、*STR*2 和 *STR*3 分别表示 3 种算法的平均运行时间,单位是 s;*avgTuple* 是回溯搜索中表约束的平均大小;*avgP* 是 *avgTuple* 与初始表约束大小的比值;*avgSval* 是回溯搜索中 *Sval* 的平均大小.在图 2 的散点图中,每个点表示一个实例.

Table 1 Mean results of different algorithms

表 1 不同算法的平均结果

Instances	#	*STR2**	*STR2*	*STR3*	*avgTuple*	*avgSval*	*avgP*
rand-3-20	50	**32.744**	61.268	54.079	222.906	1.753	0.078
rand-3-20-fcd	50	**16.343**	34.709	29.868	224.488	1.752	0.078
rand-3-24	15	**195.914**	4 T/O	4 T/O	337.876	1.741	0.067
rand-3-24-fcd	17	**80.814**	196.757	168.854	329.807	1.738	0.066
rand-5-12	50	**4.431**	20.799	**4.680**	3 077.072	1.998	0.247
rand-8-20	20	**4.000**	8.512	124.692	212.903	2.258	0.002
rand-10-20	20	**0.024**	0.052	0.132	451.551	2.634	0.045
rand-10-60	32	**37.639**	2 T/O	68.038	12 161.600	2.461	0.237
rand-15-23	25	**4.541**	9.306	227.140	299.917	2.360	0.002
half-7-25	19	**159.918**	5 T/O	14 T/O	373.558	2.177	0.009
rand-5-2X	50	**2.367**	9.164	5.227	2 512.614	2.536	0.100
rand-5-4X	50	**13.553**	60.470	37.335	2 518.494	2.408	0.050
MDD0.7	8	**150.795**	4 T/O	4 T/O	1 123.617	2.185	0.029
MDD0.9	9	**12.806**	59.773	42.123	3 751.315	2.188	0.096
bddsmall	35	**3.192**	7.073	M/O	4 799.433	1.692	0.083
lexVg	63	**1.025**	2.085	1.802	356.549	5.530	0.177
ukVg	42	**15.836**	39.812	20.248	1 462.250	3.839	0.237
wordsVg	65	**3.504**	7.518	6.673	655.858	5.321	0.193
ogdVg	43	22.229	2 T/O	**15.029**	5 327.973	4.066	0.330
cril	1	**188.401**	269.538	423.871	28.797	1.660	0.026
tsp-20	15	**1.689**	2.742	4.087	33.143	1.529	0.022
tsp-25	15	**26.743**	43.750	70.690	20.782	1.453	0.014
jnhSat	16	**0.383**	0.560	0.745	28.713	1.770	0.322
jnhUnsat	34	**0.164**	0.252	0.311	42.254	2.008	0.493
modifiedRenault	25	**19.715**	35.547	31.741	284.347	2.395	0.205
varDimacs	9	36.421	**36.373**	78.712	2.437	1.827	0.497
aim-50	24	**0.062**	**0.063**	0.091	3.405	1.453	0.576
aim-100	14	43.092	**42.630**	63.289	3.516	1.361	0.593
aim-200	8	**20.403**	21.977	36.547	4.521	1.165	0.761

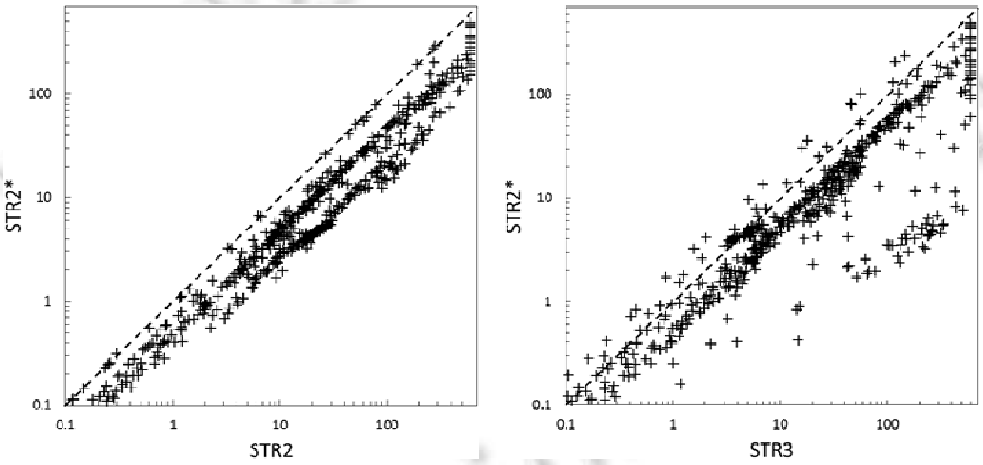

Fig.2 Comparisons of STR2 and STR2*, STR3 and STR2* in duration for finding a solution

图 2 STR2 和 STR2*、STR3 和 STR2*的求解时长的对比

- STR2* VS STR2

STR2*采用全新的动态维持元组集有效部分的方法维持GAC的效率显著高于STR2.当*avgTuple>>avgSval*时,STR2*相对于 STR2 拥有 4 倍以上的效率提升.对于一些表约束规模极小的实例,*avgTuple* 近似等于 *avgSval*, STR2 和 STR2*维持元组集有效部分的代价都很低,两种算法的效率近似.表 1 和图 2 中左图可以看出,STR2*整体优于 STR2,在大多数实例上有 2 倍~4 倍的效率提升.

- STR2* VS STR3

在一些表约束规模极小的问题上,STR2*得益于运用的数据结构简单维护代价较低,求解效率高于 STR3.

在多数实例上,随着搜索的向下进行,元组集规模缩减较快,*avgP*<30%,STR2*优于 STR3;在少数元组集缩减较慢的实例上,*avgP*>30%,并且表约束规模足够大时,STR3 优于 STR2*.从表 1 和图 2 中右图可以看出,在绝大多数实例上,STR2*优于 STR3;在部分实例上,STR2*的效率是 STR3 的 20 倍以上.

我们同样将 STR2*与基于表压缩的 STR 算法进行了对比,基于表压缩的 STR 采用分目前公认性能最优的 STRbit.表 2 中,*STR2*和 *STRbit* 分别表示 3 种算法的平均运行时间,单位是 s.图 3 是 STR2*和 STRbit 效率对比的散点图,每个点代表一个问题实例.

Table 2 Mean results of STR2* and STRbit

表 2 STR2*和 STRbit 的平均结果

Instances	#	*STR2**	*STRbit*
rand-3	132	50.355	**29.926**
rand-5	177	23.426	**17.080**
rand-8-20	20	**4.000**	9.182
rand-10-20	20	**0.024**	0.047
rand-10-60	32	37.639	**18.000**
rand-15-23	25	**4.541**	14.174
half-7-25	19	159.918	**143.964**
bddsmall	35	**3.192**	8.401
crossword	213	8.935	**2.762**
cril	1	**188.401**	196.197
tsp	30	**13.719**	18.916
jnh	50	0.231	**0.188**
modifiedRenault	25	19.715	**14.758**
varDimacs	9	**36.421**	51.415
dubois	13	**142.515**	177.520
aim	65	**122.287**	148.462

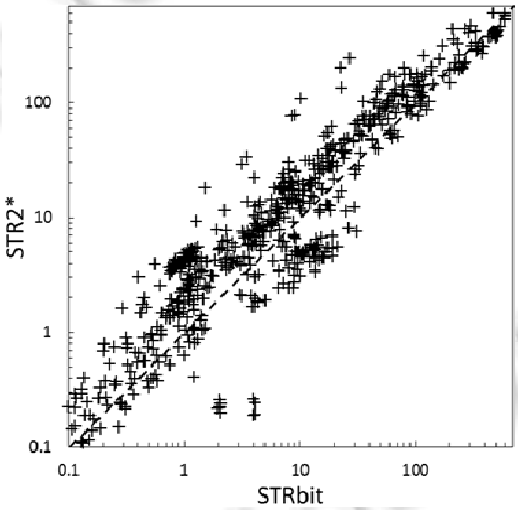

Fig.3 Comparisons of STRbit and STR2* in duration for finding a solution

图 3 STRbit 和 STR2*的求解时长的对比

- STR2* VS STRbit

在一些拓扑结构复杂且表约束规模较小的实例上,例如 aim、varDimacs 和 dubois,两种算法维持元组集有效部分的代价都比较低,STR2*得益于运用的数据结构简单维护代价较低,求解效率稍优于 STRbit.在一些元组集规模缩减较快的实例上,例如 rand-15-23、rand-8-20 和 bddsmall,STR2*优于 STRbit,拥有超过 2 倍的效率提升.表 2 可以看出,STR2*在所有 16 类问题的 9 类中优于 STRbit.由于不同类问题的实例集中的实例个数差别较大,例如 STRbit 效率更高的 rand-3、rand-5 和 crossword 这 3 类问题的实例总个数占到所有测试实例的 60%,

在图 3 的散点图中,才会有 STRbit 在多数实例上效率优于 STR2*的效果.

4 总 结

我们提出了一种表约束上维持 GAC 的算法 STR2*,采用了一种新的动态维持元组集有效部分的方法.实验结果证实,STR2*维持 GAC 的效率均高于 STR2 和 STR3,并且在元组集缩减较快的问题以及元组集规模较小的问题上优于最新的采用表压缩的 STRbit.今后,我们希望将这种新的动态维持元组集有效部分的方法运用到其他基于 STR 的算法中.

References:

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

[2] Ullmann JR. Partition search for non-binary constraint satisfaction. Information Science, 2007,177:3639−3678.

[3] Briggs P, Torczon L. An efficient representation for sparse sets. ACM Letters on Programming Languages and Systems, 1993, 2(1-4):59−69.

[4] de Saint-Marcq VLC, Schaus P, Solnon C, Lecoutre C. Sparse-sets for domain implementation. In: Proc. of the CP Workshop on Techniques for Implementing Constraint Programming Systems (TRICS). 2013. 1−10.

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

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

[7] Lecoutre C, Likitvivatanavong C, Yap RHC. A path-optimal GAC algorithm for table constraints. In: Proc. of the 20th European Conf. on Artificial Intelligence. 2012. 510−515.

[8] 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. 2014. 497−513.

[9] 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]

[10] Lecoutre C, Likitvivatanavong C, Yap RHC. Improving the lower bound of simple tabular reduction. Constraints, 2015,20(1): 100−108.

[11] Cheng K, 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.

[12] Xia W, Yap RHC. Optimizing STR algorithms with tuple compression. In: Proc. of the Int'l Conf. on Principles and Practice of Constraint Programming. 2013. 724−732.

[13] Katsirelos G, Walsh T. A compression algorithm for large arity extensional constraints. In: Proc. of the Int'l Conf. on Principles and Practice of Constraint Programming. 2007. 379−393.

[14] Jefferson C, Nightingale P. Extending simple tabular reduction with short supports. In: Proc. of the 23rd Int'l Joint Conf. on Artificial Intelligence. 2013. 573−579.

[15] Gharbi N, Hemery F, Lecoutre C, Roussel O. Sliced table constraints: Combining compression and tabular reduction. In: Proc. of the 11th Int'l Conf. on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming. 2014. 120−135.

[16] Wang R, Xia W, Yap RHC, Li Z. Optimizing simple table reduction with bitwise representation. In: Proc. of the Int'l Joint Conf. on Artificial Intelligence. 2016. 787−795.

[17] 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. Springer Int'l Publishing, 2016. 207−223.

[18] Karakashian S, Woodward R, Reeson C, Choueiry BY, Bessiere C. A first practical algorithm for high levels of relational consistency. In: Proc. of the Conf. on Artificial Intelligence (AAAI 2010). 2010. 101−107.

[19] Shant K, Woodward RJ, Choueiry BY. Improving the performance of consistency algorithms by localizing and bolstering propagation in a tree decomposition. In: Proc. of the AAAI. 2013.

[20] Mairy JB, Deville Y, Lecoutre C. Domain *k*-wise consistency made as simple as generalized arc consistency. In: Proc. of the CPAIOR 2014. 2014. 235−250.

[21] Lecoutre C, Paparrizou A, Stergiou K. Extending STR to a higherorder consistency. In: Proc. of the AAAI 2013. Washington, 2013. 576−582.

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

[23] 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.

[24] Boussemart F, Hemery F, Lecoutre C. Revision ordering heuristics for the constraint satisfaction problem. In: Proc. of the CPAI 2004 Workshop Held with CP 2004. 2004. 29−43.

附中文参考文献:

[9] 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]

杨明奇(1992－),男,山东枣庄人,硕士,CCF 学生会员,主要研究领域为约束求解.

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求解,机器学习,智能规划与调度,基于模型的诊断.

张家晨(1969－),男,博士,教授,CCF 高级会员,主要研究领域为软件维护与演化,软件自动生成方法与技术,自动推理.