Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence

Making Simple Tabular Reduction Works on Negative Table Constraints

Hongbo Li, Yanchun Liang, Jinsong Guo and Zhanshan Li*
Key Laboratory for Symbol Computation and Knowledge Engineering of National Education Ministry,
College of Computer Science and Technology, Jilin University, Changchun, 130012, China.

Abstract

Simple Tabular Reduction algorithms (STR) work well to establish Generalized Arc Consistency (GAC) on positive table constraints. However, the existing STR algorithms are useless for negative table constraints. In this work, we propose a novel STR algorithm and its improvement, which work on negative table constraints. Our preliminary experiments are performed on some random instances and a certain benchmark instances. The results show that the new algorithms outperform GAC-valid and the MDD-based GAC algorithm.

Introduction

It is widely recognized that the MAC algorithm [Sabin and Freuder 1994], which maintains Arc Consistency during the search of first solution, is quite efficient to solve hard and large CSP instances. The GAC-valid algorithm [Bessière and Régin 1997] is a universal algorithm to establish Generalized Arc Consistency (GAC) on all kinds of constraints. Table constraints explicitly list all the tuples that are either allowed or disallowed by them. A table constraint is said to be negative if the tuples in its table are disallowed by it, otherwise positive. To save space, loose constraints are usually represented by negative table constraints and tight constraints are represented by positive ones. Exploiting the structure of table constraints, the Simple Tabular Reduction (STR) algorithms [Ullmann 2007; Lecoutre 2011; Lecoutre et al. 2012] work quite well to establish GAC on positive table constraints. However, the existing STR algorithms can not directly work on negative table constraints. In this work, we propose a novel Simple Tabular Reduction algorithm, called STR-Negative (STR-N), which directly works on negative table constraints. The STR-N algorithm also dynamically maintains tables for each constraint and iterates over all the tuples in the tables. The main difference between the existing STR algorithms (STR-P) and the STR-N algorithm is how to determine whether a value has supports on the constraint. Besides, we give an improvement of original STR-N. Our preliminary results show that the STR-N algorithms works well on random instances and a certain benchmark instances.

*Corresponding author: lizs@jlu.edu.cn

Simple Tabular Reduction for Negative Table Constraints

A negative table constraint c consists of two parts, an ordered set of variables $scp(c) = \{x_{i1}, x_{i2}, \ldots, x_{ir}\}$ and a subset of the Cartesian product $D_{i1} \times D_{i1} \times \ldots \times D_{i1}$, $table(c)$, which specifies the disallowed combinations of values for the variables $\{x_{i1}, x_{i2}, \ldots, x_{ir}\}$, where D_{ij} is the domain of x_{ij}. An element of $D_{i1} \times D_{i1} \times \ldots \times D_{i1}$ is called a tuple on $scp(c)$, denoted by τ. For a negative table constraint c, a tuple τ is disallowed by c if it belongs to $table(c)$ and it is allowed by c if it does not belong to $table(c)$. A tuple τ is valid iff all values in this tuple are present in the domains of the corresponding variables.

Algorithm 1 STR-NEGATIVE

Input : c, a negative table constraint;
Output: variables whose domains have been reduced;

```
1.  if lastIndex(c) = 0 then return ∅
2.  reduced(X) ← ∅
3.  foreach x ∈ scp(c) and x is not assigned do
4.     compute count(x, a, c) for x
5.  index ← 1
6.  while index ≤ lastIndex(c) do
7.     τ ← table(c)[index]
8.     if τ is valid then
9.        foreach (x, a) ∈ τ and x is not a past variable do
10.          count(x, a, c) ← count(x, a, c) - 1
11.       index ← index +1
12.    else removeTuple(c, index)
13. foreach x in scp(c) and x is not assigned do
14.    foreach (x, a) ∈ D_x do
15.       if count(x, a, c) = 0 then
16.          remove (x, a) from D_x
17.          reduced(X) ← reduced(X) ∪ {x}
18.          if D_x = ∅ then throw inconsistency
19. return reduced(X)
```

Given a negative table constraint c, $x \in scp(c)$, if there exists a valid tuple τ involving a value (x, a) and $\tau \notin table(c)$, then (x, a) has at least one support on c. We can prove such tuples exist by the following steps:

First, we can obtain the number of all valid tuples involving (x, a) on $scp(c)$, denoted by $count(x, a, c)$, which is the

product of the current domain sizes of all the variables in $scp(c)$ except for x. Then we iterate over all tuples of $table(c)$, if a tuple is verified to be valid, for each value (x, a) involved in this tuple, $count(x, a, c)$ is decremented by 1. Finally, if $count(x, a, c)$ is greater than 0, then value (x, a) has at least one support on constraint c.

The STR-Negative algorithm is depicted in Algorithm 1. Initially, the $count(x, a, c)$ records the number of all valid tuples involving (x, a). After iterating $table(c)$, the number of disallowed tuples are subtracted. Finally, $count(x, a, c)$ records the number of valid and allowed tuples. The $lastIndex(c)$ points to last tuple in $table(c)$ that has not been verified to be invalid. Whenever a tuple is verified to be invalid, the removeTuple(c, $index$) first swaps $table(c)[index]$ and $table(c)[lastIndex(c)]$, then decrements $lastIndex(c)$ by 1.

Initially, for each variable $x_i \in scp(c)$, each $(x, a) \in D_i$, if the minimal $count(x_i, a, c)$ is greater than the number of all currently valid tuples in $table(c)$, then for each variable $x_i \in scp(c)$, each $(x_i, a) \in$ current D_i has at least one support on c. Therefore, we can avoid iterating tuples on such constraints. The algorithm with such improvement is called STR-Ni.

Preliminary Experimental Results

We have implemented some MAC solvers which use GAC-valid, mddc [Cheng and Yap 2010], original STR-N and STR-Ni as the GAC algorithms respectively. Each solver is equipped with dom/ddeg [Bessière and Régin 1996] as variable ordering heuristic and the binary branching is employed. The GAC-valid (GAC-v) solver is also equipped with the multi-directional residues technique [Lecoutre and Hemery 2007]. Firstly, the experiments are performed on hard random instances of Model RB [Xu and Li 2000] situated at the phase transition of search. These instances are generated by the RBGenerator from http://www.cril.univ-artois.fr/~lecoutre/benchmarks.html and are forced to be satisfiable [Xu et al. 2005]. For each class $<r, n, d, e, p>$, where r is the arity of the constraints, n is variables number, d is the unique domain size of the variables, e is constraints number and p is constraints tightness, 100 instances have been tested. The average results are listed in Table 1.

$<r, n, d, e, p>$	mddc	GAC-v	STR-N	STR-Ni
3, 20, 10, 200, 0.206	21.06	23.15	15.4	15.62
3, 20, 10, 300, 0.142	48.33	55.98	31.84	31.53
3, 50, 5, 200, 0.331	17.85	18.89	13.59	13.96
3, 50, 5, 300, 0.235	135.93	140.36	95.08	94.80
5, 20, 5, 200, 0.149	225.58	189.42	116.41	109.38
5, 20, 5, 300, 0.102	407.88	410.41	208.90	186.60
8, 20, 3, 200, 0.104	96.90	73.99	33.16	31.17
8, 20, 3, 300, 0.071	145.74	147.39	54.97	52.00
10, 20, 3, 200, 0.104	1155.18	694.71	313.94	300.30
8, 20, 3, 200, 0.104	2312.29	1330.53	488.78	466.92

Table 1: Results of random instances

Next, we performed experiments on Chessboard Coloration problem and Golomb Ruler problem, which involve non-binary negative table constraints. These benchmark instances are downloaded from http://www.cril.univ-artois.fr/~lecoutre/benchmarks.html. We have tested all the 20 Chessboard Coloration instances and 14 golombRulerArity4 instances. Table 2 presents the results for a certain instances which cost more than 1 second and less than 1800 seconds.

Instance	mddc	GAC-v	STR-N	STR-Ni
cc-7-7-3	18.62	25.41	12.28	11.56
cc-10-10-2	6.64	7.93	6.13	5.71
cc-12-12-2	66.00	70.59	58.77	55.93
ruler-25-8-a4	3.44	3.15	4.09	2.29
ruler-34-9-a4	59.11	51.05	75.27	40.16
ruler-44-10-a4	908.87	690.04	1213.47	608.94

Table 2: Results of a certain benchmark instances

Conclusion

In this paper, we propose a novel STR algorithm suitable for negative table constraints and its improvement. Experimental results show that the STR-N algorithms work well on some random and benchmark instances.

Acknowledgments

This work was supported in part by the China NSFC (61073075, 60873148) and the Science-Technology Development Project from Jilin Province of China (20120730).

References

Sabin, D. and Freuder, E.C. 1994. Contradicting conventional wisdom in constraint satisfaction. In *Proc. of ECAI'94*, 125–129.

Bessière, C. and Régin, J.C. 1997. Arc consistency for general constraint networks: preliminary results. In *Proc. of IJCAI'97*, 398–404.

Ullmann, J.R. 2007. Partition search for non-binary constraint satisfaction. *Information Science*. 177: 3639–3678.

Lecoutre, C. 2011. STR2: Optimized Simple Tabular Reduction for Table Constraints. *Constraints*. 16: 341–371.

Lecoutre, C., Likitvivatanavong, C. and Yap, C. 2012. A path-optimal GAC algorithm for table constraints. In *Proc. of ECAI'12*, 510-515.

Cheng, K.C. and Yap, R.H. 2010. An MDD-based generalized arc consistency algorithm for positive and negative table constraints and some global constraints. *Constraints*. 15(2): 265–304.

Lecoutre, C. and Hemery, F. 2007. A study of residual supports in arc consistency. In *Proc. of IJCAI'07*, 125–130.

Bessière, C. and Régin, J.C. 1996. MAC and combined heuristics: two reasons to forsake FC (and CBJ?) on hard problems. In *Proc. of CP'96*, 61–75.

Xu, K. and Li, W. 2000. Exact phase transitions in random constraint satisfaction problems. *Journal of Artificial Intelligence Research*. 12: 93–103.

Xu, K. , Boussemart, F. , Hemery, F. , Lecoutre, C. 2005. A simple model to generate hard satisfiable instances. In *Proc. of IJCAI'05*, 337-342.