Relational Consistency by Constraint Filtering

Shant Karakashian,
Robert J. Woodward, Berthe Y. Choueiry
Constraint Systems Laboratory
Department of Computer Science & Engineering
University of Nebraska-Lincoln
{shantk|rwoodwar|choueiry}@cse.unl.edu

Christian Bessiere
LIRMM-CNRS
University Montpellier, France
bessiere@lirmm.fr

ABSTRACT

In this paper, we propose a new algorithm for enforcing relational consistency on every set of k constraints of a finite Constraint Satisfaction Problem (CSP). This algorithm operates by filtering the constraint while leaving the topology of the graph unchanged. We study the resulting relational consistency property and compare it to existing ones. We evaluate the effectiveness of our algorithm in a search procedure for solving CSPs and demonstrate the applicability, effectiveness, and usefulness of enforcing high levels of consistency.

1. INTRODUCTION

Advancing the early work on the topic [11, 9], Dechter and van Beek [5] defined *relational consistency* for the consistency of non-binary Constraint Satisfaction Problems (CSPs). They defined the property *relational m-consistency* (RmC) to apply to every combination of m constraints in a CSP, and the more relaxed property of relational (i, m)-consistency (R(i, m)C) to apply to every combination of i variables in the scope of the m constraints. In practice, enforcing RmC or R(i, m)C may require adding $\mathcal{O}(n^i)$ non-binary constraints to the constraint network, where n is the number of variables in the CSP, thus changing its topology. Another research direction focused on the effect of consistency on the domains of the variables [10, 1, 8, 2, 3]. *Domain filtering* reduces the search space explored for solving the CSP. While most work considered constraints *individually* (GAC), [2, 12] studied the effects of combinations of pairs of constraints.

In this paper we introduce a special form of relational consistency, which we call R(∗,m)C and which operates on every combination of m constraints. The techniques explored in [12] correspond to R(∗,2)C, but are not designed to handle R(∗,m)C for $m > 2$. Unlike RmC and R(i, m)C, R(∗,m)C does *not* add new constraints to the CSP and, thus, keeps the topology and width of the network unchanged. Instead, it filters the relations defining the constraints to remove inconsistent tuples. In association with R(i,m)C, the '∗' in R(∗,m)C is used to indicate that the property is not concerned with the number of variables but only with the constraints 'whatever is the size of their scope.'

More formally, we define R(∗,m)C to ensure that every tuple in a relation can be extended to a partial solution over the variables in every set of m constraints. R(∗,m)C is semantically equivalent to relational m-wise consistency studied in [6, 7]. However, neither paper evaluated or even proposed practical algorithms for implementing relational m-wise consistency. Our contributions in this work are the (re-)definition of a new (parametric) relational consistency property, R(∗,m)C, that does not modify the topology of the constraint network, the design of the algorithm for efficiently enforcing it, and its empirical evaluation on benchmark problems.

SAC'10 March 22-26, 2010, Sierre, Switzerland.

2. BASIC DEFINITIONS

A Constraint Satisfaction Problem (CSP) is defined by $(\mathcal{V}, \mathcal{D}, \mathcal{C})$ where $\mathcal{V}$ is a set of variables, $\mathcal{D}$ set of domains, and $\mathcal{C}$ set of constraints. Each variable $V_i \in \mathcal{V}$ has a finite domain $D_i \in \mathcal{D}$, and is constrained by a subset of the constraints in $\mathcal{C}$. The scope of a constraint is given by $vars(C_i)$. Every constraint C_i is associated with a relation R_i that lists the allows combinations of values for the variables in $vars(C_i)$. A solution for the CSP is an assignment of a value for every variable such that all the constraints are satisfied simultaneously. We denote by φ a *combination of m constraints* such that the primal graph induced by the constraints in the combination is connected. Two constraints are connected if they have at least one common variable in their scope [4]. Finally, π and $\bowtie$ denote the relational operators project and join, respectively.

3. DEFINITION OF R(∗,M)C

We define R(∗,m)C using the definition format of R(i,m)C [4]:

DEFINITION 3.1. *A set of m relations $\mathcal{R} = \{R_1, \cdots, R_m\}$ is said to be R(∗,m)C iff every tuple in each relation $R_i \in \mathcal{R}$ can be extended to the variables in $\bigcup_{R_j \in \mathcal{R} \setminus R_i} vars(R_j)$ in an assignment that satisfies all the relations in $\mathcal{R}$ simultaneously. A network is R(∗,m)C iff every set of m relations is R(∗,m)C.*

R(∗,m)C can thus be enforced by filtering the existing relations using the following operation on each combination of m relations $\{R_1, \cdots, R_m\}$ and without introducing to the CSP any relation whose scope was not already constrained in the original CSP: $\forall R_i \in \{R_1, \cdots, R_m\}, R_i \subseteq \pi_{vars(R_i)}(\bowtie_{j=1}^{m} R_j)$

This expression gives us an obvious algorithm for enforcing R(∗,m)C. Even if each join is computed only once and then its tuples filtered iteratively, the space requirement of such an operation is too prohibitive to be useful in practice.

Once R(∗,m)C is enforced on a constraint network, variable domains can subsequently be filtered (i.e., domain filtering) by simple projection of the filtered relations on the domains of the variables. Unlike GAC, we do not need to loop between the filtering of the domains and that of the constraints because, on a network that is R(∗,m)C, domain filtering cannot enable further constraint filtering by R(∗,m)C. R(∗,m)C does not eliminate any solution, and is is a weaker consistency, in terms of pruning power and consistency, than relational m-consistency. We omit the proofs for lack of space.

4. AN ALGORITHM FOR R(∗,M)C

We achieve the R(∗,m)C property when every tuple τ of every relation R_i in every combination φ of m relations can be 'extended' successfully to all the $(m-1)$ remaining relations in φ, that is all tuples have the same values for the common variables. We say that the set of $(m-1)$ tuples that 'extends' τ to the constraints in φ is the 'support' of R_i's τ in φ. A tuple is deleted when it has no support in at least one combination.

Algorithm 1 enforces R(∗,m)C by processing the combination-relation queue. When a pair $\langle \varphi, R \rangle$ is poped from the queue, every tuple in the relation R is extended to all the remaining relations in φ in the loop in Line 4. The tuple that is not extended, is deleted. When a relation loses a tuple, all the other relations that appear in a combination with R may be effected only in the combinations with R. Hence all such relations are added back to the queue paired with the combinations that include R in Line 12. This is the reason for having combination-relation pairs in the queue.

The inputs to the algorithm are the initial queue $\mathcal{Q}$ that is initialized to all the combination-relation pairs, and the set of all combination ζ. The algorithm terminates when a relation loses all the tuples and returns inconsistent, or verifies that all the remaining tuples can be extended and returns consistent. The FINDSUPPORT in

Algorithm 1: PROCESSQUEUEEnforces the R(∗,m)C property

```
   Input: Q, ζ
   Output: true if the problem is R(*,m)C, false otherwise
1  while (Q ≠ ∅) do
2      ⟨φ, R⟩ ← POP(Q)
3      deleted ← false
4      foreach τ ∈ R do
5          support ←FINDSUPPORT((τ, R), φ)
6          if support = false then
7              DELETE(τ)
8              if R = ∅ then return false
9              deleted ← true
10     if deleted then foreach φ' ∈ (ζ \ {φ}) do
11         if R ∈ φ' then foreach R' ∈ (φ' \ {R}) do
12             Q ← Q ∪ {⟨φ', R'⟩}
13 return true
```

Line 5 utilizes a special data structure to avoid the scanning of the complete relation when searching for a support. We do not describe the data structure here for lack of space.

The complexity of Algorithm 1 is $\mathcal{O}(t^2e^2\delta m + mt^m)$ where t is the maximum number of tuples in a relation, e is the total number of relations, and δ in the number of combinations.

5. EXPERIMENTAL RESULTS

We conducted the experiments on benchmarks selected from the CPAI08 competition[1]. We compare R(∗,m)C to GAC2001/3.1 [1] and maxRPWC-1 [2]. R(∗,m)Ci is the algorithm with a special feature enabled on the indices of the data structures, which is not described here for lack of space. The results are reported in Table 1. We report the number of nodes visited (#NV), average and maximum CPU time in seconds over instances completed by all algorithms, number of instances completed, and the number of instances completed fastest. The time limits were set to 20 minutes for Renault, three hours for rand-8-20-5, and one hour for aim-200. We solve the CSPs by using a backtrack search procedure with the 'domain over degree' dynamic variable-ordering heuristic. The search maintains the consistency properties by applying full lookahead.

The results show that R(∗,m)C is able to solve more problems within the time limit, and has the lowest maximum time among the problems solved by all algorithms.

[1]http://cpai.ucc.ie/08/

Table 1: Results on benchmark problems.

Algorithm	#NV	Time	Max time	Comp.	Fst.
Renault benchmark					
GAC	300,195.33	66.83	610.41	21	19
maxRPWC	1,140.56	115.66	215.43	29	0
R(*,2)C	**100.28**	**10.83**	**12.48**	**46**	**28**
rand-10-20-10 benchmark (all unsatisfiable)					
GAC	210.10	7.19	9.54	20	0
maxRPWC	**0.00**	1.57	3.94	20	0
R(*,2)C	**0.00**	**0.20**	**0.23**	20	**20**
rand-8-20-5 benchmark (all satisfiable)					
GAC	60,273.27	3,527.00	10,072.60	15	3
maxRPWC	-	-	-	0	0
R(*,2)C	**1,307.67**	3,188.08	10,945.70	18	1
R(*,2)Ci	**1,307.67**	**2,073.04**	**5,549.97**	18	14
aim-200 benchmark					
GAC	1.9M	554.80	2034.54	7	2
maxRPWC	850K	**386.39**	1365.64	7	3
R(*,2)C	5.5K	520.36	1971.79	12	**7**
R(*,3)C	**222**	880.24	**925.80**	**13**	3

6. CONCLUSIONS

We presented an algorithm to enforce a *parametrized* relational consistency property. This property, unlike most other well-studied consistency properties, is enforced by tightening the existing constraints and without introducing any additional ones. Importantly, we empirically evaluated our algorithm on difficult benchmark problems and demonstrated its effectivness.

Acknowledgments

The authors acknowledge the help of Kostas Stergiou and the feedback of anonymous reviewers. Experiments were conducted on the Holland Computing Center at UNL. Karakashian was partially supported by NSF CAREER Award #0133568, and Woodward by a UCARE grant UNL.

7. REFERENCES

[1] C. Bessière, J.-C. Régin, R. H. Yap, and Y. Zhang. An Optimal Coarse-Grained Arc Consistency Algorithm. *Artificial Intelligence*, 165(2):165–185, 2005.

[2] C. Bessière, K. Stergiou, and T. Walsh. Domain Filtering Consistencies for Non-Binary Constraints. *Artificial Intelligence*, 172:800–822, 2008.

[3] K. C. K. Cheng and R. H. C. Yap. Maintaining Generalized Arc Consistency on Ad Hoc r-Ary Constraints. In *CP 08*, volume LNCS 5202, pages 509–523. Springer, 2004.

[4] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[5] R. Dechter and P. van Beek. Local and Global Relational Consistency. *Theor. Comput. Sci.*, 173(1):283–308, 1997.

[6] M. Gyssens. On the Complexity of Join Dependencies. *T. ACM Trans. Database Systems*, 11 (1):81–108, 1986.

[7] P. Jégou. On the Consistency of General Constraint-Satisfaction Problems. In *AAAI 1993*, pages 114–119, 1993.

[8] O. Lhomme and J.-C. Régin. A Fast Arc Consistency Algorithm For N-Ary Constraints. In *AAAI 2005*, pages 405–410, 2005.

[9] A. K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8:99–118, 1977.

[10] R. Mohr and G. Masini. Good Old Discrete Relaxation. In *ECAI 88*, pages 651–656, 1988.

[11] U. Montanari. Networks of Constraints: Fundamental Properties and Application to Picture Processing. *Inf. Sciences*, 7:95–132, 1974.

[12] K. Stergiou and N. Samaras. Binary Encodings of Non-binary Constraint Satisfaction Problems: Algorithms and Experimental Results. *JAIR*, 24:641–684, 1998.