University of Nebraska - Lincoln
DigitalCommons@University of Nebraska - Lincoln

CSE Conference and Workshop Papers | Computer Science and Engineering, Department of

2010

A First Practical Algorithm for High Levels of Relational Consistency

Shant Karakashian
University of Nebraska - Lincoln, shantk@cse.unl.edu

Robert J. Woodward
University of Nebraska - Lincoln, rwoodwar@cse.unl.edu

Christopher Reeson
University of Nebraska - Lincoln, creeson@cse.unl.edu

Berthe Y. Choueiry
University of Nebraska - Lincoln, choueiry@cse.unl.edu

Christian Bessiere
University of Montpellier, France, bessiere@lirmm.fr

Follow this and additional works at: http://digitalcommons.unl.edu/cseconfwork

Part of the Computer Sciences Commons

Karakashian, Shant; Woodward, Robert J.; Reeson, Christopher; Choueiry, Berthe Y.; and Bessiere, Christian, "A First Practical Algorithm for High Levels of Relational Consistency" (2010). *CSE Conference and Workshop Papers*. 175.
http://digitalcommons.unl.edu/cseconfwork/175

This Article is brought to you for free and open access by the Computer Science and Engineering, Department of at DigitalCommons@University of Nebraska - Lincoln. It has been accepted for inclusion in CSE Conference and Workshop Papers by an authorized administrator of DigitalCommons@University of Nebraska - Lincoln.

A First Practical Algorithm for High Levels of Relational Consistency

Shant Karakashian, Robert Woodward, Christopher Reeson, Berthe Y. Choueiry & Christian Bessiere

Constraint Systems Laboratory, University of Nebraska-Lincoln

LIRMM-CNRS, University of Montpellier

Acknowledgements:

- Kostas Stergiou and anonymous reviewers
- Experiments conducted at UNL's Holland Computing Center
- NSF CAREER Award #0133568
- Robert Woodward supported by a UNL undergraduate research grant (UCARE) & by a B.M. Goldwater Scholarship

Outline

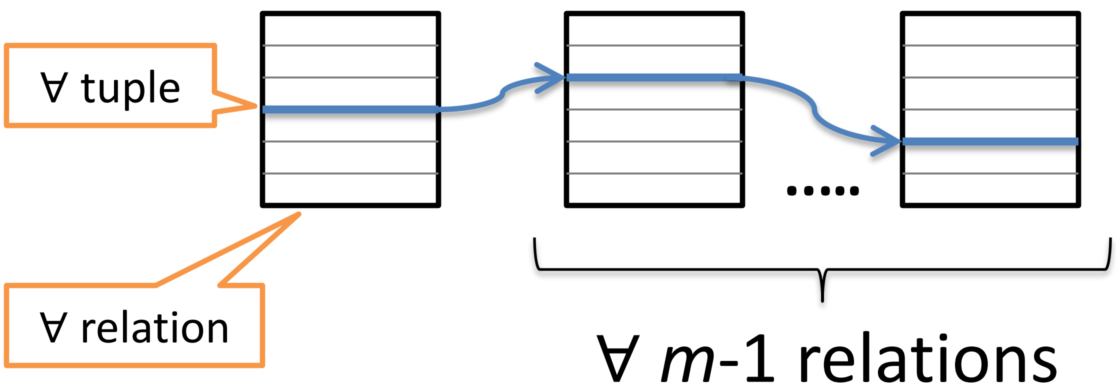
- Introduction
- Relational Consistency R(*,m)C :
  - Definition, Naïve algorithm, Properties
- Preliminaries: Dual CSP
- Our Approach
  - Algorithm
  - Index-Tree Data Structure
  - Advantages
- A weakened version of R(*,m)C: wR(*,m)C
- Experimental Evaluations
- Conclusions & Future Work

Introduction

- Local consistency techniques are at the heart of solving CSPs
- Low level consistency properties such as GAC are easy to apply & are effective for many problems
- There are problems that require higher levels of consistency for finding a solution in a reasonable amount of time
- We present a practical algorithm for enforcing relational *m*-wise consistency: R(*,*m*)C

Definition of R(*,m)C

- A CSP is R(*,m)C iff
  - Every tuple in a relation can be extended to the variables in the scope of any (m-1) other relations in an assignment satisfying all m relations simultaneously



Naïve Algorithm for R(*,*m*)C

- R(*,m)C can be enforced on a CSP by
  - joining every combination of *m* relations and
  - projecting the product on the individual relations

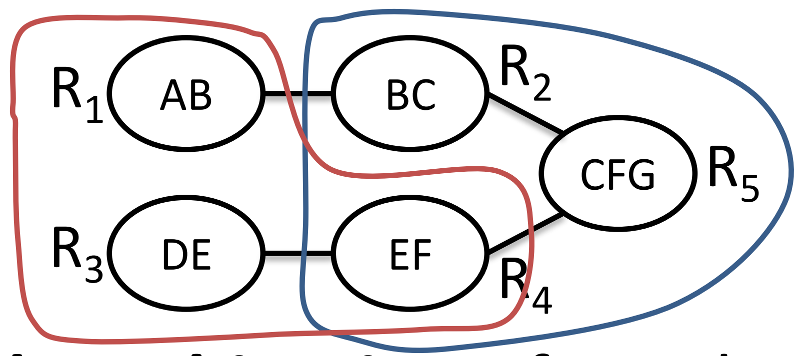
$$\forall\ R_i \in \{R_1, \ldots, R_m\},\ R_i \leftarrow \pi_{\text{scope}(R_i)}\left(\bowtie_{j=1..m} R_j\right)$$

Properties of R(*,m)C

- It does not change the structure of the constraint network
- R(*,m)C $\prec$ RmC [Dechter & van Beek '97]
- It filters the relations by removing tuples
- It is parameterized
  - We can control the level of consistency (m)

Preliminaries

- The **dual graph** of a CSP is a graph where
  - The nodes represent the relations
  - The edges are added between two relations with at least one common variable

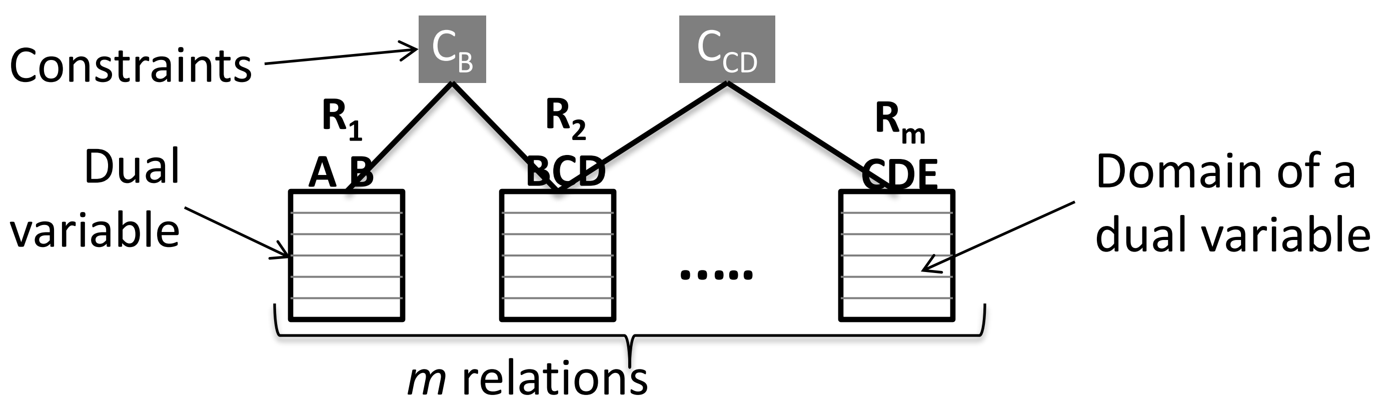


$m = 3$

- **Connected combination** of m relations is a set of relations that induce a connected component in the dual graph

The Induced Dual CSP

- Consider $\omega = \{R_1, R_2, \ldots, R_m\}$ a set of m relations
- P_ω is the dual CSP **induced** by ω where
  - The dual variables represent the m relations
  - The domains are the tuples of the relations R_i
  - The constraints in P_ω are binary & enforce equality on the CSP variables shared by the two relations



Enforcing R(*,m)C on the Induced Dual CSP P_ω

Q
$<\omega_1, R_1>$
$<\omega_1, R_2>$
$<\omega_1, R_5>$
$<\omega_2, R_2>$
$<\omega_2, R_5>$
$<\omega_2, R_4>$
$<\omega_3, R_3>$
$<\omega_3, R_4>$
$<\omega_3, R_5>$

ω_1 ω_2 ω_3

Extract $<\omega, R>$ from Q

Define CSP P_ω

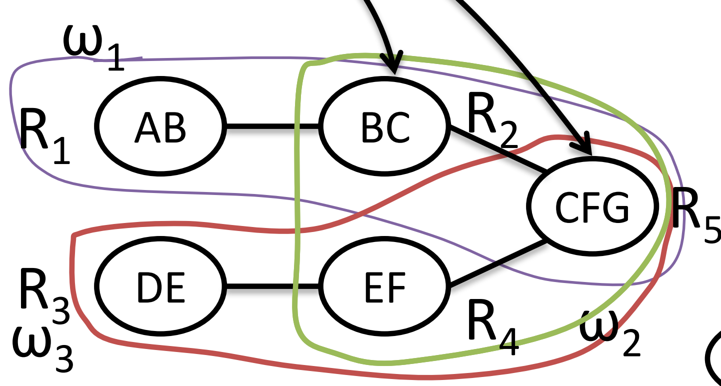
For each τ in R

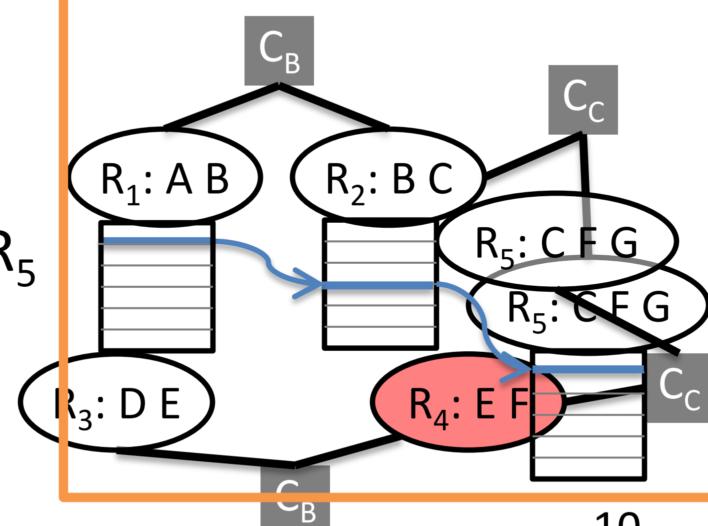
Assign τ as a value for R

Solve P_ω (with τ fixed) with forward checking

If no solution found: delete τ

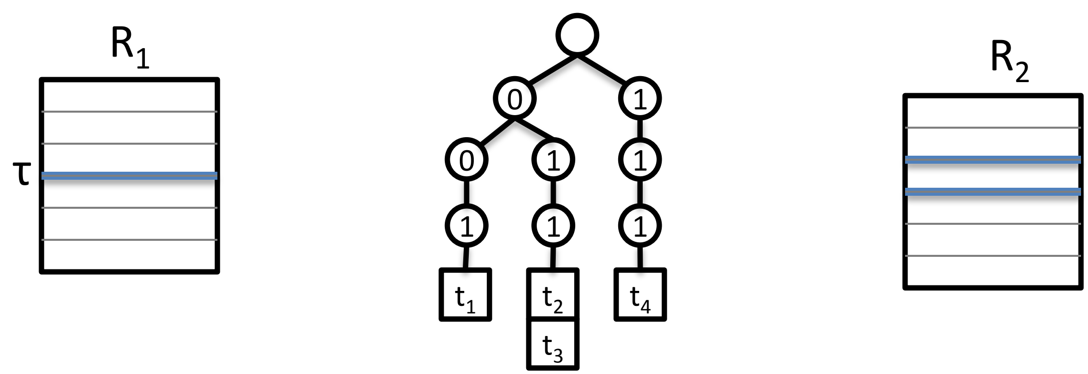
Add $<\omega', R'>$ to Q: $R_i \neq R'$, $R_i \in \omega'$ and $R' \in \omega'$





Index-Tree Data Structure

- When solving P_ω, for a tuple τ, Forward checking requires identifying all tuples matching τ in the neighboring relations
- We propose a new data structure: index-tree
  - Given a tuple τ of R_1 and a relation R_2
  - Identifies all the tuples of R_2 that match τ



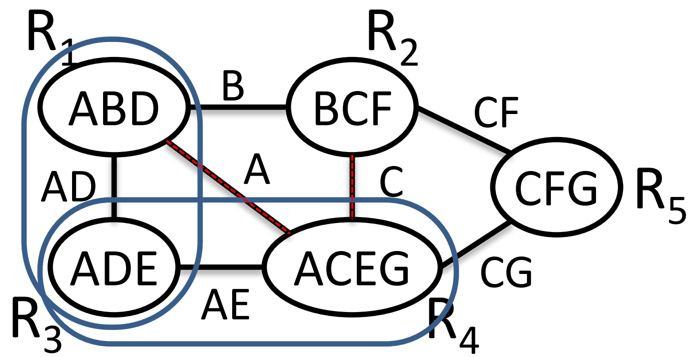
Advantages of Our Approach

- The memory requirement of the operation

  $$\forall R_i \in \{R_1, \ldots, R_m\}, R_i = \pi_{scope(Ri)} (\bowtie_{j=1..m} R_j)$$

  - $O(t^m)$, t: max number of tuples in a relation
  - For relations with 10,000 tuples, enforcing R(*,3)C requires in the order of 1TB of memory
- With our approach, the memory requirement is dominated by the index-tree structures
  - $O(kte^2)$, k: max arity of relations, e: number of relations
  - While slightly decreasing the time complexity

Weakening Relational Consistency: wR(*,*m*)C



$R_1\ R_2\ R_3$	$R_1\ R_2\ R_3$
$R_1\ R_2\ R_4$	
$R_1\ R_2\ R_5$	$R_1\ R_2\ R_5$
$R_1\ R_3\ R_4$	$R_1\ R_3\ R_4$
$R_2\ R_3\ R_4$	
$R_2\ R_4\ R_5$	$R_2\ R_4\ R_5$
$R_3\ R_4\ R_5$	$R_3\ R_4\ R_5$

- Some edges are redundant for *m*=2
- Removing them reduces the number of combinations
- For *m*>2, removal of these edges weakens R(*,m)C
- Example
  - Assume that no assignment satisfies variables A, B & C simultaneously
  - To detect this inconsistency, need to consider $R_1R_2R_4$ simultaniously
  - This inconsistency is not detected because we removed the combination $R_1R_2R_4$

R(*,m)C versus wR(*,m)C

R(*,m)C is defined for $m \geq 2$

$m = 2$	R(*,2)C ≡ wR(*,2)C [Janssen+ '89]
$m > 2$	R(*,2)C $\prec$ wR(*,m)C $\prec$ R(*,m)C
$m < n$	R(*,m)C $\prec$ R(*,n)C wR(*,m)C $\prec$ wR(*,n)C

A $\prec$ B: A is strictly weaker than B

Experimental Results

Benchmark	Algorithm	#Nodes Visited	Time [sec]	#Completed in 1 hour	#Fastest	#Backtrack Free
modifiedRenault	GAC	1,324,309.8	402.44	26	14	4/50
Max #tuples: 48,721	maxRPWC	2,110.8	305.37	31	3	19/50
	wR(*,2)C	192.5	2.99	46	**27**	41/50
	wR(*,3)C	82.5	7.55	**50**	4	48/50
	wR(*,4)C	82.5	33.88	**50**	2	**50/50**
rand-8-20-5	GAC	30,501.7	1,795.26	9	2	0/20
Max #tuples :78,799	wR(*,2)C	941.3	1,162.22	**16**	**14**	0/20
dag-rand	wR(*,2)C	0.0	27.21	**25**	**25**	**25/25**
Max #tuples: 150,000	wR(*,3)C	0.0	37.75	**25**	0	**25/25**
aim-200	GAC	1,876,247.6	542.48	8	0	0/24
Max #tuples: 7	maxRPWC	842,488.8	414.05	8	1	0/24
	wR(*,2)C	2,670.2	35.51	12	**7**	4/24
	wR(*,3)C	580.2	35.91	**14**	**7**	8/24
	wR(*,4)C	443.8	240.13	**14**	2	**9/24**

Conclusions & Future Work

- We studied the relational consistency property R(*,*m*)C
  - Proposed a weaker variant wR(*,*m*)C
  - Presented a parameterized algorithm for enforcing it
  - Designed a new data structure (index tree) for efficiently checking the consistency of tuples between two relations
  - Evaluated it against GAC & maxRPWC
- Future work:
  - Handle relations defined as conflicts or in intension by domain filtering
  - Automatically identify the appropriate consistency level
  - Use R(*,m)C in a solver to identify tractable classes of CSPs

Thank You for Your Attention

Questions?