10-2011

Relational Neighborhood Inverse Consistency for Constraint Satisfaction

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

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

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

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

Relational Neighborhood Inverse Consistency for Constraint Satisfaction

Robert J. Woodward[1] Shant Karakashian[1]
Berthe Y. Choueiry[1] Christian Bessiere[2]

[1]Constraint Systems Laboratory
University of Nebraska-Lincoln, USA
{rwoodwar|shantk|choueiry}@cse.unl.edu

[2]LIRMM-CNRS
University of Montpellier, France
bessiere@lirmm.fr

UNL-CSE-2011-0007

October 5, 2011

Abstract

Freuder and Elfe [1996] introduced Neighborhood Inverse Consistency (NIC) as a new local consistency property for Constraint Satisfaction Problems (CSPs) that filters the domains of variables. Two advantages of the algorithm for enforcing NIC is that it automatically adapts its filtering power to the local connectivity of the network and has insignificant space overhead. In this document, we discuss Relational Neighborhood Inverse Consistency (RNIC), which is an extension of NIC to filter relations introduced in [Woodward *et al.*, 2011a], how we enhance the propagation effectiveness by reformulating the dual graph of the CSP. We also describe an automated selection policy that outperforms all approaches in a statistically significant manner.

Contents

1 Introduction

An important result in Constraint Processing (CP) ties the tractability[1] of a Constraint Satisfaction Problem to the level of consistency that it satisfies. Solving difficult problems often requires enforcing higher order consistency, which typically requires the use of more costly algorithms in time and/or in space. Freuder and Elfe [1996] introduced Neighborhood Inverse Consistency (NIC) for CSPs as a particularly promising consistency property because: (1) Enforcing it is light in terms of space requirements (inverse consistency is enforced by filtering the variables domains); and (2) It focuses the attention on where a variable's value most tightly interacts with the problem, namely its neighborhood. Despite its promise and filtering effectiveness, NIC remains relatively unexploited because the algorithm for enforcing it is too costly in terms of processing time, which prevented its use on dense networks or in a lookahead scheme during backtrack search.

In [Woodward *et al.*, 2011a], we generalized NIC to Relational Neighborhood Consistency (RNIC) for filtering relations. Although, Bacchus *et al.* [2002] had already identified the same property as RNIC to hold when the arc-consistency property holds in on the dual graph[2] of the CSP, they do not provide a practical algorithm for enforcing it, study its usefulness in practice, or compare to any consistency properties other than arc consistency, all of which we examine in this document.

This document is structured as follows. Section 2 reviews background information about CSPs. Section 3 introduces RNIC and section 4 describes an algorithm for enforcing it on the dual encoding of the CSP. Section 5 discusses three variations of RNIC obtained by removing redundant edges in the dual graph and/or triangulating the considered graph, and a strategy for deciding which of the four properties to enforce. The goal of this deliberation is to reduce computational cost and/or strengthen propagation depending on the topology of the dual graph. Section 6 reviews the state of the art in relational consistency. Section 7 discusses our experimental results, we compare the performance of the resulting mechanisms, on difficult benchmark problems, with that of GAC2001 [Bessière *et al.*, 2005] and the recently introduced algorithms for m-wise consistency (i.e., wR($*$,m)C for $m = 2$, 3, 4 of [Karakashian *et al.*, 2010]). Finally, Section 8 discusses the extension of

[1]The tractability of a problem is the ability to solve it in time polynomial in the size of the input, which, in the case of the CSP, is the number of variables.

[2]The dual graph is defined in Section 2.

our approach to relations specified as conflicts or in intension and concludes this document with directions for future research.

2 Background

A Constraint Satisfaction Problem (CSP) is defined by $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ where $\mathcal{V}$ is a set of variables, $\mathcal{D}$ is a set of domains, and $\mathcal{C}$ is a 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 $\mathcal{C}$. Each constraint $C_i \in \mathcal{C}$ is specified by a relation R_i defined on a subset of the variables, called the scope of the relation and denoted $scope(R_i)$. Given a relation R_i, a tuple $\tau_i \in R_i$ is a vector of allowed values for the variables in the scope of R_i. Solving a CSP corresponds to finding an assignment of a value to each variable such that all the constraints are satisfied.

2.1 Graphical representations

A binary CSP is represented by its *constraint graph* where the vertices are the variables of the CSP and the edges represent the constraints. A non-binary CSP is similarly represented by its *hypergraph* where the hyperedges represent the non-binary constraints. Another graphical representation of a non-binary CSP is the *primal graph* where the vertices are the CSP variables and edges connect every two vertices corresponding to variables in the scope of a relation [Dechter, 2003]. Neigh(V_i) denotes the set of variables that are adjacent to V_i in the constraint graph of a binary CSP and the primal graph of a non-binary CSP.

The dual encoding of a CSP $\mathcal{P}$, denoted $\mathcal{P}^{\mathcal{D}}$, is a binary CSP whose variables are the relations of $\mathcal{P}$, their domains are the tuples of those relations, and the constraints enforce *equalities* over the shared variables. The representation as a graph of this encoding is the *dual graph* of the CSP. Neigh(R_i) denotes the set of relations adjacent to a relation R_i in the dual graph. Figure 1, illustrates the hyper, primal, and dual graphs of a small non-binary CSP where $\mathcal{V} = \{A, \ldots, F\}$ and the relations are $R_1, \ldots, R_6$.

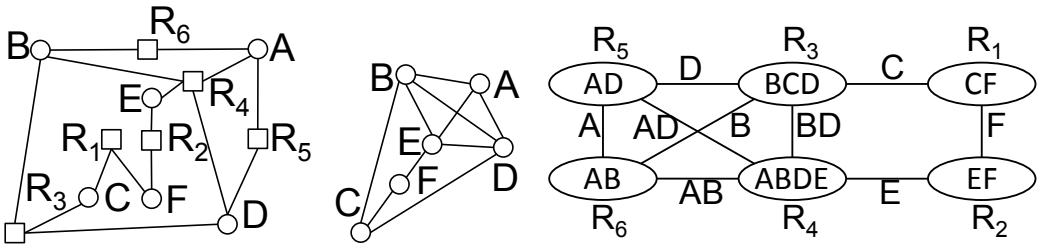

Figure 1: Hyper, primal, and dual graphs of a small CSP.

2.2 Consistency properties & algorithms

CSPs are in general $\mathcal{NP}$-complete and solved by search. To reduce the severity of the combinatorial explosion, they are usually 'filtered' by enforcing a given local consistency property [Bessiere, 2006].

One common such property is Generalized Arc Consistency (GAC). A CSP is GAC iff, for every relation, any value in the domain of any variable in the scope of the relation can be extended to a tuple satisfying the relation. Our work extends the local consistency property known as Neighborhood Inverse Consistency (NIC) introduced in [Freuder and Elfe, 1996] to relation filtering. NIC ensures that every value in the domain of a variable can be extended to a solution of the subproblem induced by the variable and the variables in its neighborhood. Algorithms for enforcing GAC and NIC typically operate by filtering the domains of the variables. Pairwise consistency[3] [Janssen *et al.*, 1989], R($*$,m)C [Karakashian *et al.*, 2010] and RNIC are consistency properties of the dual graph of the CSP. The algorithms for enforcing them typically operate by filtering the constraint definitions.

In order to compare the various consistency properties discussed in this document we use the terminology introduced in [Debruyne and Bessière, 1997]. Given two consistency properties p and p',

- p is *stronger* than p' if, in any CSP where p holds, p' also holds.
- p is *strictly stronger* than p' if p is stronger than p' and there exists at least one CSP in which p' holds but p does not.
- p and p' are *equivalent* when p is stronger than p' and vice versa.

[3]Pairwise consistency [Janssen *et al.*, 1989] is equivalent to R($*$,2)C [Karakashian *et al.*, 2010].

- Finally, p and p' are *incomparable* when there exists at least one CSP in which p holds but p' does not, and vice versa.

In practice, when a consistency property is stronger (respectively, weaker) than another, enforcing the former never yields less (respectively, more) pruning than enforcing the latter on the same problem.

3 Relational NIC

The algorithm for enforcing NIC on CSPs of [Freuder and Elfe, 1996] was tested on binary CSPs in a preprocessing step to backtrack search on instances whose constraint density[4] did not exceed 4.25%. Despite its pruning power and light space overhead, NIC received relatively little attention in the literature, likely because of the prohibitive cost of the algorithm for enforcing it. Below, we introduce RNIC as a generalization of NIC and characterize this new property in terms of other known consistency properties. Indeed, the former is a property that applies to the tuples of the relations of the CSP, while the latter applies to the values in the variables' domains (which are, in fact, unary relations).

3.1 Defining RNIC

Definition 1 *A relation R_i is said to be* RNIC *iff every tuple in R_i can be extended to the variables in $\bigcup_{R_j \in \text{Neigh}(R_i)} scope(R_j) \setminus scope(R_i)$ in an assignment that simultaneously satisfies all the relations in* Neigh(R_i). *A network is* RNIC *iff every relation is* RNIC.

Informally, every tuple τ_i in every relation R_i can be extended to a tuple τ_j in each $R_j \in \text{Neigh}(R)$ such that together all those tuples are consistent with all the relations in Neigh(R_i). Like R($*$,m)C, RNIC can be enforced by filtering the existing relations and without introducing any new relations to the CSP. A straightforward algorithm for enforcing RNIC applies the following operation to every relation R_i in the problem until quiescence:

$$R_i \leftarrow \pi_{scope(R_i)}(\bowtie_{R_j \in \{R_i\} \cup \text{Neigh}(R_i)} R_j) \tag{1}$$

[4]The constraint density of a binary CSP is equal to $\frac{2e}{n(n-1)}$, where e is the number of constraints and n the number of variables.

where π and $\bowtie$ are the relational operators project and join, respectively. The space requirement of this algorithm is prohibitive in practice because it requires storing the join of $R_i \cup \text{Neigh}(R_i)$, which is not necessary as we argue in Section 4. For the example of Figure 2, RNIC examines the six subproblems induced on the dual graph by each relation and its neighborhood as listed below:

1. For R_1, $\text{Neigh}(R_1) = \{R_2, R_3\}$.
2. For R_2, $\text{Neigh}(R_2) = \{R_1, R_4\}$.
3. For R_3, $\text{Neigh}(R_3) = \{R_1, R_4, R_5, R_6\}$.
4. For R_4, $\text{Neigh}(R_4) = \{R_2, R_3, R_5, R_6\}$.
5. For R_5, $\text{Neigh}(R_5) = \{R_3, R_4, R_6\}$.
6. For R_6, $\text{Neigh}(R_6) = \{R_3, R_4, R_5\}$.

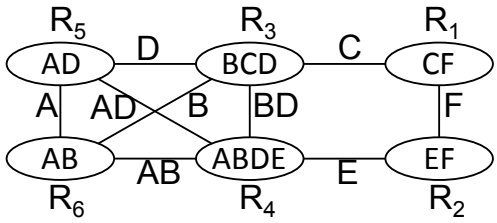

Figure 2: The dual graph of a small CSP.

Generally speaking, the number of induced subproblems to be considered is equal to e, where e is the number of relations in the CSP; and the size of the largest subproblem is equal to $\delta + 1$, where δ is the degree of the dual graph.

3.2 Comparing RNIC and R($*$,m)C

In [Karakashian *et al.*, 2010], we introduced the property R($*$,m)C with $m \geq 2$, which ensures that every tuple in every relation can be extended in a consistent assignment to every combination of $m-1$ relations in the problem. For the example shown in Figure 1, R($*$,2)C must verified on 9 combinations of two relations. Generally speaking the number of induced subproblems to be considered is $\mathcal{O}(e^m)$; and the size of the largest subproblem is equal to m. We compare RNIC with R($*$,m)C, which is defined for $m \geq 2$.

Theorem 1 *RNIC is strictly stronger than R(∗,m)C, $m \leq 3$.*

Sketch of proof: For a relation R_i, RNIC requires that each tuple of R_i and at least one tuple from each of the relations in Neigh(R_i) be consistent, all together. R(∗,2)C requires that the tuple of R_i be consistent with some tuple in each of the relations in Neigh(R_i), taken in separation. Thus, RNIC is strictly stronger than R(∗,2)C. For R(∗,3)C, at least one relation in each combination of three relations is such that its neighborhood encompasses at least the other two relations. Thus, RNIC is strictly stronger than R(∗,3)C. □

Theorem 2 *R(∗,m)C with $m \geq \delta + 1$, where δ is the degree of the dual graph, is strictly stronger than RNIC.*

Sketch of proof: When $m > \delta$, every set of relations considered by RNIC is a subset of at least one set of relations on which R(∗,m)C is enforced. □

Theorem 3 *For $4 \leq m \leq \delta$, R(∗,m)C and RNIC are not comparable.*

Sketch of proof: If a dual graph has a chain of relations of length between four and $\delta - 1$, R(∗,m)C for $4 \leq m \leq \delta$ can be stronger than RNIC. Conversely, if the dual graph is a star graph of five or more vertices, $S_{i>4}$, RNIC can be stronger than R(∗,m)C for $4 \leq m \leq \delta$. □

Figure 3 illustrates the above first three assertions. Two interesting struc-

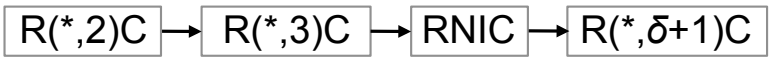

Figure 3: Comparing RNIC with R(∗,m)C.

tures of the dual graphs, trees and cycles, are such that several relational consistency properties collapse to R(∗,2)C, which is the weakest of them all:

Theorem 4 *RNIC, R(∗,2)C, and R(∗,m)C are equivalent on any dual graph that is tree structured or is a cycle of length $\geq maximum(4, m + 1)$.*

Proof: By straightforward generalization of Theorem 3. □

The theorem applies for a tree of any degree. As for the cycle, it must be length at least $m + 1$ for $m \geq 3$. Figure 4 shows two such configurations. This last theorem is important because it identifies structural configurations where the relational consistency properties RNIC and R(∗,m)C collapse to their weakest version, that is R(∗,2)C. In Section 5 we propose reformulating the dual graph of the CSP to allow RNIC to overcome this obstacle.

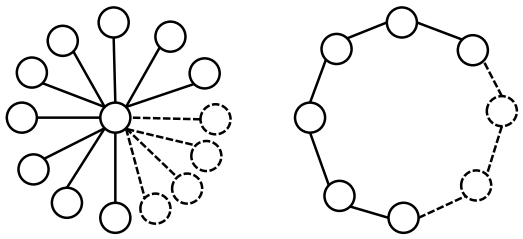

Figure 4: Configurations illustrating Theorem 4.

3.3 Comparing RNIC and domain filtering

In practice, after enforcing RNIC on a CSP (by filtering the relations), the domains of the variables are updated accordingly in order to reduce the search effort. It is important to note that variable domains can be updated by simply projecting the filtered relations on the variables. Interestingly, these domain reductions do not break the RNIC property.

Theorem 5 *If a network is RNIC, domain filtering by GAC cannot enable further constraint filtering by RNIC.*

Proof: Similar to proof of Theorem 1 in [Karakashian *et al.*, 2010]. □

Following the terminology of [Bessière *et al.*, 2008], the property of a CSP where RNIC holds and where the domains agree with the constraints is denoted RNIC+GAC. Although formally correct, we find this notation confusing because it may incorrectly suggest the need to enforce GAC, which is in general more expensive than (simply and without looping) projecting the relations on the domains. For that reason, we choose to denote this property instead RNIC+DF (i.e., RNIC followed by domain filtering).

Theorem 6 *NIC (on a binary CSP) and RNIC+DF (on the dual graph of the same binary CSP) are not comparable.*

Proof: In Figure 5, the CSP is NIC but not RNIC+DF. RNIC removes the tuples in $\{(0,2),(2,2)\}$ from R_0, $\{(0,0),(1,2)\}$ from R_1, $\{(0,2)\}$ from R_2, $\{(0,2)\}$ from R_3, and $\{(0,1),(2,1)\}$ from R_4. Therefore, RNIC+DF removes the value 0 from A. In Figure 6, the CSP is RNIC+DF but not NIC. NIC removes the value 0 from D. □

The *singleton* variant of a given consistency property guarantees that the assignment of every value in the domain of a variable yields a CSP where the

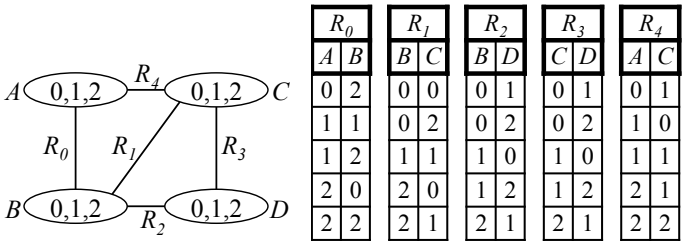

Figure 5: The binary CSP is NIC but not RNIC+DF.

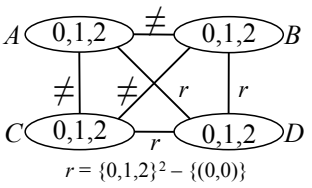

Figure 6: Binary CSP is RNIC+DF but not NIC.

consistency property holds [Debruyne and Bessière, 2001]. Singleton consistencies have been studied mainly for arc consistency (SAC) and generalized arc consistency (SGAC).

Theorem 7 *SGAC on a non-binary CSP and RNIC+DF on the corresponding dual graph are not comparable.*

Proof: In Figure 7, the CSP is RNIC+DF but not SGAC. SGAC empties all

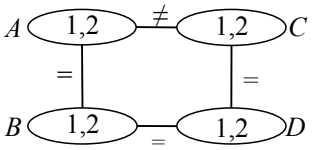

Figure 7: The CSP is RNIC+DF but not SGAC.

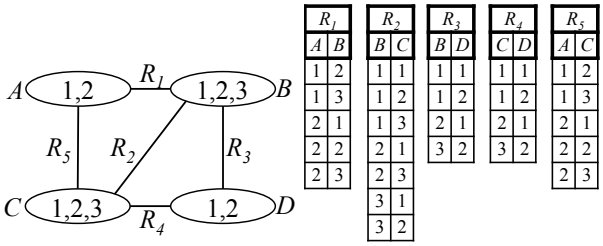

Figure 8: The CSP is SGAC but not RNIC+DF.

variables domains. In Figure 8, taken from [Debruyne and Bessière, 2001], the CSP is SGAC but not RNIC+DF. RNIC removes $\{(2,3),(3,2)\}$ from R_2, $\{(1,2),(1,3)\}$ from R_1, and $\{(1,2),(1,3)\}$ from R_5. Therefore, RNIC+DF removes the value 1 from A. □

Figure 9 shows the relationships between the domain-filtering properties discussed above.

Figure 9: Some domain filtering properties.

4 Enforcing RNIC

Below, we describe an algorithm for enforcing RNIC on a finite CSP, and analyze its complexity. The algorithm has two main components: ProcessQ (Algorithm 1) and SearchSupport.

4.1 An algorithm for RNIC

We define S_τ, the *support* of a tuple $\tau \in R$, to be the set of tuples that verify the condition: $\forall R' \in \text{Neigh}(R), \exists(\tau' \in R'), (\tau' \in S_\tau)$, and the tuples in $S_\tau \cup \{\tau\}$ agree on all shared variables. ProcessQ (Algorithm 1) enforces RNIC on a CSP $\mathcal{P}$ ensuring that every tuple in every relation has a valid support. Note that the Neigh(R) is determined by the topology of the dual graph, which we will alter in Section 5.

ProcessQ operates on a queue of relations $\mathcal{Q}_R$ initialized with all the relations of $\mathcal{P}$. For each relation R of $\mathcal{P}$, we maintain a queue of tuples $\mathcal{Q}_t(R)$ initialized with all the tuples in R. The function SearchSupport(τ, R) computes S_τ as discussed below. The function Rel(τ) returns the relation to which τ belongs. The data structure *SupportedBy*(τ) maintains the list of tuples supported by τ.

ProcessQ removes from $\mathcal{Q}_R$ one relation R at a time. It iterates over the tuples of R stored in $\mathcal{Q}_t(R)$. For each tuple $\tau \in \mathcal{Q}_t(R)$, SearchSupport seeks a support for τ. When a support is not found, τ is removed from R, and all tuples τ_i supported by τ are added to the queue of their respective relations, and the corresponding relations added to $\mathcal{Q}_t$. Finally, τ is removed from $\mathcal{Q}_t(R)$. Whenever a relation is empty, ProcessQ halts and returns false indicating that $\mathcal{P}$ is not consistent. When $\mathcal{Q}_R$ is empty ProcessQ terminates successfully indicating that $\mathcal{P}$ is RNIC.

Algorithm 1: ProcessQ enforces RNIC

Input: $\mathcal{Q}_R$ a queue of relations, $\{\mathcal{Q}_t(R)\}$ a set of queues of tuples, one for each relation

Output: *true* if the problem is RNIC, $false$ otherwise

```
1  while (Q_R ≠ ∅) do
2      R ← Pop(Q_R)
3      foreach τ ∈ Q_t(R) do
4          support ← SearchSupport(τ, R)
5          if support = false then
6              Delete(τ, R)
7              if R = ∅ then return false
8              forall τ_i ∈ SupportedBy(τ) do
9                  R_i ← Rel(τ_i)
10                 Q_t(R_i) ← Q_t(R_i) ∪ {τ_i}
11                 Q_R ← Q_R ∪ {R_i}
12         Q_t(R) ← Q_t(R) \ {τ}
13 return true
```

4.2 SearchSupport

SearchSupport(τ, R) operates by conducting a backtrack search on $\mathcal{P}_R^D$ the subproblem induced by $\{R\} \cup \text{Neigh}(R)$ on the dual encoding of $\mathcal{P}$. The variables of $\mathcal{P}_R^D$ are the relations $\{R\} \cup \text{Neigh}(R)$. Their domains are the tuples of the relations except for the variable corresponding to R, which is assigned the tuple τ. A solution to $\mathcal{P}_R^D$ is $\{\tau\} \cup S_\tau$. The search stops at the first solution, or returns false if no solution is found. The process uses forward checking and dynamic variable ordering (domain/degree). Two major mechanisms significantly contributed to the success of this search process by improving its running time:

1. *The use of the index-tree data structure* to determine whether or not two tuples of two relations adjacent in the dual graph are consistent. This data structure was proposed in [Karakashian *et al.*, 2010].

2. *The dynamic identification, after each variable instantiation, of trees in the graph of uninstantiated variables.* The instantiation of a variable eliminates, from the problem, the variable and the constraints that

link it to the uninstantiated variables, potentially breaking cycles in the graph and yielding trees. We call those trees *dangles*, and apply directional arc consistency on them to ensure that they are solvable. If they are, we isolate them from the search process. Otherwise, we force the search to backtrack. Dangle identification is linear in the number of vertices and edges. Its overhead, if any, was largely compensated by its benefits.

Note that dangle identification is a general mechanism for improving the performance of *any* backtrack search. Obviously, it cannot be used in the algorithm for enforcing GAC or R(∗,2)C (where there is no search). Further, it is not particularly useful in the algorithm for enforcing R$(*, m)$C because the values of m are small in practice.

4.3 Complexity analysis

The time complexity is dominated by ProcessQ. Let d be the maximum domain size, k the maximum constraint arity, e the number of relations, and δ the degree of the dual graph. The maximum number of tuples t in a relation is bounded by $\mathcal{O}(d^k)$. The outer loop (Line 1) iterates over the relations in $\mathcal{Q}_R$. This loop runs e times, the initial size of $\mathcal{Q}_R$, plus the number of times a relation is added to $\mathcal{Q}_R$ (Line 11). Given that a relation is adjacent to at most δ other relations, whenever a tuple is deleted, at most δ relations are added to $\mathcal{Q}_R$. There are $\mathcal{O}(te)$ tuples in $\mathcal{P}$ and each tuple is deleted at most once. Thus, Line 6 is executed $\mathcal{O}(te)$ times, each time enqueuing $\mathcal{O}(\delta)$ relations. Consequently, the outer loop (Line 1) runs $\mathcal{O}(te\delta)$ times.

The loop over the queued tuples (Line 3) executes $\mathcal{O}(t)$ times per relation. To find the support of a tuple, SearchSupport first verifies the validity of an existing support, then, if needed, it looks for a support by running a backtrack search on the subproblem induced by the relation and its neighbors. Verifying the validity of an existing support costs $\mathcal{O}(\delta)$. To build a support for a tuple, SearchSupport executes a backtrack search on a problem with $\delta + 1$ variables of maximum domain size t where the first variable is instantiated. The complexity of this search is $\mathcal{O}(t^\delta)$. Thus, ProcessQ is $\mathcal{O}(t^{\delta+1}e\delta)$. The space complexity of ProcessQ is dominated by that of the data structures. Supports require $\mathcal{O}(et\delta)$ space. The index-trees require $\mathcal{O}(ket\delta)$ [Karakashian *et al.*, 2010]. The time complexity of the obvious algorithm based on Expression (1) is $\mathcal{O}(t^{\delta+2}e\delta)$. When intermediate joins

are not stored, its space complexity is $\mathcal{O}(t^{\delta+1})$, a major bottleneck for its practical implementation. Thus, ProcessQ saves on both time and space.

4.4 Enforcing RNIC versus R(∗,m)C

The above summarized algorithm and that for enforcing R(∗,m)C [Karakashian *et al.*, 2010] are similar in that they both try to 'complete' [Freuder, 1991] each tuple in each relation over one (or more) sets of relations.

The algorithm for R(∗,m)C considers *every* combination of m connected relations. The number of those combinations is $O(e^m)$. Further, each relation needs to be 'checked' against $m-1$ relations in *each* combination where it appears.

The algorithm for enforcing RNIC does not suffer from the above drawbacks. First, the number of combinations considered is equal to the number of relations (e), and each relation is 'checked' against a unique set of relations, which is determined by its neighborhood. Further, the size of the neighborhood is determined *locally* by the connectivity of the relation in the dual graph. Thus, the 'level' of consistency enforced is not necessarily the same on all relations of the dual graph: Lower levels are enforced on sparser portions of the dual graph, and higher levels on the denser portions. In particular, on a cycle of length four or more, RNIC 'naturally' reduces to R(∗,2)C, see Theorem 4.

5 Reformulating the Dual Graph

Two topological conditions of the dual graph can seriously hinder the performance of ProcessQ (Algorithm 1):

1. *High density of the dual graph.* As the density of the dual graph increases, the neighborhood of a given relation R_i grows, which increases the cost of enforcing RNIC. To address this issue, we reformulate the dual graph by removing redundant edges.

2. *The existence of cycles of length four or more.* On a cycle of length four or more, the two adjacent relations of a given relation R_i in the cycle are prevented from 'communicating,' thus reducing RNIC to R(∗,2)C (see Theorem 4). To address this issue, we propose to reformulate the

dual graph by triangulation,[5] which eliminates cycles of length four or more.

The above two reformulations have the following effects:

- Removing redundant edges cannot strengthen the consistency property enforced by the algorithm and cannot decrease the number of nodes visited by search.
- Adding edges by graph triangulation cannot weaken the consistency property enforced and cannot increase number of nodes visited by search.

Applying ProcessQ on the dual graph reformulated by one or both of the above reformulations enforces three variations of RNIC, namely wRNIC, triRNIC, and wtriRNIC, where the prefixes 'w' and 'tri' denote the consistency properties resulting from removing redundant edges and triangulating the dual graph, respectively. Figure 10 illustrates those relationships in a partial order. Naturally, the property enforced depends on the particular

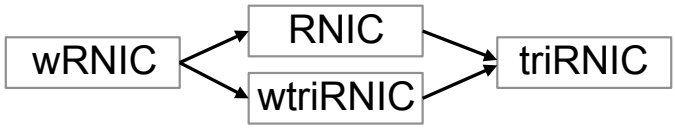

Figure 10: Variations of RNIC.

minimal and triangulated dual graph used.

While the *set of solutions to the CSP is not affected by either reformation*, it is not straightforward to predict the effect of the above reformulations on CPU time. To lay it out, we would like to remove enough edges from the dual graph to reduce the running time of ProcessQ, which is $\mathcal{O}(t^{\delta+1}e\delta)$. However, we would also like to add enough edges to the dual graph in order to boost propagation. Furthermore, we need a strategy to automatically select the appropriate reformulation. Below, we discuss the two reformulations (Sections 5.1 and 5.2) and their combination (Section 5.3). In Section 5.4, we propose a procedure to automatically select a reformulation in a preprocessing step.

[5]Graph triangulation adds an edge (a chord) between two non-adjacent vertices in every cycle of length four or more [Golumbic, 2004]. While minimizing the number of edges added by the triangulation process is NP-hard, MinFill is an efficient heuristic commonly used for this purpose [Kjærulff, 1990; Dechter, 2003].

5.1 Removing redundant edges: wRNIC

An edge between two vertices in the dual graph is *redundant* if there exists an alternate path between the two vertices such that the shared variables appear in every vertex in the path [Janssen *et al.*, 1989; Dechter, 2003]. Redundant edges can be removed without affecting the set of solutions of the CSP. Janssen et al. [1989] introduced an efficient algorithm for computing the *minimal dual graph* by removing redundant edges. Many minimal graphs may exist, but all are guaranteed to have the same number of edges. Figure 11 shows the dual graph (density 60%) and a minimal dual graph (density 40%) of the example of Figure 1. Note that R(∗,2)C ≡ wR(∗,2)C [Janssen *et al.*,

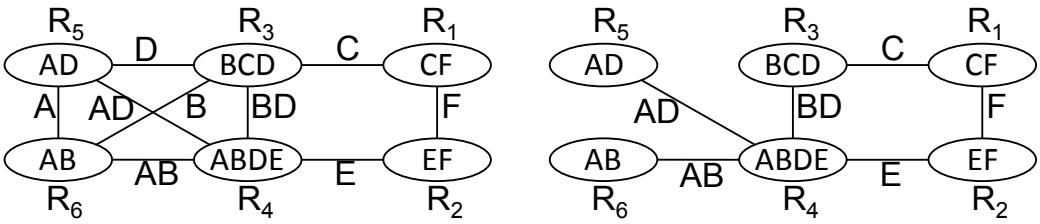

Figure 11: A minimal dual graph.

1989]. Also, computing and storing the combinations of relations necessary for enforcing R(∗,m)C is not possible in practice unless the redundant edges are first removed from the dual graph [Karakashian *et al.*, 2010].

Our experiments showed that RNIC is advantageous on dual graphs of density up to around 15%.[6] For higher density values, we propose to remove the redundant edges in the dual graph before running ProcessQ. This operation reduces the density of the original dual graph and the size of the induced subproblems on which SearchSupport is executed. It also results in a weakened consistency, denoted wRNIC, that depends of the particular minimal graph used. Because wRNIC is enforced on a minimal dual graph (i.e., a graph with no more edges than the original dual graph), wRNIC is strictly weaker than wRNIC.

[6]In a related research, we studied the density of 1689 dual graphs of (binary and non-binary) CSPs from the Solver Competition Benchmarks. We identified a sharp threshold at 15% density. Indeed, 56.6% of the dual graphs (79.9% after redundancy removal) considered had a density less than or equal to 15%. It is not yet clear to us how to interpret the value of this threshold.

Figure 12 integrates the above discussion in the partial order of Figure 3. Note that these results hold between the weakened properties provided they

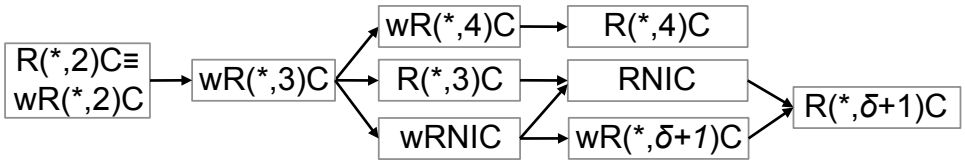

Figure 12: Relating RNIC, wRNIC, R($*$,m)C, and wR($*$,m)C.

are enforced on the *same* minimal dual graph.

5.2 Triangulating the dual graph: triRNIC

When the dual graph has only cycles of size four or more, RNIC reduces to R($*$,2)C (see Theorem 4), which significantly hampers filtering and propagation. To remedy this situation, we propose to triangulate the dual graph. This process creates loops in the dual graph and increases the size of the induced subproblems on which SearchSupport is executed, boosting the propagation process, but also raising the consistency level enforced on the CSP. For example, in the dual graph of the example of Figure 1, Neigh(R_1) $=\{R_2, R_3\}$. However, Neigh(R_1)$=\{R_2, R_3, R_4\}$ in the triangulated graph (density 67%) of Figure 13. We denote the resulting consistency property

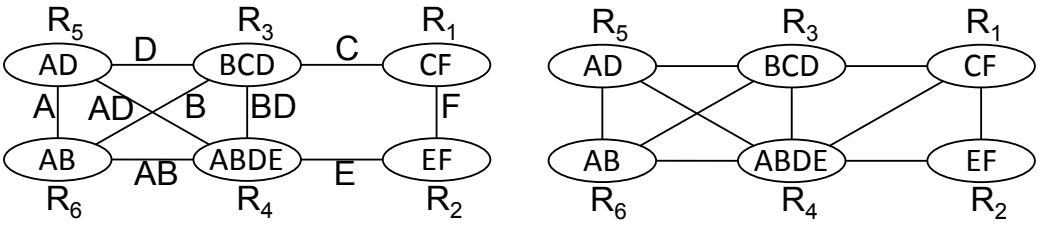

Figure 13: Triangulating a dual graph.

triRNIC. Similarly to wRNIC, triRNIC depends on the particular triangulation of the dual graph.

An important feature of the triangulation process is that it operates *locally*, adding edges only where cycles of length four or more need to be shortened, irrespective of the degree of the vertices in the graph.

5.3 Triangulate a minimal dual graph: wtriRNIC

While using a minimal dual graph allows us to cope with the high density of difficult benchmark instances, triangulating the minimal dual graph allows us to boost propagation. We denote wtriRNIC the consistency resulting from applying ProcessQ on the triangulated minimal dual graph. Figure 14 shows the dual graph (density 47%) resulting from applying both reformulations in sequence for the example of Figure 1. As shown in Figure 10,

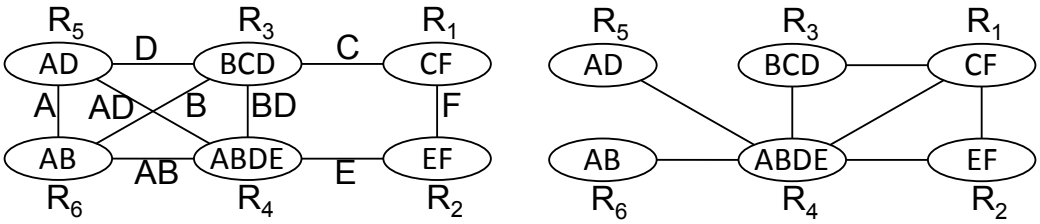

Figure 14: Triangulating a minimal dual graph.

wtriRNIC is strictly stronger than wRNIC applied on the same minimal dual graph, but strictly weaker than triRNIC. Further, it is not comparable with RNIC, which is enforced on the original dual graph. Figure 15 summarizes the relationships between RNIC, its reformulations, and R($*$,m)C based properties.

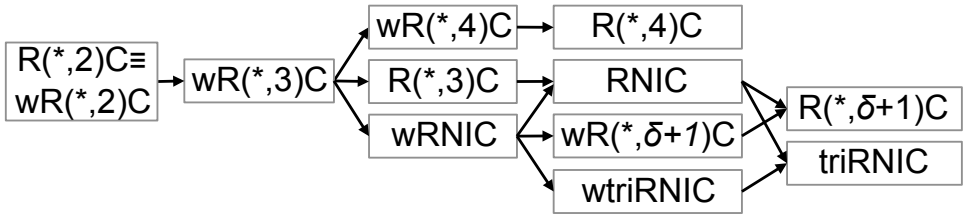

Figure 15: Relating RNIC, R($*$,m)C, and their studied variations.

5.4 Select the appropriate RNIC: selRNIC

The algorithm summarized in Section 4, ProcessQ, enforces any of the four properties RNIC, triRNIC, wRNIC, and wtriRNIC on a CSP by operating on the original dual graph or some modification of it.

- For RNIC, it uses the original dual graph (G_o).
- For wRNIC, it uses a minimal dual graph (G_w).
- For triRNIC, it uses a triangulated dual graph (G_{tri}).
- Finally, for wtriRNIC, it uses a triangulated minimal dual graph (G_{wtri}).

The selection policy shown in Figure 16 automatically chooses the dual graph on which to enforce RNIC by comparing the density d^G of a given dual graph G. The goal of this deliberation is to adjust the strength of propagation to the topology of the dual graph. Paraphrasing the content of Figure 16, we

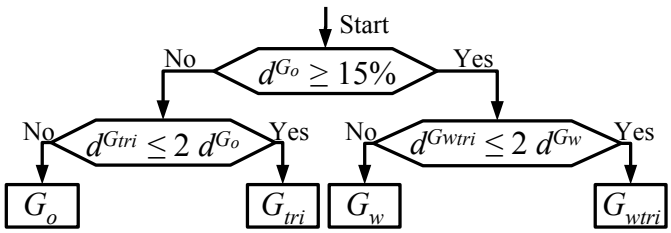

Figure 16: Selecting a dual graph for selRNIC.

consider the dual graph of density greater than or equal 15% to be too dense to be effectively processed by ProcessQ. For this reason, we choose to reformulate it by removing redundant edges. Whenever triangulation does not increase the density of a dual graph more than two fold, then the advantage of boosting propagation by creating loops and increasing neighborhood sizes outweighs the drawback of increasing the cost of operating on larger neighborhoods. For the example of Figure 1, this policy correctly chooses the triangulated minimal dual graph (density 47%). While both operations of triangulating a dual graph and computing a minimal dual graph can be done efficiently and do not add *any* perceptible overhead in our experiments, the policy of Figure 16 applies each operation at most once. The resulting mechanism, which we denote selRNIC, nicely ties together our techniques in a consistent and adaptive framework.

6 Related Work

NIC was proposed by Freuder and Elfe in [1996] and evaluated by them and others on binary CSPs. Debruyne and Bessière [2001] showed that NIC is

ineffective on sparse graph and too costly on dense graphs. Below, we restrict our discussion to non-binary CSPs.

In [Bacchus *et al.*, 2002], *nic(dual)* denotes applying NIC to the dual encoding of a CSP. As stated in the introduction, it is identical to RNIC. However, the paper does not go beyond stating that *nic(dual)* is strictly stronger than *ac(dual)* (i.e., RNIC is strictly stronger than R(∗,2)C). More generally, relational consistency properties were formalized in [Dechter and van Beek, 1997] as *relational m-consistency* and *relational (i, m)-consistency*. Enforcing those properties may require adding constraints to the problem, modifying its topology.

Beyond binary CSPs, most of the research on consistency for non-binary CSPs has focused on filtering the variables' domains, such as the study of 'variable-based' NIC [Gent *et al.*, 2000; Stergiou, 2007]. In contrast, our study focuses on the filtering of the relations (i.e., the constraints' definitions). As for relation-filtering properties, m-wise consistency was proposed in relational databases [Gyssens, 1986]. Janssen *et al.* [1989] showed that arc consistency on the dual encoding of a CSP enforces pairwise consistency. Algorithms for R(∗,m)C, which is equivalent to m-wise consistency, were proposed for arbitrary $m \geq 2$ and evaluated in [Karakashian *et al.*, 2010]. One limitation of the algorithm for R(∗,m)C is the need to manually select m and generate all combinations of m relations that form a connected graph. The number of combinations grows exponentially with m, causing space limitations. In comparison, RNIC requires storing for each relation R a unique combination of constraints $\{R\} \cup \text{Neigh}(R)$ and the size of this combination varies with the connectivity of R in the dual graph. Given the space requirement for storing all combinations of m relations, Karakashian *et al.* [2010] proposed to enforce R(∗,m)C on minimal dual graphs only, namely wR(∗,2)C, wR(∗,3)C, and wR(∗,4)C. The support structures used in ProcessQ are similar to those proposed in [Bessière *et al.*, 2005].

Finally, the insight that breaking cycles yields trees in a search space (i.e., tree, or dangle, identification in SearchSupport, Section 4) can be related to the Cycle-Cutset method [Dechter and Pearl, 1987].

7 Experimental Results

We evaluated and compared the performance of the following algorithms for enforcing consistency when used for full lookahead in a backtrack search

procedure for finding the first solution of a CSP:

- GAC
- wR($*$,m)C for $m = 2, 3, 4$
- RNIC and its variants: wRNIC, triRNIC, wtriRNIC and selRNIC.

We ran our experiments on 570 benchmark instances from the CSP Solver Competition[7] with a time limit of one and a half hours per instance.

Below we report and discuss our results on 169 instances representative. The results reported here are quantitatively but not qualitatively different from the results described in [Woodward *et al.*, 2011a; 2011b]. We also conduct statistical tests to determine the significance of our results. Since [Woodward *et al.*, 2011a], we have slightly improved the code of all our algorithms and enlarged the memory pool giving benefit to the wR($*$,m)C algorithms. The justification for those changes is to focus the evaluation on the performance of the algorithms rather than on their limitations.

In Section 7.1 (Table 1), we first present the global results of our experiments. Then, in Section 7.2 , we discuss in detail the results for four representative problems organized in three categories (Tables 2 3, and 4). We measured the following parameters:

- **Time**: The CPU time in milliseconds. A '-' entry in those columns indicates that, even though the corresponding algorithm terminated on some instances, it did not terminate on enough instances to yield an accurate statistical mean.
- **R**: The rank of the CPU time each algorithm based on the probability of the survival data analysis, breaking ties based on the time for reaching that probability.
- **S**: The equivalence classes of CPU performance. To compute the statistically significant categories, we perform a pairwise significance check between every two algorithms for a significance level of 0.05. This comparison requires a normal distribution of the non-censored data. For this analysis, we assume that all censored data points finished at the maximum cutoff time.

[7] `http://www.cril.univ-artois.fr/CPAI09/`

- $\mathbf{S}_B$:[8] Provides coarser equivalence classes based on termination only while ignoring CPU time.

- **#C**: The number of instances that were completed by a given algorithm.

- **#F**: The number of instances on which the given algorithm was the fastest among all tested ones, where ties are awarded to all parties.

- **#BF**: The number of instances that were solved by a given algorithm in a backtrack-free manner.

Section 7.1 is a global analysis ranking all tested algorithms. Section 7.2 discusses our results in detail.

7.1 Global rankings

Some data points are missing because some algorithms sometimes failed to finish within the allocated time window (90 minutes). For this reason, we consider the data to be right-censored and conduct a survival data analysis [Lee, 1992]. The survival data analysis does not make any assumption about the distribution of the data, and yields a calculated mean CPU time for each algorithm, reported in column **Time** in Table 1.

In this table, note how GAC is the best ranking algorithm in terms of the number of instances solved the fastest (column **#F**), although it is not the best ranking algorithm according to any of the remaining criteria. This result is not surprising when one notices that when GAC solves an instance, the remaining algorithms are not far behind in terms of CPU time. When looking at the statistical significance of CPU time in column **S**, GAC is indeed in the slowest equivalent class (i.e., C) while selRNIC is in the fastest equivalent class (i.e., A). Regarding the number of instances completed (**#C** and $\mathbf{S}_B$), selRNIC shows the best performance (i.e, 159 instances) and is significantly better than all other algorithms (the only algorithm in class A). Finally, selRNIC exhibits the best performance in terms of the number of instances completed in a backtrack-free manner (see column **#BF**, 142 out of 169 instances).

[8] Even though the normality assumption may or may not hold for **S** and $\mathbf{S}_B$, all our analyses yielded similar results, hinting that our conclusions are correct.

Table 1: Comparison over the 169 instances reported. In column **R**, ranks 1 and 9 indicate, respectively, the best and worst performances. In columns **S** and $\mathbf{S}_B$, ranks A and C indicate, respectively, best and worst performances.

Algorithm	**Time**	**#F**	**R**	**S**	**#C**	$\mathbf{S}_B$	**#BF**
wR(∗,2)C	944924	51	3	**A**	138	B	79
wR(∗,3)C	925004	7	4	B	134	B	92
wR(∗,4)C	1161261	2	5	B	132	B	108
GAC	1711511	**84**	7	C	119	C	33
RNIC	6161391	15	8	C	104	C	70
triRNIC	3017169	5	9	C	88	C	84
wRNIC	1184844	11	6	B	131	B	84
wtriRNIC	937904	9	2	B	145	B	128
selRNIC	**751586**	15	**1**	**A**	**159**	**A**	**142**
Benchmarks: aim-100, aim-200, lexVg, modifiedRenault, ssa							

When comparing selRNIC with a random selection of the four RNIC-based algorithms, within a 50ms error tolerance, selRNIC outperforms all four RNIC-based algorithms in a statistically significant manner. This result establishes that selRNIC is better than choosing any RNIC-based algorithm in a random manner (i.e., selRNIC is better than 'chance').

7.2 Detailed analysis

The instances tested pertain to five benchmark problems, one of them ssa is not reported in the results here because it has too few instances (only 8). We organize our results on the 161 instances the four remaining benchmark problems in three tables: Tables 2, 3, and 4. Each table gives the name of problem, its number of instances, the number of instances completed by all algorithms in parenthesis, and the range of the number of constraints e.

The tables also report the range of the density of the dual graph d^D on which a given algorithm operate and the average number of nodes visited by the corresponding search (**#NV**).[9] The averages are computed over only the

[9]Note that the values of nodes visited in all experiments comply with the partial order shown in Figure 15.

instances completed by all tested algorithms, which is the number in parenthesis in the problem description. Thus, the values reported in **#NV** should be considered in light of the number of completed instances. A '-' entry in this column indicates that, even if the corresponding search completed on some instance, no instance completed by this algorithm was completed by all others, and thus no average value can be reported. For each benchmark class, we report the number of instances in the class, with the number completed by all algorithms in parenthesis, and the range of the number of constraints e.

Table 2 illustrates the usefulness of RNIC: it solves the largest number of problems in this set, and solves, backtrack free, the largest number of instances. In terms of significance ranking, GAC, triRNIC, and wRNIC are not advantageous techniques for these problems that have low density, and high density after triangulation. selRNIC was able to select the dual graph that yielded the largest number of completions and backtrack free. Despite not always being the fastest, it was not significantly different than the algorithm that was the fastest.

Table 3 illustrates the usefulness of wRNIC and wtriRNIC. As stated above, the sheer number of relations combined with the large density in the dual graphs of the problems in this benchmark prevents us from executing RNIC and triRNIC. This situation demonstrates the benefits of using wRNIC and wtriRNIC, which were actually automatically chosen by selRNIC. Note also that wtriRNIC solves, backtrack free, all instances in this category. We cannot stress enough on the importance of this last fact: It is indicative of the tractability of this class of problems. Notice, despite selRNIC not having the smallest CPU time, there is not a statistically significant difference between the mean CPU time of selRNIC and and the mean CPU time of wR(∗,2)C. Once again, GAC was in a lower significance class than selRNIC, as with RNIC and triRNIC, as was expected. Note, that only two instances were completed by all algorithms, thus the nodes visited (**#NV**) reported in the table is not that meaningful.

In both Tables 2 and 3, selRNIC largely outperforms GAC for all measures. Even if one was to use a high-performance GAC implementation such as the one in [Cheng and Yap, 2010], the number of nodes visited by GAC remains orders of magnitude larger than that by selRNIC, and the number of instances solved backtrack-free significantly smaller. Only in Table 4 does GAC outperform the other algorithms in terms of CPU time only. Interestingly, however, on lexVg, and despite the high density ([57.6%,78.6%]) of the

Table 2: RNIC/selRNIC completes the largest number of instances, and solves, backtrack free, the largest number of instances.

Algorithm	d^D	**Time**	**R**	**S**	**#C**	**#F**	**#BF**	**#NV**
aim-100: 24(12) instances, $e \in$[150,570]								
wR(∗,2)C	[6.3%,8.1%]	412369	5	**A**	19	8	5	328
wR(∗,3)C		304816	3	**A**	20	1	7	157
wR(∗,4)C		140070	2	**A**	20	0	12	127
GAC	N/A	1923579	7	B	17	4	1	4158286
RNIC/ selRNIC	[6.3%,8.1%]	**94699**	**1**	**A**	**22**	5	**16**	101
triRNIC	[26.0%,70.5%]	2259986	8	B	14	0	14	**100**
wRNIC	[0.7%,2.6%]	1009380	4	B	20	6	7	188
wtriRNIC	[6.3%,8.1%]	1280885	6	**A**	20	6	7	151
aim-200: 24(0) instances, $e \in$[302,1169]								
wR(∗,2)C	[3.2%,4.2%]	**132205**	5	B	12	**10**	4	-
wR(∗,3)C		1006472	2	**A**	15	2	8	-
wR(∗,4)C		2015651	3	B	15	0	9	-
GAC	N/A	-	6	C	8	0	0	-
RNIC/ selRNIC	[3.2%,4.2%]	781596	**1**	**A**	**19**	6	**13**	-
triRNIC	[21.2%,71.6%]	-	8	C	1	0	1	-
wRNIC	[0.4%, 1.4%]	244643	4	B	13	4	5	-
wtriRNIC	[6.3%,11.6%]	-	7	C	7	0	7	-

redundancy removed triangulated dual graph, wtriRNIC/selRNIC solves in a backtrack-free manner all but one of the instances in this set, thus hinting to the tractability of these instances. (The last instance hit the time threshold.) Notice that even though GAC has a smaller CPU time than selRNIC, the difference between the two algorithms is not statistically significantly (see column **S**). There were not enough instances (8) for the ssa benchmark, reported in [Woodward *et al.*, 2011a], to report any statistically significant conclusions.

The 169 instances reported above are representative of the results obtained in our experiments, which were carried over 570 instances. Below, we classify the non-reported test instances into one of three qualitative categories identified by the above tables.

Table 3: RNIC is hindered by the high density of the dual graph, but its weakened versions outperform all others.

Algorithm	d^D	**Time**	**R**	**S**	**#C**	**#F**	**#BF**	**#NV**
modifiedRenault 50(2) instances, $e \in$[147,159]								
wR(∗,2)C	[35.4%,41.6%]	**3078**	6	**A**	46	**30**	41	**111**
wR(∗,3)C		8463	3	**A**	49	4	48	**111**
wR(∗,4)C		31157	**1**	**A**	**50**	2	**50**	**111**
GAC	N/A	1678928	7	B	25	14	4	124
RNIC	[35.4%,41.6%]	-	8	B	11	0	11	**111**
triRNIC	[36.4%, 43.8%]	-	9	B	8	0	8	**111**
wRNIC	[1.7%,1.9%]	8285	5	**A**	47	0	43	**111**
wtriRNIC	[2.9%,3.9%]	166652	2	**A**	**50**	0	**50**	**111**
selRNIC	[1.8%,3.6%]	166560	4	**A**	49	0	48	**111**

Table 4: GAC is best on CPU, triRNIC/selRNIC is best on #BF.

Algorithm	d^D	**Time**	**R**	**S**	**#C**	**#F**	**#BF**	**#NV**
lexVg: 63(40) instances, $e \in$[8,36]								
wR(∗,2)C	[48.5%,57.1%]	809765	4	C	55	3	27	30
wR(∗,3)C		1384983	7	C	44	0	27	30
wR(∗,4)C		1525548	7	C	41	0	35	**3**
GAC	N/A	**114827**	**1**	**A**	**63**	**60**	26	25
RNIC	[48.5%,57.1%]	1647671	6	C	45	3	27	30
triRNIC	[57.6%,78.6%]	1031882	3	B	59	5	**59**	**3**
wRNIC	[48.5%,57.1%]	1464461	5	C	45	1	27	30
wtriRNIC/ selRNIC	[57.6%,78.6%]	580935	2	**A**	62	3	**62**	**3**

- Table 2: aim-50 (24 instances), rand-10-20-10 (20 instances), dubois (13 instances), pret (8 instances).

- Table 3: renault (2 instances), travellingSalesman-20 (15 instances), travellingSalesman-25 (15 instances), varDimacs (9 instances), rand-3-20-20 (50 instances), rand-3-20-20-fcd (50 instances).

- Table 4: ssa (8 instances), ogdVg (65 instances), ukVg (65 instances),

and wordsVg (65 instances).

8 Future Work & Conclusions

Our approach opens the door to the investigation of a new type of singleton consistency properties for CSPs. Instead of assigning the value of a *single* variable before enforcing some level of consistency on the CSP, as it is usually the case for Singleton Arc Consistency (SAC) [Bessiere *et al.*, 2011], we should investigate the effectiveness of 'assigning a tuple to a relation' in the dual problem. Such an approach would yield a new class of relational consistency properties, which could be called *relation-based singleton consistency properties*. Note however, that, unlike RNIC, maintaining such properties during search is prohibitive in practice [Lecoutre and Prosser, 2006].

Our algorithm operates on relations defined in extension as consistent tuples (supports). Relations defined in extension as conflicts (no-goods) could be converted to supports, as we did here. Further, and also for constraints defined in intension, we could generate support tuples after applying GAC to the original CSP. For cases where it is important to keep all relation definitions in intension, we claim that a similar, albeit weaker, domain pruning can be achieved by executing RNIC on combinations of domain values that are consistent with the relations. We propose to mitigate the loss of information by generating new (support) constraints of some judiciously chosen scopes. We propose to investigate this approach in the future and evaluate its effectiveness.

Consistency properties and their algorithms are central to CP, and perhaps best distinguish this discipline from other fields that study the same problems. Research has focused on:

- defining new properties,
- proposing new algorithms,
- improving the performance of known ones, and
- theoretically characterizing the relationship between the consistency level and the tractability of the CSP.

Our contribution exploits and adds to the large body of literature on consistency properties and their propagation algorithms. However, our long-term

goal is to design techniques that allow a constraint solver to identify tractable problem classes and automatically select and apply the appropriate tools for solving them. In that sense, the ability of our techniques to adapt to a problem's structure and solve many difficult instances in a backtrack-free manner[10] is perhaps the most noteworthy contribution of the current research: It indicates that we may be one step closer to achieving our goal.

Acknowledgments

We are grateful to Elizabeth Claassen and David B. Marx of the Department of Statistics at the University of Nebraska-Lincoln (UNL) for their help with designing the statistical analysis. Experiments were conducted on the equipment of the Holland Computing Center at UNL. Robert Woodward was partially supported by a B.M. Goldwater Scholarship and by a National Science Foundation (NSF) Graduate Research Fellowship grant number 1041000. This research is supported by NSF Grant No. RI-111795.

References

[Bacchus *et al.*, 2002] Fahiem Bacchus, Xinguang Chen, Peter Van Beek, and Toby Walsh. Binary vs. Non-Binary Constraints. *Artificial Intelligence*, 140:1–37, 2002.

[Bessière *et al.*, 2005] Christian Bessière, Jean-Charles Régin, Roland H.C. Yap, and Yuanlin Zhang. An Optimal Coarse-Grained Arc Consistency Algorithm. *Artificial Intelligence*, 165(2):165–185, 2005.

[Bessière *et al.*, 2008] Christian Bessière, Kostas Stergiou, and Toby Walsh. Domain Filtering Consistencies for Non-Binary Constraints. *Artificial Intelligence*, 172:800–822, 2008.

[Bessiere *et al.*, 2011] Christian Bessiere, Stéphane Cardon, Romuald Debruyne, and Christophe Lecoutre. Efficient Algorithms for Singleton Arc Consistency. *Constraints*, 16 (1):25–53, 2011.

[10]Note that the complexity of RNIC is exponential in the degree of the dual graph and not in the number of variables.

[Bessiere, 2006] Christian Bessiere. *Handbook of Constraint Programming*, chapter Constraint Propagation. Elsevier, 2006.

[Cheng and Yap, 2010] Kenil C.K. Cheng and Roland H.C. Yap. An MDD-Based Generalized Arc Consistency Algorithm for Positive and Negative Table Constraints and Some Global Constraints. *Constraints*, 15 (2):265–304, 2010.

[Debruyne and Bessière, 1997] Romuald Debruyne and Christian Bessière. Some Practicable Filtering Techniques for the Constraint Satisfaction Problem. In *Proceedings of the 15 th International Joint Conference on Artificial Intelligence*, pages 412–417, 1997.

[Debruyne and Bessière, 2001] Romuald Debruyne and Christian Bessière. Domain Filtering Consistencies. *Journal of Artificial Intelligence Research*, 14:205–230, 2001.

[Dechter and Pearl, 1987] Rina Dechter and Judea Pearl. The Cycle-Cutset Method for improving Search Performance in AI Applications. In *Third IEEE Conference on AI Applications*, pages 224–230, Orlando, FL, 1987.

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

[Dechter, 2003] Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[Freuder and Elfe, 1996] Eugene C. Freuder and Charles D. Elfe. Neighborhood Inverse Consistency Preprocessing. In *Proceedings of AAAI-96*, pages 202–208, Portland, Oregon, 1996.

[Freuder, 1991] Eugene C. Freuder. Completable Representations of Constraint Satisfaction Problems. In *Second International Conference on Principles of Knowledge Representation and Reasoning (KR 91)*, pages 186–195, 1991.

[Gent *et al.*, 2000] Ian Gent, Kostas Stergiou, and Toby Walsh. Decomposable Constraints. *Artificial Intelligence*, 123 (1-2):133–156, 2000.

[Golumbic, 2004] Martin C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Elsevier, 2004. Annals of Discrete Mathematics, Vol 75.

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

[Janssen *et al.*, 1989] P. Janssen, Philippe Jégou, B. Nougier, and M.C. Vilarem. A Filtering Process for General Constraint-Satisfaction Problems: Achieving Pairwise-Consistency Using an Associated Binary Representation. In *IEEE Workshop on Tools for AI*, pages 420–427, 1989.

[Karakashian *et al.*, 2010] Shant Karakashian, Robert Woodward, Christopher Reeson, Berthe Y. Choueiry, and Christian Bessiere. A First Practical Algorithm for High Levels of Relational Consistency. In *24th AAAI Conference on Artificial Intelligence (AAAI 10)*, pages 101–107, 2010.

[Kjærulff, 1990] U. Kjærulff. Triagulation of Graphs - Algorithms Giving Small Total State Space. Research Report R-90-09, Aalborg University, Denmark, 1990.

[Lecoutre and Prosser, 2006] Christophe Lecoutre and Patrick Prosser. Maintaining Singleton Arc Consistency. In *CPAI 06 Workshop on Symmetry in Constraint Satisfaction Problems (SymCon 10)*, pages 47–61, 2006.

[Lee, 1992] Elisa T. Lee. *Statistical Methods for Survival Data Analysis*. John Wiley & Sons, New York, NY, second edition, 1992.

[Stergiou, 2007] Kostas Stergiou. Strong Inverse Consistencies for Non-Binary CSPs. In *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence*, volume 1 of *ICTAI 07*, pages 215–222, 2007.

[Woodward *et al.*, 2011a] Robert Woodward, Shant Karakashian, Berthe Y. Choueiry, and Christian Bessiere. Solving Difficult CSPs with Relational Neighborhood Inverse Consistency. In *25th AAAI Conference on Artificial Intelligence (AAAI 11)*, pages 1–8, 2011.

[Woodward *et al.*, 2011b] Robert J. Woodward, Shant Karakashian, Berthe Y. Choueiry, and Christian Bessiere. Reformulating the Dual Graphs of CSPs to Improve the Performance of Relational Neighborhood Inverse Consistency. In *Ninth International Symposium on Abstraction, Reformulation and Approximation (SARA 2011)*, pages 1–8. AAAI Press, 2011.