Solving Difficult CSPs with Relational Neighborhood Inverse Consistency

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

Abstract

Freuder and Elfe (1996) introduced Neighborhood Inverse Consistency (NIC) as a strong local consistency property for binary CSPs. While enforcing NIC can significantly filter the variables domains, the proposed algorithm is too costly to be used on dense graphs or for lookahead during search. In this paper, we introduce and characterize Relational Neighborhood Inverse Consistency (RNIC) as a local consistency property that operates on the dual graph of a non-binary CSP. We describe and characterize a practical algorithm for enforcing it. We argue that defining RNIC on the dual graph unveils unsuspected opportunities to reduce the computational cost of our algorithm and increase its filtering effectiveness. We show how to achieve those effects by modifying the topology of the dual graph, yielding new variations the RNIC property. We also introduce an adaptive strategy to automatically select the appropriate property to enforce given the connectivity of the dual graph. We integrate the resulting techniques as full lookahead strategies in a backtrack search procedure for solving CSPs, and demonstrate the effectiveness of our approach for solving known difficult benchmark problems.

1 Introduction

Solving difficult Constraint Satisfaction Problems (CSPs) remains a challenge today despite the dramatic advances of hardware technology. To counter the exponential growth of the size of the search space of CSPs, consistency properties and algorithms for enforcing them have been proposed since the inception of Constraint Programming (CP). Lower levels of consistency, such as Arc Consistency (AC) for binary constraints and Generalized Arc Consistency (GAC) for non-binary constraints, are commonly and advantageously used. However, solving difficult problems often requires enforcing higher orders 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 binary 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 this paper, we revisit NIC and generalize it to Relational Neighborhood Consistency (RNIC) for non-binary CSPs. We characterize RNIC and describe an effective algorithm for enforcing it that operates on the dual encoding of the CSP. We also introduce weakened and strengthened variations of this property to cope with the difficulties raised by the topology of the dual graph, and propose a strategy for automatically choosing the property to enforce. We integrate our algorithm as a full lookahead strategy in a backtrack search procedure for solving CSPs. We compare its performance, 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)). We conclude that the proposed techniques for enforcing RNIC

1. Achieve strong local consistency on non-binary CSPs and can be effectively used in a full lookahead scheme during search.
2. Enjoy the low space requirements of domain-filtering algorithms, such as GAC, and the pruning effectiveness of relation-filtering algorithms, such as R($*$,m)C.
3. Unlike R($*$,m)C, they are not hampered by the difficulty of choosing the appropriate consistency level to enforce; and, finally.
4. Locally adjust the consistency level enforced to the connectivity of each vertex in the dual graph.

This paper is structured as follows. Section 2 reviews background information about CSPs. Section 3 introduces RNIC and an algorithm for enforcing it on the dual encoding of the CSP. Section 4 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 5 reviews the state of the art in relational consistency. Section 6 discusses our experimental results. Finally, Section 7 discusses the extension of our approach to relations specified as conflicts or in intension and concludes this paper 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.

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). $\mathrm{Neigh}(V_i)$ denotes the set of variables that are adjacent to V_i the constraint graph of a binary CSP and the primal graph of a non-binary CSP. The dual encoding of a CSP $\mathcal{P}$ 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. $\mathrm{Neigh}(R_i)$ denotes the set of relations adjacent to a relation R_i in the dual graph. Janssen et al. (1989) and Dechter (2003) observed that, in the dual graph, an edge between two vertices is *redundant* if there exists an alternate path between the two vertices such that the shared variables appear in every vertex in the path. Redundant edges can be removed without affecting the set of solutions. 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. Figures 1, 2, 3, and 4 illustrate the hypergraph, primal graph, dual graph, and a minimal dual graph of a small non-binary CSP.

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, to non-binary CSPs, the local consistency property known as Neighborhood Inverse Consistency (NIC) introduced, for binary CSPs, in (Freuder and Elfe 1996). 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. In (2010), Karakashian et al. 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. Using the terminology introduced in (Debruyne and Bessière 1997), we say that a consistency property p is *stronger* than another one p' if, in any CSP where p holds, p' also holds. Further, we say that 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. We say that p and p' are *equivalent* when p is stronger than p' and vice versa. Finally, we say that that 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.

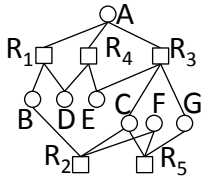

Figure 1: Hypergraph.

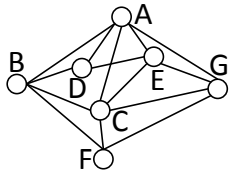

Figure 2: Primal graph.

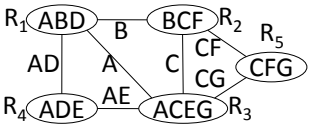

Figure 3: Dual graph.

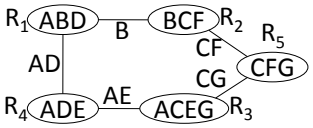

Figure 4: No redundant edges.

3 Relational NIC

The algorithm for enforcing NIC on binary CSPs of (Freuder and Elfe 1996) was tested in a preprocessing step to backtrack search on instances whose constraint density[1] 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, a generalization of NIC to non-binary CSPs, and characterize it. Then, we describe and analyze an algorithm for enforcing it.

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 $\bigcup_{R_j \in \mathrm{Neigh}(R_i)} scope(R_j) \setminus scope(R_i)$ in an assignment that simultaneously satisfies all the relations in* $\mathrm{Neigh}(R_i)$. *A network is* RNIC *iff every relation is* RNIC.

[1]The constraint density of a binary CSP is the density of its constraint graph.

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 $\text{Neigh}(R_i)$. 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:

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

where π and $\bowtie$ are the relational operators project and join. The space requirement of this algorithm is prohibitive in practice.

3.2 Characterizing RNIC

After enforcing RNIC, we filter all variable domains by projecting the filtered relations on the variables. Interestingly, these domain reductions do not break the RNIC property.

Theorem 1 *If a network is RNIC, domain filtering by GAC cannot enable further constraint filtering by RNIC.*

Proof: Similar to the corresponding proof for R(∗,m)C in (Karakashian et al. 2010).

We compare RNIC with R(∗,m)C, defined for $m \geq 2$.

Theorem 2 *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 $\text{Neigh}(R_i)$ 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 $\text{Neigh}(R_i)$, 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 3 *R(∗,m)C with $m > \delta$, 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 4 *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 wheel graph of $m + 1$ or more vertices, $W_{i>m+1}$, RNIC can be stronger than R(∗,m)C for $4 \leq m \leq \delta$. □

Theorem 5 *RNIC and R(∗,2)C are equivalent on a dual graph that is a star graph $S_{i>1}$ or a cycle of length four or more.*

Sketch of proof: Both properties consider the same combinations of vertices in the dual graph, namely every two adjacent vertices. □

Any minimal dual graph has necessarily fewer edges than the corresponding original dual graph. wR(∗,m)C, respectively wRNIC, is a weakened version of R(∗,m)C, respectively RNIC, resulting from using a minimal dual graph instead of the original one. The above results hold between the weakened properties provided they are enforced on the same minimal dual graph. Figure 5 illustrates the above discussion in a Hasse diagram.

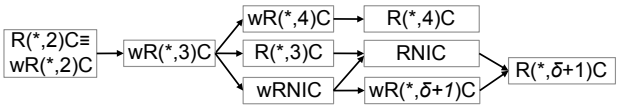

Figure 5: Relating RNIC, wRNIC, R(∗,m)C, and wR(∗,m)C.

3.3 An algorithm for enforcing RNIC

We define S_τ, the *support* of a tuple $\tau \in R$, to be the set of tuples that verify the condition: $\forall R' \in \text{Neigh}(R), \exists \tau' \in S_\tau, \tau' \in R'$, 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 $\text{Neigh}(R)$ is determined by the topology of the dual graph, which we will alter in Section 4.

PROCESSQ operates on a queue of relations $\mathcal{Q}$ 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 $\mathcal{Q}$ 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 $\mathcal{Q}$. 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 $\mathcal{Q}$ is empty PROCESSQ terminates successfully indicating that $\mathcal{P}$ is RNIC.

Algorithm 1: PROCESSQ enforces RNIC

Input: $\mathcal{Q}$ 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 ≠ ∅) do
2      R ← POP(Q)
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 ← Q ∪ {R_i}
12         Q_t(R) ← Q_t(R) \ {τ}
13 return true
```

3.4 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*: They can be floating trees or may be attached to some subgraph. We 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 can be done in linear in the number of vertices and edges using the Graham reduction (Maier 1983). Its overhead, if any, was largely compensated by its benefits.

3.5 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 $\mathcal{Q}$. This loop runs e times, the initial size of $\mathcal{Q}$, plus the number of times a relation is added to $\mathcal{Q}$ (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 $\mathcal{Q}$. 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.

3.6 Enforcing RNIC versus R($*$,m)C

The algorithms for enforcing RNIC (above) and 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 relations that are connected in the dual graph. The number of combinations considered is exponential in m ($O(e^m)$). Often, computing and storing those combinations is not possible unless redundant edges are first removed from the dual graph. Finally, a given 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, 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. For example, on a cycle of length four or more, RNIC reduces to R($*$,2)C.

4 Variations on RNIC

The following two conditions of the topology of the dual graph can seriously hinder the performance of the algorithm for enforcing RNIC (Algorithm 1):

1. High density of the dual graph.
2. The existence of cycles of length four or more.

As the density of the dual graph increases, the neighborhood of a given relation R_i grows, which increases the cost of enforcing RNIC. Further, 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 5). To address the above issues, we propose two reformulations of the dual graph, namely:

1. Removing redundant edges as introduced in Section 2.
2. Triangulating the dual graph to eliminate cycles of length four or more.

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).

Removing redundant edges reduces the density of the dual graph and thus the cost of enforcing RNIC. Triangulating

the dual graph creates loops in the graph, and thus enhances propagation and filtering. Applying one or the other of the above two reformulations, or both, the dual graph before applying Algorithm 1 yields three variations of RNIC, namely wRNIC, triRNIC, and wtriRNIC. Figure 6 illustrates the relationships between these properties as discussed in Sections 4.1, 4.2, 4.3. In Section 4.4, we propose a selection

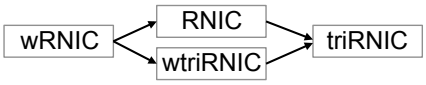

Figure 6: Variations of RNIC.

procedure to automatically decide, in a preprocessing step, which of the properties to enforce.

4.1 Use a minimal dual graph: wRNIC

Our experiments showed that RNIC is advantageous on dual graphs of density up to around 15%.[2] For higher density values, we propose to remove the redundant edges in the dual graph before running PROCESSQ (Algorithm 1). To this end, we use the efficient algorithm proposed in (Janssen et al. 1989), which guarantees a minimal dual graph. 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 obtained. Because wRNIC is enforced on a minimal dual graph (i.e., a graph with no more edges than the original dual graph), RNIC is strictly stronger than wRNIC.

4.2 Triangulate the dual graph: triRNIC

When the dual graph has only cycles of size four or more, RNIC reduces to R(∗,2)C, which significantly hampers the filtering as well as the propagation process. To remedy this situation, we propose to triangulate the dual graph, thus adding edges to it, increasing 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. We denote the resulting consistency property triRNIC. Similarly to wRNIC, triRNIC depends on the particular triangulation of the dual graph. Because triangulation may add new edges to the dual graph, the triangulated dual graph cannot have fewer edges than the original dual graph. Consequently, triRNIC is strictly stronger than RNIC.

4.3 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 enforced by this process. 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.

[2] In a related research, we studied the density of 1689 dual graph 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.

4.4 Select the appropriate RNIC: selRNIC

Algorithm 1 discussed in Section 3.3, including its improvements presented in Section 3.4, 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 7 automatically chooses the dual graph on which to apply PROCESSQ (Algorithm 1) 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.

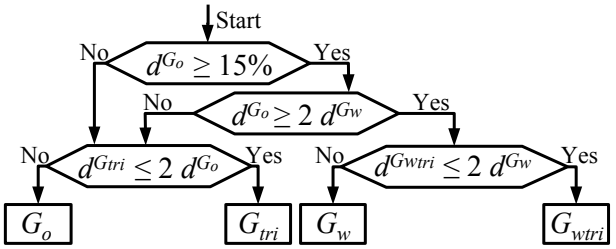

Figure 7: Selecting a dual graph for PROCESSQ for selRNIC.

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, our policy applies each operation at most once. The resulting mechanism, which we denote selRNIC, nicely ties together our techniques in a consistent and adaptive framework.

5 Related Work

NIC was proposed by Freuder and Elfe in (1996) and evaluated by them and others on binary CSPs. In (Debruyne and Bessière 2001), Debruyne and Bessiere 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. 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). Otherwise, most of the research on consistency for non-binary CSPs has focused on filtering the variables domains, not the constraints definitions, such as the study of 'variable-based' NIC (Gent, Stergiou, and Walsh 2000; Stergiou 2007).

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. 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. Figure 8 summarizes the relationships between RNIC and R($*$,m)C based properties. Bessière, Stergiou, and Walsh (2008) discussed domain-filtering properties located between GAC and R($*$,2)C.

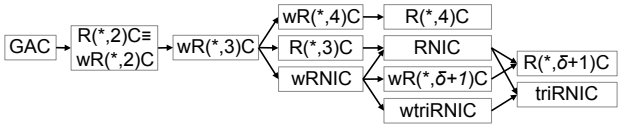

Figure 8: A partial order on RNIC, R($*$,m)C, and their variations.

The use of support structures to improve the performance of propagation in PROCESSQ is similar to (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 3.4) can be related to the Cycle-Cutset method (Dechter and Pearl 1987).

6 Experimental Results

We ran our experiments on the benchmarks of the CSP Solver Competition[3] with a time limit of one and a half hours per instance. Below, we report our most representative results organized in Tables 1, 2, and 3. In each table, we report the average density of the dual graphs (d^D) (except for GAC which operates on the hypergraph), the average number of nodes visited (#NV), the average CPU time (Time) in msec for the instances completed within the time limit. The averages are computed over only the instances completed by all compared algorithms. Thus, the values reported in the tables should be considered in light of the number of completed instances. A '-' entry 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. A 'mem' entry indicates that the corresponding algorithm ran out of memory and could not complete. This situation occurred only in Table 2. The dual graphs of these instances have over 4,200 edges, the maximum arity of the relations is $k = 10$, and the maximum domain size is $d = 42$. Such values make the polynomial space-requirement of the index-tree structures prohibitively high. Note that the weakened dual graph of those instances do not suffer from this limitation. The tables also provide: the number of completed instances (#C); the number of instances with the fastest running time (#F), where ties are awarded to all parties; and the number of instances solved backtrack free (#BF). Further, for each benchmark class, we report the number of instances in the class, the number of instances on which the averages are computed in parenthesis, the range of the number of constraints e, and the range of the density of the dual graph d^D. Finally, selRNIC is identical to RNIC in Table 1 and triRNIC on benchmark lexVg in 3. Consequently, the corresponding rows are merged.

Before we discuss the results in detail, we note that the values of nodes visited in all experiments comply with the partial order shown in Figure 8, except for one instance that can be explained by the variation of the variable ordering used during search (Table 3, lexVg, GAC and RNIC). Importantly, RNIC/selRNIC solves two large and difficult instances of the aim-200 benchmark and one instance of the ssa benchmark that *no other algorithm can solve*. Further, it solves the ssa instance in a backtrack-free manner.

Table 1 illustrates the usefulness of RNIC: it solves the largest number of problems in this set, and solves, backtrack free, the largest number of instances.

Table 1: RNIC/selRNIC completes the largest number of instances, and solves, backtrack free, the largest number of instances.

Algorithm	d^D	#NV	Time	#C	#F	#BF
aim-100 Instances: 24(6), $e \in$[150,570], $d^D \in$ [6.3%,8.1%]						
wR($*$,2)C	1.9%	383	157	19	6	5
wR($*$,3)C		111	330	20	1	7
wR($*$,4)C		107	2902	20	0	12
GAC	N/A	19034130	371588	17	4	1
RNIC/ selRNIC	6.8%	**100**	285	**22**	5	**16**
triRNIC	34.5%	**100**	1277638	9	1	9
wRNIC	1.9%	117	**118**	20	**8**	7
wtriRNIC	8.4%	108	30583	17	0	8
aim-200 Instances: 24(0), $e \in$[302,1169], $d^D \in$ [3.2%,4.2%]						
wR($*$,2)C	-	-	-	12	**10**	4
wR($*$,3)C		-	-	15	3	8
wR($*$,4)C		-	-	12	0	8
GAC	-	-	-	8	0	0
RNIC selRNIC	-	-	-	**19**	5	**13**
triRNIC	-	-	-	1	0	1
wRNIC	-	-	-	13	3	5
wtriRNIC	-	-	-	6	0	6

Table 2 illustrates the usefulness of wRNIC and wtriRNIC. As stated above, that sheer number of relations in the

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

dual graphs of the problems in this benchmark prevents us from building the index-tree structures and 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.

Table 2: RNIC is hindered by the high density of the dual graph, but its weakened versions outperform all others.

Algorithm	d^D	**#NV**	**Time**	**#C**	**#F**	**#BF**
modifiedRenault Instances: 50(22), $e \in$[147,159], $d^D \in$ [35.4%,41.6%]						
wR(∗,2)C	1.8%	248	**2445**	46	14	41
wR(∗,3)C		**75**	4887	49	4	48
wR(∗,4)C	mem	mem	mem	0	0	0
GAC	N/A	1606661	438255	23	12	4
RNIC	mem	mem	mem	0	0	0
triRNIC	mem	mem	mem	0	0	0
wRNIC	1.8%	245	4985	47	**18**	43
wtriRNIC	3.4%	**75**	181897	**50**	2	**50**
selRNIC	3.0%	221	160259	49	2	48

In both Tables 1 and 2, 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 3 does GAC outperform the other algorithms in terms of CPU time only. Interestingly, however, on lexVg, and despite the high density (72.6%) of the triangulated dual graph, triRNIC/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.)

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 1: aim-50 (24 instances), rand-10-20-10 (20 instances), dubois (13 instances), pret (8 instances).
- Table 2: 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 3: ogdVg (65 instances), ukVg (65 instances), and wordsVg (65 instances).

7 Future Work & Conclusions

Our approach opens the door to the investigation of a new type of singleton consistency properties for non-binary 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)

Table 3: GAC is best on CPU, triRNIC/selRNIC is best on #BF.

Algorithm	d^D	**#NV**	**Time**	**#C**	**#F**	**#BF**
lexVg Instances: 63(27), $e \in$[8,36], $d^D \in$ [48.5%,57.1%]						
wR(∗,2)C	52.0%	33	1117	55	4	27
wR(∗,3)C		33	13690	44	0	27
wR(∗,4)C		**4**	94413	28	0	26
GAC	N/A	23	**106**	**63**	**61**	26
RNIC	52.0%	33	6241	45	7	27
triRNIC/ selRNIC	72.6%	**4**	303804	62	7	**62**
wRNIC	52.0%	33	6239	43	1	27
wtriRNIC	72.6%	**4**	243276	59	2	59
ssa Instances: 8(3), $e \in$[177,23563], $d^D \in$ [0.1%,1.3%]						
wR(∗,2)C	0.2%	7017	5237	**6**	0	**2**
wR(∗,3)C		7017	15073	**6**	1	**2**
wR(∗,4)C		7017	60160	5	0	**2**
GAC	N/A	45927	**2770**	**6**	**5**	**2**
RNIC	1.1%	7017	158827	5	1	1
triRNIC	2.2%	**7013**	866250	4	0	0
wRNIC	0.2%	7017	11320	4	0	0
wtriRNIC	0.6%	7017	104450	4	0	0
selRNIC	1.5%	**7013**	679940	5	1	1

(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 in this paper. Further, and also for constraints defined in intension, we could generate the support tuples after applying GAC to the original CSP. For those 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 is perhaps the most noteworthy contribution of the current research: It indicates that we may be one step closer to achieving our goal.

Acknowledgments

Experiments were conducted on the equipment of the Holland Computing Center at the University of Nebraska-Lincoln. Robert Woodward was partially supported by a B.M. Goldwater Scholarship and by a Graduate Research Fellowship of the National Science Foundation.

References

Bacchus, F.; Chen, X.; Beek, P. V.; and Walsh, T. 2002. Binary vs. Non-Binary Constraints. *Artificial Intelligence* 140:1–37.

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

Bessiere, C.; Cardon, S.; Debruyne, R.; and Lecoutre, C. 2011. Efficient Algorithms for Singleton Arc Consistency. *Constraints* 16 (1):25–53.

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

Bessiere, C. 2006. *Handbook of Constraint Programming*. Elsevier. chapter Constraint Propagation.

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.

Debruyne, R., and Bessière, C. 1997. Some Practicable Filtering Techniques for the Constraint Satisfaction Problem. In *Proc. of the 15 th IJCAI*, 412–417.

Debruyne, R., and Bessière, C. 2001. Domain Filtering Consistencies. *Journal of Artificial Intelligence Research* 14:205–230.

Dechter, R., and Pearl, J. 1987. The Cycle-Cutset Method for improving Search Performance in AI Applications. In *Third IEEE Conference on AI Applications*, 224–230.

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

Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann.

Freuder, E. C., and Elfe, C. D. 1996. Neighborhood Inverse Consistency Preprocessing. In *Proc. of AAAI-96*, 202–208.

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

Gent, I.; Stergiou, K.; and Walsh, T. 2000. Decomposable Constraints. *Artificial Intelligence* 123 (1-2):133–156.

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

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

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

Karakashian, S.; Woodward, R.; Reeson, C.; Choueiry, B. Y.; and Bessiere, C. 2010. A First Practical Algorithm for High Levels of Relational Consistency. In *24th AAAI Conference on Artificial Intelligence (AAAI 10)*, 101–107.

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

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

Maier, D. 1983. *The Theory of Relational Databases*. Pitman Publishing Limited.

Stergiou, K. 2007. 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*, 215–222.