Strong Inverse Consistencies for Non-binary CSPs

Kostas Stergiou
Department of Information and Communication Systems Engineering
University of the Aegean, Samos, Greece
email: konsterg@aegean.gr

Abstract

Domain filtering local consistencies, such as inverse consistencies, that only delete values and do not add new constraints are particularly useful in Constraint Programming. Although many such consistencies for binary constraints have been proposed and evaluated, the situation with non-binary constraints is quite different. Only very recently have domain filtering consistencies stronger than GAC started to attract interest. Following this line of research, we define a number of strong inverse consistencies for non-binary constraints and compare their pruning power. We show that three of these consistencies are equivalent to maxRPC in binary CSPs while another is equivalent to PIC. We also describe a generic algorithm for inverse consistencies in non-binary CSPs and show how it can be instantiated to enforce some of the proposed consistencies. Finally, we make a preliminary empirical study that demonstrates the potential of strong inverse consistencies.

1 Introduction

One of the great strengths of Constraint Programming is the exploitation of local consistency techniques to prune inconsistent values from the domains of variables and thus avoid fruitless exploration of the search tree. The most widely studied and used local consistency is generalized arc consistency (GAC). It is widely accepted that "relation filtering" consistencies which alter the structure of the constraint graph or the constraints' relations (e.g. path consistency) tend to be less practical than "domain filtering" consistencies which only remove values from the domains of the variables. As a result, many strong domain filtering consistencies for binary constraints have been proposed and evaluated. For example, inverse and singleton consistencies [5, 3, 12]. In contrast, little work had been done on such consistencies for non-binary constraints until very recently, whereas a number of consistencies that are stronger than GAC, but not domain filtering, have been developed. For example, pairwise consistency [7], hyper-m-consistency [8], relational consistency [11], and ω-consistency [9]. However, these consistencies are rarely used in practice, mainly because they have a high space complexity.

Very recently, three domain filtering consistencies for non-binary CSPs were introduced and evaluated theoretically and empirically. These are relational path inverse consistency (rPIC), restricted pairwise consistency (RPWC), and max restricted pairwise consistency (maxRPWC)[1] [10, 2]. All these are stronger than GAC and display promising performance on certain non-binary problems. Continuing along the same lines of work, we propose a number of strong inverse consistencies for non-binary constraints and study them theoretically and empirically. These are relational $(1,3)$-consistency, relational neighborhood inverse consistency, inverse ω- consistency, extended inverse ω-consistency, and max restricted 3-wise consistency. To derive these consistencies we are mainly inspired by known relation-filtering consistencies for non-binary problems. In our theoretical study we compare the pruning power of these consistencies, most of which are stronger than maxRPWC, and show what they correspond to when restricted to binary constraints. We also describe an algorithm that can be used to apply the proposed consistencies. Finally we give preliminary experimental results that demonstrate the potential of strong inverse consistencies.

2 Background

A *Constraint Satisfaction Problem* (CSP) P is defined as a tuple (X, D, C) where: $X = \{x_1, \ldots, x_n\}$ is a finite set of n variables, $D = \{D(x_1), \ldots, D(x_n)\}$ is a set of domains, and $C = \{c_1, \ldots, c_e\}$ is a set of e constraints. For each variable $x_i \in X$, $D(x_i)$ is the finite domain of its possible values. Each constraint $c_i \in C$ is defined as a pair $(var(c_i), rel(c_i))$, where $var(c_i) = \{x_{j_1}, \ldots, x_{j_k}\}$ is an ordered subset of X called the *scope* of c_i, and $rel(c_i)$ is a subset of the *Cartesian* product $D(x_{j_1})\text{x}\ldots\text{x}D(x_{j_k})$ that specifies the allowed combinations of values for the vari-

[1] maxRPWC was called pairwise inverse consistency in [10].

ables in $var(c_i)$. Each tuple $\tau \in rel(c_i)$ is an ordered list of values $(a_1, \ldots, a_k)$. A tuple is *valid* iff none of the values in the tuple has been removed from the domain of the corresponding variable. A constraint c_i can be either defined *extensionally* by explicitly giving relation $rel(c_i)$, or (usually) *intensionally* by implicitly specifying $rel(c_i)$ through a predicate or arithmetic function. For any two constraints c_i and c_j, the set of variables that are involved in both constraints is denoted by $var(c_i) \cap var(c_j)$. If this set is not empty, the constraints *intersect*.

The assignment of value a to variable x_i is denoted by (x_i, a). Any tuple $\tau = (a_1, \ldots, a_k)$ can be viewed as a set of value to variable assignments $\{(x_1, a_1), \ldots, (x_k, a_k)\}$. In this way, an assignment of values to a set of variables $X' \subseteq X$ is a tuple over X'. The set of variables over which a tuple τ is defined is $var(\tau)$. For any subset var' of $var(\tau)$, $\tau[var']$ is the sub-tuple of τ that includes only assignments to the variables in var'. A tuple τ is *consistent*, iff it is valid and for all constraints c_i, where $var(c_i) \subseteq var(\tau)$, $\tau[var(c_i)] \in rel(c_i)$. A *solution* to a CSP (X, D, C) is a consistent tuple assigning all variables in X.

A value $a \in D(x_i)$ is consistent with a constraint c_j, where $x_i \in var(c_j)$, iff $\exists \tau \in rel(c_j)$ such that $\tau[x_i] = a$ and τ is valid. In this case, we say that τ is a GAC-support of (x_i, a) in c_j. A constraint c_j is *Generalized Arc Consistent* (GAC) iff $\forall\, x_i \in var(c_j)$, $\forall\, a \in D(x_i)$, there exists a GAC-support for a in c_j. A problem is GAC iff there is no empty domain in D and all the constraints in C are GAC. In binary CSPs, GAC is referred to a *arc consistency* (AC).

Since the allowed tuples of constraints are defined as relations, standard relational operators can be used. The *projection* $\Pi_{var'}\tau$ of a tuple $\tau \in rel(c_i)$ on var' is the subtuple $\tau[var']$. Accordingly, the projection of a constraint c_i on a set of variables var', where $var' \subseteq var(c_i)$ is a new constraint c' where $var(c') = var'$ and $rel(c') = \Pi_{var'} rel(c_i)$. The *join* of two constraints c_i and c_j is a new constraint, denoted by $c_i \bowtie c_j$, where $var(c_i \bowtie c_j) = var(c_i) \cup var(c_j)$ and $rel(c_i \bowtie c_j) = rel(c_i) \bowtie rel(c_j)$.

2.1 Local Consistencies

We now briefly review the most common local consistencies for binary and non-binary CSPs.

A binary problem is (i, j) *consistent* iff it has non-empty domains and any consistent instantiation of i variables can be extended to a consistent instantiation involving j additional variables [4]. A problem is *strong* (i, j)*-consistent* iff it is (k, j) consistent for all $k \leq i$. A problem is *arc consistent* (AC) iff it is $(1, 1)$-consistent. A problem is (strong) *path consistent* (PC) iff it is (strong) $(2, 1)$-consistent. A problem is *path inverse consistent* (PIC) iff it is $(1, 2)$-consistent [5]. A problem is *max restricted path consistent* (maxRPC) iff it is (1,1)-consistent and for each value (x_i, a) and variable x_j constrained with x_i, there exists a value $b \in D(x_j)$ that is an AC-support of (x_i, a) and this pair of values is path consistent (i.e. it can be consistently extended to any third variable). A problem is *inverse* m*-consistent* iff it is $(1, m)$ consistent. A problem is *neighborhood inverse consistent* (NIC) iff any consistent instantiation of a variable x_i can be extended to a consistent instantiation of all the variables in x_i's neighborhood. A problem P is *singleton arc consistent* (SAC) [3] iff it has non-empty domains and for any instantiation (x_i, a) of a variable $x_i \in X$, the resulting subproblem can be made AC.

Some local consistencies for binary CSPs can be easily extended to non-binary problems. For example, SAC has been extended to SGAC. However, for other consistencies (e.g. PIC and maxRPC) this extension is not straightforward. In the case of NIC there are two alternative extensions to non-binary constraints. To determine if a value $a \in D(x_i)$ is NIC, we can consider the subproblem consisting of the set of variables $neigh(x_i) = \{x_{i_1}, \ldots, x_{i_m}\}$ involved in a constraint with x_i and the constraints that only include variables from $neigh(x_i)$. Alternatively, we can consider the subproblem consisting of variables $neigh(x_i)$ and all the constraints that include any of these variables (and possibly other variables as well). In the rest of this paper we follow the first definition of NIC for non-binary constraints.

A problem is *relationally arc consistent* (rel AC) iff any consistent instantiation for all but one of the variables in a constraint can be extended to the final variable so as to satisfy the constraint [11]. A problem is *relationally path-consistent* (rel PC) iff any consistent instantiation for all but one of the variables in a pair of constraints can be extended to the final variable so as to satisfy both constraints. A problem is *relationally* m*-consistent* iff any consistent instantiation for all but one of the variables in a set of m distinct constraints can be extended to the final variable so as to satisfy all m constraints. A problem is *relationally* (i, m)*-consistent* iff any consistent instantiation for i of the variables in a set of m constraints can be extended to all the variables in the set.

A non-binary problem is *pairwise consistent* (PWC) [8]) iff it has non-empty relations and any consistent tuple in a constraint c_i can be consistently extended to any other constraint [7]. PWC has been generalized to *k-wise consistency* [6] and *hyper-*m*-consistency* [8]. A problem is k-wise consistent iff any consistent tuple for a constraint can be consistently extended to any $k-1$ other constraints. A problem is hyper-m-consistent iff any consistent combination of tuples for m-1 constraints can be consistently extended to any m^{th} constraint.

A problem is ω*-consistent* iff any tuple in a constraint c_i can be consistently extended to any other constraint c_j and to all constraints c_k such that $var(c_k) \subseteq var(c_i) \cup var(c_j)$

[9]. A problem is *generalized dual arc consistent* iff any tuple in a constraint c_i can be consistently extended to any other constraint c_j and satisfy all constraints c_k such that $var(c_k) \cap (var(c_i) \cup var(c_j)) \neq \emptyset$ [9].

Following [3], a consistency property A is stronger than B iff in any problem in which A holds then B holds, and strictly stronger (written $A \rightarrow B$) iff it is stronger and there is at least one problem in which B holds but A does not. A local consistency property A is incomparable with B (written $A \otimes B$) iff A is not stronger than B nor vice versa. Finally, a local consistency property A is equivalent to B (written $A \leftrightarrow B$) iff A is stronger than B and vice versa. Note that relationships $\rightarrow$ and $\leftrightarrow$ are transitive.

3 New Inverse Consistencies

In practice, most of the strong local consistency techniques discussed in the previous section have prohibitive space and time complexities. Freuder proposed inverse consistencies as a way to overcome the space problem [5]. Such consistencies require limited space as they only prune domains. When an inverse local consistency is enforced, it removes from the domain of a variable the values that cannot be consistently extended to some additional variables.

Until the very recent introduction of rPIC, RPWC, and maxRPWC, the study of inverse consistencies had been restricted to binary constraints. Experimental results demonstrated that applying maxRPWC, which is the strongest among the three consistencies, is more efficient than applying the other consistencies [10, 2]. We will now define a number of new inverse consistencies for non-binary problems. These are all strictly stronger than GAC. That is, if applied, they will remove any value that is not GAC. Also, each consistency may remove some additional values according to the property it enforces. For any consistency `IC`, we say that a variable x_i is `IC` iff any value $a \in D(x_i)$ is `IC`. A CSP is `IC` iff there is no empty domain and all variables are `IC`. The following definitions specify when a value is `IC` for a number of different inverse consistencies. For completeness we include the definitions of rPIC and maxRPWC.

Definition 3.1 [11, 10] A value $a \in D(x_i)$ is *relational Path Inverse Consistent* (rPIC) iff $\forall c_j \in C$, where $x_i \in var(c_j)$, and for each $c_k \in C$, there exists a GAC-support τ of (x_i, a) in $rel(c_j)$ and a valid tuple $\tau' \in rel(c_k)$ such that $\tau[var(c_j) \cap var(c_k)] = \tau'[var(c_j) \cap var(c_k)]$.

If rPIC is applied on a variable x_i it will remove any value $a \in D(x_i)$ such that for some constraint c_j where x_i participates, no GAC-support of (x_i, a) can be extended to a valid tuple in some other constraint c_k that intersects with c_j. Note that if the two constraints do not intersect then any valid tuple in $rel(c_j)$ can be extended to any valid tuple in $rel(c_k)$. Apart from rPIC we can consider other, stronger, inverse relational consistencies. We now define relational $(1,3)$-consistency (derived from [11]) and relational NIC.

Definition 3.2 [11] A value $a \in D(x_i)$ is *relational* $(1,3)$-*Consistent* (r$(1,3)$C) iff $\forall c_j \in C$, where $x_i \in var(c_j)$, and for each pair of constraints $c_k, c_l \in C$, there exists a GAC-support τ of (x_i, a) in $rel(c_j)$ and valid tuples $\tau' \in rel(c_k), \tau'' \in rel(c_l)$ s.t. $\tau[var(c_j) \cap var(c_k)] = \tau'[var(c_j) \cap var(c_k)]$, $\tau[var(c_j) \cap var(c_l)] = \tau''[var(c_j) \cap var(c_l)]$, $\tau'[var(c_k) \cap var(c_l)] = \tau''[var(c_k) \cap var(c_l)]$.

If r$(1,3)$C is applied on a variable x_i it will remove any value $a \in D(x_i)$ such that for some constraint c_j where x_i participates, no GAC-support of (x_i, a) can be extended to valid tuples in some pair of extra constraints.

Definition 3.3 A value $a \in D(x_i)$ is *relational Neighborhood Inverse Consistent* (rNIC) iff $\forall c_j \in C$, where $x_i \in var(c_j)$, there exists a GAC-support τ of (x_i, a) in $rel(c_j)$ that can be extended to a solution of the subproblem consisting of the set of variables $X_j = \{var(c_j) \cup var(c_{j_1}) \cup \ldots \cup var(c_{j_m})\}$, where $c_{j_1}, \ldots, c_{j_m}$ are the constraints that intersect with c_j.

If rNIC is applied on a variable x_i it will remove any value $a \in D(x_i)$ such that for some constraint c_j where x_i participates, no GAC-support of (x_i, a) can be extended to a consistent instantiation of all variables involved in a constraint that intersects with c_j so that all constraints between these variables are satisfied.

Definition 3.4 [10, 2] A value $a \in D(x_i)$ is *max Restricted Pairwise Consistent* (maxRPWC) iff $\forall c_j \in C$, where $x_i \in var(c_j)$, there exists a GAC-support τ of (x_i, a) in $rel(c_j)$ s.t. $\forall c_k \in C$, there exists a PW-support τ' of τ in $rel(c_k)$. A tuple τ' is a PW-support of τ iff it is valid and $\tau[var(c_j) \cap var(c_k)] = \tau'[var(c_j) \cap var(c_k)]$.

If maxRPWC is applied on a variable x_i it will remove any value $a \in D(x_i)$ such that for some constraint c_j where x_i participates, no GAC-support of (x_i, a) can be extended to a valid tuple in every other constraint (intersecting c_j).

Definition 3.5 A value $a \in D(x_i)$ is *inverse ω-consistent* (IωC) iff $\forall c_j \in C$, where $x_i \in var(c_j)$, there exists a GAC-support τ of (x_i, a) in $rel(c_j)$ s.t. $\forall c_k \in C$, there exists an ω-support τ' of τ in $rel(c_k)$. A tuple τ' is an ω-support of τ iff it is a PW-support of τ and $\forall c_l \in C$, where $var(c_l) \subseteq var(c_j) \cup var(c_k)$, $(\tau \bowtie \tau')[var(c_l)] \in rel(c_l)$.

If IωC is applied on a variable x_i it will remove any value $a \in D(x_i)$ such that for some constraint c_j where x_i participates, no GAC-support of (x_i, a) can be extended to a valid tuple in every constraint c_k that intersects with c_j and, at the same time, satisfy all constraints defined on variables $var(c_j) \cup var(c_k)$.

Definition 3.6 A value $a \in D(x_i)$ is *extended inverse ω-consistent* (EIωC) iff $\forall c_j \in C$, where $x_i \in var(c_j)$, there exists a GAC-support τ of (x_i, a) in $rel(c_j)$ s.t. $\forall c_k \in C$, there exists an extended ω-support τ' of τ in $rel(c_k)$. A tuple τ' is an *extended* ω-support of τ iff it is a PW-support of τ and $\forall c_l \in C$, where $var(c_j) \cap var(c_l) \neq \emptyset$ and $var(c_k) \cap var(c_l) \neq \emptyset$, $\Pi_{var(c_l)\cap(var(c_j)\cup var(c_k))}(\tau \bowtie \tau') \in \Pi_{var(c_l)\cap(var(c_j)\cup var(c_k))} rel(c_l)$ and can be extended to a valid tuple in $rel(c_l)$.

If EIωC is applied on a variable x_i it will remove any value $a \in D(x_i)$ such that for some constraint c_j where x_i participates, no GAC-support of (x_i, a) can be extended to a valid tuple in each constraint c_k that intersects with c_j and, at the same time, satisfy all constraints that intersect with both c_j and c_k. The difference between IωC and EIωC is that the former considers a constraint c_l only if it includes variables among $var(c_j) \cup var(c_k)$, while the latter also considers some constraints that include variables among $var(c_j) \cup var(c_k)$ and other variables as well.

Definition 3.7 A value $a \in D(x_i)$ is *max Restricted 3-wise Consistent* (maxR3WC) iff $\forall c_j \in C$, where $x_i \in var(c_j)$, there exists a GAC-support τ of (x_i, a) in $rel(c_j)$ s.t. $\forall c_k, c_l \in C$ there exist valid tuples $\tau' \in rel(c_k), \tau'' \in rel(c_l)$ s.t. $\tau[var(c_j) \cap var(c_k)] = \tau'[var(c_j) \cap var(c_k)]$, $\tau[var(c_j) \cap var(c_l)] = \tau''[var(c_j) \cap var(c_l)]$, $\tau'[var(c_k) \cap var(c_l)] = \tau''[var(c_k) \cap var(c_l)]$.

If maxR3WC is applied on a variable x_i it will remove any value $a \in D(x_i)$ such that for some constraint c_j where x_i participates, no GAC-support of (x_i, a) can be extended to valid tuples in every pair of other constraints.

3.1 Theoretical Study

To clarify the above definitions, we first give an example that demonstrates which values are deleted by the application of these consistencies.

Example 3.1 Figure 1a shows a problem with 6 variables and 4 constraints with the given allowed tuples. All domains are $\{0, 1\}$ except $D(x_1)$ which is $\{0, 1, 2\}$. Assume that we are trying to apply a given inverse consistency on variable x_1. All values of x_1 are GAC as they are GAC-supported in both c_1 and c_2. Value 0 is not rPIC (and thus not maxRPWC) as none of its GAC-supports in c_1 can be consistently extended to c_2. Values 1 and 2 are maxRPWC as their GAC supports take the same values in the variables shared by c_1 and c_2. But value 1 is not IωC as its GAC supports in c_1 and c_2 do not satisfy constraint c_4. Value 2 is IωC but it is not EIωC as its GAC-supports satisfy c_4 but do not satisfy constraint c_3. No value of x_1 is rNIC as in this problem where c_1 intersects with all other constraints, rNIC requires that these values participate in a solution.

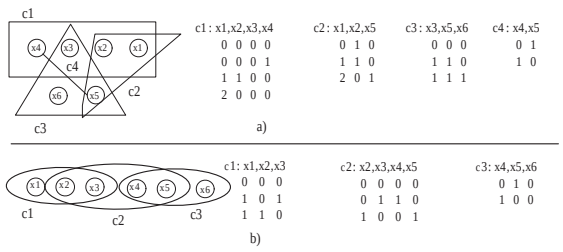

Figure 1. Applying inverse consistencies on non-binary problems.

Now consider the problem depicted in Figure 1b with five 0-1 variables and one variable (x_6) with domain $\{0\}$. Value 0 of x_1 has tuple $(0, 0, 0)$ as GAC-support in c_1. This tuple can be extended to tuple $(0, 0, 0, 0)$ in c_2 and there are no constraints that intersect with both c_1 and c_2. Therefore $(x_1, 0)$ is EIωC. However, the GAC support of 0 cannot be consistently extended to the pair of constraints c_2, c_3 since tuple $(0, 0, 0, 0)$ of c_2 has no PW-support in c_3. Hence, $(x_1, 0)$ is not maxR3WC (or r$(1, 3)$C).

Theorem 3.1 *On problems with non-binary constraints the following relationships hold:*
1) EIωC $\rightarrow$ IωC $\rightarrow$ maxRPWC $\rightarrow$ rPIC
2) maxR3WC $\rightarrow$ maxRPWC and EIω $\otimes$ maxR3WC $\otimes$ IωC
3) r$(1, 3)$C is incomparable to maxRPWC, IωC, EIωC, and maxR3WC $\rightarrow$ r$(1, 3)$C $\rightarrow$ rPIC
4) NIC is incomparable to rPIC, r$(1, 3)$C, maxRPWC, IωC, EIωC, maxR3WC and rNIC $\rightarrow$ NIC
5) rNIC $\rightarrow$ EIωC and maxR3WC $\otimes$ rNIC $\otimes$ r$(1, 3)$C

Proof:

1) By definition, the "stronger than" relationship holds between maxR3WC, EIωC, IωC, maxRPWC, and rPIC. To show maxR3WC $\rightarrow$ EIωC $\rightarrow$ IωC $\rightarrow$ maxRPWC, consider the problems in Example 3.1. The relationship between maxRPC and rPIC was proved in [10].

2) By definition, maxR3WC is stronger than maxRPWC. For strictness consider the problem in Example 3.1b which is maxRPWC but not maxR3WC. To show that maxR3WC is incomparable to EIωC and IωC first consider the same problem which is EIωC (and IωC). Now consider the problem of Figure 2a with 5 0-1 variables. This is maxR3WC but not IωC.

3) To show that r$(1, 3)$C is incomparable to EIωC, IωC and maxRPWC, it suffices to show that r$(1, 3)$C can be stronger than EIωC and weaker than maxRPWC. First consider the second problem in Example 3.1. This problem is EIωC but it is not r$(1, 3)$C. Now consider the problem in Figure 2b with 6 variables and 4 constraints intersecting on variables x_2 and x_3. Value 0 of x_2 is r$(1, 3)$C but it is

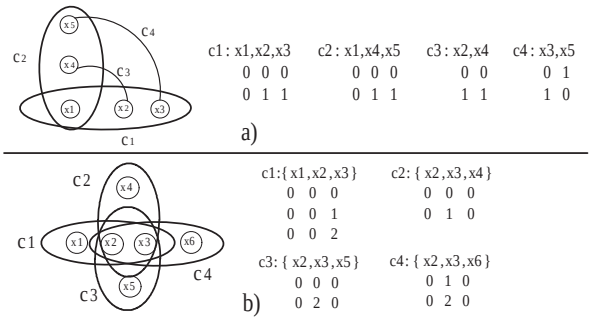

Figure 2. A problem that is maxR3WC but not IωC (a). A problem that is r$(1,3)$C but not maxRPWC (b).

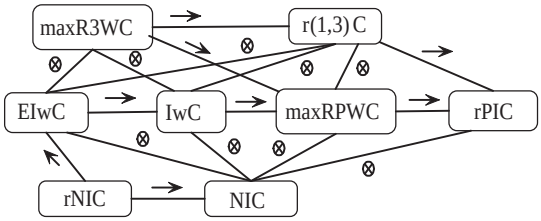

Figure 3. Relationships between inverse consistencies for non-binary CSPs.

not maxRPWC. By definition, maxR3WC is stronger than r$(1,3)$C. To show strictness consider the example of Figure 2 where value 0 of x_2 is r$(1,3)$C but it is not maxR3WC. By definition, r$(1,3)$C is stronger than rPIC. To show strictness consider the second problem in Example 3.1. This problem is rPIC but it is not r$(1,3)$C.

4) To prove that NIC is incomparable to rPIC, maxRPWC, IωC, EIωC, and maxR3WC it suffices to show that NIC can be weaker than rPIC and stronger than maxR3WC. To show the former, consider a problem with two constraints c_1, c_2, where $var(c_1) = \{x_1, x_2, x_3\}$ and $rel(c_1) = \{(0,0,0), (1,1,0), (0,1,1)\}$, $var(c_2) = \{x_1, x_2, x_4\}$ and $rel(c_2) = \{(0,0,0), (1,1,0), (1,0,1)\}$. This problem is NIC but it is not rPIC. To show the latter, consider a clique of six variables where all constraints are binary $\neq$ constraints and all domains are $\{0, \ldots, 4\}$. This problem is maxR3WC but not NIC. The same examples prove that NIC is incomparable to r$(1,3)$C.

To prove that rNIC $\rightarrow$ NIC consider a problem that is rNIC. Any assignment of a variable x_i has a GAC-support τ in each constraint c_j which involves x_i that can be consistently extended to all variables involved in constraints intersecting with c_j. Therefore, τ can be consistently extended to all variables involved in a constraint with x_i, as these constraints intersect (on at least x_i) with c_j. Hence, the problem is NIC. To show strictness, consider the previous example. This is NIC but not rNIC.

5) To prove that rNIC is incomparable to maxR3WC and r$(1,3)$C first consider again the binary problem with a clique of six variables. This is maxR3WC but not rNIC. Now consider the second problem in Example 3.1. This is rNIC but not r$(1,3)$C.

To prove rNIC $\rightarrow$ EIωC consider a problem that is rNIC. Any assignment of a variable x_i has a GAC-support τ in each constraint c_j which involves x_i that can be consistently extended to all variables involved in constraints intersecting with c_j. Therefore, τ can be extended to any constraint c_k intersecting with c_j s.t. all constraints that intersect with both c_j and c_k are satisfied. Hence, the problem is EIωC. To show strictness, consider again the binary problem with a clique of six variables. This is EIωC but not rNIC. □

Figure 3 summarizes the relationships between the various consistencies. For clarity, some relationships are omitted.

3.1.1 Binary Constraints

A natural question is what the aforementioned inverse consistencies correspond to in binary CSPs. In [10] it was shown that rPIC and maxRPWC are equivalent to GAC when all constraints intersect on at most one variable. If we assume that multiple constraints on the same variables are combined into one, then this is the case with binary constraints. Therefore, in binary problems rPIC and maxRPWC reduce to AC. We show that when restricted to binary constraints, maxR3WC, IωC and EIωC are equivalent to maxRPC while r$(1,3)$C is equivalent to PIC.

Theorem 3.2 *On binary CSPs we have maxR3WC* $\leftrightarrow$ *EIωC* $\leftrightarrow$ *IωC* $\leftrightarrow$ *maxRPC and r$(1,3)$C* $\leftrightarrow$ *PIC.*

Proof: To show IωC $\leftrightarrow$ maxRPC it suffices to show that if a value is deleted by maxRPC then it is also deleted by IωC, and vice versa. Consider a value $a \in x_i$ that is removed by maxRPC. Value a is removed because it is either not AC or because there exists a variable x_j constrained with x_i for which there is no value $b \in D(x_j)$ such that the pair $< a, b >$ is path consistent. In the former case, a will be removed by IωC since IωC is stronger than GAC (i.e. AC in binary CSPs). In the latter case, take any AC-support $b \in D(x_j)$ of (x_i, a). Since the pair $< a, b >$ is not path consistent there must be a variable x_l such that no value in $D(x_l)$ is compatible with both (x_i, a) and (x_j, b). Assume that c is the constraint between x_i and x_j and c' is the constraint between x_i and x_l. We cannot find AC-supports for a in $D(x_j)$ and $D(x_l)$ so that these supports satisfy the constraints on $var(c) \cup var(c')$, i.e. the constraint between x_j and x_l. Hence, value a is not IωC.

Now consider a value $a \in D(x_i)$ that is deleted by IωC. If a is deleted because it is not AC then maxRPC will obviously delete it. Otherwise, there must be a constraint c involving x_i and a variable x_j such that no AC-support of (x_i, a) in $D(x_j)$ can be consistently extended to any constraint c' that intersects with c so that the constraints on $var(c) \cup var(c')$ are satisfied. Take such a constraint c' and, without loss of generality, assume that $var(c') = \{x_j, x_l\}$. As we only have binary constraints, the only other constraint that can exist among variables $var(c) \cup var(c')$ is the one between x_i and x_l. Value (x_i, a) cannot be be consistently extended to x_j and x_l so that all constraints between the three variables are satisfied. Hence, a is not maxRPC.

We now show that in binary problems EIωC and maxR3WC are equivalent to IωC. Assume that a binary problem is IωC. Then any assignment (x_i, a) can be consistently extended to any constraint c that includes x_i and any other constraint c' that intersects with c so that all constraints between variables $var(c) \cup var(c')$ are satisfied. Since there is no constraint that intersects with both c and c' and includes additional variables (as all constraints are binary), (x_i, a) is also EIωC. Now consider any third constraint c''. If this intersects with both c and c' then, since (x_i, a) is IωC, there exists an AC-support of (x_i, a) in c that can be consistently extended to both c' and c''. If c'' intersects only with one of c,c' (say c') then any valid tuple of c' can be consistently extended to c'' since the problem is IωC, and hence AC. Therefore, in any case, (x_i, a) is maxR3WC.

We now show that r$(1, 3)$C is equivalent to PIC. Consider a value $a \in D(x_i)$ that is removed by PIC. It is removed either because it is not AC or because it cannot be extended to some pair of variables x_j and x_l so that the constraints between all three variables are satisfied. In the former case, a will be removed by r$(1, 3)$C since r$(1, 3)$C is stronger than GAC. In the latter case no AC-support of a in $D(x_j)$ can be consistently extended to a value in $D(x_l)$ so that the constraint between x_i and x_l is satisfied. Hence, value a is not r$(1, 3)$C. Now consider a value $a \in D(x_i)$ that is deleted by r$(1, 3)$C. There must be a constraint c involving x_i and some other variable x_j such that no AC-support of a in $D(x_j)$ can be consistently extended to some pair of constraints c' and c''. There are two cases depending on whether the three constraints form a triangle (i.e. they are the three constraints involving x_i, x_j and a third variable x_l). If they do not form a triangle then a is removed because it is not AC, in which case PIC will also remove it. If the constraints form a triangle then a cannot be be consistently extended to x_j and a third variable x_l so that all constraints between the three variables are satisfied. Hence, a is not PIC. □

4 An Algorithm for Inverse Consistencies

A generic AC-7 based algorithm for inverse local consistencies in binary CSPs was proposed in [12]. A generic GAC-3 based algorithm for inverse consistencies in non-binary CSPs was given in [10] and [2]. Also, instantiations of this algorithm that can be used to apply maxRPWC, rPIC and RPWC were presented. Here we recall the generic algorithm using a slightly different description and show how it can be instantiated to apply IωC, EIωC, and maxR3WC (Figure 4). Similar algorithms can be used to apply rPIC (see [2]) and r$(1, 3)$C. The presentation of these algorithms is omitted because of limited space. Algorithms for NIC and rNIC in general require search, as the neighborhood of a variable can be very large.

Algorithm `InvCons` takes as input a (non-binary) CSP `P` and a specified inverse consistency `IC`, and enforces `IC` on `P`. `InvCons` uses a list Q of constraints to propagate value deletions, and works as follows. Initially, all constraints are added to Q. Then constraints are sequentially removed from Q and the domains of the variables involved in these constraints are revised. For each such constraint c_j and variable x_i, the revision is performed using function `Revise(`x_i`,`c_j`,IC)`. If after the revision the domain of x_i becomes empty then the algorithm detects the inconsistency and terminates. Otherwise, if the domain of x_i is pruned then each constraint c_k involving x_i and each constraint intersecting with c_k will be put in Q. Note that in the case of maxRPWC the intersection must be on more than one variable. If Q becomes empty, the algorithm terminates having successfully enforced `IC` on `P`.

In function `Revise`, for each value a in $D(x_i)$, we first look for a GAC-support in $rel(c_j)$ (line 3). Following GAC2001/3.1 [1], for each constraint c_j and each $a \in D(x_i)$, where $x_i \in var(c_j)$, we keep a pointer $lastGAC_{x_i,a,c_j}$ (initialized to the first tuple in $rel(c_j)$). This is now the most recently discovered tuple in $rel(c_j)$ that GAC-supports (x_i, a) **and**, depending on `IC`, has some extra property. For instance, if `IC` is maxRPWC (resp. IωC) then $lastGAC_{x_i,a,c_j}$ must have PW-supports (resp. ω-supports) in all constraints that intersect with c_j. If $lastGAC_{x_i,a,c_j}$ is valid then we know that a is GAC-supported. Otherwise, we look for a new GAC-support starting from the tuple immediately after $lastGAC_{x_i,a,c_j}$ in the lexicographic order. If $lastGAC_{x_i,a,c_j}$ is valid or a new GAC-support is found then function `Seek_Support` is called to check if this GAC-support (tuple τ) satisfies the extra property of `IC`.

The implementation of `Seek_Support` depends on the consistency being enforced. For maxRPWC, IωC, and EIωC, `Seek_Support` iterates over each constraint c_k that intersects with c_jFor each such constraint it searches for a tuple τ' that is a PW-support, IωC-support, or extended IωC-support, respectively, of τ. If such tuples are found for all intersecting constraints then `Seek_Support` returns TRUE and $lastGAC_{x_i,a,c_j}$ is updated. If no `IC`-support τ' is found on some intersecting constraint, indicated by

function `InvCons(P,IC)`
1:put all constraints in Q;
2:**while** Q is not empty
3: pop constraint c_j from Q;
4: **for** each variable $x_i \in var(c_j)$
5: **if** `Revise(`x_i`,`c_j`,IC)`> 0 **then**
6: **if** $D(x_i)$ is empty **then return** INCONSISTENCY;
7: **for** each $c_k \in C$ s.t. $x_i \in var(c_k)$
8: put in Q each $c_l \in C$ s.t. $|var(c_l) \cap var(c_k)| > 0$;
9: put c_k in Q;
10:**return** CONSISTENCY;

function `Revise(`x_i,c_j,IC)
1:**for** each value $a \in D(x_i)$
2: PW $\leftarrow$ FALSE;
3: **for** each valid $\tau (\in rel(c_j)) \geq_l lastGAC_{x_i,a,c_j}$, s.t. $\tau[x_i] = a$
4: **if** `Seek_Support(`x_i,c_j,τ,IC) **then**
5: $lastGAC_{x_i,a,c_j} \leftarrow \tau$;
6: PW$\leftarrow$ TRUE; **break**;
7: **if** $\neg$PW **then** remove a from $D(x_i)$;
8:**return** number of deleted values;

function `Seek_Support(`x_i,c_j,τ,IωC)
1:**for** each $c_k \in C$ s.t. $|var(c_j) \cap var(c_k)| > 0$
2: **for** each $\tau' (\in rel(c_k))$
3: **if** τ' is valid **and** $\tau[var(c_j) \cap var(c_k)] = \tau'[var(c_j) \cap var(c_k)]$
4: ωC$\leftarrow$ TRUE;
5: **for** each $c_l \in C$, s.t. $var(c_l) \subseteq var(c_j) \cup var(c_k)$
6: **if** $(\tau \bowtie \tau')[var(c_l)] \notin rel(c_l)$
7: **then** ωC$\leftarrow$ FALSE; **break**;
8: **if** ωC **then break**;
9: **if** $\tau' = NIL$ **then return** FALSE;
10:**return** TRUE;

function `Seek_Support(`x_i,c_j,τ,EIωC)
1:**for** each $c_k \in C$ s.t. $|var(c_j) \cap var(c_k)| > 0$
2: **for** each $\tau' (\in rel(c_k))$
3: **if** τ' is valid **and** $\tau[var(c_j) \cap var(c_k)] = \tau'[var(c_j) \cap var(c_k)]$
4: EωC$\leftarrow$ TRUE;
5: **for** each $c_l \in C$, s.t.
$var(c_j) \cap var(c_l) \neq \emptyset \wedge var(c_k) \cap var(c_l) \neq \emptyset$
6: **if** $\Pi_{var(c_l) \cap (var(c_j) \cup var(c_k))}(\tau \bowtie \tau') \notin$
$\Pi_{var(c_l) \cap (var(c_j) \cup var(c_k))} rel(c_l)$
or $\Pi_{var(c_l) \cap (var(c_j) \cup var(c_k))}$
cannot be extended to a valid tuple in $rel(c_l)$
7: **then** EωC$\leftarrow$ FALSE; **break**;
8: **if** EωC **then break**;
9: **if** $\tau' = NIL$ **then return** FALSE;
10:**return** TRUE;

function `Seek_Support(`x_i,c_j,τ,maxR3WC)
1:**for** each $c_k \in C$ s.t. $|var(c_j) \cap var(c_k)| > 0$
2: **for** each valid $\tau' (\in rel(c_k))$
s.t. $\tau[var(c_j) \cap var(c_k)] = \tau'[var(c_j) \cap var(c_k)]$
3: 3W$\leftarrow$ TRUE;
4: **for** each $c_l \in C$
s.t. $|var(c_j) \cap var(c_l)| > 0 \vee |var(c_k) \cap var(c_l)| > 0$
5: **if** $\nexists$ valid $\tau'' (\in rel(c_l))$ such that
$\tau[var(c_j) \cap var(c_l)] = \tau''[var(c_j) \cap var(c_l)]$ and
$\tau'[var(c_k) \cap var(c_l)] = \tau''[var(c_k) \cap var(c_l)]$
6: **then** 3W$\leftarrow$ FALSE; **break**;
7: **if** 3W **then break**;
8: **if** $\tau' = NIL$ **then return** FALSE;
9:**return** TRUE;

Figure 4. A generic algorithm for inverse consistencies and its instantiations.

τ' becoming NIL, then `Seek_Support` returns FALSE and the algorithm looks for a new GAC-support in function `Revise`. If no GAC-support that satisfies the property of `IC` is found, a is removed from $D(x_i)$.

In the case of maxR3WC, `Seek_Support` iterates over each constraint c_k that intersects with c_j and searches for a PW-support of τ in $rel(c_k)$. If such a tuple τ' is found, the algorithm iterates over each constraint c_l that intersects with c_j or c_k (or both) and searches for a tuple $\tau'' \in rel(c_l)$ that is a PW-support of both τ and τ'. In case c_l does not intersect with c_j (resp. c_k) then obviously any valid $\tau'' \in rel(c_l)$ is a PW-support of τ (resp. τ'). If such a pair of tuples is found for all pairs of constraints c_k and c_l then `Seek_Support` returns TRUE and $lastGAC_{x_i,a,c_j}$ is updated. Otherwise `Seek_Support` returns FALSE and a new GAC-support is seeked in function `Revise`.

5 Experimental Results

We compared IωC and EIωC to maxRPWC on random problems. A more detailed comparison of all the consistencies presented in Section 3 on random and real problems is ongoing work. A random CSP is defined by the parameters $<n, d, k, p(e), q>$, where n is the number of variables, d the uniform domain size, k the uniform arity of the constraints, p the density of the problem (i.e. the ratio between the e constraints and the number of possible constraints involving k variables), and q the uniform looseness of the constraints (i.e. the ratio between the number of allowed tuples and d^k - the maximum number of tuples in a constraint).

Figure 5 (top) shows average CPU times for the three consistencies on 100 instances of class $<30, 20, 4, 0.001(27), q>$. We show both the time needed to enforce the consistencies and the time required to solve the instances with an algorithm that maintains maxRPWC during search after they have been preprocessed by each of the three consistencies (suffix s). The bottom figure shows the average percentages of instances proved to be inconsistent and values pruned by the three consistencies. The value of q is varied along the x-axis. Note that the class of Figure 5 gives rise to problems where maintaining maxRPWC is much more efficient than maintaining GAC.

IωC displays similar performance to maxRPWC in cpu times, deletion percentage, and inconsistency detection. This is not surprising given that this is a sparse class where all constraints are 4-ary. As a result, for any pair of intersecting constraints c_j, c_k there is seldom the case that some other constraint exists which only involves variables from $var(c_j) \cup var(c_k)$. EIωC detects many more inconsistent problems and deletes a higher percentage of values (for q>0.004) than IωC and maxRPWC, albeit with a higher cost. However, this preprocessing cost is negligible compared to the cost of search, and as a result, the search algorithm that uses EIωC preprocessing is more efficient than

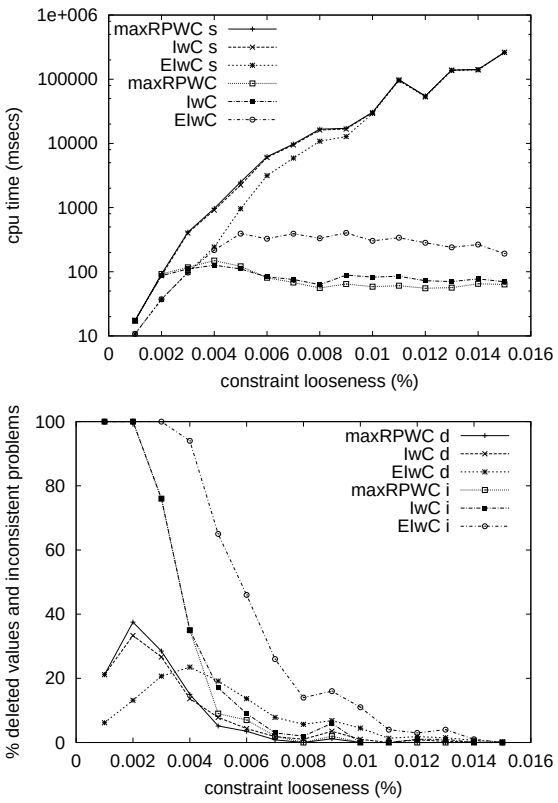

Figure 5. Cpu times (top) and percentages of deleted values (d suffix) and inconsistent problems detected (i suffix) (bottom).

the others up to the value of q where EIωC achieves a notable number of value deletions.

Table 1 gives results from problems of class $<50, 10, 4, 0.001(230), q>$ (1) and class $<100, 10, 4, 0.0001(392), q>$ (2). In each line we give the number of inconsistent instances detected, the average percentage of value deletions, and the cpu time (in msecs). The first three lines in the table refer to class 1 and correspond to parameter settings such that maxRPWC determines as inconsistent almost all, around half and only a few of the instances. Accordingly for class 2 in the next three lines. EIωC proves the inconsistency of all instances and in some cases it runs up to one order of magnitude faster than the other consistencies as it quickly wipes out some domain. IωC proves the inconsistency of many more instances than maxRPWC (especially in class 1) in competitive run times.

6 Conclusion

Although domain filtering consistencies tend to be more practical than consistencies that change the constraint relations and the constraint graph, only few such consistencies have been proposed for non-binary constraints. In this paper, we performed a detailed study of several strong inverse consistencies for non-binary constraints. All these consistencies are stronger than GAC, the consistency that is predominantly used by current constraint solvers, and most are stronger than maxRPWC. Preliminary experimental results demonstrated the potential of these strong consistencies. However, further empirical studies are necessary.

class	maxRPWC inc-%del-time	IωC inc-%del-time	EIωC inc-%del-time
1	96-28-583	99-26-275	100-7-48
1	45-15-561	90-27-441	100-10-90
1	8-3-295	53-17-470	100-13-231
2	95-24-888	95-23-813	100-10-70
2	48-14-1488	54-16-1251	100-13-302
2	9-3-412	18-5-535	100-15-674

Table 1. Average results over 100 instances on two classes of random problems.

References

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

[2] C. Bessière, K. Stergiou, and T. Walsh. Domain filtering Consistencies for Non-binary Constraints. *To appear in Artificial Intelligence*, 2007.

[3] R. Debruyne and C. Bessière. Domain Filtering Consistencies. *JAIR*, 14:205–230, 2001.

[4] E. Freuder. A Sufficient Condition for Backtrack-bounded Search. *JACM*, 32(4):755–761, 1985.

[5] E. Freuder and C. Elfe. Neighborhood Inverse Consistency Preprocessing. In *AAAI'96*, pages 202–208, 1996.

[6] M. Gyssens. On the complexity of join dependencies. *ACM Trans. Database Syst.*, 11(1):81–108, 1986.

[7] P. Janssen, P. Jégou, B. Nouguier, and M. Vilarem. A filtering process for general constraint satisfaction problems: Achieving pairwise consistency using an associated binary representation. In *Proceedings of IEEE Workshop on Tools for Artificial Intelligence*, pages 420–427, 1989.

[8] P. Jégou. On the Consistency of General Constraint Satisfaction Problems. In *AAAI'93*, pages 114–119, 1993.

[9] S. Nagarajan, S. Goodwin, and A. Sattar. Extending Dual Arc Consistency. *International Journal of Pattern Recognition and Artificial Intelligence*, 17(5):781–815, 2003.

[10] K. Stergiou and T. Walsh. Inverse Consistencies for Non-binary Constraints. In *ECAI-2006*, pages 153–157, 2006.

[11] P. van Beek and R. Dechter. On the Minimality and Global Consistency of Row-convex Constraint Networks. *JACM*, 42(3):543–561, 1995.

[12] G. Verfaillie, D. Martinez, and C. Bessière. A Generic Customizable Framework for Inverse Local Consistency. In *Proceedings of AAAI'99*, pages 169–174, 1999.