Making Bound Consistency as Effective as Arc Consistency
Christian Bessière, Thierry Petit, Bruno Zanuttini

To cite this version:

HAL Id: lirmm-00382609
https://hal-lirmm.ccsd.cnrs.fr/lirmm-00382609
Submitted on 14 Feb 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Making Bound Consistency as Effective as Arc Consistency*

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

Thierry Petit  
LINA-CNRS  
Ecole des Mines de Nantes, France  
thierry.petit@emn.fr

Bruno Zanuttini  
GREYC-CNRS  
U. Caen, France  
zanutti@info.unicaen.fr

Abstract

We study under what conditions bound consistency (BC) and arc consistency (AC), two forms of propagation used in constraint solvers, are equivalent to each other. We show that they prune exactly the same values when the propagated constraint is connected row convex / closed under median and its complement is row convex. This characterization is exact for binary constraints. Since row convexity depends on the order of the values in the domains, we give polynomial algorithms for computing orders under which BC and AC are equivalent, if any.

1 Introduction

Constraint solvers work by interleaving branching with constraint propagation. Arc consistency (AC) and bound consistency (BC) are the most common kinds of constraint propagation that solvers enforce. BC is used on a constraint when AC is too expensive. This happens when domains are too large, or when AC is too hard to enforce on the constraint. For instance, given $n$ variables and $d$ values per domain, the alldiff constraint can be made BC in $O(n \cdot \log(d))$ time ([Puget, 1998]) whereas $O(n \cdot 1.5d)$ time is required for AC ([Régin, 1994]). BC is polynomial on the Nvalue constraint whereas AC is NP-hard ([Bessiere et al., 2006]). The negative counterpart of BC is that BC generally prunes less values than AC.

Our main contribution is to characterize the unique maximal class of binary constraints on which AC and BC are equivalent however the initial domain is pruned. We give an easy test for membership of a constraint to that class.

Since BC is defined according to a total ordering of the domains, so is the case for the class of constraints on which BC is equivalent to AC. Our second contribution is to give a polynomial algorithm for reordering the domains so that a given constraint becomes a member of that class. Consider the constraint $\text{alldiff}(x_1, x_2, x_3)$, where $x_1, x_2 \in \{b, c\}$, $x_3 \in \{a, b, c, d\}$. BC does not remove any value because every bound belongs to a satisfying tuple. By reordering the domain of $x_3$ with $b < c < a < d$, BC removes $b$ and $c$ from $x_3$ and becomes equivalent to AC. A (cheaper) BC propagation can be applied to the constraint instead of AC. As a third contribution, we extend our results to non-binary constraints.

Our results give the theoretical basis for several directions of research such as incrementally detecting during search constraints for which BC becomes equivalent to AC, or designing efficient specialized reordering algorithms for global constraints. In the short run, our algorithms can be used in a preprocessing phase to detect on which constraints to use the BC propagator instead of the AC propagator. This will particularly pay off when a CSP defines a generic problem, solved repeatedly with the same kernel of constraints plus a few additional ones (e.g., a time tabling problem solved every month with different requirements). The preprocessing cost will be amortized on the several runs.

2 Background

A constraint network $\mathcal{N}$ consists of a sequence of variables $\mathcal{X} = (x_1, \ldots, x_n)$, a sequence of domains $D = \{D_1, \ldots, D_n\}$, where the domain $D_i$ is the finite set of at most $d$ values that variable $x_i$ can take, and a set $\mathcal{C}$ of $\mathcal{C}$ constraints that specify the allowed combinations of values for given subsets of variables. A relation $R$ of arity $k$ is any set of tuples of the form $(a_1, a_2, \ldots, a_k)$, where all $a_j$'s are values from the given universe $U$. A constraint of arity $k$ is a pair $\mathcal{C} = (X, R)$, where $X$ is a sequence of $k$ variables and $R$ is a relation of arity $k$. We use notation $D^X$ for the Cartesian product of domains $\Pi_{x_i \in X} D_i$, and $\mathcal{R}(D^X)$ for $R \cap D^X$, that is, the relation restricted to tuples over the $D_i$'s. The negation $\mathcal{C}^c$ of $\mathcal{C} = (X, R)$ is the constraint $(X, R^c)$, where $R^c = U^X \setminus R$ is the complement of $R$.

An assignment $t$ to a set of variables $X$ is a function from $X$ to the universe of values $U$. $t[x]$ is the value of $x$ in $t$. An assignment $t$ to a sequence $X = (x_1, \ldots, x_k)$ of variables, or to a superset of $X$, satisfies $\mathcal{C} = (X, R)$ iff $(t[x_1], \ldots, t[x_k]) \in R$. $t[x_i/b]$ is the assignment obtained from $t$ when replacing $t[x_i]$ by $b$. A solution of $\mathcal{N} = (X, D, \mathcal{C})$ is an assignment $t$ to $X$ such that $t[x_i] \in D_i$ for all $x_i \in X$, and $t$ satisfies all the constraints in $\mathcal{C}$.

Orderings An ordered signature is a triple $\Sigma = (X, D, <)$, where $X = (x_1, \ldots, x_k)$ is a sequence of variables, $D = (D_1, \ldots, D_k)$ is a sequence of domains for them, and $<$ is a sequence $(<_1, \ldots, <_k)$, where $<_i$ is a total order on $D_i$. We

*Supported by the project CANAR (ANR-06-BLAN-0383-02).
consider that $\Sigma$ is also an ordered signature for any $Y \subseteq X$. $\min_c D_i$ (resp. $\max_c D_i$) is the minimum (resp. maximum) element in $D_i$ according to $\leq_i$. Given $a, b, c \in D_i$, $\med_{\leq_i}(a, b, c)$ is the median value of $a, b, c$ according to $\leq_i$. When the ordering is clear, we will write $\min D_i$, $\max D_i$, $\med(a, b, c)$. We write $[a..c]$ for the set of values with $a \leq b \leq c$, $\overline{S}$ for $[\min S..\max S]$, and $a - 1 (a + 1)$ for the immediate predecessor (successor) of value $a$. We write $D'_t < D_t$ as a shorthand for $\forall b \in D_t, \forall b' \in D'_t, b' < b$ and we write $D' \subseteq D$ for all $D' \subseteq D_t$. For instance, given $a < b < c$ we have $\min(c, a) = a$, $\med(a, b, c) = b$, $\med(a, c, c) = c$, $\{a, b, c\} = \{a, b, c\}$, $c - 1 = b$, $\{a, b\} < \{c\}$.

**Consistencies** Constraint solvers maintain a local consistency during search. The most usual ones are defined below.

**Definition 1 (local consistencies)** Let $\Sigma = (X, D, <)$ be an ordered signature and $C = (X, R)$ be a constraint. A support for $b_i \in D_i$ is a tuple $t \in R[D^X]$ such that $t[x_i] = b_i$. A bound support for $b_i \in D_i$ is a tuple $t \in R[I_{\leq_c \subseteq X}D^X]$ such that $t[x_i] = b_i$. The domain $D_i$ is AC (resp. RC, AC) on $\Sigma$ if for all $b_i \in D_i$ and $D_i$ have a bound support. The constraint $C$ is BC (resp. RC, AC) if for all $x_i \in X$, $D_i$ is AC (resp. RC, AC) on $\Sigma$.

Our BC is the most common version of bound consistency, called BC(Z) in [Bessiere, 2006]. Slightly abusing words, for a binary constraint $((x_1, x_2), R)$ we say that value $a_2$ is a (bound) support for $a_1$ instead of the tuple $(a_1, a_2)$.

Let $(X, D, C)$ be a constraint network, $(X, D, <)$ an ordered signature, and $\Phi$ a local consistency. $\Phi(D, C)$ is the closure of $D$ for $\Phi$ on $C$, that is, the sequence of domains obtained from $D$ once for every $i$ we have removed every value $b_i \in D_i$ that is not $\Phi$-consistent on a constraint in $C$. $AC(D, C)$ is also denoted by $D^\ast$.

Two local consistencies may behave the same. Our definition of equivalence requires stability under domain reductions (due to instantiations and propagation of other constraints).

**Definition 2 (equivalence of local consistencies)** Given an ordered signature $\Sigma = (X, D, <)$ and a constraint $C$, two local consistencies $\Phi_1$ and $\Phi_2$ are called equivalent on $C$ for $\Sigma$ if for any domain $D' \subseteq D$, $\Phi_1(D', C) = \Phi_2(D', C)$.

**Classes of constraints** Monotone constraints have been introduced by Montanari [Montanari, 1974] in the binary case. We give a definition for any arity.

**Definition 3 (increasing, monotone)** Let $\Sigma = (X, D, <)$ be an ordered signature and $C = (X, R)$ a constraint. A variable $x_i \in X$ is said to be increasing (resp. decreasing) for $\Sigma$ on $C$ if for any tuple $t \in R$ and any value $a \in D_i$ such that $a > t[x_i]$ (resp. $a < t[x_i]$), the tuple $t[x_i/a]$ is in $R$. A constraint in which every variable is decreasing or increasing is said to be monotone.

**Definition 4 (row convex [van Beek and Dechter, 1995])** Let $\Sigma = (X, D, <)$ be an ordered signature. A constraint $C = (X, R)$ is row convex for $\Sigma$ if for every $x_i \in X$ and any instantiation $t_{-i}$ in $D^X \setminus \{x_i\}$, the set of values in $D_i$ that extend $t_{-i}$ to a tuple in $R[D^X]$ is an interval in $D_i$.

*Connected row convex constraints* have been introduced in [Deville et al., 1999] only for binary constraints, but [Jeavons et al., 1998] defined med-closed constraints and showed that the constraints that are binary and med-closed are exactly the connected row convex constraints [Jeavons et al., 1998, example 4.7]. We thus give the most general definition.

**Definition 5 (med-closed constraint)** Let $\Sigma = (X, D, <)$ be an ordered signature. $C = (X, R)$ is med-closed for $\Sigma$ if for any three tuples $t, t', t''$ in $R[D^X]$, the tuple $t_m$ defined by $\forall x \in X, t_m[x] = \med_{\leq_i}(t[x], t'[x], t''[x])$ is in $R$. Example 1 Let $D_1 = D_2 = D_3 = \{0, 1, 2\}$ with the natural order on each domain. $C = (\{(x_1, x_2, x_3), (0, 0, 101, 222)\})$ is not med-closed because the median of 010, 101, 222 is the forgotten tuple 111. Reordering $D_2$ such that $1 < 2 < 0$, we obtain a med-closed constraint: for any three tuples the median tuple is 000 or 101, which satisfy $C$. \(\blacktriangledown\)

3 Equivalent Consistencies in the Binary Case

Enforcing BC on a constraint generally prunes less values than enforcing AC. However, in some particular cases, BC and AC can be equivalent. This is the case, e.g., for the constraint $x < y$, whatever the domains of $x$ and $y$.

In this section, we analyze the properties that make BC and RC equivalent on binary constraints. We do the same for RC and AC, and then we derive a necessary and sufficient condition for having BC equivalent to AC on a binary constraint.

Our characterization is for convex initial domains (of the form $\overline{D}$), that is, it characterizes the cases when BC is equivalent to AC on every subdomain $D' \subseteq \overline{D}$. We will see in Section 5 why this is not a restriction. We also restrict our analysis to domains $\overline{D}^\ast$, that is, the subdomain of an initial (convex) domain $\overline{D}$ where all values have a support. This is a reasonable restriction because a single preprocessing phase of AC is enough. It discards dummy values that prevent from finding any nice characterization.

**Lemma 1** Let $X = (x_1, x_2), \Sigma = (X, D, <), C = (X, R)$. BC and RC are equivalent on $(X, (\overline{D})^\ast)$ for $C$ iff $C$ is connected row convex for $\Sigma^* = (\overline{D}, <)$.

**Proof.** ($\Rightarrow$) Assume $C$ is connected row convex for $\Sigma^*$ and let $D'_1 \subseteq (\overline{D})^\ast$. By definition, if $D' = RC$ is also BC for the same ordering. Conversely, assume $D' = BC$ for $C$. Let $a = \min D'_1$ and $c = \max x D'_1$. By definition of BC, $a$ and $c$ have bound supports $t_a$ and $t_c$ wrt $D'$ on $C$. Take any $b \in D'_1$. $b$ has a support $t_b$ in $\overline{D}^\ast$ because $b \in D'_1 \subseteq (\overline{D})^\ast$. Let $t_m$ be the tuple $\med(t_a, t_b, t_c)$, since by construction $t_a, t_b, t_c$ are in $(\overline{D})^\ast$ and satisfy $C$, and since $C$ is connected row convex for $\Sigma^*$, $t_m$ is a satisfying tuple for $C$. By definition of median, $t_m[x] \in \{t_a[x], t_b[x], t_c[x]\} \subseteq D'_1 \subseteq (\overline{D})^\ast$. In addition, $t_m[x] = b$ because $a < b < c$. Therefore, $t_m$ is a bound support for $b$ in $D'$, and $b$ is RC. The same reasoning applies for values in $D'_2$. As a result, $D'$ is RC. ($\Leftarrow$) Assume now that $C$ is not connected row convex for $\Sigma^*$. We show that there exists $D'_1 \subseteq (\overline{D})^\ast$ that is BC but not RC for $C$. Since $C$ is not connected row convex, there exist $a_1, b_1, b_2, c_1 \in (\overline{D})^\ast$ with $a_1, b_1, b_2, c_1 \in R$ but $t_m = \med(a_1, b_1, b_2, c_1) \notin R$. Without loss of generality,
assume \( a_1 \leq b_1 \leq c_1 \). We have \( t_m \neq b_1 \) because \( b_1, b_2 \in R \). Hence, \( b_2 \neq t_m[x_2] = \text{med}(a_2, b_2, c_2) \) because \( t_m[x_1] = b_1 \). Assume \( b_2 < a_2 < c_2 \) (the other cases are similar).

If \( b_1 \) has no support in \([a_2, c_2] \cap (D')^*_1\), consider the domain \( D' = ([a_1, c_1] \cap (D')^*_1, \{a_2, c_2\}) \). Then, \( a_1, a_2, c_1, c_2 \) have a bound support in \( D' \), and so \( D' \) is BC. But \( b_1 \in D' \) has no bound support in \( D' \). Conversely, if \( b_1 \) has a support \( b_1 b_2^* \) in \([a_2, c_2] \cap (D')^*_2\), consider the domain \( D' = ([b_1, b_2^*] \cap (D')^*_2) \). By construction, \( b_2 < a_2 \leq b_2^* \leq c_2 \) and thus \( b_1 b_2^* = t_m \notin R \). So, \( b_1, b_2, b_2^* \) have a bound support in \( D' \), and so \( D' \) is BC, but \( a_2 \) has no bound support in \( D' \). \[ \square \]

**Lemma 2** Let \( X = (x_1, x_2) \), \( \Sigma = (X, D, <) \), and \( C = (X, R) \). \( RC \) and \( AC \) are equivalent on \((X, (\overline{D})^*)\) for \( C \) iff \( C^- \) is row convex for \( \Sigma^* = (X, (\overline{D})^*, <) \).

**Proof.** \((\Leftarrow)\) Suppose \( C^- \) is row convex for \( \Sigma^* \) and let \( D' \subseteq (\overline{D})^* \). By definition, if \( D' \) is AC, it is also RC. Conversely, assume \( D' \) is RC for \( C \). Let \( b_1 \in D_1^i \) and \( t_m \in (D')^* \). \( t_m \) is a bound support for \( b_1 \) in \( D' \). We write \( a_i = \min D_1^i, b_i = t_m[a_i], c_i = \max D_1^i \). We have \( t_m \in R \), so \( t_m \notin R \). Hence, \( b_1, b_2 \notin R \) or \( b_1, b_2 \notin R \) because \( C^- \) is row convex for \( \Sigma^* \) and \( a_2 \leq b_2 \leq c_2 \). Thus, at least one tuple among \( b_1 b_2^* \) and \( b_1 b_2 \) is in \( R \) and \( b_1 \) has a support in \( D' \). Therefore, \( D' \) is AC.

\((\Rightarrow)\) Assume now that \( C^- \) is not row convex for \( \Sigma^* \). Without loss of generality, let \( b_1 \in (D')^*_1, b_2 \in (D')^*_2 \) with \( a_2 < b_2 < c_2 \) such that \( b_1, b_2, b_2^* \in R^* \) and \( b_1 b_2^* \notin R^* \). (Since, e.g., the unary constraint \( \ell_1 \)) is, \( \ell_1 \) and \( \ell_2 \), where \( \ell_1 \) is either \( x_1 \geq a_1 \) or \( x_1 \leq a_1 \) for some \( a_1 \in D_1^i \). Moreover, since all values in \( D^* \) have support, there is no nontrivial unary constraint in \( C \) (since, e.g., the unary constraint \( x_1 \geq 2 \) eliminates any support for \( 1 \)).

By symmetry, let \( C = (x_1 \geq a_1 \vee x_2 \geq a_2) \) be a constraint in \( C \) (case \( i \)); case \( ii \) is symmetric with a clause \((x_1 \geq a_1 \vee x_2 \leq a_2) \). We claim that any other constraint in \( C \) is of the form \((x_1 \geq a_1 \vee x_2 \geq a_2) \) or \((x_1 \leq a_1 \vee x_2 \leq a_2) \). Indeed, if there was a constraint, say, \((x_1 \geq a_1 \vee x_2 \leq a_2) \), contradicting the definition of \( D^* \), its non-supports would form a non-convex interval \( D_2^i \backslash \{a_2, a_2\} \) of \( D_2^i \), contradicting the row convexity of \( C^- \).

So \( C \) contains only \((\leq, \leq)\) and \((\geq, \geq)\)-constraints. Now we claim that every two constraints \((x_1 \geq a_1 \vee x_2 \geq a_2) \), \((x_1 \leq a_1 \vee x_2 \leq a_2) \) are such that \( a_1 - 1 < a_1^* + 1 \) and \( a_2 - 1 < a_2^* + 1 \). Indeed, if \( a_1 - 1 < a_1^* + 1 \), then either value \( a_1^* + 1 \in D_1^i \)
has no support (case \(a_2 > a_2\)), or its non-supports form a non-convex interval of \(D_2\), a contradiction in each case.

To sum up, we have that values ruled out by \(\geq\)-literals are all less than values ruled out by \(\leq\)-literals.

Let \(D_i^k = \{b_1 \in D_i | \exists a_1, a_2, b_1 > a_1, (x_1 \leq a_1 \land x_2 \leq a_2) \in C\}\), that is, values for \(D_i^k\) ruled out by \(\leq\)-literals, and similarly for \(D_i, D_i^2, D_i^3\). Moreover, let \(D_i^k = \{a_1 \in D_i | \forall a_2 \in D_2, a_1a_2 \in R\}\), and similarly for \(D_i^k\). As we can see, these collect all the remaining values. By construction and by the second claim above, we have that \((D_i, D_i^2, D_i^3)\) is a partition of \(D_i\) and \(D_i^1 < D_i^2 < D_i^3\).

Finally, by construction, only \(\leq\)-literals constrain values in \(D_i^1\), since \(x_1\) is decreasing in \((X, R[D_i^1 \times D_2])\), and similarly for the other cases. Finally, by construction again all other restrictions of \(R\) are satisfied by all tuples.

\[\square\]

5 Binary Constraints Renamability

Membership to the Crow+row \(^{-}\) class of constraints is defined according to an ordered signature. In this section, we are interested in Crow+row \(^{-}\)-renamability, that is, finding whether there exists an ordering on the domains of the variables of a constraint that makes the constraint Crow+row \(^{-}\).

We first give an easy result about the corresponding problem for monotone constraints. Observe that monotone binary constraints are a special case of Crow+row \(^{-}\) constraints.

**Proposition 2** Let \(X = (x_1, x_2), C = (X, R)\), and a couple of domains \(D\) for \(X\). One can compute an ordered signature \(\Sigma = (X, D, <)\) such that \(C\) is monotone for \(\Sigma\), or assert that there is none, in time \(O(d^3)\).

**Proof.** (Sketch) Without loss of generality, we look for a signature according to which every variable is increasing. For each variable \(x_i\), we build a directed graph \(G_i\) as follows. Consider \(x_1\) (the case of \(x_2\) is analogous). The vertices of \(G_1\) are the values in \(D_1\), and there is an edge \((b_1, a_1)\) if and only if there is \((a_1, a_2) \in R\) such that \((b_1, a_2)\) is not in \(R\). Clearly, the topological orders on \(G_i\)'s, if any, are exactly those for which \(C\) is increasing. Time complexity is straightforward. \(\square\)

We give an efficient algorithm for reordering domains so that a constraint becomes Crow+row \(^{-}\), using Proposition 1. We first try to find a Crow+row \(^{-}\)-partition of the domains. We use an undirected graph to do so. If the partition exists, we try to reorder \(D_i^1\) and \(D_i^2\) so as to satisfy the in/decreasingness constraints in the characterization of Proposition 1. By symmetry, we look for reorderings such that \(R[D_i^1 \times D_2]\) is increasing, and \(R[D_i^1 \times D_2^1]\) is decreasing, that is, letting zeros in the upper left and lower right corners of the matrix.

**Definition 7** (Crow+row \(^{-}\)-graph) Let \(X = (x_1, x_2), C = (X, R)\), and a couple \(D\) of domains for \(X\). The Crow+row \(^{-}\)-graph of \(C\) on \(D\), written \(G(C)\), is the undirected graph \((V, E)\) with \(V = (D_1 \times D_2) \setminus R\) and for all \(a_1a_2, b_1b_2 \in R\), \(a_1b_2, b_1a_2 \in E\) if and only if \(a_1 = b_1\) or \(a_2 = b_2\).

Intuitively, the Crow+row \(^{-}\)-graph is the graph of zeros in the matrix representation, where two zeros are connected if they are on the same line or column. So the following lemma is a direct consequence of Proposition 1 (with the intuition coming from the matrix representation).

**Lemma 3** Let \(X = (x_1, x_2), \Sigma = (X, D, <)\), and \(C = (X, R)\). The following are equivalent:

1. there is a Crow+row \(^{-}\) - partition \((D_i^1, D_i^2, D_i^3)\) of \(D\) such that \(R[D_i^1 \times D_2^1] = D_i^1 \times D_2, R[D_i^2 \times D_2^1] = D_i^2 \times D_2, R[D_i^1 \times D_2^1]\) is increasing, and \(R[D_i^1 \times D_2^2]\) is decreasing for \(\Sigma^* = (X, D^*, <)\),

2. the graph \(G(C)\) has exactly two connected components \(C^-, C^+\) (possibly empty), so that \(D_i^1 = \{a_1 | a_1a_2 \in C^-\}\) and similarly for \(D_i^2, D_i^3\), \(R[D_i^1 \times D_2^2]\) is increasing, and \(R[D_i^1 \times D_2^2]\) is decreasing for \(\Sigma^*\).

Before we state the results about computing renamings, we show that, though our characterization concerns only the convex hull of domains, we can search reorderings of only the values in the domain (ignoring values in \(U \setminus D\)). The proof is straightforward since row and connected row convexity are defined according to actual domains (not ranges).

**Proposition 3** Let \(X = (x_1, x_2), C = (X, R)\), and domains \(D\) for \(X\). There is an ordered signature \(\Sigma_U = (X, (\overline{D})^*, <_U)\), where \(<_U\) is over the universe of values \(U\), such that \(C\) is Crow+row \(^{-}\) for \(\Sigma_U\), if there is such a signature \(\Sigma_U = (X, (\overline{D})^*, <_U)\) for which \((D_i)^*\) is convex in \(U\) wrt \(<_U\), \(i = 1, 2\). Moreover, the ordering over \(U \setminus (D_i)^*\) does not matter.

Hence, in the remainder of the paper we assume that only the values in the initial domain \(D^*\) are reordered, instead of all the values in \((\overline{D})^*\). Equivalently, we assume \(U = D^*\).

**Proposition 4** Let \(X = (x_1, x_2), C = (X, R)\), and a couple of domains \(D\) for \(X\). One can compute an ordered signature \(\Sigma^* = (X, D^*, <)\) such that \(C\) is Crow+row \(^{-}\) for \(\Sigma^*\) or assert that there is none in time \(O(d^3)\).

**Proof.** The algorithm builds the graph \(G(C)\) in time and space \(O(d^3)\) because there are \(O(d^3)\) vertices (one for each pair \((a_1, a_2)\) of values) and each vertex \((a_1, a_2)\) can be connected to at most \(2d\) other vertices (those of the form \((a_1, b_2)\) or \((b_1, a_2)\)). Then it computes the connected components of \(G(C)\) in time linear in \(G(C)\), i.e., \(O(d^3)\). If there are at least three components, then it stops on failure (Lemma 3).

Otherwise, it computes \(D_i^1\) and \(D_i^2\) for \(i = 1, 2\) by projecting the components, in time \(O(d^2)\). Finally, it looks for ordered signatures \(\Sigma^- = (X, (D_i^1, D_i^2), <^-)\) and \(\Sigma^+ = (X, (D_i^1, D_i^2), <^+)\) so that \(R[D_i^1 \times D_2^1]\) is increasing for \(\Sigma^-\) and \(R[D_i^1 \times D_2^2]\) is decreasing for \(\Sigma^+\), in time \(O(d^3)\) using Proposition 2. If there are none, then again it stops on failure (Lemma 3). Otherwise, it computes an ordered signature compatible with both \(\Sigma^-\) and \(\Sigma^+\), which can be done by stacking the ordered subdomains over each other, in time \(O(d^2)\). The overall time complexity is \(O(d^3)\). \(\square\)

6 Renamability of Sets of Binary Constraints

In general, renaming several constraints that have variables in common can be done by storing for each variable on each
constraint a permutation function that links the original ordering with the ordering that makes that constraint Crow-row\(^-\). However, in Section 7, when decomposing a non-binary constraint into binary ones, the variables inside the non-binary constraint must be reordered the same way for each binary constraint, making all of them Crow-row\(^-\) simultaneously. Call a set of constraints \(\mathcal{C}\) on \((X, D)\) Crow-row\(^-\)renamable if there exists \(\Sigma = ((X, D, <)\) such that \(\forall C \in \mathcal{C}, C\) is Crow-row\(^-\) for \(\Sigma\). We show that deciding Crow-row\(^-\)renamability of a set of binary constraints is tractable.\(^1\) Lemma 4 says that, up to symmetries, there is essentially one signature making a constraint Crow-row\(^-\) (if any).

**Lemma 4** Let \(X = (x_1, x_2)\), \(C = (X, R)\), and \(D\) a couple of domains for \(X\). If \(C\) is Crow-row\(^-\)-renamable, then there are partitions \((D_1^1, \ldots, D_k^i)\) of \(D_i\), \(i = 1, 2\), such that the ordered signatures \((X, D_i, (<, <))\) for which \(C\) is Crow-row\(^-\) exactly those satisfying the partial order \(D_i^1 <; \cdots <; D_i^{k_i}\) or \(D_i^1 >; \cdots >; D_i^{k_i}\) for each \(i = 1, 2\).

**Proof.** The symmetry between \(<\) and \(>\) is obvious. The partitions are built by grouping together values substitutable to each other (i.e., equal rows or columns in the matrix representation). As is clear on the matrix representation, any other partial order violates the characterization of Proposition 1. \(\square\)

The algorithm which we propose for renaming sets of constraints proceeds inductively using the algorithm for an isolated constraint. Given \(\{C_1, \ldots, C_n\}\), it looks for the partial orders \(\Sigma\) which \(C_1\) is Crow-row\(^-\). If none it stops, and otherwise it looks for such orders for \(C_2\). Again, if none it stops, and otherwise it looks for orders compatible with both. If none it stops, and otherwise it goes on with \(\Sigma\) (C3, etc.).

**Example 3** Given \(D_1 = \{a_1, b_1, a_1\}, D_2 = \{a_2, \ldots, a_2\}, D_3 = \{a_3, b_3, c_3\}\) and the constraints on \(x_1, x_2\) and \(x_2, x_3\):

\[
\begin{align*}
  a_1 & : \begin{cases} 0 & 1 & 1 & 1 \\ b_1 & : \begin{cases} 1 & 1 & 1 & 0 \\ c_1 & : \begin{cases} 1 & 1 & 1 & 0 
\end{cases} 
\end{cases} 
\end{cases}
\end{align*}
\]

By Proposition 1, the first one is Crow-row\(^-\) wrt exactly those orders \(<1, <2\) such that \(\{a_1\} < \{b_1, c_1\}\) or \(\{a_1\} > \{b_1, c_1\}\), and \(\{a_2, b_2\} < \{c_2\} < \{d_2, e_2\}\) or \(\{a_2, b_2\} > \{c_2\} > \{d_2, e_2\}\). Now the second one is Crow-row\(^-\) iff \(\{a_3\} < \{b_3\} < \{c_3\}\) or \(\{a_3\} > \{b_3\} > \{c_3\}\), and \(\{a_2\} < \{b_2, c_2\} < \{d_2, e_2\}\) or \(\{a_2\} > \{b_2, c_2\} > \{d_2, e_2\}\). The set of both constraints is Crow-row\(^-\) wrt \(\rho_1, \rho_2, \rho_3\) if \(\{a_1\} \rho_1 \{b_1, c_1\}, \{a_3\} \rho_3 \{b_3\}, \rho_3 \{c_3\}\), and \(a_2 \rho_2 \{b_2, c_2\} \rho_2 \{d_2, e_2\}, \rho_1, \rho_2, \rho_3 \in (<1,2)\).

**Proposition 5** Deciding whether a set of binary constraints \(\mathcal{N}\) is Crow-row\(^-\)renamable can be done in time \(O(d^3)\).

**Proof.** We show by induction that the algorithm sketched above is correct. The base case with no constraints is obvious, with the empty order for all \(x_i\). Now assume that \((X, D, \{C_1, \ldots, C_k\}\) is Crow-row\(^-\) wrt exactly the orders \(<1, such that \(D_i^1 < D_i^2 < \cdots < D_i^{k_i}\) or \(D_i^1 > D_i^2 > \cdots > D_i^{k_i}\). Assume w.l.o.g., using Lemma 4, that \(C_{k+1} = ((x_p, x_q), R)\) is Crow-row\(^-\) if \(E_{i_1}^1 < \cdots < E_{i_q}^1 \) or \(E_{i_1}^1 > \cdots > E_{i_q}^1 \). If there are orders extending both \(<\)'s and \(<\)'s, then clearly \(\Sigma_1, \ldots, \Sigma_{k+1}\) is Crow-row\(^-\) for \(\Sigma\). Otherwise, by construction there is no order making both \(\Sigma_1, \ldots, \Sigma_k\) and \(\Sigma_{k+1}\) Crow-row\(^-\).

Now to the time complexity. Renaming each constraint requires \(O(d^3)\) (see Proposition 4). For each variable in the new constraint, two partial orders must be extended into a common one, which can be done by merging them in time \(O(d)\). Now there are \(e\) inductive steps, which concludes. \(\square\)

### 7 Non-binary Crow-row\(^-\) Constraints

So far we have considered only binary constraints. Recall that med-closed constraints give a generalization of connected row convex constraints to an arbitrary arity. Though we lose equivalence, Theorem 2 gives an interesting parallel to Theorem 1. The proof is a straightforward generalization of the (\(\leq\)) direction in Lemmas 1 and 2.

**Theorem 2 (BC vs AC, n-ary)** Let \(\Sigma = (X, D, <)\) and \(C = (X, R)\). If \(C\) is med-closed and \(C\) is row convex for \(\Sigma' = (X, (D)^*), <\), then \(BC\) and \(AC\) are equivalent on \((X, (D)^*)\) for \(C\).

The reverse does not hold, as can be seen with the ternary relation \(R = \{1, 2, 3\}\), which is not med-closed but for which \(BC\) and \(AC\) are equivalent.

Call med-row\(^-\) for a med-closed constraint with row convex complement. Proposition 6 shows that such a constraint can be recognized via its projections over pairs of variables, that are all Crow-row\(^-\). Proposition 7 shows the existence of a polynomial algorithm for med-row\(^-\)renamability for constraints given in extension. The question remains open for constraints defined by an arbitrary predicate.

A constraint is binary decomposable if it is equivalent to the conjunction of its projections onto all pairs of variables.

**Proposition 6** If \(C\) is (n-ary) med-row\(^-\) wrt some ordered signature \(\Sigma\), then \(C\) is binary decomposable, and each projection is Crow-row\(^-\) wrt \(\Sigma\).

**Proof.** Because \(C\) is med-closed \(\Sigma\), it is binary decomposable [Gil et al., 2008]. By definition of med-closure, each projection is med-closed / connected row convex wrt \(\Sigma\).

Now assume towards a contradiction that the projection \(((x_i, x_j), R_{i,j})\) of \(C\) has a non row convex complement. There is a value \(a_i\) for \(x_i\) and three values \(a_j, b_j < c_j\) for \(x_j\) (according to \(\Sigma\)) such that \(a_i a_j, a_i c_j \not\in R_{i,j} \subseteq (R_{i,j})\), but \(a_i b_j \in R_{i,j}\). By definition of projection there is a support for \(a_i b_j\) in \(R\), that is, a tuple \(a_i b_j, t_i, t_j \in R\) but no support for \(a_i a_j, a_i c_j\). So, \(t_i a_i t_j, t_i c_j \not\in R\). Thus the instantiation \(t_i t_j t_i t_j\) of all variables but \(x_i\) has a non convex support in \(R\). This violates the assumption that \(R\) is row convex. \(\square\)

Note that the binary decomposition of \(R\) may be Crow-row\(^-\) and \(R\) be non med-row\(^-\). This is the case, for instance, for the join of the two constraints in Example 3 with the tuple \(b_1 b_2 b_3\) removed.
Proposition 7. Given an n-ary constraint $C = (X, R)$ given in extension and a sequence of domains $D$ for $X$, one can decide in polynomial time whether $C$ is med+row$^*$-renamable.

Proof. First project $R$ onto every pair of variables, yielding a set of binary constraints $C$, and decide whether $(X, D, C)$ is Crow+row$^*$-renamable with Proposition 5. If it is not, then by Proposition 6 $R$ is not med+row$^*$-renamable.

Otherwise, by the construction in Section 6, for all $i$ we get a partial order $<_i$ on the domain of $x_i$, so that total orders according to which $C$ is Crow+row$^*$ are exactly those extending $<_i$ or $>_i$ for all $i$. Given a set of (arbitrary) such total orders $<_1, \ldots, <_n$, one for each $x_i$, we claim that $R$ is med+row$^*$-renamable iff it is med+row$^*$ for $<$. The “if” direction is obvious, so assume that $R$ is med+row$^*$-renamable wrt some collection of total orders $<_i$. By Proposition 6 it is binary decomposable, so logically equivalent to $C$. Especially, any two values substitutable to each other in $C$ are so in $R$ as well. So $R$ is med+row$^*$ wrt any order obtained from $<_i$ by permuting two values substitutable to each other in $C$ (as in the proof of Lemma 4). Moreover, by symmetry of med-closure and row convexity, it is also med+row$^*$ wrt any order obtained from $<_i$ by reversing the order over any domain. Since by Proposition 6 and the construction, both $< \iff$ and $<_i^\prime$ extend the partial order $<_i$ or $>_i$ for each $i$, $<_i^\prime$ can be obtained from $<_i$ by permuting substitutable values and reversing orders, so by symmetry again $R$ is also med+row$^*$ wrt $<_i$, which ends to show the claim.

To conclude, one can decide whether $R$ is med+row$^*$-renamable by considering only one (arbitrary) total extension of the $<_i$’s. Since $R$ is given in extension, this can be done in polynomial time by testing the definitions of med-closure and row-convexity directly on the tuples in $R$. □

8 Related Work

Several classes of constraints have been identified for which the CSP is tractable, e.g., monotone constraints [Montanari, 1974; Woeginger, 2002] and row convex constraints [van Beek and Dechter, 1995; Deville et al., 1999]. Domain reordering techniques have been proposed to make a constraint member of the class [van Beek and Dechter, 1995; Green and Cohen, 2008]. However, it is seldom the case that a CSP only contains constraints of one class or allows a reordering leading all constraints to this class. Therefore, those theoretical results are not used in constraint solvers.

BC and AC have been proved to be equivalent on linear inequalities [Zhang and Yap, 2000] or on some arithmetic constraints, given some conditions on the type of domain reductions [Schulte and Stuckey, 2005]. In comparison, our work gives an exact characterization of the unique maximal class of binary constraints on which AC and BC are equivalent whatever the domain reductions that will appear during search. We insist on the importance of stability of equivalence under domain reductions. It allows a solver to use a BC propagator on that constraint without any risk to hinder AC propagation deeper in the search tree when domains have been modified.

9 Summary and Perspectives

We have characterized the constraints for which bound consistency prunes as many values as arc consistency, and we have proposed efficient algorithms for recognizing such constraints or sets of constraints, possibly by reordering the domains. Since bound consistency is usually cheaper to enforce than arc consistency, recognizing such constraints can save some pruning effort during search. The tractability of renaming non-binary constraints in extension being unknown, a promising direction for future research is to design efficient specialized algorithms for renaming global constraints so that a cheap BC propagator can be applied for enforcing AC. This can be seen as a new algorithmic perspective for designing efficient AC propagators.

References


