# Permutation Problems and Channelling Constraints

**Toby Walsh**
Department of Computer Science
University of York
York
England
tw@cs.york.ac.uk

## Abstract

We perform an extensive study of several different models of permutation problems proposed by Smith in [Smith, 2000]. We first define a measure of constraint tightness parameterized by the level of local consistency being enforced. We then compare the constraint tightness in these different models with respect to a large number of local consistency properties including arc-consistency, (restricted) path-consistency, path inverse consistency, singleton arc-consistency and bounds consistency. We also compare the constraint tightness in SAT encodings of these permutation problems. These results will aid users of constraints to choose a model for a permutation problem, and a local consistency property to enforce upon it. They also illustrate a methodology, as well as a measure of constraint tightness, that can be used to compare different constraint models.

## 1 Introduction

In modeling a constraint satisfaction problem, we often have a choice as to what to make the variables, and what to make the values. For example, in an exam timetabling problem, the variables could be the exams, and the values could be the times. Alternatively, the variables could be the times, and the values could be the exams. This choice is especially difficult in permutation problems. In an empirical study of Langford's problem, Smith proposes a number of different models of permutation problems [Smith, 2000] which we study in detail here. In a permutation problem, we have as many values as variables, and each variable takes an unique value. We can therefore easily swap variables for values. Many assignment, scheduling and routing problems are permutation problems. For example, sports tournament scheduling can be modeled as finding a permutation of the games to fit into the available slots, whilst the traveling saleperson problem can be modeled as finding a permutation of the cities.

## 2 Formal background

A *constraint satisfaction problem* (CSP) consists of a set of variables, each with a finite domain of values, and a set of constraints. Each constraint is a relation defining the allowed values for a given subset of variables. A solution to a CSP is an assignment of values to variables that is consistent with all constraints. Many lesser levels of consistency have been defined for binary constraints (see [Debruyne and Bessière, 1997] for references). A problem is $(i, j)$-*consistent* iff it has non-empty domains and any consistent instantiation of $i$ variables can be consistently extended to $j$ additional variables [Freuder, 1985]. A problem is *arc-consistent* (AC) iff it is $(1, 1)$-consistent. A problem is *path-consistent* (PC) iff it is $(2, 1)$-consistent. A problem is *strong path-consistent* (ACPC) iff it is AC and PC. A problem is *path inverse consistent* (PIC) iff it is $(1, 2)$-consistent. A problem is *restricted path-consistent* (RPC) iff it is AC and if a value assigned to a variable is consistent with just one value for an adjoining variable then for any other variable there exists a compatible value. A problem is *singleton arc-consistent* (SAC) iff it has non-empty domains and for any instantiation of a variable, the resulting subproblem can be made AC. For non-binary constraints, there has been much less work on different levels of local consistency. One exception is generalized arc-consistent. A problem is *generalized arc-consistent* (GAC) iff for any value for a variable in a (non-binary) constraint, there exist compatible values for all the other variables in the constraint [Mohr and Masini, 1988]. For ordered domains, a problem is *bounds consistent* (BC) iff it has non-empty domains and the minimum and maximum values for any variable in a (binary or non-binary) constraint can be extended to satisfy the constraint [Hentenryck *et al.*, 1998].

Following [Debruyne and Bessière, 1997], we say that a local consistency property $A$ is as strong as a local consistency property $B$ (written $A \leadsto B$) iff in any problem in which $A$ hold then $B$ holds, $A$ is stronger than $B$ (written $A \rightarrow B$) iff $A \leadsto B$ but not $B \leadsto A$, $A$ is incomparable with $B$ (written $A \otimes B$) iff neither $A \leadsto B$ nor $B \leadsto A$, and $A$ is equivalent to $B$ (written $A \leftrightarrow B$) iff both $A \leadsto B$ and $B \leadsto A$. The following summarizes results from [Debruyne and Bessière, 1997] and elsewhere: ACPC $\rightarrow$ SAC $\rightarrow$ PIC $\rightarrow$ RPC $\rightarrow$ AC $\rightarrow$ BC.

Many algorithms enforce a local consistency property during search. For example, the *forward checking* algorithm (FC) maintains a restricted form of AC that ensures that current and future variables are AC. FC has been generalized to non-binary constraints [Bessière *et al.*, 1999]. nFC0 makes every $k$-ary constraint with $k-1$ variables instantiated AC. nFC1 applies (one pass of) AC to each constraint or constraint projection involving the current and exactly one future variable. nFC2 applies (one pass of) GAC to each constraint involving the current and at least one future variable. Three other generalizations of FC to non-binary constraints, nFC3 to nFC5 degenerate to nFC2 on the single non-binary constraint describing a permutation, so are not considered here. Finally, the *maintaining arc-consistency* algorithm (MAC) maintains AC during search, whilst MGAC maintains GAC.

## 3 Permutation problems

A *permutation problem* is a constraint satisfaction problem with the same number of variables as values, in which each variable takes an unique value. We also consider *multiple permutation problems* in which the variables divide into a number of (possibly overlapping) sets, each of which is a permutation problem. Smith has proposed a number of different models for permutation problems [Smith, 2000]. The *primal* not-equals model has not-equals constraints between the variables in each permutation. The *primal* all-different model has an all-different constraint between the variables in each permutation. In a *dual* model, we swop variables for values. *Primal and dual* models have primal and dual variables, and *channelling constraints* linking them of the form: $x_i = j$ iff $d_j = i$ where $x_i$ is a primal variable and $d_j$ is a dual variable. Primal and dual models can also have not-equals and all-different constraints on the primal and/or dual variables. There will, of course, typically be other constraints which depend on the nature of the permutation problem. For example, in the all-interval series problem from CSPLib, the variables and the differences between neighboring variables are both permutations. In what follows, we do not consider directly the contribution of such additional constraints to pruning. However, the ease with which we can specify and reason with these additional constraints may have a large impact on our choice of the primal, dual or primal and dual models.

We use the following subscripts: "$\neq$" for the primal not-equals constraints, "$c$" for channelling constraints, "$\neq c$" for the primal not-equals and channelling constraints, "$\neq c \neq$" for the primal not-equals, dual not-equals and channelling constraints, "$\forall$" for the primal all-different constraint, "$\forall c$" for the primal all-different and channelling constraints, and "$\forall c \forall$" for the primal all-different, dual all-different and channeling constraints. For example, SAC$_{\neq c}$ is SAC applied to the primal not-equals and channelling constraints.

## 4 Constraint tightness

To compare how different models of permutation problems prune the search tree, we define a new measure of constraint tightness. Our definition assumes constraints are defined over the same variables and values or, as in the case of primal and dual models, variables and values which are bijectively related. An interesting extension would be to compare two sets of constraints up to permutation of their variables and values. Our definition of constraint tightness is strongly influenced by the way local consistency properties are compared [Debruyne and Bessière, 1997]. Indeed, the definition is parameterized by a local consistency property since, as we show later, the amount of pruning provided by a set of constraints can depend upon the level of local consistency being enforced. This measure of constraint tightness would also be useful in a number of other applications (e.g. reasoning about the value of implied constraints).

We say that a set of constraints $A$ is *as tight as* a set $B$ with respect to $\Phi$-consistency (written $\Phi_A \leadsto \Phi_B$) iff, given any domains for their variables, if $A$ is $\Phi$-consistent then $B$ is also $\Phi$-consistent. Note that tightness is over all possible domains for the variables. It thus measures the possible pruning of domains during search as variables are instantiated and domains pruned (possibly by other constraints in the problem). We say that a set of constraints $A$ is *tighter* than a set $B$ wrt $\Phi$-consistency (written $\Phi_A \rightarrow \Phi_B$) iff $\Phi_A \leadsto \Phi_B$ but not $\Phi_B \leadsto \Phi_A$, $A$ is *incomparable* to $B$ wrt $\Phi$-consistency (written $\Phi_A \otimes \Phi_B$) iff neither $\Phi_A \leadsto \Phi_B$ nor $\Phi_B \leadsto \Phi_A$, and $A$ is *equivalent* to $B$ wrt $\Phi$-consistency (written $\Phi_A \leftrightarrow \Phi_B$) iff both $\Phi_A \leadsto \Phi_B$ and $\Phi_B \leadsto \Phi_A$. We can easily generalize these definitions to compare $\Phi$-consistency on $A$ with $\Theta$-consistency on $B$. This definition of constraint tightness has some nice monotonicity and fixed-point properties which we will use extensively throughout this paper.

**Theorem 1 (monotonicity and fixed-point)**

*1.* $AC_{A \cup B} \leadsto AC_A \leadsto AC_{A \cap B}$

*2.* $AC_A \rightarrow AC_B$ *implies* $AC_{A \cup B} \leftrightarrow AC_A$

Similar monotonicity and fixed-point results hold for BC, RPC, PIC, SAC, ACPC, and GAC. We also extend these definitions to compare constraint tightness wrt search algorithms like MAC that maintain some local consistency. For example, we say that $A$ is *as tight as* $B$ wrt algorithm $X$ (written $X_A \leadsto X_B$) iff, given any fixed variable and value odering and any domains for their variables, $X$ visits no more nodes on $A$ than on $B$, whilst $A$ is *tighter* than $B$ wrt algorithm $X$ (written $X_A \rightarrow X_B$) iff $X_A \leadsto X_B$ but not $X_B \leadsto X_A$. Similar monotonicity and fixed-point results can be given for FC, MAC and MGAC. Finally, we write $X_A \Rightarrow X_B$ if $X_A \rightarrow X_B$ and there is a problem on which $X$ visits exponentially fewer branches with $A$ than $B$.

## 5 Theoretical comparison

In an experimental study of Langford's problem, a simple permutation problem from CSPLib, Smith observes that channelling constraints remove the need for the primal not-equals constraints [Smith, 2000]. She also observes that MAC applied to a model of Langford's problem with channelling constraints explores more branches than MGAC applied to a model with a primal all-different constraint. We show that these results do not extend to algorithms that maintain higher levels of local consistency like PIC We also prove that the differences can lead to exponential reductions in runtime.

### 5.1 Arc-consistency

We first prove that, with repsect to arc-consistency, channelling constraints are tighter than the primal not-equals constraints, but less tight than the primal all-different constraint.

**Theorem 2** *On a permutation problem:*

$$\begin{array}{l} GAC_{\forall c \forall} \\ \updownarrow \\ GAC_{\forall} \rightarrow AC_{\neq c \neq} \leftrightarrow AC_{\neq c} \leftrightarrow AC_{c} \rightarrow AC_{\neq} \\ \updownarrow \\ GAC_{\forall c} \end{array}$$

**Proof:** We give proofs for the most important identities. Other results follow quickly, often using transitivity, and the monotonicity and fixed-point theorems.

To show $\text{GAC}_\forall \rightarrow \text{AC}_c$, consider a permutation problem whose primal all-different constraint is GAC. Suppose the channelling constraint between $x_i$ and $d_j$ was not AC. Then either $x_i$ is set to $j$ and $d_j$ has $i$ eliminated from its domain, or $d_j$ is set to $i$ and $x_i$ has $j$ eliminated from its domain. But neither of these two cases is possible by the construction of the primal and dual model. Hence the channelling constraints are all AC. To show strictness, consider a 5 variable permutation problem in which $x_1 = x_2 = x_3 = \{1, 2\}$ and $x_4 = x_5 = \{3, 4, 5\}$. This is $\text{AC}_c$ but not $\text{GAC}_\forall$.

To show $\text{AC}_c \rightarrow \text{AC}_{\neq}$, suppose that the channelling constraints are AC. Consider a not-equals constraint, $x_i \neq x_j$ $(i \neq j)$ that is not AC. Now, $x_i$ and $x_j$ must have the same singleton domain, $\{k\}$. Consider the channelling constraint between $x_i$ and $d_k$. The only AC value for $d_k$ is $i$. Similarly, the only AC value for $d_k$ in the channelling constraint between $x_j$ and $d_k$ is $j$. But $i \neq j$. Hence, $d_k$ has no AC values. This is a contradiction as the channelling constraints are AC. Hence all not-equals constraints are AC. To show strictness, consider a 3 variable permutation problem with $x_1 = x_2 = \{1, 2\}$ and $x_3 = \{1, 2, 3\}$. This is $\text{AC}_{\neq}$ but is not $\text{AC}_c$.

To show $\text{AC}_{\neq c \neq} \leftrightarrow \text{AC}_c$, by monotonicity, $\text{AC}_{\neq c \neq} \rightsquigarrow \text{AC}_c$. To show the reverse, consider a permutation problem which is $\text{AC}_c$ but not $\text{AC}_{\neq c \neq}$. Then there exists at least one not-equals constraints that is not AC. Without loss of generality, let this be on two dual variables (a symmetric argument can be made for two primal variables). So both the associated (dual) variables, call them $d_i$ and $d_j$ must have the same unitary domain, say $k$. Hence, the domain of the primal variable $x_k$ includes $i$ and $j$. Consider the channelling constraint between $x_k$ and $d_i$. Now this is not AC as the value $x_k = j$ has no support. This is a contradiction.

To show $\text{GAC}_{\forall c \forall} \leftrightarrow \text{GAC}_\forall$, consider a permutation problem that is $\text{GAC}_\forall$. For every possible assignment of a value to a variable, there exist a consistent extension to the other variables, $x_1 = d_{x_1}, \ldots x_n = d_{x_n}$ with $x_i \neq x_j$ for all $i \neq j$. As this is a permutation, this corresponds to the assignment of unique variables to values. Hence, the corresponding dual all-different constraint is GAC. Finally, the channelling constraints are trivially AC. QED.

## 5.2 Maintaining arc-consistency

These results can be lifted to algorithms that maintain (generalized) arc-consistency during search. Indeed, the gaps between the primal all-different and the channelling constraints, and between the channelling constraints and the primal not-equals constraints can be exponentially large. Recall that we write $X_A \Rightarrow X_B$ iff $X_A \rightarrow X_B$ and there is a problem on which algorithm $X$ visits exponentially fewer branches with $A$ than $B$. Note that $\text{GAC}_\forall$ and AC are both polynomial to enforce so an exponential reduction in branches translates to an exponential reduction in runtime.

**Theorem 3** *On a permutation problem:*

$$MGAC_\forall \Rightarrow MAC_{\neq c \neq} \leftrightarrow MAC_{\neq c} \leftrightarrow MAC_c \Rightarrow MAC_{\neq}$$

**Proof:** We give proofs for the most important identities. Other results follow immediately from the last theorem.

To show $\text{GMAC}_\forall \Rightarrow \text{MAC}_c$, consider a $n + 3$ variable permutation problem with $x_i = \{1, \ldots, n\}$ for $i \leq n + 1$ and $x_{n+2} = x_{n+3} = \{n + 1, n + 2, n + 3\}$. Then, given a lexicographical variable ordering, $\text{GMAC}_\forall$ immediately fails, whilst $\text{MAC}_c$ takes $n!$ branches.

To show $\text{MAC}_c \Rightarrow \text{MAC}_{\neq}$, consider a $n + 2$ variable permutation problem with $x_1 = \{1, 2\}$, and $x_i = \{3, \ldots, n + 2\}$ for $i \geq 2$. Then, given a lexicographical variable ordering, $\text{MAC}_c$ takes 2 branches to show insolubility, whist $\text{MAC}_{\neq}$ takes $2.(n - 1)!$ branches. QED.

## 5.3 Forward checking

Maintaining (generalized) arc-consistency on large permutation problems can be expensive. We may therefore consing using a more restricted local consistency property like forward checking. For example, the Choco finite-domain toolkit in Claire uses just nFC0 on all-different constraints. The channelling constraint remain tighter than the primal not-equals constraints wrt FC.

**Theorem 4** *On a permutation problem:*

$$\begin{array}{l} nFC2_\forall \rightarrow FC_{\neq c \neq} \leftrightarrow FC_{\neq c} \leftrightarrow FC_c \rightarrow FC_{\neq} \rightarrow nFC0_\forall \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad \uparrow \\ \qquad\qquad\qquad\qquad\quad nFC2_\forall \rightarrow nFC1_\forall \end{array}$$

**Proof:** We again prove the most important identities. Other results follow quickly, often by means of the transitivity, and the monotonicity and fixed-point theorems.

[Gent *et al.*, 2000] proves $\text{FC}_{\neq}$ implies $\text{nFC0}_\forall$. To show strictness on permutation problems (as opposed to the more general class of decomposable constraints studied in [Gent *et al.*, 2000]), consider again a 5 variable permutation problem with $x_1 = x_2 = x_3 = x_4 = \{1, 2, 3\}$ and $x_5 = \{4, 5\}$ Irrespective of the variable and value ordering, FC shows the problem is unsatisfiable in at most 12 branches. nFC0 by comparison takes at least 18 branches.

To show $\text{FC}_c \rightarrow \text{FC}_{\neq}$, consider assigning the value $j$ to the primal variable $x_i$. $\text{FC}_{\neq}$ removes $j$ from the domain of all other primal variables. $\text{FC}_c$ instantiates the dual variable $d_j$ with the value $i$, and then removes $i$ from the domain of all other primal variables. Hence, $\text{FC}_c$ prunes all the values that $\text{FC}_{\neq}$ does. To show strictness, consider a 4 variable permutation problem with $x_1 = \{1, 2\}$ and $x_2 = x_3 = x_4 = \{3, 4\}$. Given a lexicographical variable and numerical value ordering, $\text{FC}_{\neq}$ shows the problem is unsatisfiable in 4 branches. $\text{FC}_{\neq}$ by comparison takes just 2 branches.

[Gent *et al.*, 2000] proves $\text{nFC1}_\forall$ implies $\text{FC}_{\neq}$. To show the reverse, consider assigning the value $j$ to the primal variable $x_i$. $\text{FC}_{\neq}$ removes $j$ from the domain of all primal variables except $x_i$. However, $\text{nFC1}_\forall$ also removes $j$ from the domain of all primal variables except $x_i$ since each occurs in a binary not-equals constraint with $x_i$ obtained by projecting out the all-different constraint. Hence, nFC1$_\forall$ $\leftrightarrow$ FC$_{\neq}$.

To show nFC2$_\forall$ $\rightarrow$ FC$_{\neq c \neq}$, consider instantiating the primal variable $x_i$ with the value $j$. FC$_{\neq c \neq}$ removes $j$ from the domain of all primal variables except $x_i$, $i$ from the domain of all dual variables except $d_j$, instantiate $d_j$ with the value $i$, and then remove $i$ from the domain of all dual variables except $d_j$. nFC2$_\forall$ also removes $j$ from the domain of all primal variables except $x_i$. The only possible difference is if one of the other dual variables, say $d_l$ has a domain wipeout. If this happens, $x_i$ has one value in its domain, $l$ that is in the domain of no other primal variable. Enforcing GAC immediately detects that $x_i$ cannot take the value $j$, and must instead take the value $k$. Hence nFC2$_\forall$ has a domain wipeout whenever FC$_{\neq c \neq}$ does. To show strictness, consider a 7 variable permutation problem with $x_1 = x_2 = x_3 = x_4 = \{1, 2, 3\}$ and $x_5 = x_6 = x_7 = \{4, 5, 6, 7\}$ Irrespective of the variable and value ordering, FC$_{\neq c \neq}$ takes at least 6 branches to show the problem is unsatisfiable. nFC2$_\forall$ by comparison takes no more than 4 branches.

[Bessière *et al.*, 1999] proves nFC2$_\forall$ implies nFC1$_\forall$. To show strictness on permutation problems, consider a 5 variable permutation problem with $x_1 = x_2 = x_3 = x_4 = \{1, 2, 3\}$ and $x_5 = \{4, 5\}$ Irrespective of the variable and value ordering, nFC1 shows the problem is unsatisfiable in at least 6 branches. nFC2 by comparison takes no more than 3 branches. QED.

### 5.4 Bounds consistency

Another common method to reduce costs is to enforce just bounds consistency. For example, [Régin and Rueher, 2000] use bounds consistency rather than arc-consistency to efficiently prune a global constraint involving a sum of variables and a set of inequalities. As a second example, some of the experiments on permutation problems in [Smith, 2000] used bounds consistency on certain of the constraints. With bounds consistency on permutation problems, we obtain a very similar ordering of the models as with arc-consistency.

**Theorem 5** *On a permutation problem:*

$$BC_\forall \rightarrow BC_{\neq c \neq} \leftrightarrow BC_{\neq c} \leftrightarrow BC_c \rightarrow BC_{\neq} \leftarrow AC_{\neq}$$
$$BC_c \downarrow AC_{\neq}$$

**Proof:** To show BC$_c$ $\rightarrow$ BC$_{\neq}$, consider a permutation problem which is BC$_c$ but one of the primal not-equals constraints is not BC. Then, it would involve two variables, $x_i$ and $x_j$ both with identical interval domains, $[k, k]$. Enforcing BC on the channelling constraint between $x_i$ and $d_k$ would reduce $d_k$ to the domain $[i, i]$. Enforcing BC on the channelling constraint between $x_j$ and $d_k$ would then cause a domian wipeout. But this contradicts the channelling constraints being BC. Hence, all the primal not-equals constraints must be BC. To show strictness. consider a 3 variable permutation problem with $x_1 = x_2 = [1, 2]$ and $x_3 = [1, 3]$. This is BC$_{\neq}$ but not BC$_c$.

To show BC$_\forall$ $\leftarrow$ BC$_{\neq c \neq}$, consider a permutation probem which is BC$_\forall$. Suppose we assign a boundary value $j$ to a primal variable, $x_i$ (or equivalently, a boundary value $i$ to a dual variable, $d_j$). As the all-different constraint is BC, this can be extended to all the other primal variables using each of the values once. This gives us a consistent assignment for any other primal or dual variable. Hence, it is BC$_{\neq c \neq}$. To show strictness, consider a 5 variable permutation problem with $x_1 = x_2 = x_3 = [1, 2]$ and $x_4 = x_5 = [3, 5]$. This is BC$_{\neq c \neq}$ but not BC$_\forall$.

To show BC$_c$ $\leftarrow$ AC$_{\neq}$, consider a permutation problem which is BC$_c$ but not AC$_{\neq}$. Then they must be one constraint, $x_i \neq x_j$ with $x_i$ and $x_j$ having the same singleton domain, $\{k\}$. But, if this is the case, enforcing BC on the channelling constraint between $x_i$ and $d_k$ and between $x_j$ and $d_k$ would prove that the problem is unsatisfiable. Hence, it is AC$_{\neq}$. To show strictness, consider a 3 variable permutation problem with $x_1 = x_2 = [1, 2]$ and $x_3 = [1, 3]$. This is AC$_{\neq}$ but not BC$_c$. QED.

### 5.5 Restricted path consistency

Debruyne and Bessière have shown that RPC is a promising filtering technique above AC [Debruyne and Bessière, 1997]. It prunes many of the PIC values at little extra cost to AC. Surprisingly, channelling constraints are incomparable to the primal not-equals constraints wrt RPC. Channelling constraints can increase the amount of propagation (for example, when a dual variable has only one value left in its domain). However, RPC is hindered by the bipartite constraint graph between primal and dual variables. Additional not-equals constraints on primal and/or dual variables can therefore help propagation.

**Theorem 6** *On a permutation problem;*

$$GAC_\forall \rightarrow RPC_{\neq c \neq} \rightarrow RPC_{\neq c} \rightarrow RPC_c \otimes RPC_{\neq} \otimes AC_c$$

**Proof:** To show RPC$_c$ $\otimes$ RPC$_{\neq}$, consider a 4 variable permutation problem with $x_1 = x_2 = x_3 = \{1, 2, 3\}$ and $x_4 = \{1, 2, 3, 4\}$. This is RPC$_{\neq}$ but not RPC$_c$. For the reverse direction, consider a 5 variable permutation problem with $x_1 = x_2 = x_3 = \{1, 2\}$ and $x_4 = x_5 = \{3, 4, 5\}$. This is RPC$_c$ but not RPC$_{\neq}$.

To show RPC$_{\neq c}$ $\rightarrow$ RPC$_c$, consider again the last example. This is RPC$_c$ but not RPC$_{\neq c}$.

To show RPC$_{\neq c \neq}$ $\rightarrow$ RPC$_{\neq c}$, consider a 6 variable permutation problem with $x_1 = x_2 = \{1, 2, 3, 4, 5, 6\}$ and $x_3 = x_4 = x_5 = x_6 = \{4, 5, 6\}$. This is RPC$_{\neq c}$ but not RPC$_{\neq c \neq}$.

To show GAC$_\forall$ $\rightarrow$ RPC$_{\neq c \neq}$, consider a permutation problem which is GAC$_\forall$. Suppose we assign a value $j$ to a primal variable, $x_i$ (or equivalently, a value $i$ to a dual variable, $d_j$). As the all-different constraint is GAC, this can be extended to all the other primal variables using up all the other values. This gives us a consistent assignment for any two other primal or dual variables. Hence, the problem is PIC$_{\neq c \neq}$ and thus RPC$_{\neq c \neq}$. To show strictness, consider a 7 variable permutation problem with $x_1 = x_2 = x_3 = x_4 = \{1, 2, 3\}$ and $x_5 = x_6 = x_7 = \{4, 5, 6, 7\}$. This is RPC$_{\neq c \neq}$ but not GAC$_\forall$.

To show AC$_c$ $\otimes$ RPC$_{\neq}$, consider a 4 variable permutation problem with $x_1 = x_2 = x_3 = \{1, 2, 3\}$ and $x_4 = \{1, 2, 3, 4\}$. This is RPC$_{\neq}$ but not AC$_c$. For the reverse direction, consider a 5 variable permutation problem with $x_1 = x_2 = x_3 = \{1, 2\}$ and $x_4 = x_5 = \{3, 4, 5\}$. This is AC$_c$ but not RPC$_{\neq}$. QED.

## 5.6 Path inverse consistency

The incomparability of channelling constraints and primal not-equals constraints remains when we move up the local consistency hierarchy from RPC to PIC.

**Theorem 7** *On a permutation problem:*

$GAC_\forall \rightarrow PIC_{\neq c \neq} \rightarrow PIC_{\neq c} \rightarrow PIC_c \otimes PIC_{\neq} \otimes AC_c$

**Proof:** To show $\text{PIC}_c \otimes \text{PIC}_{\neq}$, consider a 4 variable permutation problem with $x_1 = x_2 = x_3 = \{1, 2, 3\}$ and $x_4 = \{1, 2, 3, 4\}$. This is $\text{PIC}_{\neq}$ but not $\text{PIC}_c$. Enforcing PIC on the channelling constraints reduces $x_4$ to the singleton domain $\{4\}$. For the reverse direction, consider a 5 variable permutation problem with $x_1 = x_2 = x_3 = \{1, 2\}$ and $x_4 = x_5 = \{3, 4, 5\}$. This is $\text{PIC}_c$ but not $\text{PIC}_{\neq}$.

To show $\text{PIC}_{\neq c} \rightarrow \text{PIC}_c$, consider a 5 variable permutation problem with $x_1 = x_2 = x_3 = \{1, 2\}$ and $x_4 = x_5 = \{3, 4, 5\}$. This is $\text{PIC}_c$ but not $\text{PIC}_{\neq c}$.

To show $\text{PIC}_{\neq c \neq} \rightarrow \text{PIC}_{\neq c}$, consider a 6 variable permutation problem with $x_1 = x_2 = \{1, 2, 3, 4, 5, 6\}$ and $x_3 = x_4 = x_5 = x_6 = \{4, 5, 6\}$. This is $\text{PIC}_{\neq c}$ but not $\text{PIC}_{\neq c \neq}$.

To show $\text{GAC}_\forall \rightarrow \text{PIC}_{\neq c \neq}$, consider a permutation problem in which the all-different constraint is GAC. Suppose we assign a value $j$ to a primal variable, $x_i$ (or equivalently, a value $i$ to a dual variable, $d_j$). As the all-different constraint is GAC, this can be extended to all the other primal variables using up all the other values. This gives us a consistent assignment for any two other primal or dual variables. Hence, the not-equals and channelling constraints are PIC. To show strictness, consider a 7 variable permutation problem with $x_1 = x_2 = x_3 = x_4 = \{1, 2, 3\}$ and $x_5 = x_6 = x_7 = \{4, 5, 6, 7\}$. This is $\text{PIC}_{\neq c \neq}$ but not $\text{GAC}_\forall$.

To show $\text{PIC}_{\neq} \otimes \text{AC}_c$, consider a 4 variable permutation problem with $x_1 = x_2 = x_3 = \{1, 2, 3\}$ and $x_4 = \{1, 2, 3, 4\}$. This is $\text{PIC}_{\neq}$ but not $\text{AC}_c$. Enforcing AC on the channelling constraints reduces $x_4$ to the singleton domain $\{4\}$. For the reverse direction, consider a 5 variable permutation problem with $x_1 = x_2 = x_3 = \{1, 2\}$ and $x_4 = x_5 = \{3, 4, 5\}$. This is $\text{AC}_c$ but not $\text{PIC}_{\neq}$. QED.

## 5.7 Singleton arc-consistency

Debruyne and Bessière also showed that SAC is a promising filtering technique above both AC, RPC and PIC, pruning many values for its CPU time [Debruyne and Bessière, 1997]. Prosser et al. reported promising experimental results with SAC on quasigroup problems, a multiple permutation problem [Prosser *et al.*, 2000]. Interestingly, as with AC (but unlike RPC and PIC which lie between AC and SAC), channelling constraints are tighter than the primal not-equals constraints wrt SAC.

**Theorem 8** *On a permutation problem:*

$GAC_\forall \rightarrow SAC_{\neq c \neq} \leftrightarrow SAC_{\neq c} \leftrightarrow SAC_c \rightarrow SAC_{\neq} \otimes AC_c$

**Proof:** To show $\text{SAC}_c \rightarrow \text{SAC}_{\neq}$, consider a permutation problem that is $\text{SAC}_c$ and any possible instantiation for a primal variable $x_i$. Suppose that the primal not-equals model of the resulting problem cannot be made AC. Then there must exist two other primal variables, say $x_j$ and $x_k$ which have at most one other value. Consider the dual variable associated with this value. Then under this instantiation of the primal variable $x_i$, enforcing AC on the channelling constraint between the primal variable $x_i$ and the dual variable, and between the dual variable and $x_j$ and $x_k$ results in a domain wipeout on the dual variable. Hence the problem is not $\text{SAC}_c$. This is a contradiction. The primal not-equals model can therefore be made AC following the instantiation of $x_i$. That is, the problem is $\text{SAC}_{\neq}$. To show strictness, consider a 5 variable permutation problem with domain $x_1 = x_2 = x_3 = x_4 = \{0, 1, 2\}$ and $x_5 = \{3, 4\}$. This is $\text{SAC}_{\neq}$ but not $\text{SAC}_c$.

To show $\text{GAC}_\forall \rightarrow \text{SAC}_c$, consider a permutation problem that is $\text{GAC}_\forall$. Consider any possible instantiation for a primal variable. This can be consistently extended to all variables in the primal model. But this means that it can be consistently extended to all variables in the primal and dual model, satisfying any (combination of) permutation or channelling constraints. As the channelling constraints are satisfiable, they can be made AC. Consider any possible instantiation for a dual variable. By a similar argument, taking the appropriate instantiation for the associated primal variable, the resulting problem can be made AC. Hence, given any possible instantiation for a primal or dual variable, the channelling constraints can be made AC. That is, the problem is $\text{SAC}_c$, To show strictness, consider a 7 variable permutation problem with $x_1 = x_2 = x_3 = x_4 = \{0, 1, 2\}$ and $x_5 = x_6 = x_7 = \{3, 4, 5, 6\}$. This $\text{SAC}_c$ but is not $\text{GAC}_\forall$.

To show $\text{SAC}_{\neq} \otimes \text{AC}_c$, consider a four variable permutation problem in which $x_1$ to $x_3$ have the $\{1, 2, 3\}$ and $x_4$ has the domain $\{0, 1, 2, 3\}$. This is $\text{SAC}_{\neq}$ but not $\text{AC}_c$. For the reverse, consider a 4 variable permutation problem with $x_1 = x_2 = \{0, 1\}$ and $x_3 = x_4 = \{0, 2, 3\}$. This is $\text{AC}_c$ but not $\text{SAC}_{\neq}$. QED.

## 5.8 Strong path-consistency

Adding primal or dual not-equals constraints to channelling constraints does not help AC or SAC. The following result shows that their addition does not help higher levels of local consistency like strong path-consistency (ACPC).

**Theorem 9** *On a permutation problem:*

$GAC_\forall \otimes ACPC_{\neq c \neq} \leftrightarrow ACPC_{\neq c} \leftrightarrow ACPC_c \rightarrow ACPC_{\neq} \otimes AC_c$

**Proof:** To show $\text{ACPC}_c \rightarrow \text{ACPC}_{\neq}$, consider some channelling constraints that are ACPC. Now $\text{AC}_c \rightarrow \text{AC}_{\neq}$, so we just need to show $\text{PC}_c \rightarrow \text{PC}_{\neq}$. Consider a consistent pair of values, $l$ and $m$ for a pair of primal variables, $x_i$ and $x_j$. Take any third primal variable, $x_k$. As the constraint between $d_l$, $d_m$ and $x_k$ is PC, we can find a value for $x_k$ consistent with the channelling constraints. But this also satisfies the not-equals constraint between primal variables. Hence, the problem is $\text{PC}_{\neq}$. To show strictness, consider a 4 variable permutation problem with $x_1 = x_2 = x_3 = x_4 = \{1, 2, 3\}$. This is $\text{ACPC}_{\neq}$ but not $\text{ACPC}_c$.

To show $\text{ACPC}_{\neq c \neq} \leftrightarrow \text{ACPC}_{\neq c} \leftrightarrow \text{ACPC}_c$, we recall that $\text{AC}_{\neq c} \leftrightarrow \text{AC}_{\neq c} \leftrightarrow \text{AC}_c$. Hence we need just show that $\text{PC}_{\neq c} \leftrightarrow \text{PC}_{\neq c} \leftrightarrow \text{PC}_c$. Consider a permutation problem. Enforcing PC on the channelling constraints alone infers both the primal and the dual not-equals constraints. Hence, $\text{PC}_{\neq c} \leftrightarrow \text{PC}_{\neq c} \leftrightarrow \text{PC}_c$.

To show $\text{GAC}_\forall \otimes \text{ACPC}_{\neq c \neq}$, consider a 6 variable permutation problem with $x_1 = x_2 = x_3 = x_4 = \{1, 2, 3\}$, and $x_5 = x_6 = \{4, 5, 6\}$. This is $\text{ACPC}_{\neq c \neq}$ but not $\text{GAC}_\forall$. For the reverse direction, consider a 3 variable permutation problem with the additional binary constraint $even(x_1 + x_3)$. Enforcing $\text{GAC}_\forall$. prunes the to $x_1 = x_3 = \{1, 3\}$, and $x_2 = \{2\}$. However, these domains are not $\text{ACPC}_{\neq c \neq}$. Enforcing ACPC tightens the constraint between $x_1$ and $x_3$ from not-equals to $x_1 = 1, x_3 = 3$ or $x_1 = 3, x_3 = 1$.

To show $\text{ACPC}_{\neq} \otimes \text{AC}_c$, consider a 5 variable permutation problem with $x_1 = x_2 = x_3 = \{1, 2\}$, and $x_4 = x_5 = \{3, 4, 5\}$. This is $\text{AC}_c$ but not $\text{ACPC}_{\neq}$. For the reverse direction, consider again the 4 variable permutation problem with $x_1 = x_2 = x_3 = x_4 = \{1, 2, 3\}$. This is $\text{ACPC}_{\neq}$ but not $\text{AC}_c$. QED.

### 5.9 Multiple permutation problems

These results extend to multiple permutation problems under a simple restriction that the problem is *triangle preserving* [Stergiou and Walsh, 1999] (that is, any triangle of not-equals constraints in the primal not-equals moel covers variables in the same permutation). For example, all-diff($x_1, x_2, x_4$), all-diff($x_1, x_3, x_5$), and all-diff($x_2, x_3, x_6$) are not triangle preserving as $x_1$, $x_2$ and $x_3$ occur in a triangle but are not in the same permutation. The following theorem collects together and generalizes many of the previous results.

**Theorem 10** *On a multiple permutation problem:*

$$\begin{array}{lllllllllll}
GAC_\forall & \otimes & ACPC_{\neq c \neq} & \leftrightarrow & ACPC_{\neq c} & \leftrightarrow & ACPC_c & \rightarrow & ACPC_{\neq} & \otimes & AC_c \\
 & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \\
GAC_\forall & \rightarrow & SAC_{\neq c \neq} & \leftrightarrow & SAC_{\neq c} & \leftrightarrow & SAC_c & \rightarrow & SAC_{\neq} & \otimes & AC_c \\
 & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \\
GAC_\forall & \rightarrow & PIC_{\neq c \neq} & \rightarrow & PIC_{\neq c} & \rightarrow & PIC_c & \otimes & PIC_{\neq} & \otimes & AC_c \\
 & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \\
GAC_\forall & \rightarrow & RPC_{\neq c \neq} & \rightarrow & RPC_{\neq c} & \rightarrow & RPC_c & \otimes & RPC_{\neq} & \otimes & AC_c \\
 & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \\
GAC_\forall & \rightarrow & AC_{\neq c \neq} & \leftrightarrow & AC_{\neq c} & \leftrightarrow & AC_c & \rightarrow & AC_{\neq} & \leftarrow & BC_c \\
\downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \\
BC_\forall & \rightarrow & BC_{\neq c \neq} & \leftrightarrow & BC_{\neq c} & \leftrightarrow & BC_c & \rightarrow & BC_{\neq} & &
\end{array}$$

**Proof:** The proofs lift in a straight forward manner from the single permutation case. Local consistencies like ACPC, SAC, PIC and RPC consider triples of variables. If these are linked together, we use the fact that the probem is triangle preserving and a permutation is therefore defined over them. If these are not linked together, we can decompose the argument into AC on pairs of variables. Without triangle preservation, $\text{GAC}_\forall$, may only achieve as high a level of consistency as $\text{AC}_{\neq}$. For example, consider again the non-triangle preserving constraints in the last paragraph. If $x_1 = x_2 = x_3 = \{1, 2\}$ and $x_4 = x_5 = x_6 = \{1, 2, 3\}$ then the problem is $\text{GAC}_\forall$, but it is not $\text{RPC}_{\neq}$, and hence neither $\text{PIC}_{\neq}$, $\text{SAC}_{\neq}$ nor $\text{ACPC}_{\neq}$. QED.

## 6 SAT models

Another solution strategy is to encode permutation problems into SAT and use a fast Davis-Putnam (DP) or local search procedure. For example, [Bejar and Manya, 2000] report promising results for propositional encodings of round robin problems, which include permutation constraints. We consider just "direct" encodings into SAT (see [Walsh, 2000] for more details). We have a Boolean variable $X_{ij}$ which is $true$ iff the primal variable $x_i$ takes the value $j$. In the primal SAT model, there are $n$ clauses to ensure that each primal variable takes at least one value, $O(n^3)$ clauses to ensure that no primal variable gets two values, and $O(n^3)$ clauses to ensure that no two primal variables take the same value. Interestingly the channelling SAT model has the same number of Boolean variables as the primal SAT model (as we can use $X_{ij}$ to represent both the $j$th value of the primal variable $x_i$ *and* the $i$th value for the dual variable $d_j$), and just $n$ additional clauses to ensure each dual variable takes a value. The $O(n^3)$ clauses to ensure that no dual variable gets two values are equivalent to the clauses that ensure no two primal variables get the same value. The following result show that DP can be placed between MAC and FC on these different models.

**Theorem 11** *On a permutation problem:*

$$\begin{array}{lllllllll}
MGAC_\forall & \rightarrow & MAC_{\neq c \neq} & \leftrightarrow & MAC_{\neq c} & \leftrightarrow & MAC_c & \rightarrow & MAC_{\neq} \\
 & & \downarrow & & \downarrow & & \downarrow & & \downarrow \\
MGAC_\forall & \rightarrow & DP_{\neq c \neq} & \leftrightarrow & DP_{\neq c} & \leftrightarrow & DP_c & \rightarrow & DP_{\neq} \\
 & & \downarrow & & \downarrow & & \updownarrow & & \updownarrow \\
MGAC_\forall & \rightarrow & FC_{\neq c \neq} & \leftrightarrow & FC_{\neq c} & \leftrightarrow & FC_c & \rightarrow & FC_{\neq}
\end{array}$$

**Proof:** $\text{DP}_{\neq} \leftrightarrow \text{FC}_{\neq}$ is a special case of Theorem 14 in [Walsh, 2000], whilst $\text{MAC}_{\neq} \rightarrow \text{FC}_{\neq}$ is a special case of Theorem 15.

To show $\text{DP}_c \leftrightarrow \text{FC}_c$ suppose unit propagation sets a literal $l$. There are four cases. In the first case, a clause of the form $X_{i1} \vee \ldots \vee X_{in}$ has been reduced to an unit. That is, we have one value left for a primal variable. A fail first heuristic in FC picks this one remaining value to instantiate. In the second case, a clause of the form $\neg X_{ij} \vee \neg X_{ik}$ for $j \neq k$ has been reduced to an unit. This ensures that no primal variable gets two values. The FC algorithm trivially never tries two simultaneous values for a primal variable. In the third case, a clause of the form $\neg X_{ij} \vee \neg X_{kj}$ for $i \neq k$ has been reduced to an unit. This ensures that no dual variable gets two values. Again, the FC algorithm trivially never tries two simultaneous values for a dual variable. In the fourth case, $X_{1j} \vee \ldots \vee X_{nj}$ has been reduced to an unit. That is, we have one value left for a dual variable. A fail first heuristic in FC picks this one remaining value to instantiate. Hence, given a suitable branching heuristic, the FC algorithm tracks the DP algorithm. To show the reverse, suppose forward checking removes a value. There are two cases. In the first case, the value $i$ is removed from a dual variable $d_j$ due to some channelling constraint. This means that there is a primal variable $x_k$ which has been set to some value $l \neq j$. Unit propagation on $\neg X_{kl} \vee \neg X_{kj}$ sets $X_{kj}$ to false. Unit propagation on $\neg X_{ij} \vee \neg X_{kj}$ then sets $X_{ij}$ to false as required. In the second case, the value $i$ is removed from a dual variable $d_j$, again due to some channelling constraint. The proof is now dual to the first case.

To show $\text{MAC}_c \rightarrow \text{DP}_c$, we use MAC $\rightarrow$ FC and $\text{FC}_c \leftrightarrow \text{DP}_c$. To show strictness, consider a permutation problem in three variables with additional binary constraints that rule out the same value for all three primal variables. Enforcing AC on the channelling constraints immediately causes a domain wipeout on the dual variable associated with this value. As their are no unit constraints, DP does not immediately solve the problem.

To show $DP_c \rightarrow DP_{\neq}$, we note that the channelling SAT model constrains more clauses. Hence, it dominates the primal SAT model. To show strictness, consider a four variable permutation problem with three additional binary constraints that if $x_1 = 1$ then $x_2 = 2$, $x_3 = 2$ and $x_4 = 2$ are all ruled out. Consider branching on $x_1 = 1$. Unit propagation on both models sets $X_{12}$, $X_{22}$, $X_{32}$, $X_{42}$, $X_{21}$, $X_{31}$ and $X_{41}$ to false. On the channelling SAT model, unit propagation against the clause $X_{12} \vee X_{22} \vee X_{32} \vee X_{42}$ then generates an empty clause. By comparison, unit propagation on the primal SAT model does no more work. QED.

## 7 Asymptotic comparison

The previous results tell us nothing about the relative cost of achieving these local consistencies. Asymptotic analysis adds detail to the results. Regin's algorithm achieves $GAC_{\forall}$ in $O(n^4)$ [Régin, 1994]. AC on binary constraints can be achieved in $O(ed^2)$ where $e$ is the number of constraints and $d$ is their domain size. As there are $O(n^2)$ channelling constraints, $AC_c$ naively takes $O(n^4)$ time. However, by taking advantage of the functional nature of channelling constraints, we can reduce this to $O(n^3)$ using the AC-5 algorithm of [van Hentenryck *et al.*, 1992]. $AC_{\neq}$ also naively takes $O(n^4)$ time as there are $O(n^2)$ binary not-equals constraints. However, we can take advantage of the special nature of a binary not-equals constraint to reduce this to $O(n^2)$ with careful implementation as each not-equals constraint needs to be made AC just once. Asymptotic analysis thus offers no great surprises: we proved that $GAC_{\forall} \rightarrow AC_c \rightarrow AC_{\neq}$ and this is reflected in their $O(n^4)$, $O(n^3)$, $O(n^2)$ respective costs.

## 8 Experimental comparison

On Langford's problem, a permutation problem from CSPLib, Smith found that MAC on the channelling and other problem constraints is often the most competitive model for finding all solutions [Smith, 2000]. $MAC_c$ (which takes $O(n^2)$ time at each node in the search tree if carefully implemented) explores a similar number of branches to the more powerful $MGAC_{\forall}$ (which takes $O(n^4)$ time at each node in the search tree). This suggests that $MAC_c$, if carefully implemented, may offer a good tradeoff between the amount of constraint propagation and the amount of search required. For finding single solutions, Smith's results are somewhat confused by the accuracy of the heuristic. She predicts that these results will transfer over to other permutation problems. To confirm this, we ran experiments in three other domains using the Sicstus finite domain constraint library.

### 8.1 All-interval series

Hoos has proposed the all-interval series problem from musical composition as a challenging benchmark for CSPLib. The $ais(n)$ problem is to find a permutation of the numbers 1 to $n$, such that the differences between adjacent numbers form a permutation from 1 to $n - 1$. Whilst polynomial solutions to $ais(n)$ exist, it remains difficult to compute all solutions. As on Langford's problem [Smith, 2000], $MAC_c$ visits only a few more branches than $MGAC_{\forall}$. Efficiently implemented, $MAC_c$ is therefore the quickest solution method.

| $n$ | $MAC_{\neq}$ | $MAC_c$ | $MGAC_{\forall}$ |
|---|---|---|---|
| 6 | 135 | 34 | 34 |
| 7 | 569 | 153 | 152 |
| 8 | 2608 | 627 | 626 |
| 9 | 12137 | 2493 | 2482 |
| 10 | 60588 | 10552 | 10476 |
| 11 | 318961 | 47548 | 47052 |

Table 1: Branches to compute all solutions to $ais(n)$.

### 8.2 Circular Golomb rulers

A perfect circular Golomb ruler consists of $n$ marks arranged on the circumference of a circle of length $n(n-1)$ such that the distances between any pair of marks, in either direction along the circumference, form a permutation. Again polynomial solutions exist for certain $n$, but it is difficult to compute all solutions or prove for some $n$ (like $n = 7$) that no perfect ruler exists. Table 2 shows that $MGAC_{\forall}$ is very competitive with $MAC_c$. Indeed, $MGAC_{\forall}$ has the smallest runtimes. We conjecture that this is due to circular Golomb rulers being more constrained than all-interval series.

| $n$ | $MAC_{\neq}$ | $MAC_c$ | $MGAC_{\forall}$ |
|---|---|---|---|
| 6 | 202 | 93 | 53 |
| 7 | 1658 | 667 | 356 |
| 8 | 15773 | 5148 | 2499 |
| 9 | 166424 | 43261 | 19901 |

Table 2: Branches to compute all order $n$ perfect circular Golomb rulers.

### 8.3 Quasigroups

Achlioptas et al have proposed completing a partial filled quasigroup as a challenging benchmark for SAT and CSP algorithms [Achlioptas *et al.*, 2000]. This can be modeled as a multiple permutation problem consisting of $2n$ intersecting permutation constraints. A complexity peak is observed when approximately 40% of the entries in the quasigroup are replaced by "holes". Table 3 shows the increase in problem difficulty with $n$. Median behavior for $MAC_c$ is competitive with $MGAC_{\forall}$. However, mean performance is not due to a few expensive outliers. A randomization and restart strategy reduces the size of this heavy-tailed distribution.

## 9 Extensions

### 9.1 Injective mappings

In many problems, variables may be constrained to take unique values, but we have more values than variables. That is, we are looking for an injective mapping from the variables to the values. For example, an optimal 5-tick Golomb ruler

| $n$ | median | | | mean | | |
|---|---|---|---|---|---|---|
| | MAC$_{\neq}$ | MAC$_c$ | MGAC$_{\forall}$ | MAC$_{\neq}$ | MAC$_c$ | MGAC$_{\forall}$ |
| 5 | 1 | 1 | 1 | 1 | 1 | 1 |
| 10 | 1 | 1 | 1 | 1.03 | 1.00 | 1.01 |
| 15 | 3 | 1 | 1 | 7.17 | 1.17 | 1.10 |
| 20 | 23313 | 7 | 4 | 312554 | 21.76 | 12.49 |
| 25 | - | 249 | 53 | - | 8782.4 | 579.7 |
| 30 | - | 5812 | 398 | - | 2371418 | 19375 |

Table 3: Median and mean branches to complete 100 order $n$ quasigroup problems with 40% holes.

has ticks at the marks 0, 1, 4, 9, and 11. The 10 inter-tick distances are all different but do not form a permutation as the distance 6 is absent. Finding a 5-tick Golomb ruler of length 11 can be modeled as a permutation problem by introducing an additional 11th variable to take on the missing value 6. In general, we can model an injective mapping from a domain of $n$ elements into an image of $m$ elements ($n \leq m$) as a permutation problem by introducing $m - n$ new primal variables. We can then post channelling constraints between the $m$ primal variables and $m$ dual variables. Most of our results about permutation problems map over to such problems with little or no modification. For example, AC on the channelling constraints of such a problem is tighter than AC on the primal not-equals constraints.

### 9.2 Bijective channelling constraints

Channelling constraints are useful in a wider class of problems than permutation problems. For example, the key modeling decision (according to [Hentenryck *et al.*, 1999]) for a tournament scheduling problem was to introduce two types of variables, one set for the teams and one for the games, with bijective channelling constraints between them. Consider a set of channelling constraints between $n$ primal variables, $x_i$ and $m$ dual variables, $d_j$ (with $n$ not necessarily equal to $m$). We say that they are bijective iff the tuples $\langle\langle x_1, \ldots, x_n\rangle, \langle d_1 \ldots, d_m\rangle\rangle$ made from assignments satisfying the channelling constraints define a bijective relation. Note that, despite the existence of a bijection, $x_i$ and $d_j$ may not have the same cardinalities as their domain sizes can be different. As in permutation problems, these quickly propagate values between the primal and dual variables and vice versa. Not all channelling constraints are bijective. The Golomb ruler provides an interesting example. The difference equations used in [Smith *et al.*, 2000], $d_{ij} = |x_i - x_j|$ can be seen as channelling constraints linking the initial variables with the auxiliary variables. However, they are not bijective. For instance, both $x_1 = 2$, $x_2 = 4$ and $x_1 = 3$, $x_2 = 5$ map onto $d_{12} = 2$.

## 10 Related work

Chen et al. studied modeling and solving the $n$-queens problem, and a nurse rostering problem using channelling constraints and “redundant models” (simultaneous primal and dual models) [Cheng *et al.*, 1999]. They show that channelling constraints increase the amount of constraint propagation. They conjecture that the overheads associated with channelling constraints will pay off on problems which require large amounts of search, or lead to thrashing behavior. They also show that redundant modeling opens the door to interesting value ordering heuristics.

As mentioned before, Smith studied a number of different models for Langford’s problem, a permutation problem in CSPLib [Smith, 2000]. This was the starting point for much of this research. Smith argues that channelling constraints make primal not-equals constraints redundant. She also observes that MAC on the model of Langford’s problem using channelling constraints explores more branches than MGAC on the model using a primal all-different constraint, and the same number of branches as MAC on the model using channelling and primal not-equals constraints. Smith also shows the benefits of being able to branch on dual variables.

## 11 Conclusions

We have performed an extensive study of models of permutation problems proposed by Smith in [Smith, 2000] with all-different constraints, channelling constraints and not-equals constraints. To compare models, we defined a measure of constraint tightness parameterized by the level of local consistency being enforced. We used this to prove that, with respect to arc-consistency, a single primal all-different constraint is tighter than channelling constraints, but that channelling constraints are tighter than primal not-equals constraints. Both these gaps can lead to an exponential reduction in search cost. For lower levels of local consistency (e.g. that maintained by forward checking), channelling constraints remain tighter than primal not-equals constraints. However, for certain higher levels of local consistency like path inverse consistency, channelling constraints are incomparable to primal not-equals constraints. On SAT encodings of permutation problems, we proved that the performance of the Davis Putnam algorithm is sandwiched between that of the MAC and FC algorithms.

Experimental results on three different permutation problems confirmed that MAC on channelling constraints outperformed MAC on primal not-equals constraints, and could be competitive with maintaining GAC on a primal all-different constraint. However, on more constrained problems, the additional constraint propagation provided by maintaining GAC on the primal all-different constraint was beneficial. We believe that these results will aid users of constraints to choose a model for a permutation problem, and a local consistency property to enforce on it. They also illustrate a methodology, as well as a measure of constraint tightness, that can be used to compare different constraint models in other problem domains.

## Acknowledgements

The author is an EPSRC advanced research fellow. He thanks the other members of the APES research group (http://apes.cs.strath.ac.uk/), especially Barbara Smith for helpful discussions, and Carla Gomes and her colleagues for providing code to generate quasigroups with holes.

## References


[Achlioptas *et al.*, 2000] Dimitris Achlioptas, Carla P. Gomes, Henry A. Kautz, and Bart Selman. Generating satisfiable problems instances. In *Proceedings of 17th National Conference on Artificial Intelligence*, pages 256–261. AAAI Press/The MIT Press, 2000.

[Bejar and Manya, 2000] R. Bejar and F. Manya. Solving the round robin problem using propositional logic. In *Proceedings of 17th National Conference on Artificial Intelligence*, pages 262–266. AAAI Press/The MIT Press, 2000.

[Bessière *et al.*, 1999] C. Bessière, P. Meseguer, E.C. Freuder, and J. Larrosa. On forward checking for non-binary constraint satisfaction. In J. Jaffar, editor, *Proceedings of Fifth International Conference on Principles and Practice of Constraint Programming (CP99)*, pages 88–102. Springer, 1999.

[Cheng *et al.*, 1999] B.M.W. Cheng, K.M.F. Choi, J.H.M. Lee, and J.C.K. Wu. Increasing constraint propagation by redundant modeling: an experience report. *Constraints*, 4:167–192, 1999.

[Debruyne and Bessière, 1997] R. Debruyne and C. Bessière. Some practicable filtering techniques for the constraint satisfaction problem. In *Proceedings of the 15th IJCAI*, pages 412–417. International Joint Conference on Artificial Intelligence, 1997.

[Freuder, 1985] E. Freuder. A sufficient condition for backtrack-bounded search. *Journal of the Association for Computing Machinery*, 32(4):755–761, 1985.

[Gent *et al.*, 2000] I.P. Gent, K. Stergiou, and T. Walsh. Decomposable constraints. *Artificial Intelligence*, 123(1-2):133–156, 2000.

[Hentenryck *et al.*, 1998] P. Van Hentenryck, V. Saraswat, and Y. Deville. Design, implementation and evaluation of the constraint language cc(fd). *Journal of Logic Programming*, 37(1–3):139–164, 1998.

[Hentenryck *et al.*, 1999] P. Van Hentenryck, L. Michel, L. Perron, and J.C. Regin. Constraint programming in OPL. In *Proceedings of the International Conference on the Principles and Practice of Declarative Programming (PPDP'99)*, 1999.

[Mohr and Masini, 1988] R. Mohr and G. Masini. Good old discrete relaxation. In *Proceedings of the European Conference on Artificial Intelligence (ECAI-88)*, pages 651–656. European Conference on Artificial Intelligence, 1988.

[Prosser *et al.*, 2000] P. Prosser, K. Stergiou, and T. Walsh. Singelton consistencies. In Rina Dechter, editor, *6th International Conference on Principles and Practices of Constraint Programming (CP-2000)*, pages 353–368. Springer-Verlag, 2000.

[Régin and Rueher, 2000] J.C. Régin and M. Rueher. A global constraint combining a sum constraint and difference constraints. In R. Dechter, editor, *Proceedings of 6th International Conference on Principles and Practice of Constraint Programming (CP2000)*, pages 384–395. Springer, 2000.

[Régin, 1994] J.C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the 12th National Conference on AI*, pages 362–367. American Association for Artificial Intelligence, 1994.

[Smith *et al.*, 2000] B. Smith, K. Stergiou, and T. Walsh. Using auxiliary variables and implied constraints to model non-binary problems. In *Proceedings of the 16th National Conference on AI*, pages 182–187. American Association for Artificial Intelligence, 2000.

[Smith, 2000] B.M. Smith. Modelling a Permutation Problem. In *Proceedings of ECAI'2000 Workshop on Modelling and Solving Problems with Constraints*, 2000. Also available as Research Report from `http://www.comp.leeds.ac.uk/bms/papers.html`.

[Stergiou and Walsh, 1999] K. Stergiou and T. Walsh. The difference all-difference makes. In *Proceedings of 16th IJCAI*. International Joint Conference on Artificial Intelligence, 1999.

[van Hentenryck *et al.*, 1992] P. van Hentenryck, Y. Deville, and C. Teng. A Generic Arc Consistency Algorithm and its Specializations. *Artificial Intelligence*, 57:291–321, 1992.

[Walsh, 2000] T. Walsh. SAT v CSP. In Rina Dechter, editor, *6th International Conference on Principles and Practices of Constraint Programming (CP-2000)*, pages 441–456. Springer-Verlag, 2000.