# Symmetry and Consistency

Ian P. Gent[1], Tom Kelsey[1], Steve Linton[1], Colva Roney-Dougal[1]

[1]School of Computer Science, University of St Andrews, St Andrews,
Fife, KY16 9SX, UK
{ipg,tom,sal,colva}@dcs.st-and.ac.uk

**Abstract.** We introduce a novel and exciting research area: symmetrising levels of consistency to produce stronger forms of consistency and more efficient mechanisms for establishing them. We propose new levels of consistency for Constraint Satisfaction Problems (CSPs) incorporating the symmetry group of a CSP. We first define Sym$(i, j)$-consistency, show that even Sym(1,0)-consistency can prune usefully, and study some consequences of maintaining Sym(i, 0)-consistency. We then present pseudocode for SymPath consistency, and a symmetrised version of singleton consistency, before presenting experimental evidence of these algorithms' practical effectiveness. With this contribution we establish the study of symmetry-based levels of consistency of CSPs.

## 1 Introduction

Symmetries arise in many Constraint Satisfaction Problems (CSPs). A rapidly growing literature looks at avoiding redundant search (and duplicate solutions) through a variety of techniques, such as enforcing a lexicographic ordering by enforcing lexicographic constraints [1], adding constraints dynamically during search [2], backtracking from the current node when it can be shown to be equivalent to a previous nogood [3] or constructing trees in which the symmetry has been eliminated [4].

Constraint solving is a balance between search and inference. There are various levels of consistency that can be maintained while searching for a solution, and many algorithms for enforcing levels of consistency. One can pick a level of consistency such as arc consistency (AC), a particular approach such as AC-3, and still find a variety of algorithms using interesting variants of that technique [5–7].

This work is foundational in establishing how symmetry and inference can be incorporated to the benefit of search in CSPs. At the heart of the thinking behind this research is the simple fact that any time we learn something about a CSP, the same is true of its symmetric equivalents. We suggest ways in which this insight can be used, specifically a number of new levels of consistency. These levels of consistency do one of two things: they either exploit the group structure to establish a higher level of consistency than corresponding notions without symmetry; or they establish the same level of consistency but the algorithm to establish consistency can exploit the group structure to potentially run faster. This paper considers both possibilities. The only precursor we are aware of is an exploitation of symmetry in a variant of AC2001 [8].

In Section 2 we introduce the fundamental definitions of CSPs, symmetries and consistency. In Section 3 we define a new, symmetric, kind of consistency called Sym$(i, j)$-consistency, and go on to study various specialisations of it such as Sym$(i, 0)$-consistency.

In Section 4 we present pseudocode for enforcing a symmetrised version of path consistency, then in Section 5 we present pseudocode for enforcing symmetrised singleton consistency. Both of these algorithms generalise the optimal known algorithms for their unsymmetrised versions. In Section 6 we present experimental evidence of the practical effectiveness of our new algorithms, before concluding with a discussion some of the possible directions for this exciting new research area.

## 2 Background and Definitions

**Definition 1.** *A CSP $P$ is a triple $(\Delta, \mathcal{D}, \mathcal{C})$, where $\Delta$ is a finite indexed set of variables $x_1, x_2, \ldots, x_n$, each of which has finite domain of possible values $D_i := \mathrm{Dom}(x_i) \subseteq \Lambda$. The set $\mathcal{D} = \{D_i : 1 \leq i \leq n\}$, and the set $\mathcal{C}$ is a finite set of constraints on the variables in $\Delta$. A* solution *to a CSP is an instantiation of all of the variables in $\Delta$ such that all of the constraints in $\mathcal{C}$ are satisfied.*

Statements of the form $(var = val)$ are *literals*: we denote the set of all literals of the CSP by $\chi(\Delta, \mathcal{D})$, or simply $\chi$ when the meaning is clear. We will write the literal $(x_i = a)$ as $(x_i, a)$, or occasionally $(i, a)$.

**Definition 2.** *A set of literals is a* partial assignment *if each variable occurs at most once in it, and a* full assignment *if each variable occurs exactly once.*

Let $C$ be constraint and $\mathcal{I}$ a partial assignment, then by $\mathrm{Var}(C)$ we denote the *scope* of $C$, namely the variables over which the constraint $C$ is defined, and by $\mathrm{Var}(\mathcal{I})$ we denote the variables in $\mathcal{I}$. We say that $\mathcal{I}$ *satisfies* $C$ if $\mathcal{I}$ contains assignments to all variables in $\mathrm{Var}(C)$, and the restriction of $\mathcal{I}$ to $\mathrm{Var}(C)$ is a permitted tuple of $C$. The partial assignment $\mathcal{I}$ *violates* $C$ if it contains assignments to all variables in the scope of $C$, but the restriction of $\mathcal{I}$ to $\mathrm{Var}(C)$ is a forbidden tuple of $C$. A partial assignment $\mathcal{I}$ is *consistent* if it satisfies all of the constraints that have no uninstantiated variables.

A permutation $f$ of a set $X$ is a bijection $f : X \to X$. We will denote the image of a point $x \in X$ under the map $f$ by $xf$. This notation (which comes from group theory) means that if we apply a permutation $f$ to $x \in X$ and then a permutation $g$ to the result we can simply write $xfg$, rather than the more cumbersome $g(f(x))$. Given any permutation $f$ on a set $X$ we will abuse notation and allow $f$ to act on (ordered or unordered) subsets of $X$, via $\{x_1, \ldots, x_n\}f := \{x_1 f, \ldots, x_n f\}$. Since $f$ is an injection the size of the image set is the same as the size of the original set.

**Definition 3.** *Given a CSP $P = (\Delta, \mathcal{D}, \mathcal{C})$, a* symmetry *of $P$ is a permutation of $\chi(\Delta, \mathcal{D})$ such that a full assignment $\mathcal{A}$ is a solution if and only if $\mathcal{A}f$ is a solution.*

It is well known that the collection of all symmetries of a CSP forms a *group*, that is, the composition of any two symmetries is itself a symmetry, and the inverse of a symmetry is a symmetry. To see this we note that if $f$ and $g$ are symmetries of a CSP $P$, then for any solution $\mathcal{S} \subseteq \chi(\Delta, \mathcal{D})$ the set $\mathcal{S}fg = (\mathcal{S}f)g$ is a solution. To see that the inverse of a symmetry is a symmetry, note that if $\mathcal{S}$ is a solution, and $\mathcal{S}f^{-1}$ is not a solution, then $f$ is not a symmetry, since $(\mathcal{S}f^{-1})f = \mathcal{S}$. Any group $G$ has a distinguished element called the *identity*, denoted $1_G$ or simply 1, with the property that acting with $1_G$ fixes everything.

Let $G$ be a group of permutations of a set $\Omega$, and let $a \in \Omega$. The *orbit*, $a^G$, of $a$ under $G$ is the set of all elements of $\Omega$ to which $a$ may be mapped by permutations in $G$.

Formally, $a^G := \{ag \ : \ g \in G\}$. The *stabiliser* of $a$ in $G$ is $G_a := \{g \in G \ : \ ag = a\}$, the set of all permutations in $G$ that map $a$ to itself. Let $A := \{a_1, \ldots, a_m\} \subseteq \Omega$. Then the *pointwise stabiliser* of $A$ is $G_{(A)} := \{g \in G \ : \ a_i g = a_i \text{ for } 1 \leq i \leq m\}$, namely the set of all permutations in $G$ that map each point in $A$ to itself. The *setwise stabiliser* of $A$ is $G_{\{A\}} := \{g \in G \ : \ \text{ for all } i \text{ there exists } j \text{ with } a_i g = a_j\}$. That is, the setwise stabiliser is the set of all permutations in $G$ that map the set $A$ to itself. It is an elementary fact that for any $a \in \Omega$ the point stabiliser $G_a$ is a subgroup of $G$ (a subset of $G$ that is itself a group under the same operation as in $G$), and also that any setwise or pointwise stabiliser is a subgroup of $G$.

Our definition of symmetry is very general. In particular, if $\mathcal{A}$ is a consistent partial assignment, then, provided that $\mathcal{A}$ is not a subset of a solution, it is possible that there exists a symmetry $g$ such that $\mathcal{A}g$ violates a constraint. For example, let $P$ be a CSP with variables $x, y, z$, each with domain $[1, 2]$, and constraint set $\{x = y, y = z\}$. Then the symmetries of $P$ allow us to freely interchange $(x, i), (y, i)$ and $(z, i)$, for any fixed $i \in [1, 2]$, as this will preserve both of the solutions. Thus in particular there is a symmetry $f$ which maps $(y, 2) \mapsto (z, 2)$, $(z, 2) \mapsto (y, 2)$, and fixes all other literals. However, the partial assignment $\{(x, 1), (z, 2)\}$ is mapped by $f$ to the partial assignment $\{(x, 1), (y, 2)\}$, and this latter partial assignment violates a constraint.

Symmetries need not even map partial assignments to partial assignments, and this is perfectly acceptable. In the 8-queens puzzle, there is a symmetry which rotates the square by 90 degrees. Suppose we use the standard model, with one variable for the placement of the queen in each row. The partial assignment $\{(Q_1, 3), (Q_2, 3)\}$ maps to $\{(Q_3, 8), (Q_3, 7)\}$, which involves two values for $Q_3$ and therefore is not a partial assignment. However, the initial assignment violates the constraint that there is only one queen in each column. This observation generalises: if a symmetry $f$ maps a partial assignment $\mathcal{A}$ to a set of literals $\mathcal{A}f$ that is not a partial assignment, then $\mathcal{A}$ is not a subset of any solution. To see this, suppose that $\mathcal{A} \subset \mathcal{S}$ for some solution $\mathcal{S}$, and note that by definition of $f$ we have $\mathcal{A}f \subset \mathcal{S}f$. Conversely, the inverse of $f$ will map a collection of literals that a not a partial assigment to one that is, in that $f^{-1}$ maps $\mathcal{A}f$ to $\mathcal{A}$. This is one reason why we define symmetries as acting on literals and then induce up to sets of literals, rather than defining them originally as acting on partial assignments.

**Definition 4.** *Given a CSP $P = (\Delta, \mathcal{D}, \mathcal{C})$, a symmetry $f$ of $P$ is* strict *if for all sets $\mathcal{A} \subseteq \chi(\Delta, \mathcal{D})$ of literals, $\mathcal{A}$ is a consistent partial assignment if and only if $\mathcal{A}f$ is a consistent partial assignment.*

A symmetry is *not* strict if it can map a partial assignment violating some constraint to a partial assignment not violating any constraints, or *vice versa*.

**Definition 5.** *Let $L = (\Delta, \mathcal{D}, \mathcal{C})$ be a CSP with symmetry group $G$. A* value symmetry *of $L$ is a symmetry $g \in G$ such that if $(x_i, a)g = (x_j, b)$ then $x_i = x_j$. Denote the elements of $D_i \in \mathcal{D}$ by $a_{ij}$. A* variable *symmetry of $L$ is a symmetry $g \in G$ such that if $(x_i, a_{ij})g = (x_k, b_{kl})$ then $j = l$. In the case where the variables have common domains then we denote the elements of $\Lambda$ as $a_k$, and the condition for a symmetry to be a variable symmetry can be simplified to: if $(x_i, a_k)g = (x_j, a_l)$ then $a_k = a_l$. That is, the symmetry fixes the values in each literal.*

We distinguish value and variable symmetries, as these have particularly nice algorithmic properties. If all elements of $G$ are *pure* value symmetries (value symmetries

such that if $(x_i, a)g = (x_i, b)$ then for all $j$ we have $(x_j, a)g = (x_j, b)$) then we can consider $G$ to consist of permutations of $\Lambda$. Similarly, if $G$ contains only pure variable symmetries then we can consider it to consist of permutations of $\Delta$. See [4, 9] for more details. Note that we cannot in general write $(x_i, a)g$ as $(x_i g, ag)$ as the variable in $(x_i, a)g$ may depend on the choice of value $a$. The 8-queens example before Definition 4 gives a symmetry which demonstrates this.

We finish this section with some definitions of consistency. A CSP $P$ is *k-consistent* if given a consistent partial assignment on any $k-1$ variables, along with a $k$-th variable, one can find a value for the $k$-th variable such that the resulting partial assignment of size $k$ is consistent. A CSP that is $k$-consistent need not be $(k-1)$-consistent: see [10].

We now recall the more general form of consistency called $(i, j)$-*consistency* [11].

**Definition 6.** *Suppose that in a CSP $P$, given any consistent partial assignment on $i$ variables, and given any other $j$ variables, it is possible to find values for the additional $j$ variables such that resulting partial assignment of size $i + j$ is consistent. Then $P$ is* (i, j)-consistent.

Thus in this notation $k$-consistency is $(k-1, 1)$-consistency, and arc consistency is $(1, 1)$-consistency. We enforce $(i, j)$-consistency by posting constraints of arity $i$ that expressly forbid each $i$-tuple that cannot be extended.

We denote a binary constraint between $x_i$ and $x_j$ by $c_{ij}$. A CSP $P = (\Delta, \mathcal{D}, \mathcal{C})$ is *path consistent* if and only if for any $c_{ij} \in \mathcal{C}$, any tuple $(a, b) \in c_{ij}$ and any variable $x_k \in \Delta \setminus \{x_i, x_j\}$, there exists a value $v \in D_k$ such that $\{(x_i, a), (x_j, b), (x_k, v)\}$ is a consistent partial assignment. For CSPs with no ternary constraints, path consistency is the same as $(2, 1)$-consistency; however in the presence of ternary constraints path consistency and 3-consistency differ.

## 3 Consistency and symmetry

In this section we extend $(i, j)$-consistency to use additional information from the symmetry group of a CSP. We will then examine when this coincides with existing levels of consistency. The following proposition is one of the main motivations for our work.

**Proposition 1.** *If the partial assignment $\{(x_i, a_i) : i \in I\}$ can be extended to a solution of a CSP, then for all symmetries $g$ the assignment $\{(x_i, a_i)g : i \in I\}$ violates no constraints.*

*Proof.* Suppose not, that is, suppose that $\{(x_i, a_i) : i \in I \cup J\}$ is a solution, but that for some symmetry $g$ the assignment $\{(x_i, a_i)g : i \in I\}$ violates a constraint $C$. Then $\{(x_i, a_i)g : i \in I \cup J\}$ is a full assignment which is not a solution, contradicting the definition of a symmetry.

Thus if one discovers that $\{(x_i, a_i) : i \in I\}$ violates a constraint then none of its images under $G$ can be part of a solution, hence they can all be forbidden without compromising soundness.

**Lemma 1.** *For any $X \subset \Delta$, the assignment $\mathcal{A} := \{(x_i, a_i) : x_i \in X\}$ can be consistently extended by $\mathcal{B} := \{(x_j, a_j) : x_j \in Y \subset \Delta\}$ if and only if for all strict symmetries $g$ the assignment $\mathcal{A}g$ can be consistently extended by $\mathcal{B}g$.*

Recall the definition of orbit at the end of Section 2. One consequence of the above lemma is to say that a support $\mathcal{J}$ exists for a literal $(x_i, a)$ if and only if there exist images of the support which are support for each literal in the orbit of $(x_i, a)$ under the group of strict symmetries. That is, when symmetries are strict we may reuse symmetric support. Conversely, if $a$ may be pruned from $D_i$, similar domain deletions occur for each element of $(x_i, a)^G$, even when $G$ contains nonstrict symmetries.

**Definition 7.** *A CSP $P$ with symmetry group $G$ is* Sym$(i, j)$-consistent *if for each consistent partial assignment $\mathcal{I}$ of size $i$, for each symmetry $g \in G$ and each set of $j$ variables that do not lie in the* $\mathrm{Var}(\mathcal{I}g)$*, it is possible to find values for those $j$ variables such that the $i + j$ values taken together (the image of the initial $i$ and the new $j$) satisfy all constraints on those $i + j$ variables.*

Note that since all symmetry groups contain the identity permutation this definition encompasses that of standard $(i, j)$-consistency given in Definition 6. Thus, Sym$(i, j)$ consistency is at least as strong as $(i, j)$-consistency.

As an initial example, we illustrate the fact that even Sym(1,0)-consistency is an interesting concept. Consider a simple graph 3-colouring problem on 4 nodes A, B, C, and D, containing all possible edges except between A and D. As a graph colouring problem, we add a not-equals constraint between all pairs of variables except A and D. Initially, each node has domain $[1, 2, 3]$, so the group $G$ of symmetries of the CSP contains a symmetry $f$ which simultaneously interchanges $(B, i)$ with $(C, i)$ for $i \in [1, 2, 3]$. There are also three symmetries $g_1, g_2, g_3 \in G$, such that $g_i$ swaps $(A, i)$ with $(D, i)$ and leaves all other literals fixed. Of course, the overall symmetry group of this CSP contains all combinations of these four symmetries. Suppose now that our first choice during search is to set $(A, 1)$. We make the problem arc consistent, giving domains $A = [1]$, $B = [2, 3]$, $C = [2, 3]$, and $D = [1, 2, 3]$. However, because of the symmetries that interchange A and D, the problem is not Sym(1,0) consistent. The new symmetry group is the stabiliser in $G$ of the positive decision $(A, 1)$. Thus it still contains $f, g_2$ and $g_3$. We can establish Sym(1,0)-consistency by removing 2 and 3 from the domain of D (as $g_2$ maps $(D, 2)$ to $(A, 2)$, which has been deleted and $g_3$ maps $(D, 3)$ to $(A, 3)$). We can see that this is a correct deduction: since A=1, then B and C have to share the values 2, 3, and since D is connected to both, only the value 1 is available for D. Thus we see Sym(1,0) making useful deductions in this simple example. We will explore more deeply the concept of Sym$(i, 0)$-consistency below.

We establish when Sym$(i, j)$-consistency is stronger than that of $(i, j)$-consistency.

**Lemma 2.** *If the symmetry group of a CSP contains only strict symmetries, Sym$(i, j)$-consistency is the same as $(i, j)$-consistency for all $i, j$.*

*Proof.* Let $P$ be a CSP whose symmetry group $G$ contains only strict symmetries, and suppose that $P$ is $(i, j)$-consistent. We show that $P$ is Sym$(i, j)$-consistent. Let $\mathcal{I}$ be a consistent partial assignment of size $i$, and let $g \in G$. Then since $g$ is a strict symmetry, $\mathcal{I}g$ is a consistent partial assignment of size $i$. Thus for any $j$ further variables there exists an extension of $\mathcal{I}g$ to a consistent partial assignment of size $i + j$, by $(i, j)$-consistency. Thus $P$ is Sym$(i, j)$-consistent. The converse is clear, considering the identity element of the symmetry group of the CSP.

Even though the levels of consistency are not different when symmetries are strict, one may use symmetries to speed up the inference process. Suppose we have three variables $x_1, x_2, x_3$, with $\mathrm{Dom}(x_1) = \mathrm{Dom}(x_2) = [2, 3, 4]$ and $\mathrm{Dom}(x_3) = [3]$. Suppose our constraints are $x_1 = x_2$ and $x_2 \leq x_3$, so that our symmetry group interchanges $x_1$ and $x_2$, and fixes $x_3$. Then in enforcing (1,1)-consistency we first prune $4$ from the domain of $x_2$ and then on a second iteration prune it from the domain of $x_1$, whereas when enforcing Sym(1, 1)-consistency we perform both domain deletions at once, as we know that there is a symmetry swapping $x_1$ and $x_2$.

Suppose that $P$ is a CSP whose symmetry group $G$ is not strict. In particular, suppose that $g \in G$ maps a consistent partial assignment $\mathcal{I}$ of size $i$ to a collection of literals $\mathcal{I}g$ which violates at least one constraint, or has two values for the same variable. It is still possible that $P$ is $(i, j)$-consistent for some $j$. However, $P$ is not Sym$(i, j)$-consistent for any $j$, as for any choice of $j$ additional variables there is no way of extending $\mathcal{I}g$ to a consistent partial assignment of size $i + j$. Thus Sym$(i, j)$-consistency is stronger than $(i, j)$-consistency.

We now consider a yet stronger type of symmetric consistency, called *total Sym*$(i, j)$*-consistency*. It differs from Sym$(i, j)$-consistency in its requirements on support: for Sym$(i, j)$-consistency we require that for each image of a partial assignment and each choice of $j$ additional variables there exists a support. Now we reverse some quantifiers and require that there is a support $\mathcal{J}$ for our initial partial assignment $\mathcal{I}$ such that for all symmetries in $G$, the image of the support is a support of the image of $\mathcal{I}$.

**Definition 8.** *A CSP $P$ is* total Sym$(i, j)$-consistent *if given any consistent partial assignment $\mathcal{I}$ of size $i$, and a further $j$-tuple of variables, there exists a $j$-tuple $\mathcal{J}$ of assignments to those variables such that for all $g \in G$ the assignment $\mathcal{I}g \cup \mathcal{J}g$ is consistent.*

Note that if $\mathcal{I} \cup \mathcal{J}$ is contained in a solution then by definition $\mathcal{I}g \cup \mathcal{J}g$ will be consistent for all $g$. Thus enforcing total Sym$(i, j)$-consistency will not jeopardise completeness. Total Sym$(i, j)$-consistency is potentially expensive to maintain: to find support for a consistent $i$-tuple of assignments may involve testing many possible $j$-tuples and symmetries $g$. However, as we will see in Section 4, if the symmetry group $G$ consists only of pure variable symmetries then total Sym$(i, j)$ consistency can be no more expensive than its non-total variant, whilst enforcing a stronger level of consistency.

We finish this section with a discussion of the special case of Sym$(i, j)$-consistency where $j = 0$. A CSP is Sym$(i, 0)$-consistent if whenever $\mathcal{I}$ is a consistent partial assignment of size $i$, the image $\mathcal{I}g$ of $\mathcal{I}$ under any symmetry $g \in G$ is also a consistent partial assignment. Since the symmetry group $G$ partitions the set of all $i$-sets of literals into orbits, each orbit is either entirely consistent or all $i$-tuples in the orbit are expressly prohibited. The reason for our interest is the following key theorem.

**Theorem 1.** *A CSP is both Sym$(i, 0)$-consistent and $(i, j)$-consistent if and only if it is Sym$(i, j)$-consistent.*

*Proof.* Let $P$ be a CSP that is both Sym$(i, 0)$-consistent and $(i, j)$-consistent. Let $\mathcal{I}$ be any consistent partial assignment of size $i$. Then since $P$ is Sym$(i, 0)$-consistent,

the image $\mathcal{I}g$ is consistent, for any $g$ in the symmetry group of $P$. Since $P$ is $(i,j)$-consistent, given the assignment $\mathcal{I}g$ and any set $\mathcal{J}$ of $j$ further variables, we can find a set of values for the variables in $\mathcal{J}$ such that the assignment of all $i+j$ variables is consistent. Thus $P$ is Sym$(i,j)$-consistent.

Conversely, let $P$ be a CSP that is Sym$(i,j)$-consistent. Then it is clearly both $(i,j)$-consistent (consider the identity permutation) and Sym$(i,0)$-consistent, for given a consistent partial assignment $\mathcal{I}$ of size $i$, and any $g$ in the symmetry group $G$ of $L$, the partial assignment $\mathcal{I}g$ can be extended to a consistent partial assignment of size $i+j$, for any choice of $j$ further variables, so must itself be consistent.

Standard (nonsymmetric) $(i,0)$-consistency is vacuous, as there is nothing to test. Sym$(i,0)$-consistency is the same as total Sym$(i,0)$-consistency, as there are no additional assignments to make.

One of the most useful levels of Sym$(i,0)$-consistency is Sym$(1,0)$, which is an intriguing strengthening of forward checking: if a domain value $(x_i,a)$ is deleted at any point, the orbit $(x_i,a)^G$ is computed and all of its images are deleted too, even though they may currently appear to be consistent, as we know that they cannot occur in any solution. Let $G$ be generated by $s$ permutations (a finite group is *generated* by a set of permutations if the group consists of all possible products of the permutations in the set with one another). The cost of computing an orbit $\mathcal{O}$ of $G$ is $O(s|\mathcal{O}|)$ [12]. Since for most practical applications the symmetry group of a CSP can be generated by a very small number of generators (typically 2), the cost of enforcing Sym$(1,0)$-consistency will generally be a small constant multiple of the number of domain deletions that it finds. Thus this is an extremely cheap and effective technique.

Before presenting an algorithm to enforce Sym$(2,1)$-consistency on binary CSPs, we briefly discuss which groups of symmetries should be used at which point in search, and the cost of computing these symmetry groups. At the root, it is clear that the group of symmetries is the full symmetry group $G$ of the CSP (or as much of $G$ as the constraint programmer has been able to identify). At later stages in search, an appropriate choice of symmetries to break is the setwise stabiliser in $G$ of the positive decisions made so far. Computing this setwise stabiliser can be moderately expensive, but the use of setwise stabilisers has been shown in [13] to be an effective technique to reduce search space, so if we wish to use these groups for inference purposes they are available "for free". If the symmetry group of the CSP consists only of value symmetries then at a node $\mathcal{N}$ it suffices to take the pointwise stabiliser of the values seen so far, as in the current partial assignment the setwise stabiliser is equal to the pointwise stabiliser. Let $d := |\Lambda|$. Then, as is shown in [4], after an initial cost of $O(d^5)$ for setup, the running time is $O\tilde{}(d^2)$ at each node, where $O\tilde{}$ is the "soft-O" notation which means that we ignore logarithmic factors. Once again, if one were using a GE-tree based approach during search, these groups would already be computed by the search process, and hence could be used during inference at no extra cost.

## 4 An algorithm to enforce SymPC

Here we present a version of PC2001/3.1 [14] which has been adapted to use symmetry. Recall the definition of path consistency from Section 2. Our algorithm reduces pre-

cisely to PC2001/3.1 when no symmetries are specified, and hence has the best known time complexity in the worst case.

For this section, variables $x_i \in \Delta$ are denoted $i$, and literals $(x_i, a)$ are denoted $(i, a)$. To enforce path consistency it is necessary to assume that there is a constraint between any pair of variables in $\Delta$. If there exist an unconstrained pair of variables, we add the universal relation between them, which permits them to take any pair of values. The relation between variables $i$ and $j$ is denoted $c_{ij}$.

```
SymPC2001/3.1(P)
1  SymInitialise(P);
2  while Q ≠ ∅ do
3     Select and delete any ((i, a), j) from Q;
4     SymRevisePath((i, a), j, Q);
```

```
SymInitialise(P)
1  for all (i, a), (j, b) ∈ χ and all k ∈ Δ do
2     Last((i, a), (j, b), k):= false;
3  for all i, j, k ∈ Δ with i ≠ j ≠ k ≠ i do
4     for all a ∈ D_i, b ∈ D_j do
5        if (a, b) ∈ c_ij and Last((i, a), (j, b), k) = false then
6           if there is no v ∈ D_k s.t. (a, v) ∈ c_ij ∧ (v, b) ∈ c_kj then
7              for all g ∈ G do
8                 (i', a') := (i, a)g; (j', b') := (j, b)g;
9                 remove (a', b') from c_i'j' and (b', a') from c_j'i';
10                Q := Q ∪ {((i', a'), j'), ((j', b'), i')};
11          else
12             Let v ∈ D_k be the first value satisfying (a, v) ∈ c_ik ∧ (v, b) ∈ c_kj
13             Last((i, a), (j, b), k) := (v,true);
14             for all g ∈ G, g ≠ 1 do
15                if (i, a)g, (j, b)g, (k, v)g is consistent and
                        Last((i, a)g, (j, b)g, Var((k, v)g)) is false then
16                   (k', v') := (k, v)g;
17                   if G contains only pure variable symmetries then
18                      Last((i, a)g, (j, b)g, k') := (v',true);
19                   else
20                      Last((i, a)g, (j, b)g, k') := (v',false);
```

The path consistency algorithm, which we have named SymPC2001/3.1, is in two parts: initialisation and propagation. The initialisation function is SymInitialize,which seeks a first support for each ordered pair of literals $((i, a), (j, b))$ and each third variable $k$. In line 6, if we cannot find support for a pair $((i, a), (j, b))$, then $(a, b)$ is removed from $c_{ij}$, and we also remove all of its images from the corresponding constraints. For the sake of clarity, we have written line 7 (and later line 14) to loop through all group elements, in fact we will loop only over the distinct images of $(i, a)$ and $(j, b)$. For each removal we enqueue in line 10 an image of $((i, a), j)$ and $((j, b), i)$ onto $Q$. If support can be found, then in line 13 we store this support in Last$((i, a), (j, b), k)$, along with a boolean value `true` to indicate that the support was found directly. If we find support for $((i, a), (j, b), k)$ then in lines 14 to 19 we reuse all of its images, but if $G$ contains any symmetries other than pure variable symmetries we set a boolean to

`false` to indicate that the value $v'$ that we are storing as support may not be the minimal possible support in $D_{k'}$. If $G$ consists only of pure variable symmetries then $v'$ is in fact minimal since $(k,v)g = (k',v)$, so the boolean value is set to `true`.

The second function is SYMREVISEPATH, which takes as input an element $((i,a),j)$ from $Q$, and checks every variable $k \in \Delta \setminus \{i,j\}$ to see if any tuple in $c_{ik}$ is affected by the modification of $c_{ij}$. There are two possibilities for the search for support. If this is the first time that $c_{ij}$ has been examined with respect to $k$, and the previous support was the image under $G$ of some other support and hence might not be minimal in $D_k$, then in line 6 SYMREVISEPATH tries to find a support from scratch. Otherwise, the boolean in Last$((i,a),(j,b),k)$ is `true` and SYMREVISEPATH starts its search in line 7 from the previous bookmarked value. If support cannot be found then in lines 10 to 14 we not only remove $(a,b)$ from $c_{ij}$ but also all of its images from the corresponding constraints. If support can be found then we store it.

```
SYMREVISEPATH((i,a),j,Q)
1  for all k with i ≠ k ≠ j do
2      for all b ∈ D_k s.t. (a,b) ∈ C_ik do
3          (v,x) := Last((i,a),(k,b),j);
4          while v ≠ NIL ∧ ((a,v) ∉ c_ij ∨ (v,b) ∉ c_jk) do
5              if x = false then
6                  v := min{D_j}; x :=true;
7              else
8                  v := succ(v,D_j);
9          if v = NIL then
10             for all g ∈ G do
11                 (i',a') := (i,a)g; (k',b') := (k,b)g;
12                 Remove (a',b') from c_i'k' and (b',a') from c_k'i';
13                 Q := Q ∪ {((i',a'),k'),((k',b'),i')};
14         elif G consists only of pure variable symmetries then
15             for all g ∈ G do
16                 (v',x):= Last((i,a)g,(k,b)g,Var((j,v)g));
17                 if v' < v then
18                     Last((i,a)g,(k,b)g,Var((j,v)g)) := (v, true);
19                 else
20                     Last((i,a),(k,b),j) := (v', true);
21         else
22             Last((i,a),(k,b),j) := (v,true);
```

If $G$ consists only of pure variable symmetries, then the algorithm has been modified to enforce total symmetric path consistency. This is because of the following observation: suppose that we are considering the pair of assignments $((i,a),(j,b))$ and the variable $k$, and suppose that we have found $v$ to be the smallest element of $D_k$ such that $(i,a),(j,b),(k,v)$ is consistent. Then since no element of $G$ affects the values in any literal, for any $g \in G$ no element of $D_{\mathrm{Var}((k,c)g)}$ that comes before $v$ can be used as support when enforcing total SymPC. To see this note that if $c < v$ then $((i,a)g,(j,b)g,(k,c)g)g^{-1}$ is inconsistent. Therefore in line 18 of the initialisation function we reuse the image of a support without needing to mark it as reused, and in lines 15 to 20 of SYMREVISEPATH we ensure that the supports agree for a whole orbit of ordered pairs of literals.

## 5 Symmetrised singleton consistencies

A type of consistency which has been attracting much attention recently is that of *singleton* consistency. Like the notion of symmetrised consistency, it is different from other consistency techniques because of its meta character: it is not a standalone technique such as AC or PC, but improves the pruning techniques of all of them.

Let $P = (\Delta, \mathcal{D}, \mathcal{C})$ be a CSP, and let $x_i \in \Delta$ and $a \in D_i$. By $P|_{(x_i,a)}$ we denote the CSP obtained from $P$ by setting $(x_i = a)$ and deleting all other values from the domain of $x_i$: we call this the CSP *induced* from $P$ with respect to $(x_i, a)$. Singleton consistency extends any consistency level $X$ by requiring that for all $x_i \in \Delta$ and all $a \in D_i$, the problem $P|_{(x_i,a)}$ is $X$-consistent. For instance, a CSP $P = (\Delta, \mathcal{D}, \mathcal{C})$ is singleton arc consistent (SAC) if for all $x_i \in \Delta$ and all $a \in D_i$, the CSP $P|_{(x_i,a)}$ is arc consistent.

One advantage of singleton consistency is that enforcing it does not change the constraints of a problem – no matter what level of consistency $X$ we choose, singleton $X$ consistency will result in domain deletions.

Singleton consistency is a good candidate for symmetrisation: the basic notion to be applied is that whenever we discover a domain deletion, we can delete an entire orbit of literals without needing to recheck the $X$-consistency of each of the corresponding induced CSPs. Thus, as in Section 3, at a cost $O(s|(x_i, a)^G|)$, where $G$ is generated by $s$ permutations and $(x_i, a)$ is a literal to be deleted, we can avoid $O(|(x_i, a)^G|)$ calls to enforce $X$-consistency on an entire induced CSP. If some symmetries are nonstrict then we may be able to delete literals that would not be deleted by singleton $X$-consistency alone, thus symmetrised singleton $X$-consistency is more pruningful than its unsymmetrised variant.

A second potential gain, which we can make at the cost of compromising completeness, is to only test the $X$-consistency of $P|_{(x_i,a)}$ when $(x_i, a)$ is the orbit representative for $(x_i, a)^G$. If all symmetries of the problem are strict then this approach is clearly complete as well as sound. In the pseudocode below we do *not* take this approach, as in our experiments in Section 6 we wished to preserve completeness, however it would only require a minor adjustment to the algorithm.

**Definition 9.** *Let $P := (\Delta, \mathcal{D}, \mathcal{C})$ be a CSP with symmetry group $G$ and let $X$ be a level of consistency. Then $P$ is* symmetrised singleton $X$-consistent *(written SymSingletonX-consistent) if for all $x_i \in \Delta$, for all $a \in D_i$ and for all $g \in G$, $P|_{(x_i,a)g}$ is $X$-consistent. We say that $P$ is* $X$+SymSingletonX-consistent *if $P$ is both $X$ consistent and SymSingletonX consistent.*

It is clear that, provided enforcing $X$-consistency is sound, so is enforcing symmetrised singleton $X$-consistency.

There are many different algorithms for enforcing singleton consistency. Most of these are for enforcing SAC, where Dubruyne and Bessière initially proposed an algorithm that is similar in style to AC1 [15]. This was upgraded in the style of AC4 by Barták and Erben [16], and then further improved to give 'SAC-Opt' by Bessiére and Dubruyne, which has optimal time complexity [17]. It is this latter, optimal algorithm which we symmetrise: we present an algorithm X+SymSingletonX, which enforces

$X$-consistency and symmetrised singleton $X$-consistency for any level $X$ of consistency. We have chosen to present code which enforces $X$-consistency on $P$ because in our experiments the level $X$ of consistency which we will test is the default level of consistency in ECL$^i$PS$^e$[18], which is enforced automatically. If the level $X$ of consistency is arc consistency, and the group of symmetries is trivial, then our algorithm reduces to SAC-Opt, and hence is time optimal in the worst case.

In X+SYMSINGLETONX we denote $D_i := \mathrm{Dom}(x_i)$, as normal, and $\chi$ denotes the set of all literals $(x_i, a)$ of $P$; by $\chi_{ia}$ we denote the set of all literals of the induced problem $P_{ia}$.

```
X+SymSingletonX(P)
1  PropagX(P, ∅);
2  for all (i, a) ∈ χ do                    /* initiation phase*/
3      P_ia := P;
4      if not PropagX(P_ia, {(i, b) : b ∈ D_i \ {a}}) then      /* set i = a in subproblem */
5          if PropagX(P, (i, a)^G) then
6              for all (j, b) ∈ (i, a)^G do
7                  for all P_kc such that (j, b) ∈ χ_kc do
8                      Q_kc := Q_kc ∪ {(j, b)};
9                      PendingList:= PendingList ∪{(k, c)};
10         else return false;
11 while PendingList ≠ ∅ do                 /* propagation phase */
12     pop (i, a) from PendingList;
13     if PropagX(P_ia, Q_ia) then Q_ia := ∅;
14     else
15         if not PropagX(P, (i, a)^G) then
16             return false;
17         for all (j, b) ∈ (i, a)^G do
18             for all P_kc such that (j, b) ∈ χ_kc do
19                 Q_kc := Q_kc ∪ {(j, b)};
20                 PendingList:= PendingList ∪{(k, c)};
21 return true;
```

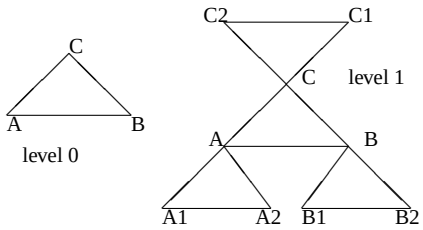
The function takes as input a CSP $P$, and runs in two phases - initialisation and propagation. In line 3 we initialise $P_{ia}$, where $(i, a) \in \chi$, to be a copy of $P$. If $X$ is a level of consistency that only results in domain deletions (rather than the posting of non-unary constraints) then in $P_{ia}$ we record only the domains and data structures of $P$, rather than all of the constraints, as we re-use the constraints in $P$. In line 4 we call the algorithm PROPAGX, which takes as input a CSP and a set of domain deletions (in this case $i$ can no longer take any value in $D_i \setminus \{a\}$), and propagates the effect of these deletions under consistency level $X$. Any algorithm to enforce $X$ can be used at this point. We assume that PROPAGX returns `false` if and only if propagating deletions in $P_{ia}$ results in an empty domain, and returns `true` otherwise. We also assume that PROPAGX will modify $P_{ia}$ to make it $X$-consistent, either by pruning values, or by posting additional constraints, or both. If a wipeout occurs in line 4 then we know that every image of $(i, a)$ under $G$ is not part of any solution to $P$, so in line 5 we compute all images of $(i, a)$, delete them from our set $\chi$ of literals (and hence also from the corresponding domains), and then check that $P$ can still be made $X$-consistent after these domain deletions. If it can be, then for each deleted literal $(j, b)$, and each already

created restricted problem $P_{kc}$ with $(j, b)$ in its set $\chi_{kc}$ of literals, in line 8 we add $(j, b)$ to the list of future domain deletions to be made in $P_{kc}$, and in line 9 we add $(k, c)$ to the list PendingList of subproblems to be processed. If $P$ cannot be made $X$-consistent after deleting $(i, a)^G$ then $P$ is unsatisfiable and in line 10 we return `false`.

Once the initialisation phase has finished, we know that PendingList contains all values $(i, a)$ for which some symmetrised singleton $X$ inconsistent value removals (contained in $Q_{ia}$) have not yet been propagated in $P_{ia}$. The loop in line 11 propagates each of these removals, along with any others which are forced by this propagation. In line 13, if we can successfully delete all of these values then we clear the $Q_{ia}$ entry and move on. When propagation fails, this means that $(i, a)$ is symmetrised singleton $X$ inconsistent, and that the same will hold for all of its images under $G$. Therefore in line 15 we delete all images of $(i, a)$ from $P$, and check whether the resulting CSP can still be made $X$-consistent. If it cannot, then $P$ has no solutions. If it can, then in lines 17-20 we update our list of subproblems which require further propagation of domain deletions.

## 6 Experiments

To test our SYMPC2001/3.1 algorithm we used a collection of graph colouring problems. We define an infinite family of graphs based around triangles, as follows. Level $0$ consists of a single triangle, as in the figure below. To make level $1$ from level $0$ we add a triangle to each vertex of the original triangle, giving a graph with $9$ vertices, shown below. To make level 2 from level one we add a triangle to each vertex of valency $2$ in the level $1$ graph. This process can clearly be carried on indefinitely, adding a triangle to each vertex of valency $2$ on level $i$ to produce level $i + 1$.




These graphs have a large symmetry group. We can freely permute the three vertices of the central triangle, giving an automorphism group of size six. For graphs of level at least 1, after determining the images of the central three vertices, we can still swap vertices A1 and A1, or vertices B1 and B2, or vertices C1 and C2, or any combination of these swaps. Thus there are a total of $2^3 \times 6 = 48$ symmetries of the level $1$ graph. When going from level $i$ to level $i + 1$ we have an additional $3 \times 2^{i-1}$ triangles, each of which can be independently flipped over, giving an extra $2^{3\times 2^{i-1}}$ symmetries. Thus the symmetry group of the graph at level $i$ contains $6 \times \prod_{j=1}^{i} 2^{3\times 2^{j-1}}$ symmetries. Specifically, the level 1 graph has 48 symmetries, the level 2 graph 3072, level 3 about 12.3 million and level 4 about $2 \times 10^{14}$. These act as pure variable symmetries on the colouring problem.

Given this basic setup, we can now assign domains to each node to produce problems which will respond differently to path consistency. We have three main families of

problems. In the first, the vertices of valency 2 have domains $[1, 2]$ and all other vertices have domains $[1, 2, 3, 4]$. This problem is unsatisfiable, a fact which will be discovered by path consistency and SymPC. We use the full group of automorphisms of the graph, as well as the pure value symmetry which interchanges 1 and 2 and the pure value symmetry which interchanges 3 and 4. In fact, since the problem is unsatisfiable, we could have used any symmetries whatsoever, as there are no solutions to preserve. However, it is reasonable to assume that the constraint programmer would not know in advance that the problem was unsatisfiable, and hence would only use the obvious graph and colour symmetries. Our results are summarised in Table 1.

**Table 1.** Results for SYMPC2001/3.1 with the first family of problem

| level | Full Syms | | Graph Syms | | No Syms | |
|---|---|---|---|---|---|---|
| | checks | time | checks | time | checks | time |
| 1 | 212 | 19 | 245 | 5 | 7627 | 49 |
| 2 | 186201 | 1160 | 177544 | 994 | 165303 | 1128 |
| 3 | 2604343 | 19802 | 2427594 | 16794 | 3427582 | 23963 |
| 4 | 26803564 | 424553 | 24559120 | 386408 | 37290536 | 714441 |

For our second family of experiments, with results given in Table 2, we give vertex $A$ domain $[1, 2, 3, 4, 5]$, vertices of valency 2 domain $[1, 2]$, and all other vertices domain $[1, 2, 3, 4]$. Here we lose a factor of three in the number of graph automorphisms, as we can no longer rotate the central triangle. The value symmetries remain unchanged from problem 1. Path consistency and SymPC will deduce that only the value 5 for $A$ is consistent, but the problem overall is satisfiable.

**Table 2.** Results for SYMPC2001/3.1 with the second family of problem

| level | Full Syms | | Graph Syms | | No Syms | |
|---|---|---|---|---|---|---|
| | checks | time | checks | time | checks | time |
| 1 | 12791 | 114 | 12930 | 105 | 19345 | 157 |
| 2 | 253719 | 5480 | 250597 | 1672 | 369540 | 2135 |
| 3 | 2972582 | 33732 | 2851446 | 30186 | 4248742 | 40004 |
| 4 | 28355217 | 495557 | 26803731 | 479862 | 40130411 | 712005 |

For our third family of experiments, with results described in Table 3, we give all vertices domain $[1, 2, 3]$, so that neither PC nor SymPC make any deductions at all.

**Table 3.** Results SYMPC2001/3.1 with the third family of problem

| level | Full Syms | | Graph Syms | | No Syms | |
|---|---|---|---|---|---|---|
| | checks | time | checks | time | checks | time |
| 1 | 4668 | 80 | 5093 | 65 | 14064 | 90 |
| 2 | 72559 | 2146 | 74970 | 1893 | 218736 | 2576 |
| 3 | 768740 | 53292 | 776944 | 53952 | 2315472 | 30090 |
| 4 | 7013763 | 638446 | 7035185 | 574108 | 21096720 | 763494 |

The vertex numbering follows in turn the pre-order traversal of the binary trees rooted at each of the vertices of the central triangle. All times are in milliseconds, on a Pentium M 2.1GHz. The implementation differed from the pseudocode in the following ways. In the loops at lines 7 and 14 of SymInitialize we use the orbit algorithm mentioned above to find all distinct images of $(i', a'), (j', b')$ (resp. $(i', a'), (j', b'), (k', v')$) without actually looping over all of $G$. We do the same at lines 10 and 15 of SymRevisePath. This is essential for handling larger groups. If the last allowed pair is removed from a constraint, then we terminate the calculation, reporting the problem unsatisfiable. We take advantage of the fact that the given symmetries of this problem are, in fact, strict symmetries to avoid the consistency check in line 15 of SymInitialize.

We have also succesfully implemented our X+SYMSINGLETONX algorithm. The generality of the algorithm meant that by implementing in the ECLIPSE constraint logic programming system [18], the base level of consistency X was that established by ECLIPSE. We do not know the internal details of the system, so we do not know what X is, but nevertheless, our algorithm is general enough to implement X+SYMSINGLETONX. For group-theoretic calculations, we used the GAP-ECLIPSE interface [19, 20]. We use GAP to obtain orbits of singleton assignments. To evaluate our implementation, we compared X+SYMSINGLETONX against X+SINGLETONX. For the latter, the algorithm we presented above specialised to the trivial group formed the comparison.

As a test, we used constraint problems based on the same triangle graph schema used above, with different constraints on triangles. For example, suppose each variable has domain $1 \ldots 4$, and we constrain the sum of each triangle. If the outermost triangles have to sum to either 3 or 12 (and not any value inbetween), ECLIPSE performs no propagation – presumably because it is using bounds consistency. Setting a variable in a triangle and propagating in ECLIPSE results in failure if it is set to 2 or 3. Singleton consistency therefore removes those values, leaving domains as $\{1, 4\}$. We performed a test in which the outermost triangles were restricted to sum to $\{3, 12\}$, while all inner triangles had to sum to a value in $\{3, 4, 12\}$. The result is that the domains of variables in outer triangles become $\{1, 4\}$ and other variables become $\{1, 2, 4\}$ since the value 3 is impossible. We obtain the same deductions with both symmetric and asymmetric versions of singleton consistency, and the computation times were comparable. An interesting feature is that the high power of singleton consistency means that, in practice, the initialisation phase does all the work, and we did not see propagation happening after that. Under any sensible X, trying every possible singleton is so powerful that it is hard to construct examples where any domain deletions remain after initialisation.

## 7 Conclusions and Future Work

We have introduced a new research area: symmetrising levels of consistency to produce stronger forms of consistency and more efficient mechanisms for establishing them.

Many forms of consistency can be adapted to take advantage of symmetries of a CSP. We have focussed on two particular levels of consistency, and given algorithms and implementations of symmetrised versions of them. In the case of path consistency, experiments showed that we could improve runtime performance, despite having to maintain data structures representing groups of size, for example $10^14$. We have shown that if the CSP has nonstrict symmetries then these new levels of consistency do not coincide with any previously defined levels of consistency. We have discussed how, even

in the case of strict symmetries, it is possible to take advantage of symmetry to improve the performance of consistency algorithms. For high levels of consistency the cost of the added group theoretic machinery is negligible compared to the cost of maintaining consistency, although there is a need for optimisations to avoid repeating work.

## Acknowledgements

We thank our reviewers for their helpful comments. Our research is supported by a Royal Society of Edinburgh SEELLD Support fellowship, and by EPSRC grant numbers GR/R29666 & GR/S30580. Special thanks go to Iain McDonald.

## References

1. P. Flener, A. M. Frisch, B. Hnich, Z. Kızıltan, I. Miguel, J. Pearson, & T. Walsh. Breaking row and column symmetries in matrix models. *Proc. CP 2002*, pages 462–476. Springer, 2002.
2. R. Backofen and S. Will. Excluding symmetries in constraint-based search. *Proc. CP-99*, pages 73–87. Springer, 1999.
3. C.A. Brown, L. Finkelstein, and P.W. Purdom Jr. Backtrack searching in the presence of symmetry. *Nordic Journal of Computing*, 3(3):203–219, 1996.
4. C.M. Roney-Dougal, I.P. Gent, T. Kelsey, and S.A. Linton. Tractable symmetry breaking using restricted search trees. *Proc. ECAI'04*, 2004.
5. A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.
6. C. Bessière and J-C. Régin. Refining the basic constraint propagation algorithm. *Proc. IJCAI'01*, 2001.
7. Y. Zhang and R.H.C. Yap. Making AC-3 an optimal algorithm. *Proc. IJCAI'01*, 2001.
8. I.P. Gent and I. McDonald. Symmetry and propagation: Refining an AC algorithm. *Proc. SymCon03*, 2003.
9. T.Kelsey, S.A.Linton, and C.M.Roney-Dougal. New developments in symmetry breaking in search using computational group theory. In *Proc. AISC'04*, 2004.
10. E.C. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29(1):24–32, 1982.
11. E.C. Freuder. A sufficient condition for backtrack-bounded search. *Journal of the ACM*, 37(4):755–761, 1985.
12. A. Seress. *Permutation group algorithms*. Number 152 in Cambridge tracts in mathematics. Cambridge University Press, 2002.
13. J.-F. Puget. Symmetry breaking using stabilizers. In *Proc. CP-03*, 2003.
14. C. Bessière, J-C. Régin, R.H.C. Yap, and Y. Zhang. An optimal coarse-grained arc consistency algorithm. *Artificial Intelligence*, 165:165–185, 2005.
15. R. Debruyne and C. Bessière. Some practicable filtering techniques for the constraint satisfaction problem. *Proc. IJCAI'97*, pages 412–417, 1997.
16. R. Barták and R. Erben. A new algorithm for singleton arc consistency. *Proc. FLAIRS 2004*, 2004.
17. C. Bessière and R. Debruyne. Optimal and suboptimal singleton arc consistency algorithms. *Proc. IJCAI'05*, 2005.
18. M. G. Wallace, S. Novello, and J. Schimpf. ECLiPSe : A platform for constraint logic programming. *ICL Systems Journal*, 12(1):159–200, May 1997.
19. I.P. Gent, W. Harvey, and T. Kelsey. Groups and constraints: Symmetry breaking during search. *Proc. CP 2002*, pages 415–430. Springer, 2002.
20. I.P. Gent, W. Harvey, T. Kelsey, and S.A. Linton. Generic SBDD using computational group theory. *Proc. CP 2003*, pages 333–347. Springer, 2003.