Proceedings of SymCon'10, the Tenth International Workshop on Symmetry in Constraint Satisfaction Problems

A workshop at the 16th International Conference on Principles and Practice of Constraint Programming (CP'10)

Monday 6 September 2010

St Andrews, Scotland, UK

Edited by:

Pierre Flener and Justin Pearson
Uppsala University
Department of Information Technology
Box 337
751 05 Uppsala
Sweden

Preface

A *symmetry* is a transformation that preserves solutions that are considered equivalent. For instance, rotating a chess board 180 degrees gives a board that is indistinguishable from the original board. In the presence of symmetry, a constraint solver may waste a lot of time considering symmetric but equivalent assignments or partial assignments. Hence, dealing with symmetry is often crucial for solving such combinatorial problems efficiently.

This is the 10th workshop of the very successful *SymCon* series of workshops on symmetry in constraint satisfaction problems, founded by us in 2001 (the series homepage is http://www.it.uu.se/research/group/astra/SymCon/).

All submitted papers were peer-reviewed and those that make a worthwhile contribution were accepted for presentation at the workshop. We hope that this snapshot of current research will act as a catalyst for further research. These proceedings are informal and are also on-line (at http://www.it.uu.se/research/group/astra/SymCon10/).

We thank Pedro Meseguer, the workshop chair of CP'10, for a smooth organisation of local matters together with the CP'10 local chairs. Many thanks also to the SymCon'10 programme committee (listed below), for precious help with the review process. Finally, we express our gratitude to Pascal Van Hentenryck, for agreeing to give the invited talk.

Uppsala, 20 August 2010 — Pierre Flener and Justin Pearson

Programme Committee

Table of Contents

A Partial Taxonomy of Substitutability and Interchangeability

Shant Karakashian[1], Robert Woodward[1], Berthe Y. Choueiry[1], Steven D. Prestwich[2] and Eugene C. Freuder[2]

[1] Constraint Systems Laboratory, University of Nebraska-Lincoln, USA
{shantk,rwoodwar,choueiry}@cse.unl.edu

[2] Cork Constraint Computation Centre, Department of Computer Science, University College Cork, Ireland {s.prestwich,e.freuder}@4c.ucc.ie

Abstract. Substitutability, interchangeability and related concepts in Constraint Programming were introduced approximately twenty years ago and have given rise to considerable subsequent research. We survey this work, classify, and relate the different concepts, and indicate directions for future work, in particular with respect to making connections with research into symmetry breaking. This paper is a condensed version of a larger work in progress.

1 Introduction

Many important problems in computer science, engineering and management can be formulated as Constraint Satisfaction Problems (CSPs). A CSP is a triple (V, D, C) where V is a set of variables, D the set of their domain values, and C a set of constraints on the variables that specify the permitted or forbidden combinations of value assignment to variables. A solution to a CSP is an assignment of values to all variables such that all constraints are satisfied. CSPs are usually solved by interleaving backtrack search with some form of constraint propagation, for example forward checking or arc consistency.

Constraint problems often exhibit symmetries. A great deal of research has been devoted to *symmetry breaking* techniques in order to reduce the size of the search space [Various, 1991 present]. The earliest works on symmetry breaking include [Glaisher, 1874; Brown *et al.*, 1988]. In this paper we will not survey the large literature on symmetry breaking, but a recent survey can be found in [Gent *et al.*, 2006].

Interchangeability, proposed in a seminal paper by Freuder [1991], is one of the first forms of symmetry identified for CSPs. Importantly, it is also the first method proposed for *detecting* symmetry as opposed to having a constraint programmer manually specify it. Although there has been since then a steady flow of research papers developing this concept in both theory and practice, it has been relatively neglected compared to other forms of symmetry. This situation is surprising: While in its basic form, interchangeability is a special case of value symmetry, its various extensions (already proposed in the 1991 paper) make

it a more general concept than is sometimes perceived, and anticipate some subsequent developments in symmetry definition and breaking. The comparison with the various types and definitions of symmetry [Benhamou, 1994; Cohen *et al.*, 2006] will be discussed in the longer version of this paper.

The goal of our endeavor is to analyze the research conducted so far on interchangeability, relate it to symmetry, and identify opportunities for future research. This paper is a work in progress and a first step towards our goal. Our survey is partial and far from complete and we welcome the feedback of the readers and workshop participants.

The advantages of detecting and exploiting interchangeability have been established on random problems, benchmarks, and real-world applications. In backtrack search, the advantages are mainly the reduction of the search space and the search effort [1], and the attainment of multiple solutions by bundling. In local search, interchangeability is used to locally repair partial solutions [Petcu and Faltings, 2003]. Real-world applications include nurse scheduling [Weil and Heus, 1998] and resource allocation in hospitals [Choueiry *et al.*, 1995].

This paper is structured as follows. In Section 2, we give the definitions of the basic interchangeability concepts and relate them to each other. In Section 3, we discuss forms of conditional interchangeability. In Section 4, we discuss other forms of interchangeability that have appeared in the literature. In Section 5, we relate the various forms of interchangeability. Finally, in Section 6, we list topics that we plan to cover more fully in the expanded version of this paper.

2 Basic Interchangeability Concepts

In this section, we review the various forms of interchangeability originally introduced in [Freuder, 1991]. We also include a few new interchangeability concepts that directly relate to the original ones. Full interchangeability, the most basic form of interchangeability, is defined as follows.

Full interchangeability (FI) [Freuder, 1991] A value a for variable v is fully interchangeable with value b iff every solution in which $v = a$ remains a solution when b is substituted for a and vice-versa.

If two values are interchangeable then one of them can be removed from the domain, reducing the size of the problem; alternatively they can be bundled together in a Cartesian product representation of solutions. Figures 1, 2 and 3 show examples of two values a and b that are FI (see below for definition of 3-I and NSub). In our figures, a small solid circle denotes a value in the domain of the variable represented by the outline circle, and edges link consistent tuples.

Notice that FI is defined 'at the solution level', which means that in order to find all FI values for a given variable, one must account for *all* constraints and may have to compute all solutions. Thus, FI is a *global* property. In [1994], Benhamou defines the equivalent notion of 'value symmetry in all solutions' as

[1] Most importantly, by factoring out no-goods [Choueiry and Davis, 2002].

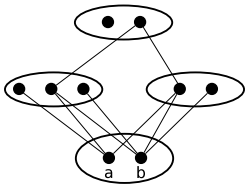


Fig. 1. FI: a and b are FI but not 3-I or NSub.

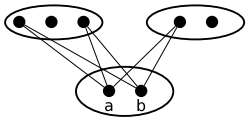


Fig. 2. NI: a and b are NI.

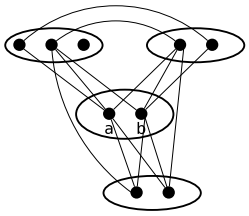


Fig. 3. KI: a and b are 3-I but not NI.

semantic symmetry. Hence, the terms 'semantic' and 'global' are equivalent. Because the detection of global forms of interchangeability is likely to be intractable, Freuder introduced *local* variants, which account only for the constraints defined on a variable, that is, the neighborhood of the variable. In [1994], Benhamou calls such relations *syntactic* symmetries. Section 2.1 discusses local interchangeability. Further, interchangeability is an equivalence relation on the domain of the variable: interchangeable values are equivalent. Such equivalences may be rare in practice. To remedy this situation, Freuder proposed various extensions to the basic concept, which are discussed in Sections 2.2 and 2.3.

In summary, one may think of interchangeability as a core concept characterized as a relation between two values either at the solution level (i.e., global or semantic) or in the neighborhood of the variable (i.e., local or syntactic). Also, the concepts may require that interchangeable values be equivalent (i.e., strong), or not 'perfectly' so (i.e., weak or approximate).

When comparing two forms of interchangeability X and Y, we say that $X \rightarrow Y$ iff any two values a and b that are related by X are also related by Y but the converse does not necessarily hold[2]*,* regardless of whether Y is derived from X by relaxing the conditions of X (i.e., Y is weaker than X) or by 'moving' from the local level to the global level (i.e., syntactic to semantic). Note that when $X \rightarrow Y$, Y leads to greater problem reduction than X.

2.1 Local forms of interchangeability

In general, the identification of a local interchangeability is tractable because it focuses on the neighborhood of the variable. Also, a given local form interchangeability usually implies the corresponding global one.

Neighborhood interchangeability (NI) [Freuder, 1991] A value a for variable v is neighborhood interchangeable with value b iff for every constraint on v, the values compatible with $v = a$ are exactly those compatible with $v = b$. Values a and b are NI in Figure 2 but not in Figures 1 or 3.

Neighborhood interchangeable values for a given variable can be detected by comparing the values in the variable's domain for consistency to all variable-value pairs in the variable's neighborhood and drawing a discrimination tree

[2] a and b are two values or two partial assignments over the same variables.

[Freuder, 1991]. At the end of the process, the leaves of the discrimination tree are annotated with the equivalence NI values for the variable. The complexity of this process is $\mathcal{O}(n^2d^2)$, where n is the number of variables and d is the maximum domain size. Alternatively, one can build a refutation tree, which proceeds by splitting the domain of the variable [Likitvivatanavong and Yap, 2008]. The lower bound of the worst-case complexity of the refutation tree is smaller than that of the discrimination tree. However, it is not clear whether the difference is meaningful in practice. Further, the discrimination tree can be directly used to implement forward checking at no additional cost [Beckwith *et al.*, 2001], but it is not clear yet whether or not the same can be done with the refutation tree.

For non-binary constraints, neighborhood interchangeable values can be detected by constructing non-binary discrimination trees for each variable [Lal *et al.*, 2005]. As described in [Lal *et al.*, 2005], the process also allows the use of forward checking during search. The complexity to build a non-binary discrimination tree for a single variable is $\mathcal{O}(n\ deg\, a^{k+1}(1-t))$, where n is the number of variables, deg is the maximum degree of a variable, a is the maximum domain size, and t is the tightness of the constraints, defined as the ratio of the number of forbidden tuples over the number of all possible tuples.

K-interchangeability (KI) [Freuder, 1991] For $k \geq 2$, two values, a and b for a CSP variable X, are *k-interchangeable* iff a and b are fully interchangeable in any subproblem of the CSP induced by X and $(k-1)$ other variables. Values a and b are 3-I in Figures 2 and 3 but not in Figure 1.

K-interchangeable values can be identified by a modification of the discrimination-tree algorithm for NI. The complexity of the process is $\mathcal{O}(n^k d^k)$ [Freuder, 1991].

Theorem 1. NI $\rightarrow$ KI $\rightarrow$ FI, see [Freuder, 1991].

For $2 < i < j < |V|$, i-interchangeability is a sufficient but not necessary condition for j-interchangeability. NI is 2-interchangeability and FI is $|V|$-interchangeability. Hence, NI $\rightarrow$ KI $\rightarrow$ FI. a and b in Figure 1 are FI but not 3-I, and in Figure 3 they are 3-I but not NI.

2.2 Extended interchangeability: Weak forms

Below, we discuss three weak forms of interchangeability introduced in [Freuder, 1991] (i.e., subproblem interchangeability, partial interchangeability, and substitutability) and a number of other related concepts.

Subproblem interchangeability (SPrI) [Freuder, 1991] Two values are *subproblem interchangeable*, with respect to a subset of variables S, iff they are fully interchangeable with regards to the solutions of the subproblem of the CSP induced by S.

Partial interchangeability (PI) [Freuder, 1991] Two values are *partially interchangeable* with respect to a subset S of variables, iff any solution involving one implies a solution involving the other with possibly different values for variables in S. In Figure 4, a and b are PI *wrt* S, shown with the dotted line.

Theorem 2. FI → PI.

If a and b are FI, they are by definition PI with respect to any subset of V. In Figure 4, a and b are PI *wrt* to the subset S but not FI.

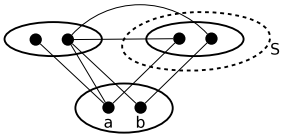


Fig. 4. PI: a and b are PI *wrt* S but not Sub, FI, CtxDepI, NTI, or NPI *wrt* any subset.

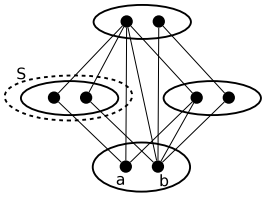


Fig. 5. PI: a and b are PI *wrt* S but not Sub or SPrI *wrt* any subset of variables.

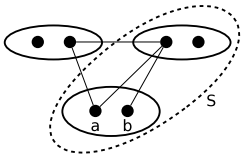


Fig. 6. SPrI: a and b are SPrI *wrt* S but not PI *wrt* any subset of variables.

Theorem 3. SPrI and PI are not comparable[3].

In Figure 5, a and b are PI but not SPrI. In Figure 6, a and b are SPrI but not PI.

Substitutability (Sub) [Freuder, 1991] For two values a and b for variable v, a is substitutable for b iff every solution in which $v = b$ remains a solution when b is replaced by a but not necessarily vice-versa. Figure 7 shows an example.

Note that the concept of substitutability is related to that of dominance [Bellicha *et al.*, 1994], which is used in the literature on symmetry breaking.

Theorem 4. FI → Sub.

If a and b are FI, they are by definition mutually substitutable. In Figure 7, a is substitutable for b, but a and b are not FI.

Again, because substitutable values are expensive to compute, neighborhood substitutability (NSub) (with the obvious definition) is computationally advantageous. In Figure 8, a is NSub for b. In [1994], Bellicha et al. propose NS-Closure, an algorithm to enforce NSub. It removes all of the neighborhood substitutable values from the network. It operates by examining every pair of values (a, b) in a variable's domain, trying to find a *splitter* for the pair. A splitter for (a, b) is a value in the neighborhood of the variable that supports a but not b. If (a, b) does not have a splitter, then a can be removed from the domain. The time complexity of the algorithm is $\mathcal{O}(md^3)$, where m is the number of constraints and d is the maximum domain size. The space complexity of storing the splitters is $\mathcal{O}(nd^2)$, where n is the number of variables.

Theorem 5. NI → NSub → Sub; FI and NSub are not comparable.

[3] This theorem corrects Theorem 5 of [Freuder, 1991].

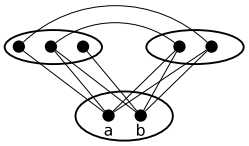


Fig. 7. Sub: a is Sub, but not NSub, for b; a and b are not FI.

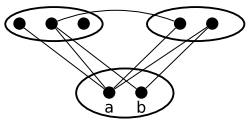


Fig. 8. a is NSub for b but a and b are not NI or FI.

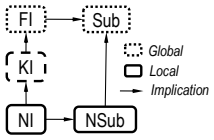


Fig. 9. Illustrating Theorems 1, 4, and 5.

Figure 9 illustrates this situation. First consider NI → NSub. Any NI values are mutually NSub by definition. In Figure 8, a is NSub for b but a and b are not NI. Now, consider NSub → Sub. Given two values a and b for a variable, a is NSub for b, the set of variable-value pairs supporting b is a subset of the one supporting a. By moving to global substitutability, the sets of supports will only lose elements, however, the set of support of b will remain a subset of that of a. Figure 7 shows an example where a is Sub, but not NSub, for b. In Figure 1, a and b are FI but not NSub. In Figure 8, a is NSub for b but a, and b are not FI.

Neighborhood Partial Interchangeability (NPI) [Choueiry and Noubir, 1998] Two values b and c for a variable v are NPI given a boundary of change S (which includes v) iff, for every constraint C defined on the variables (v,w) where $v \in S$, $w \notin S$, we have: $\{j|(b,j)$ satisfies $C\} = \{j|(c,j)$ satisfies $C\}$.

The NPI sets of a variable's domain can be detected by modifying the discrimination tree algorithm of NI to a joint discrimination tree (JDT) by considering the neighborhood of a set of variables instead of the neighborhood of a single variable as done in the discrimination tree [Choueiry and Noubir, 1998]. The complexity of the algorithm to build a JDT for a single variable is $\mathcal{O}(s(n-s)d^2)$ and the space complexity for the tree is $\mathcal{O}((n-s)d)$, where n is the number of variables, s is the size of the given set, and d is the size of the domain.

Theorem 6. NPI and PI are not comparable.[4]

In Figure 4, a and b are PI but not NPI. In Figure 11, they are NPI but not PI.

Theorem 7. NPI → SPrI.

If a and b are NPI outside the boundary of change S, then they are NI in the subproblem induced by $V \setminus S$. If they are NI in the subproblem, then they are also FI, and therefore SPrI in the subproblem induced by $V \setminus S$. Figure 10 shows an example where the converse does not hold.

Directional Interchangeability (DirI) [Naanaa, 2007a] Two values a and b in the domain of a variable X are DirI with respect to a variable ordering of the variables iff they have the same preceding support set: $\{c \mid (a,c) \in C_{XY}$ and $Y \prec X\} = \{c \mid (b,c) \in C_{XY}$ and $Y \prec X\}$.

[4] This theorem corrects [Choueiry and Noubir, 1998], which states that NPI implies PI. This error was mentioned in [Neagu and Faltings, 2005].

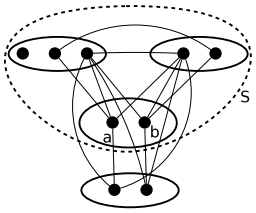


Fig. 10. SPrI: a and b are SPrI *wrt* S but not NPI *wrt* to any subset or Sub.

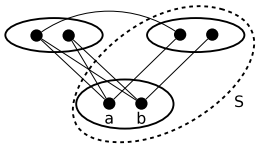


Fig. 11. NPI: a and b are NPI *wrt* S but not PI *wrt* to any subset, SUB, FI or NTI.

Theorem 8. NPI $\equiv$ DirI.

If a and b for a variable X are NPI *wrt* a boundary of change S, then they are DirI *wrt* any variable ordering such that $\forall Y \in S$, $X \prec Y$.

Directional Substitutability (DirSub) [Naanaa, 2007b; 2009] Value a is DirSub for value b for a variable X with respect to a variable ordering of the variables iff the preceding support set of b is a subset of that of b: $\{c \mid (b,c) \in C_{XY}$ and $Y \prec X\} \subseteq \{c \mid (a,c) \in C_{XY}$ and $Y \prec X\}$.

Theorem 9. DirI $\rightarrow$ DirSub.

If a and b of variable X are DirI *wrt* a given variable ordering, then, by definition, the preceding support sets for a and b are the same and consequently subsets of one another.

Neighborhood Interchangeability Relative to a Constraint ($\mathbf{NI}_C$) [Haselböck, 1993] Two values are NI_C relative to a constraint C iff they are NI in the problem induced by the variables in the scope of C.

NI_C values for variable v can be detected by restricting the discrimination tree to the considered constraint. As a result, the time complexity of finding all NI_C sets is $\mathcal{O}(eka^k)$, where e is the number of constraints, k is the maximum arity, and a is the maximum domain size. In [1993], Haselböck modified the usual Revise procedure for lookahead to exploit the (statically computed) NI_C sets during search, yielding a solution bundle. The time complexity of the new Revise procedure is thus reduced to $\mathcal{O}(a'^2)$, where $1 \leq a' \leq a$. In [Beckwith *et al.*, 2001], it was shown that the resulting bundles are never 'thinner' than those obtained in [Benson and Freuder, 1992], and never 'fatter' than those obtained by those obtained in [Hubbe and Freuder, 1992], which in turn are equivalent to those obtained by [Beckwith *et al.*, 2001].

Neighborhood Substitutability Relative to a Constraint ($\mathbf{NSub}_C$) [Bousse-mart *et al.*, 2004] Two values are NSub_C relative to a constraint C iff they are NSub in the problem induced by the variables in the scope of C.

Theorem 10. NPI $\rightarrow \mathrm{NI}_C \rightarrow \mathrm{NS}_C$.

First consider NPI $\rightarrow \mathrm{NI}_C$. If for variable X, a and b are NPI, then for every constraint C between X and a variable outside of the boundary of change, a and b are NI_C. $\mathrm{NI}_C \rightarrow \mathrm{NS}_C$ follows directly from the definition.

2.3 Other extended forms of interchangeability

Other extended forms that were initially proposed are: meta-interchangeability, dynamic interchangeability, and functional interchangeability.

Meta-interchangeability (MI) [Freuder, 1991] By grouping variables into 'meta-variables', or values into 'meta-values', we can introduce interchangeability into higher level 'meta-problem' representations of the original CSP.

Values may become interchangeable or substitutable during backtrack search after some variables have been instantiated, so even a problem with no interchangeable values may exhibit interchangeability under some search strategy.

Dynamic Neighborhood Interchangeability (DynNI)[5] [Beckwith and Choueiry, 2001] Two values a and b for variable X are DynNI with respect to a set A of variable assignments iff they are NI in the subproblem induced by $A \cup \{X\}$.

Theorem 11. *NI → DynNI.*

Consider values a and b for a variable v that are NI, assume a and b are not DynNI. Then, for an assignment for the subset of variables S, either a and b are not NI in the problem induced by $V \setminus S$, or one of a or b is deleted. The former case is impossible because a and b have the same set of supports in the original problem, and thus must have the same supports after the assignments. The latter case is also impossible because a and b having the same support sets, if a loses all its supports in a neighboring variable, then b also loses all supports because the support sets are the same.

Full Dynamic Interchangeability (FDynI) [Prestwich, 2004a] A value a for variable v is dynamically interchangeable for b with respect to a set A of variable assignments iff they are fully interchangeable in the subproblem induced by A.

Theorem 12. DynNI → FDynI.

If a and b are DynNI, then a and b are consistent with the same set of values in the assignment A. They are also NI relative to the variables in the problem induced by A that are not yet assigned and, consequently, are FI.

Functional interchangeability [Freuder, 1991] Let $S_{a|X}$ be the set of solutions including value a for variable X. Two values a for X and b for Y are *functionally interchangeable* iff there exists functions f and f' such that $f(S_{a|X}) = S_{b|Y}$ and $f(S_{b|Y}) = S_{a|X}$.

Two values a and b for a variable are *isomorphically interchangeable* [Freuder, 1991] iff there exists a 1-1 function f such that $b = f(a)$ and for any solution S involving a, $\{f(v) \mid v \in S\}$ is a solution. Also for any solution S involving b, $\{f^{-1}(v) \mid v \in S\}$ is a solution.

In the longer version of this paper, we compare functional and isomorphic interchangeability with the definitions of symmetry introduced in [Benhamou, 1994; Cohen *et al.*, 2006].

[5] Dynamic Interchangeability (DynI) property was incorrectly characterized as Dynamic Neighborhood Partial Interchangeability (DNPI) in [Beckwith and Choueiry, 2001; Choueiry and Davis, 2002; Lal and Choueiry, 2004; Lal *et al.*, 2005].

3 Conditional Forms of Interchangeability

Conditions can be added to a CSP in the form of constraints that further constrain the problem. In problems with little interchangeability, such conditions can be imposed to increase the interchangeability among the variable values. In [2004], Zhang and Freuder introduced and studied conditional interchangeability, conditional substitutability, conditional neighborhood interchangeability and conditional neighborhood substitutability.

Conditional Interchangeability (ConI) [Zhang and Freuder, 2004] Two values a and b of variable v are ConI under a condition imposed by a set of additional constraints iff they are FI in the problem with the additional constraints.

Similarly Conditional Neighborhood Interchangeability (ConNI), Conditional Substitutability (ConSub), and Conditional Neighborhood Substitutability (ConNSub) are defined by [Zhang and Freuder, 2004] where a problem is NI, Sub and NSub respectively given a set of conditions.

Theorem 13. (ConNI → ConI → ConSub), (ConNI → ConNSub → ConSub), and ConI and ConNSub are not comparable.

For ConNI → ConI and ConNSub → ConSub, see [Zhang and Freuder, 2004]. Consider the local forms: ConNI → ConNSub. For the same set of additional constraints, if a and b are ConNI in the original problem, they are NI in the problem with the additional constraints. Hence, they are also NSub in the problem with the additional constraints, and ConNSub in the original problem. The proof for the global forms (i.e., ConI → ConSub) is similar. Similar to the non-comparability of FI and NSub (see Theorem 5), ConI and ConNSub can be shown to be not comparable.

4 Other Forms of Interchangeability

In this section we review other forms of interchangeability that have appeared in the literature.

Neighborhood Tuple Interchangeability (NTI) [Neagu and Faltings, 1999]. Values a and b for variable v are NTI with respect to a set of variables S if for every consistent tuple t of value assignments to $S \cup \{v\}$ where $x = a$ there is another consistent tuple t' where $v = b$ such that t and t' are consistent with the same value combinations for variables outside of S. Additionally, the same condition must hold when a and b are exchanged. Figure 12 shows an example.

The algorithm proposed in [Neagu and Faltings, 2005] to detect NTI values determines the smallest set S using discrimination trees. The complexity of detecting NTI values is $\mathcal{O}((n^{s_{max}} s_{max}(n - s_{max})d^4)$, where n is the number of variables, d is the maximum domain size, and s_{max} is the maximum size of all possible dependent sets in the neighborhood of the variable.

Theorem 14. NI → NTI → PI and NTI → NPI.

First, consider NI → NTI. Given values a and b that are NI for a variable, for every consistent tuple t with a there is a tuple t' that only differs from t with a replaced with b. Hence t and t' are consistent with the same value combinations. Figure 12 gives an example where the converse does not hold. For (NTI → PI) and (NTI → NPI), see [Neagu and Faltings, 2005]. Figure 4 gives an example where PI ↛ NTI, and Figure 11 gives an example where NPI ↛ NTI.

In [2005], Wilson described a new approach to computation in a semiring-based system based on semiring-labeled decision diagrams (SLDDs). He defines forward neighborhood interchangeability (ForwNI) and uses it for merging nodes in SLDDs, hence compacting the search space. During search, ForwNI takes into account constraints that apply to instantiated and uninstantiated variables.

Forward Neighborhood Interchangeability (ForwNI) [Wilson, 2005] Given a subset of variables $U \subset V$, two assignments u and u' to a set of variables U are said to be ForwNI if for all constraints $c \in C$ such that $scope(c) \cap (V \setminus U) \neq \emptyset$ and $scope(c) \cap U \neq \emptyset$, $\Pi_{V \setminus U}\{t \in c \,|\, \Pi_U(t) = u\} = \Pi_{V \setminus U}\{t \in c \,|\, \Pi_U(t) = u'\}$.

Theorem 15. NTI → ForwNI.

If a and b for variable X are NTI with respect to set of variables S, then for every assignment t to $S \cup \{X\}$ where $X = a$ there is another consistent tuple t' where $X = b$ such that t and t' are consistent with the same value combinations for variables outside of S. Hence, the set of tuples consistent with t is the same for t' when projected on $V \setminus S$. Therefore, the assignments $\Pi_S(t)$ and $\Pi_S(t')$ are ForwNI.

Tuple substitutability is a global form of ForwNI:

Tuple Substitutability (TupSub) [Jeavons *et al.*, 1994] Two assignments A and B to a set of variables R are TupSub iff $\Pi_{V \setminus R}(\sigma_B(Sol)) \subseteq \Pi_{V \setminus R}(\sigma_A(Sol))$, where Sol is the set of all solutions to the problem.

Theorem 16. ForwNI → TupSub.

If two assignments u and u' are ForwNI, then for every solution in which u participates, u can be substituted with u' because they have the same supports in every constraint that links the scope of u to the rest of the problem. Therefore, the assignments u and u' are interchangeable and consequently substitutable.

Theorem 17. DynNI → ForwNI.

If a and b for variable X is DynNI *wrt* a set A of assignments, then any two assignments u and u' in $A \cup \{X\}$, such that $\Pi_X(u) = a$ and $\Pi_X(u') = b$, have the same set of support tuples because a and b are NI in $V \setminus A$. Therefore, u and u' are ForwNI.

Full Dynamic Substitutability (FDynSub) [Prestwich, 2004b] A value a for variable v is dynamically substitutable with value b with respect to a set A of variable assignments iff a is fully substitutable for b in the subproblem induced by A.

Theorem 18. FDynI → FDynSub, follows directly from the definition.

Theorem 19. Sub → FDynSub.

Consider value a Sub for b for variable v, the set of values supporting b is a subset of the set of values supporting a. Given an assignment of variables in FDynSub, the sets of supports will only lose elements, hence set of values supporting b will remain a subset of the set of values supporting a.

Theorem 20. FDynSub → ConNSub.

If a and b are FDynSub, then a set of constraints can be constructed for the original problem that removes all but the assigned values in the variables that are assigned in FDynSub. In the new problem resulting from adding those constraints, a and b are Sub. Moreover, a and b are NSub because all the values in the neighborhood that are not part of a solution are eliminated by the added constraints.

Theorem 21. TupSub and FDynSub are not comparable.

Context dependent interchangeability (CtxDepI) [Weigel *et al.*, 1996] Values a and b for a CSP variable X are CtxDepI, iff there exists a solution clique in the modified microstructure of the CSP that contain both nodes (X, a) and (X, b). The modified microstructure of the CSP is the original microstructure with edges added between values of the same variable. Figure 13 shows an example.

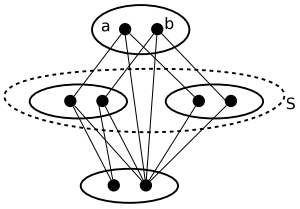


Fig. 12. NTI: a and b are NTI but not FDynNSub.

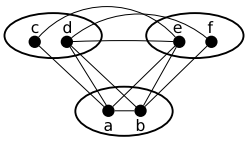


Fig. 13. CtxDepI: a and b are CtxDepI (clique $\{a, b, d, e\}$) but not Sub, FI, or PI.

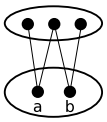


Fig. 14. GNSub: a and b are GNSub but not Sub.

Theorem 22. CtxDepI ≡ FDynI.

Connecting two CtxDepI values a and b for a CSP variable in the micro-structure of the CSP yields a solution clique with a and b. By assigning the values in the clique to the variables, we obtain an assignment set A where a and b are fully interchangeable in the subproblem induced by A. Conversely, if a and b are FDynI with respect to an assignment set A, then there is a solution clique in the modified micro-structure with a, b and all the values in A.

Theorem 23. FI → CtxDepI, FDynI.

If a and b are FI for a variable v, then a solution with $v = a$ yields another solution when replacing a with b. Thus, by connecting a and b in the micro-structure, we obtain a solution clique with a and b. Figure 13 shows an example where the converse does not hold.

Generalized Neighborhood Substitutability (GNSub) [Chmeiss and Sais, 2003] Two values of a variable are GNSub iff they share at least one support with respect to each neighboring variable. Figure 14 shows an example.

Theorem 24. NSub and GNSub are not comparable.

Figures 14 and 15 show counter examples.

Theorem 25. CtxDepI → GNSub

If a and b are CtxDepI, then the variable-value pairs connected to a in the micro-structure of the CSP are also connected to b. Thus, a and b share at least one support and are GNSub. Figure 16 shows an example where the converse does not hold.[6]

Theorem 26. GNSub → ConNI.

If a and b are GNSub for a variable, then we can construct a set of constraints that eliminates all values in the neighboring variables except the ones that are the shared support between a and b in order to make a and b ConNI. In Figure 17, the constraints that eliminate the supports of a make a and b ConNI.

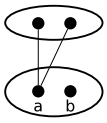


Fig. 15. NSub: a and b are NSub but not ConNI or GNSub.

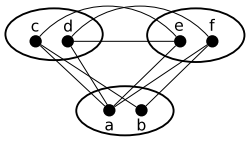


Fig. 16. GNSub: a and b are GNSub but not FDynI or CtxDepI.

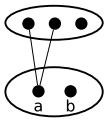


Fig. 17. ConNI: a and b are ConNI but not GNSub.

5 Relationships Between Interchangeability Concepts

The different interchangeability concepts surveyed in the previous sections are related here by the implication relation. Given two interchangeability concepts A and B, $A \rightarrow B$ if every interchangeable pair defined by A, is also defined by B. Hence, B generalizes A. Figures 18 and 19 illustrate those implication relations, and depict a partial ordering because some concepts are not comparable.

[6] Section 6.2 of [Zhang and Freuder, 2004] incorrectly states that CtxDepI and ConI are equivalent.

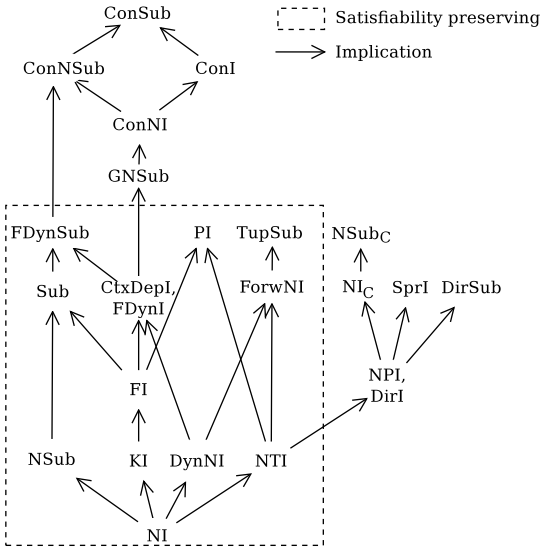


Fig. 18. Hasse Diagram of main interchangeability properties.

An interchangeability concept is *satisfiability preserving* iff when given two values a and b that are either interchangeable or a is substitutable for b, removing b from the problem does not alter the satisfiability of the problem. Not all interchangeability concepts are satisfiability preserving. Only the interchangeability concepts that are inside the dashed rectangle are satisfiability preserving.

In Figure 19, The upper horizontal plane groups concepts defined at the semantic level and thus are likely intractable. The lower horizontal plane groups concepts defined at the syntactic levels (i.e., directly on the constraints), and can likely be efficiently computed.

Interestingly, for a given form of interchangeability, when one moves vertically upward from the lower plane to the higher plane, interchangeability sets do not decrease in size while the interchangeability form is not approximated or compromised. Naturally, this advantage is not free because the computational cost does not decrease. Moving along the directed edges in either horizontal planes does not increase cost or the opportunities for interchangeability (i.e., size of interchangeability sets), but results in an approximation (i.e., weakening) of the 'quality' of the interchangeability. Finally, moving from the higher plane to the lower one allows one to likely avoid intractability and may increase the interchangeability opportunities but also results in approximations that may lose solutions.

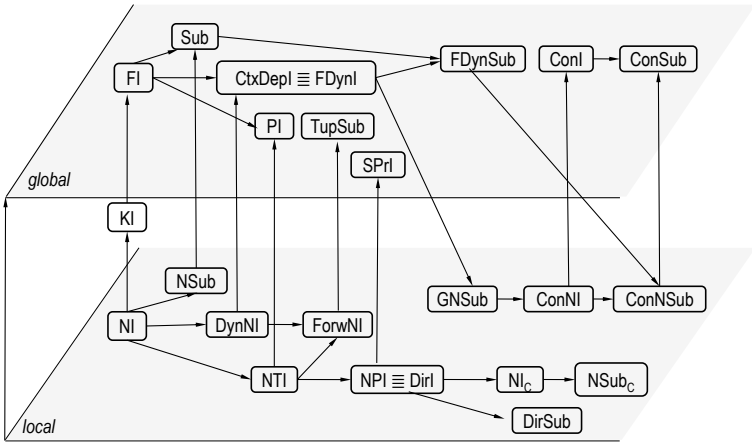


Fig. 19. Depicting qualitative relations between main interchangeability properties.

6 Work in Progress

In this paper we surveyed several forms of interchangeability, presented their definitions, and analyzed their relationship. This document is a work in progress. In the expanded version, we will address in depth the following topics:

More about interchangeability: concepts not mentioned here; missing proofs of incomparability; the satisfiability-preserving property; restrictions to constraint types; algorithms for interchangeability and their complexity; experimental results.

Beyond classical CSPs: soft constraints [Cooper, 2003; Bistarelli *et al.*, 2003; Neagu *et al.*, 2003; Neagu and Faltings, 2003]; distributed CSPs [Petcu and Faltings, 2003; Burke and Brown, 2006; Ezzahir *et al.*, 2007].

Relation to symmetry: various types [Benhamou, 1994; Cohen *et al.*, 2006]; symmetry breaking during search (SBDS) [Roney-Dougal *et al.*, 2004; Backofen and Will, 2002; Gent and Smith, 2000], symmetry breaking by dominance detection (SBDD) [Fahle *et al.*, 2001; Focacci and Milano, 2001], and symmetry breaking by enforcing variable or domain ordering [Bellicha *et al.*, 1994; Yip and Hentenryck, 2009]; restricted classes of symmetries and problems where the symmetry is broken in polynomial time [Hentenryck *et al.*, 2003; Benhamou, 2004].

Relation to search-space compaction: as in CPR [Hubbe and Freuder, 1992], AND/OR graphs [Dechter and Mateescu, 2004], SLDD [Wilson, 2005], and solution robustness [Ginsberg *et al.*, 1998; Hebrard *et al.*, 2004].

Relation to SAT solving.

Acknowledgments

This work was supported in part by Science Foundation Ireland under Grant 00/PI.1/C075. Karakashian and Woodward gratefully acknowledge the support

and hospitality of the Cork Constraint Computation Centre during Summer 2010 when this research was conducted.

References

[Backofen and Will, 2002] Rolf Backofen and Sebastian Will. Excluding Symmetries in Constraint-Based Search. *Constraints*, 7(3/4):333–349, 2002.

[Beckwith and Choueiry, 2001] Amy M. Beckwith and Berthe Y. Choueiry. On the Dynamic Detection of Interchangeability in Finite Constraint Satisfaction Problems. In *Principle and Practice of Constraint Programming (CP 01)*, volume 2239 of *LNCS*, page 760, Paphos, Cyprus, 2001.

[Beckwith *et al.*, 2001] Amy M. Beckwith, Berthe Y. Choueiry, and Hui Zou. How the Level of Interchangeability Embedded in a Finite Constraint Satisfaction Problem Affects the Performance of Search. In *AI 2001: 14th Australian Joint Conference on Artificial Intelligence*, volume 2256 of *LNAI*, pages 50–61, 2001.

[Bellicha *et al.*, 1994] Amit Bellicha, Christian Capelle, Michel Habib, Tibor Kökény, and Marie-Christine Vilarem. CSP Techniques Using Partial Orders On Domain Values. In *ECAI 1994 Workshop on Constraint Satisfaction Issues Raised by Practical Applications*, 1994.

[Benhamou, 1994] Belaid Benhamou. Study of Symmetry in Constraint Satisfaction Problems. In *Second Workshop on Principles and Practice of Constraint Programming (PPCP 94)*, pages 246–254, 1994.

[Benhamou, 2004] Belaid Benhamou. Symmetry in Not-Equals Binary Constraint Networks. In *Fourth International Workshop on Symmetry in Constraint Satisfaction Problems (SymCon 04)*, pages 2–8, 2004.

[Benson and Freuder, 1992] Brent W. Benson and Eugene C. Freuder. Interchangeability Preprocessing Can Improve Forward Checking Search. In *Tenth European Conference on Artificial Intelligence (ECAI 92)*, pages 28–30, 1992.

[Bistarelli *et al.*, 2003] Stefano Bistarelli, Boi Faltings, and Nicoleta Neagu. Interchangeability in Soft CSPs. In *Recent Advances in Constraints*, volume 2627 of *LNCS*, pages 45–68. Springer, 2003.

[Boussemart *et al.*, 2004] Frederic Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Support Inference for Generic Filtering. In *Principles and Practice of Constraint Programming (CP 04)*, volume 3258 of *LNCS*, pages 721–725. Springer, 2004.

[Brown *et al.*, 1988] Cynthia A. Brown, Larry Finkelstein, and Paul W. Purdom, Jr. Backtrack Searching in the Presence of Symmetry. In *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, volume 357 of *LNCS*, pages 99–110. Springer, 1988.

[Burke and Brown, 2006] David A. Burke and Kenneth N. Brown. Applying Interchangeability to Complex Local Problems in Distributed Constraint Reasoning. In *Workshop on Distributed Constraint Reasoning (AAMAS 06)*, pages 1–15, 2006.

[Chmeiss and Sais, 2003] Assef Chmeiss and Lakhdar Sais. About Neighborhood Substitutability In CSPs. In *Third International Workshop on Symmetry in Constraint Satisfaction Problems (SymCon 03)*, pages 41–45, 2003.

[Choueiry and Davis, 2002] Berthe Y. Choueiry and Amy M. Davis. Dynamic Bundling: Less Effort for More Solutions. In *International Symposium on Abstraction, Reformulation and Approximation (SARA 02)*, volume 2371 of *LNAI*, pages 64–82. Springer, 2002.

[Choueiry and Noubir, 1998] Berthe Y. Choueiry and Guevara Noubir. On the Computation of Local Interchangeability in Discrete Constraint Satisfaction Problems. In *Fifteenth National Conference on Artificial Intelligence (AAAI 98)*, pages 326–333, 1998.

[Choueiry *et al.*, 1995] Berthe Y. Choueiry, Boi Faltings, and Rainer Weigel. Abstraction by Interchangeability in Resource Allocation. In *14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, pages 1694–1701, 1995.

[Cohen *et al.*, 2006] David Cohen, Peter Jeavons, Christopher Jefferson, Karen E. Petrie, and Barbara M. Smith. Symmetry Definitions for Constraint Satisfaction Problems. *Constraints*, 11(2):115–137, 2006.

[Cooper, 2003] Martin C. Cooper. Reduction Operations in Fuzzy or Valued Constraint Satisfaction. *Fuzzy Sets and Systems*, 134(3):311–342, 2003.

[Dechter and Mateescu, 2004] Rina Dechter and Robert Mateescu. The Impact of AND/OR Search Spaces on Constraint Satisfaction and Counting. In *Principles and Practice of Constraint Programming (CP 04)*, volume 3258 of *LNCS*, pages 731–736, 2004.

[Ezzahir *et al.*, 2007] Redouane Ezzahir, Mustapha Belaissaoui, Christian Bessiere, and El Houssine Bouyakhf. Compilation Formulation for Asynchronous Backtracking with Complex Local Problems. In *Third International Symposium on Computational Intelligence and Intelligent Informatics (ISCIII 07)*, pages 205–211, 28-30 2007.

[Fahle *et al.*, 2001] Torsten Fahle, Stefan Schamberger, and Meinolf Sellman. Symmetry Breaking. In *Principles and Practices of Constraint Programming (CP 01)*, volume 2239 of *LNCS*, pages 93–107. Springer, 2001.

[Focacci and Milano, 2001] Filippo Focacci and Michela Milano. Global Cut Framework for Removing Symmetries. In *Seventh International Conference on Principles and Practice of Constraint Programming (CP 01)*, volume 2239 of *LNCS*, pages 77–92. Springer, 2001.

[Freuder, 1991] Eugene C. Freuder. Eliminating Interchangeable Values in Constraint Satisfaction Problems. In *National Conference on Artificial Intelligence (AAAI 91)*, pages 227–233, 1991.

[Gent and Smith, 2000] Ian P. Gent and Barbara M. Smith. Symmetry Breaking in Constraint Programming. In *Fourteenth European Conference on Artificial Intelligence (ECAI 00)*, pages 599–603. IOS Press, 2000.

[Gent *et al.*, 2006] Ian Gent, Karen Petrie, and Jean-François Puget. *Handbook of Constraint Programming*, chapter 10, pages 329–376. Elsevier, 2006.

[Ginsberg *et al.*, 1998] Matthew L. Ginsberg, Andrew J. Parkes, and Amitabha Roy. Supermodels and Robustness. In *Fifteenth National Conference on Artificial intelligence (AAAI 98)*, pages 334–339, 1998.

[Glaisher, 1874] J.W.L. Glaisher. On the Problem of the Eight Queens. *Philosophical Magazine, series 4*, 48:457–467, 1874.

[Haselböck, 1993] Alois Haselböck. Exploiting Interchangeabilities in Constraint Satisfaction Problems. In *13th International Joint Conference on Artificial Intelligence (IJCAI 93)*, pages 282–287, 1993.

[Hebrard *et al.*, 2004] Emmanuel Hebrard, Brahim Hnich, and Toby Walsh. Super Solutions in Constraint Programming. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 04)*, volume 3011 of *LNCS*, pages 157–172. Springer, 2004.

[Hentenryck *et al.*, 2003] Pascal Van Hentenryck, Pierre Flener, Justin Pearson, and Magnus Ågren. Tractable Symmetry Breaking for CSPs with Interchangeable Values. In *18th International Joint Conference on Artificial Intelligence (IJCAI 03)*, pages 277–282, 2003.

[Hubbe and Freuder, 1992] Paul D. Hubbe and Eugene C. Freuder. An Efficient Cross Product Representation of the Constraint Satisfaction Problem Search Space. In *Tenth National Conference on Artificial Intelligence (AAAI 92)*, pages 421–427, 1992.

[Jeavons *et al.*, 1994] Peter G. Jeavons, David A. Cohen, , and Martin C. Cooper. A Substitution Operation for Constraints. In *Second Workshop on Principles and Practice of Constraint Programming (PPCP 94)*, volume 874 of *LNCS*, pages 18–25. Springer, 1994.

[Lal and Choueiry, 2004] Anagh Lal and Berthe Y. Choueiry. Constraint Processing Techniques for Improving Join Computation: A Proof of Concept. In *Proceedings of the 1^{st} International Symposium on Constraint Databases, CDB'04*, volume 3074 of *LNCS*, pages 149–167. Springer, 2004.

[Lal *et al.*, 2005] Anagh Lal, Berthe Y. Choueiry, and Eugene C. Freuder. Neighborhood Interchangeability and Dynamic Bundling for Non-Binary Finite CSPs. In *20^{th} National Conference on Artificial Intelligence (AAAI 05)*, pages 397–404, 2005.

[Likitvivatanavong and Yap, 2008] Chavalit Likitvivatanavong and Roland H.C. Yap. A Refutation Approach to Neighborhood Interchangeability in CSPs. In *AI 2008: Advances in Artificial Intelligence*, volume 5360 of *LNCS*, pages 93–103. Springer, 2008.

[Naanaa, 2007a] Wady Naanaa. Directional Interchangeability for Enhancing CSP Solving. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 07)*, volume 4510 of *LNCS*, pages 200–213, 2007.

[Naanaa, 2007b] Wady Naanaa. Substitutability Based Domain Decomposition for Constraint Satisfaction. In *7^{th} International Workshop on Symmetry and Constraint Satisfaction Problems (SymCon 07)*, pages 64–71, 2007.

[Naanaa, 2009] Wady Naanaa. A Domain Decomposition Algorithm for Constraint Satisfaction. *Journal of Experimental Algorithmics (JEA)*, 13(1.13):1–23, 2009.

[Neagu and Faltings, 1999] Nicoleta Neagu and Boi Faltings. Constraint Satisfaction For Case Adaptation. In *Workshop on Formalisation of Adaptation in Case-Based Reasoning of ICCBR 99*, pages 35–41, 1999.

[Neagu and Faltings, 2003] Nicoleta Neagu and Boi Faltings. Soft Interchangeability for Case Adaptation. *Case-Based Reasoning Research and Development*, 2689:1066–1066, 2003.

[Neagu and Faltings, 2005] Nicoleta Neagu and Boi Faltings. Approximating Partial Interchangeability In CSP Solutions. In *International FLAIRS Conference (FLAIRS 05)*, pages 175–181, 2005.

[Neagu *et al.*, 2003] Nicoleta Neagu, Stefano Bistarelli, and Boi Faltings. On the Computation of Local Interchangeability in Soft Constraint Satisfaction Problems. In *International FLAIRS Conference (FLAIRS 03)*, pages 14–18, 2003.

[Petcu and Faltings, 2003] Adrian Petcu and Boi Faltings. Applying Interchangeability Techniques to the Distributed Breakout Algorithm. In *Principles and Practice of Constraint Programming (CP 03)*, volume 2833 of *LNCS*, pages 925–929. Springer, 2003.

[Prestwich, 2004a] Steven Prestwich. Full Dynamic Interchangeability with Forward Checking and Arc Consistency. In *Workshop on Modeling and Solving Problems With Constraints (ECAI 04)*, pages 1–14, 2004.

[Prestwich, 2004b] Steven Prestwich. Full Dynamic Substitutability by SAT Encoding. In *Principles and Practice of Constraint Programming (CP 04)*, volume 3258 of *LNCS*, pages 512–526. Springer, 2004.

[Roney-Dougal *et al.*, 2004] Colva M. Roney-Dougal, Ian P. Gent, Tom Kelsey, and Steve Linton. Tractable Symmetry Breaking Using Restricted Search Trees. In *Sixteenth European Conference on Artificial Intelligence (ECAI 04)*, pages 211–215, 2004.

[Various, 1991 present] Various. Proceedings of the International Workshop on Symmetry and Constraint Satisfaction Problems (SymCon) and of major AI Conferences such as AAAI, IJCAI, and CP, 1991 present.

[Weigel *et al.*, 1996] Rainer Weigel, Boi Faltings, and Berthe Y. Choueiry. Context in Discrete Constraint Satisfaction Problems. In *Twelfth European Conference on Artificial Intelligence (ECAI 96)*, pages 205–209, 1996.

[Weil and Heus, 1998] Georges Weil and Kamel Heus. Eliminating Interchangeable Values in the Nurse Scheduling Problem Formulated as a Constraint Satisfaction Problem. In *Workshop on Constraint-based reasoning in conjunction with FLAIRS'95*, Indianlantic, FL, 1998. Available from www.sci.tamucc.edu/constraint95/kamel.ps.

[Wilson, 2005] Nick Wilson. Decision Diagrams for the Computation of Semiring Valuations. In *International Joint Conference on Artificial Intelligence (IJCAI 05)*, pages 331–336, 2005.

[Yip and Hentenryck, 2009] Justin Yip and Pascal Van Hentenryck. Evaluation of Length-Lex Set Variables. In *Principles and Practice of Constraint Programming (CP 09)*, volume 5732 of *LNCS*, pages 817–832. Springer, 2009.

[Zhang and Freuder, 2004] Yuanlin Zhang and Eugene C. Freuder. Conditional Interchangeability and Substitutability. In *Fourth International Workshop on Symmetry and Constraint Satisfaction Problems (SymCon 04)*, 2004.

Internal Symmetry

Marijn Heule[1] and Toby Walsh[2]

[1] *Delft University of Technology, The Netherlands* marijn@heule.nl
[2] *NICTA and UNSW, Sydney, Australia* toby.walsh@nicta.com.au

Abstract. We have been studying the internal symmetries within an individual solution of a constraint satisfaction problem [1]. Such internal symmetries can be compared with solution symmetries which map between different solutions of the same problem. We show that we can take advantage of both types of symmetry when solving constraint satisfaction solutions within two benchmark domains. By identifying internal symmetries and breaking solution symmetries, we are able to increase the size of problems which have been solved.

1 Introduction

Symmetry occurs in several different forms. For example, when finding magic squares (prob019 in CSPLib [2]), we have the solution symmetries that describe the rotation and reflection of one solution onto another. We can also have internal symmetries which describe the mappings *within* each solution [1]. Methods for identifying and dealing with solution symmetries have been extensively studied. We can, for instance, post symmetry breaking constraints to eliminate symmetric solutions [3, 4]. Internal symmetries, on the other hand, have not received as much attention. However, they can be dealt with in a similar way. We simply add constraints that limit search to those solutions with a given internal symmetry. In addition, we can limit search further by only branching on the subset of decisions that then generate a complete solution. To demonstrate the value of exploiting such internal symmetries, we report results on two benchmark domains: Van der Waerden numbers and graceful graphs. Some (but not all) of the results reported here will first appear in [1].

2 Solution symmetry

A symmetry σ is a bijection on assignments. Given a set of assignments A and a symmetry σ, we write $\sigma(A)$ for $\{\sigma(a) \mid a \in A\}$. Similarly, given a set of symmetries Σ, we write $\Sigma(A)$ for $\{\sigma(a) \mid a \in A, \sigma \in \Sigma\}$. A special type of symmetry, called *solution symmetry* is a symmetry *between* the solutions of a problem. Such a symmetry maps solutions onto solutions. A solution is a set of assignments that satisfy every constraint in the problem. More formally, a problem has the *solution symmetry* σ iff σ of any solution is itself a solution [5]. The set of solution symmetries Σ of a problem forms a group under composition.

We say that two sets of assignments A and B are in the same *symmetry class* of Σ iff there exists $\sigma \in \Sigma$ such that $\sigma(A) = B$.

Running example. *The* magic squares *problem is to label a n by n square so that every row, column and diagonal have the same sum (prob019 in CSPLib [2]). A* normal *magic square contains the integers 1 to n^2. We model this with n^2 variables where $X_{i,j} = k$ iff the ith column and jth row is labelled with the integer k.*

"Lo Shu" is an important object in ancient Chinese mathematics. It is the smallest non-trivial normal magic square and has been known for over four thousand years:

4	9	2
3	5	7
8	1	6

(1)

The magic squares problem has a number of solution symmetries. For example, consider the symmetry σ_d that reflects a solution in the leading diagonal. This map "Lo Shu" onto a symmetric solution:

6	7	2
1	5	9
8	3	4

(2)

Any other rotation or reflection of the square maps one solution onto another. The 8 symmetries of the square (the dihedral group of order 8) are thus all solution symmetries of this problem. In fact, there are only 8 different magic square of order 3, and all are in the same symmetry class as "Lo Shu".

To eliminate such solution symmetry, we can, for instance, post symmetry breaking constraints that rule out symmetric solutions [3, 4, 6–13]. For example, to eliminate σ_d, we simply post an inequality constraint to ensure that the top left corner is smaller than its symmetry, the bottom right corner. This selects (1) and eliminates (2).

3 Internal symmetry

Symmetries can also be found within individual solutions of a constraint satisfaction problem. We say that a solution A *contains* the internal symmetry σ (or equivalently σ is a internal symmetry *within* this solution) iff $\sigma(A) = A$.

Running example. *Consider again "Lo Shu", the smallest normal magic square. This contains a simple internal symmetry. To see this, consider the solution symmetry σ_{inv} that inverts labels, mapping k onto n^2+1-k. This solution symmetry maps "Lo Shu" onto a different (but symmetric) solution. However, if we now*

apply the solution symmetry σ_{180} that rotates the square $180°$, we map back onto the original solution:

4	9	2
3	5	7
8	1	6

σ_{inv} $\Rightarrow$ $\Leftarrow$ σ_{180}

6	1	8
7	5	3
2	9	4

Consider the composition of these two symmetries: $\sigma_{inv} \circ \sigma_{180}$. This symmetry both inverts the labels in the square and rotates the square $180°$. As this symmetry maps "Lo Shu" onto itself, we see that the solution "Lo Shu" contains the internal symmetry $\sigma_{inv} \circ \sigma_{180}$.

Note that a solution symmetry is a property of every solution whilst an internal symmetry is a property of just the given solution.

Running example. *Consider the following magic square:*

14	11	5	4
1	8	10	15
12	13	3	6
7	2	16	9

This is one of the oldest known magic squares, dating from a 10th century engraving on the Parshvanath Jain temple in Khajuraho, India. $\sigma_{inv} \circ \sigma_{180}$ is not *an internal symmetry contained within this solution:*

14	11	5	4
1	8	10	15
12	13	3	6
7	2	16	9

$\Leftrightarrow$ $\sigma_{inv} \circ \sigma_{180}$

8	1	15	10
11	14	4	5
2	7	9	16
13	12	6	3

However, this internal symmetry is found within other order 4 solutions. Consider Albrecht Dürer's famous magic square:

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

(3)

This appears in his engraving "Melencolia I" of 1514 (as indicated by the two middle squares of the bottom row). It also plays a role in Dan Brown's novel "The Lost Symbol". The internal symmetry $\sigma_{inv} \circ \sigma_{180}$ is contained within (3):

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

σ_{inv} $\Rightarrow$ $\Leftarrow$ σ_{180}

1	14	15	4
12	7	6	9
8	11	10	5
13	2	3	16

Thus we can conclude that $\sigma_{inv} \circ \sigma_{180}$ is an internal symmetry contained within some but not all solutions of the normal magic squares problem. In fact, 48 out of the 880 distinct normal magic squares of order 4 contain this internal symmetry. On the other hand, $\sigma_{inv} \circ \sigma_{180}$ is a solution symmetry of normal magic square problems of every size.

A solution containing an internal symmetry can often be described by a subset of assignments and one or more symmetries acting on this subset that generate a complete set of assignments. Given a set of symmetries Σ, we write Σ^* for the closure of Σ. That is, $\Sigma^0 = \Sigma$, $\Sigma^i = \{\sigma_1 \circ \sigma_2 \mid \sigma_1 \in \Sigma, \sigma_2 \in \Sigma^{i-1}\}$, $\Sigma^* = \bigcup_i \Sigma^i$. Given a solution A, we say the subset B of A and the symmetries Σ *generate* A iff $A = B \cup \Sigma^*(B)$. In this case, we also describe A as containing the internal symmetries Σ.

Running example. *Consider the following magic square:*

1	8	12	13
14	11	7	2
15	10	6	3
4	5	9	16

(4)

This contains the internal symmetry $\sigma_{inv} \circ \sigma_{180}$. Half this magic square and $\sigma_{inv} \circ \sigma_{180}$ generate the whole solution:

1	8	12	13
14	11	7	2
-	-	-	-
-	-	-	-

$\Leftrightarrow$
$\sigma_{inv} \circ \sigma_{180}$

-	-	-	-
-	-	-	-
15	10	6	3
4	5	9	16

In fact, (4) can be generated from just the first quadrant and two symmetries: $\sigma_{inv} \circ \sigma_{180}$ and a symmetry τ which constructs a $180°$ rotation of the first quadrant in the second quadrant, decrementing those squares on the leading diagonal and incrementing those on the trailing diagonal (the same symmetry constructs the third quadrant from the fourth). More precisely, τ makes the following mappings:

a	*b*	-	-
c	*d*	-	-
-	-	-	-
-	-	-	-

$\Rightarrow$
τ

-	-	*d+1*	*c-1*
-	-	*b-1*	*a+1*
-	-	-	-
-	-	-	-

The example hints at how we can exploit internal symmetries within solutions. We will limit search to a subset of the decision variables that generates a complete set of assignments and construct the rest of the solution using the generating symmetries.

4 Theoretical properties

We list some properties of internal symmetries that will be used to help find solutions.

4.1 Set of internal symmetries within a solution

Like solution symmetries, the internal symmetries within a solution form a group. A solution A *contains* a set of internal symmetries Σ (or equivalently Σ are internal symmetries *within* the solution) iff A contains σ for every $\sigma \in \Sigma$.

Proposition 1. *The set of internal symmetries Σ within a solution A form a group under composition.*

The proof of this proposition and all subsequent propositions are given in [1].

4.2 Symmetries within and between solutions

In general, there is no relationship between the solution symmetries of a problem and the internal symmetries within a solution of that problem. There are solution symmetries of a problem which are not internal symmetries within any solution of that problem, and vice versa. The problem $Z_1 \neq Z_2$ has the solution symmetry that swaps Z_1 with Z_2, but no solutions of $Z_1 \neq Z_2$ contain this internal symmetry. On the other hand, the solution $Z_1 = Z_2 = 0$ of $Z_1 \leq Z_2$ contains the internal symmetry that swaps Z_1 and Z_2, but this is not a solution symmetry of $Z_1 \leq Z_2$ (since $Z_1 = 0$, $Z_2 = 1$ is a solution but its symmetry is not). When all solutions of a problem contain the same internal symmetry, we can be sure that this is a solution symmetry of the problem itself.

Proposition 2. *If all solutions of a problem contain an internal symmetry then this is a solution symmetry.*

By modus tollens, it follows that if σ is not a solution symmetry of a problem then there exists at least one solution which does not contain the internal symmetry σ.

4.3 Symmetries of symmetric solutions

We next consider internal symmetries contained within symmetric solutions. In general, the symmetry of a solution contains the conjugate of any internal symmetry contained within the original solution.

Proposition 3. *If the solution A contains the internal symmetry σ and τ is any (other) symmetry then $\tau(A)$ contains the internal symmetry $\tau \circ \sigma \circ \tau^{-1}$.*

In the special case that symmetries commute, the symmetry of a solution contains the same internal symmetries as the original problem. Two symmetries σ and τ *commute* iff $\sigma \circ \tau = \tau \circ \sigma$.

Proposition 4. *If the solution A contains the internal symmetry σ and τ commutes with σ then $\tau(A)$ also contains the internal symmetry σ.*

Running example. *Consider again "Lo Shu", the smallest normal magic square. This solution contains the internal symmetry* $\sigma_{inv} \circ \sigma_{180}$. *This particular symmetry commutes with any rotation symmetry. For instance, consider the rotation of "Lo Shu" by* 90° *clockwise:*

8	3	4	σ_{inv}		2	7	6
1	5	9	$\Rightarrow$		9	5	1
6	7	2	$\Leftarrow$		4	3	8
			σ_{180}				

This symmetry of "Lo Shu" also contains the internal symmetry $\sigma_{inv} \circ \sigma_{180}$.

4.4 Symmetry breaking

Finally, we consider the compatibility of eliminating symmetric solutions and focusing search on those solutions that contain particular internal symmetries. In general, the two techniques are incompatible. Symmetric breaking may eliminate all those solutions which contain a given internal symmetry.

Running example. *Consider the following magic square:*

1	4	13	16
14	15	2	3
8	5	12	9
11	10	7	6

(5)

This contains the internal symmetry $\sigma_v \circ \sigma_{inv}$ *that inverts all values and reflects the square in the vertical axis:*

1	4	13	16	σ_{inv}	16	13	4	1
14	15	2	3	$\Rightarrow$	3	2	15	14
8	5	12	9	$\Leftarrow$	9	12	5	8
11	10	7	6	σ_v	6	7	10	11

Note that this internal symmetry can only occur within magic squares of even order or of order 1.

Suppose symmetry breaking eliminates all solutions in the same symmetry class as (5) except for a symmetric solution which is a 90° *clockwise rotation of (5). This solution does not contain the internal symmetry* $\sigma_v \circ \sigma_{inv}$. *In fact, this rotation of (5) contains the internal symmetry that inverts all values and reflects the square in the* horizontal *axis.*

11	8	14	1		16	3	9	6
10	5	15	4	$\Leftrightarrow$	13	2	12	7
7	12	2	13	$\sigma_v \circ \sigma_{inv}$	4	15	5	10
6	9	3	16		1	14	8	11

We can identify a special case where symmetry breaking does not change any internal symmetry within solutions. Suppose symmetry breaking only eliminates symmetries which commute with the internal symmetry contained within a particular solution. In this case, whilst symmetry breaking may eliminate the given solution, it must leave a symmetric solution containing the given internal symmetry. Given a set of constraints C with solution symmetries Σ, we say that a set of symmetry breaking constraints S is *sound* iff for every solution of C there exists at least one solution of $C \cup S$ in the same symmetry class.

Proposition 5. *Given a set of constraints C with solution symmetries Σ, a sound set of symmetry breaking constraints S, and a solution A containing the internal symmetry σ, if σ commutes with every symmetry in Σ then there exists a solution of $C \cup S$ in the same symmetry class as A also containing the internal symmetry σ.*

Running example. *Consider the internal symmetry $\sigma_{inv} \circ \sigma_{180}$ contained within some (but not all) normal magic squares. This particular symmetry commutes with every rotation, reflection and inversion solution symmetry of the problem. Hence, if there is a solution with the internal symmetry $\sigma_{inv} \circ \sigma_{180}$, this remains true after breaking the rotational, reflection and inversion symmetries. However, as in the last example, there are internal symmetries contained within some solutions (like reflection in the vertical axis) which do not commute with all symmetries of the square.*

5 Detecting internal symmetries

Detection of internal symmetries in a class of problems consists of two steps: 1) selecting a good candidate solution, and 2) finding internal symmetries in that solution. As we already discussed earlier, an internal symmetry of one solution may not be an internal symmetry of another solution for the same problem. Recall in this context Albrecht Dürer's famous magic square (3) that has internal symmetry $\sigma_{inv} \circ \sigma_{180}$, while most magic squares with the same size don't. Before trying to detect internal symmetries, we consider a strategy to select a solution that is more likely to reveal internal symmetries.

The strategy consists of two parts: 1) determine a good size of the problem to look at and 2) filtering out solutions. First, consider the problem size. In order to exploit internal symmetries, internal symmetries observed at a certain size need also to hold for other sizes. Therefore, it makes sense to select problems of a size that have only very few (preferably only one) solutions. Yet for most problems, such as the magic squares, only the smallest problems have few solutions. Unfortunately these sizes might be too small to do some proper internal symmetry detection. However, for some problems, such as Van der Waerden numbers which we consider in Section 7, the number of solutions decreases while increasing the size. In those cases a "good" size is much larger.

After selecting the size, we filter out solutions using the following method. First, we compute the set $\mathcal{S}$ consisting of all solutions of the selected size while

applying symmetry breaking. Let set $\mathcal{V}$ be the set of variables that describe the solutions in $\mathcal{S}$. Then we repeat until $|\mathcal{S}| = 1$: Select a variable $v \in \mathcal{V}$ that is most frequently assigned the same value in the solutions of $\mathcal{S}$. Remove from $\mathcal{S}$ all solutions that have v assigned to another value and $\mathcal{V} := \mathcal{V} \setminus \{v\}$.

Running example. *Consider the magic squares problem of size* 5×5*. After symmetry breaking*[3] *(in this case* 25 *transpositions,* 4 *rotations, and* 2 *reflections), this problem has 144 solutions. In all those solutions, the first element is placed in the first (top left) entry. After fixing element* 1*, element* 6 *occurs most freqently in the same position. More specifically, it appears in 72 solutions on the last row in the fourth column. After fixing element* 6 *to that position, element* 11 *occurs in 36 of the remaining 72 solutions on second entry of the fourth row. The next elements to be placed are* 16 *and* 21 *and after fixing them, 24 solutions are left with the following pattern:*

1	-	-	-	-
-	-	21	-	-
-	-	-	-	16
-	11	-	-	-
-	-	-	6	-

By repeating the procedure, we will select the following solution:

1	7	13	19	25
14	20	21	2	8
22	3	9	15	16
10	11	17	23	4
18	24	5	6	12

(6)

This solution has an interesting internal symmetry, which will be explained in the next example.

The last step is to detect internal symmetries in the selected solution. This step is the hardest to automate. As a general approach we propose the following. Construct a set of symmetries Σ that include at least all solutions symmetries. Let A be the selected solution. We then use a search program to find $s_1, s_2, \ldots s_k \in \Sigma$ that satisfy $s_1 \circ s_2 \circ \cdots \circ s_k(A) = A$.

Although internal symmetries can be found using this approach – even when restricted to the solution symmetries – in our experience the most effective internal symmetries in the selected solution were often found manually.

[3] For this example we use the symmetry breaking based on the score function that is defined on http://www.grogono.com/magic/5x5pan144.php. In short, out of each symmetry group the one is selected for which the elements in the top left corner have the smallest elements.

Running example. *Consider the magic square we selected. We denote by* $\sigma_{\mathrm{row}(x,y,z)}$ *a transposition of the rows such the row 1 is replaced by row* x*, row 2 by row* y *and row 3 by row* z*. An automated search program that only combines solution symmetries to detect internal symmetries can find the following symmetry* $\sigma_{\mathrm{row}(2,3,1)} \circ \sigma_{180} \circ \sigma_{inv} \circ \sigma_{\mathrm{row}(3,1,2)}$ *:*

1	7	13	19	25
14	20	21	2	8
22	3	9	15	16
10	11	17	23	4
18	24	5	6	12

$\sigma_{\mathrm{row}(2,3,1)} \Rightarrow \Leftarrow \sigma_{\mathrm{row}(3,1,2)}$

14	20	21	2	8
22	3	9	15	16
1	7	13	19	25
10	11	17	23	4
18	24	5	6	12

$\sigma_{180} \Rightarrow \Leftarrow \sigma_{inv}$

12	6	5	24	18
4	23	17	11	10
25	19	13	7	1
16	15	9	3	22
8	2	21	20	14

However, this is not the most useful internal symmetry that can be observed in this magic square. A careful look at the filtering procedure above reveals that for the first five elements i *that are fixed, we have* $i = 1 (mod\ 5)$*. More importantly, all elements are fixed at a location that is a knight's move (two left, one up) from each other. A similar pattern is observable for the first five elements* $i \in \{1, \ldots, 5\}$*: They are also a knight's move (two left, one down) from each other. Combining both patterns gives a construction method for odd magic squares that is very similar to the Siamese method*[4]*: After placing the first element, place the next element on the entry reachable by a knight's move (two left, one down). If the entry is already filled, then place the element in the entry directly right of the last placed element.*

6 Exploiting internal symmetries

Once we have identified an internal symmetry which we conjecture may be contained in solutions of other (perhaps larger) instances of the problem, it is a simple matter to restrict search of a constraint solver to solutions of this form. In general, if we want to find solutions containing the internal symmetry σ, we post symmetry constraints of the form:

$$Z_i = j \Rightarrow \sigma(Z_i = j)$$

In addition, we can limit branching decisions to a subset of the decisions variables that generates a complete set of assignments. This can significantly reduce the size of the search space. Propagation of the problem and symmetry constraints may prune the search space even further.

7 Van der Waerden numbers

We illustrate the use of internal symmetries within solutions with two applications where we have been able to extend the state of the art. In the first, we

[4] see http://en.wikipedia.org/wiki/Siamese_method

found new lower bound certificates for Van der Waerden numbers. Such numbers are an important concept in Ramsey theory. In the second application, we increased the size of graceful labellings known for a family of graphs. Graceful labelling has practical applications in areas like communication theory.

The Van der Waerden number, $W(k,l)$ is the smallest integer n such that if the integers 1 to n are colored with k colors then there are always at least l integers in arithmetic progression. For instance, $W(2,3)$ is 9 since the two sets $\{1,4,5,8\}$ and $\{2,3,6,7\}$ contain no arithmetic progression of length 3, but every partitioning of the integers 1 to 9 into two sets contains an arithmetic progression of length 3 or more. The certificate that $W(2,3) > 8$ can be represented with the following blocks:

Finding such certificates can be encoded as a constraint satisfaction problem. To test if $W(k,l) > n$, we introduce the Boolean variable $x_{i,j}$ where $i \in [0,k)$, $j \in [0,n)$ and constraints that each integer takes one color ($\bigvee_{i\in[0,k)} x_{i,j}$), and that no row of colors contains an arithmetic progression of length l ($x_{i,a} \wedge \ldots \wedge x_{i,a+d(l-2)} \rightarrow \neg x_{i,a+d(l-1)}$). This problem has a number of solution symmetries. For example, we can reverse any certificate and get another symmetric certificate. We can also permute the colors and get another symmetric certificate:

Individual certificates also often contain internal symmetry. For example, the second half of the last certificate repeats the first half:

Hence, this certificate contains an internal symmetry that maps $x_{i,j}$ onto $x_{i,j+4 \mod 8}$.

In fact, many known certificates can be generated from some simple symmetry operations on just the colors assigned to the first two or three integers. For instance, the first construction method for Van der Waerden certificates [14] made use of the observation that the largest possible certificates for the known numbers $W(k,l)$[5] consist of a repetition of $l-1$ times a base pattern. All these certificates, as well as all best lower bounds, have a base pattern of size $m = \frac{n}{l-1}$. This first method only worked for certificates for which m is prime. An improved construction method [15] generalises it for non-prime m.

An important concept in both construction methods is the primitive root[6] of m denoted by r. Let p be the largest prime factor of m, then r is the smallest number for which:

$$r^i (\mathrm{mod}\ m) \neq r^j (\mathrm{mod}\ m) \quad \text{for } 1 \leq i < j < p \tag{7}$$

[5] except for $W(3,3)$
[6] our use slightly differs from the conventional definition

We identified four internal symmetries:

σ_{+m}: Apply to all elements $x_{i,j} := x_{i,j+m \pmod n}$
σ_{+p}: Apply to all elements $x_{i,j} := x_{i,j+p \pmod m}$
$\sigma_{\times r}$: Apply to all elements $x_{i,j} := x_{i,j\times r \pmod m}$
$\sigma_{\times r^t}$: At least one subset maps onto itself after applying $x_{i,j} := x_{i,j\times r^t \pmod m}$ for a $t \in \{1,\ldots,k\}$

Consider the largest known certificate for $W(5,3)$ which has 170 elements. For this certificate, $m = 85, p = 17$, and $r = 3$. Below the base pattern is shown the first 85 elements. Notice that for this certificate A, $\sigma_{+p}(A)$ and $\sigma_{\times r}(A)$ are also certificates. In fact, after sorting the elements and permuting the subsets, this certificate is mapped onto itself after applying these symmetries.

18	20	24	26	33	36	38	44	65	66	74	76	79	80	5	13	17
22	30	34	35	37	41	43	50	53	55	61	82	83	6	8	11	12
23	25	28	29	39	47	51	52	54	58	60	67	70	72	78	14	15
31	32	40	42	45	46	56	64	68	69	71	75	77	84	2	4	10
19	21	27	48	49	57	59	62	63	73	81	85	1	3	7	9	16

$\Uparrow \sigma_{+p}$

1	3	7	9	16	19	21	27	48	49	57	59	62	63	73	81	85
5	13	17	18	20	24	26	33	36	38	44	65	66	74	76	79	80
6	8	11	12	22	30	34	35	37	41	43	50	53	55	61	82	83
14	15	23	25	28	29	39	47	51	52	54	58	60	67	70	72	78
2	4	10	31	32	40	42	45	46	56	64	68	69	71	75	77	84

$\Downarrow \sigma_{\times r}$

3	9	21	27	48	57	63	81	59	62	1	7	16	19	49	73	85
15	39	51	54	60	72	78	14	23	29	47	25	28	52	58	67	70
18	24	33	36	66	5	17	20	26	38	44	65	74	80	13	76	79
42	45	69	75	84	2	32	56	68	71	77	4	10	31	40	46	64
6	12	30	8	11	35	41	50	53	83	22	34	37	43	55	61	82

Given these symmetries, we can easily construct a complete certificate. We place the first and last elements (1 and 85) in the first subset and apply $\sigma_{\times r}$ to generate all elements in this subset. We apply σ_{+p} to partition the elements $\{1,\ldots,85\}$. Finally, we obtain a complete certificate by applying σ_{+m}. We generalised this into a construction method. To find a larger certificate $W(k,l,n)$, we test with a constraint solver for increasing $n \equiv 0 \pmod{l-1}$ whether a certificate can be obtained using the following steps:

- break solution symmetry by forcing that the first subset of the partition maps onto itself after applying $\sigma_{\times r^t}$
- choose $t \in \{1,\ldots,k\}$, $q \in \{1,\ldots,\frac{m}{p}\}$
- place elements q and m in the first subset
- apply the symmetries $\sigma_{\times r^t}$, $\sigma_{\times r}$, σ_{+p}, and σ_{+m}, to construct a certificate A with n' elements
- check with a constraint solver if A lacks an arithmetic progression of length l

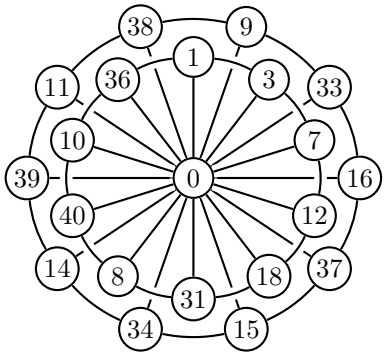
Using this method we significantly improved some of the best known lower bounds[7]:

- $W(3,7) > 48811$. The old bound was 43855.
- $W(4,7) > 420217$. The old bound was 393469.

8 Graceful graphs

Our second application of internal symmetries is graceful labelling. A graph with e edges is called graceful if its vertices can be labelled with the distinct values $\{0,\ldots,e\}$ in such a way that each edge gets a unique label when it is assigned the absolute difference of the vertices it connects. Graceful labelling has a wide range of applications in areas like radio astronomy, cryptography, communication networks and circuit design.

Whilst various classes of graphs are known to be graceful [16], there are others where it is not known but is conjectured that they are graceful. One such class is the class of double wheel graphs. The graph DW_n consists of two cycles of size n and a hub connected all the vertices. The largest double wheel graph that we have seen graceful labelled in the literature[8]has size 10.



The problem of finding a graceful labelling can be specified using $2n+1$ variables X_i with domain $\{0,\ldots,e\}$. This problem has $16n^2$ solution symmetries [17]:

- Rotation of the vertices (n^2 symmetries)
- Inversion of the order of the vertices (4 symmetries)
- Swapping of the inner and outer wheel (2 symmetries)
- Inversion of the labels, $X_i := 4n - X_i$ (2 symmetries)

To identify internal symmetries, we generated all graceful labellings for DW_4. This is the smallest double wheel graph with a graceful labelling. We observed two internal symmetries within the 44 solutions of DW_4:

σ_{4n}: In 31 solutions, the hub had label $4n$ or 0 (σ_{inv}).

σ_{+2}: If $1 \leq X_i \leq n-2$, then $X_{i+2} := X_i + 2$

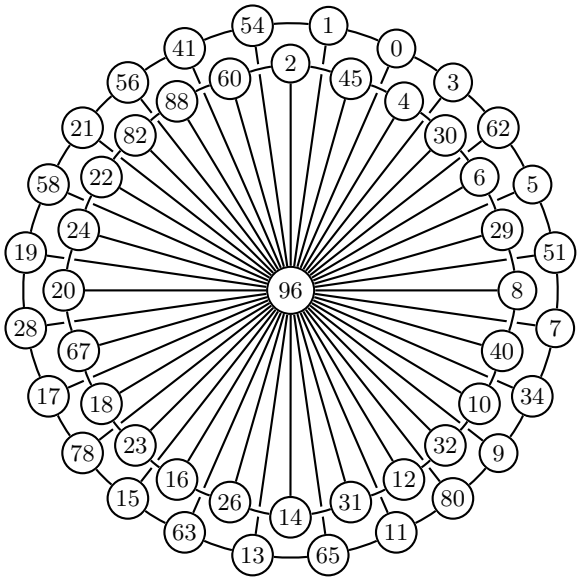
Although we observed σ_{+2}, we restrict this internal symmetry to $1 \leq X_i \leq n-4$ because it proved more effective.

When both symmetries are applied, the computational costs to find a graceful labelling is significantly reduced. Consider DW_{24}. To construct a graceful labelling, we first assign the hub to value 96 (applying σ_{4n}). Second, we label the first vertex of the outer wheel with 1 and label the first vertex of the inner

[7] see www.st.ewi.tudelft.nl/sat/∼waerden.php

[8] see www.comp.leeds.ac.uk/bms/Graceful/

wheel with 2. Third, we apply symmetry σ_{+2} to label n-1 vertices with the labels $\{1, \ldots, n-1\}$. Finally, we use a constraint solver to label the remaining vertices. Using this method we found the first known graceful labeling for DW_{24}.



The right table gives the runtime (in seconds) for our constraint solver to find graceful labellings of DW_n for the original problem (P) with and without symmetry breaking (SB) constraints [17]. The last column shows the results when we force internal symmetries within solutions. This also breaks the solution symmetries.

n	P	$P + \text{SB}$	$P{+}\sigma_{4n},\sigma_{+2}$
4	**0.01**	**0.01**	**0.01**
8	1.88	0.78	**0.1**
12	82.12	25.18	**0.3**
16	1,608	706.35	**4.97**
20	21,980	8,272	**17.25**
24	$> 36,000$	$> 36,000$	**157.87**

9 Related work

Several forms of symmetry have been identified and exploited in search. For instance, Brown, Finkelstein and Purdom defined symmetry as a permutation of the variables leaving the set of solutions invariant [18]. This is a subset of the solution symmetries. For the propositional calculus, Krishnamurthy was one of the first to exploit symmetry [19]. He defined symmetry as a permutation of the variables leaving the set of clauses unchanged. Benhamou and Sais extended this to a permutation of the literals preserving the set of clauses [20].

The symmetry of individual constraints has also been considered. For example, Puget considered permutations of the variables leaving the set of constraints invariant [3]. He proved that such symmetry can be eliminated by the

addition of static constraints. Crawford *et al.* presented the first general method for constructing static constraints for eliminating solution symmetries [4]. Perhaps closest to this work is Puget's symmetry breaking method that considers symmetries which stabilize the current partial set of assignments [21]. By comparison, we consider only those symmetries which stabilize a complete set of assignments. A stabilizer maps individual assignments onto themselves, whilst an internal symmetry maps a set of assignments onto the same set (but may change every individual assignment).

Apart from symmetry, the idea of exploiting regularities in solutions of small sized problems in order to constrain large sized problems has been studied before. For instance, streamlining constraints in CP [22] and resolution tunnels in SAT [23]. In contrast to other work, internal symmetries focuses on a specific regularity: a mapping of a set of assignments onto itself.

10 Conclusions

We have been studying internal symmetries within a single solution of a constraint satisfaction problem [1]. Internal symmetries are properties of an individual solution. They can be compared with solution symmetries which are properties of all solutions of a constraint satisfaction problem. Both types of symmetry can be profitably exploited when solving constraint satisfaction problems. We illustrated the potential of doing this on two benchmark domains: Van der Waerden numbers and graceful graphs. With the first, we improved some of the best known lower bounds by around 10%. With the second, we more than doubled the size of the largest known double wheel graph with a graceful labelling from a wheel of size 10 to a wheel of size 24.

Acknowledgments

The authors are supported by the Dutch Organization for Scientific Research (NWO) under grant 617.023.611, the Australian Government's Department of Broadband, Communications and the Digital Economy and the ARC.

References

1. Heule, M., Walsh, T.: Symmetry within solutions. In: Proceedings of AAAI'10. (2010) 77–82
2. Gent, I., Walsh, T.: CSPLib: a benchmark library for constraints. Technical report, Technical report APES-09-1999 (1999).
3. Puget, J.F.: On the satisfiability of symmetrical constrained satisfaction problems. In Komorowski, J., Ras, Z., eds.: Proceedings of ISMIS'93. LNAI 689, Springer-Verlag (1993) 350–361
4. Crawford, J., Ginsberg, M., Luks, G., Roy, A.: Symmetry breaking predicates for search problems. In: Proceedings of the 5th International Conference on Knowledge Representation and Reasoning, (KR '96). (1996) 148–159

5. Cohen, D., Jeavons, P., Jefferson, C., Petrie, K., Smith, B.: Symmetry definitions for constraint satisfaction problems. Constraints **11**(2–3) (2006) 115–137
6. Flener, P., Frisch, A., Hnich, B., Kiziltan, Z., Miguel, I., Pearson, J., Walsh, T.: Breaking row and column symmetry in matrix models. In: 8th International Conference on Principles and Practices of Constraint Programming (CP-2002), Springer (2002)
7. Frisch, A., Hnich, B., Kiziltan, Z., Miguel, I., Walsh, T.: Global constraints for lexicographic orderings. In: 8th International Conference on Principles and Practices of Constraint Programming (CP-2002), Springer (2002)
8. Walsh, T.: General symmetry breaking constraints. In Benhamou, F., ed.: 12th International Conference on Principles and Practice of Constraint Programming (CP 2006). Volume 4204 of LNCS, Springer (2006)
9. Walsh, T.: Symmetry breaking using value precedence. In Brewka, G., Coradeschi, S., Perini, A., Traverso, P., eds.: 17th European Conference on Artificial Intelligence, IOS Press (2006) 168–172
10. Law, Y.C., Lee, J., Walsh, T., Yip, J.: Breaking symmetry of interchangeable variables and values. In: 13th International Conference on Principles and Practices of Constraint Programming (CP-2007), Springer-Verlag (2007)
11. Walsh, T.: Breaking value symmetry. In: 13th International Conference on Principles and Practices of Constraint Programming (CP-2007), Springer-Verlag (2007)
12. Katsirelos, G., Walsh, T.: Symmetries of symmetry breaking constraints. In: Proceedings of the 19th ECAI, IOS Press (2010)
13. Katsirelos, G., Narodytska, N., Walsh, T.: Static constraints for breaking row and column symmetry. In: 16th International Conference on Principles and Practices of Constraint Programming (CP-2010), Springer-Verlag (2010)
14. Rabung, J.R.: Some progression-free partitions constructed using Folkman's method. Canadian Mathematical Bulletin **22**(1) (1979) 87–91
15. Herwig, P., Heule, M.J., van Lambalgen, M., van Maaren, H.: A new method to construct lower bounds for Van der Waerden numbers. The Electronic Journal of Combinatorics **14** (2007)
16. Gallian, J.A.: A dynamic survey of graph labelling. The Electronic Journal of Combinatorics **5** (1998) #DS6 Updated in 2008.
17. Petrie, K.E., Smith, B.M.: Symmetry breaking in graceful graphs. In: Principles and Practice of Constraint Programming. Volume 2833 of LNCS. (2003) 930–934
18. Brown, C., Finkelstein, L., Jr., P.P.: Backtrack searching in the presence of symmetry. In Mora, T., ed.: Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, 6th International Conference. Volume 357 of LNCS. (1988) 99–110
19. Krishnamurthy, B.: Short proofs for tricky formulas. Acta Informatica **22**(3) (1985) 253–275
20. Benhamou, B., Sais, L.: Theoretical study of symmetries in propositional calculus and applications. In: Proceedings of 11th International Conference on Automated Deduction. Volume 607 of LNCS. (1992) 281–294
21. Puget, J.F.: Symmetry breaking using stabilizers. In Rossi, F., ed.: Proceedings of 9th International Conference on Principles and Practice of Constraint Programming (CP2003), Springer (2003)
22. Gomes, C., Sellmann, M.: Streamlined constraint reasoning. In: Principles and Practice of Constraint Programming CP 2004. Volume 3258 of Lecture Notes in Computer Science. (2004) 274–289
23. Kouril, M., Franco, J.: Resolution tunnels for improved SAT solver performance. In: Theory and Applications of Satisfiability Testing. Volume 3569 of Lecture Notes in Computer Science. (2005) 143–157

Symmetries and Lazy Clause Generation

Geoffrey Chu[1], Maria Garcia de la Banda[2], Chris Mears[2], and Peter J. Stuckey[1]

[1] National ICT Australia, Victoria Laboratory,
Department of Computer Science and Software Engineering,
University of Melbourne, Australia
{gchu,pjs}@csse.unimelb.edu.au
[2] Faculty of Information Technology,
Monash University, Australia
{cmears,mbanda}@infotech.monash.edu.au

Abstract. Lazy clause generation is a powerful approach to reducing search in constraint programming. This is achieved by recording sets of domain restrictions that previously lead to failure as new clausal propagators. Symmetry breaking approaches are also powerful methods for reducing search by recognizing that parts of the search tree are symmetric and do not need to be explored. In this paper we show how we can successfully combine symmetry breaking methods with lazy clause generation. Further, we show that the more precise nogoods generated by a lazy clause solver allow our combined approach to exploit redundancies that cannot be exploited via any previous symmetry breaking method, be it static or dynamic.

1 Introduction

Lazy clause generation [6] is a hybrid approach to constraint solving that combines features of finite domain propagation and Boolean satisfiability. Finite domain propagation is instrumented to record the reasons for each propagation step. This creates an implication graph like that built by a SAT solver, which may be used to create efficient nogoods that record the reasons for failure. These nogoods can be propagated efficiently using SAT unit propagation technology. The resulting hybrid system combines some of the advantages of finite domain constraint programming (high level model and programmable search) with some of the advantages of SAT solvers (reduced search by nogood creation, and effective autonomous search using variable activities). Thanks to this lazy clause generation provides state of the art solutions to a number of combinatorial optimization problems such as Resource Constrained Project Scheduling Problems [7].

Symmetry breaking methods aim at speeding up the execution by pruning parts of the search tree known to be symmetric to those explored. While static symmetry breaking methods achieve this by adding constraints to the original problem, dynamic symmetry breaking methods alter the search. As we will see later, combining static symmetry breaking with lazy clause generation is straightforward and quite successful. However, dynamic symmetry breaking can sometimes be more effective than static symmetry breaking. Thus, we are also

interested in combining lazy clause generation with dynamic symmetry breaking methods. While this combination is much more complex, it also allows us to exploit certain types of redundancies which were previously impossible to exploit via any other traditional static or dynamic symmetry breaking method.

As we will show in this paper, the key to the success of our combination resides in the fact that dynamic symmetry breaking methods can also be defined in terms of nogoods. In particular, they can be thought of as utilising symmetric versions of nogoods derived at each search node to prune off symmetric portions of the search space. Thus, both lazy clause generation and dynamic symmetry breaking use nogoods to prune the search space. The differences arise in the kind of nogoods used and in the way these nogoods are used. Traditional dynamic symmetry breaking methods such as SBDS [4] and SBDD [3, 1], use what we will call the *choice* nogood, i.e. the nogood formed by taking the entire set of current decision assignments. On the other hand, lazy clause solvers [6] use what is called the *first unique implication point* (1UIP) nogood (described in Section 3), which has been empirically found to be much stronger than choice nogoods in terms of pruning strength as clausal propagators. As our theoretical exploration will show, this difference in pruning strength carries over to dynamic symmetry breaking methods. Combining lazy clause generation and dynamic symmetry breaking allows us to take advantage of 1UIP nogoods (as lazy evaluation does) and of symmetric 1UIP nogoods (rather than of symmetric choice nogoods, as dynamic symmetry breaking does). This leads to strictly more pruning.

2 Finite Domain Propagation

Let $\equiv$ denote syntactic identity and $vars(O)$ denote the set of variables of object O. We use $\Rightarrow$ and $\Leftrightarrow$ to denote logical implication and logical equivalence, respectively.

A *constraint problem* P is a tuple (C, D), where C is a set of constraints and D is a *domain* which maps each variable $x \in vars(C)$ to a finite set of integers $D(x)$. The set C is logically interpreted as the conjunction of its elements, while D is interpreted as $\wedge_{x \in vars(C)} x \in D(x)$. A variable x is said to be Boolean if $D(x) = [0, 1]$, where 0 represents *false* and 1 represents *true*.

An *equality literal* of $P \equiv (C, D)$ is of the form $x = d$, where $x \in vars(C)$ and $d \in D(x)$. A *valuation θ of P over set of variables $V \subseteq vars(C)$* is a set of equality literals of P with exactly one literal per variable in V. It can be understood as a mapping of variables to values. The *projection* of valuation θ over a set of variables $U \subseteq vars(\theta)$ is the valuation $\theta_U = \{x = \theta(x) | x \in U\}$.

A constraint $c \in C$ can be considered a set of valuations $solns(c)$ over the variables $vars(c)$. Valuation θ *satisfies* constraint c iff $vars(c) \subseteq vars(\theta)$ and $\theta_{vars(c)} \in c$. A *solution* of P is a valuation over $vars(P)$ that satisfies every constraint in C. We let $solns(P)$ be the set of all its solutions. Problem P is *satisfiable* if it has at least one solution and *unsatisfiable* otherwise.

An *inequality literal* for problem $P = (C, D)$ has the form $x \leq d$ or $x \geq d$ where $x \in vars(C)$ and $d \in D(x)$. A *disequality literal* for x has the form $x \neq d$ where $d \in D(x)$. The equality, inequality and disequality literals of P, together with the special literal *false* representing failure, are denoted the *literals* of P. Literals represent the basic changes in domain that occur during propagation.

A constraint c is implemented by a propagator f_c which is a function from domains to domains that ensures that $c \wedge D \Leftrightarrow c \wedge f_c(D)$. We can record the new information obtained by running f_c on domain D as the set of literals which are newly implied: $new(f_c, D) = \{l \mid D \not\Rightarrow l \wedge f_c(D) \Rightarrow l\}$. We will assume that we remove from this set literals that are redundant. Note that if the propagator detects failure we assume $new(f_c, D) = \{false\}$.

Example 1. Consider the actions of propagator f_c of constraint $c \equiv \sum_{i=1}^{5} x_i \leq 12$ on the domain $D(x_1) = \{1\}, D(x_2) = D(x_3) = D(x_4) = D(x_5) = [\,2\,..\,10\,]$. Now $D' = f_c(D)$ has $D(x_2) = D(x_3) = D(x_4) = D(x_5) = [\,2\,..\,5\,]$. Hence, as defined $new(f_c, D)$ includes $x_2 \geq 2, x_2 \leq 5, x_2 \leq 6, x_2 \leq 7, \ldots$. Since the second literal makes those following redundant, we assume they are not part of the result. □

Given a root constraint problem $P \equiv (C, D)$, constraint programming solves P by a search process that first uses a constraint solver to determine whether P can immediately be classified as satisfiable or unsatisfiable. We assume a propagation solver, denoted by solv, which when applied to P repeatedly applies propagators, updating the domain, until each returns an empty set of new literals. The final resulting domain D' is such that $D' \Rightarrow D$ and $C \wedge D \Leftrightarrow C \wedge D'$. The solver detects unsatisfiability if any $D'(x) = \emptyset$ for some $x \in vars(C)$. We assume that if the solver returns a domain D' where all variables are fixed then the solver has detected satisfiability of the problem and D' is a solution. If the solver cannot immediately determine whether P is satisfiable or unsatisfiable, the search splits P into n subproblems $P_i = (C \wedge c_i, D')$ where $C \wedge D' \Rightarrow (c_1 \vee c_2 \vee \ldots \vee c_n))$ and iteratively searches for solutions to them.

The idea is for the search to drive towards subproblems that can be immediately detected by solv as being satisfiable or unsatisfiable. This solving process implicitly defines a *search tree* rooted by the original problem P where each node represents a new (though perhaps logically equivalent) subproblem P', which will be used as the node's label. In this paper we restrict ourselves to the case where each c_i added by the search takes the form of a literal (referred to as a *decision* literal). While this is not a strong restriction, it does rule out some kinds of constraint programming search. We can identify any subproblem P' appearing in the search tree for $P = (C, D)$ by the set of decision literals $c_1, \ldots, c_n$ taken to reach P'. We define $choices(P') = C'$ where $P' = (C \cup C', D')$.

3 Lazy Clause Generation

Lazy clause generation [6] is a hybrid of finite domain and SAT solving where each FD propagator is extended to be able to explain its propagations. An integer variable x in problem $P = (C, D)$ with initial domain $D(x) = [\,l\,..\,u\,]$ is correlated to a set of Boolean variables $\{[\![x = d]\!] \mid l \leq d \leq u\} \cup \{[\![x \leq d]\!] \mid l \leq d < u\}$ that represent the domain changes possible for the variable (note that $[\![x \leq u]\!]$ is always true). Note that for variables x with initial domain $D(x) = [\,0\,..\,1\,]$ we can represent them using the single Boolean variable $[\![x = 1]\!]$. In order to prevent meaningless assignments to these Boolean variables we add Boolean constraints

to the constraint problem that define the conditions that relate them.

$$\begin{aligned} [\![x \leq d]\!] &\rightarrow [\![x \leq d+1]\!], && l \leq d \leq u-2 \\ [\![x \leq d]\!] \wedge \neg[\![x \leq d-1]\!] &\leftrightarrow [\![x = d]\!], && l < d \leq u-1 \\ [\![x \leq l]\!] &\leftrightarrow [\![x = l]\!] \\ \neg[\![x \leq u-1]\!] &\leftrightarrow [\![x = u]\!] \end{aligned}$$

Rather than directly using the Boolean variables attached to an integer variable we will use equality, inequality and disequality literals. Each equality, inequality and disequality literal can be considered as simply more explicit notation for a Boolean literal using the Boolean variables defined above:

$$\begin{aligned} x = d &\equiv [\![x = d]\!] & x \leq u &\equiv \mathit{true} \\ x \neq d &\equiv \neg[\![x = d]\!] & x \geq d &\equiv \neg[\![x \leq d-1]\!], l < d \\ x \leq d &\equiv [\![x \leq d]\!], d < u & x \geq l &\equiv \mathit{true}. \end{aligned}$$

For lazy clause generation, if a propagator f_c implementing constraint c infers a new literal (equality, inequality, or disequality) on the domain of one of its variables, or failure, it must explain this literal in terms of the Boolean representation of the variables involved in the propagator. An *explanation* for literal l is $S \rightarrow l$ where S is a set of literals. A correct explanation for l by f_c propagating on a problem with initial domain D, is an explanation $S \rightarrow l$ where $c \wedge S \wedge D \Rightarrow l$. Clearly, an explanation corresponds directly to a clause on the underlying Boolean representation. For example, the propagator for constraint $x \neq y$ may infer literal $y \neq 3$ given literal $x = 3$. This might be explained as $\{x = 3\} \rightarrow y \neq 3$ corresponding to the clause $[\![x = 3]\!] \rightarrow \neg[\![y = 3]\!]$.

In a lazy clause generation solver each new literal l inferred by a propagator f_c is recorded in a stack in the order of generation. Furthermore, the propagator returns an explanation for l that is attached to l. The *implication graph* is thus a stack of literals each with an attached explanation, or marked as a decision literal. We define the *decision level* for any literal as the number of decision literals pushed in before it in the stack.

Example 2. Consider the following constraint problem $P = (C, D)$ where $C \equiv \{\sum_{i=1}^{5} x_i \leq 12,\ \mathit{alldiff}([x_1, x_2, x_3, x_4, x_5])\}$ and $D(x_i) = [\,1\,..\,8\,], 1 \leq i \leq 5$. If the search chooses $x_1 = 1$ we arrive at subproblem $P_1 = (C \cup \{x_1 = 1\}, D)$. Then the *alldiff* constraint determines that $x_2 \neq 1$ from set of literals $\{x_1 = 1\}$ (i.e., with explanation $\{x_1 = 1\} \rightarrow x_2 \neq 1$), and similarly for x_3, x_4 and x_5. This builds the second column of the implication graph in Figure 1. Then, the domain constraints for x_2 determine that $x_2 \geq 2$ from $\{x_2 \neq 1\}$ and similarly for the domain constraints of x_3, x_4, and x_5, building the third column. The sum constraint determines that the upper bound of each of x_2, x_3, x_4 and x_5 is 5 from the lower bounds in the third column, thus building the fourth column. The new domain is $D'(x_1) = \{1\}$, $D'(x_i) = [\,2\,..\,5\,], 2 \leq i \leq 5$. If the search now chooses $x_2 = 2$, we arrive at subproblem $P_2 = (C \cup \{x_1 = 1, x_2 = 2\}, D')$. Then the *alldiff* constraint determines $x_3 \neq 2$, $x_4 \neq 2$, and $x_5 \neq 2$ (the 6th column) from $\{x_2 = 2\}$. The domain constraints determine that $x_3 \geq 3$ from $\{x_3 \geq 2, x_3 \neq 2\}$, similarly for x_4 and x_5. The sum constraint determines that $x_4 \leq 3$ from $\{x_2 = 2, x_3 \geq 3, x_5 \geq 3\}$, similarly for $x_5 \leq 3$. Then, the domain

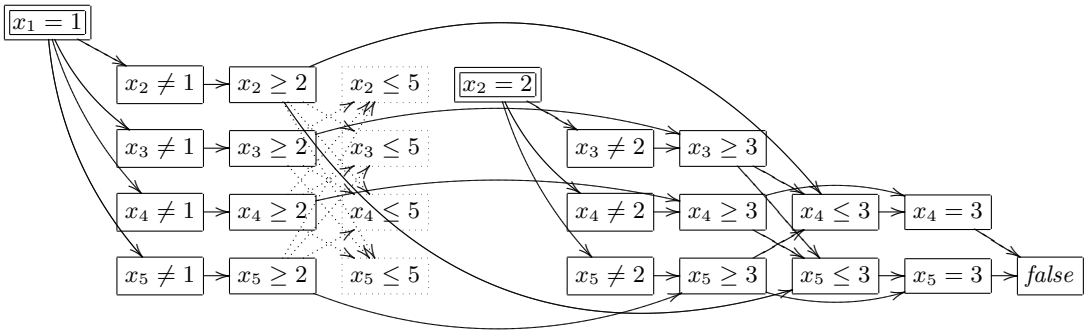


Fig. 1. Implication graph of propagation. Decision literals are double boxed.

constraints determine $x_4 = 3$ from $\{x_4 \geq 3, x_4 \leq 3\}$, similarly for $x_5 = 3$ and finally the *alldiff* constraint determines unsatisfiability of $x_4 = 3$ and $x_5 = 3$. □

A *nogood* N is set of literals. A correct nogood N from problem $P = (C, D)$ is one where $C \wedge D \Rightarrow \neg \wedge_{l \in N} l$, that is, in all solutions of P the conjunction of the literals in N is false. Once we have an implication graph we can use it determine a correct nogood that explains each failure. The usual approach to building a nogood is to use the implication graph to eliminate literals starting from original nogood N from the explanation of failure $N \rightarrow \mathit{false}$, until only one literal at the the current decision level remains. This is the 1UIP (First Unique Implication Point) nogood. The search then records this nogood as a clausal propagator and backtracks to the decision level of the second latest literal in the nogood, where it applies the newly derived nogood propagator.

Example 3. Continuing from Example 2 using the implication graph in Figure 1, we start with the explanation of failure $\{x_4 = 3, x_5 = 3\} \rightarrow \mathit{false}$ which gives us the initial nogood $\{x_4 = 3, x_5 = 3\}$. Since both literals were determined at the current decision level, we replace the last one $x_5 = 3$ by the antecedents in its explanation $\{x_5 \geq 3, x_5 \leq 3\} \rightarrow x_5 = 3$ to obtain $\{x_5 \geq 3, x_5 \leq 3, x_4 = 3\}$. We keep removing the last literal with the current decision level until only one literal remains at the current decision level. The resulting 1UIP nogood is $\{x_2 \geq 2, x_3 \geq 2, x_4 \geq 2, x_5 \geq 2, x_2 = 2\}$ which can be simplified to $\{x_3 \geq 2, x_4 \geq 2, x_5 \geq 2, x_2 = 2\}$, since the last literal implies the first.

On backtracking to undo the choice $x_2 = 2$, the search arrives at subproblem P_1 and immediately determines that $x_2 \neq 2$ using the new nogood. The important point is that if the search ever reaches a state where $\{x_3 \geq 2, x_4 \geq 2, x_5 \geq 2\}$ hold we will make the same inference, or indeed if it reaches a point where $\{x_2 = 2, x_3 \geq 2, x_4 \geq 2\}$ we will infer that $x_5 < 2$. □

In general, we restrict ourselves to creating nogoods which are *asserting*, that is, there should be only a single literal l in the nogood with the latest decision level. This allows us, upon backtracking to the decision level of the second latest literal, to *assert* $\neg l$ since the remaining literals are true. Another

possible asserting nogood generation approach is the so called *decision* nogood, where we start from the original explanation of failure and keep eliminating all literals which are not decision literals (that is, have an explanation). This builds much weaker nogoods in general than 1UIP nogoods.

Example 4. Nogood $\{x_2 \geq 2, x_3 \geq 3, x_4 \geq 3, x_5 \geq 3\}$ is correct for Example 2 but is not asserting since the last 3 literals belong to the latest decision level. The decision nogood for Example 2 is $\{x_1 = 1, x_2 = 2\}$. □

4 Symmetries and Nogoods

A *symmetry* of constraint problem $P = (C, D)$ is a bijection ρ on the equality literals of P such that, for each valuation θ of P, $\rho(\theta) = \{\rho(l) \mid l \in \theta\}$ is a solution of P iff θ is a solution of P. Variable symmetries, value symmetries and variable-value symmetries are all particular cases of symmetries.

Example 5. A variable symmetry ρ swapping variables x_1 and x_2 is defined as $\rho(x_1 = d) \equiv (x_2 = d)$, $\rho(x_2 = d) \equiv (x_1 = d)$, and $\rho(v = d) \equiv (v = d), v \notin \{x_1, x_2\}$ for all values d. We denote it $\ll x_1 \gg \leftrightarrows \ll x_2 \gg$. A value symmetry ρ swapping value 1 for 3 and 2 for 4 is defined by $\rho(v = 1) \equiv (v = 3)$, $\rho(v = 2) \equiv (v = 4)$, $\rho(v = 3) \equiv (v = 1)$, $\rho(v = 4) \equiv (v = 2)$, $\rho(v = d) \equiv (v = d), d \notin \{1, 2, 3, 4\}$ for all variables $v \in vars(P)$. We denote it $\ll 1, 2 \gg \leftrightarrows \ll 3, 4 \gg$. □

Static Symmetry Breaking effectively reduces the search required to find the first, all or the best solution to a constraint problem by adding constraints that remove symmetric solutions. In particular, lexicographical constraints have been used to statically eliminate symmetries (see e.g.[2]) with excellent results. This is good news since static symmetry breaking is obviously compatible with lazy clause generation: we only require the new symmetry breaking constraints to have explaining propagators, which are used just like other propagators. However, static symmetry breaking is not always the best option. If we have multiple symmetries, care must be taken so that the static symmetry breaking constraints for each symmetry do not interact badly. Also, static symmetry breaking constraints may interact badly with a given search strategy, making the search take even longer to find a solution. Hence, we are also interested in dynamic symmetry breaking methods.

Dynamic Symmetry Breaking techniques can be interpreted as pruning symmetric portions of the search space by propagating symmetric versions of nogoods. Consider a search strategy where only equality literals are used to split search, as it is usual for symmetry papers. Then if subproblem P' fails $choices(P')$ is a correct nogood of P. Let us denote this as the *choice nogood*. Since a generated nogood N is a globally true statement, it holds at any point during the search and, hence, any symmetric version of N is also a correct nogood. Note that the symmetric version, $\rho(N)$, of a nogood N consisting of only equality literals is easy to define: $\rho(N) = \{\rho(l) \mid l \in N\}$.

Example 6. In problem P of Example 2 the variables $\{x_1, x_2, x_3, x_4, x_5\}$ are indistinguishable (i.e., any two can be swapped). Since the subproblem P' with

$choices(P') = \{x_1 = 1, x_2 = 2\}$ fails, we have that $\{x_1 = 1, x_2 = 2\}$ is a correct nogood for P. Clearly, any symmetric version, such as $\{x_2 = 1, x_1 = 2\}$ or $\{x_3 = 1, x_5 = 2\}$, is also a correct nogood. □

Such nogoods can be used to prune search in two main ways. Symmetry breaking by dominance detection (SBDD) [3, 1] keeps a store $\mathbf{N}$ of the non-subsumed choice nogoods derived during search so far. For each subproblem P', it checks whether there exists $N \in \mathbf{N}$ and symmetry ρ, such that $choices(P') \Rightarrow \rho(N)$. If such a pair exists it can immediately fail subproblem P'. Symmetry breaking during search (SBDS) [4] works as follows. Whenever a subproblem P' with $choices(P') = \{d_1, d_2, \ldots, d_n, d_{n+1}\}$ fails, SBDS backtracks to the parent suproblem P'' in level n and, for each symmetry ρ, it locally posts in P'' the conditional constraint $(\rho(d_1) \wedge \ldots \wedge \rho(d_n)) \rightarrow \neg\rho(d_{n+1})$. Note that these constraints will only propagate when reaching a subproblem P''' such that $C \cup choices(P''')$ entails the left hand side of the constraint. This will never happen if the symmetry is *broken*, i.e., if $\exists d_i$ s.t. $\neg\rho(d_i)$ is entailed, and that is why SBDS ignores any symmetry ρ which is known to be broken at P''. Still, SBDS can post too many local constraints when the number of symmetries is high. Thus, some incomplete methods ([5] and the shortcut method in [4]) post only those constraints that are known to immediately propagate.

We decided to integrate SBDS, rather than SBDD, with our lazy clause generation since SBDS is much closer to the lazy clause generation approach: they both compute and post nogoods. The main differences being that SBDS only computes decision nogoods and posts symmetric versions of these nogoods.

5 Symmetries and Lazy Clause Generation

5.1 SBDS-choice

We can naively add SBDS to a lazy clause solver by simply using symmetric versions of the choice nogood at each node to prune off symmetric branches. Hence, we just reimplement standard SBDS in the lazy clause generation solver, but still gain the advantage of reduced search through the lazy clause generation nogoods.

5.2 SBDS-1UIP

Adapting SBDS to use 1UIP nogoods is simple: every time a 1UIP nogood $\{l_1, \ldots, l_n\} \rightarrow l_{n+1}$ is inferred for subproblem P', upon backtracking to parent P'' and for each symmetry ρ, we post the symmetric nogood $\{\rho(l_1), \ldots, \rho(l_n)\} \rightarrow \neg\rho(l_{n+1})$, ignoring those ρ that are known to be broken at P''. We can check this last condition during the construction of the symmetric nogood, as we produce the literals $\rho(l_1), \ldots, \rho(l_n)$ one at a time. If at any point, one of $\rho(l_i)$ is false in P'', we can immediately abort and move on to the next symmetry.

In contrast to SBDS-choice, in SBDS-1UIP we have to post the symmetric nogoods as global rather than local constraints. This is because in SBDS-choice, when you backtrack from parent P'' to grandparent P''', the choice nogood at P''' subsumes that at P'' and, therefore, SBDS-choice will always post a set of

symmetric nogoods that subsumes the symmetric nogoods posted below that point. In contrast, there is no guarantee that the 1UIP nogood at P''' subsumes the one at P'' (and in general it doesn't).

Example 7. Consider the problem of Example 2. On backtracking to P_1 we infer the nogood $\{x_3 \geq 2, x_4 \geq 2, x_5 \geq 2\} \rightarrow x_2 \neq 2$. With this we not only infer $x_2 \neq 2$ but also the symmetric inferences $x_3 \neq 2$ (from $\{x_2 \geq 2, x_4 \geq 2, x_5 \geq 2\}$), $x_4 \neq 2$ and $x_5 \neq 2$. At this point, a domain consistent *alldiff* will determine unsatisfiability, and generate the nogood $\emptyset \rightarrow x_1 \neq 1$, which does not imply the previously generated nogood. □

We show that SBDS-1UIP exploits strictly more symmetries than SBDS-choice if the asserting literals in the nogoods are the same, and propagation has the following property:

Definition 1. *A set of propagators for problem P has* global symmetric monotonicity *iff, for any explanation $\{d_1, \ldots, d_n\} \rightarrow l$ produced and any symmetry ρ of P, whenever $\rho(d_1), \ldots, \rho(d_n)$ are entailed, then $\rho(l)$ must also be entailed.* □

A sufficient condition for global symmetric monotonicity is the following: all propagators are monotonic, and all symmetries are propagator symmetric (propagators map to propagators under the symmetry). The proof of this is straightforward and we omit it for lack of space. Global symmetric monotonicity is therefore very common, as most propagators are monotonic, and the vast majority of symmetries that are usually exploited are propagator symmetric.

Theorem 1. *Suppose global symmetric monotonicity holds, and we derive the choice nogood $\{d_1, \ldots, d_n\} \rightarrow \neg d_{n+1}$, and the 1UIP nogood $\{l_1, \ldots, l_m\} \rightarrow \neg d_{n+1}$ from the same conflict. If the nogood $\{\rho(d_1), \ldots, \rho(d_n)\} \rightarrow \neg\rho(d_{n+1})$ propagates then so does $\{\rho(l_1), \ldots, \rho(l_m)\} \rightarrow \neg\rho(d_{n+1})$.*

Proof. Suppose the symmetric version of the choice nogood can propagate for domain D'. Then $\rho(d_1), \ldots, \rho(d_n)$ must all be entailed. From the implication graph from which we derived the 1UIP nogood, we know that $d_1 \wedge \ldots \wedge d_n \Rightarrow l_i$ for any i. Since global symmetric monotonicity holds and $\rho(d_1), \ldots, \rho(d_n)$ are entailed, we know that $\rho(l_i)$ must also be entailed for any i. This means that the symmetric version of the 1UIP nogood can also propagate. □

The theorem shows that the symmetric 1UIP nogood subsumes the symmetric choice nogood, since it will always produce any implication that the symmetric choice nogood can, but not vice versa. This means that SBDS-1UIP can exploit strictly more symmetry than SBDS-choice. This result is valid for both complete SBDS, as well as for incomplete SBDS methods which only post nogoods that will immediately produce an implication.

5.3 Beyond complete methods?

The previous result is somewhat surprising considering that SBDS-choice is a "complete" symmetry breaking method, which guarantees that once we have examined a certain partial assignment, we will never examine any symmetric

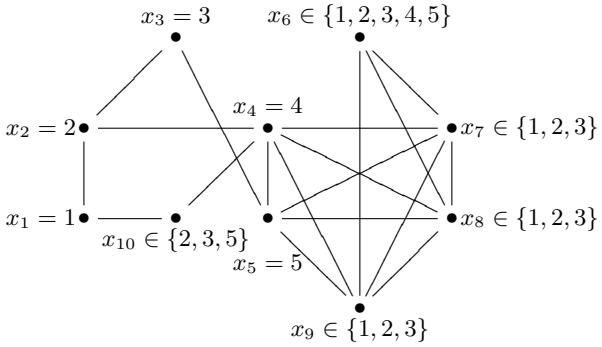


Fig. 2. A graph colouring problem where we can exploit additional symmetries

version of it. However, this does not actually mean that we have exploited all possibly redundancies arising from symmetry. Roughly speaking, SBDS can only exploit symmetries on the already labelled parts of the problem. It is incapable of exploiting symmetries in the unlabelled parts of the problem.

Example 8. Consider the graph colouring problem shown in Figure 2, where we are trying to colour the nodes with at most 5 colours (all of which are interchangeable). After making the decisions $x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 4, x_5 = 5$, we have domains as shown in Figure 2. Suppose we label $x_6 = 1$ next. Then propagation gives $x_7 \in \{2,3\}, x_8 \in \{2,3\}, x_9 \in \{2,3\}$. Now, suppose we try $x_7 = 2$. This forces $x_8 = 3, x_9 = 3$, which conflicts. The 1UIP nogood from this conflict is $\{x_8 \neq 1, x_8 \neq 4, x_8 \neq 5, x_9 \neq 1, x_9 \neq 4, x_9 \neq 5\} \rightarrow x_7 \neq 2$. After propagating this nogood, we have $x_7 = 3$, which after further propagation, once again conflicts. At this point, we backtrack to before x_6 is labelled and derive the nogood $\{x_7 \neq 4, x_7 \neq 5, x_8 \neq 4, x_8 \neq 5, x_9 \neq 4, x_9 \neq 5\} \rightarrow x_6 \neq 1$.

Now, let's examine what SBDS-1UIP can do at this point. It is clear that if we apply the value symmetries $\ll 1 \gg \leftrightarrows \ll 2 \gg$ or $\ll 1 \gg \leftrightarrows \ll 3 \gg$ to this nogood, the LHS remains unchanged while the RHS changes. Therefore, we can post these two symmetric nogoods and immediately get the inferences $x_6 \neq 2$ and $x_6 \neq 3$. On the other hand, SBDS-choice can't do anything. The choice nogood is $\{x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 4, x_5 = 5\} \rightarrow x_6 \neq 1$, and it is easy to see that no matter which value symmetry we use on it, the LHS will have a set of literals incompatible with the current set of decisions and thus cannot imply the RHS. □

The kind of redundancy we exploit here certainly arises from symmetry. However, it is extremely difficult to exploit. Roughly speaking, we can say that we are exploiting the symmetry that exists in the sub-component of a subproblem which is the actual cause of failure. In this case, they are the variables x_6, x_7, x_8, x_9, their current domains in the subproblem, and the constraints linking them. Even conditional symmetry breaking constraints are powerless to exploit such symmetries, as the subproblem shown in Figure 2 does not have the value symmetries $\ll 1 \gg \leftrightarrows \ll 2 \gg$ or $\ll 1 \gg \leftrightarrows \ll 3 \gg$ due to the existence of x_{10}. It is only because

a lazy clause solver gives us such precise information about which variables are involved in failures that we can exploit this kind of redundancy. Although the above example might seem somewhat contrived, we show in our experiments in Section 7 that these kinds of redundancies do occur in practice and can be exploited for more speedup.

6 Symmetries on 1UIP nogoods

SBDS-1UIP is much more powerful than SBDS-choice, however, having to manipulate 1UIP nogoods raises a whole host of other problems. In particular, unlike the choice nogoods which usually only involve equality literals on search variables, 1UIP nogoods can contain virtually any literal in the problem, i.e. they may include disequality literals, inequality literals, and also literals involving intermediate variables. The last is a rather serious issue as symmetries are often defined in terms of the output or search variables, and may not properly describe how intermediate variables map to each other. We now examine each of these issues in more detail.

6.1 Disequality and Inequality Literals

One of the strengths of lazy clause generation is the use of both equality literals and inequality literals in explanations and nogoods. This makes many explanations much shorter and is effectively essential for explaining bounds propagation.

Extending a literal symmetry ρ to disequality literals is straightforward: if $\rho(x = d) \equiv x' = d'$, then $\rho(x \neq d) \equiv x' \neq d'$. Extending a literal symmetry ρ to map inequality literals is harder. Given a nogood N which may involve equality, inequality and disequality literals and a *variable* symmetry σ it is easy to generate $\sigma(N)$ by simply applying the variable renaming σ to N.

Example 9. Consider the problem of Example 2. This problem has variable interchangeability symmetries since each of $\{x_1, x_2, x_3, x_4, x_5\}$ are indistinguishable. Hence, any variable renaming of the generated nogood $\{x_3 \geq 2, x_4 \geq 2, x_5 \geq 2, x_2 = 2\}$ is valid, e.g. $\{x_2 \geq 2, x_3 \geq 2, x_4 \geq 2, x_5 = 2\}$ or $\{x_1 \geq 2, x_3 \geq 2, x_5 \geq 2, x_4 = 2\}$ □

However, it's not so straightforward for value symmetries and variable-value symmetries. For such symmetries, inequality literals do not map simply to other literals. To apply such a symmetry to a nogood then, we first need to transform the nogood into an equivalent nogood involving only equality and disequality literals. After this we can apply the symmetry as usual.

Assume that in $P = (D, C)$ that $D(x) = [\, l \,..\, u \,]$ then the transformation eq on x literals is defined as

$$\mathsf{eq}(x = d) \equiv \{x = d\} \qquad \mathsf{eq}(x \neq d) \equiv \{x \neq d\}$$
$$\mathsf{eq}(x \leq d) \equiv \{x \neq d' \mid d \leq d' < u\} \qquad \mathsf{eq}(x \geq d) \equiv \{x \neq d' \mid l < d' \leq d\}$$

We can extend this to a nogood N: $\mathsf{eq}(N) = \cup_{l \in N} \mathsf{eq}(l)$. We can then define the symmetric version of a nogood N for any symmetry ρ defined as a bijection on

equality literals as $\rho(N) = \{\rho(l) \mid l \in \mathsf{eq}(N)\}$ Of course while this transformation to equality and disequality literals is theoretically fine, in practice it may create very unwieldy nogoods.

We can, in effect, implement this transformation by slightly modifying the nogood learning process. We will require that no inequality literals appear in the nogood, hence we must continue to explain them until none remain. As long as all decisions are either equality or disequalities this will still result in asserting nogoods always being discovered.

Example 10. Revisiting the explanation process of Example 3, the nogood discovered includes inequality literals, so rather than stopping the explanation process at this point we continue. The current nogood is $\{x_2 \geq 2, x_3 \geq 2, x_4 \geq 2, x_2 = 2\}$, we explain each of the inequalities using the implication graph to arrive at $\{x_2 \neq 1, x_3 \neq 1, x_4 \neq 1, x_2 = 2\}$ which can again be simplified to $\{x_3 \neq 1, x_4 \neq 1, x_2 = 2\}$ and which does involve inequality literals. □

6.2 Intermediate variables

An important problem for combining dynamic symmetry breaking and lazy clause generation is the fact that intermediate variables may be introduced in the course of converting a high level model to the low level variables and constraints implemented by the solver. For example, a high level model written in the modeling language MiniZinc is first flattened into primitive constraints, with intermediate variables introduced as necessary, and then given to a solver, which may then introduce its own variables, e.g. in global propagators implemented by decomposition. 1UIP nogoods often contain literals from such intermediate variables. However, if the symmetry declaration was made only in the high level model, it may not specify how literals on such introduced intermediate variables map to each other. Thus it is necessary to consider how symmetries can be extended to include the literals on intermediate variables.

Intermediate variables are sometimes idempotent under the symmetries, that is for each symmetry ρ of P, we can extend ρ to ρ' where $\rho'(l) = \rho(l), \mathit{vars}(l) \subseteq \mathit{vars}(C)$ and $\rho'(l) = l$ otherwise. The extended ρ' is a symmetry of the problem with intermediate variables. We can imagine automating the proof of idempotence of intermediate variables under symmetries.

Example 11. Consider a model for concert hall scheduling The problem has a value interchangeability between all values $[1..k]$ for the k identical concert halls. The model includes the constraint

```
constraint forall (i, j in Offers where i < j /\ o[i,j])
    (x[i] = k+1 \/ x[j] = k+1 \/ x[i] != x[j]);
```

which requires that for two overlapping concerts i and j (input data $o[i, j]$ is *true*) either i is not scheduled (represented as the hall used $x[i]$ is $k+1$), j is not scheduled, or the halls used are different. But this constraint is implemented, by reification, as something equivalent to

```
array[Offers]  of var bool: unscheduled;
array[Offers,Offers] of var bool: different;
```

```
constraint forall(i in Offers)(unscheduled[i] = (x[i] = k+1));
constraint forall (i, j in Offers where i < j /\ o[i,j])(
    different[i,j] = (x[i] != x[j]) /\
    (unscheduled[i] \/ unscheduled[j] \/ different[i,j]));
```

since the clausal propagator works on Boolean variables, and hence we need to reify the subexpressions. Each introduced variable $unscheduled[i]$ and $different[i, j]$ is idempotent under the value symmetries. Hence, for any symmetry ρ on the original variables we can extend it trivially. □

Sometimes we need to extend our symmetry declarations to take into account the intermediate variables.

Example 12. In the graceful graph problem each node is labelled by an number from 0 to the number of edges. The difference between each edges node labels must be different. This is encoded as

```
constraint alldifferent([ abs(m[o[i]] - m[d[i]]) | i in Edges]);
```

where m is the labelling on nodes, and $o[i], d[i]$ are the origin and destination of edge i. This constraint is implemented by flattening as something equivalent to

```
array[Edges] of var int: diff;
constraint forall(i in Edges)(diff[i] == m[o[i]]  - m[d[i]]);
srray[Edges] of var int: adiff;
constraint forall(i in Edges)(adiff[i] == abs(diff[i]));
constraint alldifferent(adiff);
```

The graceful graph problem can have symmetries arising from symmetries in the underlying graph. Suppose the underlying graph has 3 nodes and 2 edges (1,2) and (3,2) numbered 1 and 2. There is a symmetry between the two edges captures by the row interchangeability $\rho = \ll m[1], m[2] \gg \leftrightarrows \ll m[3], m[2] \gg$ which indicates we can swap the edges.

Once we consider the intermediate variables, we need to extend this symmetry to $\ll m[1], m[2], \mathit{diff}[1], \mathit{adiff}[1] \gg \leftrightarrows \ll m[3], m[2], \mathit{diff}[2], \mathit{adiff}[2] \gg$ thus interchanging all information on about the edges simultaneously. □

Sometimes it is not easy to see how to extend symmetries to all intermediate variables, and indeed quite often intermediate variables are introduced far below the modelling level. In order to handle these cases we modify learning as follows.

We extend the model to explicitly mark which literals are allowed to appear in nogoods. Then we modify the learning process to always explain any literals that are not marked. There is a requirement that all literals generated by search are allowed to appear in nogoods. This ensures that the process always terminates and always generates an asserting nogood.

Example 13. In order to tell the solver that the concert hall scheduling model that it can use only equality and disequality literals for the x variables as well as the the intermediate variables in nogoods we annotate the declarations:

```
array[Offers] of var 1..k+1:x :: symmetric_nogoods_eq;
array[Offers] of var bool: unscheduled :: symmetric_nogoods;
array[Offers,Offers] of var bool: different :: symmetric_nogoods;
```

The declarations ensure that any other literal will never appear in a nogood to which we apply symmetry. Note that his means that bounds literals $x[i] \leq d$ will be replaced by inequality literals. □

7 Experiments

We now provide experimental evidence for the claims we made in the earlier parts of the paper. The two problem we will examine are the Concert Hall Scheduling problem and the Graph Colouring problem. We take the benchmarks used by [5].The benchmarks are available at `http://www.cmears.id.au/symmetry/symcache.tar.gz`.

We implemented SBDS in Chuffed, which is a state of the art lazy clause solver. We run Chuffed with three different versions of SBDS. The first version is choice, where we use symmetric versions of choice nogoods. The second is 1UIP, where we use symmetric versions of 1UIP nogoods. The third version we call crippled, where we use symmetric versions of 1UIP nogoods, but only those nogoods derived from symmetries where choice could also exploit the symmetry. We compare against Chuffed with no symmetry breaking (none) and with statically added symmetry breaking constraints (static). Finally, we compare against an implementation of SBDS in [5], which is called Lightweight Dynamic Symmetry Breaking (LDSB). LDSB is implemented on the Eclipse constraint programming platform and was the fastest implementations of dynamic symmetry breaking on the two problems we examine, beating GAP-SBDS and GAP-SBDD by significant margins.

All versions of Chuffed are run on Xeon Pro 2.4GHz processors. The results for LDSB were run on an Core i7 920 2.67 GHz processor. We group the instances by size, so that the times displayed are the average run times for the instances of each size. A timeout of 600 seconds was used. Instances which timeout are counted as 600 seconds.

The results are shown in Tables 1 and 2. Eclipse LDSB fails to solve many instances before timeout, and choice fails to solve a few instances. 1UIP, crippled and static all solve every instance in the benchmarks. In fact, this set of instances, which is of an appropriate size for normal CP solvers, is a bit too easy for lazy clause solvers such as Chuffed, as is apparent from the run times.

Comparison between choice and 1UIP shows that SBDS-1UIP is superior to SBDS-choice. Comparison between crippled and 1UIP shows that the additional symmetries that we can only exploit with SBDS-1UIP indeed gives us reduced search and additional speedup. Comparison with static shows that dynamic symmetry breaking can be superior to static symmetry breaking on appropriate problems. The comparison with LDSB shows that lazy clause solvers can be much faster than normal CP solvers, and that they retain this advantage when integrated with symmetry breaking methods. It also show by proxy that SBDS-1UIP is superior to GAP-SBDS or GAP-SBDD on these problems.

The total speed difference between 1UIP and LDSB is up to 2 orders of magnitude for the Concert Hall problems and up to 4 orders of magnitude for the Graph Colouring problems. Most of this speedup can be explained by the dramatic reduction in search space, which is apparent from the node counts in the

Table 1. Comparison of three SBDS implementations in Chuffed, static symmetry breaking in Chuffed, and LDSB in Eclipse, on the Concert Hall Scheduling problem

Size	none		1UIP		crippled		choice		static		LDSB	
	Time	Fails	Time	Fails	Time	Fails	Time	Fails	Time	Fails	Time	Nodes
20	259.8	686018	0.04	84	0.05	130	0.07	350	0.05	134	0.29	3283
22	381.5	749462	0.07	181	0.08	299	0.17	1207	0.07	183	0.73	7786
24	576.9	1438509	0.10	275	0.11	316	0.78	3426	0.15	486	2.70	12611
26	483.4	1189930	0.10	282	0.19	677	2.26	5605	0.25	685	2.71	12724
28	530.7	1282797	0.68	1611	1.12	2613	3.64	10530	0.42	1041	9.94	57284
30	581.3	1251980	0.27	761	0.53	2042	19.52	48474	0.52	2300	121.50	722668
32	–	–	0.40	1522	1.01	4845	21.48	65157	1.31	5712	97.90	641071
34	–	–	1.10	2636	3.22	8761	19.86	48837	1.60	4406	72.73	425718
36	–	–	1.40	3156	5.02	13606	59.70	131142	2.37	5707	171.14	938439
38	–	–	1.91	5053	12.56	26556	82.77	178170	3.51	10518	268.05	1211086
40	–	–	2.96	6648	10.27	27028	102.1	219454	6.40	18169	240.84	1220934

Table 2. Comparison of three SBDS implementations in Chuffed, static symmetry breaking in Chuffed, and LDSB in Eclipse, on the Graph Colouring problems

Uniform

Size	none		1UIP		crippled		choice		static		LDSB	
	Time	Fails	Time	Fails	Time	Fails	Time	Fails	Time	Fails	Time	Nodes
30	140.7	282974	0.00	14	0.06	474	0.26	3049	0.02	277	19.63	56577
32	211.4	390392	0.00	17	0.00	146	0.24	3677	0.00	84	14.11	27178
34	213.9	272772	0.00	25	0.29	1182	3.53	11975	0.03	433	22.06	30127
36	–	–	0.00	36	0.04	467	6.91	23842	0.01	200	35.66	85505
38	–	–	0.00	55	0.04	516	23.55	69480	0.03	526	51.18	107574
40	–	–	0.00	83	0.31	1879	21.07	78918	0.06	878	84.16	185707

Biased

Size	none		1UIP		crippled		choice		static		LDSB	
	Time	Fails	Time	Fails	Time	Fails	Time	Fails	Time	Fails	Time	Nodes
20	13.25	39551	0.00	27	0.00	32	0.01	639	0.00	29	0.72	1376
22	11.53	63984	0.00	25	0.00	34	0.02	727	0.00	25	0.16	538
24	66.60	154409	0.00	35	0.00	47	0.07	1992	0.00	32	1.91	2114
26	74.77	277290	0.00	55	0.00	93	0.12	3385	0.00	104	9.56	34210
28	130.5	280649	0.00	62	0.00	84	0.58	6402	0.00	103	9.14	37738
30	267.6	480195	0.00	101	0.01	239	10.48	43835	0.01	359	57.18	215932
32	331.7	600772	0.01	232	0.24	1597	9.98	44216	0.16	1864	110.16	288707
34	219.9	387213	0.20	806	0.40	1946	10.26	47470	0.45	1730	64.14	165943
36	–	–	0.01	317	0.04	857	27.39	113252	0.80	3226	152.62	327472
38	–	–	0.10	798	1.01	5569	31.63	138787	4.12	9413	193.40	508164
40	–	–	0.02	410	0.36	2660	24.68	91847	0.12	2133	196.02	476486

results table. The redundancies exploited by lazy clause solvers are different from those redundancies caused by symmetries, and it is very clear here that by exploiting both at the same time with SBDS-1UIP, we get much higher speedups than possible with either of them separately. It should also be noted that Chuffed with static symmetry breaking constraints is also reasonably fast. While symme-

try breaking constraints cannot exploit the extra redundancies that SBDS-1UIP can, it does have very low overhead and integrates well with lazy clause.

8 Conclusion

In this paper we have examined dynamic symmetry breaking methods, and understood them as manipulating the choice nogoods created by normal depth first search. We show how we can extend these approaches to make use of the better nogoods generated by lazy clause solvers. This extension introduces a number of new issues, such as how to deal with disequality and inequality literals, and literals from intermediate variables. We have built a prototype implementation combining SBDS with lazy clause generation, which we call SBDS-1UIP. The resulting system can exploit types of redundancies previously impossible to exploit, and can outperform LDSB by several orders of magnitudes on some problems.

Acknowledgements. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council.

References

1. T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In *Principles and Practice of Constraint Programming - CP 2001, 7th International Conference*, pages 93–107, 2001.
2. Pierre Flener, Alan M. Frisch, Brahim Hnich, Zeynep Kiziltan, Ian Miguel, Justin Pearson, and Toby Walsh. Breaking row and column symmetries in matrix models. In *Principles and Practice of Constraint Programming - CP 2002, 8th International Conference*, pages 462–476, 2002.
3. F. Focacci and M. Milano. Global cut framework for removing symmetries. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, pages 77–92, 2001.
4. I. Gent and B.M. Smith. Symmetry breaking in constraint programming. In *14th European Conference on Artificial Intelligence*, pages 599–603, 2000.
5. C. Mears. *Automatic Symmetry Detection and Dynamic Symmetry Breaking for Constraint Programming*. PhD thesis, Clayton School of Information Technology, Monash University, 2010.
6. O. Ohrimenko, P.J. Stuckey, and M. Codish. Propagation via lazy clause generation. *Constraints*, 14(3):357–391, 2009.
7. A. Schutt, T. Feydy, P.J. Stuckey, and M. Wallace. Why cumulative decomposition is not as bad as it sounds. In I. Gent, editor, *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming*, volume 5732 of *LNCS*, pages 746–761. Springer-Verlag, 2009.

Arities of Symmetry Breaking Constraints

Tim Januschowski*

Cork Constraint Computation Centre
Computer Science Department
University College Cork Ireland
janus@cs.ucc.ie

Abstract. Static symmetry breaking is a well-established technique to speed up the solving process of symmetric Constraint Satisfaction Programs (CSPs). Static symmetry breaking suffers from two inherent problems: symmetry breaking constraints come in great numbers and are of high arity. Here, we consider the problem of high arity. We prove that not even for binary CSPs can we always reduce the arity of the commonly used lexleader constraints. We further prove that for binary CSPs we sometimes have to rely on at least ternary constraints to break all symmetries. On the positive side, we prove that symmetry breaking constraints with arity $\lfloor n/2 \rfloor + 1$ exist that always break the symmetries of a CSP completely. For various special cases of CSPs, we prove that binary symmetry breaking constraints may break all symmetries.

1 Introduction

Symmetry breaking has been the subject of intense investigation for almost two decades in Constraint Programming, see e.g. [5]. Symmetry breaking has an empirically proved potential to speed up constructive search methods. The classic and practically most used technique in symmetry breaking is the addition of symmetry breaking constraints before search [2, 11].

For complete symmetry breaking, we have to add one lexleader constraint (LLC) [2] per symmetry, and the arity of each symmetry breaking constraint is the number of variables. In the worst case, a CSP can have an exponential number of symmetries. Adding an exponential number of constraints to a CSP is prohibitively costly. High arity slows down propagation [3]. Various remedies have been proposed: for special symmetry groups, we can find polynomial sized sets of constraints with reasonable arity that break all symmetries [6], or sometimes one can use the problem structure in combination with the symmetries to reduce the number and arity of the constraints [8, 12]. Also, various reduction rules [4, 6, 10] are known that reduce both the number and the arity of LLCs while maintaining the ability to completely break symmetries. Another remedy that we do not want to consider here is partial symmetry breaking.

* Tim Januschowski is supported by the Embark initiative of the Irish Research Council for Science, Engineering and Technology.

In this paper, we consider upper and lower bounds on the arities of symmetry breaking constraints for complete symmetry breaking in binary CSPs. Our lower bounds hold for *any* symmetry breaking constraint and *any* reduction rule. We prove the following for CSPs with n variables:

upper bound (i): the arity of LLCs cannot always be reduced, even if we assume the CSP to be binary,
upper bound (ii): symmetry breaking constraints of arity $\lfloor n/2 \rfloor + 1$ exist that completely break the symmetries of any CSP,
lower bound: for complete symmetry breaking in binary CSPs, we sometimes have to rely on ternary constraints, and
special cases: we identify various special cases, where binary symmetry breaking constraints suffice for complete symmetry breaking.

To the best of our knowledge, these results are new. We think that our results are an important theoretical contribution to the study of symmetry breaking constraints. Our results could be applied to study the optimality of reduction rules in terms of the arity of symmetry breaking constraints. If a reduction rule can reduce the arity of symmetry breaking constraints for all binary CSPs to at most three, the reduction rule could be considered optimal. On the other hand, if the reduction rule does not succeed to reduce the arity of symmetry breaking constraints below $\lfloor n/2 \rfloor + 1$ for all CSPs, then the rule is surely not optimal.

2 Notation and Definitions

A *constraint satisfaction problem* CSP is a triple $(V, D, Cons)$, where V is the set of variables of the CSP, every variable x has a *domain* $D(x) \in D$, and $Cons$ is the set of constraint of the CSP. Every constraint has an *arity*. The k-ary constraint c is a pair $\langle s, r \rangle$, where s is a list of k variables $x_1, \ldots, x_k$ which is called the *scope* and $r \subseteq D(x_1) \times \cdots \times D(x_k)$ is called *the relation* of c consisting of the tuples that c allows. The arity of a CSP is the maximum arity over all constraints in the CSP. A CSP is called *binary* if all constraints are of arity at most 2. A *literal* is a (variable,value)-assignment. A *partial assignment* is a set of literals in which no variable appears twice. If a partial assignment is allowed by the constraints of the CSP we call it *consistent*. A *solution* is a consistent assignment on all variables. If a CSP has a solution, the CSP is *satisfiable*, otherwise it is *unsatisfiable*.

We associate to any CSP a hypergraph called the *microstructure complement* (MSC), see e.g. [1]. The MSC has as nodes the literals of the CSP. We have a hyperedge between every set of literals that is not contained in the relation of a constraint of the CSP. For a binary CSP, the MSC is a graph and for binary CSPs we define the *microstructure* as the complement graph of the MSC. The *constraint symmetries* [1] of a CSP are the automorphisms of the MSC. Symmetries partition the set of solutions of a CSP into a set of equivalences classes or *orbits*. For a solution T, we denote by orbit(T) the set of solutions that are symmetric to T. Apart from constraint symmetries, other symmetries exist as well, notably solution symmetries. However, constraint programmers work with

constraint symmetries mostly [1]. Here, we only consider constraint symmetries which we shall abbreviate to symmetries.

Given a CSP P, a *valid reduction* P' [7, 11] is a CSP on the same variables, subsets of the domains and supersets of the constraints of P, such that for every orbit of solutions in P, at least one solution in P' exists. Among the solutions in P' that exist for every orbit of solutions in P, we can choose one as an *orbit-representative solution* per orbit in P. A *single-representative valid reduction* (SRVR) is a valid reduction that has *exactly* one solution per orbit of solutions in P. If P' is a SRVR, the set of solutions of P' is a *complete* set of orbit representative solutions.

We call members of a family of constraints *symmetry breaking constraints*, if adding the family to a CSP leads to a valid reduction. Lexleader constraints (LLCs) [2, 13] are well-known symmetry breaking constraints. To construct a LLC for a symmetry ϕ of a CSP, we choose a variable order, say $x_1 \prec x_2 \prec \cdots \prec x_n$ and an order on the domains. The LLC enforces that any assignment on the variables is lexicographically less than its symmetric counterpart with respect to the chosen orders:

$$([x_1, \ldots, x_n]) \leq_{lex} \phi([x_1, \ldots, x_n])\,.$$

We note that ϕ may not map a complete assignment to another complete assignment. In this case however, the assignment under consideration cannot be a solution and must be forbidden by some constraints of the CSP under consideration, otherwise ϕ would not be a symmetry.

SRVRs exist for any CSP and we can always find a SRVR using LLCs. With LLCs, the choice of orbit representative solutions depends on the order of the variables that we choose to define the LLCs as well as the orders on the domains. However, also for arbitrary choices of orbit representatives, a SRVR always exists: we could simply add an n-ary constraint forbidding any non-representative solution [7]. If the arity of the constraints we add to a CSP to obtain the SRVR is at most k, we say that the SRVR is *k-ary realisable*.

In the context of this paper, we consider the *full* group of (constraint) symmetries. Our intuition is that the full group of symmetries is more closely connected to the MSC than *sub*groups. Results obtained for subgroups may not hold for the full group as orbits of solutions tend to interact in a less complicated way for smaller groups. Indeed, the number of orbits of solutions typically decreases for increasing sizes of subgroups. This also helps to explain, why it seems more difficult to obtain results for the full group of symmetries than for subgroups. In this paper, we investigate *complete* symmetry breaking, or rather the existence of SRVRs.

3 Related Work

This work is based on Puget's systematic approach to introduce symmetry breaking constraints via valid reductions [11]. This approach was generalised and extended in [7]. There, we also posed the question, whether for any k-ary CSP, there always is a k-ary realisable SRVR. Here, we will give a negative answer to

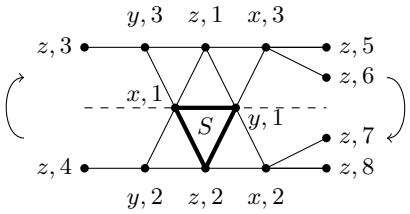


Fig. 1: A CSP with 3 variables where we have to use a ternary LLC in order to disallow the thickly edged solution S, see Example 1. The only symmetry that maps solutions to solutions is a reflection about the dashed line.

this question for the case $k = 2$. However, we also provide a range of special cases where we can answer the question affirmatively.

Reduction rules for LLCs have been studied in [4, 6, 10]. These reduction rules reduce the arity and number of LLCs. Grayland *et al.* [6] manage to show the minimality of certain sets of constraints with respect to the reduction rules under consideration. In a way, we study minimality of symmetry breaking constraints both independently of concrete reduction rules and independently of concrete constraints used for symmetry breaking. Solely based on the way orbits of solutions may interact, we show that LLCs exist whose arity cannot be reduced.

We consider the arity of symmetry breaking constraints on the original variables. For value symmetries, the introduction of auxiliary variables has been shown to be useful for finding sets of sometimes binary symmetry breaking constraints [9]. However, as we are interested in general results here, we focus on the original variables: any constraint can be turned into a binary constraint with the help of auxiliary variables.

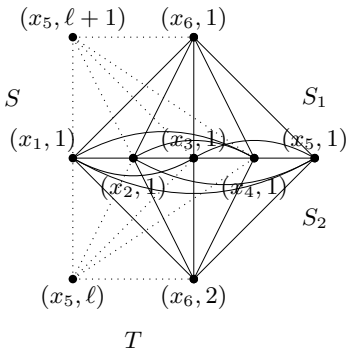
4 Upper Bounds on the Arities

In this section, we want to consider upper bounds on the arities of symmetry breaking constraints. More precisely, we prove that for the commonly used LLCs with a given variable order, no reduction rule may always reduce the arity. We show, however, that symmetry breaking constraints always exist whose arity does not exceed $\lfloor n/2 \rfloor + 1$.

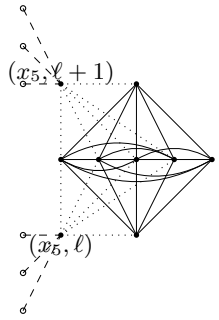
4.1 Fixed-order LLCs Are Intrinsically n-ary

In this section, we show that we cannot reduce the arity of LLCs in all CSPs. Before giving a general proof, let us consider the following example.

Example 1. Let $P = (\{x, y, z\}, D, \mathit{Cons})$. We have $D(x) = D(y) = \{1, 2, 3\}$, and $D(z) = \{1, 2, \ldots, 8\}$. The constraints are such that P has a microstructure as depicted in Figure 1. By inspection, it is obvious that the only symmetry that



(a) The initial solutions S_1 and S_2, with thin edges, contain all literals corresponding to assignments of value 1 to all variables in the set $\{x_1, \ldots, x_5\}$ and value 1, respectively 2 to variable x_6. Here, we also depict two solutions S and T with dotted edges. Solution S contains all literals of S_1 except for $(x_5, 1)$ and solution T is the symmetric counterpart of S under the reflection.



(b) To ensure that the solutions S and T are only symmetric to themselves, we add more literals to the literals (x_5, ℓ), and $(x_5, \ell+1)$, we depict the edges as dashed.

Fig. 2: Construction of the CSP as in the proof of Theorem 1 for the 6 variable case. The only symmetry that maps solutions to different solutions is a reflection about an axis going through $\{(x_1, 1), \ldots, (x_5, 1)\}$.

permutes a solution to another solution is a reflection about the dashed line. We choose $x \prec y \prec z$ and the natural order on the integers for lexicographic comparison. The solution $S = \{(x, 1), (y, 1), (z, 2)\}$ is not a lex-minimal solution in its orbit, but all pairs of literals in S are contained in lex-minimal solutions. In order to exclude any solution that is not lex-minimal, as LLCs would, we need a ternary constraint in order to disallow S.

We generalise the example in the following and show that for every $n > 1$, we can find a CSP with the same property that the CSP in Example 1 has: we need n-ary constraints to obtain a SRVR whose orbit representative solutions are the lex-minimal ones. We illustrate our proof with a binary CSP with 6 variables whose microstructure we partially depict in Figure 2. From the construction of the CSP in our proof, it will be clear that we cannot expect LLCs for a *binary* CSP with n variables to have arity less than n. This means that the various reduction rules that are known to reduce the complexity of the LLCs will not be able to reduce the arity of LLCs for all CSPs—in fact, we show that this is true for *any possible* reduction rule.

Theorem 1. *For any $n > 1$, a binary CSP with n variables and an order on its variables exist such that any SRVR that has lex-minimal (wrt to the variable order) solutions as orbit representative solutions is n-ary realisable.*

Proof. We construct a CSP with n variables ordered as $x_1 \prec x_2 \prec \cdots \prec x_n$. We construct the domains of the variables during the proof and we use the natural order on the integers for lexicographic comparison. Whenever we say that we construct a solution of the CSP, we really mean that we add or modify constraints to the CSP such that they allow these solutions. We construct the solutions of the CSP in such a way that the symmetries of the CSP will be immediate. We start with two solutions $S_i = \{(x_1, 1), \ldots, (x_{n-1}, 1), (x_n, i)\}$ where $i = 1, 2$. We have depicted the solutions S_1 and S_2 in Figure 2a for the case where $n = 6$.

Our aim is to add solutions to the CSP such that every subset of size up to $n-1$ in S_2 is contained in a lex-minimal solution. When we add the solutions, care has to be taken in two aspects. First, we need to ensure that S_1 and S_2 remain symmetric under the symmetry of the CSP that swaps $(x_n, 1)$ with $(x_n, 2)$ while leaving all other literals unchanged. We call this symmetry a *reflection* due to the geometric intuition, see Figure 2. Every time we add literals to the CSP, we have to ensure that we can extend the reflection to the new literals. Second, we need to ensure that S_1 and S_2 are symmetric only to each other. Solution S_1 is, with respect to the chosen orders, lex-less than S_2. Hence, we need to exclude S_2. Having constructed a lex-minimal solution for every subset of size up to $n-1$ in S_2, it follows that if we want to find a SRVR with the lex-minimal orbit representatives, we have to use an n-ary constraint to remove S_2.

We construct symmetric solutions for symmetric subsets of solutions S_1 and S_2 with size up to $n-1$. We first note that any subset of S_2 not involving literal $(x_n, 2)$ is contained in S_1 by construction and hence, we cannot add a constraint that forbids this subset. We note next that it suffices to consider only subsets of size exactly $n-1$. For any subset of size $n-1$ that includes literal $(x_n, 2)$, we construct a solution that uses this subset. For the subset of size $n-1$ to be a solution, we introduce a literal (x, ℓ), such that ℓ does not yet appear in the CSP and ensure that the subset with the new literal is a solution T. We construct another solution S containing all literals corresponding to value 1 in T, it contains $(x_n, 1)$ as well as a new literal $(x, \ell+1)$. We ensure that S is symmetric to T with respect to the reflection by extending the reflection to swap (x, ℓ) and $(x, \ell+1)$. In Figure 2a, we have depicted T for the subset of 5 literals of solution S_2 consisting of the set of literals $\{(x_1, 1), (x_2, 1), (x_3, 1), (x_4, 1), (x_6, 2)\}$. Solution T contains the new literal (x_5, ℓ), where ℓ is chosen appropriately.

With our construction so far, we have ensured that T and S are symmetric and that S_1 and S_2 remain symmetric. To ensure that T and S are not symmetric to other solutions, we add a set of literals L. In the microstructure, we make half of the literals in L adjacent to (x, ℓ) and the other half adjacent to $(x, \ell+1)$. We choose the cardinality of L in such a way that (x, ℓ) and $(x, \ell+1)$ have a (node) degree in the microstructure that only these two literals have, see Figure 2b. By addition of the literals, the CSP gains more non-trivial symmetries that permute the recently added literals while leaving all other literals unchanged. However, we do not have to consider these new symmetries, as they are the identity when restricted to solutions.

With this construction, solutions S_1 and S_2 remain symmetric. Furthermore, solutions S and T are the only members of their orbit, due to the degrees of the literals (x, ℓ), and $(x, \ell + 1)$. The lex-minimal solution in this orbit is T. Hence, we cannot forbid any set of literals of size up to $n-1$ in S_2. This shows that the constraint that forbids solution S_2 must be n-ary. □

In other words, Theorem 1 shows that no reduction rule will be able to reduce the arity of LLCs below the original arity for all CSPs. The step of the proof depicted in Figure 2b is not necessary, if we want to prove the theorem only for a subgroup of the symmetry group.

4.2 A $\lfloor n/2 \rfloor + 1$-ary Realisable SRVR Always Exist

In this section, we prove the existence of symmetry breaking constraints with arity $\lfloor n/2 \rfloor + 1$. In order to show this, we need more notation.

In the following we introduce distinguished sets of orbit representative solutions for a CSP. We refer to the set of literals of a CSP by *lits*. For a complete set of orbit representative solutions R, we define a function $c(\cdot, R)$. For a literal $v \in lit$, we have $c(v, R) = 0$ if v is contained in at most one solution in R. If v is contained in k orbit representative solutions in R, where $k \geq 2$, then $c(v, R) = k-1$. We define $c(R) = \sum_{v \in lits} c(v, R)$. For the set $\mathcal{S}$ of complete sets of orbit representative solutions, we call a set $R \in \mathcal{S}$ such that $R = \arg\max_{Q \in \mathcal{S}} c(Q)$ a *max-common set*. The advantage of a max-common set is that it helps to break down the globality of the symmetries into some desirable "local" properties which we will then explore. We need the following observation to prove such a property.

Proposition 1. *Let S and T be two solutions and let ϕ be any symmetry of the CSP. Then $|S \cap T| = |\phi(S) \cap \phi(T)|$.*

This is enough to prove following property of any max-common set.

Proposition 2. *For a max-common set of orbit representative solutions $R = \{S_1, \ldots, S_t\}$ and a non-representative solution $T \in \text{orbit}(S_j)$, any S_i with $|S_i \cap T| = k$ has $|S_i \cap S_j| \geq k$.*

Proof. By contradiction. Let S_i be a solution in a max-common set such that $|S_i \cap T| = k$. We assume that $|S_i \cap S_j| < k$. Let ϕ be a symmetry such that $\phi(S_j) = T$. By Proposition 1, we know that $|\phi(S_i) \cap \phi(S_j)| = k$. Hence, for the set $\phi(R) := \{\phi(S_1), \ldots, \phi(S_t)\}$, we have $c(\phi(R)) > c(R)$. This is a contradiction to the fact that $S_1, \ldots, S_t$ is a max-common set of orbit representative solutions.

We can now prove the main result of this section.

Theorem 2. *A $\lfloor n/2 \rfloor + 1$-ary realisable SRVR exists for any CSP with n variables.*

Proof. Consider a CSP with n variables and a max-common set of orbit representative solutions R. Let T be any solution with $T \notin R$. Let $S \in R$ be such that $T \in \text{orbit}(S)$..

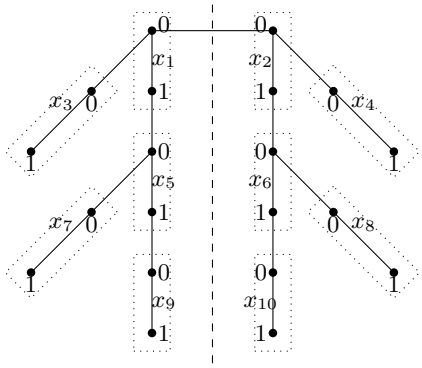


Fig. 3: The MSC of a binary CSP for which no binary realisable SRVR exists. The only symmetry of the CSP is a reflection about the dashed line. Nodes in the dotted boxes are literals corresponding to the same variable.

Let n be even. We assume first that $|S \cap T| < n/2$. We show that a subset of $n/2+1$ literals in $T \setminus S$ exists that is not contained in any orbit representative in R. For the sake of contradiction, we assume that no such subset exists, so we assume all $n/2+1$ literals are contained in a member of R. By Proposition 2, any $S_1 \in R$ that contains $n/2+1$ literals in $T \setminus S$ contains at least $n/2+1$ literals in $S \setminus T$. Hence, S_1 contains more than n literals, which is a contradiction. This means that a set of literals in T exists that is not in any orbit representative solution and we can forbid this set safely.

Next, we assume that $|S \cap T| \geq n/2$. We show that all of the, say k, literals in $T \setminus S$ and $n/2-k$ literals in $T \cap S$ cannot be contained in any $S_1 \in R$. If the $n/2$ literals were contained in S_1, then S_1 would also contain at least k literals in $S \setminus T$. However, the literals in $S \setminus T$ have the same variables as the k literals in $T \setminus S$. Therefore, S_1 is not a solution, which is a contradiction and we can forbid a set of $n/2+1$ literals in T safely.

For an odd number of variables the argumentation is similar and we omit this case due to space restrictions. □

5 A Lower Bound on the Arity

In this section, we show that for binary CSPs we sometimes have to rely on non-binary constraints to break all symmetries. This provides a lower bound for the afore-mentioned reduction rules: If a reduction rule for binary CSPs can reduce the arity of symmetry breaking constraints for any CSP to the lower bound we provide here, then the reduction rule is optimal.

Theorem 3. *A binary CSP exists that does not have a binary realisable SRVR.*

Proof. We consider a 10 variable CSP P_{tree} where every variable has domains $\{0,1\}$. The MSC of P_{tree} is a tree. We depict it in Figure 3. By inspection of the MSC, it is obvious that there is only one non-trivial symmetry of the MSC. It swaps literals (x_i, j) with (x_{i+1}, j) for odd i and $j \in \{0,1\}$. The symmetry is a reflection about the dotted line in Figure 3. We define a set $L := \cup_{i\in\{1,2,5,6,9,10\}}\{(x_i,1)\}$. The solutions of P_{tree} we consider in the following, have L in common. We consider the following four self-symmetric solutions of P_{tree}:

$$L\cup\{(x_3,1),(x_4,1),(x_7,0),(x_8,0)\}, \qquad L\cup\{(x_3,1),(x_4,1),(x_7,1),(x_8,1)\},$$
$$L\cup\{(x_3,0),(x_4,0),(x_7,1),(x_8,1)\}, \text{ and } \quad L\cup\{(x_3,0),(x_4,0),(x_7,0),(x_8,0)\}.$$

Orbits of solutions that consist of more than one solution contain exactly two solutions because the CSP only has one non-trivial symmetry. From the orbits of solutions with two members, we consider the following eight solutions:

$$\begin{aligned}
A_1 &= L \cup \{(x_3,1),(x_4,0),(x_7,1),(x_8,1)\},\\
A_2 &= L \cup \{(x_3,0),(x_4,1),(x_7,1),(x_8,1)\},\\
B_1 &= L \cup \{(x_3,1),(x_4,1),(x_7,0),(x_8,1)\},\\
B_2 &= L \cup \{(x_3,1),(x_4,1),(x_7,1),(x_8,0)\},\\
C_1 &= L \cup \{(x_3,1),(x_4,0),(x_7,0),(x_8,1)\},\\
C_2 &= L \cup \{(x_3,0),(x_4,1),(x_7,1),(x_8,0)\},\\
D_1 &= L \cup \{(x_3,1),(x_4,0),(x_7,1),(x_8,0)\},\\
D_2 &= L \cup \{(x_3,0),(x_4,1),(x_7,0),(x_8,1)\}.
\end{aligned}$$

Members of the same orbit have the same capital letter. In order to find a SRVR we need to choose an orbit representative among solutions A_1 and A_2. The only pair of literals in A_1 that is not contained in any of the four self-symmetric solutions is pair $p := \{(x_3,1),(x_4,0)\}$. Disallowing p, makes A_2 the orbit representative of $\{A_1, A_2\}$ and removes C_1 and D_1 and hence, C_2 and D_2 must be the orbit representatives of the respective orbits. Consider $\{B_1, B_2\}$ next. The only pair of literals in B_1 that we can forbid is the pair of literals $q := \{(x_7,0),(x_8,1)\}$. Disallowing q however would disallow solution D_2, an orbit representative. The same is true for B_2. The only pair of literals we could disallow is $\{(x_7,1),(x_8,0)\}$ which would lead to orbit representative C_2 being forbidden. Hence, solution A_2 cannot be an orbit representative.

A symmetric argumentation holds if we choose solution A_2 as an orbit representative, which shows that P_{tree} does not have a binary realisable SRVR. □

The CSP P_{tree} in the proof of Theorem 3 has rather many solutions and orbits. Unfortunately, our attempts to construct a less complex example have failed. In the next section, we shall see reasons for the complexity of the example: for various special cases of CSPs, binary realisable SRVRs always exist.

6 Special Cases

In this section, we present special cases in which binary realisable SRVRs exist.

Theorem 4 (Sufficient Conditions). *In all of the following cases, a (not necessarily binary) CSP P has a binary realisable SRVR:*

- *for any pair of solutions S, T of P from different orbits we have $S \cap T = \emptyset$,*
- *P has only one orbit of solutions with cardinality greater or equal to 1, and*
- *P has exactly 2 orbits of solutions.*

Proof. If solutions in different orbits do not share literals, we can consider each orbit on its own. In each orbit, we choose an orbit representative S. Any other solution T in orbit(S) differs from S in at least one literal ℓ. Since solutions from different orbits are disjoint from T, literal ℓ is not contained in any orbit representative. Hence, we can forbid this literal and even unary constraints suffice.

The second case is a corollary of Theorem 15 in [7].

For the case of exactly two orbits of solutions, we choose a solution S_1 from the first orbit and a solution S_2 from the second orbit. Any other solution S_3 with $S_3 \neq S_1$ and $S_3 \neq S_2$ has a literal $v \in S_3 \setminus S_1$ and a literal $w \in S_3 \setminus S_1$. If $v = w$ we can add a constraint to forbid w and thus forbid S_3 without affecting neither S_2 nor S_1, since w is contained exclusively in solution S_2. If $v \neq w$, we can add a constraint forbidding $\{v, w\}$ without affecting neither S_1 nor S_2, since the pair of literals $\{v, w\} \not\subseteq S_1$ and $\{v, w\} \not\subseteq S_2$. So, a binary realisable SRVR exists that has S_1 and S_2 as orbit representative solutions. □

The first two cases of Theorem 4 subsume the case where there is only one orbit of solutions, like a CSP consisting of an alldifferent constraint. For this case, Puget [12] showed that LLCs reduced to binary constraints suffice. Other CSPs that only have one orbit of solutions include n-queens for $n = 4, 5, 6$ and the graceful graph problem for $K_5 \times P_2$ [1]. Another example that can easily be generalised to a class of problems is the CSP with 4 variables x_1, x_2, x_3, x_4, the domain $\{1, \ldots, 4\}$ and constraints $x_1 \neq x_2$, $x_3 \neq x_4$.

In the reminder of this section, we focus on binary CSPs. First we consider CSPs with a restriction on the number of orbits and then, we consider CSPs with a restriction on the MSC.

6.1 Binary CSPs with Three Orbits of Solutions

In this section, we prove that for binary CSPs with three orbits of solutions, there always is a binary realisable SRVR. We need some more notation. In the microstructure of a binary CSP with a complete set of orbit representative solutions, we call an edge *erasable* if it does not belong to any orbit representative solution in the complete set. Similar to Section 4.2, we set apart certain complete sets of orbit representative solutions. We call a complete set of orbit representative solutions a *max-erasable set*, if the number of erasable edges is maximal. A max-erasable set is a rather special complete set of orbit representative solution for which we show that a binary realisable SRVR exists. We first prove two properties of max-erasable sets.

Proposition 3. *For a CSP with three orbits of solutions, let $R = \{S_1, S_2, S_3\}$ be a max-erasable set and let $T \notin R$ be a solution with the following property. Every pair of literals in T is contained in at least one member of R. Then, T and the orbit representative solution $S_1 \in R$ of* orbit(T) *have at least one pair of literals in common.*

Proof. All pairs of literals of T are contained in some orbit representative solution in R by assumption in the proposition. We prove that a pair of literals in T exists, that is contained in S_1.

Solution S_2 may contain all literals in T apart from, say, literal v. Solution S_3 may contain all of the literals in T apart from, say, literal w. If $v = w$, then any edge $\{\cdot, w\} \subset S_1$, since we have $\{\cdot, w\} \not\subset S_2$ and $\{\cdot, w\} \not\subset S_3$. If $v \neq w$, then the pair of literals $\{v, w\}$ must be contained in either S_1, S_2 or S_3, because of the assumption that all pairs in T are contained in an orbit representative solution. Since $v \notin S_2$ and $w \notin S_3$, we have $\{v, w\} \subseteq S_1$. □

Proposition 4. *With the prerequisites of Proposition 3 the following holds. A literal $x \in S_1 \setminus T$ exists that is not contained in $S_2 \cup S_3$.*

Proof. For the sake of contradiction we assume that all literals in $S_1 \setminus T$ are contained in either S_2 or S_3. With this assumption, we show the existence of a clique of size $n+1$ in the microstructure. As we noted in Proposition 3, solution S_1 contains at least a pair of literals of T. Assume that $|S_1 \cap T| = 2$, the case $|S_1 \cap T| > 2$ is similar.

We first show, that a literal t in $T \setminus S_1$ exists with $t \in S_2$ and $t \in S_3$. Assume that this is not the case, say, literal $t \in S_2$ and $t \notin S_3$. A literal $s \in T$ exists with $s \notin S_2$ because $T \neq S_2$. The pair of literals $\{s, t\}$, we have $\{s, t\} \subseteq S_2$ because $s \notin S_2$. Furthermore, $\{s, t\} \not\subseteq S_3$ by the assumption $t \in S_2$, $t \notin S_3$ and literal t is chosen in such that $t \notin S_1$. Hence, the pair $\{s, t\}$ is not contained in any orbit representative solution, which is a a contradiction to the assumption that all pair of literals in T are contained in an orbit representative solution.

So, a literal $t \in T \setminus S_1$ exists with $t \in S_2$ and $t \in S_3$. By assumption, all literals in $S_1 \setminus T$ are either contained in S_2 or S_3. Hence, any literal in $S_1 \setminus T$ is adjacent in the microstructure to literal t with $t \in S_2$ and $t \in S_3$. All literals in T, in particular the literals in $T \cap S_1$ are adjacent to literal t because $t \in T$. Hence, $S_1 \cup \{t\}$ is a $n+1$-clique which can never occur in a microstructure of a CSP with n variables. □

We can now prove the main result of this section.

Theorem 5. *A binary realisable SRVR exists for any binary n-variable CSP with three orbits of solutions.*

Proof. By contradiction. Consider a max-erasable set of orbit representative solutions consisting of solutions S_1, S_2 and S_3. We assume that in some orbit of solutions orbit(S_1), a solution $T \neq S_1$ exists such that all pairs of literals in T are contained in solutions in the max-erasable set, otherwise there is nothing to prove. By Proposition 4, one of the literals, say, $x \in S_1 \setminus T$ is not contained

$S_2 \cup S_3$. Hence, making T the orbit representative solution of orbit(T), we augment the number of erasable pairs of literals which is a contradiction to the assumption that $\{S_1, S_2, S_3\}$ is a max-erasable set. □

The above theorem could probably be generalised to more orbits of solutions, as a more direct use of the symmetries should provide extra leverage for proofs. We leave such generalisations for future work.

6.2 Binary CSPs with Restricted MSCs

So far, we have proved that for CSPs with a restricted number of orbits, there always is a binary realisable SRVR. In general, however, we assume solutions to be unknown to the constraint programmer and hence, our existence results are of limited practical importance. Next, we consider CSPs with a restriction on the MSC without restricting the solutions of the CSP directly.

We define PATH as the class of CSPs whose MSC is a path. We note that any CSP in PATH has domains of size at most 2. The constraints of the CSPs in PATH can be thought of as generalised implications.

Proposition 5. *A binary realisable SRVR exists for any CSP in PATH.*

Proof. We note first that any CSP in PATH that has a variable whose domain is of size 1, admits at most one solution, hence we consider the case where all domains have size 2. We also assume that the CSP has more than 2 literals, otherwise we simply forbid one of the two literals. Let the CSP have variables $x_1, \ldots, x_n$. Without loss of generality, we may assume that each variables has domain $\{1, 2\}$ and that $\{(x_i, 2), (x_{i+1}, 1)\}$ is an edge in the MSC for every $i \in \{1, \ldots, n-1\}$. We note that a path has one non-trivial symmetry. For all $i < \lceil n/2 \rceil$ and $j \in \{1, 2\}$, the symmetry swaps (x_i, j) with $(x_{n-i+1}, j + 1 \bmod 3)$. This symmetry can be thought of as a reflection about the centre of the path. In a CSP in PATH we have the following solutions. The solutions where all variables have value 1 is symmetric to the solution where all variables have value 2. Next, we have a solution, where the first i variables have value 1 and $n - i$ variables have value 2. This solution is symmetric to a solution where the first $n - i$ variables have value 1 and the last i variables have value 2. The CSP has no other solutions.

We forbid the pair of literals $\{(x_1, 2), (x_2, 2)\}$. This removes the solution where all variables have value 2, but leaves the other orbit member. Next, we consider the other solutions where the first i variables have value 1 and the next $n - i$ variables have value 2. If we have $n - i = i$, then we do not add a binary constraint, because in this case the orbit only has one member. If we have $n - i \neq i$, then the orbit has two solutions. We forbid $\{(x_i, 1), (x_{i+1}, 2)\}$. Certainly, this leaves the solution where all variables have value 1. It also does not disallow any other solution from other orbits, but it forbids one member of its own orbit. Hence, a binary realisable SRVR exists for CSPs in PATH. □

Next, we consider CSPs with MSC whose connected components are paths. We denote this class by PATH*.

(a) A connected component corresponding to one variable.

(b) The c-solution is self-symmetric with respect to the p-symmetries.

(c) The c-solution is c-symmetric to the partial solution in Figure 4c.

Fig. 4: Connected components of a MSC that are allowed by the constraints of Proposition 5 as in the proof of Theorem 6. The literals in the c-solutions are depicted as black.

Theorem 6. *A binary realisable SRVR exists for any CSP in PATH*.*

Applying Proposition 5 to every connected component is not enough to prove the above theorem. If the MSC of a CSP consists of connected components as depicted in Figures 4b and 4c, the symmetry that swaps the two connected components is not broken by the constraints presented in the proof of Proposition 5.

Proof. Consider a CSP P whose MSC is a disjoint union of paths. We call a partial solution that is a solution in the connected component a *c-solution*. We distinguish between two types of symmetries. We call symmetries that permute literals within a connected component *p-symmetries*. We call symmetries that permute entire connected components *c-symmetries*. Any symmetry of P is a composition of p- and c-symmetries. As in the proof of Proposition 5, we assume without loss of generality that the domains of the variables are subsets of $\{1, 2\}$ and that the only edges in the MSC are $\{(x_i, 2), (x_{i+1}, 1)\}$. Unlike Proposition 5, we need to consider variables that have a single value in their domains.

We first consider the case of a CSP whose MSC has symmetrically distinct connected component, i.e., P has no c-symmetries. We can rely on the arguments in the proof of Proposition 5. We consider every connected component on its own. For a connected component with an even number of literals greater than 3, we add a constraint as in the proof of Proposition 5, i.e., forbidding $\{(x_i, 2), (x_{i+1}, 2)\}$. The set of constraints we have introduced so far, forbid all solutions that contain no c-solution that is a) self symmetric under the p-symmetries and b) that consists of a variable and its domain (see Figure 4 for examples). By the assumption of connected components not being symmetric, a c-solution that is p-self-symmetric can only be in a solution that has an orbit of size greater than 1, if another c-solution is not p-self-symmetric. Hence, we only need to consider solutions that contain a c-solution on a connected component similar to Figure 4a and consists of, say, literals $(x_i, 1)$ and $(x_i, 2)$. Here, it is easy to show that binary realisable SRVR exists. Any solution that contains $(x_i, 2)$ can be disallowed, since a symmetric equivalent exists that contains $(x_i, 1)$. So, unary constraints suffice.

Next, we consider the case where P has symmetric connected components. In this case, a p-self-symmetric c-solution can be symmetric to a c-solution that a p-symmetry maps to a different c-solution, see Figures 4b and 4c. For this case, we need arguments that are beyond the proof of Proposition 5. The interesting case of p-self-symmetric c-solutions are solutions with an even number of variables

and every variable has domain of size 2 as we argue in the following. As soon as a variable has a domain of size 1 in the connected component there can only be one c-solution in the connected component. This case is uninteresting because a p-self-symmetric c-solution is c-symmetric to a p-self-symmetric c-solution. So, we assume to have k symmetric connected components each of which has ℓ variables. Each variable has domain of size 2. We number the connected components and denote variables with two indices, the first for the connected component and the second for the number in which the variable appears in a straight-forward order of the literals in the path. We add the following constraints. Any solution that contains $(x_{i,1+\ell/2}, 2)$ we forbid by simply adding a constraint that does not allow the combination of this literal with $(x_{i-1,1+\ell/2}, 1)$. Let us prove that this constraint does not remove an entire orbit. The literal $(x_{i-1,1+\ell/2}, 1)$ is only contained in the p-self-symmetric c-solution. No other c-solution contains this literal. We note however that we cannot simply remove the literal $(x_{2,1+\ell/2}, 2)$. This would remove an orbit of solutions, namely the orbits where all c-solutions in the symmetric connected components are the same and self-symmetric. Hence, we have shown that a binary realisable SRVR exists. □

For CSPs with a MSC whose connected components are cycles, denoted by CYCLE*, we can prove an analogous result to Theorem 6.

Proposition 6. *A binary realisable SRVR exists for any CSP in CYCLE*.*

Proof. We consider the case of a single connected component first. For an odd number of literals in the MSC, the only case where a solution exists, is the case of a CSP with 1 variable and domain size 3. In this case, we can simply add constraints that forbid 2 literals with unary constraints. A MSC that is an odd cycle of length greater than 3 does not have a solution. For a satisfiable CSP with a MSC that is an even cycle of size $2n$, the CSP has n variables each with domain size 2 and we have one orbit of solutions, so we can rely on Theorem 4.

For CSPs with more than one connected component, we note that we do not have the case of self-symmetric solutions in the connected components as in the proof of Theorem 6. Hence, the arguments for the case of a single connected component suffice to terminate this proof. □

Theorem 6 and Proposition 6 extend naturally to any CSP with a MSC that consists of connected components that are either cycles or paths.

7 Conclusion and Further Work

We considered complete symmetry breaking for the full group of constraint symmetries in this paper. We obtained the following bounds on the arities of symmetry breaking constraints. We showed that no reduction rule for LLCs will succeed in reducing the arity of the LLCs of all CSPs, therefore obtaining an *upper bound* on the arities of LLCs. However, we also showed that constraints of arity $\lfloor n/2 \rfloor + 1$ always exist that provide complete symmetry breaking. The latter is an existence

result and does not come with an algorithm. Further work could try to find variable ordering strategies, such that reduction rules will always reduce the arity of LLCs. We found a *lower bound* on the arities of symmetry breaking constraints: we proved that binary CSPs exist where complete symmetry breaking constraints must be at least ternary. However, we also presented various special cases of CSPs where binary constraints break all symmetries.

Acknowledgements

The author thanks Barbara Smith for discussions and Marc van Dongen for the same as well as insightful comments during the preparation of this paper.

References

1. D. Cohen, P. Jeavons, C. Jefferson, K. Petrie, and B. Smith. Symmetry definitions for constraint satisfaction problems. *Constraints 11*, 2006.
2. J. Crawford, M. Ginsberg, E. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In *Principles of Knowledge Representation and Reasoning (KR '96)*. Morgan Kaufmann, 1996.
3. A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, and T. Walsh. Global constraints for lexicographic orderings. In *Principles and Practice of Constraint Programming (CP 02)*, 2002.
4. A. M. Frisch and W. Harvey. Constraints for breaking all row and column symmetries in a three-by-two matrix. In *In Proceedings of SymCon'03*, 2003.
5. I. P. Gent, K. E. Petrie, and J.-F. Puget. Symmetry in constraint programming. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, pages 329–376. Elsevier, 2006.
6. A. Grayland, C. Jefferson, I. Miguel, and C. Roney-Dougal. Minimal ordering constraints for some families of variable symmetries. *Annals of Mathematics and AI*, 2009.
7. T. Januschowski, B. M. Smith, and M. R. C. van Dongen. Foundations of symmetry breaking revisited. In *Proceedings of SymCon'09*, 2009.
8. V. Kaibel and M. E. Pfetsch. Packing and partitioning orbitopes. *Mathematical Programming*, 114, Number 1 / July, 2008:1–36, 2008.
9. Y. C. Law and J. H. Lee. Symmetry breaking constraints for value symmetries in constraint satisfaction. *Constraints*, 11(2-3):221–267, 2006.
10. H. Öhrmann. Breaking symmetries in matrix models. Master's thesis, Dept. Information Technology, Uppsala University, 2005.
11. J.-F. Puget. On the satisfiability of symmetrical constrained satisfaction problems. In *Methodologies for Intelligent Systems*, pages 350–361, London, UK, 1993. Springer-Verlag.
12. J.-F. Puget. Breaking symmetries in all-different problems. In *International Joint Conferences on Artificial Intelligence*, pages 272–277, 2005.
13. T. Walsh. General symmetry breaking constraints. In *Principles and Practice of Constraint Programming – CP 2006*, volume 4204/2006 of *Lecture Notes in Computer Science 4204, 2006*, pages 650–664, 2006.