Constraints (2009) 14:506–538
DOI 10.1007/s10601-008-9059-7

Dynamic structural symmetry breaking for constraint satisfaction problems

Pierre Flener · Justin Pearson · Meinolf Sellmann · Pascal Van Hentenryck · Magnus Ågren

Published online: 30 September 2008

Abstract In recent years, symmetry breaking for constraint satisfaction problems (CSPs) has attracted considerable attention. Various general schemes have been proposed to eliminate symmetries. In general, these schemes may take exponential space or time to eliminate all the symmetries. We identify several classes of CSPs that encompass many practical problems and for which symmetry breaking for various forms of value or variable interchangeability is tractable using dedicated search procedures. We also show the limits of efficient symmetry breaking for such dominance-detection schemes by proving intractability results for some classes of CSPs.

Keywords Symmetry breaking · Dominance detection · Tractability · CSP

The authors' names are ordered according to the Swedish alphabet.

P. Flener (✉) · J. Pearson
Department of Information Technology, Uppsala University,
Box 337, 751 05 Uppsala, Sweden
e-mail: pierref@it.uu.se

J. Pearson
e-mail: justin@it.uu.se

M. Sellmann · P. Van Hentenryck
Department of Computer Science, Brown University,
Box 1910, Providence, RI 02912, USA

M. Sellmann
e-mail: sello@cs.brown.edu

P. Van Hentenryck
e-mail: pvh@cs.brown.edu

M. Ågren
SICS, Box 1263, 164 29 Kista, Sweden
e-mail: agren@sics.se

1 Introduction

Many constraint satisfaction problems (CSPs) naturally exhibit symmetries. Symmetry breaking may drastically improve performance [3, 22, 24, 31]. An important contribution in this area has been the development of various general schemes for symmetry breaking *during* search in CSPs (e.g., symmetry breaking during search (SBDS) [2, 17] and symmetry breaking by dominance detection (SBDD) [9, 15, 24], the latter being described briefly in Section 3). Unfortunately, in general, these dynamic symmetry-breaking schemes may require exponential resources to break all the symmetries. Indeed, some schemes may require exponential space to store all the nogoods generated through symmetries, while others may take exponential time to discover whether a partial assignment is symmetric to one of the existing nogoods. As a consequence, practical applications often place limits on how many nogoods can be stored and/or which symmetries to break. Other than eliminating symmetries by re-modelling the problem (see, e.g., [30]), another important approach is to break symmetries statically by adding constraints *before* search starts (e.g., [7, 23]). Unfortunately, in general, a super-exponential number of constraints may be needed to break all the symmetries. For instance, the lex-leader scheme of [7] adds one constraint per symmetry, but the number of symmetries is often super-exponential (an $m \times n$ matrix with fully interchangeable rows and columns has $m! \cdot n!$ symmetries). As a consequence, practical applications often add only some of these symmetry-breaking constraints (see, e.g., [11, 29]).

We approach symmetry breaking from a different, orthogonal standpoint. Our goal is to identify classes of CSPs that are practically relevant and for which symmetry breaking is tractable, i.e., polynomial in time and space, using dedicated search procedures exploiting the problem structure. We identify several such classes whose CSPs feature various forms of value or variable interchangeability and encompass many practical problems. For some of them, such *dynamic structural symmetry breaking* can even be performed with a *constant* overhead with respect to both time and space at every node explored. We also introduce the new notion of abstract nogood, which is used to derive the results for some of the CSP classes. We believe that this notion is helpful to derive many other classes of tractable symmetries. As such, this paper should be viewed only as a first step in this fascinating area. Finally, we also show the limits of efficient dynamic structural symmetry breaking for dominance-detection schemes like SBDD by proving intractability results for certain classes of CSPs.

The main objective of this theoretical paper is to establish the bounds of tractability of dynamic symmetry breaking for a landscape of CSP classes (see Table 1 in the conclusion). A runtime comparison with static symmetry breaking, even if structural [12, 13], is beyond the scope of this paper.

It is useful to contrast our approach with the research avenue pioneered by Freuder [16] on value interchangeability. He also introduced various forms of value interchangeability. However, his goal was to *discover* symmetries inside CSPs and to remove them through a preprocessing reformulation. Unfortunately, discovering symmetries in CSPs is not tractable for many interesting classes of CSPs. This paper, in contrast, assumes that the symmetries in a CSP are known. It focuses on how to exploit this knowledge *during search* to break symmetries efficiently. In [34], we address the companion issue of how to automatically detect symmetries in CSP models.

Example 1 Consider the scene allocation problem featured in [32]. It aims at producing a movie (or a series) at minimal cost by deciding when to shoot which scenes. Each scene involves a number of actors and at most five scenes a day can be shot. All the actors of a scene must be present on the day the scene is shot. The actors have fees representing the amount to be paid per day they spend in the studio. An optimal solution can be modelled as an assignment of scenes to days that minimises the production costs. The exact days assigned to the scenes have no importance and are fully interchangeable. What is important is how the scenes are clustered. In fact, the original problem formulation only has a *number*, say n, of days. It is the often necessary *naming* of these days while modelling the problem, say as $1 \dots n$, that induces these symmetries. Our dynamic structural symmetry breaking approach does not aim at discovering this fact; it rather focuses on how to exploit it to break the symmetries it induces.

This theoretical paper, which unites and extends[1] our work published in [27, 33], is structured as follows. First, in Section 2, we define CSPs and assignments in a non-standard way that gives rise to elegant formulations and proofs of our results. Then, in Sections 3 to 7, we formally establish those results, for various forms of value and/or variable interchangeability. Finally, Section 8 summarises the results and concludes this paper.

2 Preliminaries

Our definition of CSPs, although it captures their informal meaning, is non-standard but simplifies the proofs and other definitions considerably. The basic idea is that the set of constraints of a CSP is abstracted by a Boolean function that returns **true** if all these constraints are satisfied. We are not interested in the constraint structure. Solutions are then also represented as functions, namely from the variables to the possible values.

Definition 1 (CSP, Assignment, Solution) A *CSP* is a triplet $\langle V, D, C \rangle$, where V denotes the set of variables, D denotes the set of possible values for these variables and is called their *domain*, and $C : (V \to D) \to \mathit{Bool}$ is a constraint that specifies which assignments of values to the variables are solutions. An *assignment* for a CSP $\mathcal{P} = \langle V, D, C \rangle$ is a function $\alpha : V \to D$. If the domain D is the power-set of some other set, called the *universe*, we say that the CSP is a *set-CSP* and has *set variables*, while we call α a *set assignment*; otherwise, we say that the CSP has *scalar variables*. A *solution* to a CSP $\mathcal{P} = \langle V, D, C \rangle$ is an assignment σ for $\mathcal{P}$ such that $C(\sigma) = \mathbf{true}$. The set of all the solutions to a CSP $\mathcal{P}$ is denoted by $\mathit{Sol}(\mathcal{P})$.

[1] Sections 5.3, 7.2, 7.3, and the epilogue to Corollary 2 are new, while Section 5.2 was generalised. The originally omitted proofs of Proposition 1, Proposition 2, and Theorem 3 are now provided, while the proofs of Theorem 1 and Corollary 1 were expanded into greater detail. The full version of this paper, including other originally omitted proofs, but excluding Section 7.3, is in [14].

Algorithms to solve CSPs manipulate partial assignments. It is often important to reason about which variables are already assigned (the *scope* of the partial assignment) and the set of values they are assigned to (the *image* of the partial assignment).

Definition 2 (Partial Assignment, Scope, Image) A *partial assignment* for a CSP $\mathcal{P} = \langle V, D, C\rangle$ is a function $\alpha : W \to D$, where $W \subseteq V$. The *scope* of α, denoted by $scope(\alpha)$, is W. The *image* of α, denoted by $image(\alpha)$, is the set $\{\alpha(v) \mid v \in scope(\alpha)\}$. For each value $d \in image(\alpha)$, we use $\alpha^{-1}(d)$ to denote the set $\{v \mid v \in scope(\alpha)\ \&\ \alpha(v) = d\}$. We denote the empty partial assignment by ϵ.

Note that every assignment and solution to a CSP $\mathcal{P} = \langle V, D, C\rangle$ is a partial assignment for $\mathcal{P}$, with scope V. We often denote a partial assignment α by a conjunction of equations, and then see it as a constraint:

$$v_{i_1} = \alpha(v_{i_1})\ \ \&\ \ \cdots\ \ \&\ \ v_{i_k} = \alpha(v_{i_k})$$

where $scope(\alpha) = \{v_{i_1}, \ldots, v_{i_k}\}$.

Example 2 The partial assignment $v_1 = 1\ \ \&\ \ v_2 = 2\ \ \&\ \ v_3 = 3$ represents the function whose scope is $\{v_1, v_2, v_3\}$ and that assigns the value i to v_i.

Definition 3 (Extension of a Partial Assignment) A partial assignment θ for a CSP $\mathcal{P}$ is an *extension* of a partial assignment α for $\mathcal{P}$ if $scope(\alpha) \subseteq scope(\theta)$ and $\forall v \in scope(\alpha) : \theta(v) = \alpha(v)$.

Definition 4 (Completion of a Partial Assignment) A *completion* of a partial assignment α for a CSP $\mathcal{P} = \langle V, D, C\rangle$ is an extension θ of α with $scope(\theta) = V$. The set of all the completions of α for $\mathcal{P}$ is denoted by $Comp(\alpha, \mathcal{P})$.

Note that the set of all the completions of a solution σ is the singleton $\{\sigma\}$.

Definition 5 (Nogood) A *nogood* for a CSP $\mathcal{P}$ is a partial assignment α for $\mathcal{P}$ that cannot be extended into a solution, that is $Comp(\alpha, \mathcal{P}) \cap Sol(\mathcal{P}) = \emptyset$.

The idea behind the noun 'nogood' is that no partial assignment should ever extend any previously identified nogood.

Definition 6 (Violating a Nogood) A partial assignment θ for a CSP $\mathcal{P}$ *violates* a nogood α for $\mathcal{P}$ if θ is an extension of α.

The verb 'violates' is justified here, as we also view a partial assignment, and hence a nogood, as a constraint, with the form of a conjunction of equations. Strictly speaking, this notion of nogood violation is redundant with the notion of nogood extension, but we keep it for its more intuitive appeal.

Any extension of a nogood is itself a nogood:

Proposition 1 *If a partial assignment θ for a CSP $\mathcal{P}$ violates a nogood for $\mathcal{P}$, then θ is itself a nogood for $\mathcal{P}$.*

Proof Assume that a partial assignment θ for $\mathcal{P}$ violates a nogood α for $\mathcal{P}$ and assume that θ is not a nogood for $\mathcal{P}$. Then there exists a completion γ of θ such that $\gamma \in Sol(\mathcal{P})$. Since γ is a completion of θ and θ is an extension of α, we have, by transitivity of $\subseteq$ and $=$, that γ is a completion of α. But then α cannot be a nogood, since $\gamma \in Sol(\mathcal{P})$. This is a contradiction, so θ must be a nogood. □

Furthermore, a partial assignment that can only be extended into a nogood is itself a nogood:

Proposition 2 *Let $\mathcal{P} = \langle V, D, C\rangle$ be a CSP where $D = \{d_1, \ldots, d_m\}$. Let α be a partial assignment for $\mathcal{P}$ with scope$(\alpha) = \{v_{i_1}, \ldots, v_{i_k}\}$. If every α & $v_{i_{k+1}} = d_i$ $(1 \leq i \leq m)$ is a nogood for $\mathcal{P}$, then α is itself a nogood for $\mathcal{P}$.*

Proof Assume that α is not a nogood for $\mathcal{P}$. Then there exists an extension γ of α such that $\gamma \in Sol(\mathcal{P})$. But γ must include α & $v_{i_{k+1}} = d_i$ for some $1 \leq i \leq m$. But then γ is also an extension of α & $v_{i_{k+1}} = d_i$ for that i. But α & $v_{i_{k+1}} = d_i$ is a nogood for every $1 \leq i \leq m$, so there cannot exist such a γ. Hence α is a nogood for $\mathcal{P}$. □

In other words, nogoods can be lifted from the children to their parent in a search tree: when all the child nodes have been explored, their nogoods can be forgotten and only the parent nogood needs to be kept.

With respect to the symmetry considered in this paper, in [5], two definitions of symmetry are presented: solution symmetries, which are essentially bijections on the set of variable-value pairs that make up assignments and preserve solutions; and constraint symmetries, which are bijections on the structure of the constraints in the problem. It is shown that the group of constraint symmetries of a CSP is a subgroup (most often strict) of the group of solution symmetries. In this paper, the symmetries are defined as subgroups of the set of solution symmetries without reference to the constraint symmetries. Specific definitions of the particular symmetry considered are given in the respective parts of the paper.

3 Structural symmetry breaking for variable and value symmetry

We start our investigation by showing that there exists an efficient structural symmetry-breaking algorithm for CSPs where both the set of values and the set of variables can be partitioned into subsets such that, within each subset, all variables or values, respectively, are interchangeable. We call these problems piecewise interchangeable CSPs:

Definition 7 (Piecewise Bijection) Let $S = \cup_i P_i$ such that the sets P_i are disjoint, i.e., $P_i \cap P_j \neq \emptyset$ implies $i = j$. Then, we write $S = \sum_i P_i$ and call $\sum_i P_i$ a *partition* of S. A bijection $b : S \to S$ is a *piecewise bijection* over $\sum_i P_i$ if and only if $b(P_i) = P_i$, where $b(P_i) = \{b(e) \mid e \in P_i\}$.

Definition 8 (Piecewise / Fully Interchangeable CSP) A CSP $\mathcal{P} = \langle \sum_{k=1}^{m} V_k, \sum_{\ell=1}^{n} D_\ell, C\rangle$ is a *piecewise interchangeable CSP* if and only if, for each solution $\sigma \in Sol(\mathcal{P})$, each piecewise bijection a over $\sum_k V_k$, and each piecewise bijection b over

$\sum_\ell D_\ell$, we have $b \circ \sigma \circ a \in Sol(\mathcal{P})$. If the only piecewise bijection over $\sum_k V_k$ (or $\sum_\ell D_\ell$) is the identity, then the CSP is a *piecewise value-interchangeable (or variable-interchangeable) CSP*. If $m = 1$ (or $n = 1$), then the CSP is a *fully value-interchangeable CSP* (or a *fully variable-interchangeable CSP*). If $m = 1 = n$, then the CSP is a *fully interchangeable CSP*.

We will show how to break all symmetry in piecewise interchangeable CSPs by means of SBDD [9, 15]. SBDD is a technique to break symmetries during search. The idea is as follows: At any given choice point during search, we check whether the subtree rooted at the current node maps, under application of symmetry, into another subtree that has been fully explored earlier. If so, then the current node need not be investigated further and can be pruned. Different ways to control and limit the number of previously expanded subtrees that must be checked against have been developed in [9, 15]. With these results, the core procedure of any SBDD code that determines its efficiency is the dominance detection algorithm that checks whether a given partial (set) assignment is dominated by another one. Formally, we define:

Definition 9 (Dominating an Assignment) Let $\mathcal{P} = \langle \sum_k V_k, \sum_\ell D_\ell, C \rangle$ be a piecewise interchangeable CSP. Assignment α *dominates* assignment β if and only if there exist piecewise bijections a over $\sum_k V_k$ and b over $\sum_\ell D_\ell$ such that for every $v \in scope(\alpha)$ we have $\beta(a(v)) = b(\alpha(v))$.

Given two assignments α and β for a piecewise interchangeable CSP, we call the problem of determining whether α dominates β the *dominance detection problem*. Consequently, if we can solve the dominance detection problem efficiently, then we can also break symmetries efficiently.

The key idea to tackle the dominance detection problem for piecewise interchangeable CSPs consists in the introduction of structural abstractions: to model a CSP, we need to name uniquely each value and each variable. When certain variables and certain values are actually interchangeable, such a naming is of course not natural. We can rectify this by viewing each variable and each value as a member of a symmetry class. In the beginning, these classes correspond directly to the sets V_k and D_ℓ. When assignments are committed, though, some of those initial symmetries are broken. Then, in order to check which CSP objects are still interchangeable, we need to introduce subclasses of the original symmetry classes. We will see that we can detect the remaining symmetries by naming each of those subclasses with an appropriate signature that is defined by the set of initial symmetries and the given assignments. We will see also that it is really these signatures that capture our intuitive wish to abstract from the CSP model at hand to the actual structure of the problem.

3.1 Signatures

First consider the following example.

Example 3 Take variables $V = \{v_1, \ldots, v_8\}$ over the domain $D = \{d_1, \ldots, d_6\}$. Assume that the first four and the last four variables are interchangeable: $V_1 = \{v_1, \ldots, v_4\}$ and $V_2 = \{v_5, \ldots, v_8\}$. Assume that the first three and the last three val-

ues are interchangeable: $D_1 = \{d_1, \ldots, d_3\}$ and $D_2 = \{d_4, \ldots, d_6\}$. Consider the following two partial assignments (see Fig. 1a): $\alpha_1 = (v_1 = d_1 \ \& \ v_2 = d_1 \ \& \ v_3 = d_2 \ \& \ v_6 = d_5 \ \& \ v_7 = d_1 \ \& \ v_8 = d_2)$ and $\alpha_2 = (v_1 = d_6 \ \& \ v_2 = d_1 \ \& \ v_3 = d_2 \ \& \ v_4 = d_2 \ \& \ v_5 = d_1 \ \& \ v_6 = d_6 \ \& \ v_7 = d_2 \ \& \ v_8 = d_2)$. When looking at α_1, we see that:

1. There is one value (namely d_1) in D_1 that is taken by two variables in V_1 and one variable in V_2.
2. There is one value (namely d_2) in D_1 that is taken by one variable in V_1 and one variable in V_2.
3. There is one value (namely d_5) in D_2 that is taken by one variable in V_2.

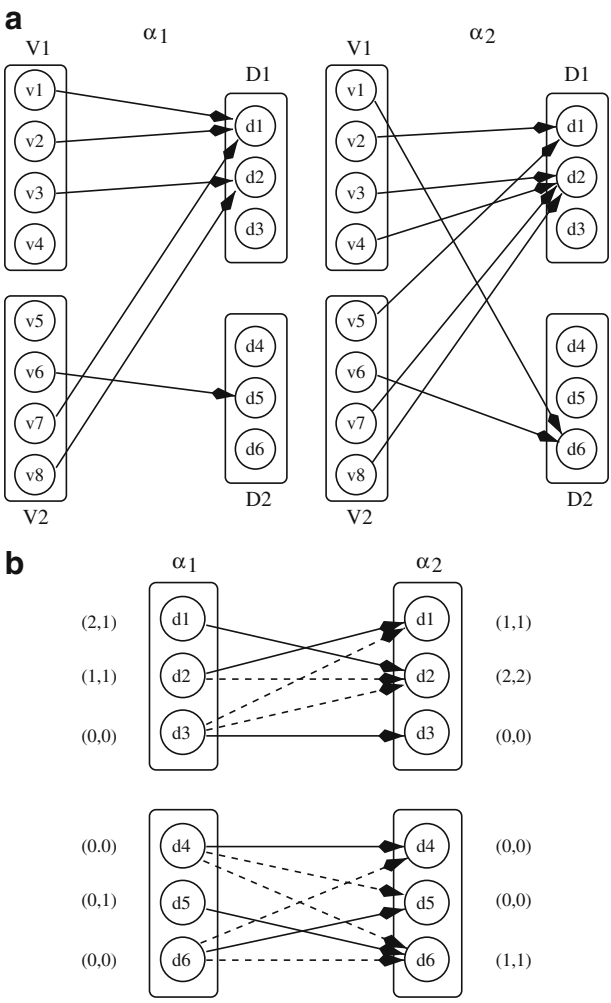

Fig. 1 Part **a** illustrates assignments α_1 and α_2. Part **b** gives the signatures for each value, links pairs of values where the one in assignment α_1 dominates the one in α_2, and designates by *solid lines* a perfect matching that proves that α_1 dominates α_2

On the other hand, in α_2, we see that:

I. There is one value (namely d_2) in D_1 that is taken by two variables in V_1 and two variables in V_2.
II. There is one value (namely d_1) in D_1 that is taken by one variable in V_1 and one variable in V_2.
III. There is one value (namely d_6) in D_2 that is taken by one variable in V_1 and one variable in V_2.

Lining up 1-I ($d_1 \mapsto d_2$, $\{v_1, v_2\} \mapsto \{v_3, v_4\}$, $\{v_7\} \mapsto \{v_7, v_8\}$), 2-II ($d_2 \mapsto d_1$, $\{v_3\} \mapsto \{v_2\}$, $\{v_8\} \mapsto \{v_5\}$), and 3-III ($d_5 \mapsto d_6$, $\{v_6\} \mapsto \{v_6\}$), we see that α_2 is structurally a partial assignment extended from α_1, or, in other words, that α_1 dominates α_2 (see also Fig. 1b).

What we have done in this small example is to abstract from the given model and the (arbitrary) names of the variables and values to the actual structure of the problem. That is, instead of talking about specific variables and values, we have considered members of classes. Specifically, for each partial assignment, we implicitly assigned each value a signature that captures by how many members of each variable-symmetry class it was taken. For instance, in α_1, the value d_1 has the signature $(2 \times V_1, 1 \times V_2)$, or, in shorter writing, the signature of d_1 under α_1 is $sig_{\alpha_1}(d_1) = (2, 1)$. Under α_2, on the other hand, the signature of d_2 is $sig_{\alpha_2}(d_2) = (2, 2)$. Consequently, d_2 in α_2 can be viewed as more specialised than d_1 in α_1, or one may also say that d_1 in α_1 dominates d_2 in α_2. In this terminology, d_1 in α_2 has signature $sig_{\alpha_2}(d_1) = (1, 1)$ and therefore dominates d_1 in α_1. Note that $sig_{\alpha_2}(d_6)$ is also $(1, 1)$, but that d_6 in α_2 does not dominate d_1 in α_1 since $d_6 \in D_2$ whereas $d_1 \in D_1$. In general:

Definition 10 (Dominating a Value) A value d in a partial assignment α *dominates* a value e in a partial assignment β if and only if d and e belong to the same value-symmetry class and $sig_\alpha(d) \leq sig_\beta(e)$.[2]

A value d in a partial assignment α is *structurally equivalent* to a value e in a partial assignment β if and only if d and e belong to the same value-symmetry class and $sig_\alpha(d) = sig_\beta(e)$.

In the following sub-section, we will show how these notions of dominance and structural equivalence can be exploited to devise a polynomial-time algorithm that solves the dominance detection problem on piecewise interchangeable CSPs.

3.2 Dominance detection using signatures

The following lemma shows how signature abstractions can help to detect dominance relations among partial assignments:

Lemma 1 *A partial assignment α dominates another partial assignment β in a piecewise interchangeable CSP if and only if there exists a piecewise bijection b over $D = \sum_\ell D_\ell$ such that d in α dominates $b(d)$ in β for every $d \in D$.*

[2]The $\leq$-relation on vectors is defined as the usual component-wise comparison that yields the so-called dominance ordering, which is different from a lexicographic ordering.

Proof First, assume that α dominates β. Then, there exist piecewise bijections a over $\sum_k V_k$ and b over $\sum_\ell D_\ell$ such that for every $v \in scope(\alpha)$ we have $\beta(a(v)) = b(\alpha(v))$. Since both v and $a(v)$ belong to the same symmetry class, we have $sig_\alpha(d) \leq sig_\beta(b(d))$ for all values $d \in D$, which is the same as to say that d in α dominates $b(d)$ in β.

Second, assume that there exists a piecewise bijection b over $\sum_\ell D_\ell$ such that $sig_\alpha(d) \leq sig_\beta(b(d))$ for every $d \in D$. Then, since each variable is assigned to at most one value, there exists a piecewise bijection a over $\sum_k V_k$ such that $\beta(a(v)) = b(\alpha(v))$ for every $v \in scope(\alpha)$. Thus, we have that α dominates β. □

Consequently, we have that α dominates β if and only if there exists a perfect matching in a bipartite graph where the edges are defined by the signature relation of values (see Fig. 1b). Let us denote by D' a set of duplicates of the values in D obtained by appending a prime sign to their names (that is, $D' := \{d' \mid d \in D\}$).

Definition 11 (Dominance Detection Graph) Given two partial assignments α and β, the *dominance detection graph* $DDG(\alpha, \beta)$ is $(D \cup D', E)$, where $E := \{(d, e') \mid d \text{ in } \alpha \text{ dominates } e \text{ in } \beta\}$ denotes the set of arcs.

Theorem 1 *The dominance detection problem between two partial assignments α and β for a piecewise interchangeable CSP has complexity $O(M + m^2 + mn)$, where $M = O(m^{2.5})$ is the time needed to determine whether there exists a perfect matching in $DDG(\alpha, \beta)$, with m being the number of values and n the number of variables. Hence all symmetric subtrees caused by value and variable symmetries of a piecewise interchangeable CSP can be eliminated with a polynomial time overhead at every node explored.*

Proof With Lemma 1, it is clear that the dominance detection problem can be solved basically by determining whether there exists a perfect bipartite matching in $DDG(\alpha, \beta)$. The additional complexity denoted in the theorem is due to the necessity of constructing $DDG(\alpha, \beta)$ first. This can be achieved in time $O(nm^2)$, which already proves that symmetry breaking in this scenario is tractable. However, the runtime can be improved to the complexity that is claimed here by using sparse representations of signatures. Instead of writing down entire signatures, for each value we hold a sparse list that only contains the non-zero entries of a signature, together with the information to which variable partition an entry in the sparse list belongs. To set up this sparse representation, we first order the variable instantiations in a given partial assignment according to the partition that the corresponding variable belongs to. This can be done in time linear in the number of variables, since this is also the maximum number of symmetry classes that can exist. In this order, we now scan through the partial assignments and set up the sparse signatures. Then, we iterate through the signatures of all the values in α and compare them with all the signatures of the values in β. With the sparse representation of signatures, this takes time $O(m(|\alpha| + |\beta|))$. □

This was the first result establishing that all the symmetries of a piecewise interchangeable CSP can be broken in polynomial time, in the sense that there are

no symmetrical subtrees in the remaining search tree. Note that this fact does not contradict the results from [35], which show that filtering all assignments that do not obey static constraints for breaking piecewise symmetry is NP-hard. As it is the case with some constraints (see for example the shorter path constraints [28]) where the constraint check is tractable while the filtering problem is not, we have shown that detecting a symmetrically dominated search node can be done in polynomial time while [35] has shown that identifying assignments that must eventually lead to a symmetrically dominated solution (in the sense that it is not the selected representative of an equivalence class of solutions) is hard.

Interestingly, it can also be shown that *every* bipartite graph can also be viewed as a dominance detection graph of a CSP and assignments α and β that can be determined in time linear in the size of the given graph. Therefore, a perfect bipartite matching exists if and only if α dominates β, which makes the dominance detection problem at least as hard as bipartite matching. In other words, we can show that dominance detection takes time T, where $T \in \Omega(M) \cap O(M + m^2 + mn)$.

Theorem 1 trivially has two interesting consequences. First, dropping the assumed piecewise *value* interchangeability or tightening the assumed *piecewise* interchangeabilities into *full* interchangeabilities will not worsen its tractability result, hence all symmetric subtrees caused by the symmetries of fully or piecewise *variable*-interchangeable CSPs and of *fully* value- and variable-interchangeable CSPs can be eliminated with a polynomial time overhead at every node explored. Conversely, when dropping the assumed piecewise *variable* interchangeability, we achieve tractability for the symmetries of fully and piecewise *value*-interchangeable CSPs. We will study these special cases later where we will devise highly efficient symmetry breaking methods that do not require complex matchings to be solved and that minimise the computational overhead needed for symmetry breaking in these special cases.

Second, dropping the assumed piecewise value interchangeability and switching to set-CSPs will not worsen the tractability result. Indeed, set variables that take subsets of a universe of *non*-interchangeable values can be seen as scalar variables that take scalar values from a domain of *non*-interchangeable values, hence the tractability results of symmetry breaking are those for fully or piecewise variable interchangeability of (scalar) CSPs: all symmetric subtrees caused by the symmetries of fully or piecewise *variable*-interchangeable set-CSPs can be eliminated with a polynomial time overhead at every node explored. These are special cases of Theorem 10 in Section 7.3.

4 Symmetry-based filtering

With Theorem 1, we can eliminate all symmetric subtrees caused by the symmetries of a piecewise interchangeable CSP in polynomial time at every node explored when using a symmetry-breaking by dominance detection (SBDD) approach [9, 15]. What is annoying in this setting is that we still have to check at every choice point to see whether it is not dominated by one that was previously expanded, that is we still have to touch the garbage in order to see that it is garbage. We will now develop an algorithm that does not suffer from this disadvantage.

We achieve this goal by using dominance detection also for *filtering* rather than just for pruning.[3] A brute-force approach could try assignments out and use the dominance detection algorithm above to perform filtering as well. This procedure would lead to a very poor runtime, though. In the following, we will show that filtering based on symmetry can be performed much more efficiently.

Within SBDD, there exists a natural distinction between two types of filtering that apply: The first consists in making sure that none of the newly created children are symmetric to a node that was fully expanded before the node that is currently branching off. When applying unary branching constraints (which we assume are used here), this can be achieved by shrinking the domains of variables accordingly. The other, fundamentally different type of "filtering" consists in the creation of children that are also not symmetric to each other. Both types need to be addressed to achieve a symmetry-free search tree (which corresponds to the GE-trees in [26]). We distinguish the two types of filtering by naming them differently: *symmetric-ancestor based filtering* and *symmetric-sibling based filtering*.

4.1 Symmetric-ancestor based filtering

The goal of symmetric-ancestor based filtering is to shrink the domains such that instantiating a variable with one of its domain values will not result in the creation of a search node that is symmetric to one that was previously expanded.

Definition 12 (Ancestor-Symmetry Resistance) Given a depth-first-search tree T, we say that a choice point β (associated with its homonymous partial assignment β that captures previously committed unary branching decisions) is *ancestor-symmetry resistant* if and only if for all previously fully expanded nodes $\alpha \in T$ (where α is called an *ancestor* of β) and for all variables v and values $d \in Dom(v)$ we have that α does not dominate $\beta \ \& \ (v = d)$.

Assume that we are currently investigating choice point β and that α is some ancestor node that does not dominate β. Observe that instantiating one more variable $v \in V_k$ for some k by setting $v \mapsto e \in D_\ell$ for some ℓ will change only the signature of e from $sig_\beta(e)$ to $sig_\beta(e) + e_k$, where e_k denotes the unit vector with a 1 in the kth component. We set $\beta' := \beta \ \& \ (v = e)$. Then, $G_1 := DDG(\alpha, \beta)$ and $G_2 := DDG(\alpha, \beta')$ only differ in that the latter bipartite graph may contain some additional edges that must all be incident to e' in the right partition. If G_2 contains an m-matching, this matching must contain exactly one of those additional edges. Consequently, if α dominates β', then G_1 must contain an $(m-1)$-matching. Only if this is the case, work needs to be done to make β ancestor-symmetry resistant with respect to α.

So let us assume that G_1 contains an $(m-1)$-matching. Provided with that matching, using some straightforward matching theory we can identify efficiently those and only those additional edges that would allow us to transform the existing

[3] With 'filtering', we refer to the idea of domain reduction in constraint programming, whereas with 'pruning', we refer to the detection of a sufficient reason for backtracking.

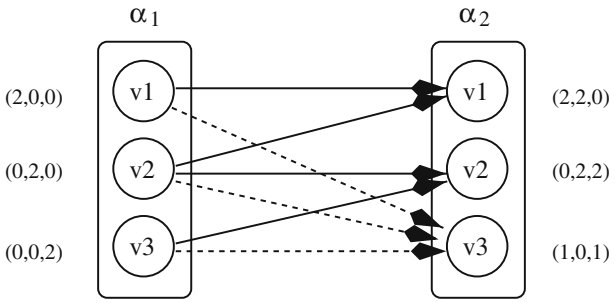

Fig. 2 Ancestor-based filtering: to the left and right of the bipartite graph are the signatures of symmetric values $v1$, $v2$, and $v3$ under two assignments α_1 and α_2 for a problem where the variable partition has three parts. *Solid lines* indicate dominance relationships between values. *Dashed lines* indicate critical edges

matching into a perfect one (for an introduction to matching theory we refer to [1]): a matching can be viewed as a flow in some network that closely corresponds to the bipartite graph. We consider the usual residual network with respect to that flow that has an additional source node s that connects to all nodes in the first partition, and a sink node t that is connected from all nodes in the second partition. The capacities of the residual network are given by the residual capacity of edges after the flow has been routed, including reverse edges for edges with positive flow.

Then, a maximum matching (corresponding to a maximum flow) defines two cuts. The first is given by the nodes that are reachable from the source in the residual network. If we denote this set with S, then (S, S^C) is an s-t-cut. The second cut is given by the set of nodes from which the sink is reachable in the residual network. If we denote this set with T, then (T^C, T) is also an s-t-cut. Note that both cuts can be computed in time linear in the size of the given networks after the maximum matching has been computed.

Now, the core observation is that, given those two cuts, the critical edges are exactly those that run from $S \cap D$ to $T \cap D'$: clearly, adding such an edge yields an improving path in the residual network and therefore an m-matching. On the other hand, note that $T \subseteq S^C$ and $S \subseteq T^C$. Any edge added from S to $S^C \setminus T$ would leave the cut (T^C, T) untouched, which proves that no such edge can improve the matching. Edges that run from $T^C \setminus S$ to T follow analogously.

Among those critical edges that, if added, would allow us to construct an m-matching, the only ones that we need to consider are those that run between nodes d and e' with $d, e \in D_\ell$ for some ℓ and for which there exists k such that $sig_\alpha(d) \leq sig_\beta(e) + e_k$. If and only if we find such a pair of nodes, a single extra assignment added to β will result in a successful dominance detection. Precisely, every assignment of e to a previously unassigned variable $v \in V_k$ will result in a dominated choice point. Thus, if we remove e from the domain of v for every unassigned $v \in V_k$, we keep the unique parts of the search space and we never produce any choice points that are symmetric to one that was expanded previously to β.

Example 4 We illustrate ancestor-based filtering in Fig. 2. We consider a problem where the variable partition has three parts and where all three values are symmetric. The solid lines indicate the matching graph that yields an almost-perfect matching. The dashed lines indicate critical edges whose addition to the graph would yield a perfect matching. Our algorithm implicitly enumerates those critical edges (of which there may exist an exponential number) to find critical assignments that

would lead to a successful dominance check. In this example, any assignment of value $v3$ to any unassigned variable in variable parts 1 or 3 would lead to a successful dominance check. Consequently, value $v3$ must be removed from the domains of all such variables.

In summary, with Theorem 1, the runtime needed for the initial value-matching algorithm is bounded by $O(m^{2.5} + mn)$. Then, the entire filtering algorithm runs in time $O(m^2 + mn)$. Therefore, since within SBDD at most $n(m-1)$ ancestor nodes need to be considered, we can prove the following theorem:

Theorem 2 *For a piecewise interchangeable CSP, we can achieve ancestor-symmetry resistance for a given search node in time* $O(nm^{3.5} + n^2m^2)$.

4.2 Symmetric-sibling based filtering

To achieve full symmetry prevention, we also need to guarantee that newly created siblings are not symmetric to each other. Therefore, after choosing the next variable to be assigned, but before branching on it, we need to perform one more "filtering" step (it is actually more of an implicit pruning step), where we choose a single representative value out of each equivalence class of values that, when assigned to the chosen variable, would result in the creation of symmetric choice points. Due to the fact that, whenever a sibling dominates another one, they both must already be structurally equivalent (see Definition 10), we can avoid producing symmetric siblings by choosing exactly one representative value among those that are structurally equivalent. The complexity of this filtering step is dominated by that of symmetric-ancestor based filtering.

Putting ancestor and sibling-based filtering together, we have completed our development of an effective symmetry-breaking algorithm for piecewise interchangeable CSPs that runs in polynomial time. Note that the practical performance of the algorithms sketched can be enhanced in practise: for example, it is fully sufficient to check against previously expanded nodes for which an $(m-1-h)$-maximum matching was found only after variable instantiations to h different values have been committed [18].

5 Fast algorithms to break value symmetry

We review the special case of piecewise interchangeable CSPs with no variable symmetry, which we call piecewise value-interchangeable CSPs. With the results of the previous section, we know that this symmetry breaking can be achieved in polynomial time. Now, we focus on the development of specialised algorithms that break value symmetry with minimal overhead. First, in Section 5.1, we describe our new approach on the class of *fully* value-interchangeable CSPs, showing how it leads to the well-known result from combinatorial enumeration (e.g., [8, 19]) that all symmetric subtrees caused by their value symmetries can be eliminated by a dedicated search procedure with a *constant* overhead with respect to both time and space at every node explored (Theorem 4). Then, following the same approach, we show in Sections 5.2 and Section 5.3 that this result actually generalises to

piecewise value-interchangeable CSPs (Theorem 5) and even holds for fully value-interchangeable *set*-CSPs (Theorem 6).

5.1 Fully value-interchangeable CSPs

When all values are interchangeable and no variable symmetry is present, we speak of a fully value-interchangeable CSP.

Definition 13 (Fully Value-Interchangeable CSP) A CSP $\mathcal{P} = \langle V, D, C\rangle$ is a *fully value-interchangeable CSP* if, for each solution $\sigma \in Sol(\mathcal{P})$ and each bijection b over D, we have $b \circ \sigma \in Sol(\mathcal{P})$.

In the following, we show that in this case symmetry breaking can be performed with *constant* overhead with respect to both time and space at every node explored. Our method is based on nogoods. The following theorem gives a fundamental characterisation of nogoods for fully value-interchangeable CSPs. It states that nogoods are preserved under value interchanges:

Theorem 3 *Let α be a nogood for a fully value-interchangeable $\mathcal{P} = \langle V, D, C\rangle$ and let $b : D \to D$ be a bijection. Then $b \circ \alpha$ is a nogood for $\mathcal{P}$.*

Proof Let g be a completion of $b \circ \alpha$ and assume that $g \in Sol(\mathcal{P})$. Since b is a bijection, we have $b^{-1} \circ g \in Sol(\mathcal{P})$. But $(b^{-1} \circ g)(v) = (b^{-1} \circ (b \circ \alpha))(v) = \alpha(v)$, for all $v \in scope(\alpha)$, by the definition of a completion, that is α can be extended into a solution. This contradicts the fact that α is a nogood. Hence $b \circ \alpha$ cannot be extended into a solution and is thus actually a nogood. □

The closure of a nogood α for a fully value-interchangeable CSP is the set of nogoods obtained from α by applying each possible value interchange, or value symmetry, to α:

Definition 14 (Closure of a Nogood) Let α be a nogood for a fully value-interchangeable $\mathcal{P} = \langle V, D, C\rangle$. The *closure* of α for $\mathcal{P}$, denoted by $Closure(\alpha, \mathcal{P})$, is the set $\{b \circ \alpha \mid b \text{ is a bijection over } D\}$.

The main idea of our approach is to try and abstract such closures of nogoods so that their representation takes polynomial space and that membership to a closure can be tested during search in polynomial time. It will then become possible to write a search procedure that eliminates all symmetric subtrees caused by the value symmetries by never extending any member of the closures of all the nogoods generated during search.

5.1.1 Abstract nogoods

We first show that the closure of a nogood for a fully value-interchangeable CSP can be characterised compactly.

Definition 15 (Abstract Nogood) Let α be a nogood for a fully value-interchangeable $\mathcal{P} = \langle V, D, C\rangle$. Let $image(\alpha) = \{d_1, \ldots, d_k\}$ and let $v_{r_i} \in \alpha^{-1}(d_i)$, for

$1 \leq i \leq k$. The *abstract nogood* of α with respect to $\mathcal{P}$, denoted by $Anogood(\alpha, \mathcal{P})$, is the set of all functions $\gamma : scope(\alpha) \rightarrow D$ satisfying the condition

$$\forall i \in 1 \ldots k : allequal\left(\gamma(v_j) \mid v_j \in \alpha^{-1}(d_i)\right) \ \& \ alldiff\left(\gamma(v_{r_1}), \ldots, \gamma(v_{r_k})\right)$$

where $allequal(a_1, \ldots, a_n)$ holds if all the a_i are the same value, and $alldiff\,(a_1, \ldots, a_n)$ holds if all the a_i are different values (and is not to be mixed up with the *allDifferent* global constraint).

By abuse of language, we identify an abstract nogood, which is a set of functions, with the condition that its members have to satisfy.

Example 5 Consider a nogood β, written as a conjunction of equations:

$$\beta(v_1) = 1 \ \& \ \beta(v_2) = 2 \ \& \ \beta(v_3) = 3 \ \& \ \beta(v_4) = 1 \ \& \ \beta(v_5) = 2$$

The abstract nogood of β is the following condition:

$$allequal\big(\gamma(v_1), \gamma(v_4)\big) \ \& \ allequal\big(\gamma(v_2), \gamma(v_5)\big) \ \& \ alldiff\left(\gamma(v_1), \gamma(v_2), \gamma(v_3)\right)$$

or, more precisely, the set of functions $\gamma : scope(\beta) \rightarrow D$ satisfying this condition.

An abstract nogood precisely captures the closure of its nogood [14, Lemmas 2 and 3], and membership to the closure of a nogood can be tested in linear time [14, Lemmas 4 and 5]. Abstract nogoods are needed only for the current *frontier nodes* of the search tree (i.e., the closed nodes whose parents are open). Once its child nodes are explored, the abstract nogood of a parent node subsumes the abstract nogoods of these child nodes. Hence, maintaining the nogood takes space $O(|F||V|)$, where F is the set of frontier nodes [14, Theorem 4].

5.1.2 Maintaining nogoods

We now show that search procedures exploring a search tree for a fully value-interchangeable CSP can remove *all* the value symmetries while causing only *constant* overhead with respect to both time and space at every node explored. Before presenting the theoretical results, we illustrate the idea using an example with depth-first search. The basic intuition comes from the structure of the abstract nogoods.

Example 6 Consider the partial assignment

$$\theta(v_1) = 1 \ \& \ \theta(v_2) = 2 \ \& \ \theta(v_3) = 3 \ \& \ \theta(v_4) = 1 \ \& \ \theta(v_5) = 2$$

and assume that depth-first search tries next to label variable v_6, whose set of possible values is $1 \ldots 10$. The failure of $v_6 = 1$ produces the abstract nogood

$$allequal(\gamma(v_1), \gamma(v_4), \gamma(v_6)) \ \& \ allequal(\gamma(v_2), \gamma(v_5)) \ \& \ alldiff\,(\gamma(v_1), \gamma(v_2), \gamma(v_3)).$$

Since $v_1, \ldots, v_5$ remain instantiated when the next value is tried for v_6, the abstract nogood for this part of this next branch simplifies to $\gamma(v_6) = 1$, imposing that v_6 be labelled with a value different from 1. The failures of $v_6 = 2$ and $v_6 = 3$ produce

Fig. 3 A labelling procedure for fully value-interchangeable CSPs

```
bool fValIlabel(P) {
    return fValIlabelA(P,ϵ);
}
bool fValIlabelA(⟨V, D, C⟩,θ) {
    if scope(θ) = V then
        return C(θ);
    select v in V \ scope(θ);
    A := image(θ);
    if A ≠ D then
        select f in D \ A; A := A ∪ {f};
    forall(d ∈ A)
        θ′ := θ & v = d;
        if ¬Failure(⟨V, D, C⟩, θ′) then
            if fValIlabelA(⟨V, D, C⟩,θ′) then
                return true;
    return false;
}
```

similar abstract nogoods for the other values already used in θ. Now consider the values not already used in θ and observe what happens for a failed labelling of v_6 with a value in $4 \ldots 10$, say 6. The abstract nogood then is

$$\mathit{allequal}(\gamma(v_1), \gamma(v_4)) \;\&\; \mathit{allequal}(\gamma(v_2), \gamma(v_5)) \;\&\; \mathit{alldiff}\big(\gamma(v_1), \gamma(v_2), \gamma(v_3), \gamma(v_6)\big)$$

which simplifies to $\mathit{alldiff}(1, 2, 3, \gamma(v_6))$. The disjunction of the four simplified abstract nogoods obtained so far is the condition

$$\gamma(v_6) = 1 \;\vee\; \gamma(v_6) = 2 \;\vee\; \gamma(v_6) = 3 \;\vee\; \mathit{alldiff}(1, 2, 3, \gamma(v_6))$$

which must not be satisfied by any labelling of v_6. It follows that v_6 need only be labelled with the previously used values in $1 \ldots 3$ or with exactly one new value in $4 \ldots 10$.

In other words, in a search tree, only *some* of the child nodes of a partial assignment θ need to be explored, namely those that label the next variable $v_{i_{k+1}}$ with a value in $\mathit{image}(\theta)$ or with exactly one other value. Clearly, deciding in this fashion which child nodes to explore only takes constant time. *Note that this result is independent of the set of constraints.* It is the essence of the labelling procedure for graph colouring in [19] and of the set-partition enumeration procedure in [8]. This procedure, which eliminates all symmetric subtrees caused by the value symmetries for fully value-interchangeable CSPs, is formalised in Fig. 3 as procedure `fValIlabel`. It uses a function $\mathit{Failure}(\mathcal{P}, \theta)$, which returns **false** if at least one extension of the partial assignment θ is a solution to the CSP $\mathcal{P} = \langle V, D, C\rangle$. In other words, it satisfies the property

$$\mathit{Failure}(\mathcal{P}, \theta) \Rightarrow \forall \beta \in \mathit{Comp}(\theta, \mathcal{P}) \;:\; \neg C(\beta).$$

We establish the correctness of `fValIlabel`:

Theorem 4 *Procedure* `fValIlabel` *eliminates all symmetric subtrees caused by the value symmetries of a fully value-interchangeable CSP with a constant overhead with respect to both time and space at every node explored, i.e., it never extends any member of the closure of any nogood generated during search.*

Proof See the proof of Theorem 5 in [14].

Other search strategies, e.g., limited-discrepancy search, can also be adapted to remove all the value symmetries of fully value-interchangeable CSPs with a *constant* overhead with respect to both time and space at every node explored.

Experiments establishing speed-ups of several orders of magnitude with this known labelling procedure have been reported elsewhere, e.g., in [19, 32, 33].

5.2 Piecewise value-interchangeable CSPs

We now present a generalisation for *piecewise* value-interchangeable CSPs of the previous results.

Definition 16 (Piecewise Value-Interchangeable CSP) A CSP $\mathcal{P} = \langle V, \sum_\ell D_\ell, C\rangle$ is a *piecewise value-interchangeable CSP* if, for each solution $\sigma \in Sol(\mathcal{P})$ and each piecewise bijection b over $\sum_\ell D_\ell$, we have $b \circ \sigma \in Sol(\mathcal{P})$.

Example 7 For scene allocation (see Example 1), we can imagine a version of the problem where the days are divided into morning and afternoon sessions. The actors probably have strong preferences (and thus different fees for these sessions), but the day of the session may still not matter.

Definition 17 (Closure of a Nogood) Let α be a nogood for a piecewise value-interchangeable CSP $\mathcal{P} = \langle V, \sum_\ell D_\ell, C\rangle$. The *closure* of α for $\mathcal{P}$, denoted by $Closure(\alpha, \mathcal{P})$, is the set $\{b \circ \alpha \mid b$ is a piecewise bijection over $\sum_\ell D_\ell\}$.

We now define abstract nogoods for piecewise value-interchangeable CSPs. The key intuition is to separate the values from each D_ℓ.

Definition 18 (Abstract Nogood) Let α be a nogood for a piecewise value-interchangeable CSP $\mathcal{P} = \langle V, D, C\rangle$, where $D = \sum_{\ell \le s} D_\ell$. Let $image(\alpha) = \{d_1^1, \ldots, d_{s_1}^1, \ldots, d_1^s, \ldots, d_{s_s}^s\}$, where $d_i^\ell \in D_\ell$, and let $v_{r_i^\ell} \in \alpha^{-1}(d_i^\ell)$, for $1 \le i \le s_\ell$ and $1 \le \ell \le s$. The *abstract nogood* of α with respect to $\mathcal{P}$, denoted by $Anogood(\alpha, \mathcal{P})$, is the set of all functions $\gamma : scope(\alpha) \to \sum_\ell D_\ell$ satisfying the condition

$$\forall i \in 1 \ldots s_\ell : allequal\left(\gamma(v_j) \mid v_j \in \alpha^{-1}\left(d_i^\ell\right)\right) \;\&$$

$$\forall i \in 1 \ldots s_\ell : \forall v_j \in \alpha^{-1}\left(d_i^\ell\right) : v_j \in D_\ell \;\&\; alldiff\left(\gamma\left(v_{r_1^\ell}\right), \ldots, \gamma\left(v_{r_{s_\ell}^\ell}\right)\right)$$

for $1 \le \ell \le s$.

Figure 4 depicts the labelling procedure `pValIlabel` for piecewise value-interchangeable CSPs. It generalises `fValIlabel` of Fig. 3 by considering the already assigned values in the sets D_ℓ, as well as one new value (if any) from each set: the procedure `fValIlabel` is obtained when the partition of D has only one part (that is, when $s = 1$). Its correctness proof is similar to the one of Theorem 4.

Theorem 5 *Procedure* `pValIlabel` *eliminates all symmetric subtrees caused by the value symmetries of a piecewise value-interchangeable CSP with a constant overhead with respect to both time and space at every node explored.*

Experiments establishing significant speed-ups have been reported elsewhere, e.g., for partitioned graph colouring in [33].

5.3 Fully value-interchangeable set-CSPs

We now show that symmetry breaking for fully value-interchangeable set-CSPs is tractable. Given a finite set S, we denote by 2^S the set of subsets of S.

Definition 19 (Set Bijection) A bijection $b : 2^S \to 2^S$ is a *set bijection* over 2^S if $b(T) = \{b'(e_i) \mid e_i \in T\}$ for $T \in 2^S$, where $b' : S \to S$ is a bijection. We say that b is *induced* by b'.

Definition 20 (Fully Value-Interchangeable Set-CSP) A set-CSP $\mathcal{P} = \langle V, 2^D, C\rangle$ is a *fully value-interchangeable set-CSP* if, for each solution $\sigma \in Sol(\mathcal{P})$ and each set bijection b over 2^D, we have $b \circ \sigma \in Sol(\mathcal{P})$.

Fig. 4 A labelling procedure for piecewise value-interchangeable CSPs

```
bool pValIlabel(P) {
    return pValIlabelA(P,ϵ);
}
bool pValIlabelA(⟨V, ∑_{ℓ≤s} D_ℓ, C⟩,θ) {
    if scope(θ) = V then
        return C(θ);
    select v in V \ scope(θ);
    forall(ℓ ∈ 1...s)
        A_ℓ := image(θ) ∩ D_ℓ;
    forall(ℓ ∈ 1...s)
        if A_ℓ ≠ D_ℓ then
            select f in D_ℓ \ A_ℓ; A_ℓ := A_ℓ ∪ {f};
    forall(d ∈ ⋃_ℓ A_ℓ)
        θ' := θ & v = d;
        if ¬Failure(⟨V, ∑_{ℓ≤s} D_ℓ, C⟩, θ') then
            if pValIlabelA(⟨V, ∑_{ℓ≤s} D_ℓ, C⟩, θ') then
                return true;
    return false;
}
```

To get an impression where such problems can be of interest, consider the following example.

Example 8 Let V be any set of v elements, called *varieties*. A *balanced incomplete block design* (BIBD) [6] is a multi-set of b subsets of V, called *blocks*, each of size k (constraint C_1), such that each pair of distinct varieties occurs together in exactly λ blocks (constraint C_2), with $2 \leq k < v$. Implied constraints are that each variety occurs in the same number of blocks (constraint C_3), namely $r = \lambda(v-1)/(k-1)$, as well as that $bk = vr$ and $\lambda < r$. A BIBD is parametrised by a 5-tuple $\langle v, b, r, k, \lambda \rangle$ of parameters, not all of which are independent. Originally intended for the design of statistical experiments, BIBDs also have applications in cryptography and elsewhere. Note that the varieties and the blocks are fully interchangeable. Finding a BIBD means finding a fixed number of same-size subsets of a fully interchangeable set: either find b subsets of size k of the set V, or, dually, find v subsets of size r of the set $\{1, \ldots, b\}$, subject to the constraint C_2.

Definition 21 (Closure of a Nogood) Let α be a nogood for a fully interchangeable set-CSP $\mathcal{P} = \langle V, 2^D, C \rangle$. The *closure* of α for $\mathcal{P}$, denoted by $Closure(\alpha, \mathcal{P})$, is the set $\{b \circ \alpha \mid b \text{ is a set bijection over } 2^D\}$.

5.3.1 Abstract nogoods

We now define abstract nogoods for fully value-interchangeable set-CSPs, first showing the intuition using Example 8. We take the first mentioned modelling approach (namely finding v subsets of size r of the set $\{1, \ldots, b\}$) and, for simplicity, only tackle the full interchangeability of the blocks. We will come back to the full interchangeability of the varieties just after Theorem 6.

Consider the $\langle 6, 10, 5, 3, 2 \rangle$ BIBD (which has one solution modulo all symmetries): we want to find $v = 6$ subsets v_i of size $r = 5$ of the universe $D = \{1, \ldots, 10(= b)\}$, each giving the blocks to which variety i of V belongs, such that each block is mentioned in $k = 3$ subsets and any two subsets have an intersection of size $\lambda = 2$. Consider the (consistent) partial assignment:

$$\alpha(v_1) = \{1, 2, 3, 4, 5\} \;\&\; \alpha(v_2) = \{1, 2, 6, 7, 8\} \;\&\; \alpha(v_3) = \{1, 3, 6, 9, 10\}$$

and assume that α becomes a nogood on backtracking. Note that the values 4 and 5 are indistinguishable because they are the only ones to appear only in the first set. Similarly, the value 6 is not indistinguishable from any other value because it is the only value that appears only in the second and third sets. Formally:

Definition 22 (Indistinguishable Values, Cluster) The values x and y are *indistinguishable* under a partial assignment θ, which is denoted by $x \sim y$, if $x \in \theta(v) \leftrightarrow y \in \theta(v)$ for all $v \in scope(\theta)$. The *clusters* of values that always appear together, and are thus indistinguishable, are the equivalence classes of $\sim$ in D under α.

In our example, there are seven clusters:

$$\{1\},\ \{2\},\ \{3\},\ \{4, 5\},\ \{6\},\ \{7, 8\},\ \{9, 10\}. \tag{1}$$

Definition 23 (Signature of a Cluster) The *signature* of a cluster c relative to a partial assignment θ, denoted by $sig(c, \theta)$, is the list of indices i of the variables v_i, which are given a set value by θ, of which c is a subset: $sig(c, \theta) = \{i \mid v_i \in scope(\theta) \ \& \ c \subseteq \theta(v_i)\}$.

For instance, $sig(\{3\}, \alpha) = [1, 3]$ because $\{3\}$ is a subset of both $\alpha(v_1)$ and $\alpha(v_3)$. The signatures of the seven clusters in (1) relative to α respectively are:

$$[1, 2, 3],\ [1, 2],\ [1, 3],\ [1],\ [2, 3],\ [2],\ [3]. \tag{2}$$

We get the following condition for the *abstract nogood* of α:

$$\begin{gathered} partition\big(D, \big[\big(\gamma(v_1) \cap \gamma(v_2) \cap \gamma(v_3)\big),\ \big(\gamma(v_1) \cap \gamma(v_2)\big) \setminus \gamma(v_3), \\ \big(\gamma(v_1) \cap \gamma(v_3)\big) \setminus \gamma(v_2),\ \gamma(v_1) \setminus \big(\gamma(v_2) \cup \gamma(v_3)\big),\ \big(\gamma(v_2) \cap \gamma(v_3)\big) \setminus \gamma(v_1), \\ \gamma(v_2) \setminus \big(\gamma(v_1) \cup \gamma(v_3)\big),\ \gamma(v_3) \setminus \big(\gamma(v_1) \cup \gamma(v_2)\big)\big], [1, 1, 1, 2, 1, 2, 2]\big) \end{gathered}$$

where the order of the clusters is the same as in (1), and where $partition(S, P, N)$ holds if the elements P_i of the set list P are non-empty, mutually disjoint, union up to the set S, and have N_i elements respectively, with the N_i being the elements of the integer list N. Note that the cluster size conditions are necessary in general, but actually implied in this example.[4] If there had been values of D that do not appear in any of the set values for the variables in the scope of α, then their cluster, which would have the empty list as signature, would have been equal to $D \setminus (\gamma(v_1) \cup \gamma(v_2) \cup \gamma(v_3))$, as D is the intersection of an empty collection of sets drawn from D.

We first show that the closure of a nogood for a fully value-interchangeable set-CSP can be characterised compactly and that membership to the closure of a nogood can be tested in polynomial time in this case.

Definition 24 (Abstract Nogood) Let α be a nogood for a fully value-interchangeable set-CSP $\mathcal{P} = \langle V, 2^D, C\rangle$. Let I be the set of indices of the variables of V that are in $scope(\alpha)$. Let E be the list of equivalence classes of $\sim$ in the universe D under α, and let N be the list of their respective sizes. The *abstract nogood* of α with respect to $\mathcal{P}$, denoted by $Anogood(\alpha, \mathcal{P})$, is the set of all functions $\gamma : scope(\alpha) \to 2^D$ satisfying the condition

$$partition\left(D, \left[\bigcap_{j \in sig(e,\alpha)} \gamma(v_j) \ \setminus \bigcup_{j \in I \setminus sig(e,\alpha)} \gamma(v_j) \mid e \in E\right], N\right).$$

An abstract nogood precisely captures the closure of its nogood [14, Lemmas 8 and 9].

[4] Consider a domain of five elements and a partial assignment for two set variables, S_1 and S_2, of size 3 that have one or two elements in common, that is $S_1 = e_1 \cup e_2$ and $S_2 = e_1 \cup e_3$ where e_1, e_2, e_3 are disjoint. Then e_1 can be of size 1 or 2.

5.3.2 Maintaining nogoods

Let us now consider depth-first search, for instance, and see what happens when the assignment to v_3 is undone, making α a nogood. By the definition of clusters, the search procedure should treat the elements of a cluster as indistinguishable. Then, imposing an ordering on the elements of each cluster, the idea is to select the i^{th} element of a cluster only when the $(i-1)^{\text{st}}$ element of that cluster has already been selected as a member for the next subset variable.

Figure 5 depicts the labelling procedure `fValIsetLabel` for fully value-interchangeable set-CSPs. It uses a function $Failure'(\mathcal{P}, \theta, v, S)$, which returns **false** if at least one extension of the partial assignment $\theta \ \& \ v = S \cup T$ for some $T \subseteq D$ is a solution to $\mathcal{P} = \langle V, 2^D, C \rangle$. In other words, it satisfies the property

$$Failure'(\langle V, 2^D, C\rangle, \theta, v, S) \Rightarrow$$
$$\forall T \subseteq D : |S \cup T| = n \ : \ \forall \beta \in Comp(\theta \ \& \ v = S \cup T, \langle V, 2^D, C\rangle) \ : \ \neg C(\beta).$$

Procedure `fValIsetLabel` also uses a procedure `UPDATE(`E`,`S`)`, which returns the equivalence classes (clusters) of $T \cup S$, with those of T being E.

Theorem 6 *Procedure* `fValIsetLabel` *eliminates all symmetric subtrees caused by the value symmetries of a fully value-interchangeable set-CSP with a constant overhead with respect to both time and space at every node explored.*

Proof See the proof of Theorem 7 in [14].

Procedure `fValIsetLabel` performs what is called *canonical labelling* in [10]. There, it is also shown that canonically labelling along one dimension of a matrix of variables amounts to lexicographically ordering (a flattening of) the other dimensions of that matrix. Experimental results have been reported elsewhere, e.g., in [10]. We conjecture that Theorem 6 generalises to piecewise value-interchangeable set-CSPs.

Let us now return to the full interchangeability of the v varieties. Breaking these extra $v!$ symmetries at the same time is hard, as they compose with the $b!$ block symmetries into $v! \cdot b!$ symmetries. Lexicographically ordering both the rows and the columns of the mentioned $v \times b$ matrix of zero/one variables does not break all these symmetries, but gives reasonable performance due to the constraint C_2 [11]. This leads to the issue whether a suitable abstract nogood can be formulated and a tractable labelling procedure be derived. In this case, it is not sufficient to store only the nogoods at the frontier nodes in the search tree; nogoods have to be stored from higher up in the search tree, as in SBDS [17] and SBDD [9]. Further, testing if a partial assignment extends a nogood is NP-complete. To see this, consider a BIBD where the blocks are of size 2; a nogood can then be thought of as a graph, each block specifying an edge. Then testing if a partial assignment is in the closure of the nogood is equivalent to subgraph isomorphism, which is NP-complete. A formal proof of this result will be given in the next section.

```
bool fValIsetLabel(⟨V, 2^D, C⟩) {
    return fValIsetLabelA(⟨V, 2^D, C⟩, ϵ, [D]);
}
bool fValIsetLabelA(⟨V, 2^D, C⟩, θ, E) {
    if scope(θ) = V then
        return C(θ);
    select v in V \ scope(θ);
    (b, S) := fValIsetLabelB(⟨V, 2^D, C⟩, θ, v, ∅, E);
    if b = true then
        θ' := θ & v = S;
        E' := UPDATE(E, S);
        return fValIsetLabelA(⟨V, 2^D, C⟩, θ', E');
    return false;
}
(bool, set) fValIsetLabelB(⟨V, 2^D, C⟩, θ, v, S, [e_1, e_2, ..., e_m]) {
    if |S| = n then
        return (true, S);
    S' := S ∪ {head(e_1)};
    if ¬Failure'(⟨V, 2^D, C⟩, θ, v, S') then
        (b, S'') := fValIsetLabelB(⟨V, 2^D, C⟩, θ, v, S', [tail(e_1), e_2, ..., e_m]);
        if b = true then
            return (true, S'');
    (b, S'') := fValIsetLabelB(⟨V, 2^D, C⟩, θ, v, S, [e_2, ..., e_m]);
    if b = true then
        return (true, S'');
    return false;
}
```

Fig. 5 A labelling procedure for fully value-interchangeable set-CSPs

6 Limits of efficient symmetry breaking

Until now, we have dealt with cases where symmetric subtrees could be eliminated efficiently. Particularly, we have shown how symmetric subtrees caused by piecewise variable and value symmetries can be eliminated efficiently, and we have given extremely low-overhead algorithms for some cases of value symmetry only. Unfortunately, as we will see in this section, there are limits to efficient symmetry breaking. We consider set-CSPs with interchangeable variables and values:

Definition 25 (Piecewise Interchangeable Set-CSP) A set-CSP $\mathcal{P} = \langle \sum_k V_k, 2^{\sum_\ell D_\ell}, C\rangle$ is a *piecewise interchangeable set-CSP* if, for each solution $\sigma \in Sol(\mathcal{P})$, each piecewise bijection a over $\sum_k V_k$, and each piecewise set bijection b over $2^{\sum_\ell D_\ell}$, we have $b \circ \sigma \circ a \in Sol(\mathcal{P})$.

When trying to break the symmetry in piecewise interchangeable set-CSPs by means of SBDD, we need to solve the following dominance detection problem efficiently.

Definition 26 (Dominating a Set Assignment) Let $\mathcal{P} = \langle \sum_k V_k, 2^{\sum_\ell D_\ell}, C \rangle$ be a piecewise interchangeable set-CSP. Set assignment α *dominates* set assignment β if and only if there exist a piecewise bijection a over $\sum_k V_k$ and a piecewise set bijection b over $2^{\sum_\ell D_\ell}$ such that for every $v \in scope(\alpha)$ we have $\beta(a(v)) = b(\alpha(v))$.

We will show that solving this problem is NP-hard, thus proving that SBDD is not able to break piecewise symmetry in set-CSPs efficiently. More precisely, we reduce the corresponding dominance detection problem to subgraph-isomorphism. To achieve the desired reduction, we construct a set assignment from a graph in the following way:

Definition 27 (Set Assignment α_G) Given an undirected graph $G = (V, E)$ with $c := |V|$, we create a set of interchangeable values $N := \{n_1, \ldots, n_c\}$ and a set of interchangeable variables $V := \{p_{ij} \mid \{i, j\} \in E\}$. Then, the *set assignment* α_G is defined as $\alpha_G := \bigwedge_{\{i,j\} \in E} (p_{ij} = \{n_i, n_j\})$.

Theorem 7 *Given two undirected graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$, we have that G_1 is sub-isomorphic to G_2 if and only if α_{G_1} dominates α_{G_2} when all variables and values are considered to be interchangeable.*

Proof We start by showing that α_{G_1} dominates α_{G_2} if G_1 is sub-isomorphic to G_2. Let $\sigma : V \to V$ be bijective such that $\{i, j\} \in E_1$ implies $\{\sigma(i), \sigma(j)\} \in E_2$. Then, for all $p_{ij} \in scope(\alpha_{G_1})$ with $\alpha_{G_1}(p_{ij}) = \{n_i, n_j\}$ we have that $\alpha_{G_2}(p_{\sigma(i),\sigma(j)}) = \{n_{\sigma(i)}, n_{\sigma(j)}\}$. Therefore, α_{G_1} dominates α_{G_2}.

Now, let us assume that α_{G_1} dominates α_{G_2}. Then, there exist functions $a : E_1 \to E_2$ and $b : V \to V$ such that for all $p_{ij} \in scope(\alpha_{G_1})$ with $\alpha_{G_1}(p_{ij}) = \{n_i, n_j\}$ we have that $\alpha_{G_2}(p_{a(\{i,j\})}) = \{n_{b(i)}, n_{b(j)}\}$. By construction of α_{G_2}, this is equivalent to $\{n_{b(i)}, n_{b(j)}\} \in E$ for all $\{i, j\} \in E$. Thus, b is a sub-isomorphism between G_1 and G_2. □

With Theorem 7, it is possible to prove the following corollary:

Corollary 1 *The dominance detection problem for piecewise interchangeable set-CSPs is NP-hard.*

Proof We reduce the problem to subgraph-isomorphism. In order to apply Theorem 7, we need to ensure that both graphs operate over the same set of nodes. When the sets of nodes of the given graphs differ, it is possible to see that G_1 cannot be sub-isomorphic to G_2 if G_1 contains more nodes than G_2. When G_1 actually contains fewer nodes than G_2, it is possible to see that we can add isolated nodes to G_1 without affecting subgraph-isomorphism. Then, we have that both graphs contain the same number of nodes, and, by relabelling the nodes in both graphs, we may assume that both graphs operate on the same set of nodes. □

Note that, despite this negative result, in some important special cases the dominance detection problem for piecewise interchangeable set-CSPs is still tractable. For example, when the set variables cannot take overlapping sets as values, the algorithm developed in Section 3 can be adapted (by exchanging the roles of values and variables) to break all the symmetries efficiently. Hence the following corollary of Theorem 1:

Corollary 2 *The dominance detection problem for piecewise interchangeable set-CSPs is tractable for non-overlapping sets.*

Note that the dominance detection problem as we consider it here regards arbitrary partial assignments. This implies that, when the detection problem is tractable, we can break symmetries efficiently. However, the situation changes when we achieve an intractability result like the previous one.

Within methods like SBDD, the partial assignments that need to be compared can only differ in a rather specific fashion. We can also show that these more specific dominance detection problems are NP-hard as well, therefore proving that SBDD in its general form is incapable of breaking symmetries in piecewise interchangeable set-CSPs efficiently. The specific dominance detection problems that SBDD considers differ from the general dominance detection problem by the fact that the partial assignments α and β that are compared are not arbitrary. We know that there exists exactly one assignment $v = d$ such that $\alpha = \gamma \ \& \ (v = d)$, while $\beta = \gamma \ \& \ \delta$, and $v \in Dom(\delta)$ for some partial assignments γ and δ.

We prove that dominance detection even for this limited problem is still NP-hard by using the same idea as before, but this time we only consider complete subgraphs, i.e., we reduce to the clique problem rather than to arbitrary subgraph isomorphism. Given a graph G and a value k, the first assignment is based on a complete graph of size k and it is defined in accordance with Definition 27. The second assignment is based on G with an additional, disconnected component that is a complete graph of size k with just one edge missing. With this setting, the first and second assignments have the same structural relationship as assignments that need to be compared within SBDD. Moreover, the given graph contains a clique of size k if and only if the first assignment dominates the second. Consequently, for piecewise interchangeable set-CSPs, SBDD is not capable of breaking symmetries efficiently.

As a final note on this negative result, we would like to stress that this does not imply that symmetry breaking is NP-hard in general since we do not consider other methods here like remodelling or the adaptation of the branching scheme.

7 Generalisations: wreath interchangeability

So far, we have focused on piecewise symmetry only. In this section, we generalise some of our tractability results to the more complex class of CSPs where each variable is assigned a pair of values (d_1, d_2) from a domain $D_1 \times D_2$. All values in D_1 are interchangeable and, for a fixed value in D_1, all values in D_2 are interchangeable as well. These problems are here called *wreath value-interchangeable* CSPs, because

the symmetry group corresponds to a wreath product of groups [4]. Such problems arise naturally in a variety of applications, e.g., in resource allocation and scheduling.

Example 9 Consider the problem of scheduling a meeting where different groups must meet some day of the week in some room, subject to constraints. The days are fully interchangeable and, on a given day, the rooms are fully interchangeable.

7.1 Wreath value-interchangeable CSPs

We now formally define the class of wreath value-interchangeable CSPs. Our definitions and results only consider *two* sets of *fully* interchangeable values, for simplicity. They can be generalised to an arbitrary *fixed* number of sets, and to sets of *piecewise* interchangeable values.

Definition 28 (Wreath Bijection) Let $S = S_1 \times S_2$ be a Cartesian product. A bijection $b : S \to S$ is a *wreath bijection* over $S_1 \times S_2$ if $b(\langle e_1, e_2\rangle) = \langle b_1(e_1), b_2^{e_1}(e_2)\rangle$, where $b_1 : S_1 \to S_1$ is a bijection and each $b_2^{e_1} : S_2 \to S_2$ (for $e_1 \in S_1$) is a bijection.

Definition 29 (Wreath Value-Interchangeable CSP) A CSP $\mathcal{P} = \langle V, D_1 \times D_2, C\rangle$ is a *wreath value-interchangeable CSP* if, for each solution $\sigma \in Sol(\mathcal{P})$ and each wreath bijection b over $D_1 \times D_2$, we have $b \circ \sigma \in Sol(\mathcal{P})$.

Thus, in a wreath value-interchangeable CSP, a value in the domain $D_1 \times D_2$ is assigned to each variable, where the values in D_1 are fully interchangeable, and, for a fixed value in D_1, the values in D_2 are fully interchangeable as well.

We now propose a highly efficient symmetry breaking algorithm for wreath value-interchangeable CSPs.

We use the following notations. If $d = (d_1, d_2)$ is a pair, then $d[1] = d_1$ and $d[2] = d_2$. If T is a set of tuples, then $T[i]$ denotes the set $\{d[i] \mid d \in T\}$ and $filter(T, i, d_i)$ denotes the set $\{d \mid d \in T \ \&\ d[i] = d_i\}$. If $\alpha : D_1 \times D_2 \to D_1 \times D_2$ is an assignment, then $\alpha^{-1}(d_1, D_2)$ denotes the set $\{\alpha^{-1}(d_1, d_2) \mid d_2 \in D_2\}$.

Definition 30 (Closure of a Nogood) Let α be a nogood for a wreath value-interchangeable CSP $\mathcal{P} = \langle V, D_1 \times D_2, C\rangle$. The *closure* of α for $\mathcal{P}$, denoted by $Closure(\alpha, \mathcal{P})$, is the set $\{b \circ \alpha \mid b$ is a wreath bijection over $D_1 \times D_2\}$.

We now define the relevant abstract nogoods.

Definition 31 (Abstract Nogood) Let α be a nogood for a wreath value-interchangeable CSP $\mathcal{P} = \langle V, D_1 \times D_2, C\rangle$. Let $image(\alpha)[1] = \{d_1, \ldots, d_k\}$, let $filter(image(\alpha), 1, d_i) = \{d_1^i, \ldots, d_{\ell_i}^i\}$, let $v_{r_i} \in \alpha^{-1}(d_i, D_2)$, for $1 \le i \le k$, and let $v_{r_j^i} \in \alpha^{-1}(d_i, d_j)$, for $1 \le i \le k$ and $1 \le j \le \ell_i$. The *abstract nogood* of α with respect to $\mathcal{P}$, denoted by $Anogood(\alpha, \mathcal{P})$, is the set of all functions $\gamma : scope(\alpha) \to D_1 \times D_2$

satisfying the condition

$$\begin{aligned}
\forall i \in 1 \ldots k: \ & \mathit{allequal}(\gamma(v_j)[1] \mid v_j \in \alpha^{-1}(d_i, D_2)) \ \ \& \\
& \mathit{alldiff}(\gamma(v_{r_1})[1], \ldots, \gamma(v_{r_k})[1]) \ \ \& \\
\forall i \in 1 \ldots \ell_1: \ & \mathit{allequal}(\gamma(v_j)[2] \mid v_j \in \alpha^{-1}(d_1, d_i^1)) \ \ \& \\
& \mathit{alldiff}(\gamma(v_{r_1^1})[1], \ldots, \gamma(v_{r_{\ell_1}^1})[1]) \ \ \& \\
\ldots & \\
\forall i \in 1 \ldots \ell_k: \ & \mathit{allequal}(\gamma(v_j)[2] \mid v_j \in \alpha^{-1}(d_1, d_i^k)) \ \ \& \\
& \mathit{alldiff}(\gamma(v_{r_1^k})[1], \ldots, \gamma(v_{r_{\ell_k}^k})[1])
\end{aligned}$$

Figure 6 depicts the labelling procedure `wValIlabel` for wreath value-interchangeable CSPs. Its correctness proof is similar to the one of Theorem 4.

Theorem 8 *Procedure* `wValIlabel` *eliminates all symmetric subtrees caused by the value symmetries of a wreath value-interchangeable CSP with a constant overhead with respect to both time and space at every node explored.*

7.2 Wreath value-interchangeable set-CSPs

Symmetry breaking for wreath value-interchangeable *set*-CSPs is also tractable.

Definition 32 (Wreath Set Bijection) Let $S = S_1 \times S_2$ be a Cartesian product. A bijection $b : 2^S \to 2^S$ is a *wreath set bijection* over $2^{S_1 \times S_2}$ if b is induced by a wreath bijection over $S_1 \times S_2$.

Fig. 6 A labelling procedure for wreath value-interchangeable CSPs

```
bool wValIlabel(P) {
    return wValIlabelA(P, ε);
}
bool wValIlabelA(⟨V, D1 × D2, C⟩, θ) {
    if scope(θ) = V then
        return C(θ);
    select v in V \ scope(θ);
    A1 := image(θ)[1];
    if A1 ≠ D1 then
        select f in D1 \ A1; A1 := A1 ∪ {f};
    forall(d1 ∈ A1)
        A2 := filter(image(α), 1, d1)[2];
        if A2 ≠ D2 then
            select f in D2 \ A2; A2 := A2 ∪ {f};
        forall(d2 ∈ A2)
            θ' := θ & v = (d1, d2);
            if ¬Failure(⟨V, D1 × D2, C⟩, θ') then
                if wValIlabelA(⟨V, D1 × D2, C⟩, θ') then
                    return true;
    return false;
}
```

Definition 33 (Wreath Value-Interchangeable Set-CSP) A set-CSP $\mathcal{P} = \langle V, 2^{D_1 \times D_2}, C\rangle$ is a *wreath value-interchangeable set-CSP* if, for each solution $\sigma \in Sol(\mathcal{P})$ and each wreath set bijection b over $2^{D_1 \times D_2}$, we have $b \circ \sigma \in Sol(\mathcal{P})$.

Consider the following example.

Example 10 Take the set $V = \{v_1, v_2\}$ of set variables over the universe $D_1 \times D_2$, with $D_1 = \{d_1, d_2\}$ and $D_2 = \{e_1, e_2, e_3\}$, such that the set-CSP $\mathcal{P} = \langle V, 2^{D_1 \times D_2}, C\rangle$ is wreath value-interchangeable, the constraint set C being arbitrary. Suppose that we have already tried the partial assignment

$$\alpha_1 = \big(v_1 = \big\{(d_1, e_1), (d_1, e_2), (d_2, e_2), (d_2, e_3)\big\} \;\&\; v_2 = \big\{(d_2, e_1), (d_2, e_2)\big\}\big)$$

and that now we are about to investigate the partial assignment

$$\alpha_2 = \big(v_1 = \big\{(d_1, e_1), (d_1, e_2), (d_2, e_1), (d_2, e_2)\big\} \;\&\; v_2 = \big\{(d_1, e_2), (d_1, e_3)\big\}\big).$$

How can we decide whether α_2 is a symmetric variant of α_1 or not? One way to do that is to construct a permutation of D_1, as well as corresponding permutations of D_2, so that the sets in α_2 are transformed into those of α_1.

In order to construct a permutation σ of D_1, let us assess whether d_1 can be mapped to itself. If $\sigma(d_1) = d_1$, then $v_2 = \{(d_1, e_2), (d_1, e_3)\}$ in α_2 cannot be mapped to $v_2 = \{(d_2, e_1), (d_2, e_2)\}$ in α_1, no matter how we permute D_2. Algorithmically, we can infer this by checking whether the number of tuples starting with d_1 in α_2 is the same as the number of tuples starting with $\sigma(d_1)$ in α_1 for all assigned set variables. For v_1, the important tuples in α_2 are (d_1, e_1) and (d_1, e_2). That means that there are two such tuples, which matches the number of respective tuples for v_1 in α_1, namely (d_1, e_1) and (d_1, e_2). For v_2, the respective tuples in α_2 are (d_1, e_2) and (d_1, e_3), i.e., there are two such tuples. In α_1, on the other hand, there are no tuples starting with $\sigma(d_1) = d_1$ at all, which shows that d_1 cannot be mapped to itself.

Now let us investigate whether d_1 can be mapped to d_2. First, we check whether the numbers of tuples match. For v_1 we have two tuples starting with d_1 in α_2, and in α_1 we have two tuples starting with d_2. Moreover, for v_2 we have two tuples starting with d_1 in α_2, and also two tuples starting with d_2 in α_1. Therefore, the initial check on setting $\sigma(d_1) = d_2$ is inconclusive. To check fully whether we can construct a permutation σ of D_1 with $\sigma(d_1) = d_2$, we need to find out whether there exists a permutation of D_2 such that the respective tuple sets map exactly, and not just in number. That is, we need to construct a permutation τ of D_2 such that, with $\sigma(d_1) = d_2$, we have

$$\big\{\big(\sigma(d_1), \tau(e_1)\big), \big(\sigma(d_1), \tau(e_2)\big)\big\} = \big\{\big(d_2, e_2\big), \big(d_2, e_3\big)\big\} \tag{3}$$

and

$$\big\{\big(\sigma(d_1), \tau(e_2)\big), \big(\sigma(d_1), \tau(e_3)\big)\big\} = \big\{\big(d_2, e_1\big), \big(d_2, e_2\big)\big\}, \tag{4}$$

or we need to show that no such permutation exists. The equations above pose the following constraints on the permutation τ that we are trying to construct: $\tau(e_1) \in \{e_2, e_3\}$, $\tau(e_2) \in \{e_2, e_3\} \cap \{e_1, e_2\}$, and $\tau(e_3) \in \{e_1, e_2\}$.

Fortunately, constructing τ or proving that no such permutation exists can be done by solving a maximum matching problem in a bipartite graph. The node set N is

defined as the union of the sets $N_1 := \{e_1, e_2, e_3\}$ and $N_2 := \{e'_1, e'_2, e'_3\}$, where the e'_i are copies of the e_i. We define the edge set E in accordance with the constraints as given before, i.e., we add an edge $(e_i, e'_j) \in N_1 \times N_2$ to E if and only if $\tau(e_i) = e_j$ is allowed. Then, a perfect matching in $G = (N, E)$ exists if and only if there exists a permutation τ that satisfies equations (3) and (4). As we can see in Fig. 7a, a maximum matching, and consequently a permutation τ, exists that shows that we can potentially set $\sigma(d_1) = d_2$.

We continue to check whether setting $\sigma(d_2) = d_1$ and $\sigma(d_2) = d_2$ are possible. We find that for both these mappings we can construct a corresponding legal permutation τ of D_2.

Now, equipped with that knowledge, we can try at last to construct σ where we must ensure that $\sigma(d_1) \in \{d_2\}$ and $\sigma(d_2) \in \{d_1, d_2\}$. Following the same idea as before, we check whether such a permutation exists by solving a maximum matching problem in a bipartite graph: see Fig. 7b. Since a perfect matching exists, we have a proof that indeed assignment α_2 is symmetric to α_1. On the other hand, the construction of σ could only have failed if no permutation of D_1 and corresponding permutations of D_2 existed.

Generally, we state:

Theorem 9 *All symmetric subtrees caused by the value symmetries of a wreath value-interchangeable set-CSP can be eliminated with a polynomial time overhead at every node explored.*

Proof As in all pure cases of value symmetry, we only need to check search nodes against their previously expanded siblings. We show how this dominance check can be performed by abstracting from the concrete example above. For all potential mappings $\sigma(d) = e$, and for all set variables v_i that were assigned values in α_1, we first check whether the number of tuples in the set $\alpha_1(v_i)$ starting with e matches the number of tuples in the set $\alpha_2(v_i)$ starting with d. If that is not the case, we note that setting $\sigma(d) = e$ is not feasible. Otherwise, we set up a bipartite graph $G_{d,e} = (N_{d,e}, E_{d,e})$ where $N_{d,e}$ consists of all possible second tuple entries f and their copies f'. An edge (f, g') is an element of $E_{d,e}$ if and only if, for all set variables v_i that were assigned values in α_1, either $(d, f) \notin \alpha_2(v_i)$ or $(d, f) \in \alpha_2(v_i)$ & $(e, g) \in \alpha_1(v_i)$. We note $\sigma(d) = e$ as feasible if and only if there exists a perfect matching in $G_{d,e}$. Finally, we set up a bipartite graph $G = (N, E)$ where N consists of all possible first tuple entries d and their copies d'. An edge (d, e') is an element of E if and only if

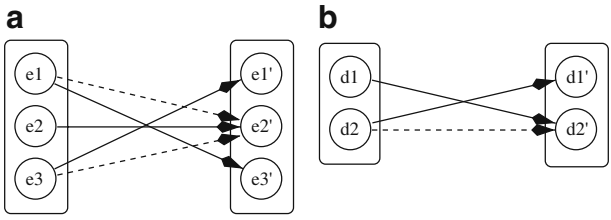

Fig. 7 Part **a** gives the bipartite graph constructed to assess whether d_1 can be mapped to d_2. Part **b** shows the bipartite graph constructed to find a feasible permutation of D_1 or to show that none exists

$\sigma(d) = e$ is feasible. Then, we report α_2 as dominated by α_1 if and only if there exists a perfect matching in G.

This method either constructs permutations that prove the dominance of α_1 or shows that no such permutation exists. When p denotes the number of possible first tuple entries and q the number of possible second tuple entries, our algorithm can be implemented to run in $O(p^2q^{2.5})$ time. □

Note that the dominance checker that we outlined in the proof above can be generalised for tuples with k entries. However, the run-time is then exponential in k. We leave open whether an efficient labelling algorithm can be formulated to break this type of value symmetry. The point here was to show that wreath value symmetry allows tractable symmetry breaking for set-CSPs.

Also note that this tractability results subsumes the special case discussed in Section 5.3, where a very efficient symmetry-breaking method was given for fully value-interchangeable set-CSPs.

7.3 The grapes of wreath:[5] wreath variable-interchangeable (Set-)CSPs

The previous ideas transfer to wreath *variable*-interchangeable CSPs, which were not considered in [27, 33]. A variable set $V = V_1 \times V_2$ is a two-dimensional matrix of variables, with index sets V_1 for the rows and V_2 for the columns. Wreath variable interchangeability in such a matrix means that the rows are piecewise interchangeable and, for a given row index in V_1, the row variables with column indices in V_2 are piecewise interchangeable as well. Such problems also arise naturally in a variety of applications.

Example 11 Consider Steiner triple systems, where there is a set B of 3-element subsets, called *triples*, of a set X of $v \geq 3$ elements, such that every pair of distinct elements of X appears in exactly one triple of B. If we use a $\frac{v(v-1)}{6}$-by-3 matrix of scalar variables in X to represent the triples, then the triples (rows) are fully interchangeable and, in a given triple, the elements are fully interchangeable. (Note that the values in X are also fully interchangeable.)

Definition 34 (Wreath Variable-Interchangeable CSP) A CSP $\mathcal{P} = \langle V_1 \times V_2, D, C\rangle$ is a *wreath variable-interchangeable CSP* if, for each solution $\alpha \in Sol(\mathcal{P})$ and each wreath bijection σ over $V_1 \times V_2$, we have $\alpha \circ \sigma \in Sol(\mathcal{P})$.

Let us start from just wreath variable interchangeability, and it then does not matter whether it is a scalar CSP or a set-CSP.

Theorem 10 *The dominance detection problem for wreath variable-interchangeable (set-)CSPs is tractable.*

[5] With apologies to John Steinbeck, author of *The Grapes of Wrath*, 1939.

Proof A wreath variable-interchangeable CSP $\mathcal{P} = \langle V_1 \times V_2, D, C\rangle$ can be reformulated as the piecewise variable-interchangeable set-CSP

$$\mathcal{P}' = \left\langle V_1, 2^D, C' \cup \{card(i) = card(V_2) \mid i \in V_1\}\right\rangle$$

whose set variables are all constrained to be $|V_2|$-element sub-multisets of D, and whose constraints C' are a reformulation of C for multiset variables. Indeed, the order of the elements in a (multi)set is irrelevant, and this effectively models the fact that for a given row index $i \in V_1$, the variables $(V_1 \times V_2)[i, j]$ are piecewise interchangeable for all the column indices $j \in V_2$. Since it is already known that all symmetric subtrees caused by the variable symmetries of a piecewise interchangeable CSP can be eliminated with a polynomial-time overhead at every node explored (see Theorem 1), it is also so for piecewise variable-interchangeable set-CSPs, where the piecewise value interchangeability is dropped (with no negative impact on the tractability) and where set variables replace the scalar variables. Indeed, a set variable that takes a subset of a universe U of *non*-interchangeable values can be seen as a collection of scalar variables that take (scalar) values from U. (In other words, set variables only complicate matters when the universe has interchangeable values.) The stated result thus follows from the reformulation. The argument goes similarly for wreath variable-interchangeable set-CSPs. □

Even adding only full value interchangeability, we unfortunately lose the tractability, at least for dynamic symmetry-breaking approaches based on dominance detection. The following is our first intractability result for a class of *scalar* CSPs.

Theorem 11 *The dominance detection problem for fully value-interchangeable and wreath variable-interchangeable CSPs is NP-hard. Hence the dominance detection problem is NP-hard also for fully value-interchangeable and wreath variable-interchangeable set-CSPs, for piecewise value-interchangeable and wreath variable-interchangeable (set-)CSPs, and for wreath value-interchangeable and wreath variable-interchangeable (set-)CSPs.*

Proof Consider a fully value-interchangeable and wreath variable-interchangeable CSP. Temporarily dropping the full value interchangeability, we get a wreath variable-interchangeable CSP, which we can reformulate as a fully variable-interchangeable set-CSP, like at the beginning of the proof of Theorem 10, without losing the tractability of dominance detection. Bringing back the full value interchangeability, we get a fully interchangeable set-CSP. Since the dominance detection problem for fully interchangeable set-CSPs is NP-hard (see Corollary 1), the stated results for scalar CSPs follow. The stated results on set-CSPs follow from the intractability on scalar CSPs because scalar variables can replace set variables that are all constrained to take singleton values. □

Example 12 The matrix model of Example 11 for Steiner triple systems is a fully value-interchangeable and wreath variable-interchangeable CSP. Hence the dominance detection problem is NP-hard for this model. Note that there are $\frac{v(v-1)}{6}!3!^v$ symmetries in it, that is already $1,410,877,440$ symmetries for the 7-by-3 variable matrix for $v = 7$.

Table 1 Tractability of symmetry breaking and dominance detection

	Variable interchangeability				
	None	Full	Piecewise	Wreath	
Value interchangeability					
None		P (Thm 1)	P (Thm 1)	P (Thm 10)	scalar CSP
		P (Thm 1)	P (Thm 1)	P (Thm 10)	set-CSP
Full	P (Thm 4)	P (Thm 1)	P (Thm 1)	NP (Thm 11)	scalar CSP
	P (Thm 6)	NP (Cor. 1)	NP (Cor. 1)	NP (Thm 11)	set-CSP
Piecewise	P (Thm 5)	P (Thm 1)	P (Thm 1)	NP (Thm 11)	scalar CSP
	P (Thm 9)	NP (Cor. 1)	NP (Cor. 1)	NP (Thm 11)	set-CSP
Wreath	P (Thm 8)	P [13, Thm 4]	P [13, Thm 4]	NP (Thm 11)	scalar CSP
	P(Thm 9)	NP [13, Cor. 1]	NP [13, Cor. 1]	NP (Thm 11)	set-CSP

8 Conclusion

We have theoretically studied several classes of CSPs for which symmetry breaking is tractable, in the sense that all symmetric subtrees caused by the symmetries of CSPs in those classes can be eliminated with a polynomial time overhead at every node explored. These CSP classes, which encompass many practical problems, feature various forms of value or variable interchangeability and allow symmetry breaking to be performed with a *polynomial* overhead (which is often even a *constant* overhead) with respect to both time and space at every node explored, using dedicated search procedures. Unfortunately, efficient symmetry breaking by such dominance-detection schemes has its limits, as we have identified some CSP classes where dominance detection is intractable.

Table 1 summarises our main results, where "P (Thm *i*)" means that breaking all the symmetries mentioned in the corresponding row is feasible with a polynomial overhead with respect to both time and space at every node explored for the corresponding (set-)CSP in the column, as proved in Theorem *i* or a trivial consequence thereof. However, no specialised labelling procedures are given for these particular CSP classes in this paper. The negative tractability results, marked "NP (Thm/Cor. *i*)" and referring to Theorem/Corollary *i*, only concern the NP-hardness of dominance-detection schemes like SBDD; it remains an open research issue whether other schemes can break those symmetries in polynomial time.

In [26] it is proved that all value symmetries of a CSP are polynomial-time tractable; this is proved using group theoretic notions and although the resulting complexities are the order of a low-degree polynomial they are in general not as efficient as the specialised algorithms presented in this paper. A key component in the proofs is the notion of a minimal GE-tree, which is essentially the search tree that results from a search procedure that eliminates all symmetric subtrees. All the search procedures in this paper produce GE-trees.

In [12] we have provided a static counterpart of the here considered dynamic structural symmetry breaking for piecewise interchangeable CSPs, that is we have exploited the concept of signature to devise a set of symmetry-breaking constraints that break all the considered symmetries. Other methods for static symmetry breaking are discussed in [7, 11, 20, 21, 23, 25, 29], for instance.

There are many directions for future research. Of particular interest is the study of tractable classes of CSPs exhibiting variable symmetries where the variable set has a more complex structure than the partitions studied in this paper. In particular, when the variable set is obtained by a Cartesian product over some index sets, we get what is known as a *matrix model*. There are many interesting forms of interchangeability in matrix models, such as the full/piecewise/wreath interchangeability of matrix slices (rows, columns, ...) [11]. For many of these forms of variable interchangeability, including their compositions with various forms of value interchangeability, tractability results for symmetry breaking are still missing and finding effective search procedures is a challenging problem. Also, as Corollary 2 has shown, negative tractability results call for the identification of special cases where symmetry breaking is tractable.

Acknowledgements All authors were partly supported by institutional grant IG2001-67 of STINT, the Swedish Foundation for International Cooperation in Research and Higher Education. The Sweden-based authors were also supported by grants 221-99-369 and 70644501 of VR, the Swedish Research Council, during part of this work. Pierre Flener did some of this work while a Visiting Faculty Member in 2006/07 at Sabancı University in İstanbul, Turkey. Meinolf Sellmann is supported by the National Science Foundation through the Career: Cornflower Project (NSF award number 0644113). Pascal Van Hentenryck was partly supported by NSF ITR Award DMI-0121495. Many thanks to the anonymous reviewers, for their comments on this paper.

References

1. Ahuja, R., Magnati, T., & Orlin, J. (1993). *Network flows*. Englewood Cliffs: Prentice Hall.
2. Backofen, R., & Will, S. (1999). Excluding symmetries in constraint-based search. In J. Jaffar (Ed.), *Proceedings of CP'99*, *LNCS* (Vol. 1713, pp. 73–87). New York: Springer.
3. Barnier, N., & Brisset, P. (2002). Solving the Kirkman's schoolgirl problem in a few seconds. In P. Van Hentenryck (Ed.), *Proceedings of CP'02*, *LNCS* (Vol. 2470. pp. 477–491). New York: Springer.
4. Cameron, P. (1999). *Permutation groups*. Number 45 in London Mathematical Society Student Texts. Cambridge: Cambridge University Press.
5. Cohen, D. A., Jeavons, P., Jefferson, C., Petrie, K. E., & Smith, B. M. (2005). Symmetry definitions for constraint satisfaction problems. In P. van Beek (Ed.) *Proceedings of CP'05*, *LNCS* (Vol. 3709, pp. 17–31). New York: Springer.
6. Colbourn, C. J., & Dinitz, J. H. (Eds.) (1996). *The CRC handbook of combinatorial designs*. Boca Raton: CRC.
7. Crawford, J. M., Ginsberg, M., Luks, E., & Roy, A. (1996). Symmetry-breaking predicates for search problems. In L. C. Aiello, J. Doyle, & S. C. Shapiro (Eds.), *Proceedings of KR'96* (pp. 148–159). San Francisco: Morgan Kaufmann.
8. Er, M. C. (1988). A fast algorithm for generating set partitions. *The Computer Journal, 31*(3), 283–284.
9. Fahle, T., Schamberger, S., & Sellmann, M. (2001). Symmetry breaking. In T. Walsh (Ed.), *Proceedings of CP'01*, *LNCS* (Vol. 2239, pp. 93–107). New York: Springer.
10. Flener, P., Frisch, A. M., Hnich, B., Kızıltan, Z., Miguel, I., Pearson, J., et al. (2001). Symmetry in matrix models. In P. Flener & J. Pearson (Eds.), *Proceedings of SymCon'01*. http://www.it.uu.se/research/group/astra/SymCon01/.
11. Flener, P., Frisch, A. M., Hnich, B., Kızıltan, Z., Miguel, I., Pearson, J., et al. (2002). Breaking row and column symmetries in matrix models. In P. Van Hentenryck (Ed.), *Proceedings of CP'02*, *LNCS* (Vol. 2470, pp. 462–476). New York: Springer.
12. Flener, P., Pearson, J., Sellmann, M., & Van Hentenryck, P. (2006). Static and dynamic structural symmetry breaking. In F. Benhamou (Ed.), *Proceedings of CP'06*, *LNCS* (Vol. 4204, pp. 695–699). New York: Springer.

13. Flener, P., Pearson, J., & Sellmann, M. (2008). Static and dynamic structural symmetry breaking. Technical Report 2008-023, Department of Information Technology, Uppsala University, Sweden, September. http://www.it.uu.se/research/reports/2008-023/.
14. Flener, P., Pearson, J., Sellmann, M., & Ågren, M. (2007). Structural symmetry breaking for constraint satisfaction problems. Technical Report 2007-032, Department of Information Technology, Uppsala University, Sweden, November. http://www.it.uu.se/research/reports/2007-032/.
15. Focacci, F., & Milano, M. (2001). Global cut framework for removing symmetries. In T. Walsh (Ed.), *Proceedings of CP'01*, *LNCS* (Vol. 2239, pp. 77–92). New York: Springer.
16. Freuder, E. C. (1991). Eliminating interchangeable values in constraint satisfaction problems. In *Proceedings of AAAI'91* (pp. 227–233). Menlo Park: AAAI.
17. Gent, I. P., & Smith, B. M. (2000). Symmetry breaking during search in constraint programming. In *Proceedings of ECAI'00* (pp. 599–603). Amsterdam: IOS.
18. Heller, D. S., & Sellmann, M. (2006). Dynamic symmetry breaking restarted. In F. Benhamou (Ed.), *Proceedings of CP'06*, *LNCS* (Vol. 4204, pp. 721–725). New York: Springer.
19. Kubale, M., & Jackowski, B. (1985). A generalized implicit enumeration algorithm for graph coloring. *CACM, 28*(4), 412–418.
20. Law, Y., & Lee, J. (2006). Symmetry breaking constraints for value symmetries in constraint satisfaction. *Constraints, 11*(2–3), 221–267.
21. Law, Y., Lee, J., Walsh, T., & Yip, J. (2007). Breaking symmetry of interchangeable variables and values. In C. Bessière (Ed.), *Proceedings of CP'07*, *LNCS* (Vol. 4741, pp. 423–437). New York: Springer.
22. Meseguer, P., & Torras, C. (2001). Exploiting symmetries within constraint satisfaction search. *Artificial Intelligence, 129*(1–2), 133–163.
23. Puget, J.-F. (1993). On the satisfiability of symmetrical constrained satisfaction problems. In J. Komorowski & Z. Raś (Ed.), *Proceedings of ISMIS'93*, *LNAI* (Vol. 689, pp. 350–361). New York: Springer.
24. Puget, J.-F. (2002). Symmetry breaking revisited. In P. Van Hentenryck (Ed.), *Proceedings of CP'02*, *LNCS* (Vol. 2470, pp. 446–461). New York: Springer.
25. Puget, J.-F. (2006). An efficient way of breaking value symmetries. In *Proceedings of AAAI'06*. Menlo Park: AAAI.
26. Roney-Dougal, C. M., Gent, I. P., Kelsey, T., & Linton, S. (2004). Tractable symmetry breaking using restricted search trees. In R. L. de Mántaras & L. Saitta (Eds.), *Proceedings of ECAI'04* (pp. 211–215). Amsterdam: IOS.
27. Sellmann, M., & Van Hentenryck, P. (2005). Structural symmetry breaking. In *Proceedings of IJCAI'05* (pp. 298–303). IJCAI.
28. Sellmann, M., Gellermann, T., & Wright, R. (2007). Cost-based filtering for shorter path constraints. *Constraints, 12*(2), 207–238.
29. Shlyakhter, I. (2001). Generating effective symmetry-breaking predicates for search problems. *Electronic Notes in Discrete Mathematics* (Vol. 9). Proceedings of SAT'01.
30. Smith, B. M. (2001). Reducing symmetry in a combinatorial design problem. In C. Gervet & M. Wallace (Eds.), *Proceedings of CP-AI-OR'01*.
31. Smith, B. M., Brailsford, S. C., Hubbard, P. M., & Williams, H. P. (1996). The progressive party problem: Integer linear programming and constraint programming compared. *Constraints, 1*, 119–138.
32. Van Hentenryck, P. (2002). Constraint and integer programming in OPL. *INFORMS Journal on Computing, 14*(4), 345–372.
33. Van Hentenryck, P., Flener, P., Pearson, J., & Ågren, M. (2003). Tractable symmetry breaking for CSPs with interchangeable values. In *Proceedings of IJCAI'03* (pp. 277–282). San Francisco: Morgan Kaufmann.
34. Van Hentenryck, P., Flener, P., Pearson, J., & Ågren, M. (2005). Compositional derivation of symmetries for constraint satisfaction. In J.-D. Zucker & L. Saitta (Eds.), *Proceedings of SARA'05*, *LNAI* (Vol. 3607, pp. 234–247). New York: Springer.
35. Walsh, T. (2007). Breaking value symmetry. In C. Bessière (Ed.), *Proceedings of CP'07*, *LNCS* (Vol. 4741, pp. 880–887). New York: Springer.