Excluding Symmetries in Constraint-Based Search

Rolf Backofen and Sebastian Will*
({backofen,wills}@informatik.uni-muenchen.de)
Institut für Informatik, LMU München
Oettingenstraße 67, D-80538 München

Abstract. We introduce a new method, called *symmetry excluding search (SES)*, for excluding symmetries in constraint based search. To our knowledge, it is the first declarative method that can be applied to *arbitrary* symmetries. The SES-method is based on the notion of symmetric constraints, which are used in our modification of a general constraint based search algorithm. The method does not influence the search strategy. Furthermore, it can be used with either the full set of symmetries, or a subset of all symmetries.

We proof correctness, completeness and symmetry exclusion properties of our method. Then, we show how to apply the SES-method in the special case of geometric symmetries (rotations and reflections) and permutation symmetries. Furthermore, we give results from practical applications.

Keywords: Symmetry Exclusion, Geometric Symmetries, Permutation Symmetries, Constraint-Based Search, Symmetry Excluding Search (SES)

1. Introduction

In many search problems, one is faced with the existence of symmetries. Symmetries give rise to many different solutions found by the search procedure, which are all considered to be *equivalent*. Often, one is not interested in also getting all these *symmetric* solutions to every found solution. Without exclusion of symmetries, whenever a solution is found or proven to be inconsistent with the search problem, the search algorithm still considers all symmetric solutions. In consequence, those symmetries give rise to an (often exponential) amplification of the search space. Hence, symmetry exclusion promises to efficiently prune the search tree.

For example, consider constraint problems, where finite domain variables have a geometric interpretation such as in the N-queens problem or the square-tiling problem (where a set of squares of given sizes must fit exactly into a fixed square). A more complex, real-world problem is the lattice protein structure prediction problem. In [2], it is shown how to find solutions for this problem using constraint programming techniques. In this case, different solutions are symmetric if they can be generated using reflections or rotations. In other problems, symmetric

* supported by the PhD programme GKLI of Deutsche Forschungsgemeinschaft.

solutions can be generated by performing permutations on variable valuations (like in the map coloring (or graph coloring) problem).

In the following, we consider search problems that stem from constraint satisfaction problems (CSP). A common approach is to transform (problem specifically) a CSP C_1 into a CSP C_2 that excludes at least some of the symmetries of the original problem C_1.

Unfortunately, symmetry exclusion was often not straightforward to implement. Even then, it had to be redesigned for every special problem or enumeration strategy. This led to an inflexibility in the program structure once it was introduced. Thus, the widespread usage of symmetry exclusion was strongly hindered by its complexity. Often symmetry exclusion was not done at all or only for a small set of symmetries. In other cases, where symmetry exclusion was implemented it distracted the programmers attention from more important tasks.

In this paper, we present a new approach for symmetry exclusion, called *symmetry excluding search (SES)*, which works by modifying the search algorithm. The technique of modifying the search algorithm to exclude symmetries was already used in the literature. In contrast to previous approaches, SES is (to our knowledge) the first declarative method that can be applied to *arbitrary* symmetries, which can be defined declaratively. Furthermore, it does not restrict the search strategy (i.e., the symmetry exclusion can be done independently of the search strategy). This is important, since in many constraint problems major part of the knowledge of the problem is encoded in the search strategy.

We have implemented our method in the concurrent constraint programming language Oz [10] using the programmable search engine described in [9]. However, the method can be implemented in any system that handles implication (via entailment) and allows to modify the search procedure such that additional constraints are added in one branch of a branching step.

Related Work. Previous work on symmetry exclusion by modifying the search algorithm handled only restricted forms of symmetries. In [7], the notion of interchangeable values is introduced. Two values a and b are interchangeable for a variable V iff for every solution that maps V to a, the corresponding valuation that maps V to b is also a solution.

The method in [3] handles only permutations of domain values in the case of binary constraints (thus, e.g. the all-distinct constraint has to be translated into binary inequalities, which implies that efficient propagation methods for all-distinct cannot be applied). Furthermore, it works by introducing a symmetry excluding form of constraint propagation, which is a modified form of domain propagation (thus, the method cannot make use of interval propagation).

[1] and [4] consider only propositional calculus. The symmetries there are permutations of Boolean variables. These methods do not apply to general constraint satisfaction problems.

The work presented in [8] is essentially an implementation and application of our method to solve practical problems. Their work is compared to the SES-method in greater detail in Section 2.

An example of a symmetry exclusion method that works by a (problem independent) transformation of the constraint satisfaction problem is [5]. They introduce symmetry breaking predicates, which are true only for the smallest solution within an equivalence class (where the ordering is fixed in advance). Thus, if the search strategy enumerates a non-minimal (according to the pre-fixed ordering) solution first, then it will be excluded by the symmetry breaking predicates. In consequence, not every search strategy remains possible given the pre-fixed ordering.

In contrast, there is no prefixed ordering in which symmetrical solutions will be excluded by our method. This implies that we respect user-defined search strategies as much as possible.[1]

Overview of the paper. The formal definition of symmetry excluding search trees as wells as completeness and correctness proofs are presented in Section 2. Additionally, we show that our method guarantees to exclude all specified symmetries. Section 3 discusses an optimized implementation of symmetry excluding search. In Section 4, we treat geometric symmetries and present the results of applying the problem to a complex problem. In Section 5, we consider symmetries which are generated by permuting the values of the variables. For example, such symmetries occur in the graph coloring problem. We proof, that in this case, it is sufficient to exclude the subset of transpositions in order to exclude all permutations and present results for graph coloring.

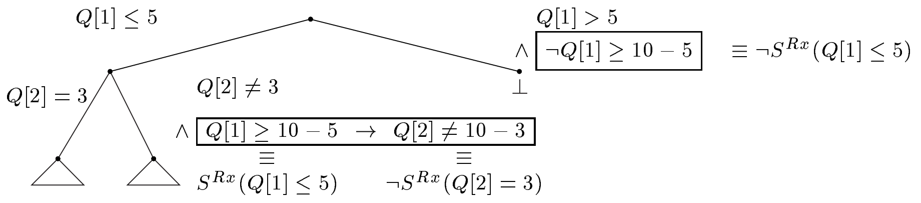
Introductory Example. We start with a simple example to explain the main concepts. Consider the N-queens problem, where we have an array $Q[0..N]$, whose elements can take values from $0..N$. $Q[i] = j$ states that there is a queen at position (i, j). Now consider the symmetry corresponding to the reflection at the axis $y = \frac{N}{2}$, which is parallel to the x-axis (in the sequel denoted by Rx). Clearly, for every solution of the N-queens, the Rx-symmetric version is also a solution of N-queens and thus should be excluded.

To use our method, first we have to introduce symmetric versions of the constraints which are inserted by the search procedure. One kind of constraint is $Q[i] = j$. If we have any distribution of queens on

[1] Clearly, if the search strategy depends on the constraint store, then any method of symmetry exclusion that prunes the search tree must influence the way solutions are encountered (since the constraint store is changed by the symmetry exclusion).

the board satisfying $Q[i] = j$, then the Rx-symmetric distribution will satisfy $Q[i] = N - j$. Hence, the constraint $Q[i] = N - j$ is the Rx-symmetric version of the constraint $Q[i] = j$. Similarly, we get that the Rx-symmetric constraint of $Q[i] \leq j$ is $Q[i] \geq N - j$. In the following, we write $S^{Rx}(c)$ to denote the Rx-symmetric constraint to c.

Our method works by inserting additional constraints at the right branches of the search tree, which exclude the symmetric solutions that are enumerated in the right branch. Consider the following search tree for the 11-queens problem, where we indicate the constraints added by symmetry exclusion via frames:


$Q[1] \leq 5$
$Q[1] > 5$
$\wedge\ \boxed{\neg Q[1] \geq 10 - 5}$
$\equiv \neg S^{Rx}(Q[1] \leq 5)$
$\perp$
$Q[2] = 3$
$Q[2] \neq 3$
$\wedge\ \boxed{Q[1] \geq 10 - 5 \ \rightarrow\ Q[2] \neq 10 - 3}$
$\equiv$
$\equiv$
$S^{Rx}(Q[1] \leq 5)$
$\neg S^{Rx}(Q[2] = 3)$

This can be interpreted intuitively as follows. In the topmost branch, if we search for a solution in the right branch satisfying $Q[1] > 5$, then the Rx-symmetric solution will satisfy $Q[1] < 5$. Hence, the symmetric solution was already encountered earlier in the left branch (under $Q[1] \leq 5$), which is the reason that we can close the topmost right branch. For the second right branch labeled $Q[2] \neq 3$, we want again to exclude the Rx-symmetric solutions found in the right branch. Hence, we would like to add the constraint $\neg S^{Rx}(Q[2] = 3)$ to the right branch. But this would exclude too many solutions. The prerequisite of the Rx-symmetric solution to be found in the right branch is that both the solution and its Rx-symmetric version satisfies $Q[1] \leq 5$. Now the only solutions that satisfying this conditions are solutions that satisfy $Q[1] = 5$. Hence, we can add $\neg S^{Rx}(Q[2] = 3)$ *under the condition* that $Q[1] = 5$. But this is exactly the effect that we achieve by adding $(Q[1] \geq 10 - 5) \rightarrow (Q[2] \neq 10 - 3)$ at the second right branch (since the constraint store contains already $Q[1] \leq 5$).

Preliminaries. We fix a first-order signature Σ including the equality $\doteq$ and a set of variables $\mathcal{V}$. Constraints are literals, and constraint formulae are quantifier-free formulae over Σ. We identify $t \doteq t'$ with $t' \doteq t$. $\mathcal{C}$ denotes the set of all constraints. A set of constraints $C \subseteq \mathcal{C}$ is interpreted as the conjunction of the constraints contained in C, and we will freely mix set notation and conjunction. The set of free variables of C is denoted by $\mathcal{V}(C)$.

We fix a standard interpretation $\mathcal{A}$ with domain $\mathcal{D}^{\mathcal{A}}$, which describes our constraint theory. In the following, we assume a fixed constraint set C_{Pr} describing the problem to be solved. An *assignment* α *in* $\mathcal{A}$

is a partial function $\alpha : \mathcal{V} \to \mathcal{D}^{\mathcal{A}}$. We say that a (possibly partial) assignment α *satisfies* ϕ (short $\alpha \models \phi$) if there is a total assignment $\alpha' \supseteq \alpha$ with $\mathcal{A}, \alpha' \models \phi$. We write $\phi \models \psi$ if for every α we have $\alpha \models \phi \to \psi$. A constraint set C *(syntactically) determines* a set of variables $\mathcal{X}$ to an assignment α iff for all $x \in \mathcal{X}$ exists a ground term t such that $x \doteq t \in C$ and $\alpha(x) = t^{\mathcal{A}}$.

In many problems, one is interested only in a subset of all variables.

DEFINITION 1 (Solution Variables). *Let $\mathcal{X} \subseteq \mathcal{V}$ be a set of variables. $\mathcal{X}$ is a* solution variables set for C *if for all α*

$$\Big(\mathrm{dom}(\alpha) = \mathcal{X} \wedge \alpha \models C\Big) \Rightarrow \Big(\exists! \alpha' : \alpha \subseteq \alpha' \wedge \mathrm{dom}(\alpha') = \mathcal{V}(C) \wedge \alpha' \models C\Big),$$

*where $\exists! \alpha \phi(\alpha)$ ('exists a unique α satisfying $\phi(\alpha)$') is short for the expression $\exists \alpha(\phi(\alpha) \wedge \forall \alpha' : \phi(\alpha') \Rightarrow \alpha = \alpha')$. For a solution variables set $\mathcal{X}$, we say that α is a $\mathcal{X}$-*solution for C *if* $\mathrm{dom}(\alpha) = \mathcal{X}$ *and* $\alpha \models C$. *With $\|C\|^{\mathcal{X}}$ we denote the set of $\mathcal{X}$-solutions of C.*

In the following, we fix $\mathcal{X}$. Hence, we use the term 'solution' as short for '$\mathcal{X}$-solution for C', and we write $\|C\|$ as short for $\|C\|^{\mathcal{X}}$.

DEFINITION 2 (Symmetry). *A* symmetry s for C_{Pr} *is a bijective function $s : \|C_{\mathrm{Pr}}\| \to \|C_{\mathrm{Pr}}\|$. A* symmetry set $\mathcal{S}$ for C_{Pr} *is a set of symmetries for C_{Pr}. A* symmetry group $\mathcal{S}$ *is a symmetry set for C_{Pr} which is a group.*

Note that for every symmetry s for C_{Pr}, also s^{-1} is a symmetry for C_{Pr}. We denote the identity function on $\|C_{\mathrm{Pr}}\|$ with $\mathrm{id}_{C_{\mathrm{Pr}}}$ (which is a symmetry by definition). Clearly, the set of all symmetries for C_{Pr} is a group. In many cases, we do not want to consider all symmetries, since either there are too many of them, or some of them do not have an intuitive characterization.

2. Symmetry Excluding Search Trees

Usually, a search tree in constraint programming is a labeled tree, where the nodes are labeled with the constraints in the constraint store, and the edges are labeled by constraints or negated constraints. Nodes are labled by constraint stores. The root node is labeled with $P(C_{\mathrm{Pr}})$ (where $P(\cdot)$) describes the effect of propagation). For a binary node with constraint store C_{store} that branches over the constraint c, the left (resp. right) subnode has the constraint store $P(C_{\mathrm{store}} \wedge c)$ (resp. $P(C_{\mathrm{store}} \wedge \neg c)$.

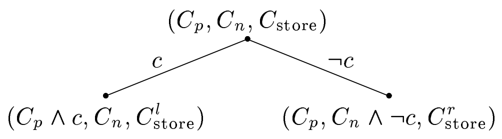

$(C_p, C_n, C_{\text{store}})$
c
$\neg c$
$(C_p \wedge c, C_n, C^l_{\text{store}})$
$(C_p, C_n \wedge \neg c, C^r_{\text{store}})$

Figure 1. General form of binary search nodes

We need a more general form of search trees, which allow us to exclude some solutions (namely symmetrical ones). In addition, we need at a node v the positive and negative constraints that have been used along the path from the root to v to generate the node v.

DEFINITION 3 (Search Tree). *Let t be a finite, binary, rooted and ordered tree, whose edges are labeled by literals, and whose nodes are labeled by triples of constraint sets. The tree t is a* search tree for C_{Pr} *if the following conditions are satisfied:*

1. *The root node has the label $(\emptyset, \emptyset, P(C_{\text{Pr}}))$,*
2. *every binary node has the form as given by Figure 1, where $C^l_{\text{store}} \models C_{\text{store}} \wedge c$ and $C^r_{\text{store}} \models C_{\text{store}} \wedge \neg c$.*

Intuitively, the reason for distinguishing between positive and negative constraints is just that the negative constraints C_{n} describe the previously found solutions, i.e. $\forall v' \prec_t v \forall \alpha \in \|C_{\text{store}}(v')\| : (\alpha \models \neg C_{\text{n}})$. Here, we denote with $\prec_t$ the partial ordering of nodes induced by t.

Note that we do not force C^l_{store} (resp. C^r_{store}) to be equivalent to $C_{\text{store}} \wedge c$ (resp. $C_{\text{store}} \wedge \neg c$). The reason is that we must add some additional constraints during search for excluding symmetries. Since C_{store} describes all constraints valid at some node v, we set $\|v\| = \|C_{\text{store}}\|$ for a node v in t with label $(C_{\text{p}}, C_{\text{n}}, C_{\text{store}})$.

DEFINITION 4 (Expanded, C_{Pr}-Complete and S-Reduced Trees). *The search tree t is* completely expanded *if for every unary node $v = (C_{\text{p}}, C_{\text{n}}, C_{\text{store}})$, either $\perp \in C_{\text{store}}$, or $\|v\| = \{\alpha\}$ and C_{store} syntactically determines $\mathcal{X}$ to α. A search tree t is* C_{Pr}-complete *if for every $\alpha \in \|C_{\text{Pr}}\|$ there is a leaf v in t with $\{\alpha\} = \|v\|$. Let $\mathcal{S}$ be a symmetry set for C_{Pr}. A search tree is* C_{Pr}-complete w.r.t. $\mathcal{S}$ *if for every $\alpha \in \|C_{\text{Pr}}\|$ there is a leaf v such that*

$$\|v\| = \{\alpha\} \ \vee \ \exists s \in \mathcal{S} : \|v\| = \{s(\alpha)\}.$$

A search tree is $\mathcal{S}$-reduced *if for every leaf v with $\|v\| = \{\alpha\}$ we have*

$$\forall s \in \mathcal{S} \ \forall \textit{leafs } v' \textit{ with } v' \prec_t v : (\|v'\| = \{\alpha'\} \Rightarrow s(\alpha') \neq \alpha). \qquad (1)$$

Note that whenever $\mathcal{S}$ is closed under inversion, (1) is equivalent to $\forall\, s \in \mathcal{S} \;\; \forall\, v' \neq v : \|v'\| = \{\alpha'\} \Rightarrow s(\alpha') \neq \alpha$.

Before we can show how to produce an $\mathcal{S}$-reduced search tree that is C_{Pr}-complete w.r.t. $\mathcal{S}$, we first have to define how symmetries operate on constraints. In the following, we will assume that for every constraint c and every s, there is a constraint c' such that $s^*(\|c\|) = \|c'\|$, where we define s^* on sets of C_{Pr}-solutions by $s^*(A) = \{s(\alpha) \mid \alpha \in A\}$. For every c, there are usually more than one different constraints c' with $s^*(\|c\|) = \|c'\|$. Hence, fix a function s_{con} on constraints such that

$$s^*(\|c\|) = \|s_{\mathrm{con}}(c)\|.$$

PROPOSITION 1. *Let s_{con} be defined as before. Then s_{con} distributes over the Boolean operators, i.e. $s^*(\|c \wedge c'\|) = \|s_{\mathrm{con}}(c) \wedge s_{\mathrm{con}}(c')\|$, $s^*(\|c \vee c'\|) = \|s_{\mathrm{con}}(c) \vee s_{\mathrm{con}}(c')\|$ and $s^*(\|\neg c\|) = \|s_{\mathrm{con}}(\neg c)\|$.*

Hence, we identify s_{con} with its homomorphic extension to constraint sets and arbitrary formulae.

DEFINITION 5 ($\mathcal{S}$-excluding Search Tree). *Let $\mathcal{S}$ be a symmetry set. A search tree for C_{Pr} is* $\mathcal{S}$-excluding *if every binary node v has the form as given by Figure 1, and*

$$C^l_{\mathrm{store}} = P(C_{\mathrm{store}} \wedge c) \quad \wedge \quad C^r_{\mathrm{store}} = P(C_{\mathrm{store}} \wedge \neg c \wedge E^v(\neg c)),$$

where $E^v(\neg c) = \bigwedge_{s \in \mathcal{S}} s_{\mathrm{con}}(C_{\mathrm{p}}) \rightarrow \neg s_{\mathrm{con}}(c)$.

Before we prove $\mathcal{S}$-reducedness and C_{Pr}-completeness w.r.t. $\mathcal{S}$ of $\mathcal{S}$-excluding search trees, we state a proposition to precise the effect of adding the implications in definition 5 to C^r_{store}. For convenience, we write C^v_{p}, C^v_{n} and C^v_{store} to access C_{p}, C_{n} and C_{store} in the label $(C_{\mathrm{p}}, C_{\mathrm{n}}, C_{\mathrm{store}})$ of v. For a binary node v, we refer to its left child as v_l and to its right child as v_r.

PROPOSITION 2. *Let $\mathcal{S}$ be a symmetry set and t an $\mathcal{S}$-excluding search tree. For every symmetry $s \in \mathcal{S}$ and for every node v of t we have $C^v_{\mathrm{store}} \models s_{\mathrm{con}}(C^v_{\mathrm{p}}) \rightarrow s_{\mathrm{con}}(C^v_{\mathrm{n}})$.*

For notational convenience, we introduce the notation $C_{path}(v) = C^v_p \wedge C^v_n \wedge C_{\mathrm{Pr}}$. A good intuition is to think of $C_{path}(v)$ describing the constraint store of v in a simple not symmetry excluding search tree.

LEMMA 1. *Let $\mathcal{S}$ be a symmetry set. Every $\mathcal{S}$-excluding search tree t for C_{Pr} satisfies for every node v that $\|v\| = \|C_{path}(v)\| - A^v_{\mathcal{S}}$, where $A^v_{\mathcal{S}} = \{s(\alpha) \mid s \in \mathcal{S} \wedge \exists v' \prec_t v : \alpha \in \|C_{path}(v')\|\}$.*

Proof. We proof this by tree induction. For the root this is valid, since $A^v_{\mathcal{S}} = \emptyset$ and $\|v\| = \|C_{\mathrm{Pr}}\| = \|C_{path}(v)\|$. Assume that we have proven the claim for a binary node v as given by Figure 1.

For the left child v_l, the claim follows immediately from the induction hypotheses, since $A^{v_l}_{\mathcal{S}} = A^v_{\mathcal{S}}$.

For v_r, we have to show $\|v_r\| = \|C_{path}(v_r)\| - A^{v_r}_{\mathcal{S}}$. Note that $A^{v_r}_{\mathcal{S}} = \{s(\alpha) \mid s \in \mathcal{S} \land \exists v' \prec_t v_r : (\alpha \in \|C_{path}(v')\|)\}$ subdivides into its two subsets $A^v_{\mathcal{S}}$ and $\{s(\alpha) \mid s \in \mathcal{S} \land \exists v'(v' \not\prec_t v \land v' \prec_t v_r \land \alpha \in \|C_{path}(v')\|)\}$. Further, note that $C_{path}(v_r)$ is equivalent to $C_{path}(v) \land \neg c$.

To show that $\|v_r\| \subseteq \|C_{path}(v) \land \neg c\| - A^{v_r}_{\mathcal{S}}$, fix a symmetry $s \in \mathcal{S}$. We have to show for every $v' \prec_t v$ that $\forall \alpha \in \|C_{path}(v')\| : s(\alpha) \notin \|v_r\|$.

The first case is $v' \prec_t v$. Let $\alpha \in \|C_{path}(v')\|$. Then $s(\alpha) \notin \|v\|$ by induction hypotheses. Since $\|v_r\| \subseteq \|v\|$, this immediately implies $s(\alpha) \notin \|v_r\|$.

The second case is $(v' \not\prec_t v)$ and $v' \prec_t v_r$, i.e. v' is a subnode of v_l. Let $\alpha \in \|v_l\|$ and assume $s(\alpha) \in \|v_r\|$. From $\alpha \in \|v_l\|$ we have $\alpha \models C_{\mathrm{p}} \land c$ and $s(\alpha) \models s_{\mathrm{con}}(C_{\mathrm{p}} \land c)$. That's a contradiction, because from definition of $\mathcal{S}$-excluding search tree $s(\alpha) \models s_{\mathrm{con}}(C_{\mathrm{p}}) \to \neg s_{\mathrm{con}}(c)$, since $(s_{\mathrm{con}}(C_{\mathrm{p}}) \to \neg s_{\mathrm{con}}(c)) \in E^v(\neg c)$.

It remains to be shown that $\|v_r\| \supseteq \|C_{path}(v_r)\| - A^{v_r}_{\mathcal{S}}$ Let $\alpha \notin \|v_r\|$. We have to show that $\alpha \notin \|C_{path}(v_r)\| - A^{v_r}_{\mathcal{S}}$

We have the following cases:

1. $\alpha \notin \|C_{path}(v_r)\|$. Then the claim follows immediately.
2. $\alpha \in \|C_{path}(v_r)\|$ and $\alpha \notin \|v\|$. We have to show $\alpha \in A^{v_r}_{\mathcal{S}}$.

Now $\alpha \in \|C_{path}(v_r)\|$ implies $\alpha \in \|C_{path}(v)\|$ and henceforth $\alpha \in A^v_{\mathcal{S}} \subseteq A^{v_r}_{\mathcal{S}}$ by induction hypotheses.

3. $\alpha \in \|C_{path}(v_r)\|$ and $\alpha \in \|v\|$. We will show that $\alpha \in A^{v_r}_{\mathcal{S}} - A^v_{\mathcal{S}}$

$$\begin{aligned} &= \{s(\beta) \mid s \in \mathcal{S} \land \exists v' : (v' \prec_t v_r \land v' \not\prec_t v \land \beta \in \|C_{path}(v')\|\} \\ &= \{s(\beta) \mid s \in \mathcal{S} \land \beta \in \|C_{path}(v_l)\|\} \end{aligned} \tag{2}$$

Since $\alpha \models C_{\mathrm{store}} \land \neg c$, but $\alpha \not\models C^r_{\mathrm{store}}$, there is at least one $s \in \mathcal{S}$ with $\alpha \not\models s_{\mathrm{con}}(C_{\mathrm{p}}) \to \neg s_{\mathrm{con}}(c)$ by the definition of an S-excluding tree. Fix one of these symmetries s. Then $\alpha \models s_{\mathrm{con}}(C_{\mathrm{p}}) \land s_{\mathrm{con}}(c)$. Since $C_{\mathrm{store}} \models s_{\mathrm{con}}(C_{\mathrm{p}}) \to s_{\mathrm{con}}(C_{\mathrm{n}})$ by Proposition 2, we get

$$\alpha \models s_{\mathrm{con}}(C_{\mathrm{p}}) \land s_{\mathrm{con}}(c) \land s_{\mathrm{con}}(C_{\mathrm{n}}).$$

Hence, $s^{-1}(\alpha)$, which exists by definition of symmetry, satisfies $s^{-1}(\alpha) \models C_{path}(v) \land c$ (recall that $C_{path}(v) = C_{\mathrm{p}} \land C_{\mathrm{n}}$). That is, $s^{-1}(\alpha) \in \|C_{path}(v_l)\|$. Hence, with $\beta = s^{-1}(\alpha)$ we have a valuation such that $\beta \in \|C_{path}(v_l)\|$ and $\alpha = s(\beta)$, which shows that α is in the set defined by (2). □

THEOREM 1. *Let $\mathcal{S}$ be a symmetry set. Every $\mathcal{S}$-excluding search tree t for C_{Pr} is $\mathcal{S}$-reduced.*

Proof. Fix a symmetry $s \in \mathcal{S}$, let v be a leaf of t with $\|v\| = \{\alpha\}$. We have to show for every node v' of t with $v' \prec_t v$ that $\|v'\| = \{\alpha'\} \rightarrow s(\alpha') \neq \alpha$. Assume v' to be a node of t with $v' \prec_t v$ and $\|v'\| = \{\alpha'\}$ and $s(\alpha') = \alpha$. It follows $\alpha \in \{s(\alpha) \mid s \in \mathcal{S} \wedge \exists v' \prec_t v : \alpha \in \|C_{path}(v')\|\}$, and from this the contradiction $\alpha \notin \|v\|$ by Lemma 1. □

By now, we understand that an $\mathcal{S}$-excluding search tree may exclude more symmetries than actually declared by $\mathcal{S}$ directly. Hence, we have to investigate the completeness property of $\mathcal{S}$-excluding search trees.

THEOREM 2. *Let $\mathcal{S}$ be a symmetry set. Every $\mathcal{S}$-excluding search tree t for C_{Pr} is C_{Pr}-complete w.r.t. $\mathcal{S}'$, where $\mathcal{S}'$ is the closure of $\mathcal{S}$ under composition and inversion.*

Proof. We have to show that t is C_{Pr}-complete w.r.t. to $\mathcal{S}'$, where $\mathcal{S}'$ is the closure under composition of $\mathcal{S}$, i.e. for every $\alpha \in \|C_{\mathrm{Pr}}\|$ there is a leaf v with $\|v\| = \{\alpha\} \ \vee \ \exists s \in \mathcal{S}' : \|v\| = \{s(\alpha)\}$. Hence fix an $\alpha \in \|C_{\mathrm{Pr}}\|$. Then there exists a leaf v' in t with $\alpha \in \|C_{path}(v')\|$ from definition 5. There are two cases:
1.) $\alpha \in \|v'\|$. Then v' itself is the node v searched for.
2.) $\alpha \notin \|v'\|$. By Lemma 1 it follows $\alpha \in A^{v'}_{\mathcal{S}}$, i.e. there is a $s_1 \in \mathcal{S}$ and a $v'' \prec_t v'$ such that $\alpha = s_1(\alpha')$, where $\alpha' \in \|C_{path}(v'')\|$. By induction, we get a sequence of symmetries $s_1, \ldots, s_n \in \mathcal{S}$ and a leaf $v^{(n+1)}$ of t, where $\alpha = s_1 \circ \cdots \circ s_n(\alpha^{(n)})$ and $\{\alpha^{(n)}\} = \|v^{(n+1)}\|$. Then $v^{(n+1)}$ is the node v searched for since $\mathcal{S}'$ is the closure of $\mathcal{S}$. □

We will given an example for symmetry excluding search trees in Section 5, where the symmetry group of permutation are excluded by the subset of all transpositions (as indicated by the preceeding theorem).

Now, we are able to give a more detailed comparison to [8]. The *SBDS-method* therein is essentially our symmetry exclusion restricted to the case, where the search tree branches over constraints of the form $x \doteq t$, where x is a variable and t is a ground term (think of it as a value). In this case, the constraint collected in C_{p} is of the form $x_1 \doteq t_1 \wedge \ldots \wedge x_n \doteq t_n$, i.e. C_{p} is a partial assignment α.

We motivate the *restricted SBDS-method* with an example. Let a symmetry S_σ be defined by a permutation σ such that $S_\sigma(x \doteq t)$ is $x \doteq \sigma(t)$. Now, let C_{p} be $x_1 \doteq t_1 \wedge \ldots \wedge x_n \doteq t_n$ at some node v of the search tree, which branches over $x \doteq t$. Then, the constraint added by our method for excluding the symmetry S_σ is $S_\sigma(x_1 \doteq t_1 \wedge \ldots \wedge x_n \doteq t_n) \implies \neg S_\sigma(x \doteq t)$, which is by the definition of S_σ the same as

$$x_1 \doteq \sigma(t_1) \wedge \ldots \wedge x_n \doteq \sigma(t_n) \implies x \neq \sigma(t).$$

Now, the antecedent is nothing else than a test, whether S_σ leaves C_{p} (i.e., the partial assignment at the node v) unchanged. Thus, the entailment can be judged directly at the right subnode of v. In general, the entailment can not be decided immediately. However, the restricted SBDS-method essentially replaces the entailment test by the test, whether the symmetry leaves the assignment unchanged. Only those symmetries are excluded, for that the test can be decided immediately at the very node, where our method would introduce the corresponding implication. In our approach, the corresponding implication delays this test to a later point, if it is not decidable yet.

3. Symmetry Excluding Search Algorithms

Algorithms. We will discuss two algorithms implementing symmetry-excluding search trees. A straightforward approach yields procedure naive_ses at the end of this paragraph, here called the *naive algorithm.* This algorithm works by keeping track of C_p to add the implications on the right branch to C_store.

However, the antecedents of these implications can be computed during the search more efficiently, since the antecedents of the implications added at a node are prefixes of the ones added in its offspring. More precisely, let v be a node, which branches over c. Let v^l (resp. v^r) denote its left (resp. right) daughter. Now, $s_\mathrm{con}(C_\mathrm{p}^{v^l}) = s_\mathrm{con}(C_\mathrm{p}^v \wedge c) = s_\mathrm{con}(C_\mathrm{p}^v) \wedge s_\mathrm{con}(c)$. Further, $s_\mathrm{con}(C_\mathrm{p}^{v^r})$ equals $s_\mathrm{con}(C_\mathrm{p}^v)$. Therefore, the antecedents $s_\mathrm{con}(C_\mathrm{p})$ can be computed incrementally. For this aim we maintain a family of Boolean variables $\overrightarrow{Cp} = (Cp_s)_{s\in\mathcal{S}}$ that reify the trueth values of $s_\mathrm{con}(C_\mathrm{p})$ for each symmetry $s \in \mathcal{S}$. Employing this technique we get the *improved algorithm ses.* In ses, for the left branch, we add reified constraints $Cp_s^l \iff Cp_s \wedge s_\mathrm{con}(c)$. The family $\overrightarrow{Cp^l}$ is passed to the left subtree. For the right branch, we insert only the now modified implications.

```
procedure naive_ses($C_\mathrm{p}, C_\mathrm{store}$)
if $C_\mathrm{store}$ determines solution $\alpha$
  then write $\alpha$
elsif $\bot \notin C_\mathrm{store}$ then
  choose a constraint $c$
  $C^l_\mathrm{store} := C_\mathrm{store} \wedge c$
  $C^l_\mathrm{p} := C_\mathrm{p} \wedge c$
  naive_ses($C^l_\mathrm{p}, C^l_\mathrm{store}$)
  $C^r_\mathrm{store} := C_\mathrm{store} \wedge \neg c$
    $\wedge \forall s \in \mathcal{S} : s_\mathrm{con}(C_\mathrm{p}) \to s_\mathrm{con}(\neg c)$
  naive_ses($C_\mathrm{p}, C^r_\mathrm{store}$)
endif.
```

```
procedure ses($\overrightarrow{Cp}, C_\mathrm{store}$)
  if $C_\mathrm{store}$ determines solution $\alpha$
    then write $\alpha$
  elsif $\bot \notin C_\mathrm{store}$ then
    choose a constraint $c$
    $C^l_\mathrm{store} := C_\mathrm{store} \wedge c$
      $\wedge \forall s \in \mathcal{S} : \overrightarrow{Cp^l_s} \iff \overrightarrow{Cp_s} \wedge s_\mathrm{con}(c)$
    ses($\overrightarrow{Cp^l}, C^l_\mathrm{store}$)
    $C^r_\mathrm{store} := C_\mathrm{store} \wedge \neg c$
      $\wedge \forall s \in \mathcal{S} : \overrightarrow{Cp_s} \to s_\mathrm{con}(\neg c)$
    ses($\overrightarrow{Cp}, C^r_\mathrm{store}$)
  endif.
```

Analytical Run Time Comparison. We will concentrate on the question how many symmetric constraints are computed in the different algorithms for one search. For the approximation of the speedup, we have to employ some standard definitions for binary trees.

We define a *(binary) tree* t as a prefix-closed subset of $\{0,1\}^*$. The *depth of a tree* t is $depth(t) = max\{|w| \mid w \in t\}$. A *node* v *of a tree* t is a $v \in t$. A *leaf of a tree* t is a $v \in t$, such that $v1 \notin t \wedge v0 \notin t$. The *left subtree of a tree* t is $\mathrm{left}(t) = 1^{-1}t$ and the *right subtree* $\mathrm{right}(t) = 0^{-1}t$. The *path length of a node* v is $|v|$. We define the *left path length of a node* $v = a_1 \dots a_{|v|}$ as $|v|_l = |\{i \in \{1, \dots, |v|\} \mid a_i = 1\}| = \sum_{1 \le i \le |v|} a_i$. The *right path length of a node* v is given by $|v|_r = |v| - |v|_l$. A tree t is *completely expanded* iff there is a d, such that $t = \{0,1\}^{\le d}$. We define a certain number of nodes

$$n_d^l(k) = |\{v \mid v \text{ node of } t = \{0,1\}^{\le d} \wedge |v|_l = k\}|.$$

PROPOSITION 3. *For a completely expanded tree* t*, its left and right subtrees are completely expanded. For a tree* t *and every node* $v \in t$*, it is satisfied* $|v|_l \le |v|$ *and* $|v| \le depth(t)$.

LEMMA 2. *Let* $k, d \in \mathbb{N},\ k, d \ge 0$*. Then,* $n_d^l(k) = \begin{cases} \binom{d+1}{k+1} & \text{for } k \le d \\ 0 & \text{otherwise.} \end{cases}$

Proof. Let t be the completely expanded tree with depth d. Let a function $r:\ t \to \{0,1\}^{d+1} - \{0^{d+1}\}$ be given by $r:\ v \mapsto v10^{d-|v|}$.

For every $v \in t$, $r(v)$ is an element of $\{0,1\}^{d+1}$ and contains exactly $|v|_l + 1$ ones. Since r is bijective, the number of nodes v with $|v|_l = k$ is equal to the number of words $r(v)$ of length $d+1$ containing $k+1$ ones. There are $\binom{d+1}{k+1}$ such words. □

In a search tree t, we call the constraints c and $\neg c$ that are inserted at the branches of a binary node *basic constraints.* We will need a measure for the total time the algorithms spend on the computation of symmetric constraints. As this measure, we just count the number of computed symmetric basic constraints. This is reasonable, since we may assume for an approximation that the computation of each symmetric basic constraint takes equal time and further that the computation of a complex symmetric constraint $s_{\mathrm{con}}(c_1 \wedge \dots \wedge c_n)$ is done via computing the basic symmetric constraints $s_{\mathrm{con}}(c_1), \dots, s_{\mathrm{con}}(c_n)$.

THEOREM 3. *To produce a completely expanded tree of depth* d *the naive algorithm computes* $|\mathcal{S}|\left(2^{d-1}(d-2)+1\right)$ *many symmetric basic constraints, while the improved algorithm computes* $|\mathcal{S}|\left(2^d - 1\right)$ *such constraints.*

Proof. In a completely expanded tree t of depth d, the naive algorithm computes for a node v with $|v| < d$ exactly $|v|_l$ symmetric basic constraints for every symmetry $s \in \mathcal{S}$. For nodes v with path length $|v| = d$, it does not compute any symmetric constraint. Hence, we get our number for one symmetry by summing over all nodes, i.e. by computing $\sum_{v \in \{1,0\}^{\leq d-1}} |v|_l$. We simplify further

$$\sum_{i=0}^{d-1} i \cdot n^l_{d-1}(i) = \sum_{i=0}^{d-1} i \cdot \binom{d}{i+1} = \sum_{i=1}^{d} (i-1) \cdot \binom{d}{i} = \sum_{i=1}^{d} i \cdot \binom{d}{i} - \sum_{i=1}^{d} \binom{d}{i}$$
$$= \sum_{i=0}^{d} i \cdot \binom{d}{i} - \sum_{i=0}^{d} \binom{d}{i} + 1 = \frac{1}{2} d \cdot 2^d - 2^d + 1 = (d-2) \cdot 2^{d-1} + 1.$$

In the same tree t, the improved algorithm computes for every node v with $|(|v) < d$ exactly one symmetric basic constraint for one symmetry, i.e. the total number of symmetric basic constraints is the number of such nodes v, i.e. $2^d - 1$.

Both numbers of symmetric basic constraints are derived for one symmetry, and thus have to be multiplied by $|\mathcal{S}|$, finally. $\square$

As a corollary, simple calculation shows the speedup by the improved algorithm in the computation of symmetric basic constraints, i.e. $\frac{|\mathcal{S}|(2^{d-1}(d-2)+1)}{|\mathcal{S}|(2^d-1)} \approx \frac{d-2}{2}$, to be linear.

The second most runtime-relevant factor is the check of the antecedents of the inserted implications by the constraint solver. For the naive algorithm, it is reasonable to assume that the solver checks each antecedent of the inserted implications at each node at least once. In contrast to this, the improved algorithm checks each branching constraint of C_{p} once for each node. Under this assumption, again the speedup turns out to be linear in the depth of the tree.

4. Geometric Symmetries

Now, we will give a concrete example for symmetries, namely geometry symmetries. We will treat the special case where we have points in $\mathbb{Z}^d$, although our method is not restricted to this case. We will exemplify the symmetry constructions in two dimensions, and give an example for the exclusion of additional symmetries as indicated by Theorem 2.

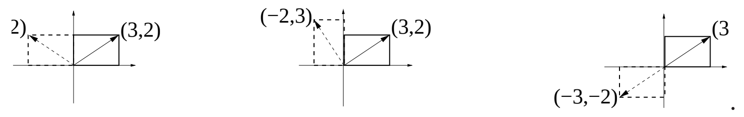
Definition of Geometric Symmetries. There are many problems where one encodes the two-dimensional position of an object i using finite domain integer variables X_i, Y_i. Examples are the N-queens and the tiling problem. A more complex and realistic example (in three dimensions) is the lattice protein structure prediction problem [2].

The symmetries for $\mathbb{Z}^2$ have exactly the same structure as for the general case $\mathbb{Z}^d$. They are defined by affine mappings $S : \mathbb{Z}^d \to \mathbb{Z}^d$ with $S(\vec{x}) = A_S\vec{x} + \vec{v}_S$ that map $\mathbb{Z}^d$ onto $\mathbb{Z}^d$. That is, the matrix A_S is an orthogonal matrix with the property that the set of columns $\{\vec{v}_1, \ldots, \vec{v}_d\}$ of A_S equals $\{\pm\vec{e} \mid \vec{e}$ is a unit-vector of $\mathbb{Z}^d\}$. For example, for $\mathbb{Z}^2$, the matrix $\left(\begin{smallmatrix} 0 & -1 \\ 1 & 0 \end{smallmatrix}\right)$ denotes the rotation by 90°. For $\mathbb{Z}^2$, we have 8 symmetries consisting of the identity, the 4 reflections (at x- and y-axis, and the two diagonals) and the 3 rotations by 90°, 180° and 270°. For $\mathbb{Z}^3$, we have 48 symmetries including the identity.

Fixing the Symmetries. By now, the vector $\vec{v}_S$ is not yet fixed. There are two different approaches for fixing the symmetry. We consider $\mathbb{Z}^2$ as an example. The methods work for all other dimensions as well.

The first case is that every possible solution lies within a fixed square (in the general case, within a hypercube).[2] This is equivalent to the proposition that there are integers $x_{min}, x_{max}, y_{min}, y_{max}$ such that for all $\alpha \in \|C_{\mathrm{Pr}}\|$ we have $\min\{\alpha(X_i)\} = x_{min}$, $\max\{\alpha(X_i)\} = x_{max}$, $\min\{\alpha(Y_i)\} = y_{min}$, and $\max\{\alpha(Y_i)\} = y_{max}$. Thus, the minimal square around the position of all objects is defined by the points (x_{min}, y_{min}) and (x_{max}, y_{max}) in every solution. We call this the *frame* of the problem C_{Pr}. In the N-queens problem, this is just the board.

Now, knowing the frame of a problem C_{Pr}, we can fix the vector $\vec{v}_S$ for all symmetries. Consider as an example a problem whose frame is defined by $(0,0)$ and $(3,2)$. Furthermore, consider the three symmetries reflection at the y-axis, rotation by 90° and rotation by 180°, which we will name S_1, S_2 and S_3 in the following. The corresponding mappings are defined by $S_i(\vec{x}) = A_{S_i}(\vec{x}) + \vec{v}_{S_i}$, where $A_{S_1} = \left(\begin{smallmatrix} -1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$, $A_{S_2} = \left(\begin{smallmatrix} 0 & -1 \\ 1 & 0 \end{smallmatrix}\right)$ and $A_{S_3} = \left(\begin{smallmatrix} -1 & 0 \\ 0 & -1 \end{smallmatrix}\right)$. The corresponding mappings of the frame are



A symmetry S is compatible with the above defined frame if the frame is mapped to itself, i.e. if $\{\vec{v} \mid (0,0) \leq \vec{v} \leq (3,2)\} = \{S(\vec{v}) \mid (0,0) \leq \vec{v} \leq (3,2)\}$. For a given matrix A_S, there exists a $\vec{v}_S$ such that $S(\vec{x}) = A_S\vec{x} + \vec{v}_S$ satisfies this condition if and only if A_S satisfies $A_S(3,2) = (\pm 3, \pm 2)$. For the matrices A_{S_1}, A_{S_2} and A_{S_3}, we get $(-3,2)$, $(-2,3)$ and $(-3,-2)$, which excludes the symmetry characterized by A_{S_2}. We finally get $\vec{v}_{S_1} = (3,0)$ and $\vec{v}_{S_3} = (3,2)$.

[2] The technique can be extended to the case that the hypercube is not fixed in advance, but during search.

The second case is that we know a point $\vec{p} = (p_x, p_y)$ which should remain unchanged under the symmetries. In that case, we know that the symmetries are defined by $S_i(\vec{x}) = A_{S_i}(\vec{x}-\vec{p})+\vec{p} = A_{S_i}(\vec{x}) - A_{S_i}(\vec{p})+\vec{p}$. Hence, $\vec{v}_{S_i} = \vec{p} - A_{S_i}(\vec{p})$.

The remaining part is to define symmetric constraints. We use a specific example where we leave the point $(5,5)$ fix. Consider the two symmetries reflection at the y-axis and rotation by 90°. By what we have said above, the corresponding mappings are

$$S^{\mathrm{Ry}}(\vec{x}) = \left(\begin{smallmatrix} -1 & 0 \\ 0 & 1 \end{smallmatrix}\right)\vec{x} + \left(\begin{smallmatrix} 10 \\ 0 \end{smallmatrix}\right) \quad \text{and} \quad S^{90^\circ}(\vec{x}) = \left(\begin{smallmatrix} 0 & -1 \\ 1 & 0 \end{smallmatrix}\right)\vec{x} + \left(\begin{smallmatrix} 10 \\ 0 \end{smallmatrix}\right)$$

Now suppose that we have modeled points $p_1, \dots, p_n$ using variables $X_1, \dots, X_n$ and $Y_1, \dots, Y_n$, and we want to define S^{Ry}_{con} and $S^{90^\circ}_{con}$ for the constraints of the form $X_i = c$ (for other kind of constraints c, the definition $S_{con}(c)$ is analogous). Now the symmetric constraints $S^{\mathrm{Ry}}_{con}(X_i = c)$ (resp. $S^{90^\circ}_{con}(X_i = c)$) must express the constraint valid for $S^{\mathrm{Ry}}(\alpha)$ (resp. $S^{90^\circ}(\alpha)$) for every possible α with $\alpha \models X_i = c$. Then,

$$\begin{aligned} \alpha \models & X_i = c \\ & \Leftrightarrow \vec{p_i} = (c, \alpha(Y_i)) \\ & \Leftrightarrow S^{\mathrm{Ry}}(\vec{p_i}) = (10-c, \alpha(Y_i)) \\ & \Leftrightarrow S^{\mathrm{Ry}}(\alpha) \models X_i = 10 - c \end{aligned} \qquad \begin{aligned} \alpha \models & X_i = c \\ & \Leftrightarrow \vec{p_i} = (c, \alpha(Y_i)) \\ & \Leftrightarrow S^{90^\circ}(\vec{p_i}) = (10-\alpha(Y_i), c) \\ & \Leftrightarrow S^{90^\circ}(\alpha) \models Y_i = c \end{aligned}$$

Since $X_i = c$ does not restrict the valuation of $\alpha(Y_i)$, we know that $S^{\mathrm{Ry}}_{con}(X_i = c)$ is $X_i = 10 - c$, and $S^{90^\circ}_{con}(X_i = c)$ is $Y_i = c$. Analogously, we get that $S^{\mathrm{Ry}}_{con}(Y_i = c)$ is $Y_i = c$, and that $S^{90^\circ}_{con}(Y_i = c)$ is $X_i = 10 - c$. Note that $S^{\mathrm{Ry}}(c)$ has the same type as c (i.e., both have the same variable). This does not hold for c and $S^{90^\circ}(c)$.

Results. We have applied the method to the lattice protein structure prediction [2], which is a hard combinatorial problem. Table I shows the number of solutions, search steps and runtimes for finding all minimal energy structures for 4 sequences, both without and with symmetry exclusion. We have added simple exclusion methods in both cases, which is the reason that we have only 16 (instead of 48) symmetries left in sequences 1, 2 and 4. As one can see from the table, we have a nearly linear speedup in number of search steps, and a close to linear speedup in runtime. In sequence 3, the simple exclusion does not apply, which gives rise to 48 symmetries. Furthermore, the optimal conformations and the symmetric ones are very similar, which implies that we have less speedup (since detecting the symmetric solution is harder).

Table I. Results for searching all minimal energy structures without and with symmetry exclusion.

len	without sym. excl.			with sym. excl.			ratio		
	$\#_{noex}$	n_{noex}	t_{noex}	$\#_{ex}$	n_{ex}	t_{ex}	$\frac{\#_{noex}}{\#_{ex}}$	$\frac{n_{noex}}{n_{ex}}$	$\frac{t_{noex}}{t_{ex}}$
27	4,752	45,924	15 m	297	2,998	69 s	16	15.32	13.17
27	408,864	2,465,728	6.2 h	25,554	155,693	26.3 m	16	15.84	14.18
31	53,472	351,101	3.4 h	1,114	11,036	7.4 m	48	31.81	27.43
36	56,448	732,952	3.7 h	3,528	55,086	19.9 m	16	13.31	11.24

5. Value Permutation symmetries

We will consider the class of all problems that have permutations of values as symmetries. An instance of this problem class is the graph coloring problem.

In the following, we consider finite domain integer problems, where the variables have the domain $D \subseteq \mathbb{N}$ associated. We denote with $\mathrm{Perm}(D)$ the set of all permutations of D, and with $\mathrm{Trans}(D)$ the set of all transpositions of D (i.e., those permutations, which exchange just two elements of D). With $\mathcal{S}^{\mathcal{X}}_{\mathrm{Perm}(D)}$, we denote the set of all symmetries that permute the values of the variables in $\mathcal{X}$.

DEFINITION 6. *Let $\mathcal{X}$ be the set of solution variables. The set of all value permutations of $\mathcal{X}$ is defined as the symmetry set*

$$\mathcal{S}^{\mathcal{X}}_{\mathrm{Perm}(D)} = \{S \mid \exists \pi \in \mathrm{Perm}(D) \forall \alpha : S(\alpha)(X) = \pi(\alpha(X))\}.$$

The subset of all transposition symmetries $\mathcal{S}^{\mathcal{X}}_{\mathrm{Trans}(D)}$ is defined analogously. For every symmetry $S \in \mathcal{S}^{\mathcal{X}}_{\mathrm{Perm}(D)}$, we say that π is the underlying permutation of S *if $S(\alpha)(X) = \pi(\alpha(X))$.*

Now, we consider search trees that branch over constraints of the forms $X = k$, $X = Y$ or $X \neq Y$. These are the most often used constraints in the case of permutative symmetries. The interesting part is that if a search tree branches over those constraints, then every $\mathcal{S}^{\mathcal{X}}_{\mathrm{Trans}(D)}$-excluding tree is even $\mathcal{S}^{\mathcal{X}}_{\mathrm{Perm}(D)}$-reduced. Thus, we need to exclude only the quadratically many transpositions for excluding the exponentially many permutations in this case.

THEOREM 4. *Let C_{Pr} be a problem that has $\mathcal{S}^{\mathcal{X}}_{\mathrm{Perm}(D)}$ as a symmetry group. Let t be a $\mathcal{S}^{\mathcal{X}}_{\mathrm{Trans}(D)}$-excluding search tree such that for every branch v, the constraint c is of the forms $X = d$, $X = Y$, or $X \neq Y$ with $X, Y \in \mathcal{X}$. Then, t is $\mathcal{S}^{\mathcal{X}}_{\mathrm{Perm}(D)}$-reduced.*

Proof. It is sufficient to show that for every branch v in t, we have

$$\{S(\alpha) \mid S \in \mathcal{S}^{\mathcal{X}}_{\mathrm{Perm}(D)} \wedge \alpha \in \|v_l\|\} \cap \|v_r\| \;=\; \emptyset,$$

where v_l is the left node, and v_r the right node of the branch v.

Assume that there is a valuation α and a symmetry $S \in \mathcal{S}^{\mathcal{X}}_{\mathrm{Perm}(D)}$ such that $\alpha \in \|C_{path}(v_l)\|$ and $S(\alpha) \in \|C_{path}(v_r)\|$. Let π_S be the underlying permutation of the symmetry S. Let c be the constraint introduced in the left node v_l. Then $\alpha \models c$ and $S(\alpha) \models \neg c$.

The first observation is that c cannot be of the forms $X = Y$ or $X \neq Y$. If c were of the form $X = Y$, then $\alpha(X) = \alpha(Y)$, and hence we would have $S(\alpha)(X) = \pi_S(\alpha(X)) = \pi_S(\alpha(Y)) = S(\alpha)(Y)$, which would be a contradiction to $S(\alpha) \models \neg c$. Similarly, if c were of the form $X \neq Y$, then $\alpha(X) \neq \alpha(Y)$, and hence $S(\alpha)(X) = \pi_S(\alpha(X))$ and $S(\alpha)(Y) = \pi_S(\alpha(Y))$ must be different since π_S is a bijection.

So let c be of the form $X = d$. Since c is $X = d$, $\alpha \models c$ and $S(\alpha) \models \neg c$, we know that $\alpha(X) = d \neq S(\alpha)(X)$. By the definition of the search tree t, we know that $X \in \mathcal{X}$. Hence, there is a transposition symmetry $T \in \mathcal{S}^{\mathcal{X}}_{\mathrm{Trans}(D)}$ that exchanges $\alpha(X) = d$ and $S(\alpha)(X) = d'$. We want to show that $(T^{-1} \circ S)(\alpha) = (T \circ S)(\alpha) \in \|C_{path}(v_l)\|$, which implies that $S(\alpha)$ is excluded by Lemma 1.

In the following, let π_T be the underlying transposition of the symmetry T. Note that $\pi_T(d) = d'$, $\pi_T(d') = d$ and for all d'' different from d, d', we have $\pi_T(d'') = d''$. This implies $(T \circ S)(\alpha) \models c$ since $c \equiv (X = d)$, $S(\alpha)(X) = d'$ and $\pi_T(d') = d$. Furthermore, note that $\pi_S(d) = d'$, but not necessarily $\pi_S(d') = d$. Since π_S is a bijection, we know that $\pi_S(d') \neq d'$.

For showing $(T \circ S)(\alpha) \in \|C_{path}(v_l)\|$, it is sufficient to prove that $(T \circ S)(\alpha) \in \|C_{path}(v)\|$ as follows. If $(T \circ S)(\alpha) \in \|C_{path}(v)\|$, then we know that $(T \circ S)(\alpha) \in \|C_{path}(v_l)\| \cup \|C_{path}(v_r)\|$ since $\|C_{path}(v)\|$ is equal to $\|C_{path}(v_l)\| \cup \|C_{path}(v_r)\|$. Since we branch over c at node v and $(T \circ S)(\alpha) \models c$ by our assumption, this implies immediately

$$(T \circ S)(\alpha) \in \|C_{path}(v_l)\|.$$

For showing $(T \circ S)(\alpha) \in \|C_{path}(v)\|$, we consider the following cases for $c' \in C_{path}(v)$:

1. c' is $Y = Z$. Since $S(\alpha) \models c'$, we get $S(\alpha)(Y) = S(\alpha)(Z)$. Hence, $\pi_T(S(\alpha)(Y)) = \pi_T(S(\alpha)(Z))$, which implies $(T \circ S)(\alpha)(Y) = (T \circ S)(\alpha)(Z)$ and hence $(T \circ S)(\alpha) \models Y = Z$.

2. c' is $Y \neq Z$. Since $S(\alpha) \models c'$, we get $S(\alpha(Y)) \neq S(\alpha)(Z)$. Since π_T is a bijection, we have for any values k', k'' that $\pi_T(k') = k = \pi_T(k'')$ implies $k' = k''$. Hence, $S(\alpha)(Y) \neq S(\alpha)(Z)$ implies $\pi_T(S(\alpha)(Y)) \neq \pi_T(S(\alpha)(Z))$. Hence $(T \circ S)(\alpha) \models Y \neq Z$.

3. c' is $Y = k$. Since both $\alpha \models c'$ and $S(\alpha) \models c'$, we get $\alpha(Y) = k = \pi_S(\alpha(Y)) = \pi_S(k)$ Hence $\pi_S(k) = k$, which implies that k must be different from d and d'. Hence, $\pi_T(k) = k$, which implies $(T \circ S)(\alpha) \models Y = k$.

In any case, we get $(T \circ S)(\alpha) \models c'$ for all $c' \in C_{path}(v)$. $\square$

Graph Coloring. As noted above the graph coloring is an example for a problem with value permutations as symmetries. The graph coloring problem is as follows. Given a graph $G = (V, E)$ and a set of colors. An admissible coloring c of G, is a mapping of the vertices V to the set of colors, satisfying the constraint that for all $(v_1, v_2) \in E$: $c(v_1) \neq c(v_2)$. We search for the minimal set of colors, where we still can find an admissible coloring.

In this problem colorings are considered to be symmetric, if they are just permuted in colors to each other. We compare two implementations of a solver for this problem. The first one is a naive implementation that uses a simple first fail heuristic. This implementation does not have symmetry exclusion. The second implementation with full exclusion of symmetries is just a simple extension of the first one, where we added our symmetry exclusion mechanism for transpositions (which is sufficient by Theorem 4). The results are given in Table II.

Table II. Results for some randomly generated problem instances

problem size			with sym. ex.		without sym. ex.		ratio	
verts	edges	colors	cloned	time/s	cloned	time/s	cloned	time
20	114	6	24	50	162	90	6.8	1.8
18	101	7	35	70	887	450	25.3	6.4
25	185	8	61	170	52,733	29,710	864.5	487.0
50	506	8	434	2,280	1,440,549	1,588,340	3,319.2	696.6
30	292	9	97	230	551,022	385,100	5,680.6	1674.3
35	339	9	168	370	3,428,853	2,569,200	20,409.8	6943.8

6. Conclusion and Future Work

We have introduced the first method, namely symmetry excluding search, that can exclude *arbitrary* symmetries in constraint-based search. In contrast to many other methods that can be found in the literature, the SES-method does not restrict the search strategy.

SES is based on a declarative description of the symmetries in the form of symmetric constraints, which usually can be obtained easily. We have given completeness and correctness proofs as well as the proof, that the method exclude all considered symmetries. In the case that the full symmetry group is too large, our method can handle arbitrary subsets of the full symmetry group. We have shown for the case of permutation symmetries that there it suffices to exclude the (quadratically many) transposition to exclude all (exponentially many) permutations.

SES can be implemented in any constraint-based search machinery that handles implications and allows to introduce additional, user-defined constraints during search. This holds for the most modern constraint programming systems. Since the method is very general, we plan to investigate in which other logic-based search methods our symmetry exclusion can be used. Furthermore, we intend to investigate more general conditions under that the exclusion of only a subset of the symmetries excludes the whole symmetry group (as in the case of value permutations).

References

1. Aguirre, A. S. M.: 1993, 'How to Use Symmetries in Boolean Constraint Solving'. In: F. Benhamou and A. Colmerauer (eds.): *Constraint Logic Programming, Selected Research.* The MIT Press, Chapt. 16, pp. 287–306.
2. Backofen, R.: 1998, 'Constraint Techniques for Solving the Protein Structure Prediction Problem'. In: *Proceedings of* 4^{th} *International Conference on Principle and Practive of Constraint Programming (CP'98).*
3. Benhamou, B.: 1994, 'Study of symmetry in Constraint Satisfaction Problems'. In: A. Borning (ed.): *Principles and Practice of Constraint Programming, Second International Workshop, PPCP'94,*, Vol. 874 of *Lecture Notes in Computer Science.*
4. Benhamou, B. and L. Sais: 1994, 'Tractability through Symmetries in Propositional Calculus'. *Journal of Automated Reasoning* **12**, 89–102.
5. Crawford, J. M., M. Ginsberg, E. Luks, and A. Roy: 1996, 'Symmetry Breaking Predicates for Search Problems'. In: *Proc. of the* 5^{th} *International Conference on Principles of Knowledge Representation and Reasoning (KR'96).* pp. 149–159.
6. de la Tour, T. B.: 1990, 'Minimizing the Number of Clauses by Renaming'. In: M. E. Stickel (ed.): *Proc. of the* 10^{th} *Int. Conf. on Automated Deduction (CADE90).* pp. 558–572.
7. Freuder, E.: 1991, 'Eliminating interchangeable values in constraint satisfaction problems'. In: *Proc. of AAAI'91.* pp. 227–233.
8. Gent, I. and B. Smith: 2000, 'Symmetry Breaking During Search in Constraint Programming'. In: W. Horn (ed.): *Proceedings of ECAI 2000.* pp. 599–603.
9. Schulte, C., G. Smolka, and J. Würtz: 1994, 'Encapsulated Search and Constraint Programming in Oz'. In: A. Borning (ed.): *Second Workshop on Principles and Practice of Constraint Programming.* Orcas Island, Washington, USA, pp. 134–150.
10. Smolka, G.: 1995, 'The Oz Programming Model'. In: J. van Leeuwen (ed.): *Computer Science Today*, Lecture Notes in Computer Science, vol. 1000. Berlin: Springer-Verlag, pp. 324–343.