Alcom-FT Technical Report Series
ALCOMFT-TR-02-91

Symmetry Breaking *

Torsten Fahle, Stefan Schamberger and Meinolf Sellmann
{tef,schaum,sello}@uni-paderborn.de

University of Paderborn
Department of Mathematics and Computer Science
Fürstenallee 11, D-33102 Paderborn

Abstract. Symmetries in constraint satisfaction or combinatorial optimization problems can cause considerable difficulties for exact solvers. One way to overcome the problem is to employ sophisticated models with no or at least less symmetries. However, this often requires a lot of experience from the user who is carrying out the modeling. Moreover, some problems even contain inherent symmetries that cannot be broken by remodeling. We present an approach that detects symmetric choice points during the search. It enables the user to find solutions for complex problems with minimal effort spent on modeling.

Keywords. symmetry breaking during search, graph partitioning, n-queens problem, golfer problem

1 Introduction

Symmetries can give rise to severe problems for solution algorithms as equivalent search regions are unnecessarily being explored more than just once. There are several ways of handling symmetries. One is to model the problem in such a way that no or at least less symmetries remain. This may also imply the adding of constraints which will only be satisfied by one assignment in each equivalence class. The major disadvantage of this approach is that it requires the user to have a certain level of experience, and sometimes it is even not possible to remove symmetries from a problem formulation as they are inherent to the given problem. Another way to break symmetries is to add constraints during the search for a solution. Those constraints can e.g. be derived from functions mapping single assignments to their symmetric versions.

We refer to all methods that avoid the exploration of symmetric parts of the search space as symmetry breaking strategies. However, there is of course a difference between approaches adding constraints to the model either statically or dynamically, and pruning/propagation approaches like the one we describe in this paper. Whenever a complete search is performed, all those methods have the same effect in that they do not expand symmetric choice points. Note, that for incomplete searches, the approach presented here may be used as well, but requires caution with respect to the handling of previously visited search nodes (see Section 2.2).

* This work was partly supported by the German Science Foundation (DFG) project SFB-376, and by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

1.1 State of the Art

Whereas model reformulations have been used successfully for quite a few specific problems in combinatorial optimization or constraint satisfaction, only very little research has been carried out on the topic of breaking symmetries systematically. In [7], Rothberg presents ways to remove symmetries from mixed integer problems (MIPs) by using cuts. Sherali and J.C. Smith discuss the effectiveness of adding constraints to a basic model in a number of case studies [10]. In [4], Gent and B. Smith develop a generic approach called SBDS. In every choice point, SBDS may extend the model dynamically by adding symmetry breaking constraints. For a combinatorial design problem, this approach has been shown to be efficient in combination with refined problem formulations which are used to remove symmetries already in the model [11]. As the number of symmetries in the given problem is enormous, the approach presented is not able to detect all of them and thus also gives non-unique solutions. In [6], Meseguer and Torras introduce a symmetry avoiding approach that works by adapting the search strategy.

We introduce a method that detects symmetric choice points within the search procedure. Every time the search algorithm generates a new choice point, we check if it is equivalent to or dominated by a node that has been expanded earlier. If so, the current choice point can be pruned. If not, it is processed normally. By checking whether a value assignment to a variable yields a symmetric search node, we can also use symmetries to shrink the domains of variables. However, that propagation can be very costly and thus is not suited in all cases. As the method is based on the detection of dominance relations between subtrees, we call it *Symmetry Breaking via Dominance Detection (SBDD)*.

The remaining part of the paper is structured as follows: In Section 2, we formally introduce the SBDD approach. In the Sections 3, 4, and 5, it is applied to three different examples from combinatorial optimization and combinatorial design. Numerical results are given that circumstantiate the effectiveness of the approach.

2 Breaking Symmetries

The goal of breaking symmetries is to avoid the exploration of a search space $\triangle$ that can be mapped into a previously considered part $\square$ via a symmetry function. Because if $\square$ does not contain any solution, nor does $\triangle$. And otherwise, all solutions in $\triangle$ are symmetric to those already computed during the investigation of $\square$. Thus, symmetries can be used to prune the search tree, and also to remove values from variable domains that would yield the search to a symmetric part of the search space.

Before we outline the concept more formally, first we introduce some helpful definitions.

Definition 1. *Let* $\mathcal{X} = \{x_1 \dots x_n\}$ *denote the set of variables of the model to solve,* $D(x)$ *denote the domain of variable* $x \in \mathcal{X}$*.*

The tuple $P^c = (D^c(x_1), \ldots, D^c(x_n))$ *denotes the current state in choice point* c. *We refer to the representation* P^c *as a* pattern.

Definition 2. *Let* $P^c = (D^c(x_1), \ldots, D^c(x_n))$, $P^{c'} = (D^{c'}(x_1), \ldots, D^{c'}(x_n))$.

- *We say that* $P^{c'}$ includes P^c *(*$P^c \subseteq P^{c'}$*), iff* $\forall\, x \in \mathcal{X}\ : D^c(x) \subseteq D^{c'}(x)$.
- *We set* $\mathcal{MD}^c := D^c(x_1) \times \cdots \times D^c(x_n)$.
- *Given a* symmetry mapping function $\varphi : \mathcal{MD}^c \to \mathcal{MD}^c$, *we say that* $P^{c'}$ dominates P^c *(under the symmetry* φ*), iff* $\varphi(P^c) \subseteq P^{c'}$. *Then, we write* $P^c \sqsubseteq P^{c'}$.

Property 1. Given two choice points c and c', where c' is a successor of c in the search tree. Then it holds: $P^{c'} \sqsubseteq P^c$.

To ease the presentation, in the following we assume that the partitioning of the search space is achieved by using unary branching constraints. However, the concept can be generalized by adding information on the branching constraints active in a search node to the definition of a pattern that is used to reflect the current situation in the search. Then, the definitions of symmetry mapping functions etc. have to be adapted accordingly.

The approach we suggest for the pruning of symmetric parts of the search space is based on the following ingredients:

- A database $\mathcal{T}$ that stores information on the search space already explored.
- A problem specific function $\Phi : (P^{\triangle}, P^{\square}) \longrightarrow \{false, true\}$ that yields $true$ iff the pattern $P^{\triangle}$ is dominated by $P^{\square}$ under some symmetry function φ.
- If symmetries shall also be used for propagation, a similar function is needed that, for all variables x, removes all values b from the domain of x for which $\Phi(P^{\triangle}[x = b], P^{\square}) = true$.

In every choice point, we check whether the current pattern $P^{\triangle}$ is dominated by some pattern in $\mathcal{T}$. And if so, the current node is pruned. Otherwise, we can use the function Φ for propagation. Thus, we perform Symmetry Breaking via Dominance Detection (SBDD). Figure 1 visualizes the general procedure. White nodes are still active, black nodes have been fully expanded already. Boxes represent patterns in $\mathcal{T}$, circles are patterns not or no longer contained in $\mathcal{T}$. Finally, $\triangle$ marks the current node. Originally, a pattern $\triangle$ must be checked against all fully expanded nodes (see Figure (1a)).

Obviously, it is problematic if we are to store all expanded nodes in $\mathcal{T}$. In the next Section, we describe how to handle $\mathcal{T}$ efficiently for depth first search (DFS). Then, we generalize the result to arbitrary search strategies.

2.1 Efficient Realization in a Depth First Search

The key for an efficient realization of the general SBDD concept as described above is the observation that, within a DFS, we do not need to keep the information of all previously expanded nodes in the search tree. Instead, we can merge sibling entries in $\mathcal{T}$ on backtracking, thus summarizing and compressing the information gathered.

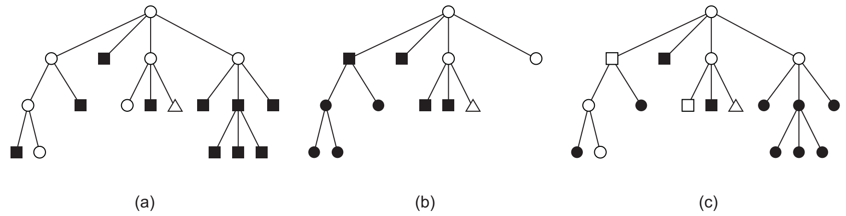

Fig. 1. The concept of SBDD

Lemma 1. *Let c be a choice point with state*

$$P^c = (D^c(x_1), \ldots, D^c(x_i), \ldots, D^c(x_n)),$$

where i is the index of the branching variable in c, and $D^c(x_i) = \{v_1, \ldots, v_l\} \subseteq D(x_i)$. Further, denote with $P^{c_k} = (D^{c_k}(x_1), \ldots, \{v_k\}, \ldots, D^{c_k}(x_n))\ \forall\, 1 \leq k \leq l$ the states of the children $c_1, \ldots, c_n$ of c. Finally, let $P^{c'}$ the state in choice point c' with $P^{c'} \sqsubseteq P^{c_k}$ for some $1 \leq k \leq l$. Then, it holds $P^{c'} \sqsubseteq P^c$.

Proof. For all $x \in \mathcal{X}$ it holds that $D^{c_k}(x) \subseteq D^c(x)$. Thus, $\varphi(P^{c'}) \subseteq P^{c_k} \subseteq P^c$.

Using Lemma 1, SBDD in combination with DFS can now be realized efficiently: We start with $\mathcal{T} = \emptyset$ and process each choice point as follows:

---

1. Check the pattern P^c of the current choice point c against all patterns in $\mathcal{T}$. If $\exists\, P \in \mathcal{T}:\ \Phi(P^c, P)$ then fail. (Alternatively encapsulate this function in a constraint and use it for propagation as well.)
2. (normal processing within the choice point)
3. on backtracking: if there are more siblings to be expanded, then add the current pattern to $\mathcal{T}$, else delete all patterns of the other siblings from $\mathcal{T}$.

To subsume, when using DFS the current pattern needs only be compared with patterns left-adjacent to the path from the root to $\triangle$ (see Figure (1b)). Notice, that step 2 refers to the normal processing of a choice point that also takes place when no additional symmetry breaking framework is utilized, including the choice of a branching variable and the exploration of the children.

The efficiency of the approach mainly depends on the number of patterns that have to be checked. The number of patterns is at most as large as the depth of the search tree times the cardinality of the largest domain.

2.2 Arbitrary Search Strategies

Referring to the discussion on the size of $\mathcal{T}$, it seems to be impractical to combine SBDD with search strategies other than DFS, because the number of previously expanded nodes, and thus the size of $\mathcal{T}$ may be enormous. Or, the method becomes ineffective, because many nodes are closed late, as it is the case for breadth first search, for instance.

Nevertheless, with a slight modification, it is possible to cope with general search strategies. Let c be the current choice point, and P^c the corresponding pattern. The idea now is to check whether a symmetry function maps P^c to a pattern of a choice point c' that would have been processed *before c if DFS would have been applied* on a statical variable ordering (see Figure (1c)). If so, c is rejected, otherwise we proceed normally. Like that, we prune the tree because we detect that the work has either been carried out already or because we decide to do it later. Notice, that the current path in the search tree contains all information necessary to identify the patterns that are relevant for checking. The assumption of a statical variable ordering defines an ordering of all choice points. The approach rejects the current choice point iff a dominating pattern exists left of it in a DFS tree, i.e. iff the current choice point is *greater* than one that already has been or will be explored later. As an exhaustive search eventually will consider the leftmost nodes as well, we can be sure not to miss a solution.

Notice, that the search strategy is slightly affected by this procedure, because the exploration of choice points can be postponed by the symmetry breaking algorithm. However, one might expect a reasonable search strategy to rate symmetric parts of the search tree as equally important. In that case, the expanding of the current choice point is only postponed formally, but in fact is carried out next in a symmetric version.

After having outlined the general approach, in the following Sections we apply it to three different applications in the field of combinatorial optimization and constraint satisfaction.

3 Graph Partitioning

The first application of the method described in Section 2 is the graph bipartitioning problem. Given an undirected graph $G = (V, E)$, the graph bipartitioning problem asks for a set $V' \subset V$ such that the number of nodes in V' and $V \setminus V'$, differs at most by one, and the number of edges between both sets is minimal. This optimal number is often referred to as the *bisection width* of the graph. Graph bipartitioning is known to be NP-hard, exact solutions can only be computed for small graphs, i.e. $|V| < 200$. Interestingly, graph bipartitioning alone already induces a symmetry as the sets V' and $V \setminus V'$ can be exchanged.

An obvious symmetry breaking strategy in this case is the assignment of node 0 to set V'. Unfortunately, if graph G itself introduce symmetries, such an assignment does not break the resulting combined symmetries.

In parallel computing, connection networks are typically nicely structured and their symmetries are known. Graphs of the hypercube family have been studied intensively (see [1, 5]). One popular network is the so-called *de Bruijn network* which is defined as follows:

Definition 3 (de Bruijn Network DB(k)). *The* de Bruijn Network of di-*mension k is a directed graph $DB(k) = (V_k, E_k)$. The edge set can be described best by associating the nodes with their corresponding binary representation, i.e. $V_k = \{(b_0 \dots b_{k-1}) \in \{0,1\}^k\}$. Then,*

$$E_k = \{(b\alpha, \alpha b),\ (b\alpha, \alpha\overline{b}) \mid \alpha \in \{0,1\}^{k-1},\ b \in \{0,1\}\}$$

where $\overline{b}$ denotes inverting bit b, i.e. $\overline{b} = 1 - b$.

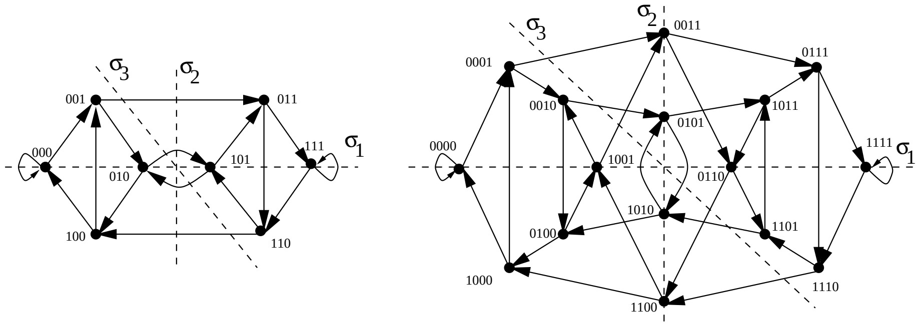

Fig. 2. de Bruijn networks of dimension 3 (left) and 4 (right). A node is marked by the binary string corresponding to its number. The dashed lines mark the symmetries of the de Bruijn network.

DB(k) contains 2^k nodes, each having degree 4, and 2^{k+1} edges. Furthermore, DB(k) contains 3 symmetries described by the following automorphisms:

$$\begin{aligned}
\sigma_1 &: V \to V,\ (b_0, b_1, \dots, b_{k-1}) \mapsto (b_{k-1}, b_{k-2}, \dots, b_0) \\
\sigma_2 &: V \to V,\ (b_0, b_1, \dots, b_{k-1}) \mapsto (\overline{b_0}, \overline{b_1}, \dots, \overline{b_{k-1}}) \\
\sigma_3 &: V \to V,\ (b_0, b_1, \dots, b_{k-1}) \mapsto (\overline{b_{k-1}}, \overline{b_{k-2}}, \dots, \overline{b_0})
\end{aligned}$$

Symmetries σ_1, σ_2 and σ_3 are visualized in Figure 2, where DB(3) and DB(4) are shown.

In the following, for the graph partitioning problem we will interpret any directed arc of DB(k) as an undirected edge.

3.1 Bisection Width of the de Bruijn Graph

It can be shown that the bisection width of DB(k) is $\Theta(\frac{2^k}{k})$, but there are only few results for concrete graphs. In [3], an optimal bisection width of 30 for DB(7) has been computed. At the time that paper was written, the algorithm based on LP bounds ran for about two weeks. To our knowledge, no exact bisection widths for bigger de Bruijn networks were known at that time.

Recently, Sensen [9] improved the well known bound based on clique embeddings (equivalent to 1-1 multi-commodity flows) by introducing variable multicommodity flows. Using interior point methods for the resulting linear programs, he was able to prove an exact bisection width of 54 for DB(8). The symmetry detection routine described in Section 2 was used to avoid the consideration of symmetric parts of the search space. We refer to [9] for details on the overall approach. Here, we concentrate on the symmetry breaking. We use this example to show an easy application of SBDD rather than to underline its efficiency. For comparisons with SBDS we refer to Sections 4 and 5.

3.2 Symmetry Breaking for Graph Partitioning

When bipartitioning de Bruijn networks, seven symmetries have to be encoded in Φ. They stem from the three automorphisms of the network itself, the exchange of V' against $V \setminus V'$ and the combination of these symmetries.

For the graph bipartitioning problem, a pattern is implemented as an n-tuple $p \in \{0, 1, *\}^n$. $p_i = 0$ ($p_i = 1$) means, that node $i \in V'$ ($i \in V \setminus V'$). $p_i = *$ means, that node i has not been assigned yet. The symmetry functions $\varphi_1, \ldots, \varphi_7$ permute the nodes according to σ_1, σ_2 or σ_3 and/or invert the entries. A pattern $P^{\triangle}$ is dominated by $P^{\square}$ iff there is a symmetry function φ_k, $1 \leq k \leq 7$ such that for all $0 \leq i < n$ it holds $\varphi_k(P^{\square})_i = *$ or $P_i^{\triangle} = \varphi_k(P^{\square})_i$.

It is also possible to use pattern information for propagation. Assume that there is a symmetry function φ_k and an index j, $0 \leq j < n$, such that $\varphi_k(P^{\square})_i = *$ or $P_i^{\triangle} = \varphi_k(P^{\square})_i \; \forall 1 \leq i < n, i \neq j$ and $p_j^{\triangle} = *$. Let $\varphi_k(p^{\square})_j = 0$ (or $\varphi_k(p^{\square})_j = 1$). Then we can force that node j is in $V \setminus V'$ (or V', respectively).

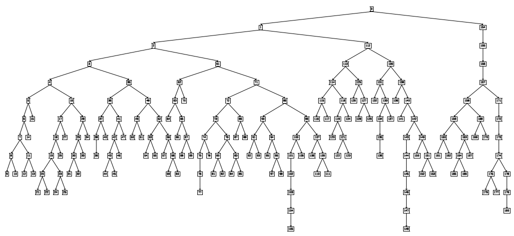

Fig. 3. The search tree for DB(8) bipartitioning when breaking all possible symmetries. Notice, that chains of choice points with only one successor result from detecting symmetric parts that are not explored.

Figure 3 and 4 show the different branching trees resulting from a computation of DB(8) with and without breaking symmetries. As expected, the search tree is much smaller in the first case. Notice, that huge parts of the solution space are cut off by lower bound information. Thus, many symmetric subtrees are pruned early, thereby diminishing the effect of symmetry breaking. However, since in this approach the effort per choice point is very high due to expensive

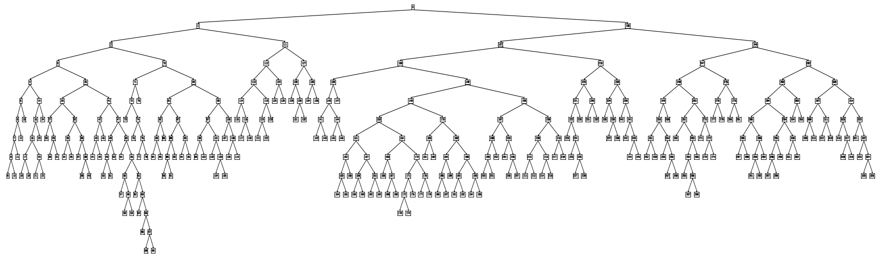

Fig. 4. The search tree for DB(8) bipartitioning without breaking any symmetries.

bound computations ($\approx$ 14 minutes per choice point), any reduction of the tree size reduces the overall cpu time consumption significantly. Thus, for the computation of the bisection width of DB(8), the breaking of symmetries was able to reduce the running time by roughly 2 days, whereby the remaining overall computation time then took 37.5 hours.

4 The Golfer Problem

We also applied SBDD to find solutions for the *Golfer Problem* (Problem 10 in CSPLib [2]) which is:

> *32 golfers want to play in 8 groups of 4 each week, in such way that any two golfers play in the same group at most once. How many weeks can they do this for?* [1]

This problem can be generalized by parameterizing it to w weeks and g groups of s players each, written as g-s-w from now on. In case of $(s-1)w = gs-1$, we achieve a specification where every player must play with every other exactly once. This problem is also known as the *Schoolgirl Problem* (see Section 4.3).

4.1 Symmetries in the Golfer Problem

Obviously, there is a lot of symmetry in the problem. First, players can be placed at any position within a group (φ_P), groups can be exchanged within their week (φ_G), and also the weeks can be ordered arbitrarily (φ_W). Furthermore, the players can be permuted (φ_X).

Following the idea that symmetry detection should also work well in combination with simple models, we have chosen a straightforward one that can be implemented with little effort using the ILOG Solver environment. The groups are modeled as sets of players with the cardinality of each set fixed to s. Each

[1] In the original problem it is clear that the golfers cannot play for more than 10 weeks. On the other hand, a solution for 5 weeks can be found easily without backtracking by always choosing the first possible player for a group in each week.

week contains g such sets, and the full pattern covers w weeks. To shrink the search space, we fix all players in the first week in increasing order. Additionally, we insert the first s players into the first s groups for all weeks thereafter. Finally, the first group of the second week is filled with the smallest players possible. All these assignments can be made without increasing the complexity of the model nor losing unique solutions.

4.2 Breaking Symmetries

By using set variables for each group, the model does not contain symmetry φ_P anymore. To detect the domination of patterns with respect to the other symmetries, we describe three symmetry detection functions Φ_G, $\Phi_{W,G}$ and $\Phi_{W,G,X}$, that are used during the search. Function $\Phi_{W,G}$ includes checks performed by Φ_G, and $\Phi_{W,G,X}$ includes those done by $\Phi_{W,G}$.

Φ_G Given two week indices $1 \leq i, j \leq w$, Φ_G is used to check if a week i of pattern $P^{\square}$ dominates week j of pattern $P^{\triangle}$ with respect to symmetry φ_G. This is done by checking whether all groups of week i of pattern $P^{\square}$ can be mapped to groups in week j of pattern $P^{\triangle}$. In the example shown in Figure 5, week 2 of pattern $P^{\square}$ cannot be mapped to week 1 of pattern $P^{\triangle}$, because players 1 ans 2 are in the same group in pattern $P^{\triangle}$, but are in different groups in pattern $P^{\square}$. A similar reasoning for players 2 and 3 prevents mapping week 3 of pattern $P^{\square}$ to week 2 of pattern $P^{\triangle}$. However, week 2 of pattern $P^{\square}$ can be mapped to week 2 of pattern $P^{\triangle}$, as the latter is just a specialization of the first.

$\Phi_{W,G}$ To break symmetries φ_W and φ_G, function $\Phi_{W,G}$ constructs a bipartite graph G containing a node for each week of $P^{\square}$ and $P^{\triangle}$. An edge is inserted, iff a week of $P^{\square}$ dominates a week of $P^{\triangle}$, which is determined using φ_G. If G contains a matching of cardinality w, $P^{\square}$ dominates $P^{\triangle}$. Again, Figure 5 shows an example.

$\Phi_{W,G,X}$ Incorporating also the last symmetry φ_X results in a huge computational effort, as $\Phi_{W,G}$ has to be applied for $(g \cdot s)!$ different permutations. To reduce the cost of this check, we use the fact that the first week of a pattern is always complete due to the fixed entries. Since it has to be matched to some other week, "only" $w \cdot (s!)^g \cdot g!$ possibilities are left. However, the test remains expensive. Therefore, we tried some variations reducing the frequency when $\Phi_{W,G,X}$ is applied. A parameter q can be set to restrict full symmetry checks to every q-th level of the search tree. Optionally, it can be limited to be performed on full patterns, i.e. leaves, only, which is the default.

4.3 Numerical Results

The model described has been implemented in ILOG Solver 5.0 and run for different configurations on a Sun Enterprise 450 (400 MHz UltraSparc-II). Tables 1 and 2 show the results of the experiments. Apart from the time (in seconds) needed to find the first solution (t_1) and the time to find all solutions (t_{all}), the number of calls to the symmetry detection functions $\Phi_{W,G}$ and $\Phi_{W,G,X}$ is given.

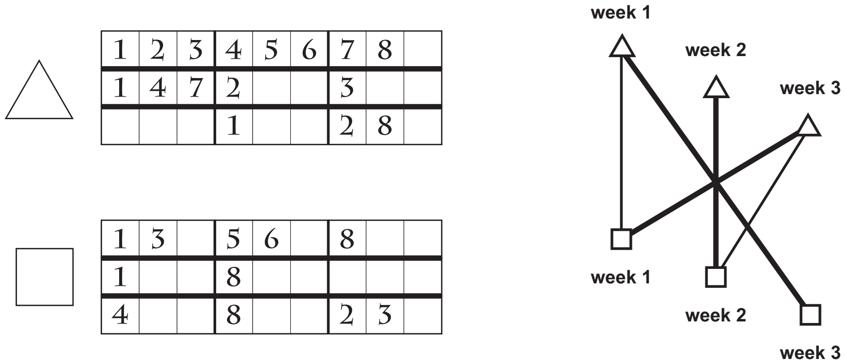

Fig. 5. The left hand side shows two patterns $P^{\triangle}$ and $P^{\square}$. Each pattern consists of three weeks (horizontal) of three groups of three players. Unbounded variables are left empty. On the right hand side, the corresponding bipartite graph is shown, containing a node for each week of both patterns. Since a matching of cardinality 3 exists (bold edges), $P^{\triangle}$ is dominated by $P^{\square}$.

In the *sym*-section, $\Phi_{W,G}$ is applied to check for symmetries φ_W and φ_G in each node of the search tree. Since symmetries φ_X are not detected, there are many non-unique solutions found. In the *nosym*-section, $\Phi_{W,G}$ is also applied in every node of the search tree, and additionally $\Phi_{W,G,X}$ is applied in leaves preventing symmetric solutions from being written out. The Tables continue with the number of detected symmetries (*symmetries*), the number of choice points (*cp*), and the number of fails. Since we are using a very simple model for the problem, an approach that does not prevent the exploration of symmetric parts of the search tree is not applicable in practice as shown in [11]. Therefore, a comparison with such an approach is left out here.

problem	*solutions*	t_1	t_{all}	$\Phi_{W,G}$	$\Phi_{W,G,X}$	*symmetries*	*cp*	*fails*
				sym				
4-3-2	48	0.00	0.03	226	0	0	195	148
4-3-3	2688	0.02	6.09	99454	0	0	28299	25612
4-3-4	1968	0.05	26.70	382120	0	2808	94845	92878
4-3-5	0	0.00	36.34	412456	0	3120	100389	200390
				nosym				
4-3-2	1	0.00	0.04	226	47	47	195	194
4-3-3	4	0.01	10.00	99454	2687	2684	28299	28296
4-3-4	3	0.04	29.18	382120	1967	4773	94845	94843
4-3-5	0	0.00	36.28	412456	0	3120	100389	200390

Table 1. Results of the golfer 4-3-X problem.

problem	*solutions*	t_1	t_{all}	$\Phi_{W,G}$	$\Phi_{W,G,X}$	*symmetries*	*cp*	*fails*
				sym				
4-4-2	216	0.00	0.09	735	0	0	555	340
4-4-3	5184	0.01	8.71	74175	0	0	43755	38572
4-4-4	1296	0.01	20.53	140595	0	1296	82635	81340
4-4-5	432	0.01	25.90	132531	0	2160	75723	75292
4-4-6	0	0.00	30.76	114027	0	0	72267	72268
				nosym				
4-4-2	1	0.01	0.17	735	215	215	555	555
4-4-3	2	0.01	136.31	74175	5183	5182	43755	43754
4-4-4	1	0.01	22.09	140595	1295	2591	82635	82634
4-4-5	1	0.02	26.51	132531	431	2591	75723	75723
4-4-6	0	0.00	30.71	114027	0	0	72267	72268

Table 2. Results of the golfer 4-4-X problem.

Since invoking the symmetry detection function $\Phi_{W,G,X}$ is computationally very expensive, applying it in every search node does not improve the overall runtime, although the number of choice points is reduced. Thus, there is a trade-off between the reduction of choice points and the effort spent for the detection of symmetries. We have tested a scheme that applies $\Phi_{W,G,X}$ not only in leaves but also performs additional checks for all symmetries in every node in the q-th level of the search tree. Table 3 shows that invoking $\Phi_{W,G,X}$ too often rather increases the overall runtime, but applying it too rarely (e.g., only in leaves) is not the best choice, either. For the 4-4-4 problem, an invocation in about every 8-th level has shown to be the best. Similar observations have been made for other instances as well. Table 4 shows the improved running times for the 4-4-X problem.

level of $\Phi_{W,G,X}$	*solutions*	t_1	t_{all}	$\Phi_{W,G}$	$\Phi_{W,G,X}$	*symmetries*	*cp*	*fails*
				nosym				
1	1	0.01	698.51	0	26	18	82	82
2	1	0.02	271.35	29	27	24	123	123
4	1	0.02	101.26	156	79	79	339	339
8	1	0.01	14.51	5292	1296	1296	4730	4730
leaves	1	0.01	22.09	140595	1295	2591	82635	82634

Table 3. Results of the golfer 4-4-4 problem performing additional checks for symmetry φ_X in search tree nodes of every q-th depth.

SBDS versus SBDD In [11], an SBDS approach is developed for the golfer problem. As has been mentioned before, to break symmetries SBDS inserts additional constraints to the model during the search, and hands them over to the

problem	*solutions*	t_1	t_{all}	$\Phi_{W,G}$	$\Phi_{W,G,X}$	*symmetries*	*cp*	*fails*
			nosym, level of $\Phi_{W,G,X} = 8$					
4-4-2	1	0.00	0.17	735	215	215	555	555
4-4-3	2	0.01	134.10	5283	1298	1297	6492	2891
4-4-4	1	0.01	14.51	5292	1296	1296	4730	4730
4-4-5	1	0.02	15.68	5291	1295	1296	4722	4722
4-4-6	0	0.00	17.16	5290	1295	1295	4714	4715

Table 4. Improved results of the golfer 4-4-X performing additional checks for symmetry φ_X in search tree nodes of every 8-th depth.

solver. Even in combination with complex models, due to the large amount of symmetries in the golfer problem, the approach presented is not able to add all constraints necessary to break all symmetries. However, SBDS allows ro reduce the number of search nodes significantly.

When using SBDD for the golfer problem, it is possible to find unique solutions only, even in combination with a very simple model. Obviously, the performance of the approach presented here can be further improved by using more sophisticated problem formulations. However, the focus in this paper was not to develop a most efficient approach for the golfer problem, but to present a method for symmetry breaking that can be used efficiently also by inexperienced users and in combination with simple models.

We are currently working on an approach combining SBDD and a refined model for the golfer problem, that is able to solve the so called schoolgirl problem.

In 1850, Thomas Kirkman stated the following problem, which in fact is equivalent to the golfer 5-3-7 problem:

> *How can 15 schoolgirls walk in 5 rows of 3 each for 7 days so that no girl walks with any other girl in the same triplet more than once?*

Preliminary experimentation shows that this approach is able to compute all 7 unique solutions to the schoolgirl problem in less than 2 hours.

5 The n-Queens Problem

Finally, we consider the classical n-queens problem. It consists of placing n queens on a $n \times n$ chessboard such that no two queens can capture each other. That is, no two queens are allowed to be placed on the same row, the same column, or the same diagonal.

Nowadays it is possible to find *one* solution using CP for 1 000-queens in a few seconds. Asking for *all non-symmetric* solutions of n-queens requires some more effort. In the following, we describe the SBDS approach of Gent and Smith [4] on the n-queens problem and compare it to SBDD.

5.1 Breaking Symmetries in n-Queens

It is easy to see that the n-queens problem incorporates seven symmetries, namely reflections in the horizontal and vertical axis, reflections in the main diagonals, and rotations through $90°, 180°, 270°$.

SBDS In [4], SBDS is introduced first and tested on a variety of problems. The approach is general and compatible with different search strategies. A user of the concept only needs to provide symmetry functions mapping a single assignment to its symetric version.

In a choice point where we assign, $x = v$ on the left and $x \neq v$ on the right branch, SBDS adds all constraints that are necessary to prevent the solver from exploring a subtree symmetric to an already investigated one. By keeping track of all already broken symmetries, only necessary constraints are posted, thus keeping the overhead small.

SBDD For the n-queens problem, a pattern p is an n-tuple where p_i is the column number in which the queen covering row i is placed, or, in case the position of the queen in row i has not been set yet, $p_i = *$. E.g., the pattern corresponding to the first chessboard in Figure 6 is $p = (0, 4, 1, 5, 2, 6, 3)$.

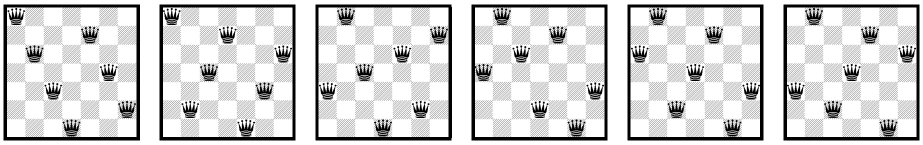

Fig. 6. Six out of forty solutions of 7-queens are unique

5.2 Experimental Evaluation

For our experiments, we used the following standard model for n-queens:

- Each row $i = 0, \ldots, n-1$ is represented by an integer variable x_i. Assigning $x_i = j$ corresponds to placing a queen in row i and column j.
- Additional integer variables y_i and w_i, $i = 0, \ldots, n-1$, are used to check the diagonals of the chessboard. We post the constraints $y_i = x_i + i$, $w_i = x_i - i$.
- The domains are $x \in \{0, \ldots, n-1\}$, $y \in \{0, \ldots, 2n\}$, $w \in \{-n, \ldots, n\}$.
- AllDiff constraints on x, y, and w ensure that no two queens can capture each other.

In contrast to the algorithm we developed for the golfer problems, here we use symmetry also for propagation. A constraint is posted to the model that keeps track of the current situation in the search. As propagation turned out to be rather expensive, we limited the number of calls to the propagation routine to one.

We also implemented a version of SBDS and tested it on the model described above. Both codes were running on the same Sun Enterprise as the program for the golfers problem in Section 4.

Table 5 compares the number of solutions, the number of fails, and the computation time for calculating *all* solutions (*sym*), calculating only *unique* solutions via SBDS, and *unique* solutions using SBDD, respectively. We omit the number of solutions for *SBDD* as it is identical to *SBDS*. The results given for SBDS are similar to those given in [4]. Only the number of fails slightly differs, which we expect to be caused by small variations in the implementation and the different CP engines used (Solver 4.3 vs. Solver 5.0).

Obviously, SBDD does not perform as well as SBDS on the n-queens problem. The reason for this is, that the number of symmetries is rather small (compared to the golfer problem), which makes the application of different additional symmetry breaking constraints on backtracking favorable.

		sym			*SBDS*		*SBDD*	
n	*solutions*	*fails*	*time*	*solutions*	*fails*	*time*	*fails*	*time*
4	2	4	0.01	1	3	0.00	6	0.00
5	10	4	0.00	2	4	0.00	13	0.00
6	4	35	0.01	1	11	0.02	31	0.01
7	40	69	0.02	6	19	0.01	56	0.02
8	92	289	0.04	12	63	0.01	130	0.03
9	352	1111	0.16	46	216	0.04	397	0.08
10	724	5072	0.57	92	851	0.13	1464	0.29
11	2680	22124	2.49	341	3808	0.53	5991	1.26
12	14200	103956	11.88	1787	17673	2.52	27731	6.27
13	73712	531401	61.56	9233	89534	12.55	140348	33.11
14	365596	2932626	337.00	45752	483214	69.62	746530	189.07
15	2279184	16920396	1946.07	285053	2784876	403.16	4391877	1213.36
16	14772512	105445065	12154.60	1846955	17277508	2608.51	27153758	7463.62

Table 5. Solving n-queens without breaking symmetries (*sym*), with breaking symmetries via *SBDS*, and by avoiding them via *SBDD*. Computing times are given in seconds.

6 Conclusion

We have suggested an approach for breaking symmetries that is based on the detection of dominance relations between choice points. The method is generally applicable and works in combination with all exhaustive search strategies while it may overrule strategies other than DFS. Moreover, it removes symmetric parts of the search tree efficiently in combination with any model. Thus, it can also be used easily by inexperienced users on straightforward models that do not break symmetries themselves.

The ease of use mainly results from the fact that it is only necessary to define the pattern structure and a function that checks if one pattern dominates another. This algorithmic approach allows somewhat more flexibility than a model that breaks symmetries itself, as has been demonstrated for the golfer problem when adapting the frequency of certain symmetry considerations.

The method has shown to be easily applicable without causing a big implementation overhead on three very different applications from combinatorial optimization and constraint satisfaction. Moreover, it worked efficiently even in combination with very easy models and also on highly symmetric problems.

As a disadvantage, the use of patterns appears to be less efficient on transparent and – with respect to symmetry considerations – manageable problems such as the n-queens problem. There, the dynamic adding of constraints in an SBDS fashion is clearly favorable.

Acknowledgment

We would like to thank Barbara Smith, Warwick Harway, and three anonymous referees for helpful comments.

References

1. J.C. Bermond and C. Peyrat. De bruijn and kautz networks: a competitor for the hypercube? *Proc. of the 1st Europ. Workshop on Hypercubes and Distributed Computers*, pp. 279–293, North-Holland, 1989.
2. *CSPLib: a problem library for constraints*, maintained by I.P. Gent, T. Walsh, B. Selman, `http://www-users.cs.york.ac.uk/~tw/csplib/`
3. R. Feldmann, B. Monien, P. Mysliwietz, and S. Tschöke. A Better Upper Bound on the Bisection Width of de Bruijn Networks. Tech. Report University of Paderborn. short version: *Proc. of STACS'97*, Springer LNCS 1200:511-522, 1997.
4. I.P. Gent and B. Smith. Symmetry Breaking During Search in Constraint Programming. *Proc. of ECAI'2000*, Berlin, pp. 599–603, 2000.
5. F.T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann Publishers, 1992.
6. P. Meseguer and C. Torras. Exploiting symmetries within constraint satisfaction search. *Artificial Intelligence*, 129 (1–2) pp. 133–163, 2001.
7. E. Rothberg. Using Cuts to Remove Symmetry. *ISMP'00*, Atlanta, 2000.
8. ILOG. Ilog Solver. Reference manual and user manual. V5.0, ILOG, 2000.
9. N. Sensen. Lower Bounds and Exact Algorithms for the Graph Partitioning Problem using Multicommodity Flows. *Europ. Symp. on Algorithms, ESA'01*, 2001.
10. H.D. Sherali and J. Cole Smith. Improving Discrete Model Representation Via Symmetry Considerations. *ISMP'00*, Atlanta, 2000.
11. B. Smith. Reducing Symmetry in a Combinatorial Design Problem. *Proc. of CPAIOR'01*, Wye/UK, pp. 351–360, April 2001.