# Dynamic Symmetry Breaking Restarted

Daniel S. Heller and Meinolf Sellmann

Brown University
Department of Computer Science
115 Waterman Street, P.O. Box 1910
Providence, RI 02912
{dheller, sello}@cs.brown.edu

Symmetry breaking by dominance detection (SBDD) [4,6,1,12], has proven to excel on problems that contain large symmetry groups. The core task of SBDD is the dominance detection algorithm. The first automated dominance detection algorithms were based on group theory [7], while the first provably polynomial-time dominance checkers for specific types of value symmetry were devised in [15]. This work was later extended to tackle any kind of value symmetry in polynomial time [13]. Based on these results, for specific "piecewise" symmetric problems, [14] showed that breaking variable and value symmetry can be broken simultaneously in polynomial time. The method was named structural symmetry breaking (SSB) and is based on the structural abstraction of a given partial assignment of values to variables.

Compared with other symmetry breaking techniques, the big advantage of dynamic symmetry breaking is that it can accommodate dynamic variable and value orderings. Dynamic orderings have been shown to be vastly superior to static orderings in many different types of constraint satisfaction problems. However, robust heuristics for the selection of variables and values are hard to come by. For the task of variable selection, a bias towards variables with smaller domains often works comparably well, but there always remains a fair probability that we hit instances on which a solver gets trapped in extremely long runs. Particularly, heavy-tailed runtime distributions have been reported [9]. One way to circumvent this problematic situation is to randomize the solver and to restart the search when a run takes too long [10]. We show how symmetry no-goods can be used in restarted methods and introduce practical enhancements of SSB for its application in restarted solvers.

## 1 Symmetry No-Goods and Restarts

Unfortunately, due to space restrictions, we cannot review SSB here. For definitions and a detailed description of the method, we must therefore refer the reader to [14]. SSB (as a special form of SBDD) stores the most general previously fully expanded search nodes as a list of no-goods. In contrast to ordinary no-goods, an SBDD no-good implicitly represents a whole set of no-goods (namely the set of all its symmetric variants), and it is the algorithmic task of the dominance checker to see whether this set contains a no-good that is relevant with respect to the current search node.

What is interesting to note is that SBDD no-goods also keep a record of those parts of the search space that have already been searched through. In that regard, it is of interest to store them (or at least the most powerful ones) between restarts. There is a trade-off, however: No-goods will only be beneficial if the method that prevents us from exploring the same part of the search space more than once does not impose a greater

F. Benhamou (Ed.): CP 2006, LNCS 4204, pp. 721–725, 2006.

computational cost than what the exploration would cost anyway. One simple thing that we can do is to remove those no-goods from the list that have very little impact anyway because they only represent a small part of the search space. This is an idea that is commonly used in SAT, too. However, for symmetry-nogoods we can do more.

## 2 Delayed Ancestor-Based Filtering

We introduce delayed symmetry filtering. The core idea here is to apply an inexpensive inference mechanism that quickly identifies which no-goods cannot possibly cause effective symmetry-based filtering at a given search node. Like this, we hope to save many of the expensive calls to SSB-based domain filtering. Note that no-goods are only used for ancestor-based filtering, which is why this idea will only be applied for this type of symmetry filtering. We will discuss special methods to improve the performance of sibling-based filtering in Section 3.

### 2.1 A Simple Pretest

We start by introducing a very simple pretest before a full-fledged ancestor-based filtering call is being made. What we need to identify are simple conditions under which a previously expanded node $\alpha$ (as usual, $\alpha$ is identified with the partial assignment that leads to the node) cannot "almost dominate" the current search node $\beta$. To make this more precise, with "almost" we mean to say that one more assignment to $\beta$ could result to a successful dominance relation with $\alpha$, which is a necessary condition for SSB filtering to have any effect.

First, we observe that $\beta$ must contain at least as many variable assignments as $\alpha$ minus 1. This is a trivial condition which is always true in a one-shot tree search as all no-goods stored by SBDD were taken from search nodes at the same or lower depth as that of the current node. However, for no-goods stored in earlier restarts, this test can quickly reveal that ancestor-based filtering will not be effective.

Only if the above condition holds, we perform one more test before applying the full-fledged filtering call: we can look a little bit closer at the two assignments $\alpha$ and $\beta$ and see whether $\alpha$ is close to dominating $\beta$. Before looking at each value individually, determining all their signatures and which ones dominates which, we can do the same on the level of value classes: For each value partition $Q_l$, we determine how many variables in each value partition are taking a value in it under assignment $\gamma$, thus computing a signature for each partition of mutually symmetric values: $sign_\gamma(Q_l) := (|\{X \in P_k \mid \gamma(v) \in Q_l\}|)_{k \leq r}$ for all $1 \leq l \leq s$.

**Lemma 1.** *Given assignments $\alpha$ and $\beta$ such that $\alpha$ dominates $\beta$, we have that, for all $1 \leq l \leq s$, it holds that $sign_\alpha(Q_l) \leq sign_\beta(Q_l)$ (whereby with $\leq$ we denote the component-wise comparison of the two tuples).*

*Proof.* Let $l \in \{1, \ldots, s\}$. Since $\alpha$ dominates $\beta$, we have that, for all $v \in Q_l$, it holds $sign_\alpha(v) \leq sign_\beta(w)$ for some value $w \in Q_l$ that is the unique matching partner of $v$. Consequently, it holds that $sign_\alpha(Q_l) = \sum_{v \in Q_l} sign_\alpha(v) \leq \sum_{w \in Q_l} sign_\beta(w) = sign_\beta(Q_l)$. □

Thus, SSB filtering can only be effective if the inequality holds for all but at most one value partition $l$, and if for that partition we have that $sign_\alpha(Q_l) \leq sign_\beta(Q_l) + e_k$,