

Model Restarts for Structural Symmetry Breaking*

Daniel Heller, Aurojit Panda, Meinolf Sellmann, Justin Yip

Brown University, Department of Computer Science
115 Waterman Street, P.O. Box 1910, Providence, RI 02912

Abstract

Based on structural abstractions of the problem, structural symmetry breaking (SSB) can be used to break symmetry dynamically during search, or statically by adding symmetry breaking constraints to the problem. While dynamic symmetry breaking incurs larger costs per choice point, it is able to accommodate dynamic search orderings. Static symmetry breaking, on the other hand, has low costs per choice point, but only works efficiently in combination with static search orderings, which in turn cause high variance in runtime for different instances. By introducing "model restarts," we combine the benefits of lean and fast static symmetry breaking with those of randomized dynamic search orderings. The first empirical comparison of dynamic and static SSB shows that static SSB with model restarts works robustly and an order of magnitude faster than other methods.

1 Introduction

Symmetry breaking has received considerable and increasing interest in past years. It is widely accepted now that symmetries can cause significant problems to systematic solvers that unnecessarily explore redundant parts of the search-tree many times over. Methods to avoid this undesirable behavior range from adapting ordering heuristics [Brown et al., 1988; Backofen & Will, 1999], adding static constraints to the problem [Crawford et al., 1996; Flener et al., 2002], adding constraints during search [Gent & Smith, 2000], and filtering values based on a symmetric dominance analysis when comparing the current search-node with those that were previously expanded [Fahle et al., 2001; Focacci & Milano, 2001; Puget, 2002].

Especially the latter technique, known as symmetry breaking by dominance detection (SBDD), has proven to excel on problems that contain large symmetry groups. The core task of SBDD is dominance detection. The first automated dominance detection algorithms were based on group theory [Gent et al., 2003], while the first provably polynomial-time dominance checkers for specific types of value symmetry were devised in [Van Hentenryck et al., 2003]. This work was

later extended to tackle arbitrary types of value symmetry in polynomial time [Roney-Dougal et al., 2004]. Based on these results, for specific "piecewise" symmetric problems, [Sellmann & Van Hentenryck, 2005] showed that symmetric subtrees resulting from variable and value symmetry simultaneously can be eliminated from the search-tree 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. Interestingly, the structural abstractions introduced to tackle piecewise symmetries dynamically could also be exploited to devise a set of static symmetry-breaking constraints that break all piecewise variable and value symmetry [Flener et al., 2006].

Compared with other symmetry-breaking techniques, the big advantage of dynamic symmetry breaking is that it can accommodate dynamic search orderings without running an increased risk of thrashing. Dynamic orderings have often been shown to vastly outperform static orderings in many different types of constraint satisfaction problems. However, when adding static symmetry-breaking constraints that are not aligned with the variable and value orderings, it is entirely possible that we dismiss perfectly good solutions just because they are not the ones that are favored by the static constraints, which (ideally) leave only one representative solution in each equivalence class of solutions. To address this problem, Puget suggested an elegant semi-static symmetry breaking method that provably does not remove the first solution found by a dynamic search-method [Puget, 2006]. It is not clear how this method can be generalized, though, and for the case of piecewise variable and value symmetry, no method with similar properties is known yet. On the other hand, static methods are generally easy to use, enjoy a low overhead per choice point, and exhibit an anticipatory character that emerges from filtering symmetry-breaking constraints in combination with constraints in the problem.

In this paper, for the first time ever we compare static and dynamic SSB in practice. We show that static SSB works much faster than dynamic SSB. However, this gain comes at a cost: Static SSB introduces a huge variance in runtime as static symmetry breaking constraints may clash with dynamic search orderings. Using static search orderings, on the other hand, can also cause large variances in runtime as they are not equally well suited for different problem instances. To avoid this core problem of static symmetry breaking, we introduce the idea of "model restarts." We show how they allow us to efficiently combine static symmetry breaking with semi-

*This work was supported by the National Science Foundation through the Career: Cornflower Project (award number 0644113).

dynamic search-orderings. The method is very simple to use and we show that model restarts greatly improve the robustness of static symmetry breaking.

In particular, in the following sections, we give some basic definitions, and review dynamic and static structural symmetry breaking in the following two sections. Then, we introduce model restarts. Finally, we show the results of our experiments.

2 Basic Definitions

First, let us review the basic definitions of constraint programs and piecewise symmetry.

Definition 1. • A Constraint Satisfaction Problem (CSP) is a tuple (Z, V, D, C) where $Z = \{X_1, \dots, X_n\}$ is a finite set of variables, $V = \{v_1, \dots, v_m\}$ is a set of values, $D = \{D_1, \dots, D_n\}$ is a set of finite domains where each $D_i \subseteq V$ is the set of possible instantiations to variable X_i , and $C = \{c_1, \dots, c_p\}$ is a finite set of constraints where each $c_i \in C$ is defined on a subset of the variables in Z and specifying their valid combinations.

- Given a CSP, an assignment A is a set of pairs $(X, v) \in Z \times V$ such that $(X, v), (X, w) \in A$ implies $v = w$. An assignment of cardinality n is called complete, otherwise it is called partial. A complete assignment satisfying all constraints is called a solution.

Definition 2. • Given a set S and a set of sets $P = \{P_1, \dots, P_r\}$ such that $\bigcup_i P_i = S$ and the P_i are pairwise non-overlapping, we say that P is a partition of S , and we write $S = \sum_i P_i$.

- Given a set S and a partition $S = \sum_i P_i$, a bijection $\pi : S \mapsto S$ such that $\pi(P_i) = P_i$ (where $\pi(P_i) = \{\pi(s) \mid s \in P_i\}$) is called a piecewise permutation over $S = \sum_i P_i$.

The type of symmetry that the method can tackle efficiently is defined as follows:

Definition 3. • Given a CSP (Z, V, D, C) , and partitions $Z = \sum_{k \leq r} P_k$, $V = \sum_{l \leq s} Q_l$, we say that the CSP has piecewise variable and value symmetry iff all variables within each P_k and all values within each Q_l are considered symmetric [Cohen et al., 2006].

- Given two assignments A and B on a piecewise symmetric CSP, we say that A dominates B iff there exist piecewise permutations π over $Z = \sum_{k \leq r} P_k$ and α over $V = \sum_{l \leq s} Q_l$ such that for all $(X, v) \in A$ it holds that $(\pi(X), \alpha(v)) \in B$.
- Given two arbitrary assignments A and B for a piecewise symmetric CSP, we call the problem of determining whether A dominates B the Dominance Detection Problem.

Equipped with these definitions, we briefly review how dynamic and static SSB work on piecewise symmetric CSPs.

3 Structural Symmetry Breaking

Among dynamic symmetry breaking methods, for problems with vast amounts of symmetries, Symmetry Breaking by Dominance Detection (SBDD) has shown to perform most

efficiently [Fahle et al., 2001; Focacci & Milano, 2001]. The idea of SBDD is very simple: Before a new subtree is explored, we first check if it does not map, under some symmetry, into a subtree that has been fully explored earlier. Or, in the terminology of SBDD, that the current partial assignment is not *symmetrically dominated* by one that has been studied earlier (see Definition 3). Due to the fact that we check for dominance rather than equivalence, it is sufficient to compare with only a linear number of previously explored partial assignments [Fahle et al., 2001; Focacci & Milano, 2001]. While Symmetry Breaking during Search (SBDS [Gent & Smith, 2000]) adds a constraint for each symmetry at every backtrack, SBDD essentially searches for an applicable symmetry which makes it much more efficient for problems with a lot of symmetry.

The algorithmically interesting part of SBDD is to find efficient ways to perform the actual dominance check, that is, the search for a variable permutation π and a value permutation α which would show that the current partial assignment B is dominated by some previously explored partial assignment A according to Definition 3. For general problems, it has been shown that computational group theory can be exploited to perform this task [Gent et al., 2002; 2003]. While this is convenient from the user's perspective, there are no guarantees that the dominance check will be performed in polynomial time. As a matter of fact, it was shown that dominance checking is NP-hard in general [Sellmann & Van Hentenryck, 2005]. However, for special cases of symmetry it is still possible to check dominance efficiently. Structural symmetry breaking was originally developed to do exactly that, namely for the task of performing efficient dominance checks for piecewise symmetric CSPs.

3.1 Dynamic Structural Symmetry Breaking

The core idea for devising an efficient dominance checker for piecewise symmetric CSPs lies in the definition of *signatures* of values under an assignment.

Definition 4. • Given a partial assignment A , for all values v , we define

$$\text{sign}_A(v) := (|\{X_i \in P_k \mid (X_i, v) \in A\}|)_{k \leq r},$$

where k indexes the different variable partitions $\sum_{k \leq r} P_k$. That is, the signature of v under A is the tuple that counts, for each variable partition, by how many variables in the partition the value is taken in A .

- We say that a value v in an assignment A dominates a value w in assignment B iff v and w belong to the same value-symmetry class and $\text{sign}_A(v) \leq \text{sign}_B(w)$.¹
- We say that a value v in an assignment A is structurally equivalent to a value w in assignment B iff v and w belong to the same value-symmetry class and $\text{sign}_A(v) = \text{sign}_B(w)$.

To understand the definition better, consider the following example: We have variables X_1, \dots, X_8 over domains $D(X_i) = \{v_1, \dots, v_6\}$. Now assume that the first four and the last four variables are indistinguishable, i.e. $P_1 = \{X_1, \dots, X_4\}$ and $P_2 = \{X_5, \dots, X_8\}$. Furthermore,

¹Where the \leq -relation on vectors is defined as the usual component-wise comparison, i.e.: $x \leq y$ iff $x_i \leq y_i \forall i$.

assume that $Q_1 = \{v_1, \dots, v_3\}$, $Q_2 = \{v_4, \dots, v_6\}$, and that we are given the following two assignments: $A_1 = \{(X_1, v_1), (X_2, v_1), (X_3, v_2), (X_6, v_5), (X_7, v_1), (X_8, v_2)\}$ and $A_2 = \{(X_1, v_6), (X_2, v_1), (X_3, v_2), (X_4, v_2), (X_5, v_1), (X_6, v_6), (X_7, v_2), (X_8, v_2)\}$. Now, the signature of v_1 under A_1 tells us, for each variable partition, how many variables have already been assigned to v_1 . Therefore, we have $\text{sign}_{A_1}(v_1) = (2, 1)$.

In the first assignment, we see that: 1. There is one value (v_1) in Q_1 that is taken by two variables in P_1 and one in P_2 . 2. There is one value (v_2) in Q_1 that is taken by one variable in P_1 and one in P_2 . 3. There is one value (v_5) in Q_2 that is taken by one variable in P_2 . On the other hand, in the second assignment: I. There is one value (v_2) in Q_1 that is taken by two variables in P_1 and two variables in P_2 . II. There is one value (v_1) in Q_1 that is taken by one variable in P_1 and one in P_2 . III. There is one value (v_6) in Q_2 that is taken by one variable in P_1 and one in P_2 . Lining up I-I ($v_1 \mapsto v_2$, $\{X_1, X_2\} \mapsto \{X_3, X_4\}$, $\{X_7\} \mapsto \{X_7, X_8\}$), 2-II ($v_2 \mapsto v_1$, $\{X_3\} \mapsto \{X_2\}$, $\{X_8\} \mapsto \{X_5\}$), and 3-III ($v_5 \mapsto v_6$, $\{X_6\} \mapsto \{X_6\}$), we see that A_2 is structurally (that is modulo application of some symmetries) an assignment that extends A_1 , or, in other words, that A_1 symmetrically dominates A_2 according to Definition 3.

A lemma from [Sellmann & Van Hentenryck, 2005] is essential for performing a dominance check efficiently, like the one that we just performed in an ad hoc manner:

Lemma 1. *An assignment A dominates another assignment B in a piecewise symmetric CSP iff there exists a piecewise permutation α over $\sum_{l \leq s} Q_l$ such that v in A dominates $\alpha(v)$ in B for all $v \in V$ according to Definition 4.*

The lemma allows us to efficiently check dominance between assignments A and B : We set up a bipartite graph where, for each value v , there is one node on the left and one on the right. An edge connects two nodes with associated values v and w from the same value partition iff $\text{sign}_A(v) \leq \text{sign}_B(w)$. Then, A dominates B iff the bipartite graph contains a perfect matching. Computationally, piecewise symmetric dominance is thereby not harder than solving a bipartite matching problem (see Figure 1).

To summarize, dynamic SSB is a special case of SBDD. Before we expand a new search-node we first check if the partial assignment that led us to the current node is not dominated by any partial assignment that has been fully explored earlier. SBDD ensures that there is only a linear number of dominance checks needed. SSB performs the dominance checks by computing the signatures of values under the partial assignments in question, sets up a bipartite graph and prunes the current node if and only if a perfect matching can be found. In [Sellmann & Van Hentenryck, 2005] it was shown how dynamic SSB can also be used for filtering purposes in time $O(nm^{3.5} + n^2m^2)$, where m is the number of values and n the number of variables in the given CSP.

3.2 Static Structural Symmetry Breaking

In [Flener et al., 2006] the signature abstractions of SSB could be exploited to devise a linear set of constraints which provably leaves one and only one solution in each equivalence class of solutions. When we assume a total ordering of variables $Z = \{X_1, \dots, X_n\}$ and values $V = \{v_1, \dots, v_m\}$, as

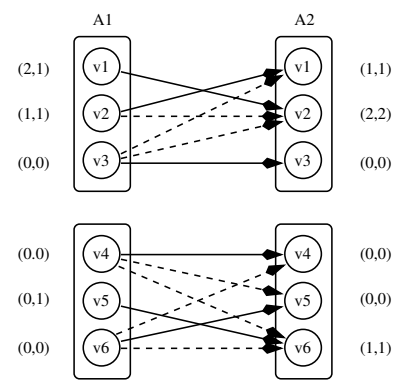


Figure 1: A bipartite graph set up to see if there exists a permutation of values that shows dominance between two assignments according to Lemma 1. The Figure gives the signatures for each value, links pairs of values where the one in assignment A_1 dominates to the one in A_2 , and a perfect matching that proves that A_1 dominates A_2 is designated by solid lines.

usual, we can break variable symmetry within each variable component simply by requiring that lower-indexed variables take smaller or equal values. To break the value symmetry simultaneously, we resort to the signatures of values in complete assignments: Within each value component, we require that smaller values have lexicographically greater or equal signatures. Then the problem boils down to computing the signatures of values efficiently. Fortunately, in CP this is an easy task when using the existing global cardinality constraint (GCC) which allows us to restrict and count how many times a value is taken by a given set of variables [Regin, 1996].

Formally, static SSB consists in adding the following static set of constraints to a piecewise symmetric CSP $\langle \sum_{k=1}^a P_k, \sum_{l=1}^b Q_l, C \rangle$ with $P_k = \{X_{i_k}, \dots, X_{i_{k+1}-1}\}$ and $Q_l = \{v_{j_l}, \dots, v_{j_{l+1}-1}\}$:

1. $X_h \leq X_{h+1}$ for all $k \in \mathbb{N}_a$, $i_k \leq h < i_{k+1} - 1$,
2. $\text{GCC}(X_{i_k}, \dots, X_{i_{k+1}-1}, d_1, \dots, d_m, f_1^k, \dots, f_m^k)$ for all $k \in \mathbb{N}_a$,
3. $(f_h^1, \dots, f_h^a) \geq_{lex} (f_{h+1}^1, \dots, f_{h+1}^a)$ for all $l \in \mathbb{N}_b$, $j_l \leq h < j_{l+1} - 1$,

where \mathbb{N}_w denotes the numbers $\{1, \dots, w\}$, i_k denotes the index in \mathbb{N}_n of the first variable in Z of variable component P_k , with $i_{a+1} = n + 1$, and j_l denotes the index in \mathbb{N}_m of the first value in V of value component Q_l , with $j_{b+1} = m + 1$.

That is, with the help of GCC, we count the frequency $f_h^k = |\{X_g \mid X_g \in P_k \ \& \ (X_g, v_h) \in \alpha\}|$ of how often each value v_h is taken under assignment α by variables in each variable component P_k . For a solution α , we denote by $\text{sig}_\alpha(v_h) := (f_h^1, \dots, f_h^a)$ the signature of v_h under α . For all consecutive values v_h, v_{h+1} in the same value component, we require that their signatures are lexicographically non-increasing, i.e., $\text{sig}_\alpha(v_h) \geq_{lex} \text{sig}_\alpha(v_{h+1})$. [Flener et al., 2006] showed that these constraints leave exactly one representative in each class of symmetric solutions. Using Regin's filtering algorithm for the GCC [Regin, 1996], filtering all static SSB constraints does not take longer than

$O(\sum_{k=1}^a |P_k|^2 m) = O(n^2 m)$, where m is the number of values and n the number of variables in the given CSP. Note that this is significantly faster than the time for dynamic symmetry breaking!

4 Model Restarts

As our review of dynamic and static structural symmetry breaking exemplified, static symmetry-breaking techniques usually impose much less overhead, both regarding the programming burden as well as the workload per choice point. However, they suffer from one important drawback, and that is the fact that they are much more sensitive to search-orderings. Both in [Kiziltan, 2004] and [Smith, 2005] it has been noted that static symmetry-breaking constraints cause great variances in the expected runtime. Knowing that static symmetry-breaking constraints work by excluding all but one representative out of each equivalence class of solutions, this is hardly surprising: When the symmetry-breaking constraints are not aligned with the search-orderings, static symmetry-breaking constraints may interrupt the construction of many perfectly good solutions, simply because they are not the representatives we have chosen. Therefore, in [Puget, 2006] Puget devised semi-static constraints that provably do not exclude the first solution of a search without symmetry-breaking constraints. Unfortunately, no such method is known for the case of piecewise symmetry yet.

On the other hand, when using static search orderings, they themselves are much less robust and perform with greatly varying performance on different problem instances. The question arises how we can combine the benefits of being able to change the search orderings while using lean static symmetry breaking.

We exploit the idea of randomization and restarts [Gomes et al., 1997; Kautz et al., 2002], which has been shown to greatly improve the robustness of systematic search: When the search takes too long (as determined by exceeding a given fail-limit) we interrupt our search, and try again with an updated fail-limit [Kautz et al., 2002]. To avoid that we conduct the same search over and over again, a random component is added to the selection heuristics for the branching variable and/or the branching value. This method marks one of the great break-throughs in the past fifteen years. It has had a profound impact on the way how we conduct systematic search today. The method has been proven both experimentally and theoretically to eliminate heavy-tailed run-time distributions [Gomes et al., 1997]. The latter have been observed to occur frequently in SAT and constraint satisfaction problems where they can cause “infinite” mean performance (whereby in actuality the distributions are of course truncated due to the finite character of concrete problem instances) while the median runtime may even be constant. Complemented by non-chronological backtracking and no-good learning, the idea of randomization and restarts marks one of the backbones of modern systematic SAT solvers.

In order to improve the robustness of static symmetry breaking, we therefore propose to exploit randomization and restarts. Note that, when posing static symmetry breaking constraints, there is often a lot of freedom in how we determine the representatives that we leave in each equivalence class of solutions (see, e.g., [Smith, 2005]). In our case, with respect to the ordering of variables, we have the freedom to

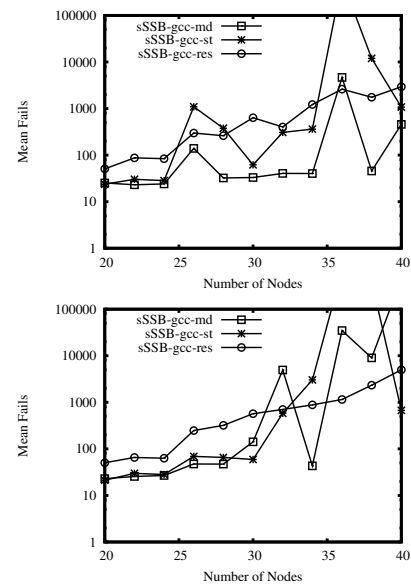


Figure 2: The figures give mean number of fails (log-scale) on 20 instances of the biased graph-coloring benchmark with $q = 0.5$ (top) and $q = 1$ (bottom). The cutoff was set to one hour.

choose the ordering of variables within each variable partition. However, this choice is hardly of any significance as the variables are interchangeable anyway. More important is our freedom to arbitrarily choose the ordering of the variable partitions which defines the ordering within the value signatures.

We start our search and use the static variable ordering as induced by the static symmetry breaking constraints. This avoids clashes between static SSB and the search ordering. However, the particular search ordering we choose may not be suited well for the concrete instance we need to solve. Consequently, we interrupt our search when a fail-limit is reached. Now, we would like to choose a new and somewhat randomized search ordering, but if we do, then it is likely to clash with our static constraints. Consequently, rather than updating the search ordering only, *we also change the underlying CP model!* That is, at every restart we do not only change the search-orderings, but also the corresponding static symmetry-breaking constraints. This way, we avoid clashes between the search-orderings and static symmetry-breaking constraints, and are still not bound to one static search-ordering which may be bad for the given problem instance. We refer to this simple and easy to use idea as “model restarts.”

5 Experimental Results

We experiment on the benchmark introduced in [Law et al., 2007] that consists of graph coloring problems over symmetric graphs. Note that, in graph coloring, nodes that have the same set of neighbors are interchangeable and form a partition of piecewise interchangeable variables in the corresponding CSP. Furthermore, all colors are interchangeable values. In [Law et al., 2007], randomized graph coloring problems are generated with either a uniform or a biased distribution of

partitions of interchangeable nodes in the graphs, and a parameter q influences the density of the graphs. For details, please see [Law et al., 2007]. For reasons of comparability, our experimental set-up is the same as in [Law et al., 2007]. Each data point we report represents the mean of runs on 20 different instances with a cutoff of one hour. For each data point, at least 90% of all runs finish within this time-frame. For the restarted methods, fail-limits grow linearly as multiples of 100. We also use the same CSP model as in [Law et al., 2007], whereby it is worth noting that this model does not contain any lower bounds on the objective function which should be added if one was actually interested in solving the benchmark efficiently. However, for the sake of being able to compare our results with those published in [Law et al., 2007], we did not add a lower bound to the model.

In Figure 2, we study three different algorithms on the biased graph-coloring benchmark: static SSB in combination with a min-domain heuristic (sSSB-gcc-md), static SSB in combination with a corresponding static variable ordering (sSSB-gcc-st), and static SSB with model restarts (sSSB-gcc-res), where the ordering of variable partitions is permuted at every model restart, with a bias to place larger partitions earlier in the ordering. The figure shows the average number of failures, but since the time per choice point for all variants is practically the same, we get the exact same picture when comparing running times (for the actual runtime of sSSB-gcc-res, compare with Figure 3). We can see clearly that sSSB-gcc-md and sSSB-gcc-st are both not robust at all. The curves are highly erratic. The reason why sSSB-gcc-st shows such a high variance in solution time is that the static ordering that is chosen is good for some instances and bad for others. Dynamic variable orderings like the min-domain heuristic usually lead to much more robust performance. However, in the case of symmetric problems, the dynamic orderings may clash with the static constraints, and sSSB-gcc-md is not performing consistently well either. On the other hand, we can see clearly how model restarts greatly improve the robustness: sSSB-gcc-res behaves much more predictably, and while it loses against the other variants where these happen to work fine, as the instances become larger and harder to solve, we observe large benefits when using the more robust method.

Our next experiment regards a different variant of static SSB that was introduced in [Law et al., 2007]. This variant leaves the same representatives in each equivalence class as the symmetry-breaking constraints that we reviewed earlier, but in a way that promises greater filtering power. As it is based on regular constraints, we refer to this version as sSSB-reg. After comparing the new decomposition with the [Flener et al., 2006] variant of static SSB on the graph coloring benchmark (to which we refer to as sSSB-gcc), the conclusion in [Law et al., 2007] was that sSSB-reg worked orders of magnitude faster than sSSB-gcc. However, in this work it was only investigated how a different ordering of the variable partitions for the value-signatures affects solution efficiency, while the algorithms tested all used the min-domain criterion as branching variable selector. However, as we just showed, static SSB works best in combination with a static variable ordering.

In Figure 3 we compare restarted static SSB based on GCC constraints (sSSB-gcc-res) with the static SSB variant

based on regular constraints that was introduced in [Law et al., 2007] (with min-domain heuristic, sSSB-reg-md). The first algorithm was run on an AMD-Athlon 64-X2 3800+ (2.0GHz), the latter was run on a Sun Blade 2500 (1.6GHz) and the curve shown is an adaptation of that shown in [Law et al., 2007]. Based on the data given in that paper, we can infer that their machine can process about 20K failures per second when running sSSB-gcc-md, while we measured 30K failures per second on our architecture for the same method. We conclude that our machine works about a factor 1.5 faster and thus divided the data-points underlying the curve shown in [Law et al., 2007] by that factor to make the comparison fair.

We see that the restarted method works equally robust as sSSB-reg-md, but roughly one order of magnitude faster. While the motivation for the new decomposition in [Law et al., 2007] was to eliminate the use of “global cardinality constraints which can be expensive to propagate,” we find that sSSB-gcc actually incurs much less overhead per choice point than sSSB-reg. As a matter of fact, sSSB-gcc visits about two orders of magnitude more choice points than sSSB-reg: In [Law et al., 2007] it is reported that sSSB-reg visits less than 100 choice points on the biased instances (which makes it highly unlikely that restarts will lead to any further improvements), and the number of failures of sSSB-gcc-res is shown in Figure 2. This implies that sSSB-gcc works almost three orders of magnitude faster per choice point! It is this efficiency which gives the simple sSSB-gcc-res the advantage over the much more effective filtering of sSSB-reg-md.

Finally, consider the curve denoted with sSSB-gcc-tradres which shows the performance of a traditional restart strategy where the model is static while the search orderings are randomized. As we can see, traditional restarts help to make the method more robust, but the performance is much improved (roughly by an order of magnitude) when both search orderings and symmetry-breaking constraints are being aligned between restarts.

6 Static vs. Dynamic Symmetry Breaking

In our final section, we provide the first ever comparison of static and dynamic SSB. Consider Figure 3 again. The curve denoted with dSSB-md denotes dynamic SSB in combination with the min-domain heuristic. As the theoretical runtime comparison of dSSB and sSSB in Section 2 already suggested, we find that the dynamic variant cannot compete with the static methods, despite our great efforts to tune the method as best as possible using the heuristic improvements introduced in [Heller & Sellmann, 2006].

We were curious if this would also hold for other types of problems. We introduce a random benchmark generator which produces piecewise symmetric CSPs with different characteristics: Given a number of variables n and values m , as well as the number of variables per constraint n_c and the number of values per constraint m_c , we generate a given number of global cardinality constraints (GCC), each over a set of n_c randomly chosen variables and m_c randomly chosen values, whereby we enforce that all variables in the constraint together take each chosen value exactly once. We vary this basic concept by either adding one more GCC over all variables and values that enforces that each value be chosen at most 2 times, or by adding an AllDifferent constraint over all

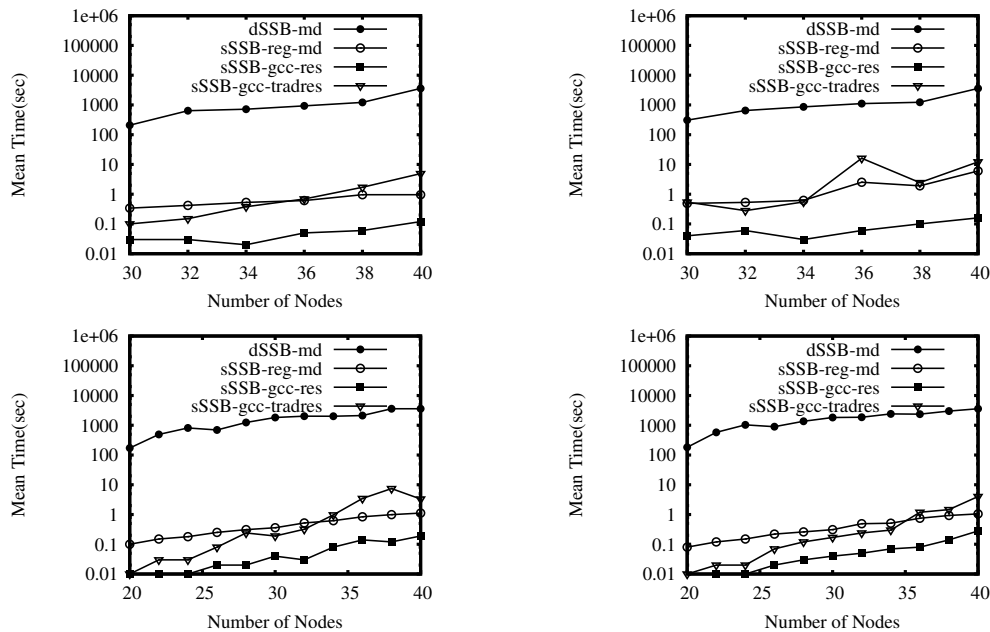


Figure 3: Mean time (seconds, log-scale) on 20 instances of the uniform (top) and biased (bottom) graph-coloring benchmark with $q = 0.5$ (left) and $q = 1$ (right).

variables and values. Note how every constraint in the problem partitions both the set of variables and the set of values in those that are involved in the constraint (all of them are treated equally by the constraint), and those that are not. In the end, we know that all variables (and values) that always appear together in the same constraints are interchangeable. In particular, we consider problems with five constraints (one global over all variables and values, plus four other), with 15 variables and values. The number of variables per constraint is fixed at 12 while the number of values per constraint runs from 2 to 12, thus giving us a range of differently constrained instances.

In Figure 4 we compare three different algorithms on this new benchmark: dynamic SSB that uses a min-domain heuristic to choose the branching variable (dSSB-md), static SSB with the min-domain heuristic (sSSB-md), and static symmetry breaking with a static variable ordering that is in accordance to the static symmetry-breaking constraints (sSSB-st). As for the graph coloring benchmark, we observe that static SSB can lead to orders of magnitude speed-ups over dynamic SSB, especially in the critically constrained region. The more erratic curve of sSSB-md again results from a much higher variance in running time that is caused by the combination of static symmetry breaking with a dynamic variable ordering.

7 Conclusions

We implemented static and dynamic structural symmetry breaking (SSB) and tested it on an existing as well as a novel benchmark set for CSPs with interchangeable variables and values. In order to improve the robustness of static symmetry breaking, we introduced the idea of model restarts where we reset the static symmetry-breaking constraints in accordance to a new variable ordering. When comparing with a differ-

ent variant of static SSB introduced in [Law et al., 2007], we found that the latter method is able to reduce the number of choice points significantly. However, the decomposition that was introduced in [Flener et al., 2006] is still an order of magnitude faster, and in combination with model restarts it is equally robust as the method from [Law et al., 2007]. When comparing dynamic and static SSB, we found that the ability of dynamic SSB to accommodate dynamic branching variable selection does not counter-balance its much larger costs per choice point. Static SSB outperforms dynamic SSB by orders of magnitude. Future research regards the question whether model restarts can be helpful in other domains as well, for example when a given problem can be modeled in different ways by an automatic modelling tool.

References

- [Backofen & Will, 1999] R. Backofen and S. Will. Excluding symmetries in constraint-based search. *CP*, 73–87, 1999.
- [Brown et al., 1988] C. Brown, L. Finkelstein, P. Purdom Jr. Backtrack searching in the presence of symmetry. *AAECC-6*, 99–110, 1988.
- [Cohen et al., 2006] D.A. Cohen, P. Jeavons, C. Jefferson, K.E. Petrie, B.M. Smith. Symmetry Definitions for Constraint Satisfaction Problems. *Constraints*, 11(2–3): 115–137, 2006.
- [Crawford et al., 1996] J. Crawford, M. Ginsberg, E. Luks, A. Roy. Symmetry-breaking predicates for search problems. *KR*, 149–159, 1996.
- [Fahle et al., 2001] T. Fahle, S. Schamberger, M. Sellmann. Symmetry Breaking. *CP*, 93–107, 2001.

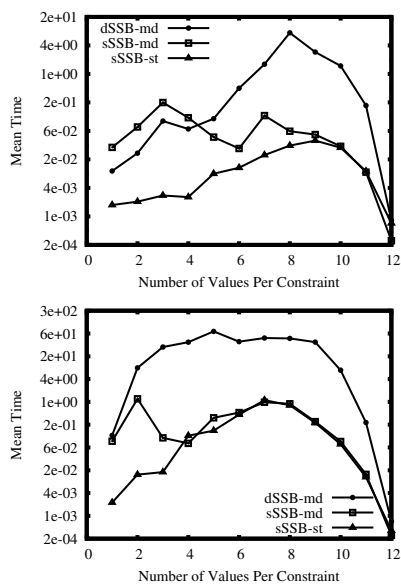


Figure 4: The figures give mean times in seconds (log-scale) on 100 instances with 15 variables and values, 12 variables per constraint and AllDifferent (top) or GCC (bottom) as constraint over all variables and values. The cutoff was set to 600 seconds.

[Flener et al., 2002] P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, T. Walsh. Breaking row and column symmetries in matrix models. *CP*, 462–476, 2002.

[Flener et al., 2006] P. Flener, J. Pearson, M. Sellmann, P. Van Hentenryck. Static and Dynamic Structural Symmetry Breaking. *CP*, 695–699, 2006.

[Focacci & Milano, 2001] F. Focacci and M. Milano. Global cut framework for removing symmetries. *CP*, 77–92, 2001.

[Gent et al., 2003] I. Gent, W. Harvey, T. Kelsey, S. Linton. Generic SBDD using computational group theory. *CP*, 333–347, 2003.

[Gent et al., 2002] I.P. Gent, W. Harvey, T. Kelsey. Groups and Constraints: Symmetry Breaking During Search. *CP*, 415–430, 2002.

[Gent & Smith, 2000] I. P. Gent and B. M. Smith. Symmetry breaking during search in constraint programming. In *ECAI*, pages 599–603, 2000.

[Gomes et al., 1997] C.P. Gomes, B. Selman, N. Crato. Heavy-Tailed Distributions in Combinatorial Search. *CP*, 121–135, 1997.

[Heller & Sellmann, 2006] D. Heller and M. Sellmann. Dynamic Symmetry Breaking Restarted. *CP*, 721–725, 2006.

[Kautz et al., 2002] H. Kautz, E. Horvitz, Y. Ruan, C. Gomes, B. Selman. Dynamic Restart Policies. *AAAI*, 674–682, 2002.

[Kiziltan, 2004] Z. Kiziltan. Symmetry Breaking Ordering Constraints. *PhD Thesis*, Uppsala University, 2004.

[Law et al., 2007] Y.-C. Law, J. Lee, T. Walsh, J. Yip. Breaking symmetry of interchangeable variables and values. *CP*, 423–437, 2007.

[Puget, 2006] J. F. Puget. Dynamic Lex Constraints. *CP*, 453–467, 2006.

[Puget, 2002] J.-F. Puget. Symmetry breaking revisited. *CP*, 446–461, 2002.

[Regin, 1996] J.-C. R egin. Generalized arc-consistency for global cardinality constraint. In *AAAI*, pages 209–215. AAAI Press, 1996.

[Roney-Dougal et al., 2004] C. Roney-Dougal, I. Gent, T. Kelsey, S. Linton. Tractable symmetry breaking using restricted search trees. *ECAI*, 211–215, 2004.

[Sellmann & Van Hentenryck, 2005] M. Sellmann and P. Van Hentenryck. Structural Symmetry Breaking. *IJCAI*, 298–303, 2005.

[Smith, 2005] B.M. Smith. Sets of Symmetry Breaking Constraints. *SymCon05*, 2005.

[Van Hentenryck et al., 2003] P. Van Hentenryck, P. Flener, J. Pearson, M. Agren. Tractable symmetry breaking for CSPs with interchangeable values. *IJCAI*, 277–282, 2003.