An Empirical Study of Structural Symmetry Breaking

Aurojit Panda
Department of Computer Science
Brown University
apanda@cs.brown.edu

People who helped with far too much of this:

Meinolf Sellman, Dan Heller and Justin Yip with all the work that went into making this happen, and for help with the writing. Itay Neeman, Jason Davis and Lian Garton for all the help they gave, and for the long nights. Tom Doeppner for helping, just because we asked.

Abstract

We present an empirical comparison between dynamic and static methods for structural symmetry breaking (SSB). SSB was recently introduced as the first method for breaking all piecewise value and variable symmetries in a constraint satisfaction problem. So far it is unclear which of the two techniques is better suited for piecewise symmetric CSPs, and this study presents results from the first empirical comparison between the two. The empirical study is conducted using graph coloring, and a novel test bed for generating piecewise symmetric problems also described in this work.

1 Introduction

Symmetries can cause a significant performance hit for systematic constraint solvers. Constraint solvers can loose time exploring redundant parts of the search tree many times over. All sorts of attempts have been made to solve this problem, including adapting ordering heuristics, addition of static constraints, addition of constraints during search, and filtering values using dominance detection. The later has been found especially useful for problems involving large symmetry domains.
The core of symmetry breaking using dominance detection (SBDD) is based on the act of dominance detection. The first methods proposed for doing this in general were based on computational group theory, as done in [4] and [8], while the first methods which were provably polynomial time were designed for specific kinds of value symmetries in [6]. This work was later extended to handle any kind of value symmetry in general, in polynomial time, [11] . Based on these results, [12] showed that for specific piecewise symmetric CSPs, both value and variable symmetry could be broken in polynomial time using structural symmetry breaking (SSB).

SSB is based on introducing a structural abstraction of a given partial assignment of variables to values. This abstraction, originally proposed for a dynamic method also gives rise to a set of static constraints which can be used to break all piecewise symmetric variable and value constraints [2]. Dynamic symmetry breaking methods have the advantage of being able to accommodate dynamic variable and value ordering without running an increased risk of underperforming. Dynamic ordering has been shown to be superior in a vast majority of cases, however when used with static symmetry-breaking methods it runs the risk of not being aligned with the symmetry breaking constraints. We could for instance be exploring and dismissing

perfectly reasonable solutions because they are not favored by the static constraints, which in general pick one representative solution from an equivalence class of solutions. To address this problem [9] suggests a semi-static method that provably does not remove the first solution encountered by a dynamic-search method, however no such method is known for piecewise symmetric problems. Static methods are simpler to use, enjoy a lower-overhead per choice point, and exhibit an anticipatory character that emerges from filtering symmetry breaking constraints in combination with constraints imposed by the problem.

Ease of use aside, it is unclear which of these two methods work better: quick, anticipatory yet inflexible static methods, or heavier yet more flexible dynamic methods. This work provides the first empirical comparison between the two.

2 Background

We begin by defining terms, and exploring the differences between static and dynamic methods.

2.1 Definitions

Constraint Satisfaction Problem A constraint satisfaction problem (CSP) is defined to be a tuple (Z, V, D, C), where $Z = \{X_1, X_2, \ldots, X_n\}$ is a finite set of variable, $V = \{v_1, v_2, \ldots, v_n\}$ is a finite set of values, $D = \{D_1, D_2, \ldots, D_n\}$ is a finite set of domains, such that $D_i \subseteq V$ is the set of possible instantiations of X_i, and $C = \{C_1, C_2, \ldots, C_n\}$ is a finite set of constraints over subsets of Z, specifying their valid combinations.

Assignment Given a CSP, (Z, V, D, C), an assignment A is a set of pairs $(X, v) \in Z \times V$, such that $(X, v), (X, w) \in A \iff v = w$. Assuming the cardinality of Z is n, an assignment of cardinality n is called complete, and a complete assignment satisfying all the constraints in C is called a solution. An assignment that is not complete is defined to be a partial assignment.

Partition A partition P of set S is a set of sets $P = \{P_1, P_2, \ldots P_n\}$ such that $S = \bigcup P_i$ and $P_i \bigcap P_j == \emptyset \forall i \neq j$. We generally write $S = \sum_i P_i$.

Piecewise Permutation Given a partition P over set S, a bijection $\pi : S \rightarrow S$ is called a piecewise permutation if $\pi(P_i) = P_i \forall P_i \in P$, where $\pi(P_i) = \{\pi(x) \,|\, x \in P_i\}$

Piecewise Symmetric CSP Given a CSP (Z, V, D, C), and partition P for Z, and partition Q for V a CSP has piecewise variable and value symmetry *iff* there exists $\pi : Z \rightarrow Z$ and $\phi : Q \rightarrow Q$ such that both π and ϕ are piecewise permutations. Equivalently, given $Z = \sum_{I \leq k} P_i$ and $V = \sum_{j \leq l} Q_j$, we say the CSP has piecewise variable and value symmetry *iff* all variables within each P_i and all values within each Q_j are considered to be interchangeable.

Dominance Given two assignments A and B for a CSP, we say that A dominates B *iff* there exists a piecewise permutation π over $Z = \sum_i P_i$ and ϕ over $V = \sum_j Q_j$ such that $\forall (X, v) \in A$ we have $(\pi(X), \phi(V)) \in B$.

Dominance Detection Problem Given two arbitrary assignments A and B we call the problem of detecting whether A dominates B the dominance detection problem.

2.2 Dynamic Symmetry Breaking

There are two popular methods of breaking symmetries dynamically. The historically earlier one was Symmetry Breaking During Search (SBDS). SBDS works by adding constraints to the problem during the backtracking phase of the search, thus preventing exploration of symmetric search regions. This works well for problems with relatively few symmetries (n-queens for instance), but suffers from the fact that one constraint per symmetry is added upon backtracking. This is an obvious problem for problems involving vast symmetries.

For problems with large numbers of symmetries, Symmetry Breaking by Dominance Detection (SBDD) has been shown to be the method of choice [1, 3]. The idea behind SBDD is to check before exploring a new subtree, whether it is is symmetric to a subtree that has already been explored (or more precisely, to check whether it maps, under some symmetry, into a fully explored subtree). In the terminology of SBDD, check to make sure that the subtree currently being explored is not symmetrically dominated by a subtree that has been explored earlier. Since a check is made for dominance, rather than equivalence, it can be shown that the number of

comparisons required is a linear factor of the number of previously explored assignments [1, 3].

The algorithmically interesting part of SBDD is finding efficient ways of performing the actual dominance check, that is finding a variable permutation π and a value permutation α under which the current partial assignment B is dominated by some previously explored partial assignment A. In general, computational group theory can be exploited to perform this task, however there are no guarantees that this can be accomplished in polynomial time. In General, dominance checking is NP-hard [12]. However, for special cases of symmetry it is possible to check dominance efficiently. Structural symmetry breaking is one such efficient technique, originally targeted at piecewise symmetric CSPs.

2.2.1 Structural Dominance Detection

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

Signatures Given a partial assignment A, for all values v we define

$$sign_A(v) := (|X_i \in (X_i, v) \in A|)_{k \leq r}$$

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 an assignment B iff v and w belong to the same value-symmetry class and $sign_A(v) \leq sign_B(w)$. Here the $\leq$ relation is defined component wise, i.e. $x \leq y \iff x_i \leq y_i \, \forall i$ where x_i denotes the ith component of the tuple x.
- We say that v and w are structurally equivalent iff $sign_A(v) = sign_B(w)$.

Consider the following example. We have a CSP with variables $\{X_1, \ldots, X_8\}$ over domains $D(X_i) = \{v_1, \ldots, v_6\}$. Assume that the first four, and the last four variables are symmetric with each other, i.e. $P_1 = \{X_1, \ldots, X_4\}$, and $P_2 =$

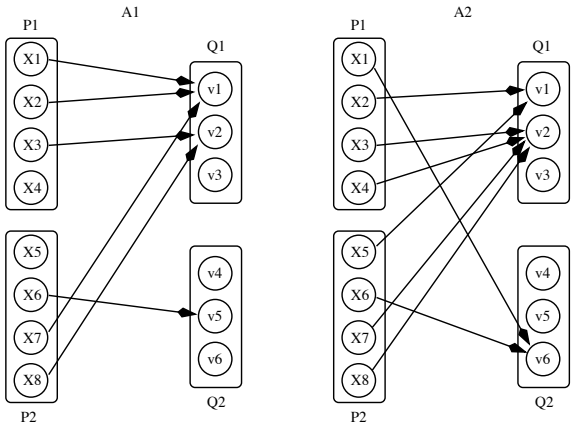

Figure 1: Assignments A_1 and A_2.

$\{X_5, \ldots, X_8\}$. Further assume that $Q_1 = \{v_1, \ldots, v_3\}$ and $Q_2 = \{v_4, \ldots, v_6\}$. Now given assignments $A_1 = \{(X_1, v_1), (X_2, v_1), (X_3, v_2), (X_6, v_5), (X_7, v_1), (X_7, v_2)\}$, and $A_2 = \{(X_1, v_6), (X_2, v_1), (X_3, v_2), (X_4, v_2), (X_5, v_1), (X_6, v_6), (X_7, v_2), (X_8, v_2)\}$ (Figure 1), we have $sign_{A_1}(v_1) = (2, 1)$, the number of variables in each variable partition assigned to v_1. Now we can see from above that:

A In assignment 1:

1 $v_1 \in Q_1$ has signature $(2, 1)$

2 $v_2 \in Q_1$ has signature $(1, 1)$

3 $v_5 \in Q_2$ has signature $(0, 1)$

B In assignment 2:

I $v_2 \in Q_1$ has signature $(2, 2)$

II $v_1 \in Q_1$ has signature $(1, 1)$

III $v_6 \in Q_2$ has signature $(1, 1)$

Lining these conditions up we see that:

1 - I ($v_1 \mapsto v_2$, $\{X_1, X_2\} \mapsto \{X_3, X_4\}$, $\{X_7\} \mapsto \{X_7, X_8\}$)

2 - II ($v_2 \mapsto v_1$, $\{X_3\} \mapsto \{X_2\}$, $\{X_8\} \mapsto \{X_5\}$)

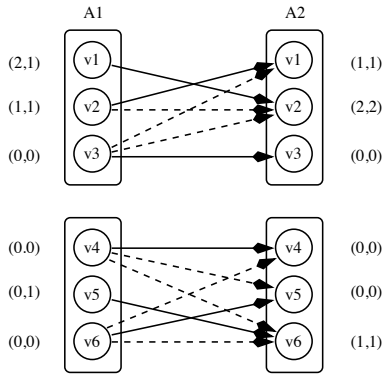

Figure 2: A bipartite graph illustrating a method for determining dominance between two assignments using Lemma 1. The Figure gives the signatures for each value, links pairs of values where the one in assignment A_1 dominates the one in A_2, and a perfect matching that proves that A_1 dominates A_2 is designated by solid lines.

3 - III ($v_5 \mapsto v_6$, $\{X_6\} \mapsto \{X_6\}$)

The relationships above show that A_2 is structurally an assignment that extends A_1, i.e. modulo application of symmetries, and hence reordering, the assignment A_1 extends A_2. In other words, A_1 *symmetrically dominates* A_2 according to the definition of dominance given previously.

We now use a result from [12] to find an efficient way of performing a dominance check similar to what was performed above.

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 $v \in V$. Value dominance is as defined using signatures.*

We can use this lemma to build an efficient method for checking dominance relations between two assignments. Given assignments A and B we can 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 if and only if $sign_A(v) \leq sign_B(w)$. A dominated B if and only if the bipartite graph contains a perfect matching (see Figure 2). Computationally, piecewise symmetry detection is hence no harder than solving a bipartite matching problem.

To summarize, we use SBDD to perform dynamic structural symmetry breaking. Before expanding a new search-node we check if the partial assignment leading up to the current node is dominated by a partial assignment that has been fully explored earlier. We explore only if this is not the case. Dominance detection ensures that we have to carry out no more than a linear number of dominance checks. SSB performs dominance checks by computing signatures of values in the partial assignments under consideration, sets up a bipartite graph, and checks to see if it can find a perfect matching.

2.2.2 Filtering

As previously noted, one of the advantages of the static method is its anticipatory character. We can extend the above dominance checking algorithm to provide us with a method to filter values from domains which would lead to a symmetric choice point. Put differently, we an use symmetry-based filtering to anticipate when variable assignments will result in symmetric configurations.

To do so we have to distinguish between two different types of filtering: *ancestor-based filtering*, i.e. comparing extensions to the current partial assignment against fully explored assignments, and *sibling-based filtering*, i.e. comparing extensions to the current partial assignment against other such extensions.

For the later we assume that all siblings are generated by branching on the same variable, in which case we note that sibling symmetry can only be caused by value symmetry in the problem. We can thus break all sibling symmetry by choosing one arbitrary value out of each group of values within the same value partition that have the same signature.

Ancestor-based filtering can be performed by considering almost successful dominance checks. When the bipartite graph that we set up (as shown above), contains an almost perfect match, i.e. only one edge is missing in the graph preventing us from finding a perfect match, we can quickly identify *critical edges*, and check whether a variable assignment would cause a critical edge to be added to the graph. If this this the case, we have found a critical variable assignment, that should be avoided by removing that value from the variable's domain. [12] shows that ancestor-based filtering for a single node can be carried out in time $O(nm^{3.5} + n^2m^2)$, where the

problem has m values, and n variables. Since the complexity of filtering is dominated by ancestor-based filtering, filtering for a single node can be carried out in $O(nm^{3.5} + n^2m^2)$ time.

2.3 Static Structural Symmetry Breaking

As was shown in [2], we can exploit the signature abstraction to devise a linear set f constraints which provably leaves only one solution in each equivalence class of solutions. We start by assuming a total ordering of the variables $Z = \{X_1, \ldots, X_n\}$ and values $V = \{v_1, \ldots, v_m\}$. We can break variable symmetry by requiring that variables with smaller indices take smaller or equal values.

To break value symmetry we resort to using the signatures of values in complete assignments. Within each value component, we require that smaller values have lexicographically larger or equal signatures than larger values. The problem then reduces to efficiently computing the signatures for values. This can be accomplished 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, as described in [10].

Formally, given a piecewise symmetric CSP $(\sum_{k=1}^{a} P_k, \sum_{l=1}^{b} Q_l, C)$, with $P_k = \{X_{i_k}, \ldots, X_{i_{k+1}-1}\}$ and $Q_l = \{v_{j_l}, \ldots, v_{j_{l+1}-1}$, the static constraints added are:

$$\begin{aligned} X_h &\leq X_{h+1} && \forall 1 \leq k \leq a, i_k \leq h < i_{k+1} \\ GCC(X_{i_k}, \ldots, X_{i_{k+1}-1}, v_1, \ldots, v_b, f_1^k, \ldots, f_m^k) && \forall 1 \leq k \leq a \\ (f_h^1, \ldots, f_h^a) &\geq_{lex} (f_{h+1}^1, \ldots, f_{h+1}^a) && \forall 1 \leq l \leq b, j_l \leq h < j_{l+1} \end{aligned}$$

where i_k denotes the index of the first variable of variable component P_k with $i_{a+1} = n + 1$, and j_l denotes the index of the first value in value component Q_l with $j_{b+1} = m + 1$.

As an example, consider the problem of scheduling study groups for two sets of five indistinguishable students. There are six identical groups of tutors with four

slots each. Let $\{S_1, \ldots, S_5\} + \{S_6, \ldots, S_{10}\}$ be two sets of piecewise interchangeable variables denoting each of the students. Let the domain $\{t_1, \ldots t_6\}$ denote the set of interchangeable tutor groups. The static constraints are:

$$\begin{aligned} S_1 \leq \ldots \leq S_5 \\ S_6 \leq \ldots \leq S_10 \\ GCC(S_1, \ldots, S_5, t_1, \ldots, t_6, f_1^1, \ldots, f_6^1) \\ GCC(S_6, \ldots, S_{10}, t_1, \ldots, t_6, f_1^2, \ldots, f_6^2) \\ (f_1^1, f_1^2) \geq_{lex} \ldots \geq_{lex} (f_6^1, f_6^2) \end{aligned}$$

Now consider the assignment:
$\alpha = \{(S_1, t_1), (S_2, t_1), (S_3, t_2), (S_4, t_3), (S_5, t_3)\} \bigcup \{(S_6, t_1), (S_7, t_2), (S_8, t_3), (S_9, t_4), (S_{10}, t_5)\}$.
Within each variable component, the first two ordering constraints are satisfied. We can use the GCC constraints to determine frequencies, and we find that the lexicographic constraints are satisfied since

$$(2,1) \geq_{lex} (2,1) \geq_{lex} (1,1) \geq_{lex} (0,1) \geq_{lex} (0,1) \geq_{lex} (0,0)$$

On the other hand, if student 10 moves from tutor 5, to tutor 6, the lexicographic constraints are no longer satisfied, because

$$(2,1) \geq_{lex} (2,1) \geq_{lex} (1,1) \geq_{lex} (0,1) \geq_{lex} (0,1) \not\geq_{lex} (0,1)$$

Régin provides a filtering algorithm for the GCC constraints [10], such that filtering all static structural symmetry breaking constraints takes no longer than $O(\sum_{k=1}^{a} |P_k|^2 m) = O(n^2 m)$, where m is the number of values, and n is the number of variables in a given CSP.

2.4 Model Restarts

[5] introduces the concept of model restarts. As has been shown above, static symmetry breaking imposes a smaller time overhead per choice point. However they are

very sensitive to search orderings, and can introduce a large variance in runtime. Since static symmetry-breaking algorithms work by excluding all but one representative of each equivalence class of solutions, in cases where static symmetry-breaking constraints are not aligned with search-orderings, they may interrupt the construction of many reasonable solutions simply because the solution was not the representative chosen by the constraints.

In general we choose to enforce a static search ordering which does not clash with our static-symmetry breaking constraint. However since dynamic search-orderings are known to perform better than static-search orderings, we choose to utilize randomization and restarts to provide better performance. When a search takes too long (as determined by exceeding a given fail-limit), we interrupt the search, generate a new ordering, generate a new set of static symmetry breaking constraints which do not clash with this new ordering, and restart our search with an updated fail-limit. These restarts are referred to as "model restarts".

3 Benchmarks for Comparison

We use two classes of CSPs to serve as a benchmark for our comparisons. We introduce a random benchmark generator for producing piecewise symmetric CSPs with different characteristics, and we also test on the graph coloring benchmark introduced in [7].

3.1 Random Benchmark Generator

We introduce a simple benchmark generator that produces hard instances of piecewise symmetric CSPs. Given n variables, m values, n_c, the number of variables per constraint, and m_c, the number of values per constraint, we generate a given number of global cardinality constraints (GCC), each over a set of n_c randomly chose variables, and m_c randomly chosen values, whereby we enforce that all variables in the constraint taken together, take each value in the constraint exactly once. We employ two variations on this basic concept:

- We either add a constraint to make sure that all variables take distinct values, or a GCC over all variables and values that forces each value to be taken at most two times.

	UNBIASED		BIASED	
	15		15	
VPC	AllDiff	GCC	AllDiff	GCC
2	100 (7.05 : 5.87)	100 (7.14 : 5.85)	100 (4.60 : 5.86)	100 (4.56 : 5.87)
3	100 (7.03 : 7.17)	100 (7.20 : 7.17)	100 (4.64 : 7.05)	100 (4.59 : 7.10)
4	100 (7.15 : 8.42)	100 (7.13 : 8.23)	99 (4.35 : 8.29)	99 (4.62 : 8.23)
5	100 (7.09 : 9.23)	100 (7.21 : 9.46)	87 (4.54 : 9.39)	88 (4.58 : 9.56)
6	100 (7.08 : 10.18)	100 (7.05 : 10.28)	60 (4.46 : 10.29)	62 (4.65 : 10.39)
7	99 (7.12 : 10.70)	100 (7.17 : 10.79)	33 (4.59 : 10.74)	33 (4.60 : 10.72)
8	90 (7.04 : 10.83)	95 (7.09 : 10.35)	16 (4.60 : 10.72)	15 (4.63 : 10.72)
9	81 (7.22 : 10.47)	81 (7.18 : 10.35)	1 (4.60 : 10.41)	2 (4.70 : 9.39)
10	52 (6.99 : 9.36)	61 (7.17 : 9.49)	0 (4.54 : 9.32)	0 (4.60 : 9.39)
11	18 (7.23 : 8.37)	19 (7.08 : 8.23)	0 (4.60 : 8.22)	0 (4.72 : 8.31)
12	1 (7.12 : 7.12)	1 (7.14 : 7.13)	0 (4.48 : 7.27)	0 (4.59 : 6.96)

Table 1: Percentages of feasible solutions in the different benchmark sets with 15 variables and values for different numbers of values per constraint (VPC). In brackets the average number of variable and value partitions are noted.

- We draw variables or values either uniformly or from a biased distribution where variables with higher indices are more likely to be chosen than ones with lower indices.

Every constraint in the problem partitions variables and values in the problem into two sets, those involved in the constraints, and those that are not. Variables and values which appear together in all the constraints that they appear in, are in the same equivalence class, and are hence interchangeable.

As can be observed, not all randomly generated problems are feasible. For this set of benchmarks we generated files with 15 variables, and values. The number of variables per constraint was fixed at 12, while each constraint involved anywhere from 2 to 12 values. We had instances where variables selected from both uniform, and biased instances, while values were always selected uniformly. We also varied our choice for the global constraint (either 0-2GCC, or AllDifferent). Table 1 summarizes the property of instances in our benchmark.

3.2 Graph Coloring Benchmark

We also use a benchmark introduced in [7], which consists of graph coloring problems over symmetric graphs. Nodes in the graph with the same set of neighbors are interchangeable, and constitute a variable partition. Further more all colors under consideration are interchangeable, and form a value partition. In [7], graph coloring problems are generated on the basis of four parameters: n, the number of nodes, $r \leq n$, $p, q \in [0, 1]$, parameters governing graph structure. In particular, we start by selecting the sizes, between 1 and r, for the initial node partitions, ether uniformly or with a bias towards smaller partitions, until we reach a total of n nodes (the last partition can be truncated to guarantee that we have exactly n nodes). For each partition, with probability q, we connect all nodes in the partition to form a clique. Finally for each pair of partitions we connect all nodes in one partition with all nodes in the other partition with probability p.

As is done in [7], we set $r = 8$, $p = 0.5$, $q \in \{0.5, 1\}$, and $n \in \{20, \ldots, 40\}$.

4 Results

As [5] shows, dynamic structural symmetry breaking, despite its large computational overhead per choice point, is a practical method for improving the amount of time spent solving structurally symmetric CSPs. The main motivation for the current work is to study whether it is more effective to use dynamic SSB, with its ability to accommodate dynamic variable selection heuristics, or static SSB, with its lower overhead per choice point.

We benchmarked algorithms on 2 GHz AMD Athlon64 3800+ dual core CPUs, with 2 GB of main memory. In Figure 3, we show times from three different algorithms: dynamic SSB using a min-domain heuristic to choose the branching variable (dSSB-md), static SSB using a min-domain heuristic to choose the branching variable (sSSB-md), and static symmetry breaking with a static variable ordering, in accordance with the static symmetry breaking constraints (sSSB-st). We note that static SSB can lead to speed-ups of orders of magnitude when compared to dynamic SSB, especially in the critically constrained region. The more erratic curve of running static SSB with the min-domain heuristic is a result of the high variance in running time, cause by the interaction of static symmetry breaking constraints with

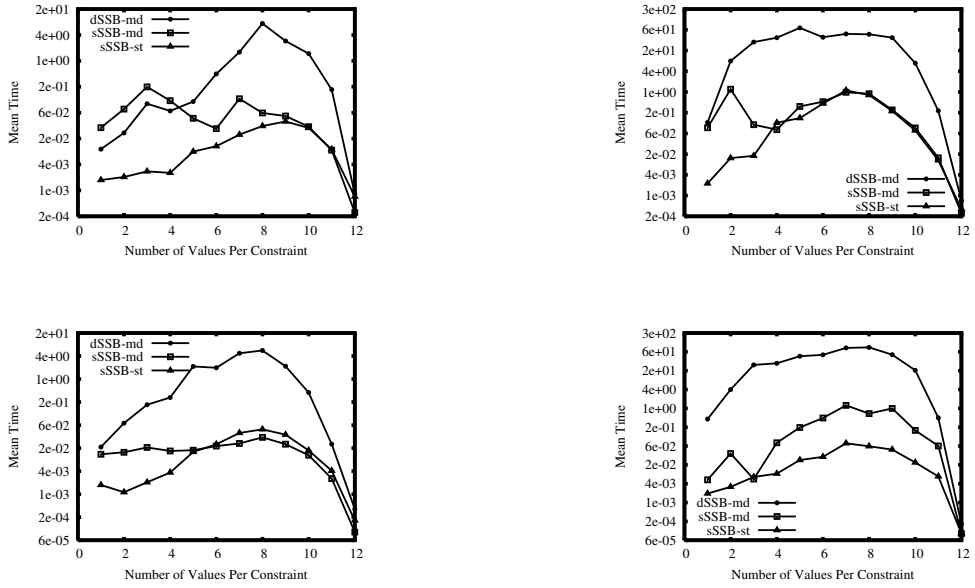

Figure 3: The figures give mean times in seconds (log-scale) on 100 instances with 15 variables and values, 12 variables per constraint, unbiased (top) or biased (bottom) variable selection, and AllDifferent (left) or GCC (right) as constraint over all variables and values. The cutoff was set to 600 seconds.

dynamic variable ordering. Even though we observe instances in which dynamic search ordering is beneficial, overall we believe that deviating from a static search-ordering when using static SSB is a bad idea.

In Figure 4 we compare the performance of our static SSB, with model restarts (sSSB-gcc-res), the static SSB variant from [7], which utilized a min-domain heuristic (sSSB-reg-md, curves taken from the paper), and the dynamic SSB, again utilizing a min-domain heuristic (dSSB-md). We observe that the dynamic method is orders of magnitude worse than both static techniques. We also observe that our static SSB method works somewhat better than the variant discussed in [7].

5 Conclusions

We observe that despite the seemingly obvious benefits offered by dynamic symmetry breaking, as opposed to static symmetry breaking, static symmetry breaking

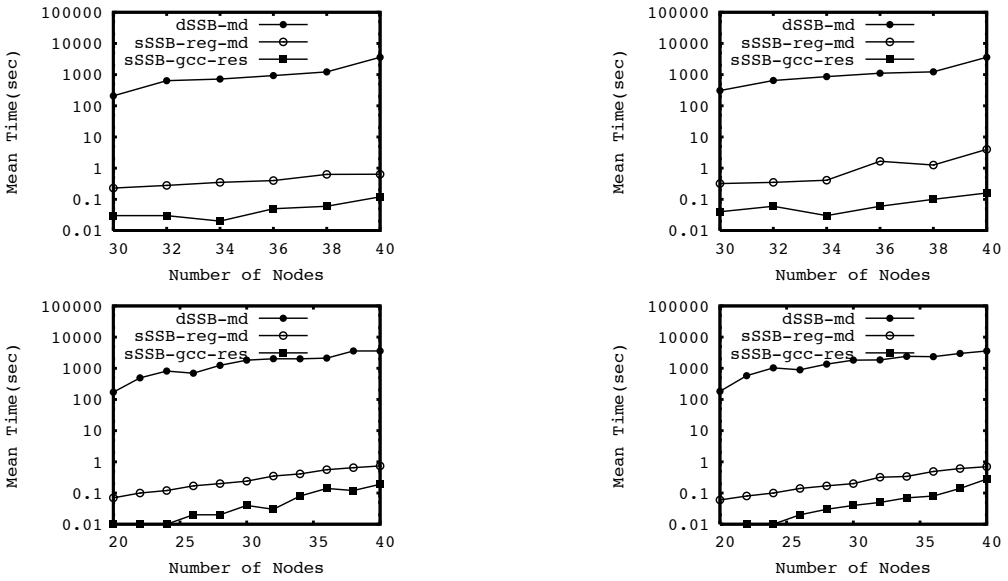

Figure 4: We give mean time (in seconds, log-scale) on 100 instances of the uniform (top) and biased (bottom) graph-coloring benchmark with $q = 0.5$ (left) and $q = 1$ (right). The cutoff was set to one hour.

outperforms the dynamic version by orders of magnitude. We observe this in both a random benchmark proposed by us, and graph coloring instances shown in [7].

References

[1] Fahle, T., Schamberger, S., and Sellmann, M. Symmetry breaking. *Proceedings of CP'01* (2001), 93– 107.

[2] Flener, P., Pearson, J., Sellmann, M., and Hentenryck., P. V. Static and dynamic structural symmetry breaking. *Proceedings of CP'06* (2006), 695–699.

[3] Focacci, F., and Milano, M. Global cut framework for removing symmetries. *Proceedings of CP'01* (2001), 77–92.

[4] Gent, I., Harvey, W., Kelsey, T., and Linton, S. Generic sbdd using computational group theory. *Proceedings of CP'03* (2003), 333–347.

[5] Heller, D., Panda, A., Sellmann, M., and Yip, J. The practice of structural symmetry breaking. unpublished.

[6] Hentenryck, P. V., Flener, P., Pearson, J., and Agren., M. Tractable symmetry breaking for csps with interchangeable values. *Proceedings of IJCAI'03* (2003), 277–282.

[7] Law, Y. C., Lee, J., Walsh, T., and Yip, J. Breaking symmetry of interchangeable variables and values. *Proceedings of CP'07* (2007), 423–437.

[8] Margot, F. Exploiting orbits in symmetric ilp. *Mathematical Programming 98*, 1-3 (2003), 3–21.

[9] Puget, J. F. Dynamic lex constraints. *Proceedings of CP'06* (2006), 453–467.

[10] Régin, J. Generalized arc-consistency for global cardinality constraint. In *Proceedings of AAAI'96* (1996), AAAI Press, pp. 209–215.

[11] Roney-Dougal, C., I. Gent, T. K., and Linton, S. Tractable symmetry breaking using restricted search trees. In *Proceedings of ECAI'04* (2004), pp. 211–215.

[12] Sellmann, M., and Hentenryck, P. V. Structural symmetry breaking. *Proceedings of IJCAI'05* (2005), 298–303.