Model Restarts for Structural Symmetry Breaking*

Daniel Heller, Aurojit Panda, Meinolf Sellmann, and Justin Yip

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

There exist various methods to break symmetries. The two that concern us in this paper are static symmetry breaking where we add static constraints to the problem (see e.g. [1,3]) and symmetry breaking by dominance detection (SBDD) where we filter values based on a symmetric dominance analysis when comparing the current search-node with those that were previously expanded [2,5]. The core task of SBDD is dominance detection. The first provably polynomial-time dominance checkers for value symmetry were devised in [18] and [14]. For problems exhibiting both "piecewise" symmetric values and variables, [15] devised structural symmetry breaking (SSB). SSB is a polynomial-time dominance checker for piecewise symmetries which, used within SBDD, eliminates symmetric subproblems from the search-tree. Piecewise symmetries are of particular interest as they result naturally from symmetry detection based on a static analysis of a given CSP that exploits the knowledge about problem substructures as captured in global constraints [17]. Static SSB was developed in [4] and is based on the structural abstractions that were introduced in [17].

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 [12]. 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 will 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 will show

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

P.J. Stuckey (Ed.): CP 2008, LNCS 5202, pp. 539–544, 2008.
© Springer-Verlag Berlin Heidelberg 2008

how they allow us to efficiently combine static symmetry breaking with semi-dynamic search-orderings. The method is very simple to use and we show that model restarts greatly improve the robustness of static symmetry breaking.

1 Static and Dynamic Structural Symmetry Breaking

A *Constraint Satisfaction Problem (CSP)* is a tuple (Z, V, D, C) where Z is a finite set of variables, V is a set of values, $D = \{D_1, \ldots, D_n\}$ is a set of finite domains where each $D_i \subseteq V$ is the set of possible instantiations to variable $X_i \in Z$, and $C = \{c_1, \ldots, c_p\}$ 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 set S and a set of sets $P = \{P_1, \ldots, P_r\}$ 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 $S = \sum_i P_i$. 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.

As mentioned in the introduction, 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 setting up a bipartite graph and pruning the current node if and only if a perfect matching can be found. In [15] it was shown how dynamic SSB can be used for filtering 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.

Static SSB is based on the abstractions used by the dominance checker in dynamic SSB. In [4] a linear set of (global) constraints was devised which provably leaves one and only one solution in each equivalence class of solutions. Using Regin's filtering algorithm [13] for the global cardinality constraints (GCCs) in this set, filtering all static SSB constraints does not take longer than amortized $O(\sum_{k=1}^{a} |P_k|^2 m) = O(n^2 m)$, where m is the number of values and n the number of variables in the given CSP.

2 Model Restarts

As the runtime comparison shows, static symmetry-breaking imposes much less overhead. However, it suffers from one important drawback, and that is the fact that it is much more sensitive to search-orderings. Both in [10] and [16] it has been noted that static symmetry-breaking constraints can cause great variances in 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. 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 [7,9], 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 [9]. 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. The method has been proven both experimentally and theoretically to eliminate heavy-tailed run-time distributions [7]. 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.

To improve on 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., [16]). In our case, with respect to the ordering of variables, we have the freedom to arbitrarily choose the ordering of the variable partitions. We start our search and use the static variable ordering as induced by the ordering of variable partitions used in the static symmetry breaking constraints. This avoids clashes between static SSB and the search ordering used. 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."

3 Experimental Results

We experiment on the benchmark introduced in [11] that consists of graph coloring problems over symmetric graphs. Colors are interchangeable values and nodes that have the same set of neighbors form a partition of piecewise interchangeable variables. 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, see [11]). For reasons of comparability, our experimental set-up is the same as in [11]. Each data point we report represents the

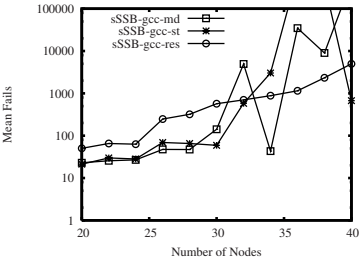

Fig. 1. Mean number of fails (log-scale) for the biased graph-coloring benchmark (q=1)

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 use the same CSP model as in [11].

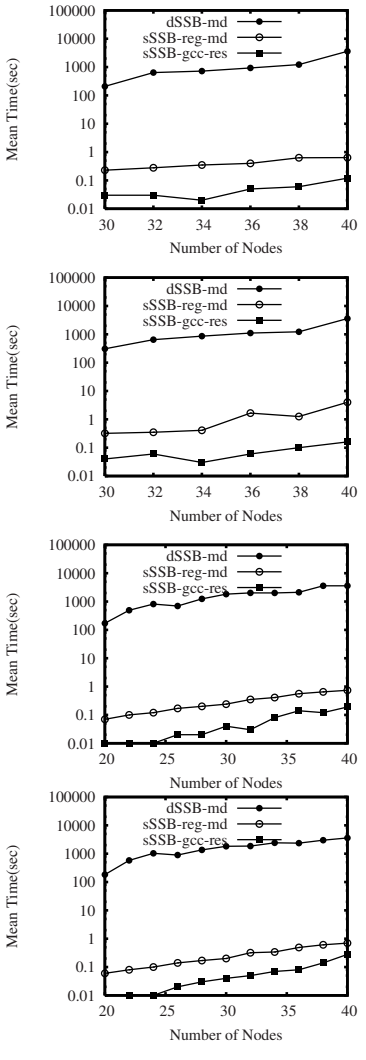

Fig. 2. Mean times (seconds, log-scale) for the uniform (top two) and biased (bottom two) graph-coloring benchmark with $q = 0.5$ (1st and 3rd) and $q = 1$ (2nd and 4th)

In Figure 1, 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 2). We observe that sSSB-gcc-md and sSSB-gcc-st are both not robust at all. Due to a high variance in runtime 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 of sSSB.

In Figure 2 we compare restarted static SSB based on GCC constraints (sSSB-gcc-res) with dynamic SSB (with min-domain heuristic, dSSB-md) and a different variant of static SSB based on regular constraints that was introduced in [11] (with min-domain heuristic, sSSB-reg-md). The first two algorithms were 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 [11]. 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 [11] 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. In [11] 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), the number of failures of sSSB-gcc-res is shown in Figure 1. Consequently, sSSB-gcc-res visits about two orders of magnitude more choice points than sSSB-reg. This implies that sSSB-gcc works almost three orders of magnitude faster per choice point. This efficiency gives the simple sSSB-gcc-res the advantage over the much more effective filtering of sSSB-reg-md. When comparing with dynamic SSB, finally, as the theoretical runtime comparison of dSSB and sSSB in Section 1 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 [8].

In Figure 3 we compare dSSB with sSSB on a different benchmark where we generate random AllDifferent constraints which each partition a set of 12 values and 15 variables, thus leaving a piecewise symmetric CSP. By varying the number of values per constraint, we achieve a range of more and more restricted piecewise symmetric problems which allows us to compare methods over an entire regime of constrainedness. Again, we see that static symmetry breaking vastly outperforms dynamic symmetry breaking. Although we cannot show the result of those tests here, we would also like to note that restarts do not lead to performance improvements for dynamic SSB.

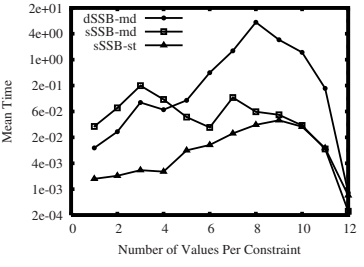

Fig. 3. Mean times (seconds, log-scale, cutoff 600 seconds) on 100 random AllDiff-CSP instances

We conclude that dynamic SSB for piecewise symmetry is inferior to its static counterpart. Moreover, we found that static SSB based on GCC constraints is far less effective than static SSB based on regular constraints as it visits many more choice points. However, its extremely low cost per choice points causes it to run faster, and when used with model restarts it works equally robustly.

References

1. Crawford, J., Ginsberg, M., Luks, E., Roy, A.: Symmetry-breaking predicates for search problems. In: KR, pp. 149–159 (1996)
2. Fahle, T., Schamberger, S., Sellmann, M.: Symmetry Breaking. In: Walsh, T. (ed.) CP 2001. LNCS, vol. 2239, pp. 93–107. Springer, Heidelberg (2001)
3. Flener, P., Frisch, A., Hnich, B., Kiziltan, Z., Miguel, I., Pearson, J., Walsh, T.: Breaking row and column symmetries in matrix models. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 462–476. Springer, Heidelberg (2002)
4. Flener, P., Pearson, J., Sellmann, M., Van Hentenryck, P.: Static and Dynamic Structural Symmetry Breaking. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 695–699. Springer, Heidelberg (2006)

5. Focacci, F., Milano, M.: Global cut framework for removing symmetries. In: Walsh, T. (ed.) CP 2001. LNCS, vol. 2239, pp. 77–92. Springer, Heidelberg (2001)
6. Gent, I., Harvey, W., Kelsey, T., Linton, S.: Generic SBDD using computational group theory. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 333–347. Springer, Heidelberg (2003)
7. Gomes, C.P., Selman, B., Crato, N.: Heavy-Tailed Distributions in Combinatorial Search. In: CP, pp. 121–135 (1997)
8. Heller, D., Sellmann, M.: Dynamic Symmetry Breaking Restarted. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 721–725. Springer, Heidelberg (2006)
9. Kautz, H., Horvitz, E., Ruan, Y., Gomes, C., Selman, B.: Dynamic Restart Policies. In: AAAI, pp. 674–682 (2002)
10. Kiziltan, Z.: Symmetry Breaking Ordering Constraints. PhD Thesis (2004)
11. Law, Y.-C., Lee, J., Walsh, T., Yip, J.: Breaking symmetry of interchangeable variables and values. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 423–437. Springer, Heidelberg (2007)
12. Puget, J.F.: Dynamic Lex Constraints. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 453–467. Springer, Heidelberg (2006)
13. Régin, J.-C.: Generalized arc-consistency for global cardinality constraint. In: AAAI, pp. 209–215. AAAI Press, Menlo Park (1996)
14. Roney-Dougal, C., Gent, I., Kelsey, T., Linton, S.: Tractable symmetry breaking using restricted search trees. In: ECAI, pp. 211–215 (2004)
15. Sellmann, M., Van Hentenryck, P.: Structural Symmetry Breaking. In: IJCAI, pp. 298–303 (2005)
16. Smith, B.M.: Sets of Symmetry Breaking Constraints. In: SymCon 2005 (2005)
17. Van Hentenryck, P., Flener, P., Pearson, J., Agren, M.: Compositional derivation of symmetries for constraint satisfaction. In: Zucker, J.-D., Saitta, L. (eds.) SARA 2005. LNCS (LNAI), vol. 3607, pp. 234–247. Springer, Heidelberg (2005)
18. Van Hentenryck, P., Flener, P., Pearson, J., Agren, M.: Tractable symmetry breaking for CSPs with interchangeable values. In: IJCAI, pp. 277–282 (2003)