Proc. SymCon'03

Using Constraint Programming to Compute Symmetries

Jean-François Puget

ILOG, 9 avenue de Verdun, 94253 Gentilly, France,
puget@ilog.fr

Abstract. Symmetry breaking methods in constraint programming rely more and more on an explicit definition of the symmetry group. Several methods take as input a set of strong generator for the symmetry group, see [6], [7] for instance. These generators are then used to compute various useful information using computational group theory (CGT). Other methods use a second CSP to compute the same information. For instance, the SBDD method can use an auxiliary CSP to compute dominance relation [3], [8]. We present in this paper a thorough explanation of how we use an auxiliary CSP to search for useful symmetries, in the context of the STAB method [11]. We also show how this method can be significantly improved using a little CGT.

1 Introduction

Computing with symmetries is an essential ingredient of symmetry breaking methods. For instance, the SBDD method[5] [3] requires to solve the following dominance problem: given a past state A of the search and the current state B, is there a symmetry σ of the problem such that $\sigma(A)$ is included in B? This sub problem can be solved with a description of the symmetry group [7]. Others have explored the use of an auxiliary CSP [8] [10]. We will explore the latter approach in this paper in the context of the STAB method [11]. This method requires the computation of the set of symmetries that leave a given partial assignment unchanged. Such sets are called stabilizers in group theory. We will show how to use an auxiliary CSP for the computation of stabilizers. We will also show that our auxiliary CSP ca be used to compute generators of the stabilizers instead of all their elements. An experimental evaluation will show that stating symmetry breaking constraints using generators is more efficient in practice than the original STAB method.

The rest of the paper is organized as follows. Section 2 presents the STAB method briefly. Section 3 presents the CSP used for the computation of stabilizers. Section 4 shows how to modify the CSP in order to compute only generators for stabilizers. Section 5 contains an experimental evaluation using BIBDs. Last, section 6 contains a discussion of the results obtained so far as well as an indication of future research topics.

2 The STAB method

We will only consider variable permutations. We will define a symmetry σ by how it maps any partial assignment A into another partial assignment $\sigma(A)$.

In [2], it is shown that any CSP can be turned into a CSP without symmetries by the addition of the following constraints to the CSP.

$$V \leq_{Lex} \sigma(V), \text{ for all } \sigma \in G \qquad (1)$$

where $\leq_{LEX}$ stands for lexicographic ordering.

Although very appealing, this technique is not scalable because of the potentially large size of the group G. Indeed, we exhibit in section 5 a problem that has more than 10^{105} symmetries.

Given a partial assignment A, let us define the set of symmetries that leave A unchanged:

$$stab(A) = \{\sigma \in G \mid \sigma(A) = A\}$$

This set is called the *stabilizer* of A, and it is a subgroup of G. Moreover, its size divides the size of G. In practice the size of the stabilizers is often much smaller than the size of G. The STAB method [11] amounts to add the following set of constraints at each node of the search tree:

$$V \leq_{Lex} \sigma(V), \text{ for all } \sigma \in stab(A) \qquad (2)$$

or some subset of these constraints.

3 Computing stabilizers

There exists algorithms in the computation group theory that efficiently compute stabilizers if the symmetry group is given as input. These algorithms could be used to implement STAB. We chosed to compute directly the stabilizers from the partial assignments A because we think that providing the symmetry group as input may be too complex for users. We will consider symmetrical matrix problems[1] from now on, but the idea of using an auxiliary CSP to compute symmetries can be used for other problems.

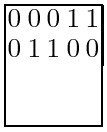
3.1 An example

Let us look at an example to explain how STAB works in more detail. For instance let us consider a 4×5 matrix model. The matrix of variables is:

$$\begin{array}{|ccccc|}\hline x_1 & x_2 & x_3 & x_4 & x_5 \\ x_6 & x_7 & x_8 & x_9 & x_{10} \\ x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ x_{16} & x_{17} & x_{18} & x_{19} & x_{20} \\ \hline\end{array}$$

[1] A symmetrical matrix model is a model where the variables can be arranged into a matrix such that any row permutation or any column permutation is a symmetry.

Let us consider the partial assignment A where the first 10 variables, are assigned values in the following way.



Every symmetry for A is defined by a row permutation $\psi \in S^4$ and a column permutation $\phi \in S^5$ such that $A = A(\psi, \phi)$. We will see later how to compute these symmetries. They are listed below.

$$\begin{array}{l}
\sigma_1 = (\psi_1, \phi_1), \ \ \psi_1 = [1,2,3,4], \phi_1 = [1,2,3,4,5] \\
\sigma_2 = (\psi_1, \phi_2), \ \ \psi_1 = [1,2,3,4], \phi_2 = [1,3,2,4,5] \\
\sigma_3 = (\psi_1, \phi_3), \ \ \psi_1 = [1,2,3,4], \phi_3 = [1,2,3,5,4] \\
\sigma_4 = (\psi_1, \phi_4), \ \ \psi_1 = [1,2,3,4], \phi_4 = [1,3,2,5,4] \\
\sigma_5 = (\psi_2, \phi_5), \ \ \psi_2 = [2,1,3,4], \phi_5 = [1,4,5,2,3] \\
\sigma_6 = (\psi_2, \phi_6), \ \ \psi_2 = [2,1,3,4], \phi_6 = [1,4,5,3,2] \\
\sigma_7 = (\psi_2, \phi_7), \ \ \psi_2 = [2,1,3,4], \phi_7 = [1,5,4,2,3] \\
\sigma_8 = (\psi_2, \phi_8), \ \ \psi_2 = [2,1,3,4], \phi_8 = [1,5,4,3,2] \\
\sigma_9 = (\psi_3, \phi_1), \ \ \psi_3 = [1,2,4,3], \phi_1 = [1,2,3,4,5] \\
\sigma_{10} = (\psi_3, \phi_2), \ \psi_3 = [1,2,4,3], \phi_2 = [1,3,2,4,5] \\
\sigma_{11} = (\psi_3, \phi_3), \ \psi_3 = [1,2,4,3], \phi_3 = [1,2,3,5,4] \\
\sigma_{12} = (\psi_3, \phi_4), \ \psi_3 = [1,2,4,3], \phi_4 = [1,3,2,5,4] \\
\sigma_{13} = (\psi_4, \phi_5), \ \psi_4 = [2,1,4,3], \phi_5 = [1,4,5,2,3] \\
\sigma_{14} = (\psi_4, \phi_6), \ \psi_4 = [2,1,4,3], \phi_6 = [1,4,5,3,2] \\
\sigma_{15} = (\psi_4, \phi_7), \ \psi_4 = [2,1,4,3], \phi_7 = [1,5,4,2,3] \\
\sigma_{16} = (\psi_4, \phi_8), \ \psi_4 = [2,1,4,3], \phi_8 = [1,5,4,3,2]
\end{array}$$

3.2 Computing matrix automorphisms

From the example above, it should be clear that the stabilizer is the group of the automorphisms of the matrix A representing the partial assignment.

In order to compute it, we construct a labeled graph $g(A)$ whose nodes are the rows and the columns of A. There exists an arc between row i and column j with label $A[i,j]$. Then the symmetry group of this graph is what we are looking for. We can use an auxiliary CSP to compute it.

Given a graph with labeled edges, we construct a CSP as follows. There is one variable y_i per node i. The domain of the variables are the set of nodes. There are two constraints. First of all, the variables are all different. Second, there is a constraint stating that neighbors are mapped onto neighbors. The rest of this section describes the second constraint into detail.

If i is a node, and a a label, let $\Gamma^a(i)$ be the set of nodes j such that there exists an arc labeled a whose ends are i and j. If $dom(y_i)$ is the domain of y_i, let $\Gamma^a(y_i)$ be,

$$\Gamma^a(y_i) \ = \ \bigcup_{j \in dom(y_i)} \Gamma^a(j) \qquad (3)$$

Then the neighbor constraint says that

$$y_j \in \Gamma^a(y_i) \text{ for all } j \in \Gamma^a(i) \qquad (4)$$

The propagation of this constraint is straightforward. The sets $\Gamma^a(y_i)$ are maintained incrementally. Whenever they are reduced, the above condition is used to reduce the domains of the variables y_j for each j neighbor of i.

3.3 Revisiting the example

Let us look at our example to see how the neighbor constraint work. We consider the following matrix. The stabilizer is in fact the product of the automorphism group of this matrix and the set of permutations of rows 3 and 4.

$$\begin{array}{|ccccc|}\hline 0&0&0&1&1\\ 0&1&1&0&0\\ \hline\end{array}$$

We create one node for each row and one for each column. Let integers 1 to 5 represent the columns, and let 6 and 7 represent the rows. For each node we create a variable. Let y_1, y_2, y_3, y_4 and y_5, be the variables corresponding to the columns, and y_6 and y_7 be the variables corresponding to the rows.

The sets $\Gamma^a(i)$ are

$$\begin{array}{llll} \Gamma^0(1) = \{6,7\} & \Gamma^1(1) = \{\} \\ \Gamma^0(2) = \{6\} & \Gamma^1(2) = \{7\} \\ \Gamma^0(3) = \{6\} & \Gamma^1(3) = \{7\} \\ \Gamma^0(4) = \{7\} & \Gamma^1(4) = \{6\} \\ \Gamma^0(5) = \{7\} & \Gamma^1(5) = \{6\} \\ \Gamma^0(6) = \{1,2,3\} & \Gamma^1(6) = \{4,5\} \\ \Gamma^0(7) = \{1,4,5\} & \Gamma^1(7) = \{2,3\} \end{array}$$

Then we can compute the sets $\Gamma^a(y_i)$ using (3). For instance, we have for the first row:

$$\Gamma^1(y_6) = \bigcup_{j \in dom(y_6)} \Gamma^0(j)$$

i.e.

$$\Gamma^1(y_6) = \{2,3,4,5,6,7\}$$

Then, using (4), we have that

$$y_j \in \Gamma^1(y_6) \text{ for all } j \in \Gamma^1(6)$$

i.e.

$$y_4, y_5 \in \{2,3,4,5,6,7\}$$

Similar deductions plus the fact that the variables are all different enables to deduce that $y_1 = 1$. From this, the set $\Gamma^0(y_1)$ becomes $\{6,7\}$. Applying (4) gives

$$y_6, y_7 \in \{6,7\}$$

Upon completion of the propagation of the constraints, the domains are

$$\begin{gathered} y_1 = 1 \\ y_2, y_3, y_4, y_5 \in \{2,3,4,5\} \\ y_6, y_7 \in \{6,7\} \end{gathered}$$

Similar propagations are made during the search for all the solutions of this auxiliary CSP. This computes the following 8 solutions. We list the values for the variables y_i for each solution:

$$\begin{gathered} [1,2,3,4,5,6,7] \\ [1,3,2,4,5,6,7] \\ [1,2,3,5,4,6,7] \\ [1,3,2,5,4,6,7] \\ [1,4,5,2,3,7,6] \\ [1,4,5,3,2,7,6] \\ [1,5,4,2,3,7,6] \\ [1,5,4,3,2,7,6] \end{gathered}$$

This method is quite efficient for computing graph automorphisms. In the experiments described in section 5, the number of failed nodes explored when solving the auxiliary CSP is less than the number of automorphisms found. The time spent in the search for automorphisms ranges from 20 to 80 percents of the total running time.

3.4 Aggregating identical column

For matrix models, the complexity of symmetry breaking constraints can be further simplified. We can replace identical adjacent columns by a single representative. Indeed, these columns can be freely permuted. We also assume that the columns and the rows are lexicographically ordered, which implies that identical lines are adjacent.

Let us look at our example. The idea is to replace each group of adjacent columns in V by a single representative, and to replace entries in the matrix by the original value and a cardinality. The A matrix becomes

$$\begin{array}{|ccc|} \hline (0,1) & (0,2) & (1,2) \\ (0,1) & (1,2) & (0,2) \\ \hline \end{array}$$

Then we can use the process described earlier for constructing and solving an auxiliary CSP. In our example there are only two symmetries for the reduced matrix.

$$\sigma_{17} = (\psi_5, \phi_9),\ \psi_5 = [1,2],\ \phi_9 = [1,2,3]$$
$$\sigma_{18} = (\psi_6, \phi_{10},\ \psi_6 = [2,1],\ \phi_{10} = [1,3,2]$$

The original symmetries can then be reconstructed by composing those with the permutations of identical columns. The idea can be extended to collapsing identical rows as well.

For instance, let us consider $\sigma_1 8$. The second and third columns in the aggregated matrix are swapped. This means that in the original matrix, the group of columns corresponding to the second column of the aggregated matrix should be swapped with the group corresponding to the third column of the aggregated matrix. This means that $\sigma_1 8$ stands for all the permutations of the original matrix where the second and third columns are swapped with the columns 4 and 5, and where the two lines are exchanged. These permutations are σ_5, σ_6, σ_7, σ_8, $\sigma_1 3$, $\sigma_1 4$, $\sigma_1 5$, $\sigma_1 6$.

4 STAB with generators

In [1], the authors suggest to only state constraints (1) for group generators, as it leads to a much smaller set of constraints while breaking many symmetries. A similar idea can be used in our setting. Let's look at our example again. Given the partial assignment A, we computed the automorphism group $Aut(A)$ of the partial matrix containing the rows for which variables have been assigned a value. The STAB method then states symmetry breaking constraints for each automorphism found [11].

4.1 The Schreier Sims representation

We can modify the STAB method by using a set of generators for $Aut(A)$ instead of $Aut(A)$ itself. It is easy to modify the CSP approach we described in the previous section to compute a Schreier Sims representation for this group[**?**]. This representation is based on a chain of stabilizers and coset representatives. We define $Stab_g(i)$ to be the stabilizer of i in the group G.

Let us look at our example again. There are 8 permutations in the automorphism group G:

$$[1,2,3,4,5,6,7]$$
$$[1,3,2,4,5,6,7]$$
$$[1,2,3,5,4,6,7]$$
$$[1,3,2,5,4,6,7]$$
$$[1,4,5,2,3,7,6]$$
$$[1,4,5,3,2,7,6]$$
$$[1,5,4,2,3,7,6]$$
$$[1,5,4,3,2,7,6]$$

The first step is to consider the stabilizer of the first node.
$Stab_G(1) = G$

In this case, all permutations fix i. We then compute the stabilizer of the second node in this stabilizer (let $G_1 = Stab_G(1)$):

$Stab_{G_1}(2) = [1,2,3,4,5,6,7], [1,2,3,5,4,6,7]$

We partition the permutations of G_1 according to the element to which 2 is mapped (Let $G_2 = Stab_{G_1}(2)$):

$C_2 = G_2$
$C_3 = [1,3,2,4,5,6,7], [1,3,2,5,4,6,7]$
$C_4 = [1,4,5,2,3,7,6], [1,4,5,3,2,7,6]$
$C_5 = [1,5,4,2,3,7,6], [1,5,4,3,2,7,6]$

and we select one permutation from each of C_3, C_4, and C_5. Those sets are called right cosets for G_2 in G_1.

We then continue with the next node. We compute the stabilizer G_3 of the third node in G_2:

$G_3 = Stab_{G_2}(3) = G_2$

Since $G_3 = G_2$ cosets are empty. We compute the stabilizer G_4 of the third node in G_3:

$G_4 = Stab_{G_3}(4) = [1,2,3,4,5,6,7]$

There is one coset:

$C_5 = [1,2,3,5,4,6,7]$

The representation is complete because we are only left with the identity permutation.

The Schreier Sims representation is the set of coset representatives, plus the identity, i.e.:

$$[1,2,3,4,5,6,7]$$
$$[1,2,3,5,4,6,7]$$
$$[1,3,2,4,5,6,7]$$
$$[1,4,5,2,3,7,6]$$
$$[1,5,4,2,3,7,6]$$

4.2 Computing the Schreier Sims Representation

There is a simple way to compute this representation using the CSP. Remember that the permutations are obtained as the values of the variables y_i for each solution of the CSP.

Let $G_0 = G$, and $G_i = Stab_{G_{i-1}}(i)$

The step is to see that G_{i-1} is the set of the permutations that fix all the nodes 1 to $i-1$ included. They correspond to solutions where

$y_j = j$ for all $j < i$

Then we must select one coset representative for each coset in G_{i-1}. This means that we must find one permutation for each k such that there exist a permutation in G_{i-1} that maps i to k. That is, for each k we search for a solution that extends the partial assignment :

$y_j = j$ for all $j < i$
$y_i = k$ such that $k \neq j$

The algorithm for finding a Schreier Sims representation is then straightforward. We first generate all partial assignments such that there exists i and k such that

$$\begin{gathered} y_j = j \text{ for all } j < i \\ y_i = k \text{ such that } k \neq j \\ y_j \text{ aren't assigned a value, for all } j > i \end{gathered}$$

Then we try to extend each of these partial assignment to a solution. We only need one solution for each partial assignment.

The set of all the solutions obtained this way is the Schreier Sims representation we are looking for. This modified search is easy to implement in any modern CP system.

For instance, in our example, we will only produce the following 5 solutions. Remember that the full automorphism group contains 8 elements.

$$\begin{gathered} [1,2,3,4,5,6,7] \\ [1,2,3,5,4,6,7] \\ [1,3,2,4,5,6,7] \\ [1,4,5,2,3,7,6] \\ [1,5,4,2,3,7,6] \end{gathered}$$

4.3 The STABGEN method

We can then modify the STAB method to only state the symmetry breaking constraints for the permutations appearing in the Schreier Sims representation of the automorphism group. Let's call STABGEN the resulting modified STAB method.

5 Experimental results

In order to evaluate the effectiveness of STABGEN, we chose the BIBD problems studied in [11]. We are using the same CSP representation, and the same value and variable orderings for the search.

We report running times to find all solutions in Table 1. We report the times obtained for three methods: ordering both row and columns (Lex^2)[4], the STAB method (Stab), and the STABGEN method (*StabGen*). For each method we report the running time, in seconds, on a 1.4 GHz Pentium Mobile laptop with 1GB of memory. We also report the number of solutions found and the number of symmetry constraints added to the original CSP.

		Lex^2			Stab			StabGen		
v k λ	$\sharp sym$	time	$\sharp sols$	$\sharp ct$	time	$\sharp sols$	$\sharp ct$	time	$\sharp sols$	$\sharp ct$
6 3 8	5.9e+50	0.47	494	44	0.1	15	206	0.1	17	144
9 4 3	2.3e+21	1.3	2600	25	0.11	41	537	0.1	41	379
16 6 2	4.3e+26	0.33	46	30	0.34	3	11543	0.15	3	1636
7 3 4	1.5e+33	0.69	3209	33	0.16	116	537	0.17	123	400
21 5 1	2.6e+39	0.28	12	40	0.95	1	37050	0.2	1	2461
6 3 10	2.2e+67	1.3	1366	54	0.22	26	222	0.24	30	178
9 3 2	2.2e+29	0.82	5987	31	0.34	344	1895	0.36	383	1413
15 5 2	6.6e+31	6.9	0	34	0.37	0	3006	0.38	0	1700
13 3 1	2.5e+36	4.6	12800	37	0.39	21	2045	0.41	26	1656
15 7 3	1.7e+24	0.5	118	28	0.54	19	12689	0.42	19	4680
25 5 1	4.1e+57	21.6	864	53	0.94	1	30690	0.42	1	3674
22 7 2	1.2e+42	10.8	0	42	3.6	0	94148	0.43	0	4567
7 3 5	5.2e+43	6.9	33304	40	0.47	542	1086	0.46	555	729
10 5 4	2.3e+22	11.8	8031	26	0.94	302	2843	0.94	302	2547
31 6 1	6.7e+67	53	864	60	17.7	1	743434	1.47	1	9158
7 3 6	7.0e+54	64	250878	47	1.5	2334	1816	1.6	2574	1549
7 3 7	3.1e+66	460	1459585	54	6.1	8821	3286	6.8	9905	3049
8 4 6	1.2e+34	605	2058523	34	7.7	17890	7210	8.1	18773	6763
10 3 2	9.6e+38	83	724662	38	9.5	24563	26204	10.1	26683	26247
19 9 4	1.4e+34	544	6520	36	9.6	71	46981	10.7	79	42326
43 7 1	3.6e+105	>120000			>100000		>5400000	20	0	78642
7 3 8	3.6e+78	2664	6941124	61	25	32038	6863	28	35613	6589
36 6 1	5.2e+92	95057	0	76	684	0	963630	42	0	178347
9 3 3	1.3e+47	1871	14843772	43	76	315531	69458	85	344543	78186
29 8 2	7.8e+61	30401	0	56	4545	0	1131265	89	0	328703
7 3 9	1.0e+91	13059	38079394	68	98	105955	13695	113	120201	13373
15 3 1	1.3e+52	13522	32127296	48	128	6782	280361	167	11187	368468
21 6 2	1.6e+49	26654	0	47	325	0	477517	261	0	654662
13 4 2	2.5e+36	15139	3664242	37	410	83337	811396	448	92005	878793
11 5 4	4.5e+28	15734	6142308	31	481	106522	557127	507	113305	580824

We can see that STABGEN is always faster than Lex^2, which is not true of the original STAB method. STABGEN is about 35 times faster on average[2] than Lex^2 when the latter can solve the problem. We can also see that STABGEN is about 2 times faster on the average than STAB. It is also more regular. Indeed, when STABGEN is slower than STAB, it is about 10 percents slower. However, when STABGEN is faster than STAB, it is about 6 times faster on average! Memory consumption is not shown, but we see that the number of constraints stated by STABGEN is very often smaller than for STAB. When it is greater, it is not by a big margin. STABGEN may state more constraints than STAB because it prunes less nodes, therefore the increase in the number of explored nodes may offset the reduction in the number of constraints stated at each node.

[2] We use geometric means instead of the usual arithmetic mean. This avoids to bias results towards the exceptional values.

Last, we can see that the number of solutions found by STABGEN is greater than for STAB, but not more than twice as large.

We also collected information about the computation one solution. We only report conclusions here because of lack of space. It has been shown in [11] that STAB and Lex^2 have the same efficiency. The STABGEN method is about 40 percents faster on average than both methods for finding the first solution.

6 Conclusion

We presented how to use an auxiliary CSP to compute the stabilizers used in the STAB method. We discussed a number of improvements as well as a new global constraint enforcing neighbor conditions. We aslo showed that using a little computation group theory, one could significantly reduce the number of symmetry constraints added by the STAB method. An experimental comparison shows that the resulting STABGEN method is much more efficient and scalable than previously published methods.

We plan to generalize the STABGEN method to problems others than matrix problems. We also want to extend the method in order to take into account value symmetries.

References

[1] F.A. Aloul, I.L. Markov, and K.A. Sakallah. Symmetry Breaking for Boolean Satisfiability: The Mysteries of Logic Minimization. In proceedings of SymCon'02.
[2] Crawford, J., Ginsberg, M., Luks E.M., Roy, A. Symmetry Breaking Predicates for Search Problems. In proceedings of KR'96, 148-159.
[3] Fahle, T., Shamberger, S., Sellmann, M. Symmetry Breaking. Proceedings of CP01 (2001) 93-107.
[4] P. Flener, A. M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, T. Walsh. Breaking Row and Column Symmetries in Matrix Models. Proceedings of CP'02, pages 462-476, 2002
[5] Focacci, F., Milano, M.: Global Cut Framework for Removing Symmetries. Proceedings of CP'01 (2001) 75-92.
[6] Gent, I.P., Harvey, W., and Kelsey, T.: Groups and Constraints: Symmetry Breaking during Search. Proceedings of CP'02, 415-430.
[7] Gent, I.P., Harvey, W., Kelsey, T., and Linton, S.: Generic SBDD using Computational Group Theory. To appear in proceedings of CP'03
[8] Harvey, W.: Symmetry breaking and the social golfer problem. Proceedings of SYMCON'01, 9-16.
[9] Donald L. Kreher, Douglas R. Stinson Combinatorial Algorithms: Generation, Enumeration, and Search CRC press, 1998.
[10] Puget, J.-F.: Symmetry Breaking Revisited. Proceedings of CP'02, 446-461.
[11] Puget, J.-F.: Symmetry Breaking Using Stabilizers. To appear in proceedings of CP'O3.