

Future and trends of Constraint Programming

Frédéric BENHAMOU
Narendra JUSSIEN
Barry O'SULLIVAN

December 10, 2006

Contents

Chapter 1. GAPLex: Generalised Static Symmetry Breaking	11
Chris JEFFERSON, Tom KELSEY, Steve LINTON, Karen PETRIE	
1.1. Overview	11
1.2. Background and Introduction	12
1.2.1. Group theory for CSPs	14
1.2.2. Using GAP to break CSP symmetries	14
1.3. GAPLex	15
1.3.1. Motivation and rationale	15
1.3.2. Motivating example	16
1.3.3. The GAPLex algorithms	17
1.4. Empirical evaluation	20
1.4.1. Combining GAPLex with Incomplete Static SB methods	20
1.4.2. Combining GAPLex with Puget’s all-different constraints	21
1.5. Conclusions and Future Work	22
1.6. Bibliography	23

Chapter 1

GAPLex: Generalised Static Symmetry Breaking

1.1. Overview

We introduce GAPLex, a novel method for breaking for breaking symmetries in constraint satisfaction problems. The idea behind GAPLex is to take advantage of recent advances in computational group theory that allow us to efficiently decide whether a set of numbers is minimal with respect to a set of permutations (and, of course, with respect to an ordering relation on the numbers). We apply this decision procedure during the search for solutions to constraint satisfaction problems: any partial assignment can be thought of as a set of literals (a literal is a variable-value pair), and each such assignment is either lex-minimal with respect to the group of symmetries acting on literals, or it isn't. We describe how GAPLex allows us to safely backtrack away from non-minimal assignments and provide information on "near-misses" that allows us to further prune the search tree. We believe that GAPLex combines the best aspects of breaking symmetries by lex-ordering – simplicity and effective propagation – with the best aspects of dynamic, computational group theory-based symmetry breaking methods – general applicability, soundness and completeness. Empirical evidence to support this belief is provided and discussed, along with an outline of future avenues of related research.

Chapter written by Chris JEFFERSON and Tom KELSEY and Steve LINTON and Karen PETRIE.

1.2. Background and Introduction

Constraint satisfaction problems (CSPs) are often highly symmetric. Given any solution, there are others which are equivalent in terms of the underlying problem. Symmetries may be inherent in the problem, or be created in the process of representing the problem as a CSP. Without symmetry breaking (henceforth SB), many symmetrically equivalent solutions may be found and, in some ways more importantly, many symmetric equivalent parts of the search will be explored. A SB method aims to avoid both of these problems.

Most SB algorithms fall into one of two groups. The first, known as static SB algorithms, decide before search which assignments will be permitted and which will be forbidden, and are therefore independent of search ordering. The second group, known as dynamic symmetry breaking methods, instead choose which assignments will be permitted during search.

Existing static symmetry breaking methods work by adding extra constraints to the CSP. This typically involves constraints that rule out solutions by enforcing some lexicographic-ordering on the variables of the problem [CRA 96, FRI 02]. STAB [PUG 03] is a SB algorithm for solving BIBDs which avoids adding an exponential number of constraints at the start of search by adding constraints during search on the row of the BIBD currently being assigned.

Dynamic symmetry breaking methods operate in a number of ways, including posting constraints that rule out search at states that are symmetrically equivalent to the current assignment [BAC 99, GEN 00], building the search tree such that symmetric nodes are avoided [RON 04] or backtracking from nodes that are symmetrically equivalent to root nodes of subtrees that have previously been fully explored [FAH 01, FOC 01, PUG 03]. These methods, especially in the case that symmetries are represented by permutation groups, are related to a class of algorithms first described in [BRO 88].

The major weakness of static SB methods, compared to dynamic methods, is that they can increase the number of search nodes visited [GEN 02], whenever the first solution which would have been found is forbidden by the symmetry breaking constraints. The major advantage of static symmetry breaking methods is that ad-hoc problem-specific simplifications often perform very well, and powerful implied constraints can be derived from these constraints [FRI 05]. Moreover, since static SB does not depend on previously encountered search nodes, it can be used when searching in parallel on multiple machines.

Other methods of breaking symmetry exist, for example a CSP can be reformulated so that either the number of symmetries is reduced, or a bespoke SB approach can be applied more effectively [KEL 04, MES 01], or both. However, these methods

are largely problem-specific, and we are concerned with general methods that can be applied to any formulation of a class of CSPs.

Permutation groups are the mathematical structures that best encapsulate symmetry. Many powerful algorithms for investigating group-theoretic questions are known, and have been efficiently implemented in systems such as GAP [GAP00] and MAGMA [BOS 93]. We describe the symmetries of a CSP as a permutation group of the literals (variable-value pairs) of the CSP, and obtain information regarding symmetric equivalence of search states from the GAP Computational Group Theory (CGT) system.

Using literals in this way has been successful, but it is easy to lose sight of the fact that each variable can only take one value. Our generalised static symmetry breaking methodology is motivated by the potential for efficiencies induced by the use of this elementary concept. We therefore address the open research question: can we implement lex-ordering with a CGT approach in such a way that we retain the best features of static symmetry breaking while gaining the speed and flexibility of a general group-theoretic framework? Our contribution is twofold. We first describe a novel SB method, GAPLex, which involves both ordering constraints and symmetry information regarding the current state of search to break as many solution symmetries of a CSP as are required. We also demonstrate that GAPLex can be combined with previous fast – but incomplete – static lex-orderings to provide fast and complete SB. We are not aware of any existing SB method that effectively combines static and dynamic approaches, although Harvey has made an interesting initial attempt [HAR 04].

Puget [PUG 93] proved that whenever a CSP has symmetry, it is possible to find a ‘reduced form’, with the symmetries eliminated, by adding constraints to the original problem and showed such a form for three CSPs. Following this, the key advance was to show a method whereby such a set of constraints could be generated. Crawford, Ginsberg, Luks and Roy showed a general technique, called “lex-leader”, for generating such constraints for any variable symmetry [CRA 96]. In later work, Aloul *et al.* also showed how the lex-leader constraints for symmetry breaking can be expressed more efficiently [ALO 03]. This method was developed in the context of Propositional Satisfiability (SAT), but the results can also be applied to CSPs.

The idea behind lex-leader is essentially simple. For each equivalence class of assignments under our symmetry group, we choose one to be canonical. We then add constraints before search starts which are satisfied by canonical assignments and not by any others. We generate canonical assignments by choosing an ordering of the variables and representing assignments as tuples under this variable ordering. Any permutation of variables g maps tuples to tuples, and the lexicographically least of these is our canonical assignment. This gives the set of constraints

$$\forall g \in G, V \preceq_{\text{lex}} V^g$$

where V is the vector of the variables of the CSP, \preceq_{lex} is the standard lexicographic ordering relation, defined by $AD \preceq_{\text{lex}} BC$ iff either $A < B$ or $A = B$ and $D \leq C$, and V^g denotes the permutation of the variables by application of the group element.

1.2.1. Group theory for CSPs

DEFINITION.— A CSP L is a set of constraints \mathcal{C} acting on a finite set of variables $\Delta := \{A_1, A_2, \dots, A_n\}$, each of which has finite domain of possible values $D_i := D(A_i) \subseteq \Lambda$. A solution to L is an instantiation of all of the variables in Δ such that all of the constraints in \mathcal{C} are satisfied.

Constraint logic programming systems typically model CSPs using constraints over finite domains. The usual search method is depth-first, with values assigned to variables at choice points. After each assignment a partial consistency test is applied: domain values that are found to be inconsistent are deleted, so that a smaller search tree is produced.

Statements of the form $(Var = val)$ are called *literals*, so a partial assignment is a conjunction of literals. We denote the set of all literals by χ , and denote variables by Roman capitals and values by lower case Greek letters.

DEFINITION.— Given a CSP L , with a set of constraints \mathcal{C} , and a set of literals χ , a symmetry of L is a bijection $f : \chi \rightarrow \chi$ such that a full assignment A of L satisfies all constraints in \mathcal{C} if and only if $f(A)$ does.

We denote the image of a literal $(X = \alpha)$ under a symmetry g by $(X = \alpha)^g$. The set of all symmetries of a CSP form a *group*: that is, they are a collection of bijections from the set of all literals to itself that is closed under composition of mappings and under inversion. We denote the symmetry group of a CSP by G .

DEFINITION.— Let G be a group of symmetries of a CSP. The stabiliser of a literal $(X = \alpha)$ is the set of all symmetries in G that map $(X = \alpha)$ to itself. This set is itself a group. The orbit of a literal $(X = \alpha)$, denoted $(X = \alpha)^G$, is the set of all literals that can be mapped to $(X = \alpha)$ by a symmetry in G . The orbit of a node is defined similarly.

Given a collection \mathcal{S} of literals, the *pointwise* stabiliser of \mathcal{S} is the subgroup of G which stabilises each element of \mathcal{S} individually. The *setwise* stabiliser of \mathcal{S} is the subgroup of G that consists of symmetries mapping the set \mathcal{S} to itself.

1.2.2. Using GAP to break CSP symmetries

There have been three successful implementations of SB methods which use GAP to provide answers to symmetry related questions during search. All three combined

GAP with the constraint solver ECLⁱPS^e [WAL 97]. GAP-SBDS [GEN 02] is an implementation of symmetry breaking during search: at each search node, constraints are posted which ensure that no symmetrically equivalent node will be visited later in search. The overhead comprises maintenance (in GAP) of a stabiliser chain of the symmetry group, the search for group elements which map the current state to a future state (also in GAP), and the posting of the SB constraints. Enough pruning of the search tree is made, in general, to make GAP-SBDS more efficient than straightforward search. The number of SB constraints is linear in the size of the group, making GAP-SBDS unattractive for groups of size greater than about 10^9 .

GAP-SBDD [GEN 03] uses GAP to check that the next assignment is not equivalent to a state which is the root of a previously explored sub-tree. Again, the overhead of finding (or failing to find) these group elements is usually more than offset by the reduction in search due to early backtracking. Larger groups – up to about 10^{25} – can be dealt with, simply because the answer from GAP is a straight yes or no to the dominance question; the overhead of passing constraint information is not present. GAP also reports literals that can be safely deleted because setting them would have lead to dominance. Provided that the cost of computing these safe deletions is low enough, the domain reductions are a gain over not making them. We follow the same idea in this paper; we search for literals that, if set, would have lead to a non-lex-least assignment.

The third use of GAP is the building of search trees that, by construction, have no symmetrically equivalent nodes and contain a member from each solution equivalence class: GE-trees [KEL 04]. In the event that all the symmetries act only on variables (either by suitable formulation or the use of channelling constraints from an unsuitable formulation), these trees can be constructed in low-degree polynomial time per node. This is due to pointwise stabilisers being polynomial time obtainable, whilst setwise stabilisers (the general case) are not known to be obtainable in polynomial time.

In this paper we aim to build upon the strengths of these existing frameworks by using lex-ordering as the main SB technique, using GAP to decide if the current partial assignment is lex-smallest of the orbit of the assignment under the symmetry group.

1.3. GAPLex

1.3.1. *Motivation and rationale*

Both lex-ordering and CGT-based SB methods are effective and attractive options for breaking symmetries in CSPs. Our aim is to implement lexicographic static symmetry breaking using the CGT methods used in GAP-SBDS and GAP-SBDD, thus combining useful features of both approaches.

We want to enforce a lex ordering on the literals of a CSP so that only solutions that are minimal in the ordering are returned after complete backtrack search with propagation. Moreover, we want this to work with any symmetry structure induced by the formulation of the CSP.

The key idea is that we can compute the minimum image, under a symmetry group G , of those literals that represent the ground variables in any branch of the search tree. By minimum image, we mean the lex-least ordered list of literals that can be obtained by applying group elements (symmetries) to our set of ground literals. If the minimum image of our current partial assignment is lex-smaller than that assignment, then it is safe to backtrack from the current search node: further search will either fail to find a solution or return a solution that is not the lex-smallest in its equivalence class.

Clearly, the cost of computing minimum images is crucial to our methodology: we are no better off if the time taken is not offset by the induced reduction in search for solutions to CSPs. Recent advances in algorithms for finding minimal images have lead to dramatic improvements in both worst-case and apparent average-case time complexities [LIN 04]. We use these more efficient CGT methods to obtain the minimal images and decide the ordering predicate. As a useful extension to the main technique, we can also use CGT to identify those literals involving non-ground variables that, if taken as assignments, would result in failure of the lex-smaller test. The assignments of any such literals can be ruled out immediately by deleting the values from their respective domains. These domain deletions, together with the propagation of the domain deletion decisions, reduce the search required to find solutions (or confirm that no solutions exist) for the CSP. This process is analogous to the deletion of "near misses" in GAP-SBDD, as discussed in [FOC 01, GEN 03].

Since the cost of computing minimal images and comparing lists is generally outweighed by the reduction in search due to early backtracking and early domain deletion, our method is a useful and generic extension of lex-ordering as a symmetry breaking technique.

1.3.2. *Motivating example*

Suppose that we wish to solve

$$\frac{A}{10B + C} + \frac{D}{10E + F} + \frac{G}{10H + I} = 1$$

where each variable takes a value from 1 to 9. There are $3!$ permutations of the summands which preserve solutions. Suppose now that during search for all solutions we have made the partial assignment $PA : A = 2, B = 3, C = 8, D = 2, E = 1$. This is mapped by the group element $(DAG)(EBH)(FCI)$ as a sequence to $G = 2$,

$H = 3, I = 8, A = 2, B = 1$; or, in sorted order, $A = 2, B = 1, G = 2, H = 3, I = 8$.

Under the variable ordering $ABCDEFGHI$, this is lexicographically smaller than PA (since $B = 1$ is smaller than $B = 3$) so we would backtrack from this position. Notice that although we map a state which includes $A = 2$ into a smaller state we *can't* do it by mapping the literal $A = 2$ to itself.

Even if we can't immediately backtrack, there are often safe domain deletions that can be made. For example if we have only assigned $A = 7$, it is safe to remove values 1 through 6 from the domains of D and G , since making any of these assignments would lead to an immediate backtrack. This combination of early backtrack and domain reduction provides the search pruning needed to offset the overhead of computing and comparing images of assignments.

1.3.3. The GAPLex algorithms

We have a CSP and a symmetry group, G , for the CSP. G acts on the set of literals (variable-value pairs) of the problem, written as an initial subset of the natural numbers. The set of literals forms a variables \times values array, so that the literals of the lex-least variable are $1, 2, \dots, |dom(V_1)|$, etc. The GAPLex method, applied at a node N in search, proceeds as follows :

Require: $PA \leftarrow$ current partial assignment
Require: $Var \leftarrow$ next variable w.r.t. any fixed choice heuristic
Require: $val \leftarrow$ next value w.r.t. lex-least value ordering

- 1: set $Var = val$ and propagate
- 2: add $Var = val$ to PA
- 3: $T \leftarrow$ literals involving unassigned variables
- 4: pass PA and T to the GAPLex CGT test
- 5: **if** the test returns **false then**
- 6: backtrack
- 7: **else**
- 8: the test returns **true** and a list of literals, D
- 9: **for** $(X = \alpha) \in D$ **do**
- 10: remove α from the domain of X and propagate
- 11: **end for**
- 12: continue search
- 13: **end if**
- 14: **if** $Var = val$ does not lead to a solution **then**
- 15: set $Var \neq val$ and propagate
- 16: **if** a solution is obtained **then**
- 17: check that the solution is not isomorphic to any previous solution

```

18:  else
19:    move to next search node
20:  end if
21: end if

```

There are several points of interest. We assume that some consistency heuristics are in place, which propagate search decisions, backtracking away from no-goods and either stopping at the first solution or continuing search until all solutions are found. Every time we assign a variable during search, the consistency heuristics can provide additional domain reductions. The list T passed to the CGT test contains only those literals that involve the current domains of non-ground variables, as opposed to the domains before search. In this sense T is the smallest list we can pass, making the CGT test as efficient as possible. The method – if applied at every node in search – is sound, since we only backtrack away from solutions that are not lex-least, and we make no domain deletions involving lex-least solutions. This method is very much in the spirit of GAP-SBDD; the aim is to replace dominance detection by the power and simplicity of lex-ordering SB heuristics. Another way of looking at the method is as a propagator for (unposted) lex-ordering SB constraints. The method backtracks and reduces domains in line with the constraints that a static lex-ordering would have posted before search. Indeed, this conceptualisation motivates the notion that we can *combine* GAPLex with static lex-ordering constraints. The combination will be sound, since the same constraints apply, with only the order of their posting or propagation affecting the dynamics of CSP search.

As in other CGT-based SB methods, we can only expect a win if the cost of the CGT test is less than the cost of performing the search needed without early backtracking and early domain deletions provided by the test. By using highly efficient implementations of powerful permutation group algorithms, we can achieve this goal. The GAPLex CGT test, written in GAP, is specified as follows:

Require: G – a symmetry group for a CSP
Require: PA and T – as ordered lists of literals

```

1: if  $PA$  is not lex-least in its orbit under  $G$  then
2:   result = false
3: else
4:    $D \leftarrow t \in T$  if added to  $PA$  would make  $PA$  not lex-least
5:   result = true and  $D$ 
6: end if

```

The test proceeds by recursive search, similar to that described in [LIN 04], terminating when either the elements of PA have been exhausted, or the group at the bottom of the stabiliser chain consists only of the identity permutation. This stabiliser chain is the sequence stabiliser of the literals involving the decisions made above the

current node during search. More precisely, a recursive routine with the following specification is applied:

Require: G – a permutation group
Require: $SOURCE$ and $EXTRA$ – as ordered lists of points
Require: $TARGET$ – an ordered list of points

- 1: **if** $\exists g \in G : g(SOURCE) \prec_{\text{lex}} TARGET$ **then**
- 2: result = false
- 3: **else**
- 4: $D \leftarrow \{t \in EXTRA : \exists g \in G : g(SOURCE \cup \{t\}) \prec_{\text{lex}} TARGET\}$
- 5: appropriate subset of D is added to a global list DL
- 6: result = true
- 7: **end if**

Calling this routine with the same G , and with $SOURCE$ and $TARGET$ both equal to PA and $EXTRA$ equal to T clearly achieves the specification above. The implementation of this routine is:

- 1: **GAPLexSearch**($G, SOURCE, TARGET, EXTRA$)
- 2: **if** $TARGET$ is empty **then**
- 3: return true
- 4: **end if**
- 5: $x \leftarrow TARGET[1]$
- 6: **for** $y \in SOURCE$ **do**
- 7: **if** $\exists g \in G : g(y) < x$ **then**
- 8: return false
- 9: **else**
- 10: **if** $\exists g \in G : g(y) = x$ **then**
- 11: $G' \leftarrow$ the stabiliser of x in G
- 12: $S' \leftarrow SOURCE \setminus \{y\}$
- 13: $T' \leftarrow TARGET \setminus \{x\}$
- 14: $res \leftarrow \text{GAPLexSearch}(G', S', T', EXTRA)$
- 15: **if** $res = \text{false}$ **then**
- 16: return false
- 17: **end if**
- 18: **end if**
- 19: **end if**
- 20: **end for**
- 21: **for** $y \in EXTRA$ **do**
- 22: **if** $y \notin DL$ and $\exists g \in G : g(y) < x$ **then**
- 23: add y to DL
- 24: **end if**
- 25: **end for**
- 26: **return true**

1.4. Empirical evaluation

Our implementation uses the GAP-ECLⁱPS^e system, with CSP modelling and search performed in ECLⁱPS^e, and with GAP providing black-box answers to symmetry questions. We have tested our implementation on two classes of CSP. The first is Balanced Incomplete Block Designs (BIBDs), problem class 28 in *csplib*. This class was chosen to be a stern test of the effectiveness of GAPLex, with a large number of symmetries and small domains. We would therefore expect the search for lex-inspired early backtracks to be expensive, with not many useful domain deletions being returned. This expectation is realised in our results given in Table 1.1. The better results for GAP-SBDD are in part because GAP-SBDD has special support for problems with Boolean variables. We tried posting the complete set of lexicographic lex-leader constraints on each BIBD instance, but the number of constraints was too great.

						GAP-SBDD		GAP-LEX		Double Lex		GAP-Lex no prop		Combined	
V	B	R	K	λ	Δ	\bigcirc	Δ	\bigcirc	Δ	\bigcirc	Δ	\bigcirc	Δ	\bigcirc	
7	7	3	3	1	3	470	3	1150	3	20	21	1389	3	1150	
6	10	5	3	2	4	869	29	80100	5	30	29	80100	4	50340	
7	14	6	3	2	13	502625	-	-	30	110	-	-	-	-	
9	12	4	3	1	12	451012	-	-	30	120	-	-	-	-	
11	11	5	5	2	11	68910	-	-	20	140	-	-	-	-	
8	14	7	4	3	14	219945	-	-	143	720	-	-	-	-	

- > 2 hours Δ Number of Backtracks \bigcirc Total runtime in ms

Table 1.1. GAP-SBDD vs GAPLex. Problem class: BIBDs modelled as binary matrices.

The second problem class is Graceful Graphs, a graph labelling problem described in [PET 03]. The symmetries that arise are any symmetries of the graph, combined with symmetries of the labels. In this class the domains are larger, and, in general, there are fewer symmetries. The results for this class of problems (given in Table 1.2) are – as expected – more encouraging. We see that, in contrast to BIBDs, GAPLex provides fewer backtracks but performs faster than GAP-SBDD. GAPLex performs as well as GAP-SBDD on these problems. We also tested the heuristic observation that no GAPLex tests will fail (resulting in a backtrack) until the first search-related backtrack occurs, although propagation may occur. The test involved simply turning GAPLex tests off until the first (if any) backtrack occurring in normal search. Our results for this heuristic are inconclusive for this class of problems.

1.4.1. Combining GAPLex with Incomplete Static SB methods

Much research has concentrated on symmetry-breaking constraints for *matrix models* – a constraint program that contains one or more matrices of decision variables – which occur frequently as CSPs. The prime example of this body of work is “double lex”, which imposes that both the rows and the columns are lexicographically

Instance	GAP-SBDS				GAP-SBDD			
	Δ	Φ	\triangle	\circ	Δ	Φ	\triangle	\circ
$K_3 \times P_2$	9	290	110	400	22	310	180	490
$K_4 \times P_2$	165	1140	3590	4730	496	3449	8670	12110
$K_5 \times P_2$	4390	35520	166149	201669	17977	174180	501580	675760
Instance	GAP-LEX				Partial GAP-LEX			
	Δ	Φ	\triangle	\circ	Δ	Φ	\triangle	\circ
$K_3 \times P_2$	10	160	100	260	12	150	130	280
$K_4 \times P_2$	184	1550	4020	5570	202	670	4980	5650
$K_5 \times P_2$	4722	47870	176200	224070	5024	18820	224310	243130

Δ Number of Backtracks Φ Gap Time in ms \triangle Eclipse time in ms \circ Total runtime in ms

Table 1.2. Comparison of symmetry breaking methods. Partial GAP-LEX is where GAP-LEX checks do not commence until after the first backtrack.

ordered [FLE 02]. This does *not* break all the compositions of the row and column symmetries.

Frisch *et al* introduced an optimal algorithm to establish generalised arc-consistency for the \preceq_{lex} constraint [FRI 02]. This gives an attractive point on the tradeoff: a linear time to establish a high level of consistency on constraints which often break a large proportion of the symmetry in matrix models. The algorithm can be used to establish consistency in any use of \preceq_{lex} , so in particular is useful for any use of lex-leader constraints.

Our approach is straightforward. We add static double-lex constraints before search. At each node in the search tree we run GAPLex, without supplying the list of candidate domain deletions. This clearly means that the test is computationally more efficient: the final for-loop in the CGT algorithm isn't performed. We justify this with the hypothesis that any safe deletion found would almost certainly be already ruled out by the static double-lex constraints.

The results set out in Table 1.3 show that GAPLex does not perform as well as simply posting double-lex constraints before search. However, GAPLex returns the correct number of solutions, whilst double-lex returns many symmetrically equivalent solutions. It seems that combining GAPLex with double-lex is a win over just using GAPLex. These results are not unexpected, as GAPLex was shown to behave poorly on this formulation of BIBDs in Table 1.1. We feel that the proof of concept is, however, interesting and useful.

1.4.2. Combining GAPLex with Puget's all-different constraints

Puget has recently presented a method of implementing lex-leader constraints for variable symmetries in CSPs with all-different constraints [PUG 05] in linear time. In

V	B	R	K	λ	Double Lex		GAP-Lex no Prop				Combined			
					Δ	\bigcirc	Δ	Φ	Δ	\bigcirc	Δ	Φ	Δ	\bigcirc
7	7	3	3	1	3	20	21	1330	59	1389	3	1130	20	1150
6	10	5	3	2	5	30	29	79970	130	80100	4	50290	50	50340
7	14	6	3	2	30	110	-	-	-	-	-	-	-	-
9	12	4	3	1	30	120	-	-	-	-	-	-	-	-
11	11	5	5	2	20	140	-	-	-	-	-	-	-	-
8	14	7	4	3	143	720	-	-	-	-	-	-	-	-

- > 2 hours Δ Number of Backtracks Φ Gap Time in ms
 Δ Eclipse time in ms \bigcirc Total runtime in ms

Table 1.3. *Static double-lex vs GAPLex with no search for safe deletions vs combined GAPLex and double-lex. Problem class: all solutions of BIBDs modelled as binary matrices.*

problems with both variable and value symmetries, these can be usefully combined with GAPLex. Our approach is the same as for double-lex: we post the static all-different constraints before search, and run the GAPLex test at each search node.

Our results, given in Table 1.4, show that GAPLex performs slightly better than static constraints, and that combining the two methods is better than using either in isolation. These encouraging results are made better by noting that, for this class of problems, using only static constraints results in twice as many solutions being returned as necessary.

Instance	Constraints		GAP-Lex no Prop				Combined			
	Δ	\bigcirc	Δ	Φ	Δ	\bigcirc	Δ	Φ	Δ	\bigcirc
$K_3 \times P_2$	16	800	12	150	130	280	10	140	100	240
$K_4 \times P_2$	369	4530	202	600	5140	5740	188	510	3300	3810
$K_5 \times P_2$	9887	297880	5024	19010	224740	243750	4787	14820	188820	203640

Δ Number of Backtracks Φ Gap Time in ms
 Δ Eclipse time in ms \bigcirc Total runtime in ms

Table 1.4. *Static symmetry breaking all-different constraints vs GAPLex with no search for safe deletions vs combined GAPLex and static constraints. Problem class: all solutions of Graceful Graphs in the standard model.*

1.5. Conclusions and Future Work

We have used and extended recent advances in Computational Group Theory to add lex-ordering to the class of symmetry breaking techniques that can be effectively implemented by using a CGT system to provide black-box answers to symmetry related questions. Our implementation, GAPLex, is competitive with GAP-SBDS and GAP-SBDD. The choice of which method to use for which class of CSPs appears to be an interesting research question. Answers to this question could provide insight

into yet more symmetry breaking methods. Also the CGT algorithms used are still new and based on experience with GAP-SBDD may improve by orders of magnitude with further investigation.

We have, moreover, demonstrated the first combination of static and search-based symmetry breaking methods that is (for certain classes of CSP) more efficient than using either the static or search-based method in isolation. This result is important, since the successful combination of symmetry breaking methods is taxing and open area of CSP research, with great potential benefits attached to positive answers.

More work is needed in two areas. Firstly, we must address the questions relating to why a particular symmetry breaking approach works better for some CSPs than others. Secondly, we need to investigate other potentially successful combinations of symmetry breaking techniques.

1.6. Bibliography

- [ALO 03] ALOUL F. A., SAKALLAH K. A., MARKOV I. L., “Efficient Symmetry Breaking for Boolean Satisfiability.”, GOTTLOB G., WALSH T., Eds., *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, Morgan Kaufmann, p. 271-276, 2003.
- [BAC 99] BACKOFEN R., WILL S., “Excluding Symmetries in Constraint-Based Search.”, JAFFAR J., Ed., *Principles and Practice of Constraint Programming - CP’99, 5th International Conference, Alexandria, Virginia, USA, October 11-14, 1999, Proceedings*, vol. 1713 of *Lecture Notes in Computer Science*, Springer, p. 73-87, 1999.
- [BOS 93] BOSMA W., CANNON J., *Handbook of MAGMA functions*, Sydney University, 1993.
- [BRO 88] BROWN C. A., FINKELSTEIN L., JR. P. W. P., “Backtrack Searching in the Presence of Symmetry.”, MORA T., Ed., *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, 6th International Conference, AAECC-6, Rome, Italy, July 4-8, 1988, Proceedings*, vol. 357 of *Lecture Notes in Computer Science*, Springer, p. 99-110, 1988.
- [CRA 96] CRAWFORD J., GINSBERG M., LUKS E., ROY A., “Symmetry-Breaking Predicates for Search Problems”, *Proceedings of Knowledge Representation 96*, p. 149-159, November 1996.
- [FAH 01] FAHLE T., SCHAMBERGER S., SELLMANN M., “Symmetry Breaking.”, Walsh [WAL 01], p. 93-107, 2001.
- [FLE 02] FLENER P., FRISCH A. M., HNICHT B., KIZILTAN Z., MIGUEL I., PEARSON J., WALSH T., “Breaking Row and Column Symmetries in Matrix Models.”, Hentenryck [HEN 02], p. 462-476, 2002.
- [FOC 01] FOCACCI F., MILANO M., “Global Cut Framework for Removing Symmetries.”, Walsh [WAL 01], p. 77-92, 2001.

- [FRI 02] FRISCH A. M., HNICH B., KIZILTAN Z., MIGUEL I., WALSH T., “Global Constraints for Lexicographic Orderings.”, Hentenryck [HEN 02], p. 93-108, 2002.
- [FRI 05] FRISCH A. M., JEFFERSON C., MIGUEL I., “Symmetry Breaking as a Prelude to Implied Constraints: A Constraint Modelling Pattern.”, KAELBLING L. P., SAFFIOTTI A., Eds., *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh*, Professional Book Center, August 2005.
- [GAP00] The GAP Group, GAP – Groups, Algorithms, and Programming, Version 4.2, 2000, (<http://www.gap-system.org>).
- [GEN 00] GENT I. P., SMITH B. M., “Symmetry Breaking in Constraint Programming.”, HORN W., Ed., *ECAI 2000, Proceedings of the 14th European Conference on Artificial Intelligence, Berlin, Germany, August 20-25, 2000*, IOS Press, p. 599-603, 2000.
- [GEN 02] GENT I. P., HARVEY W., KELSEY T., “Groups and Constraints: Symmetry Breaking during Search.”, Hentenryck [HEN 02], p. 415-430, 2002.
- [GEN 03] GENT I. P., HARVEY W., KELSEY T., LINTON S., “Generic SBDD Using Computational Group Theory.”, Rossi [ROS 03], p. 333-347, 2003.
- [HAR 04] HARVEY W., “A Note on the Compatibility of Static Symmetry Breaking Constraints and Dynamic Symmetry Breaking Methods”, HARVEY W., KIZILTAN Z., Eds., *Fourth International Workshop on Symmetry and Constraint Satisfaction Problems, (in conjunction with the Tenth International Conference on Principles and Practice of Constraint Programming), Toronto, Proceedings*, p. 42-47, September 2004.
- [HEN 02] HENTENRYCK P. V., Ed., *Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002, Ithaca, NY, USA, September 9-13, 2002, Proceedings*, vol. 2470 of *Lecture Notes in Computer Science*, Springer, 2002.
- [KEL 04] KELSEY T., LINTON S., RONEY-DOUGAL C. M., “New Developments in Symmetry Breaking in Search Using Computational Group Theory.”, BUCHBERGER B., CAMPBELL J. A., Eds., *Artificial Intelligence and Symbolic Computation, 7th International Conference, AISC 2004, Linz, Austria, September 22-24, 2004, Proceedings*, vol. 3249 of *Lecture Notes in Computer Science*, Springer, p. 199-210, 2004.
- [LIN 04] LINTON S., “Finding the smallest image of a set.”, GUTIERREZ J., Ed., *Symbolic and Algebraic Computation, International Symposium ISSAC 2004, Santander, Spain, July 4-7, 2004, Proceedings*, ACM, p. 229-234, 2004.
- [MES 01] MESEGUER P., TORRAS C., “Exploiting Symmetries within Constraint Satisfaction Search”, *AI*, vol. 129, p. 133-163, 2001.
- [PET 03] PETRIE K. E., SMITH B. M., “Symmetry Breaking in Graceful Graphs.”, Rossi [ROS 03], p. 930-934, 2003.
- [PUG 93] PUGET J.-F., “On the Satisfiability of Symmetrical Constrained Satisfaction Problems.”, KOMOROWSKI H. J., RAS Z. W., Eds., *Methodologies for Intelligent Systems, 7th International Symposium, ISMIS '93, Trondheim, Norway, June 15-18, 1993, Proceedings*, vol. 689 of *Lecture Notes in Computer Science*, Springer, p. 350-361, 1993.
- [PUG 03] PUGET J.-F., “Symmetry Breaking Using Stabilizers.”, Rossi [ROS 03], p. 585-599, 2003.

- [PUG 05] PUGET J.-F., “Breaking symmetries in all different problems.”, KAEHLING L. P., SAFFIOTTI A., Eds., *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30-August 5, 2005*, Professional Book Center, p. 272-277, 2005.
- [RON 04] RONEY-DOUGAL C. M., GENT I. P., KELSEY T., LINTON S., “Tractable symmetry breaking using restricted search trees”, DE MÁNTARAS R. L., SAITTA L., Eds., *Proceedings, 16th European Conference on Artificial Intelligence: ECAI-04*, IOS Press, p. 211–215, 2004.
- [ROS 03] ROSSI F., Ed., *Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings*, vol. 2833 of *Lecture Notes in Computer Science*, Springer, 2003.
- [WAL 97] WALLACE M. G., NOVELLO S., SCHIMPF J., “ECLiPSe : A Platform for Constraint Logic Programming”, *ICL Systems Journal*, vol. 12, num. 1, p. 159–200, May 1997.
- [WAL 01] WALSH T., Ed., *Principles and Practice of Constraint Programming - CP 2001, 7th International Conference, CP 2001, Paphos, Cyprus, November 26 - December 1, 2001, Proceedings*, vol. 2239 of *Lecture Notes in Computer Science*, Springer, 2001.

