

Symmetry Breaking by Local Search^{*}

S. D. Prestwich¹, B. Hnich², H. Simonis¹, R. Rossi³, and S. A. Tarim⁴

¹Cork Constraint Computation Centre, University College Cork, Ireland

²Faculty of Computer Science, Izmir University of Economics, Turkey

³Logistics, Decision and Information Sciences Group, Wageningen UR, The Netherlands

⁴Department of Management, Hacettepe University, Turkey

s.prestwich@cs.ucc.ie, brahim.hnich@ieu.edu.tr,
h.simonis@4c.ucc.ie, roberto.rossi@wur.nl,
armagan.tarim@hacettepe.edu.tr

Abstract. The presence of symmetry in constraint satisfaction problems can cause a great deal of wasted search effort, and several methods for breaking symmetries have been reported. We describe a new approach to partial symmetry breaking: using local search in the symmetry group to detect violated lex-leader constraints. The local search is interleaved with the backtrack search performed by the constraint solver, and violations are used to backjump in the search tree. The method works with any form of symmetry and any constraint solver, and can handle very large symmetry groups. In combination with double-lex symmetry breaking it gives good results on balanced incomplete block designs. This opens up a fruitful new connection between the fields of symmetry breaking and metaheuristics.

1 Introduction

Many constraint satisfaction problems (CSPs) contain symmetries. For example the N -queens problem has 8 (each solution may be rotated through 90, 180 or 270 degrees, and independently reflected) while other problems may have exponentially many symmetries. The presence of symmetry implies that search effort is being wasted by exploring symmetrically equivalent regions of the search space. By eliminating the symmetry (*symmetry breaking*) we may speed up the search significantly. Several distinct methods have been reported for symmetry breaking in CSPs.

Reformulating a problem to eliminate its symmetries is an excellent approach where possible, but in many problems it is difficult or impossible to eliminate all symmetries. Probably the most common approach is to break symmetries by *adding constraints* to the model. It has been shown that all symmetries can in principle be broken by this method [20], which was developed into the *lex-leader* method for Boolean variables and variable symmetries by [5], extended to non-Boolean variables and independent

^{*} S. A. Tarim and B. Hnich are supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under Grant No. SOBAG-108K027. R. Rossi is supported by Science Foundation Ireland under Grant No. 03/CE3/I405 as part of the Centre for Telecommunications Value-Chain-Driven Research (CTVR). R. Rossi and H. Simonis are supported by Grant No. 05/IN/I886.

variable and value symmetries by [17, 23], and to arbitrary symmetries by [25]. In practice too many constraints might be needed if there are exponentially many symmetries. Instead of explicitly adding lex-leader constraints to a model, a computational group theory system such as GAP [10] can be used during search to find relevant (unposted) constraints, as in the GAPLex method [16]. Good results have been obtained by adding subsets of the constraints to obtain *partial* symmetry breaking. For example in matrix models it is common to have permutation symmetry on both rows and columns, but breaking all such symmetries is NP-hard [5] and requires an exponential number of symmetry breaking constraints. Breaking row and column symmetries separately (*double-lex* or *lex²*) [7] does not break all combined symmetries but is tractable and quite powerful.

Symmetry Breaking During Search (SBDS) was invented by [2] and further elucidated by [12]. In SBDS constraints are added during search so that, after backtracking from a decision, future symmetrically equivalent decisions are disallowed. SBDS has been implemented by combining a constraint solver with the GAP system, giving GAP-SBDS [11], which allows symmetries to be specified more compactly via group generators. SBDS can still suffer from the problem that too many constraints might need to be added: it can handle billions of symmetries but some problems require many more. A related method to SBDS called *Symmetry Breaking Using Stabilizers* (STAB) [21] only adds constraints that do not affect the current partial variable assignment, and has other optimisations to reduce the arity and number of constraints. It does not break all symmetries but has given very good results on problems with up to 10^{91} symmetries.

Symmetry Breaking by Dominance Detection (SBDD) was independently invented by [6, 9] (a similar algorithm was also described by [3]) and combined with GAP to give GAP-SBDD [11]. SBDD breaks all symmetries but does not add constraints before or during search, so it does not suffer from the space problem of some methods. Instead it detects when the current search state is symmetrical to a previously-explored “dominating” state. It was shown by [22] that the dominance tests can be combined into a single auxiliary CSP then solved by standard constraint programming methods.

In this paper we describe and test a new approach to partial symmetry breaking, which uses local search to detect unposted, violated, generalised lex-leader constraints. Section 2 describes the new method, Section 3 presents a case study, Section 4 reports experimental results, Section 5 discusses related work, and Section 6 concludes the paper. We assume a basic knowledge of group theory (see [13] or any standard textbook on the subject) and its application to symmetry breaking in constraint programming (see the works cited above). This paper is an extension of previous work [18, 19].

2 Detecting violated lex-leader constraints

Suppose that we wish to solve a CSP using a standard constraint solver with depth-first search (DFS) and constraint processing. Suppose also that the problem has symmetry defined by a group G . [25] shows that any form of symmetry can be broken by adding *generalised lex-leader constraints* $X \preceq_{\text{lex}} X^g$ for all $g \in G$, where X is a total assignment on a fixed ordering of the problem variables, X^g is the image of X under g , X^g is *admissible* (it is also a total assignment), and \preceq_{lex} is the standard lexicographical

ordering relation. These constraints prune all solutions except the *canonical* (lex-least) ones. But in general an exponential number of the constraints are needed, making the method impractical for problems with large symmetry groups.

2.1 The detection problem

We propose a new way of using generalised lex-leader constraints. To show that it is valid for all forms of symmetry and all constraint solvers, we use the results of [25]. At a search tree node with partial assignment A , if we can find a group element $g \in G$ such that A^g is admissible (it is also a partial assignment) and $A^g \prec_{\text{lex}} A$, then we can backtrack because A violates a lex-leader constraint. Lex-ordering is easily extended to partial assignments: if totally-assigned prefixes of A^g and A have the property that $A^g \prec_{\text{lex}} A$ then the constraint is violated.

As an example, consider the 4-queens problem with the usual 8 symmetries including reflection about the vertical axis: the group element denoted by x . Suppose that we solve this problem using a matrix model in which each square on the board corresponds to a binary variable, 1 denotes a queen and 0 no queen at that position. Suppose also that we apply DFS and assign variables in a static row-by-row then column-by-column order. Consider the partial assignment $A = (1, 0, 0, 0, ?, \dots)$ corresponding to the board configuration in Figure 1(i), where a space denotes no queen, “•” denotes a queen, and “?” denotes no decision. Now A^x is the partial assignment $(0, 0, 0, 1, ?, \dots)$ corresponding to the board configuration in Figure 1(ii). But $A^x \prec_{\text{lex}} A$ whatever values are chosen for the unassigned variables, so A is symmetric to the lex-smaller node A^x and can be backtracked from. Note that it is also possible to reason on unassigned variables — for example $(1, 0, x, 0, 0) \prec_{\text{lex}} (1, 0, 1, y, 1)$ whatever the values of the unassigned variables x and y — but in experiments we found this to be unnecessary.

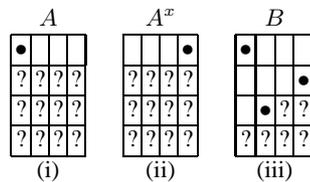


Fig. 1. Search states in 4-queens

We shall apply local search to the auxiliary problem of finding a g that causes generalised lex-leader violation. The method is justified by the following proposition:

Proposition 1. *Suppose that we have a full set of unposted generalised lex-leader constraints, under some fixed variable ordering. At a search tree node with partial assignment A , if we can find a $g \in G$ such that A^g is admissible and $A^g \prec_{\text{lex}} A$ under the same variable ordering, then A violates a lex-leader constraint.*

Proof. If A^g is admissible and $A^g \prec_{\text{lex}} A$ then $\theta(A^g) \prec_{\text{lex}} \theta(A)$ for any assignment θ of the unassigned variables in A leading to an admissible total assignment (at least one such θ always exists). Therefore for each such θ the lex-leader $\theta(A) \preceq_{\text{lex}} \theta(A^g)$ is violated by A .

Conversely, all symmetries can in principle be detected by search on the symmetry group:

Proposition 2. *If A violates a generalised lex-leader constraint then there exists $g \in G$ such that A^g is admissible and $A^g \prec_{\text{lex}} A$.*

Proof. Suppose that A violates a lex-leader constraint $X \preceq_{\text{lex}} X^g$ for some $g \in G$. Therefore $X^g \prec_{\text{lex}} X$ at A , so it must also be true that $A^g \prec_{\text{lex}} A$.

These results do not depend on the details of the constraint solver (for example its value and variable ordering heuristics, or its propagation algorithms) and apply to all forms of symmetry. However, lex-leader violation will be detected most often if the constraint solver uses the same fixed variable ordering that was used to define the lex-leader constraints. The value ordering used by the constraint solver also has an effect: pruning by lex-leader violation does not necessarily respect the constraint solver search heuristics, and might therefore search more of the tree to find a first solution. It is possible to make the search heuristics respect lex-leader violation by using lexicographical value orderings, but in this paper we are interested only in all-solution search (modulo symmetry) so the issue does not arise.

2.2 Detection as nonstationary optimisation

We can model the detection problem as an optimisation problem with G as the search space, so that each $g \in G$ is a search state. The objective function of g to be minimised is the lex ranking of A^g . On finding a state g with sufficiently small objective value (less than the lex ranking of A) we have solved the detection problem. This opens up the field of symmetry breaking to a wide range of metaheuristic algorithms.

A practical question here is: how much effort should we devote to detection at each DFS node? If an incomplete search algorithm fails to find a dominating state, this might be because there is no such state — but it could also be because the algorithm has not searched hard enough. Too little search might miss important symmetries, while too much will slow down DFS. We expend limited effort at each search node to ensure reasonable computational overhead. For example if we apply local search then we might apply one or a few local moves per search tree node, or only at some nodes. The optimisation problem now has an objective function that changes in time: as DFS changes variable assignments A , the objective value of any given g changes because it depends on A^g . This is called *nonstationary optimisation* in the optimisation literature, so the framework is called *Symmetry Breaking by Nonstationary Optimisation* (SBNO).

Note that even if detection fails at a node, it might succeed a few nodes later. DFS can then backtrack, possibly jumping many levels in the search tree. For example consider the 4-queens problem again. Suppose we did not manage to find group element x

at search state A , but instead continued with DFS and only discovered x on reaching search state B shown in Figure 1(iii). Now $B^x \prec_{\text{lex}} B$ so we can backtrack from B . On successful detection we backtrack until it is no longer the case that $A^g \prec_{\text{lex}} A$ for the current partial assignment A . Apart from some wasted DFS effort (during which we might find additional non-canonical solutions) the effect is the same as if we had detected the symmetry immediately. Thus SBNO effectively continues to try to break symmetry at a node until DFS backtracks past that node. This gives it an interesting property: a symmetry that would only save a small amount of DFS effort is unlikely to be detected, because DFS might backtrack past A before an appropriate g is discovered; in contrast, one that would save a great deal of DFS effort has a long time in which to be detected by local search. So SBNO should detect and break the *important* symmetries, which we define to be those that make a significant difference to the total execution time. This adaptive behaviour distinguishes it from other partial symmetry breaking methods such as lex^2 and STAB.

2.3 Detection by local search

To make SBNO more concrete we now show how to use local search for detection, though in principle any metaheuristic algorithm can be used. We have already defined the search space (G) and objective function (the lex ranking of A^g). Local search also requires a neighbourhood structure defining the possible local moves from each search state. To impose a neighbourhood structure on G we choose some subset $H \subset G$: from any search state g the possible local moves are the elements of H leading to neighbouring states $g \circ H$. Thus all G elements are local search states, and some of them (H) are also local moves. To apply hill climbing, from each state g we try to find a local move h such that the objective function is reduced ($A^{g \circ h} \prec_{\text{lex}} A^g$). If a series of moves (h_1, h_2, \dots) reduces the lex ranking sufficiently then we will find $A^{g \circ h_1 \circ h_2 \circ \dots} \prec_{\text{lex}} A$ and can backtrack from A . This method has a very low memory requirement, as it maintains just one dynamically changing group element representing the current local search state.

There is a relationship between local search and generators. A *generator* for a group is a subset H of the group G that can be used to generate all elements of G (denoted $\langle H \rangle = G$). A local search space is *connected* if there exists a series of local moves from any state to any other state. Connectedness is an important property for local search, because a disconnected space may prevent it from finding an optimal solution. It is easy to show that the search space induced by H is connected if and only if H is a generator set for G .

Proposition 3. *H is a generator set for G if and only if the search space induced by H is connected.*

Proof. Suppose that H is a generator for G . We can move from any g to any g' via element $g^{-1} \circ g'$ because $g \circ (g^{-1} \circ g') = (g \circ g^{-1}) \circ g' = g'$. H is a generator so we can always find a series of elements h_1, h_2, \dots such that $h_1 \circ h_2 \circ \dots = g^{-1} \circ g'$. Therefore $g \circ h_1 \circ h_2 \circ \dots = g'$ and the space is connected. Conversely, suppose that H is not a generator for G . Then there exists a $g^* \in G$ such that no series of elements satisfies

$h_1, h_2, \dots = g^*$. But for any g it holds that $g^* = g^{-1} \circ g'$ for some unique g' . Therefore there exists an unreachable state g' from any state g .

Thus if a non-generator H is used then the local search can become trapped in local minima, so random moves from $G \setminus H$ must be used to counteract this. Random restarts are a well-known technique for both local and backtrack search, but here they are necessary not only for heuristic reasons but because the space is disconnected. In our experiments we first used a generator H . This is a natural approach which can yield neighbourhoods of manageable size, because any group G has a generator of size $\log_2(|G|)$ or smaller [13]. However, we found better results using a non-generator H and restoring connectedness by allowing occasional random moves.

We use the following simple local search algorithm. Initialise g to be any group element (we use the identity element). At each search tree node A call the following procedure:

```

procedure SBNO
  if  $A^g \prec_{\text{lex}} A$ 
    backtrack to the first node  $B$  such that
       $B^g \prec_{\text{lex}} B$  cannot be proved
  else if  $A$  is a local minimum
     $g \leftarrow \text{RANDOMISE}(g)$ 
  else
     $g \leftarrow \text{IMPROVE}(g)$ 
  SBNO

```

This procedure performs hill-climbing until either (i) finding a solution that enables backjumping, or (ii) reaching a local minimum, in which case it applies random moves. The IMPROVE function applies an improving local move to g , that is a move h such that $A^{g \circ h} \prec_{\text{lex}} A^g$. The neighbourhood is explored in random order to find these moves. If no such move exists then the state is a local minimum and we exit after calling the RANDOMISE function, which (wholly or partially) randomises g .

3 Application to BIBDs

We test SBNO on a problem with very large symmetry groups, which has been used to test several symmetry breaking methods. Balanced Incomplete Block Design (BIBD) generation is a standard combinatorial problem, originally used in the statistical design of experiments but finding other applications such as cryptography. A BIBD is defined as an arrangement of v distinct objects into b blocks such that each block contains exactly k distinct objects, each object occurs in exactly r different blocks, and every two distinct objects occur together in exactly λ blocks. Another way of defining a BIBD is in terms of its *incidence matrix*, which is a binary matrix with v rows, b columns, r ones per row, k ones per column, and scalar product λ between any pair of distinct rows. A BIBD is therefore specified by its parameters (v, b, r, k, λ) . An example is shown in Figure 2.

For a BIBD to exist its parameters must satisfy the conditions $rv = bk$, $\lambda(v - 1) = r(k - 1)$ and $b \geq v$, but these are not sufficient conditions. Constructive methods can

be used to design BIBDs of special forms, but the general case is very challenging and there are surprisingly small open problems, the smallest being (22,33,12,8,4). One source of intractability is the very large number of symmetries: given any solution, any two rows or columns may be exchanged to obtain another solution. The symmetry group is the direct product $S_v \times S_b$ so there are $v!b!$ symmetries. A survey of known results is given in [4] and some references and instances are given in CSPLib¹ (problem 28).

$$\begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Fig. 2. A solution to BIBD instance (6, 10, 5, 3, 2)

We use the most direct CSP model for BIBD generation, which represents each matrix element by a binary variable. There are three types of constraint: (i) v b -ary constraints for the r ones per row, (ii) b v -ary constraints for the k ones per column, and (iii) $v(v-1)/2$ $2b$ -ary constraints for the λ matching ones in each pair of rows.

BIBDs have matrix symmetry, so the rows and/or columns of any solution can be permuted arbitrarily to find another solution. For matrix symmetry the SBNO local move neighbourhood we use is the set of row or column swaps involving the matrix entry corresponding to the binary variable v_f at which the last \prec_{lex} test failed. This heuristic is inspired by conflict-directed heuristics used in many local search algorithms — it focuses search effort on the source of failure. A drawback is that by using a non-generator of G we might fail to find an improving local move (recall Proposition 3). But using random moves at local minima compensates for this.

The RANDOMISE function of SBNO exchanges randomly chosen pairs of rows and columns. We found best results with a variable number $i+1$ of row and column exchanges, choosing each value $i = 0, 1, 2, \dots$ with probability $p^i(1-p)$ where $p = 0.1$. Choosing a random move g might not be practicable for all problems, as it is not always possible to efficiently generate a random group element [13]. But in the case of matrix symmetry it is easy: we simply exchange randomly selected rows or columns. Because we use an unbounded number of random moves at each local minimum, the local search algorithm is *probabilistically approximately complete* [14]: it is guaranteed to find a solution given sufficient time. This property might seem unnecessary for a nonstationary problem because changes to the objective function can cause escape from local minima, but because of the exponential nature of backtrack search the required changes might not occur for a long time.

¹ <http://www.csplib.org>

4 Experiments

We implemented SBNO in the Eclipse constraint logic programming system [1]. SBNO alone is a rather weak method so we do not present pure SBNO results; instead we aim to show that $\text{SBNO}+\text{lex}^2$ is a better symmetry breaking method than lex^2 alone, and competitive with other methods. When combining symmetry breaking methods care must be taken that not all solutions are excluded, but this combination is correct because lex^2 constraints can be derived from the lexicographically-smallest property of SBNO solutions.

Different researchers use different BIBD instances to test their algorithms. We use those of [21] which are the hardest instances used for all-solution search in the literature, and contain most other problem sets. Table 1 compares lex^2 alone with STAB and lex^2+SBNO in terms of the number of solutions found (the column “asym” shows the number of non-symmetrical solutions). All our results use a single run despite the nondeterminism of SBNO, because in experiments we found that the variation in results decreased with problem hardness. Best results are shown in **bold** and unknown results are denoted “?”.

The leading *partial* symmetry breaking method for BIBDs is currently STAB [21], which breaks more symmetries and solves larger instances than other methods. The results show that lex^2+SBNO breaks more symmetries than STAB in almost all cases. The two methods are implemented on different systems — ILOG Solver and Eclipse — so a direct comparison of runtimes is not possible until we reimplement SBNO in Solver, but we expect SBNO overheads to be low.

The leading *complete* symmetry breaking method for BIBDs is currently SBDD+STAB [22]. But complete methods pay a price in computation effort and the lex^2 results in the same paper are often faster. As shown in Figure 3, as problem hardness increases the advantage of lex^2+SBNO over lex^2 increases. The difference is up to a factor of 26, making it considerably faster than current complete methods. The plot shows two versions of SBNO: the version described above (“SBNO1”), and a version that only calls SBNO at only half the DFS nodes, and performs at most one local move at each call (“SBNO2”). This reduces the runtime overhead so that adding SBNO to lex^2 can speed it up by a factor of up to 40, and improves the average runtime. This version breaks fewer symmetries than STAB in most cases (not shown) but still far more than lex^2 . Which version of SBNO is recommended depends on whether we are interested in minimising the number of solutions or the runtime, though a more efficient implementation might make the question of overhead irrelevant.

In conclusion, lex^2+SBNO is certainly one of the most scalable symmetry breaking methods for BIBDs, and (subject to verification with a Solver implementation) might be the most scalable.

5 Related work

There is often a trade-off in tree search between (i) performing expensive reasoning at each node to potentially eliminate large subtrees, and (ii) processing nodes cheaply to reduce overheads. Partial reasoning can be applied in the hope of finding something

v	b	r	k	λ	asym	lex ²	lex ² +	STAB	SBNO
6	10	5	3	2	1	1	1	1	1
7	7	3	3	1	1	1	1	1	1
6	20	10	3	4	4	21	4	4	4
9	12	4	3	1	1	2	1	1	1
7	14	6	3	2	4	12	7	5	5
8	14	7	4	3	4	92	6	5	5
6	30	15	3	6	6	134	7	6	6
11	11	5	5	2	1	2	1	1	1
10	15	6	4	2	3	38	4	3	3
7	21	9	3	3	10	220	24	14	14
13	13	4	4	1	1	2	1	1	1
6	40	20	3	8	13	494	15	15	15
9	18	8	4	3	11	2,600	41	34	34
16	20	5	4	1	1	12	1	1	1
7	28	12	3	4	35	3,209	116	68	68
6	50	25	3	10	19	1,366	26	23	23
9	24	8	3	2	36	5,987	344	311	311
16	16	6	6	2	3	46	3	7	7
15	21	7	5	2	0	0	0	0	0
13	26	6	3	1	2	12,800	21	101	101
7	35	15	3	5	109	33,304	542	282	282
15	15	7	7	3	5	118	19	19	19
21	21	5	5	1	1	12	1	1	1
25	30	6	5	1	1	864	1	5	5
10	18	9	5	4	21	8,031	302	139	139
7	42	18	3	6	418	250,878	2,334	1,247	1,247
22	22	7	7	2	0	0	0	0	0
7	49	21	3	7	1,508	1,460,332	8,821	4,353	4,353
8	28	14	4	6	2,310	2,058,523	17,890	11,424	11,424
19	19	9	9	4	6	6,520	71	17	17
10	30	9	3	2	960	724,662	24,563	15,169	15,169
31	31	6	6	1	1	864	1	2	2
7	56	24	3	8	5,413	6,941,124	32,038	14,428	14,428
9	36	12	3	3	22,521	14,843,772	315,531	85,605	85,605
7	63	27	3	9	?	28,079,394	105,955	43,259	43,259
15	35	7	3	1	80	32,127,296	6,782	35,183	35,183
21	28	8	6	2	0	0	0	0	0
13	26	8	4	2	2461	3,664,243	83,337	31,323	31,323
11	22	10	5	4	4393	6,143,408	106,522	32,908	32,908
12	22	11	6	5	?	?	228,146	76,572	76,572
25	25	9	9	3	?	?	17,016	1,355	1,355
16	24	9	6	3	?	?	769,482	76,860	76,860

Table 1. Number of solutions found by different methods

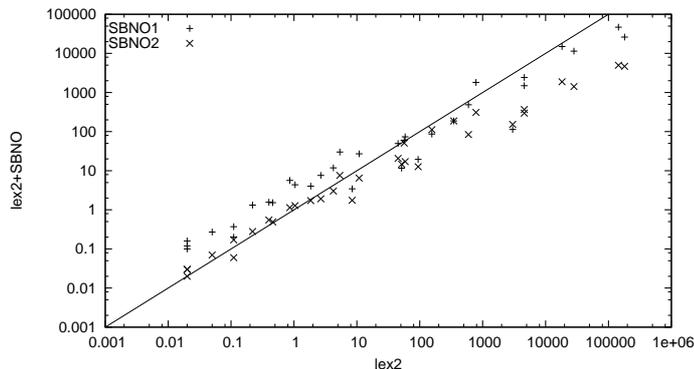


Fig. 3. Runtime scatter plots: lex^2 vs lex^2+SBNO

useful in a short time: for example [24] use local search within backtrack search to generate tight redundant constraints, an approach they call *heuristic propagation*. SBNO is another example of this type of integration, but its novel architecture allows it to continue reasoning about a search tree node long after leaving it behind. With respect to the general area of hybrid search algorithms, SBNO is a new integration of local and tree search. [8] survey such hybrids, but we believe that the nonstationary optimisation aspect of SBNO is unique. SBNO can also be seen as a form of *lifting*: representing a large set by an abstraction, and searching the abstraction instead of the set. [15] apply symmetry breaking to lifted SAT-encoded CSPs but are more concerned with detecting symmetry, and in SBNO only the *lex*-leader constraints are lifted.

Among symmetry breaking methods, SBNO is most closely related to GAPLex [16]. Both methods backtrack away from non-canonical solutions by detecting unposted *lex*-leader constraints that are currently violated. But whereas GAPLex uses computational group theory software to guarantee detection, SBNO uses incomplete local search. GAPLex turns out to be unsuited for breaking symmetry in BIBDs, and is able to solve only the first two instances of Table 1 in a reasonable time. This shows the advantage of using incomplete optimisation algorithms for partial symmetry breaking. GAPLex has also been defined only for variable symmetries, though it can be extended to arbitrary symmetries using generalised *lex*-leader constraints.

In our previous work SBNO was used with simple backtrack search without constraint propagation, breaking symmetry by TABU search [19] and a memetic algorithm [18]. It did quite well, but the experiments in this paper confirm our earlier speculation that lack of propagation was the factor that prevented it from solving the hardest BIBD instances. In previous versions it was also found necessary to modify the search algorithms to make them more likely to detect pure row and column symmetries; in this paper lex^2 is used to break the pure symmetries, freeing SBNO to detect the more general combined symmetries. Finally, in previous work SBNO was not characterised as a *lex*-leader-based method, but as a variant of SBDD. The new characterisation is

more correct and shows that SBNO can be used with any symmetry and any constraint solver.

6 Conclusion

This paper presented a new characterisation and more powerful implementation of SBNO, a recently developed framework for applying metaheuristic search to symmetry breaking during backtrack search. Previous methods have used constraint programming or computational group theory algorithms to solve auxiliary problems arising in symmetry breaking, but as far as we know SBNO is the first use of metaheuristics for this purpose. This new connection between symmetry breaking and metaheuristics is likely to be very fruitful for constraint programming. The small memory requirement and modest computational overhead of SBNO make it suitable for handling very large symmetry groups. In experiments on balanced incomplete block designs, SBNO with lex^2 broke more symmetries than two other partial symmetry breaking methods (lex^2 and STAB) and was faster than complete symmetry breaking methods.

In future work we will experiment with other metaheuristics and applications, in particular to problems with value symmetry and conditional symmetry. We also hope to combine SBNO with partial symmetry breaking methods other than lex^2 , in particular STAB. Combining two good techniques does not always yield further improvement but STAB and SBNO are to some extent orthogonal: STAB breaks symmetry among the unlabelled variables to increase constraint propagation, while SBNO breaks symmetry among the labelled variables and is closer to an intelligent backtracking technique. In fact SBNO can potentially boost the performance of *any* partial symmetry breaking method, as it may discover any violated generalised lex-leader constraint (see Proposition 2 above). SBNO could also be extended to conditional symmetry breaking by exploiting the results of [25].

Acknowledgement Thanks to Raphael Finkel for helpful comments.

References

1. K. R. Apt and M. G. Wallace. *Constraint Logic Programming Using Eclipse*. Cambridge University Press, 2007.
2. R. Backofen and S. Will. Excluding symmetries in constraint-based search. In *5th International Conference on Principles and Practice of Constraint Programming*, volume 1713 of *Lecture Notes in Computer Science*, pages 73–87. Springer, 1999.
3. C. A. Brown, Finkelstein, and P. W. Purdom. Backtrack searching in the presence of symmetry. In *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, volume 357 of *Lecture Notes in Computer Science*, pages 99–110. Springer, 1988.
4. C. J. Colbourn and J. H. Dinitz, editors. *The CRC Handbook of Combinatorial Designs*. CRC Press, 1996.
5. J. Crawford, M. L. Ginsberg, E. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In *Principles of Knowledge Representation and Reasoning*, pages 148–159. Morgan Kaufmann, 1996.

6. T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In *7th International Conference on Principles and Practice of Constraint Programming*, volume 2239 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2001.
7. P. Flener, A. M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. In *8th International Conference on Principles and Practice of Constraint Programming*, volume 2470 of *Lecture Notes in Computer Science*, pages 462–476. Springer, 2002.
8. F. Focacci, F. Laburthe, and A. Lodi. *Handbook of Metaheuristics*, chapter Local Search and Constraint Programming, pages 369–403. Kluwer Academic Publishers, 2003.
9. F. Focacci and M. Milano. Global cut framework for removing symmetries. In *7th International Conference on Principles and Practice of Constraint Programming*, volume 2239 of *Lecture Notes in Computer Science*, pages 77–92. Springer, 2001.
10. The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.4.12*, 2008.
11. I. P. Gent, W. Harvey, and T. Kelsey. Groups and constraints: Symmetry breaking during search. In *8th International Conference on Principles and Practice of Constraint Programming*, volume 2470 of *Lecture Notes in Computer Science*, pages 415–430. Springer, 2002.
12. I. P. Gent and B. M. Smith. Symmetry breaking in constraint programming. In *14th European Conference on Artificial Intelligence*, pages 599–603, 2000.
13. D. F. Holt, B. Eick, and E. A. O’Brien. *Handbook of Computational Group Theory*. Chapman & Hall/CRC, 2005.
14. H.H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, 2004.
15. D. Joslin and A. Roy. Exploiting symmetry in lifted csp. In *Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference*, pages 197–202. AAAI Press / The MIT Press, 1997. Providence, Rhode Island, USA.
16. T. Kelsey, C. Jefferson, K. Petrie, and S. Linton. Gaplex: Generalised static symmetry breaking. In *Sixth International Workshop on Symmetry Breaking*, pages 17–23, 2006. Nantes, France.
17. K. E. Petrie and B. M. Smith. Symmetry breaking in graceful graphs. Technical Report APES-56a-2003, APES Research Group, 2003.
18. S. D. Prestwich, B. Hnich, R. Rossi, and S. A. Tarim. Symmetry breaking by metaheuristic search. In *8th International Workshop on Symmetry and Constraint Satisfaction Problems*, 2008.
19. S. D. Prestwich, B. Hnich, R. Rossi, and S. A. Tarim. Symmetry breaking by nonstationary optimisation. In *19th Irish Conference on Artificial Intelligence and Cognitive Science*, 2008.
20. J.-F. Puget. On the satisfiability of symmetrical constraint satisfaction problems. In *Methodologies for Intelligent Systems*, volume 689 of *Lecture Notes in Artificial Intelligence*, pages 350–361. Springer, 1993.
21. J.-F. Puget. Symmetry breaking using stabilizers. In *9th International Conference on Principles and Practice of Constraint Programming*, volume 2833 of *Lecture Notes in Computer Science*, pages 585–599. Springer, 2003.
22. J.-F. Puget. Symmetry breaking revisited. *Constraints*, 10(1):23–46, 2005.
23. J.-F. Puget. An efficient way of breaking value symmetries. In *Twenty-First National Conference on Artificial Intelligence*, pages 117–122. AAAI Press / The MIT Press, 2006. Stanford, California, USA.
24. M. Sellmann and W. Harvey. Heuristic constraint propagation. In *4th International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 191–204, 2002. Le Croisic, France.

25. T. Walsh. General symmetry breaking constraints. In *12th International Conference on Principles and Practice of Constraint Programming*, volume 4204 of *Lecture Notes in Computer Science*, pages 650–664. Springer, 2006.