Boosting Partial Symmetry Breaking by Local Search

S. D. Prestwich[1], B. Hnich[2], H. Simonis[1], R. Rossi[3], S. A. Tarim[4]

[1]Cork Constraint Computation Centre
[2]Izmir University of Economics
[3]Wageningen UR
[4]Nottingham University Business School

introduction

a surprising variety of SB methods have been reported

we revisit a recent partial method: **Symmetry Breaking by Nonstationary Optimisation** (SBNO) [Prestwich, Hnich, Rossi, Tarim 2008]

SBNO interleaves *incomplete search in the symmetry group* with *backtrack search on the constraint problem*

advantages of SBNO

- can handle very large groups (eg 2^{200})
- potential use of the entire symmetry group
- low memory requirement
- low computational overhead

this paper

- a more accurate characterisation of SBNO (previously characterised as a form of SBDD)
- extended to arbitrary symmetries & constraint solvers
- reimplemented in a real constraint solver
- combination with lex^2
- new BIBD results: SBNO+lex^2 is one of the most scalable known methods

background

many CSPs contain symmetries: bijections on decision variables that preserve solutions and non-solutions

eg N-queens has 8 (each solution may be rotated through 90, 180 or 270 degrees, and reflected), BIBDs & others have many

symmetry wastes search: we must explore symmetrically equivalent regions of the search space

by eliminating the symmetry (*symmetry breaking*, SB) we may speed up the search significantly

several distinct methods have been reported for SB in CSPs

SBNO is a recent approach to *partial* SB: we don't insist that all symmetries are broken (sometimes more practical)

it interleaves (i) local search (LS) or evolutionary search in the symmetry group with (ii) backtrack search on the CSP, to detect broken symmetry

only limited time & memory are devoted to the LS

previous work in SymCon08 we described an SBNO prototype, implemented in C with only trivial constraint handling

it was competitive on all but the largest BIBD benchmarks from the literature

new work we now believe that SBNO is most useful as a method for boosting other partial SB methods, eg lex^2

we demonstrate this idea, formalise SBNO better, and present new results

detecting violated lex-leaders

we break symmetry by detecting violated *generalised lex-leader constraints* (GLLCs)

suppose that we wish to solve a CSP using a standard constraint solver with DFS & CP, and the CSP has symmetry defined by a group G

[Walsh 2006]: any form of symmetry can be broken by adding GLLCs $X \preceq_{\mathsf{lex}} X^g$ for all $g \in G$, where

- X is a total assignment on a fixed ordering of the CSP variables

- X^g is the image of X under g

- X^g is *admissible* (a valid total assignment)

- $\preceq_{\text{lex}}$ is the standard lexicographical ordering relation

these constraints prune all solutions except the canonical (lex-least) ones

but an exponential number of the constraints may be needed, so the method is impractical for problems with very large symmetry groups

we now characterise SBNO in terms of GLLCs, & show that it's valid for all forms of symmetry and all constraint solvers using [Walsh 2006]

the detection problem

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 and $A^g \prec_{\mathsf{lex}} A$, then we can backtrack because A violates a GLLC

lex-ordering is easily extended to partial assignments: if completely-assigned prefixes P, P' of A^g, A respectively have the property that $P \prec_{\mathsf{lex}} P'$ then the constraint is violated

eg 4-queens with usual 8 symmetries, including reflection about vertical axis (group element x)

suppose we solve this using a *matrix model*: each square on the board corresponds to a binary variable, 1 denotes a queen and 0 no queen

suppose we apply DFS and assign variables in a static row-by-row/column-by-column order

consider partial assignment $A = (1, 0, 0, 0, ?, \ldots)$ corresponding to board configuration

●			
?	?	?	?
?	?	?	?
?	?	?	?

then A^x is the partial assignment $(0, 0, 0, 1, ?, \ldots)$ corresponding to board configuration

			●
?	?	?	?
?	?	?	?
?	?	?	?

but $A^x \prec_{\mathsf{lex}} A$ whatever values are chosen for the unassigned variables, so A is symmetric to the lex-smaller node A^x and backtracking can safely occur from A

(it is also possible to reason on unassigned variables — eg $(1, 0, x, 0, 0) \prec_{\text{lex}} (1, 0, 1, y, 1)$ whatever the values of x & y — but in experiments we found this to be unnecessary)

SBNO applies LS to the "auxiliary problem" of finding a g that causes GLLC violation

this is justified by:

> **proposition** suppose we have a full set of unposted GLLCs 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

conversely, all symmetries can (in principle) be detected by search on the symmetry group:

> **proposition** if A violates a GLLC then there exists $g \in G$ such that A^g is admissible and $A^g \prec_{\mathsf{lex}} A$

these results don't depend on the details of the constraint solver (value/variable ordering heuristics, filtering algorithms) or the type of symmetry

but GLLC violation will be detected most often if the constraint solver uses the same fixed variable ordering that was used to define the GLLCs

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

objective function to be minimised: lex ranking of A^g

on finding an element g with sufficiently small objective value we have solved the detection problem

this opens up SB to a wide range of metaheuristic algorithms!

a practical question: how much effort should we devote to detection at each DFS node?

if an incomplete search algorithm fails to find an appropriate g, this might be because there is no such element — 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

our solution: expend limited effort at each search node to ensure reasonable computational overhead

eg if we apply LS then we might apply one or a few local moves per search tree node, or only at some nodes

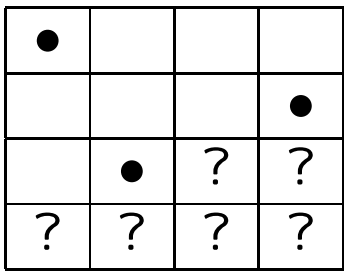
the objective function changes in time: as DFS changes variable assignments A, the objective value of any g changes because it depends on A^g

this is called *nonstationary optimisation* in the optimisation literature, hence SBNO

note: 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

eg consider 4-queens 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:



now $B^x \prec_{\mathsf{lex}} B$ so we can backtrack from B

a heuristic: on successful detection we backtrack until it is no longer the case that $A^g \prec_{\mathsf{lex}} A$ for 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

so 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

- a symmetry 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 that make a significant difference to the total execution time

this distinguishes it from other partial SB methods such as lex^2 and STAB

detection by LS

we now show how to use LS for detection, though any metaheuristic algorithm can be used

we have a search space (G) and objective function (lex ranking of A^g); LS also needs a *neighbourhood structure* defining possible local moves from each 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 $g \circ H$

so all G elements are local search states, and some (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 $(h_1, h_2, \ldots)$ reduces the lex ranking sufficiently then we will find $A^{g \circ h_1 \circ h_2 \circ \ldots} \prec_{\text{lex}} A$ and can backtrack from A

generators & LS

there's a relationship between LS & group 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 LS because a disconnected space may prevent it from finding an optimal solution

the search space induced by H is connected iff H is a generator set for G

so if a non-generator H is used then the LS can become trapped in a subspace that does not contain an appropriate g

random moves from $G \backslash H$ can be used to counteract this, eg via *random restarts*: a well-known technique for both LS and backtracking, but if H is not a generator then they are necessary not only for heuristic reasons but because the space is disconnected

we first used a generator H (a natural approach which can yield neighbourhoods of manageable size, because any group G has a generator of size $\log_2(|G|)$ or smaller), but got better results using a non-generator H, restoring connectedness by allowing occasional random moves

the LS algorithm

we use the following simple LS algorithm

initialise g to be any group element (we use the identity element)

at each search tree node A call:

procedure SBNO(g,A)
  if $A^g \prec_{\text{lex}} A$
    backtrack to the first node B s.t.
      $B^g \prec_{\text{lex}} B$ can't be proved
  else if A is a local minimum
    $g \leftarrow$ RANDOMISE(g)
  else
    $g \leftarrow$ IMPROVE(g)
    SBNO(g,A)

this hill-climbs until either (i) finding a solution that enables backjumping, or (ii) reaching a local minimum (when it applies random moves)

IMPROVE applies an improving local move to g, ie a move h s.t. $A^{g \circ h} \prec_{\mathsf{lex}} A^g$

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 RANDOMISE, which partly randomises g

this method uses very little memory: it maintains just *one* dynamically changing group element g representing the current LS state, and adds no constraints to the constraint store

application to BIBDs

we test SBNO on a problem with very large symmetry groups

BIBDs are used to test several SB methods & a standard combinatorial problem

a BIBD 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

specified by parameters (v, b, r, k, λ)

eg

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

is a $(6, 10, 5, 3, 2)$ instance

challenging (many unsolved cases) and highly symmetrical: rows and columns can be permuted, so $G = S_v \times S_b$ with $v!\, b!$ symmetries

we use the most direct CSP model: each matrix element is a binary variable, and there are 3 types of constraint:

- v b-ary constraints for the r ones per row
- b v-ary constraints for the k ones per column
- $v(v-1)/2$ $2b$-ary constraints for the λ matching ones in each pair of rows

heuristics

for the SBNO local move neighbourhood we use *the set of row or column swaps involving the matrix entry corresponding to the binary variable v_f at which the last $\prec_{\text{lex}}$ test failed*

inspired by conflict-directed heuristics used in many LS algorithms

focuses search effort on the source of failure

a drawback: by using a non-generator of G we might fail to find an improving local move, but using random moves at local minima compensates

RANDOMISE exchanges randomly chosen pairs of rows and columns, a random number of times (details in paper), making it PAC

experiments

we implemented SBNO in ECLiPSe & added it to a BIBD program with lex^2

(when combining SB 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 lex-least property of SBNO solutions)

we use a standard set of BIBD instances from [Puget 2003], which contain most other problem sets

we compare STAB, lex^2, SBNO & lex^2+SBNO in terms of #solutions

v	b	r	k	λ	asym	STAB	lex^2	SBNO	lex^2+ SBNO
6	10	5	3	2	1	**1**	**1**	1,098	**1**
7	7	3	3	1	1	**1**	**1**	5,856	**1**
6	20	10	3	4	4	**4**	21	26,412	**4**
9	12	4	3	1	1	**1**	2	2,988	**1**
7	14	6	3	2	4	7	12	5,856	**5**
8	14	7	4	3	4	6	92	11,438	**5**
6	30	15	3	6	6	7	134	281,764	**6**
11	11	5	5	2	1	**1**	2	9,443	**1**
10	15	6	4	2	3	4	38	33,290	**3**
7	21	9	3	3	10	24	220	44,932	**14**
13	13	4	4	1	1	**1**	2	18,388	**1**
6	40	20	3	8	13	**15**	494	3,191,087	**15**
9	18	8	4	3	11	41	2,600	139,999	**34**
16	20	5	4	1	1	**1**	12	561,879	**1**
7	28	12	3	4	35	116	3,209	343,393	**68**
6	50	25	3	10	19	26	1,366	—	**23**
9	24	8	3	2	36	344	5,987	706,648	**311**
16	16	6	6	2	3	**3**	46	1,482,986	7
15	21	7	5	2	0	0	0	—	0
13	26	6	3	1	2	**21**	12,800	706,648	101
7	35	15	3	5	109	542	33,304	2,109,417	**282**
15	15	7	7	3	5	**19**	118	—	**19**
21	21	5	5	1	1	**1**	12	—	**1**
25	30	6	5	1	1	**1**	864	—	5
10	18	9	5	4	21	302	8,031	1,402,133	**139**
7	42	18	3	6	418	2,334	250,878	—	**1,247**
22	22	7	7	2	0	0	0	—	0
7	49	21	3	7	1,508	8,821	1,460,332	—	**4,353**
8	28	14	4	6	2,310	17,890	2,058,523	—	**11,424**
19	19	9	9	4	6	71	6,520	—	**17**
10	30	9	3	2	960	24,563	724,662	—	**15,169**
31	31	6	6	1	1	**1**	864	—	2
7	56	24	3	8	5,413	32,038	6,941,124	—	**14,428**
9	36	12	3	3	22,521	315,531	14,843,772	—	**85,605**
7	63	27	3	9	—	105,955	28,079,394	—	**43,259**
15	35	7	3	1	80	**6,782**	32,127,296	—	35,183
21	28	8	6	2	0	0	0	—	0
13	26	8	4	2	2461	83,337	3,664,243	—	**31,323**
11	22	10	5	4	4393	106,522	6,143,408	—	**32,908**
12	22	11	6	5	—	228,146	—	—	**76,572**
25	25	9	9	3	—	17,016	—	—	**1,355**
16	24	9	6	3	—	769,482	—	—	**76,860**

SBNO alone is fairly weak: weaker than lex^2 and our prototypes (not shown)

in fact the new SBNO is weaker than the prototypes because it is designed for use with lex^2, so it does not need to detect pure row or pure column symmetries

but lex^2+SBNO is stronger than either method alone, and also stronger than previous SBNO versions

STAB [Puget 2003] is currently the leading partial SB method for BIBDs, breaking more symmetries than other partial methods and solving larger instances than complete methods

lex^2+SBNO breaks more symmetries than STAB in almost all cases, but we can't compare runtimes yet: different platforms & ECLiPSe vs Solver

SBNO and GAPLex

interesting to compare results with GAPLex [Kelsey, Jefferson, Petrie, Linton 2006]

both are based on violated lex-leader detection

SBNO uses LS for detection while GAPLex uses GAP

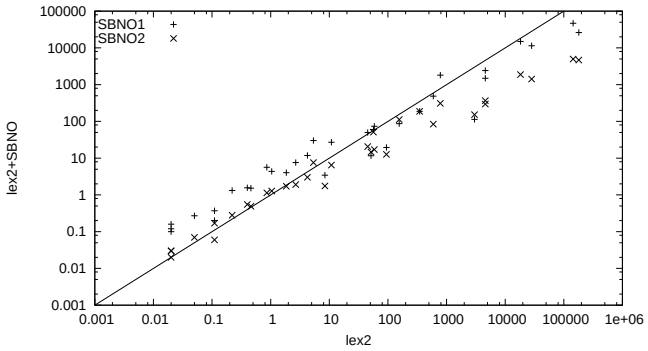
GAPLex gave poor results on BIBDs, solving only the first two instances (but it performs well on other problems)

presumably because GAPLex is a *complete* SB method, which can pay a high computational price (hence partial methods)

runtime overhead

profiling shows that only $\approx \frac{1}{3}$ of the total runtime is spent on SBNO processing, which is dwarfed by the improvement in total runtime

a scatter plot compares runtimes of lex^2 and lex^2+SBNO



2 versions of SBNO: the version described above ("SBNO1"), and one that only calls SBNO at only half the DFS nodes, and performs 1 local move at each call ("SBNO2")

as problem hardness increases, so does runtime advantage: up to a factor of 26 with SBNO1 and 40 with SBNO2 (but SBNO2 breaks fewer symmetries)

this makes it considerably faster than current complete methods: the leading one for BIBDs is SBDD+STAB [Puget 2005] but the lex^2 results in the same paper are often faster, and lex^2+SBNO is much faster than lex^2

so SBNO greatly boosts the power of lex^2

so much so that lex^2+SBNO is one of the most scalable symmetry breaking methods for BIBDs

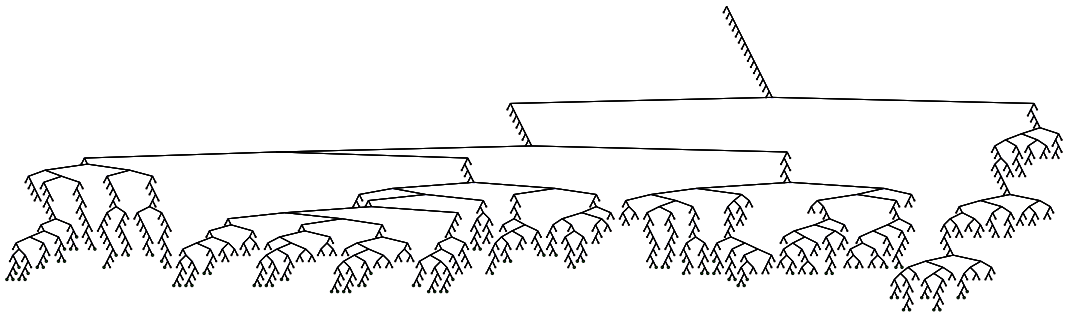
surprising that such a weak method becomes so strong when combined with lex^2?

but this shows that the two partial methods are complementary: lex^2 efficiently breaks *pure* row and column symmetries, while SBNO can potentially break *any* symmetry though it misses many

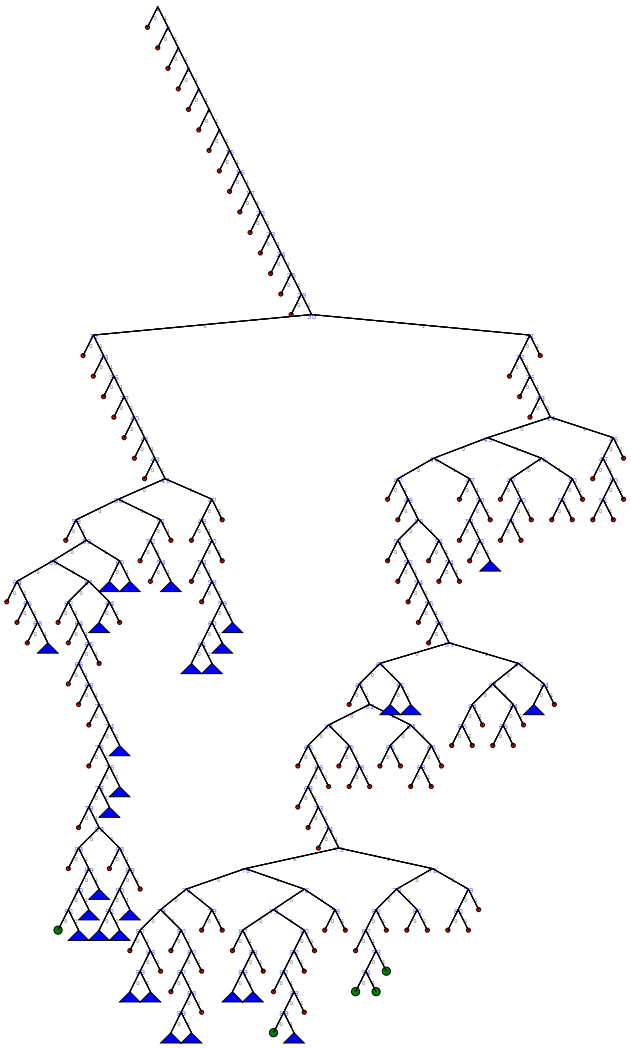
illustration

we examined the effect of SBNO on an all-solution search tree for instance (8,14,7,4,3)

using lex^2 alone:



and using lex^2+SBNO:



(triangles show where SBNO causes backtracking)

there are 2 main branches after an initial fixed assignment

on the left SBNO dramatically reduces the size of the tree, by cutting off large subtrees containing most of the removed solutions

on the right only a few nodes are removed, most of which are solutions

notice some “chains” of failure, in which a useful symmetry group element discovered at a lower level in the tree immediately prunes higher nodes

related work: adding constraints

adding constraints to the CSP [Puget 1993] is popular and can break all symmetries

developed into the *lex-leader* method for Boolean variables & variable symmetries [Crawford, Ginsberg, Luks, Roy 1996]

extended to non-Boolean variables and independent variable & value symmetries [Petrie, Smith 2003; Puget 2006]

extended to arbitrary symmetries [Wal 2006]

in practice too many constraints might be needed, but CGT systems can be used to detect (unposted) constraints, eg GAPLex

partial SB

adding a subset of the SB constraints can give very good results:

- *lex*^{2} [Flener, Frisch, Hnich, Kiziltan, Miguel, Pearson, Walsh 2002] for matrix models with row and column symmetry

  simple, tractable and quite powerful

- *Symmetry Breaking Using Stabilizers* (STAB) [Puget 2003] only adds constraints that do not affect the current partial variable assignment

  other techniques reduce arity & number of constraints

  more powerful, handling up to 10^{91} symmetries

SBDS

Symmetry Breaking During Search [Backofen, Will 1999; Gent, Smith 2000]

a complete method that adds constraints during search s.t., after backtracking from a decision, future symmetrically equivalent decisions are disallowed

has been implemented with CGT [Gent, Harvey, Kelsey 2002], allowing symmetries to be specified more compactly via group generators

may still add too many constraints but can handle billions of symmetries

SBDD

Symmetry Breaking by Dominance Detection [Fahle, Schamberger, Sellmann 2001; Focacci, Milano 2001; Brown, Finkelstein, Purdom 1988]

a complete method that does not add constraints before or during search, so it does not suffer from the space problem of some methods

detects when the current search state is symmetrical to a previously-explored "dominating" state

respects search heuristics (many do not, including SBNO)

also combined with CGT [Gent, Harvey, Kelsey 2002]

dominance testing can be solved as a CSP [Puget 2005]

GAPLex

SBNO is most closely related to GAPLex

both backtrack away from non-canonical solutions by detecting unposted lex-leader constraints that are violated

but GAPLex uses CGT for detection and is complete, while SBNO uses resource-bounded LS and is incomplete

GAPLex turns out to be unsuited for breaking symmetry in BIBDs, showing the advantage of using incomplete optimisation algorithms for partial SB

GAPLex has been defined only for variable symmetries, though it could be extended to arbitrary symmetries by using GLLCs

hybrid search

often a trade-off in tree search between (i) expensive reasoning at each node to potentially eliminate large subtrees, (ii) processing nodes cheaply to reduce overheads

partial reasoning can be used to hopefully find something useful in a short time

eg [Sellmann, Harvey 2002] use LS within backtrack search to generate tight redundant constraints (*heuristic propagation*)

SBNO is another example of this type of integration, but its nonstationary approach allows it to continue reasoning about a node long after leaving it behind

conclusion

other methods have used CP or CGT to solve auxiliary problems arising in SB, but SBNO is the first use of metaheuristics for this purpose

new connection between SB & metaheuristics should be fruitful for CP

SBNO's small time & memory overheads make it ideal for handling very large symmetry groups

on BIBDs, SBNO+lex^2 broke more symmetries than 2 other partial SB methods (lex^2 and STAB) and was faster than complete methods

future work

other metaheuristics & applications, especially with value symmetry & conditional symmetry (using results of [Walsh 2006])

could combine SBNO with other partial SB methods: it can potentially boost the performance of *any* partial symmetry breaking method, as it may discover any violated GLLC

combining two good techniques does not always yield further improvement but STAB and SBNO are quite orthogonal:

- STAB breaks symmetry among the *unlabelled* variables to increase propagation

- SBNO breaks symmetry among the *labelled* variables, cf intelligent backtracking