Snake Lex: an Alternative to Double Lex

Andrew Grayland[1], Ian Miguel[1] and Colva M. Roney-Dougal[2]
(andyg@cs, ianm@cs, colva@mcs).st-and.ac.uk

(1. School of Comp. Sci., 2. School of Maths and Stats), St Andrews, UK.

Abstract. Complete row and column symmetry breaking in constraint models using the lex leader method is generally prohibitively costly. Double lex, which is derived from lex leader, is commonly used in practice as an incomplete symmetry-breaking method for row and column symmetries. Double lex is based on a *row-wise* canonical variable ordering. However, this choice is arbitrary. We investigate other canonical orderings and focus on one in particular: *snake ordering*. From this we derive a corresponding incomplete set of symmetry breaking constraints, *snake lex*. Experimental data comparing double lex and snake lex shows that snake lex is substantially better than double lex in many cases.

1 Introduction

A *variable symmetry* in a constraint model is a bijective mapping from the set of variables to itself that maps (non-)solutions to (non-)solutions. The set of (non-)solutions reachable by applying all symmetry mappings to one (non-)solution forms an *equivalence class*. Restricting search to one member (or a reduced set of members) of each equivalence class can dramatically reduce systematic search: *symmetry breaking*. The *lex leader* method for breaking variable symmetries adds a constraint per symmetry so as to allow only one member of each equivalence class [3]. The lex leader method can produce a huge number of constraints and sometimes adding them to a model can prove counterproductive. One commonly-occurring case is when trying to break symmetries in a matrix, where any row (or column) can be permuted with any other row (or column): *row and column symmetries* [4].

It is often possible to achieve good results by breaking a smaller set of symmetries: *incomplete symmetry breaking*. One method to do this for row and column symmetries is *double lex* [4], which constrains adjacent pairs of rows and adjacent pairs of columns. Although this method is incomplete, it can reduce search dramatically. Double lex is derived from a reduction of a complete set of lex constraints created with a *row-wise* ordering as the canonical member of each equivalence class. The use of row-wise ordering is, however, arbitrary. The possible benefits of varying canonical orderings of lex constraints in graceful graph models was investigated in [11]. This paper investigates other canonical orderings and selects one (*snake ordering*) that looks promising to investigate further. From it, we create a new set of incomplete symmetry breaking constraints (*snake lex*). An empirical analysis demonstrates that snake lex can often deliver substantially better results than double lex.

Acknowledgements:A. Grayland is supported by a Microsoft Research EPSRC CASE studentship. I. Miguel is supported by a Royal Academy of Engineering/EPSRC Research Fellowship. C. M. Roney-Dougal is partially supported by EPSRC grant number EP/C523229/1.

2 Background

A constraint satisfaction problem is a triple $(\mathcal{X}, D, \mathcal{C})$, where $\mathcal{X}$ is a finite set of variables. Each $x \in \mathcal{X}$ has a finite set of values $D(x)$ (its *domain*). The finite set $\mathcal{C}$ consists of constraints on the variables. Each $c \in \mathcal{C}$ is defined over a sequence, $\mathcal{X}'$, of variables in $\mathcal{X}$. A subset of the Cartesian product of the domains of the members of $\mathcal{X}'$ gives the set of allowed value combinations. A *complete assignment* maps every variable to a member of its domain. A complete assignment satisfying all constraints is a *solution*. A *variable symmetry* of a CSP is a bijection $f : \mathcal{X} \to \mathcal{X}$ such that $\{\langle x_i, a_i\rangle \ : x_i \in \mathcal{X}, a_i \in D(x_i)\}$ is a solution if and only if $\{\langle f(x_i), a_i\rangle \ : \ x_i \in \mathcal{X}, a_i \in D(f(x_i))\}$ is a solution. The set of all variable symmetries of a CSP is closed under composition of functions and inversion, and so forms a group, the *variable symmetry group* of the CSP.

Row and column symmetries appear commonly in CSP models that contain matrices [2, 7–9]. When it is possible to map any ordered list of distinct rows to any other such list of the same length, with the same being true for columns, then there is *complete* row and column symmetry. For an n by m matrix there are $n! \times m!$ symmetries.

For a symmetry group of size s the lex leader method produces a set of $s - 1$ lex constraints to provide complete symmetry breaking. We first decide on a canonical order for the variables in $\mathcal{X}$, then post constraints such that this ordering is less than or equal to the permutation of the ordering by each of the symmetries. Consider the following 2×2 matrix with complete row and column symmetry, where $x_i \in \mathcal{X}$:

$$\begin{matrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{matrix}$$

If we choose a row-wise canonical variable ordering, in this case $x_{11}x_{12}x_{21}x_{22}$, then we can generate the following 3 lex constraints to break all the symmetries.

row swap:	$x_{11}x_{12}x_{21}x_{22} \leq_{\text{lex}} x_{21}x_{22}x_{11}x_{12}$
column swap:	$x_{11}x_{12}x_{21}x_{22} \leq_{\text{lex}} x_{12}x_{11}x_{22}x_{21}$
both swapped:	$x_{11}x_{12}x_{21}x_{22} \leq_{\text{lex}} x_{22}x_{21}x_{12}x_{11}$

Although this example is trivial, breaking all row and column symmetries by adding lex constraints is generally counter-productive since we have to add $(n! \times m!) - 1$ symmetry breaking constraints to the model. Double lex [4] is a commonly used incomplete symmetry breaking method for row and column symmetries. This method involves ordering the rows of a matrix and (independently) ordering the columns. This produces only $n + m - 2$ symmetry breaking constraints, shown below for the 2×2 example:

$$x_{11}x_{12} \leq_{\text{lex}} x_{21}x_{22}$$
$$x_{11}x_{21} \leq_{\text{lex}} x_{12}x_{22}$$

The double lex constraints can be derived from the lex leader generated based upon a row-wise canonical variable ordering. One method of doing so is to use reduction rules 1, 2 and 3′ as given in [5] and [6]. Let α, β, γ, and δ be strings of variables, and x and y be individual variables. The reduction rules are:

1 If $\alpha = \gamma$ entails $x = y$ then a constraint c of the form $\alpha x \beta \leq_{\text{lex}} \gamma y \delta$ may be replaced with $\alpha\beta \leq_{\text{lex}} \gamma\delta$.

2 Let $C = C' \cup \{\alpha x \leq_{\text{lex}} \gamma y\}$ be a set of constraints. If $C' \cup \{\alpha = \gamma\}$ entails $x \leq y$, then C can be replaced with $C' \cup \{\alpha \leq_{\text{lex}} \gamma\}$.

3′ Let $C = C' \cup \{\alpha x \beta \leq_{\text{lex}} \gamma y \delta\}$ be a set of constraints. If $C' \cup \{\alpha = \gamma\}$ entails $x = y$, then C can be replaced with $C' \cup \{\alpha\beta \leq_{\text{lex}} \gamma\delta\}$.

Rule 1 is subsumed by Rule 3′, but is often useful as a preprocess, as it reasons over an individual constraint. Algorithms to implement these rules are described in [6, 10].

3 In Search of an Alternative Canonical Ordering

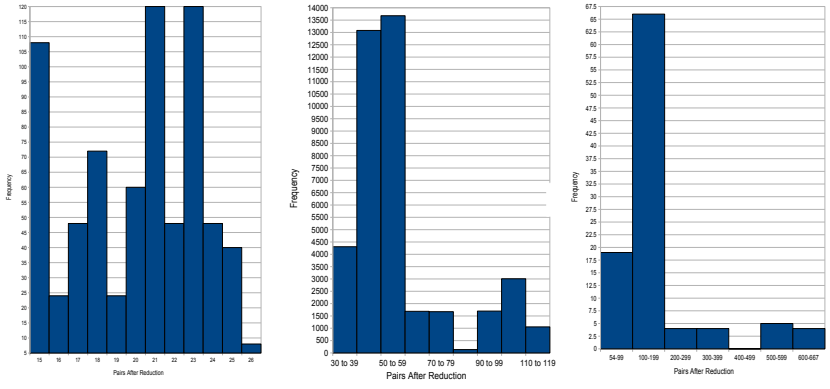
Double lex performs well [4], but we are not aware of work exploring similar-sized sets of incomplete symmetry-breaking constraints derived from other canonical variable orderings. In order to identify other useful canonical orderings we examine a large number of small cases and test interesting candidates on benchmark problems. A lex constraint of the form $x_1x_2 \dots x_m \leq_{\text{lex}} y_1y_2 \dots y_m$ consists of m *pairs* of variables. We compare canonical orderings by counting the pairs remaining after reducing the entire set of corresponding lex leader constraints by Rules 1, 2 and 3'. We hypothesize that fewer remaining pairs will promote reduced search through more effective propagation.

We began by examining all 6! = 720 canonical variable orderings of a 2×3 matrix. The algorithms described in [10] and [6] were used to reduce the lex constraints, and the distribution of pairs remaining is shown in Fig. 1. The smallest number of pairs remaining after reduction was 15, which was obtained for 108 of the canonical orderings. The ordering $x_{11}x_{21}x_{22}x_{12}x_{13}x_{23}$ reduces to 15 pairs and is interesting because it has a regular form, and might therefore be expected to produce a regular set of incomplete symmetry-breaking constraints equivalent to double lex. Standard row-wise ordering, $x_{11}x_{12}x_{13}x_{21}x_{22}x_{23}$, resulted in 23 pairs, so by this measure is a poor ordering.

Fig. 1 also shows the results of a similar experiment for all 8! canonical orderings of the 2×4 matrix. The ordering $x_{11}x_{21}x_{22}x_{12}x_{13}x_{23}x_{24}x_{14}$, the natural extension to that identified in the 2×3 results, reduces to just 30 pairs (row-wise reduces to 109). In general this ordering is simple to describe: columnwise the ordering starts at the top-left corner then moves down the column. It then moves to the neighbouring element in the next column and then upwards. This pattern continues until all variables have been listed. The row-wise snake variant is described similarly. We call this *snake ordering*.

We sampled 100 orderings of the 2×5 matrix, and also tested columnwise snake and row-wise orderings. Figure 1 summarises the results, in which snake ordering again proves to be one of the best measured by remaining pairs (reducing to 54 vs. 655 for

Fig. 1. Distribution of pairs remaining after reduction over **all** orderings for matrices with dimensions 2×3 (left) , 2×4 (centre). Distribution (right) of pairs remaining after reductions for a 2×5 matrix on 100 random, columnwise snake and row-wise orderings.



row-wise). To gain an indication of the importance of matrix dimension we compared row-wise against snake ordering on a 3×3 matrix. In this case, the two orderings produced much more similar results (reduction to 88 pairs for snake, 92 for row-wise), suggesting that matrix dimension *is* significant. Nonetheless, our results suggest that snake ordering is worth investigating further.

4 Snake Lex

Recall that double lex is derived by reducing a complete set of lex constraints with a row-wise ordering. In this section, we derive a corresponding small, easily-described set of constraints for the snake ordering, called *snake lex*. First we give a formal definition of columnwise snake ordering. Row-wise snake ordering is defined similarly.

Definition 1. *Let $\mathcal{X} = (x_{ij})_{n \times m}$ be a matrix of variables. The* columnwise snake ordering *on variables is $x_{11}, x_{21}, \ldots, x_{n1}, x_{n2}, \ldots, x_{12}, \ldots, x_{1m}, \ldots x_{nm}$, if m is odd, and $x_{11}, x_{21}, \ldots, x_{n1}, x_{n2}, \ldots, x_{12}, \ldots, x_{nm}, \ldots x_{1m}$ if m is even. That is, snake order on variables starts at row 1, column 1. It goes down the first column to the bottom, then back* up *along the second column to the first row. It continues, alternating along the columns until all variables have been ordered.*

From the columnwise snake ordering, columnwise snake lex can be derived:

Definition 2. *Let $\mathcal{X}(x_{ij})_{n \times m}$ be a matrix of variables. The* columnwise snake lex *set of constraints, $\mathcal{C}$, is defined as follows. $\mathcal{C}$ contains $2m-1$ column constraints, beginning*

$$\begin{array}{ll} c_1 & x_{11}x_{21}\ldots x_{n1} \leq_{\text{lex}} x_{12}x_{22}\ldots x_{n2} \\ c_2 & x_{11}x_{21}\ldots x_{n1} \leq_{\text{lex}} x_{13}x_{23}\ldots x_{n3} \\ c_3 & x_{n2}x_{(n-1)2}\ldots x_{12} \leq_{\text{lex}} x_{n3}x_{(n-1)3}\ldots x_{13} \\ c_4 & x_{n2}x_{(n-1)2}\ldots x_{12} \leq_{\text{lex}} x_{n4}x_{(n-1)4}\ldots x_{14} \\ & \vdots \end{array}$$

and finishing with

$$c_{2m-1} \quad x_{1(m-1)}\ldots x_{n(m-1)} \leq_{\text{lex}} x_{1m}\ldots x_{nm}$$

if m is odd and

$$c_{2m-1} \quad x_{n(m-1)}\ldots x_{1(m-1)} \leq_{\text{lex}} x_{nm}\ldots x_{nm}$$

if m is even. $\mathcal{C}$ contains $n-1$ row constraints. If m is odd then these are

$$\begin{array}{ll} r_1 & x_{11}x_{22}x_{13}\ldots x_{1m} \leq_{\text{lex}} x_{21}x_{12}x_{23}\ldots x_{2m} \\ r_2 & x_{21}x_{32}x_{23}\ldots x_{2m} \leq_{\text{lex}} x_{31}x_{22}x_{33}\ldots x_{3m} \\ & \vdots \\ r_{n-1} & x_{(n-1)1}\ldots x_{(n-1)m} \leq_{\text{lex}} x_{n1}\ldots x_{nm}. \end{array}$$

If m is even then these are:

$$\begin{array}{ll} r_1 & x_{11}x_{22}x_{13}\ldots x_{2m} \leq_{\text{lex}} x_{21}x_{12}x_{23}\ldots x_{1m} \\ r_2 & x_{21}x_{32}x_{23}\ldots x_{3m} \leq_{\text{lex}} x_{31}x_{22}x_{33}\ldots x_{2m} \\ & \vdots \\ r_{n-1} & x_{(n-1)1}\ldots x_{nm} \leq_{\text{lex}} x_{n1}\ldots x_{(n-1)m}. \end{array}$$

The following theorem shows that columnwise snake lex is derived from the columnwise snake lex leader, and is therefore sound.

Theorem 1. *The columnwise snake lex constraints are sound.*

PROOF We show that each constraint can be derived from a constraint in the full set of lex leader constraints by applying Rule 1 and then using only a prefix.

In each case the left hand side of the unreduced lex leader constraint is

$$x_{11}x_{21}\dots x_{n1}x_{n2}x_{(n-1)2}\dots x_{12}x_{13}\dots x_{n3}x_{n4}\dots$$

We first consider the column constraints, c_k. First let $k \equiv 1 \bmod 4$ and $a = (k+1)/2$. The symmetry which swaps columns a and $a+1$ and fixes everything else gives

$$Ax_{1a}x_{2a}\dots x_{na}B \leq_{\text{lex}} Cx_{1(a+1)}x_{2(a+1)}\dots x_{n(a+1)}D,$$

where A, B, C and D are strings of variables and $A = C$. Rule 1 removes A and C, and then considering only the first n pairs gives constraint c_k. If $k \equiv 2 \bmod 4$ then let $a = k/2$, replace $a+1$ by $a+2$ on the right hand side, and apply the same argument.

Next let $k \equiv 3 \bmod 4$ and $a = (k+1)/2$. The symmetry which swaps columns a and $a+1$ and fixes everything else gives a constraint of the form

$$Ax_{na}x_{(n-1)a}\dots x_{1a}B \leq$$
$$Cx_{n(a+1)}x_{(n-1)(a+1)}\dots x_{1(a+1)}D,$$

where A, B, C and D are strings of variables and $A = C$. Again, using Rule 1 on A and C, and then taking only a prefix gives constraint c_k. If $k \equiv 0 \bmod 4$ then let $a = k/2$, replace $a+1$ by $a+2$ on the right hand side, and apply the same argument. Consider now the rows. There is a symmetry that interchanges rows a and $a+1$ and fixes everything else. The unreduced lex leader constraint for this symmetry is:

$$x_{11}\dots x_{a1}x_{(a+1)1}\dots x_{n1}x_{n2}\dots x_{(a+1)2}x_{a2}\dots \leq_{\text{lex}}$$
$$x_{11}\dots x_{(a+1)1}x_{a1}\dots x_{n1}x_{n2}\dots x_{a2}x_{(a+1)2}\dots$$

Rule 1 deletes all pairs of the form (x_{ij}, x_{ij}) to obtain:

$$x_{a1}x_{(a+1)1}x_{(a+1)2}x_{a2}x_{a3}x_{(a+1)3}x_{(a+1)4}x_{a4}\dots \leq_{\text{lex}}$$
$$x_{(a+1)1}x_{a1}x_{a2}x_{(a+1)2}x_{(a+1)3}x_{a3}x_{a4}x_{(a+1)4}\dots$$

Rule 1 simplifies $x_{ai}x_{(a+1)i} \leq_{\text{lex}} x_{(a+1)i}x_{ai}$ to $x_{ai} \leq x_{(a+1)i}$, and similarly for $x_{(a+1)i}x_{ai} \leq x_{ai}x_{(a+1)i}$, resulting in constraint r_k:

$$x_{a1}x_{(a+1)2}x_{a3}x_{(a+1)4}\dots \leq_{\text{lex}} x_{(a+1)1}x_{a2}x_{(a+1)3}x_{a4}\dots$$

Since each of the snake lex constraints is derived by first applying Rule 1 to a lex leader constraint and then taking only a prefix, the snake lex constraints are sound. □

There are similarities between the columnwise snake lex and double lex constraints on columns. Columnwise snake lex constrains the first column to be less than or equal to both the second and the third columns. It also constrains the *reverse* of the second column to be less than or equal to the *reverse* of the third and fourth columns. This pattern continues until the penultimate column is compared with the last. As an example, consider a 4×3 matrix. Double lex constrains adjacent columns (left), while snake lex produces the set of constraints on the columns on the right:

$$x_{11}x_{21}x_{31} \leq_{\text{lex}} x_{12}x_{22}x_{32},$$
$$x_{12}x_{22}x_{32} \leq_{\text{lex}} x_{13}x_{23}x_{33},$$
$$x_{13}x_{23}x_{33} \leq_{\text{lex}} x_{14}x_{24}x_{34}.$$

$$x_{11}x_{21}x_{31} \leq_{\text{lex}} x_{12}x_{22}x_{32},$$
$$x_{11}x_{21}x_{31} \leq_{\text{lex}} x_{13}x_{23}x_{33},$$
$$x_{32}x_{22}x_{12} \leq_{\text{lex}} x_{33}x_{23}x_{13},$$
$$x_{32}x_{22}x_{12} \leq_{\text{lex}} x_{34}x_{24}x_{14},$$
$$x_{13}x_{23}x_{33} \leq_{\text{lex}} x_{14}x_{24}x_{34}.$$

Generally, given m columns and n rows, double lex adds $m-1$ constraints on columns, each with n pairs. Snake lex adds $2m-1$ constraints on columns, each with n pairs. We could increase the number of double lex constraints to the same number as snake lex by allowing each column to be less than or equal to the column two to its right, however *lex-chain*, which considers an entire set of rows or columns globally, has been shown to perform no better than double lex in practice[1].

We next consider the rows. Double lex gives the following for our sample matrix:

$$x_{11}x_{12}x_{13}x_{14} \leq_{\text{lex}} x_{21}x_{22}x_{23}x_{24},$$
$$x_{21}x_{22}x_{23}x_{24} \leq_{\text{lex}} x_{31}x_{32}x_{33}x_{34}.$$

The snake lex method is slightly more complicated, but gives the same number of constraints. We take the first two rows and zig zag between them to produce a string of variables starting at row 1, column 1, and a second string starting at row 2, column 1. We then constrain the first of these strings to be lexicographically less than or equal to the second one. Next we produce a similar constraint between rows i and $i+1$ for all i. The set of constraints for our 3×4 matrix are:

$$x_{11}x_{22}x_{13}x_{24} \leq_{\text{lex}} x_{21}x_{12}x_{23}x_{14},$$
$$x_{21}x_{32}x_{23}x_{34} \leq_{\text{lex}} x_{31}x_{22}x_{33}x_{24}.$$

Generally, double lex and snake lex both add $n-1$ row constraints, each with m pairs.

Thus far, we have considered columnwise snake ordering. We can also consider row-wise snake ordering, which may be useful if, for example, the rows are more heavily constrained (by the problem constraints) than the columns. To do so we simply *transpose* the matrix and then order as before. The transpose of our example 3×4 matrix is shown below (left), along with the corresponding constraints (right):

$$\begin{array}{lr}
 & x_{11}x_{12}x_{13}x_{14} \leq_{\text{lex}} x_{21}x_{22}x_{23}x_{24}, \\
x_{11}\ x_{21}\ x_{31} & x_{11}x_{12}x_{13}x_{14} \leq_{\text{lex}} x_{31}x_{32}x_{33}x_{34}, \\
x_{12}\ x_{22}\ x_{32} & x_{24}x_{23}x_{22}x_{21} \leq_{\text{lex}} x_{34}x_{33}x_{32}x_{31}, \\
x_{13}\ x_{23}\ x_{33} & x_{11}x_{22}x_{31} \leq_{\text{lex}} x_{12}x_{21}x_{32}, \\
x_{14}\ x_{24}\ x_{34} & x_{12}x_{23}x_{32} \leq_{\text{lex}} x_{13}x_{22}x_{33}, \\
 & x_{13}x_{24}x_{33} \leq_{\text{lex}} x_{14}x_{23}x_{34}.
\end{array}$$

Note that the double lex constraints do not change for this transposition, hence double lex is insensitive to switching between a row-wise and columnwise search ordering.

5 Experimental Results

We used four benchmark problem classes to compare snake and double lex empirically. All models used exhibit row and column symmetry. Preliminary experimentation revealed the superiority of row-wise snake lex on the tested instances, which we therefore used throughout. This is correlated with the rows being significantly longer than the columns in 3 of 4 classes. For each class we carried out four experiments per instance. We tested double lex and snake lex, each with row-wise and then snake static variable heuristics, separating the effects of the search order from those of symmetry breaking. Each time given is the mean of five trials and all results are presented in Fig. 2. The solver used was MINION.

The first problem class studied is *balanced incomplete block design* (BIBD) [9], in which b blocks and v objects must be related such that: each block contains k objects, each object appears in r blocks, and each pair of objects must appear together in a block λ times. We use the standard model, a $v \times b$ 0/1 matrix, with columns summing to k, rows summing to r, and the scalar product of every pair of rows equal to λ. Note that the parameters v, k and λ of the BIBD fix the values of b and r. Results show that snake lex outperforms double lex in every tested case. Where it was possible to find all solutions, snake lex both reduces search and breaks more symmetry (finds fewer symmetrically-equivalent solutions). The single solution cases show a speed up over double lex of several orders of magnitude, possibly due to interaction with the λ constraint.

The second problem class is the *equidistant frequency permutation array* problem (EFPD) [8], which is to find a set of c codewords drawn from q symbols such that each

BIBD:

Lex:	**Double**		**Snake**	
Search:	Row	Snake	Row	Snake
(v, k, λ)				
(7, 3, 5)	8.02,600557	8.38,606002	6.79,490784	6.78,455984
(7, 3, 6)	70.47,4979664	75.86,4321932	55.87,3984264	50.47,3448478
(7, 3, 20)	0.52,17235	0.47,15010	0.04,577	0.02,321
(7, 3, 30)	2.80,67040	2.53,60600	0.05,754	0.02,481
(7, 3, 40)	9.14,182970	8.58,168915	0.05,1063	0.03,641
(7, 3, 45)	16.09,278310	14.94,258860	0.06,1526	0.03,721
(7, 3, 50)	25.11,406525	23.70,380455	0.08,1671	0.05,801

EFPD:

Lex:	**Double**		**Snake**	
Search:	Row	Snake	Row	Snake
(q, λ, d, c)				
(3, 3, 5, 9)	4.1,164755	4.3,174677	3.6,120106	2.9,110666
(3, 4, 5, 9)	26.1,941061	27.1,1127011	15.9,537270	11.5,448329
(3, 5, 5, 10)	45.9,1556198	53.5,1841688	29.6,855950	20.0,678540
(3, 6, 5, 10)	68.9,2064850	76.8,2439205	39.7,1046091	27.4,811734
(3, 7, 5, 11)	94.5,2496190	103.0,2890581	54.5,1194583	35.0,910269
(3, 8, 5, 12)	124.6,2756291	123.4,3187617	71.0,1337286	43.1,1003119

FLECC:

Lex:	**Double**		**Snake**	
Search:	Row	Snake	Row	Snake
(q, d, c)				
(9, 5, 5)	23.34,662995	8.23,199720	13.09,293735	0.83,19817
(8, 6, 4)	0.73,5607	0.74,6122	0.53,6850	0.52,6359
(15, 5, 22)	0.77,15100	0.72,15136	0.41,8235	0.39,8205
(20, 5, 30)	3.28,51216	3.28,51223	1.31,19601	1.30,19603
(25, 5, 40)	3.88,49030	4.00,49002	1.44,17558	1.42,17600
(30, 5, 50)	4.56,49175	4.83,49112	1.86,17668	1.70,17750

Howell:

Lex:	**Double**		**Snake**		**Nodes**
Search:	Row	Snake	Row	Snake	All
(s, n)					
(4, 9)	0.98	0.97	1.19	1.23	60,799
(5, 11)	28.18	27.22	29.65	30.27	3,557,740
(6, 13)	1148.3	1153.8	1179.6	1196.2	231,731,046

Fig. 2. Double vs. row-wise snake lex, with both row and snake search ordering. BIBD & FLECC: searches above double line are for all solutions, remainder for one solution. Format: (time(s), nodes), except Howell, where nodes uniform throughout.

symbol occurs λ times in each codeword and the hamming distance between every pair of codewords is d. Our model [8] is a $c \times q\lambda$ matrix of variables with domain $\{1, ..., d\}$. In order to give a fair comparison between the four combinations of search order and lex constraints, we picked six instances with no solutions. This means that we are *not* examining the case where we did best in the previous experiment, that of finding the first solution, but instead exhausting the search space. Solve time decreases by around 30% when the lex method is changed from double to snake lex, and then by a further 30% when the search order is changed to snake. Notice also that the search time *increases* when double lex is used in conjunction with the snake order, suggesting both that it is important for the search order to mirror the constraints, and that it is the snake lex constraints that are causing the improved solve times.

The third problem class is the *fixed length error correcting codes* (FLECC) [7] problem, which is to find d length-q codewords drawn from 4 symbols such that the *Lee* distance (see cited paper for definition, or CSPLib 36) between every pair of codewords is c. Our model [7] uses a $d \times q$ matrix, with each of q symbols appearing once in each row, and Lee distance c between each pair of rows. As with the BIBD test case both all

solution and single solution problems were tested. Results show a speedup of almost two orders of magnitude in the all solution case. In the all solutions case, snake lex (with either order) requires on average less than half the time that the double lex with row-wise order uses, and the speedup seems to be increasing with instance size.

The final experiment involved solving the *Howell design* [2] problem. A Howell design is an s by s matrix M, whose coefficients are unordered pairs of symbols from a set $\mathcal{S}$ of size n. Each of the $n(n-1)/2$ pairs occurs at most once. Every row and every column of M contain at least one copy of every symbol from $\mathcal{S}$. There exists an *empty* symbol which can be used as many times as necessary, provided all other constraints are met. Three unsolvable instances were tested. Results show that, in this case, snake lex is slightly slower than double lex. One reason for this could be that Howell Designs are square (cf. Section 3). Note, however, that the nodes taken is the same throughout. The extra time taken could therefore simply be due to the overhead incurred by snake lex adding a slightly larger number of constraints than double lex.

6 Discussion

This paper highlights the need for further investigation into incomplete row and column symmetry breaking. We have demonstrated that double lex can be outperformed by snake lex in many instances and indeed, there may be another variable ordering that can create a small set of constraints capable of beating both. Future research will investigate the production on the fly of tailored sets of lex constraints, depending on a variable ordering specified by a modeler. The initial direction of future work will be to examine the other lex leaders found to have a small number of pairs upon reduction comparing them to both double lex and the snake lex and to investigate whether the shape of the matrix affects solve times, particularly with square matrices.

References

1. M. Carlsson and N. Beldiceanu. Arc-consistency for a chain of lexicographic ordering constraints. Technical Report T2002-18, SICS, 2002.
2. C. J. Colbourn and J. H. Dinitz. *The CRC Handbook of Combinatorial Designs*. CRC Press, Inc, Florida, US, 1996.
3. J. Crawford, M. Ginsberg, E. Luks, and A. Roy. Symmetry-breaking predicates for search problems. *Proc. KR*, 148–159, 1996.
4. P. Flener, A. M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. *Proc. CP*, 462–476, 2002.
5. A. M. Frisch and W. Harvey. Constraints for breaking all row and column symmetries in a three-by-two-matrix. *Proc. Symcon*, 2003.
6. A. Grayland, C. Jefferson, I. Miguel, and C. Roney-Dougal. Minimal ordering constraints for some families of variable symmetries. *Annals Math. A. I.*, 2009. To appear.
7. A. R. Hammons, P. Vijay Kumar, A. R. Calderbank, N. J. A. Sloane, and P. Sol. The z 4 -linearity of kerdock, preparata, goethals, and related codes. *IEEE Trans. Inform. Theory* 40(2), 301-319, 1994.
8. S. Huczynska. Equidistant frequency permutation arrays and related constant composition codes. Circa Preprint 2009/2, 2009.
9. P. Meseguer and C. Torras. Solving strategies for highly symmetric csps. *IJCAI*, 400-405, 1999.
10. H. Öhrman. Breaking symmetries in matrix models. MSc Thesis, Dept. Information Technology, Uppsala University, 2005.
11. Barbara M. Smith. Sets of Symmetry Breaking Constraints. Proceedings of SymCon05, the 5th International Workshop on Symmetry in Constraints, 2005.