In search of a better method to break row and column symmetries

Andrew Grayland[1], **Ian Miguel**[1] **and Colva M. Roney-Dougal**[2]
(1. School of Comp. Sci., 2. School of Maths and Stats), St Andrews, UK.
`(andyg@cs, ianm@cs, colva@mcs).st-and.ac.uk`

Abstract

Complete row and column symmetry breaking in constraint satisfaction problems 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. This technique uses a row-wise ordering to construct the lex leader. For this reason, it is generally counterproductive to choose a search ordering that is not also row-wise. It seems logical that the search order should be used to pick the symmetry breaking technique, rather than the other way around. This paper surveys other possible orderings and investigates one particular ordering, *snake ordering*. From this we derive a corresponding incomplete set of symmetry breaking constraints, *snake lex*. We present experimental data comparing double lex and the snake lex, showing that snake lex is substantially faster than double lex in many cases.

Introduction

Constraints offer a powerful means of solving a wide variety of combinatorial problems. Constraint solving of a combinatorial problem, such as planning or scheduling, proceeds in two phases. First, the problem is modelled as a set of decision variables and a set of constraints on those variables that a solution must satisfy. A decision variable represents a choice that must be made in order to solve the problem. The domain of potential values associated with each decision variable corresponds to the options for that choice. The second phase consists of using a constraint solver to search for solutions to the model: assignments of values to decision variables satisfying all constraints.

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 we can reach by applying all symmetry mappings to one (non-)solution forms an *equivalence class*. It can be advantageous to restrict search to one member (or a reduced set of members) of each equivalence class, because this can dramatically reduce the search space. This is known as *symmetry breaking*.

One symmetry breaking technique for variable symmetries is the *lex leader* method, which adds a constraint per symmetry so as to allow only one member of each equivalence class (Crawford et al. 1996). 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). We call these symmetries *row and column symmetries* (Flener et al. 2002).

When breaking all symmetries proves too difficult, 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* (Flener et al. 2002), which imposes an ordering on the rows, and an ordering on the columns. Whilst this method is not complete, in practice it can help to reduce search nodes and, ultimately, reduce solve times. 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. This means that we generally have to use a row-wise search order to achieve good results. It would be advantageous to be able select a variable ordering that is specific to the problem being solved, rather than being forced to use a particular search order because of the choice of symmetry breaking method.

This paper surveys a number of possible lex leader orderings for row and column symmetries, and selects one that looks promising to investigate further. We create a new set of incomplete symmetry breaking constraints from different variable ordering. In most cases that we have examined this new ordering proves to deliver substantially better results than double lex.

Background

A constraint satisfaction problem is a triple $(\mathcal{X}, D, \mathcal{C})$, where $\mathcal{X}$ is a finite set of variables. For each variable $x \in \mathcal{X}$ there is a finite set of values $D(x)$ (its *domain*). The finite set $\mathcal{C}$ consists of constraints on the variables. Each constraint $c \in \mathcal{C}$ 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 combinations of values. A *complete assignment* maps every variable to a member of its domain. A complete assignment that satisfies all constraints is a *solution*.

A *variable symmetry* of a CSP is a bijection $f : \mathcal{X} \rightarrow \mathcal{X}$

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, called the *variable symmetry group* of the CSP.

Row and column symmetries appear commonly in CSP models that contain matrices (Colbourn and Dinitz 1996; Hammons et al. 1994; Huczynska 2009; Meseguer and Torras 1999). 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 a 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 $x_i \in \mathcal{X}$:

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

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 (Flener et al. 2002) 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:

$$\begin{array}{ccc} x_{11}x_{12} & \leq_{\text{lex}} & x_{21}x_{22} \\ x_{11}x_{21} & \leq_{\text{lex}} & x_{12}x_{22} \end{array}$$

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 (Frisch and Harvey 2003) and (Grayland et al. 2009). 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 $C' \cup \{\alpha\beta \leq_{\text{lex}} \gamma\delta\}$.

Although Rule 1 is subsumed by Rule 3′, it is often useful to apply Rule 1 as a preprocess for efficiency reasons, since it reasons over an individual constraint.

An algorithm to implement these rules is described in (Öhrman 2005). We first create a list $L = \{l_1, \dots, l_s\}$ of lex constraints. Let $l \in L$ be

$$x_{11} \dots x_{1m} \leq_{\text{lex}} x_{21} \dots x_{2m}.$$

Then l is a sequence of m *pairs*, $p_i = (x_{1i}, x_{2i})$. The *most significant* pair is p_1, and the *least significant* is p_m.

We now construct a directed graph G, with nodes labelled by variables, and directed edges representing pairs, where the pair $x_i \leq x_j$ is a directed edge from x_i to x_j. Starting with $l_1 \in L$, we select the least significant pair in that lex constraint (x_{1m}, x_{2m}). For $i < m$ we add directed edges (x_{1i}, x_{2i}) and (x_{2i}, x_{1i}), representing equality between the two variables, for each more significant pair in l_1. For $2 \leq j \leq s$ we also add a single directed edge representing the inequality between the most significant pair of each $l_j \in L$. We define these pairs as *active*. Furthermore, the tth pair in the lex constraint l is active if the jth pairs, for $1 \leq j < t$, are equal and the tth pair is not equal.

The transitive closure G' of G is then computed. We now test whether equality is implied in the active pair for any constraint $l' \neq l_1$ (by a pair of directed edges). If so, we consider the next pair in l'. We continue to move one pair to the right until we find an example where the pair is not equal, or we reach the end of the constraint l'. In the former case, we add an edge representing this new pair to G'. The transitive closure of G' is then computed to attain G''.

This process is repeated until no new edges are added to the graph. We now check if there exists a directed edge (x_{1m}, x_{2m}). If there does, then $x_{1m} \leq x_{2m}$ is implied, and the pair can be removed by Rule 2. This process is repeated for the next least significant pair, and successive least significant pairs if one is removed, for every constraint in L.

The algorithm for Rule 3′ is similar, except that we test every pair in a constraint, not just the least significant, and we look for equality rather inequality for removal.

As an example of how a reduction works, consider the complete lex constraint

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

If the need arises to enforce the second clause in this constraint, $x_{12} \leq x_{11}$, then the variables in the first pair, $x_{11} \leq x_{12}$, must be equal. If $x_{11} = x_{12}$ then $x_{12} = x_{11}$. We can therefore remove the second clause by Rule 3′. We can apply a similar reduction to the fourth clause, by considering the third. The final reduced constraint is

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

Double lex is shown in (Flener et al. 2002) to perform favorably against both complete lex leader symmetry breaking and no symmetry breaking. Unfortunately, double lex does not work well in every case.

It is well known that the search order chosen to solve the problem can have a dramatic effect on the solve time. Since double lex is generated using a row-wise lex leader, it is generally necessary to use a row-wise search ordering. In many

Figure 1: Results for a $2 \times n$ matrix

n	Canonical Ordering	Pairs
4	$x_{11}x_{12}x_{13}x_{14}x_{21}x_{22}x_{23}x_{24}$	109
4	$\mathbf{x_{11}x_{21}x_{22}x_{12}x_{13}x_{23}x_{24}x_{14}}$	**30**
5	$x_{11}x_{12}x_{13}x_{14}x_{15}x_{21}x_{22}x_{23}x_{24}x_{25}$	655
5	$\mathbf{x_{11}x_{21}x_{22}x_{12}x_{13}x_{23}x_{24}x_{14}x_{15}x_{25}}$	**54**

cases, using a row-wise search order would not be the best option if double lex had not been used to break the symmetry in the model.

In search of an alternative ordering

Recall from the background that double lex is derived from the complete set of lex constraints based on a row-wise canonical ordering. This row-wise ordering may not correspond well to the problem. Indeed, there may exist other canonical orderings which can produce incomplete sets of constraints able to outperform double lex. In order to survey various canonical orderings, we first present a way to compare them.

As described in the previous section, a lex constraint of the form

$$x_1x_2\dots x_m \leq_{\text{lex}} y_1y_2\cdots y_m$$

consists of m *pairs* of variables. We compare canonical orderings by counting the pairs that remain after reducing the entire set of lex leader constraints by Rules 1, 2 and 3'. Fewer remaining pairs suggests that the canonical ordering may perform better: all reduced sets of lex constraints for the same symmetries are logically equivalent, so a reduced set that uses fewer pairs may give better propagation.

First, all of the 720 variable orderings of a 2×3 matrix

$$\begin{matrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{matrix}$$

were tested. The algorithms described in (Öhrman 2005) and (Grayland et al. 2009) were used to reduce the lex constraints. The smallest number of pairs remaining after reduction was 15, with 108 possible canonical orderings. The canonical 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. Standard row-wise canonical ordering, $x_{11}x_{12}x_{13}x_{21}x_{22}x_{23}$, resulted in 23 pairs. Examining the complete set of results in Fig 2 shows that by this measure row-wise is one of the worst possible canonical orderings.

This experiment shows that not only is it likely that a better alternative to double lex could exist, but that row-wise may be one of the *worst* canonical orderings.

Similar experiments were performed on $2 \times n$ matrices.

$$\begin{matrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \end{matrix} \quad \text{and} \quad \begin{matrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \end{matrix}$$

All $8!$ possible canonical orderings for the 2×4 matrix were tested and 100 random canonical orderings were tested for the 2×5 matrix. Additionally we tested the row-wise

Figure 2: Distribution of pairs remaining after reduction for a 2×3 matrix on all orderings.

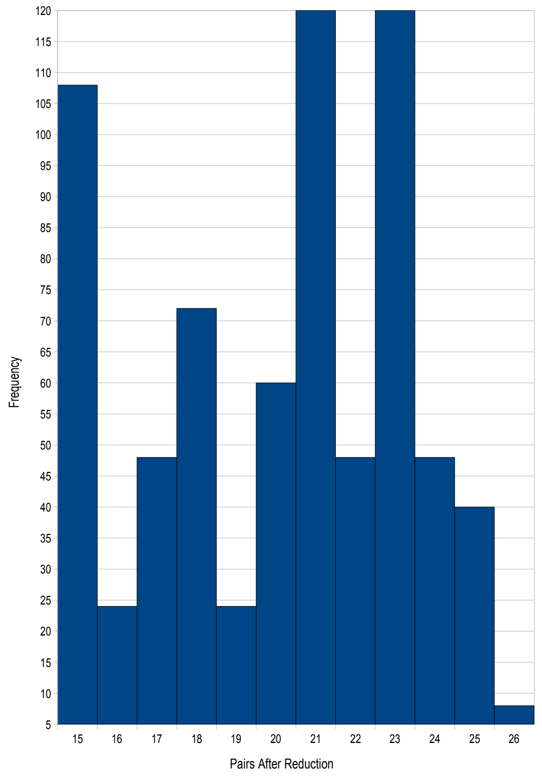
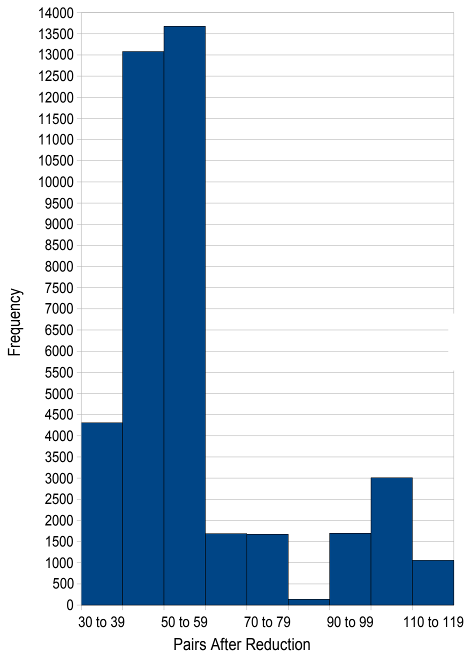


Figure 3: Distribution of pairs remaining after reduction for a 2×4 matrix on all orderings.



canonical ordering, and the interesting canonical ordering from above on the 2×5.

Some sample results are given in Figure 1, where conventional row-wise canonical ordering is the first one listed in each case and the other is the canonical ordering which reduced to 15 pairs in the 2×3 case. previously.

The canonical ordering that begins $x_{11}x_{21}x_{22}x_{12}x_{13}x_{23}$ (shown in bold in Figure 1) has fewer pairs after reduction than the row-wise canonical ordering, and it has a relatively simple pattern to describe. The canonical ordering starts at the top left element of a matrix then moves down the column. It then moves to the neighbouring element in the next column and then up that column. This pattern continues until all variables have been listed. We call this *snake ordering*: in each case we see that snake ordering results in significantly fewer pairs after reduction than row-wise canonical ordering.

A further experiment was carried out on

$$\begin{matrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{matrix}$$

to determine whether snake lex has fewer pairs than row-wise canonical ordering in a larger example, and the results are given in Figure 5. Again, row-wise canonical ordering is in row 1 and snake ordering is in bold. Once again snake ordering has fewer remaining pairs than row-wise canonical ordering, although not by such a large margin as the $2 \times n$ cases. This suggests that the matrix dimensions are significant.

Figure 4: Distribution of pairs remaining after reductions for a 2×5 matrix on 100 random orderings and the two from Figure 1.

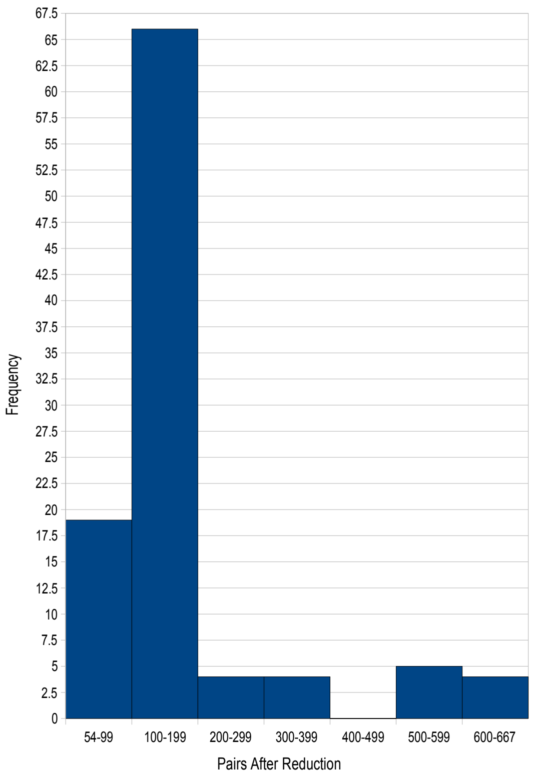


Figure 5: Results for a 3×3 matrix

Ordering	Pairs
$x_{11}x_{12}x_{13}x_{21}x_{22}x_{23}x_{31}x_{32}x_{33}$	92
$\mathbf{x_{11}x_{21}x_{31}x_{32}x_{22}x_{12}x_{13}x_{23}x_{33}}$	**88**

The results from Figures 2, 3 and 4 suggest that snake ordering is worth investigating further. However, the number of pairs may not be an accurate measure of performance. In the next section, we construct a reduced set of constraints for partial symmetry breaking based on the snake lex, and in the following section we present experimental results using our reduced set of constraints.

Snake Lex

Recall that double lex is derived from a reduction of a complete set of lex constraints with a row-wise canonical ordering. In this section, we derive a 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.*

The following matrix describes the columnwise ordering with an odd number of columns:

$$\begin{array}{ccccc} 1 & 2n & 2n+1 & \ldots & (m-1)n+1 \\ 2 & 2n-1 & 2n+2 & & (m-1)n+2 \\ 3 & 2n-2 & 2n+3 & & (m-1)n+3 \\ \vdots & \vdots & \vdots & & \vdots \\ n & n+1 & 3n & \ldots & mn \end{array}$$

The pattern for an even number of columns is similar, except that the final column will read upwards rather than down.

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*

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

and finishing with

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

if m is odd and

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

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

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

If m is even then these are:

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

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

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

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 k and $k+1$ and fixes everything else gives

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

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 k and $k+1$ and fixes everything else gives a constraint of the form

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

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:

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

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

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

Rule 1 simplifies $x_{ai}x_{(a+1)i} \leq_{\rm lex} x_{(a+1)i}x_{ai}$ 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:

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

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:

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

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 on columns by allowing each column to be less than or equal to the column two to it's right, it has however already been shown that this has no effect on propagation (Carlsson and Beldiceanu 2002).

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

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

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:

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

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):

$$\begin{array}{ccc} x_{11} & x_{21} & x_{31} \\ x_{12} & x_{22} & x_{32} \\ x_{13} & x_{23} & x_{33} \\ x_{14} & x_{24} & x_{34} \end{array}$$

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

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.

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 the results for each benchmark are presented after a brief description of the problem class.

Lex Method	**Double**		**Snake**	
Search Order	Row	Snake	Row	Snake
(v, k, λ)				
(7, 3, 5)	8.02	8.38	7.42	6.78
(7, 3, 6)	70.47	75.86	61.93	50.47
(7, 3, 20)	0.52	0.47	0.02	0.02
(7, 3, 30)	2.80	2.53	0.03	0.02
(7, 3, 40)	9.14	8.58	0.03	0.03
(7, 3, 45)	16.09	14.94	0.04	0.03
(7, 3, 50)	25.11	23.70	0.06	0.05

Figure 6: A comparison of search times, in seconds, on the BIBD problem. The searches above the double line are for all solutions, whilst those under are for one solution.

The first problem class chosen is a standard benchmark, the balanced incomplete block design problem. A *balanced incomplete block design* (BIBD) (Meseguer and Torras 1999) is a $v \times b$ Boolean matrix, with the columns summing to k, the rows summing to r, and exactly λ positions where two rows both have a 1, for any pair of rows. For example, here is a BIBD with parameters $(7, 7, 3, 3, 1)$:

$$\begin{array}{ccccccc} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{array}$$

The parameters v, k and λ of the BIBD fix the values of b and r, so the problem is to find all (or one) BIBDs with specified values of (v, k, λ). Given a (non-) solution, it is possible to freely permute all of the rows of the matrix to get other (non-) solutions, and it is also possible to freely reorder the rows, thus double lex may be used to reduce search.

The results in Figure 6 show that snake lex outperforms double lex in every tested case. In the cases where it was possible to find all solutions, snake lex gives a faster node time than double lex, and generally finds fewer symmetrically-equivalent solutions. The single solution cases show a speed up over double lex of several orders of magnitude. This is possibly due to the λ constraint being better suited to a snake search order, which in turn is aided by the presence of the snake lex symmetry breaking constraints and also the fact that the matrix has many more rows than columns.

A second problem class is the *equidistant frequency permutation array* problem (EFPD) (Huczynska 2009). An EFPD is a $c \times q\lambda$ matrix, with entries taken from a set of q symbols. Each symbol occurs λ times in each row of the matrix. Each row is a codeword from an error correcting code, and the *Hamming distance* (the number of positions

Lex Method	**Double**		**Snake**	
Search Order	Row	Snake	Row	Snake
(q, λ, d, c)				
(3, 3, 5, 9)	4.1	4.3	3.6	2.9
(3, 4, 5, 9)	26.1	27.1	15.9	11.5
(3, 5, 5, 10)	45.9	53.5	29.6	20.0
(3, 6, 5, 10)	68.9	76.8	39.7	27.4
(3, 7, 5, 11)	94.5	103.0	54.5	35.0
(3, 8, 5, 12)	124.6	123.4	71.0	43.1

Figure 7: A comparison of search times, in seconds, on the EFPD problem. In each case there are no solutions.

with different symbols) between any two codewords is d. On input (q, λ, d, c) the problem is to find one (or all) EFPDs with those parameters. In order to give a fair comparison between the four combinations of search order and lex constraints, we picked six test cases where it is known that there are 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.

In Figure 7 the solve time decreases by around 30% when the lex method is changed from double lex to snake lex, and then decreases by a further 30% when the search order is changed to snake order. 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. We believe that once again this is due to the problem's constraints.

A third problem class is the *fixed length error correcting codes* (FLECC) (Hammons et al. 1994) problem. The *Lee distance* between two codewords $[a_1, \ldots, a_d]$ and $[b_1, \ldots, b_d]$ over the alphabet $\{0, 1, \ldots, q-1\}$ is

$$\sum_{i=1}^{d} \min\{(a_i - b_i) \bmod q, (b_i - a_i) \bmod q\}.$$

Thus for example over the alphabet $[0, 1, 2, 3]$ the distance between $[1, 2, 1, 3]$ and $[0, 3, 2, 1]$ is $1 + 1 + 1 + 2 = 5$. On input a triple (q, d, c) the FLECC problem is to find 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.

The results in Figure 8 show snake lex performing 30 times faster than double lex, in the single solution case, similar to the results in Figure 6. In the all solutions case, the 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 as the instances get bigger.

The final experiment involved solving the *Howell design* (Colbourn and Dinitz 1996) 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 $n(n-1)/2$

Lex Method	**Double**		**Snake**	
Search Order	Row	Snake	Row	Snake
(q, d, c)				
(9, 5, 5)	23.34	8.23	13.09	0.83
(8, 6, 4)	0.73	0.74	0.53	0.52
(15, 5, 22)	0.77	0.72	0.41	0.39
(20, 5, 30)	3.28	3.28	1.31	1.30
(25, 5, 40)	3.88	4.00	1.44	1.42
(30, 5, 50)	4.56	4.83	1.86	1.70

Figure 8: A comparison of search times, in seconds, on the FLECC problem. The test case above the double line searches for a single solution, whilst those under search for all solutions.

Lex Method	**Double**		**Snake**	
Search Order	Row	Snake	Row	Snake
(s, n)				
(4, 9)	0.98	0.97	1.19	1.23
(5, 11)	28.18	27.22	29.65	30.27
(6, 13)	1148.3	1153.8	1179.6	1352.9

Figure 9: A comparison of search times, in seconds, on insoluble instances of the Howell design problem.

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. A series of instances that were known to be insoluble (due to arithmetic constraints on the parameters) were tested.

Figure 9 shows that, in this case, snake lex is slightly slower than double lex. One reason for this could be that Howell Designs are square. It is also the case that the amount of symmetry involved is very small. With the exception of the empty symbol, every entry in the matrix is distinct. Thus with minor modifications to the model, Puget's *all different* symmetry breaking lex constraints (Puget 2005) could be used to break the symmetry. The empty symbol appears very infrequently, so the effort used to treat all of its occurrences as symmetric and then break the symmetries may be unnecessary. This experiment shows that snake lex is not a silver bullet for row and column symmetry breaking.

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

Carlsson, M., and Beldiceanu, N. 2002. Arc-consistency for a chain of lexicographic ordering constraints. Technical Report T2002-18, SICS.

Colbourn, C. J., and Dinitz, J. H. 1996. *The CRC Handbook of Combinatorial Designs*. Florida, US: CRC Press, Inc.

Crawford, J.; Ginsberg, M.; Luks, E.; and Roy, A. 1996. Symmetry-breaking predicates for search problems. *Proc. KR* 148–159.

Flener, P.; Frisch, A. M.; Hnich, B.; Kiziltan, Z.; Miguel, I.; Pearson, J.; and Walsh, T. 2002. Breaking row and column symmetries in matrix models. *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming* 462–476.

Frisch, A. M., and Harvey, W. 2003. Constraints for breaking all row and column symmetries in a three-by-two-matrix. *Proc. Symcon*.

Grayland, A.; Jefferson, C.; Miguel, I.; and Roney-Dougal, C. 2009. Minimal ordering constraints for some families of variable symmetries. *Annals Math. Artificial Intelligence*. To appear.

Hammons, A. R.; Vijay Kumar, P.; Calderbank, A. R.; Sloane, N. J. A.; and Sol, P. 1994. The z 4 -linearity of kerdock, preparata, goethals, and related codes. *IEEE Trans. Inform. Theory*.

Huczynska, S. 2009. Equidistant frequency permutation arrays and related constant composition codes. Circa Preprint 2009/2.

Meseguer, P., and Torras, C. 1999. Solving strategies for highly symmetric csps. *IJCAI*.

Öhrman, H. 2005. Breaking symmetries in matrix models. MSc Thesis, Dept. Information Technology, Uppsala University.

Puget, J.-F. 2005. Breaking symmetries in all different problems. *IJCAI* 272–277.