Symmetry Breaking via LexLeader Feasibility Checkers

Justin Yip and **Pascal Van Hentenryck**
Brown University, Box 1910
Providence, RI 02912, USA

Abstract

This paper considers matrix models, a class of CSPs which generally exhibit significant symmetries. It proposed the idea of LexLeader feasibility checkers that verify, during search, whether the current partial assignment can be extended into a canonical solution. The feasibility checkers are based on a novel result by [Katsirelos *et al.*, 2010] on how to check efficiently whether a solution is canonical. The paper generalizes this result to partial assignments, various variable orderings, and value symmetries. Empirical results on 5 standard benchmarks shows that feasibility checkers may bring significant performance gains, when jointly used with DOUBLELEX or SNAKELEX.

1 Introduction

Matrix models are a class of Constraint Satisfaction Problems that often exhibit significant symmetries and effective symmetry-breaking techniques are often critical in solving them in reasonable time. The LEXLEADER method is a common and elegant symmetry-breaking approach: it consists in posting a lex-ordering constraint for each symmetry to ensure that all non-canonical solutions are removed. Unfortunately, even for simple symmetry classes, the LEXLEADER method may generate an exponential number of constraints. A traditional way to overcome this limitation is to use only a subset of the symmetry-breaking constraints, which is the approach adopted in the DOUBLELEX and SNAKELEX methods for matrix models. This paper takes an orthogonal and complementary approach: instead of enumerating all the symmetry-breaking constraints for a symmetry class, it introduces the idea of a LexLeader feasibility checker that succeeds if a partial assignment can be extented into a canonical solution and fails otherwise. The implementation of the feasibility checker exploits a very interesting result from [Katsirelos *et al.*, 2010]: there exists an $O(n!nm \log m)$ algorithm to decide whether a solution is canonical in a $n \times m$ matrix model with row and column interchangeability. The paper shows how to use this algorithm for building LexLeader feasibility checkers. Moreover, the paper shows how LexLeader feasibility checkers can accommodate value symmetries and various variable orderings. The experimental results on 5 standard benchmarks show that LexLeader feasibility checkers may produce huge performance gains and are very robust.

This paper is organized as follows. Section 2 describes the background and notations used in this paper. Section 3 introduces the novel idea of LexLeader Feasibility Checkers. Sections 4–6 present several extensions and improvements to the core idea. Section 7 describes the empirical results and Section 8 concludes the paper.

2 Background

A Constraint Satisfaction Problem (CSP) consists of a set of variables taking their values in a domain and a set of constraints. The problem is to find an assignment of values to variables satisfying all constraints. This paper focuses on matrix models [Flener *et al.*, 2002] with n rows and m columns and variables are usually subscripted with row and column indices $X_{i,j}$. The domains are subsets of $\{1, ..., v\}$. A constraint specifies the allowed combinations of values for a subset of variables. An assignment is a function that maps all variables to values $\alpha(X_{i,j}) = a_{i,j}$ and a partial assignment is a partial function which maps a subset of variables to values. An assignment extends a partial assignment if they agree on the values of variables in the partial assignment.

A symmetry is a permutation of variables or values under which solutions are preserved. A permutation σ is denoted by, say, (23154), meaning $\sigma(1) = 2$, $\sigma(2) = 3$, and so on. This paper mostly focuses on the common symmetry types in matrix models $[X_{i,j}]$: a row symmetry σ_r is a row permutation $[X_{\sigma_r(i),j}]$, a column symmetry σ_c is a column permutation $[X_{i,\sigma_c(j)}]$, and a value symmetry σ_c is a value permutation $[\sigma_v(X_{i,j})]$. Of course, these various symmetries can be applied together, e.g., $[\sigma_v(X_{\sigma_r(i),\sigma_c(j)})]$, making symmetry breaking particularly challenging.

2.1 The LexLeader Method

The LexLeader method is a very common approach for breaking symmetries [Crawford *et al.*, 1996]: it eliminates symmetrically-equivalent solutions by keeping only a predefined canonical solution α. The canonical solution is usually the lexicographically smallest assignment for a predefined variable ordering. Hence, to eliminate a variable symmetry σ, it suffices to post the following constraint:

$$[X_1, ..., X_n] \leq_{lex} [X_{\sigma(1)}, ..., X_{\sigma(n)}].$$

Example 1. *Consider a CSP with two variables* $X_1, X_2 \in \{0, 1\}$ *and a constraint* $X_1 \neq X_2$. *There are two solutions:* $\alpha_1(X_1) = 0, \alpha_1(X_2) = 1$ *and* $\alpha_2(X_1) = 1, \alpha_2(X_2) = 0$. *There is a variable symmetry* $\sigma = (21)$. *Hence, the LexLeader method posts the lex-ordering constraint* $[X_1, X_2] \leq_{lex} [X_2, X_1]$. *The solution* α_2 *violates the ordering constraint and is therefore removed.*

Similarly, to eliminate value symmetry σ, it suffices to post

$$[X_1, ..., X_n] \leq_{lex} [\sigma(X_1), ..., \sigma(X_n)].$$

2.2 The LexLeader Method in Matrix Models

Many matrix models exhibit both row and column interchangeability, a property called *full-interchangeability*. To eliminate symmetries in a fully-interchangeable matrix model with n rows and m columns, the LexLeader method may define a row-wise variable ordering $row([X_{ij}]) \equiv [X_{1,1}, \ldots, X_{1,m}, X_{2,1}, \ldots, X_{2,m}, \ldots, X_{n,m}]$ and post the lex-ordering constraint

$$row([X_{i,j}]) \leq_{lex} row([X_{\sigma_r(i),\sigma_c(j)}])$$

for each row symmetry σ_r and column symmetry σ_c. There are respectively $n!$ and $m!$ different row and column permutations. Hence breaking all symmetries this way is forbiddingly expensive since there are $n!m!$ such lex-ordering constraints. In fact, breaking symmetries in fully-interchangeable matrix models is particularly challenging, since deciding whether a solution to such a model is canonical is already NP-complete [Bessière *et al.*, 2004].

2.3 The DOUBLELEX Method

The DOUBLELEX method is a popular method for breaking symmetries in fully-interchangeable matrix models [Flener *et al.*, 2002].

Specification 1. *The* DOUBLELEX *method takes a matrix model and enforces lex-ordering among pairs of rows and columns.*

$$\bigwedge_{1 \leq i < i' \leq n} [X_{i,1}, ..., X_{i,m}] \leq_{lex} [X_{i',1}, ..., X_{i',m}]$$
$$\wedge \bigwedge_{1 \leq j < j' \leq m} [X_{1,j}, ..., X_{n,j}] \leq_{lex} [X_{1,j'}, ..., X_{n,j'}].$$

The DOUBLELEX method does not break all symmetries.

Example 2. *Consider a* 2×3 *fully-interchangeable matrix model and the following two assignments which satisfy the* DOUBLELEX *constraints:*

$$\begin{array}{ll} 112 & 122 \\ 221 & 211 \end{array}$$

They are symmetrical under $\sigma_r = (21)$ *and* $\sigma_c = (321)$.

In addition, complete filtering of DOUBLELEX constraints is computationally difficult.

Theorem 1 ([Katsirelos *et al.*, 2010]). *Enforcing domain consistency on the* DOUBLELEX *constraints is NP-hard.*

As a result, in practice, as well as in the evaluation section of this paper, the DOUBLELEX constraints are posted as a set of lex-ordering constraints among pairs of rows and columns independently.

2.4 The ROWWISELEXLEADER Method

[Katsirelos *et al.*, 2010] introduced an interesting method for checking if an assignment is a canonical solution to a fully-interchangeable matrix model. The ROWWISELEXLEADER method determines whether there exists a symmetrical solution which is smaller than a given assignment. If such solution exists, by transitivity, the current assignment cannot be canonical. Let S_k be the set of all permutations of $\{1, ..., k\}$.

Specification 2. *The* ROWWISELEXLEADER *method takes an assignment* α *on a matrix model and returns*

$$\exists \sigma_r \in S_r, \sigma_c \in S_c : \\ row([\alpha(X_{\sigma_r(i),\sigma_c(j)})]) <_{lex} row([\alpha(X_{i,j})]).$$

The method is based on the observation that, if there is *only* one type of symmetry, the above test reduces to sorting. As a result, the ROWWISELEXLEADER method first enumerates all row symmetries and, for each of them, sorts the matrix and compares the resulting and original assignments.

Example 3. *Consider a* 2×3 *fully-interchangeable matrix model and the assignment (below on the left)*

$$\begin{array}{ll} 122 & 211 \\ 211 & 122 \end{array}$$

Applying the row symmetry $\sigma_r = (21)$ *produces the assignment (above on the right). Now it remains to check if there exists a smaller assignment under column interchangeability. Sorting the columns produces the assignment*

$$\begin{array}{l} 112 \\ 221 \end{array}$$

which is lexicographically smaller than the original assignment. The original assignment is not canonical.

Theorem 2 ([Katsirelos *et al.*, 2010]). *For a* $n \times m$ *fully-interchangeable matrix model, the* ROWWISELEXLEADER *method runs in* $O(n!nm \log m)$ *time.*

[Katsirelos *et al.*, 2010] also showed that the DOUBLELEX method may leave $n!$ symmetries on some $2n \times 2n$ matrix models. Moreover, by applying Theorem 2 at the leaves of the search tree, they showed empirically that DOUBLELEX may leave a large number of symmetries in some benchmarks.

3 LexLeader Feasibility Checkers

The key idea behind this paper is to turn Theorem 2 into a practical tool for removing symmetries during search. We first generalize Theorem 2 to partial assignments. If a partial assignment α of the first n' rows is such that another partial assignment of the same rows is lexicographically smaller than α, then any solution extending α is not canonical and the subtree corresponding to α may be pruned. Consider Example 3 and assume that the matrix has more than two rows. The partial assignment [(122),(211)] has exactly two rows filled. Since it is not canonical for the submatrix, any solution extending it is not canonical either and the algorithm can backtrack at this stage without trying to extend the assignment.

A LexLeader feasibility checker can directly use the implementation idea behind Theorem 2. Moreover, since the

bottleneck in Theorem 2 is the $n!$ enumeration of the row symmetries, checking partial assignments early in the search, i.e., partial assignments with a small n', will be more efficient and may potentially prune large portions of the search space.

Specification 3 (RowCol Feasibility Checker). ROWCOLFC *takes a partial assignment α of the first n' rows of a matrix model and returns*

$$\exists \sigma_r \in S_{n'}, \sigma_c \in S_c : \\ row_{n'}([\alpha(X_{\sigma_r(i),\sigma_c(j)})]) <_{lex} row_{n'}([\alpha(X_{i,j})]).$$

where $row_{n'}$ only linearize the first n' rows. Note that σ_r only considers the interchangeability among the first n' rows.

Theorem 3. ROWCOLFC *takes $O(n'!n'm \log m)$ time.*

Theorem 4. ROWCOLFC *removes all but the canonical solution in a n by m fully-interchangeable matrix model.*

Proof. We first prove soundness: only non-canonical solutions are removed. Then we prove completeness: all non-canonical solutions are removed.

Soundness: The lex-ordering relation is dominated by the prefix. Consider a partial assignment α of the first n' rows and a solution α_c extending it. If α has a symmetric partial assignment that is strictly smaller lexicographically than under any symmetries $\sigma_r \in S_{n'}$ and $\sigma_c \in S_c$, then α_c also has a symmetric assignment smaller than it. Formally,

$$\forall \sigma_r \in S_{n'}, \sigma_c \in S_c : \\ row_{n'}([\alpha(X_{\sigma_r(i),\sigma_c(j)})]) <_{lex} row_{n'}([\alpha(X_{i,j})]) \\ \Rightarrow row([\alpha_c(X_{\sigma_r(i),\sigma_c(j)})]) <_{lex} row([\alpha_c(X_{i,j})]).$$

ROWCOLFC returns true when α cannot be extended into a canonical solution and no canonical solutions are removed.

Completeness: When the partial assignment is complete, ROWCOLFC is equivalent to ROWWISELEXLEADER. ☺

For those cases in which the domain size v is much smaller than the number of rows and columns. In these cases, the running time of the algorithm can be improved with a bucket sort, reducing the complexity by a factor of $\log m$.

Theorem 5. ROWCOLFC *takes $O(n'!n' \max(m, v))$ time.*

4 Variable Orderings

Canonical solutions depend on a pre-defined variable ordering. This section shows how to generalize LexLeader feasibility checkers to different variable orderings. In the literature, most models apply either a row-wise or column-wise canonical ordering. Recently, [Grayland *et al.*, 2009] introduced a very interesting variable ordering called SNAKELEX. This section restricts attention to row-wise SNAKELEX ordering since the column-wise counterpart is essentially equivalent. The SNAKELEX ordering orders variables in a snake fashion. It takes variables from left to right in the first row, from right to left in the second, from left to right again in the third, and all the way until the last row. Empirical results in [Grayland *et al.*, 2009; Katsirelos *et al.*, 2010] demonstrated that SNAKELEX sometimes breaks more symmetries. A LexLeader feasibility checker can be naturally defined for the SNAKELEX ordering.

Definition 1 (Snake Linearization). $snake([X_{ij}]) \equiv [X_{1,1}, ..., X_{1,c}, X_{2,c}, X_{2,c-1}, ..., X_{2,1}, X_{3,1}, ...]$

Specification 4 (RowCol-Snake Feasibility Checker). ROWCOL_SNAKEFC *takes a partial assignment α of the first n' rows of a matrix model and returns*

$$\exists \sigma_r \in S_{n'}, \sigma_c \in S_m : \\ snake_{n'}([\alpha(X_{\sigma_r(i),\sigma_c(j)})]) <_{lex} snake_{n'}([\alpha(X_{i,j})]).$$

It is not difficult to see that ROWCOL_SNAKEFC has the same time complexity as ROWCOLFC. For instance, with the conventions used in this paper, it suffices to negate the even rows and to apply ROWCOLFC.

5 Value Symmetries

This section generalizes LexLeader feasibility checkers to value symmetries, starting with value interchangeability.

Definition 2 (ValRowCol Feasibility Check). VALROWCOLFC *takes a partial assignment α of the first n' rows of a matrix model and returns*

$$\exists \sigma_r \in S_{n'}, \sigma_c \in S_c, \sigma_v \in S_v : \\ row([\alpha(\sigma_v(X_{\sigma_r(i),\sigma_c(j)}))]) <_{lex} row([\alpha(X_{i,j})]).$$

The implementation for VALROWCOLFC adds an extra layer of value permutation to the top of ROWCOLFC.

Theorem 6. VALROWCOLFC *takes $O(v!n'!n' \max(m, v))$ time.*

Example 4. *Consider the partial assignment in a matrix model with row, column, and value interchangeability,*

$$112233$$
$$121233$$

To check whether it can be extended into the canonical solution, we enumerate all value permutations and apply ROWCOLFC. *For instance, $\sigma_v = (231)$ produces the submatrix*

$$223311$$
$$232311$$

ROWCOLFC *returns true and the partial assignment cannot be extended into a canonical solution.*

This approach is not limited to value interchangeability only; the same principle can be applied to any kind of value symmetries. The key is simply to enumerate all but one type of symmetry and to exploit the semantics of the remaining one. We illustrate the approach by presenting a feasibility checker for a specific value symmetry class.

Definition 3 (Error Correcting Code, Lee Distance (ECCLD)). *The problem is to find d codewords of length-q that drawn from 4 symbols $(1, 2, 3, 4)$ such that the Lee Distance between every-pair of codeword is exactly c. The Lee Distance between two symbols a, b is $\min(|a - b|, 4 - |a - b|)$.*

The ECCLD problem can be modelled as a matrix model. It has row and column symmetries and an interesting class of value symmetries. Indeed, the values are not interchangeable but the symmetry class Σ_{lee} contains 8 symmetries:

$$\{(1234), (1432), (2143), (2341), (3214), (3412), (4123), (4321)\}$$

The value symmetries apply to each column independently, since the only constraint is the Lee distance between corresponding columns in each row.

Example 5. *The two ECCLD solutions*

$$\begin{array}{cc} 1122 & 1111 \\ 2434 & 2223 \end{array}$$

are symmetric. The first column is obtained by identity, the second by (1432)*, the third by* (4123)*, and the last by* (2143)*.*

A LexLeader feasibility checker for the ECCLD problem can be obtained by enumerating all row and column symmetries and leaving the value symmetry to the sorting step. This is more efficient than enumerating all value symmetries since there are 8^m of them.

Specification 5 (RowColLee Feasibility Checker). ROWCOLLEEFC *takes a partial assignment* α *of the first* n' *rows of a matrix model and returns*

$$\exists \sigma_r \in S_{n'}, \sigma_c \in S_c, \sigma_v \in \Sigma_{lee} : \\ row_{n'}([\alpha(\sigma_v(X_{\sigma_r(i),\sigma_c(j)}))]) <_{lex} row_{n'}([\alpha(X_{i,j})]).$$

Theorem 7. ROWCOLLEEFC *runs in* $O(n!m!nm)$ *time.*

Proof. First, we enumerate all possible row and column symmetries: there are $n!m!$ of them. The resulting matrices only contain value symmetries and the task is to determine if there exists a value symmetry for each column that would produce a new assignment lexicographically smaller than α. Let $Y_{ij} = X_{\sigma_r(i),\sigma_c(j)}$ for $1 \leq i \leq n, 1 \leq j \leq m$ be such a matrix. For each column j, we introduce a variable $Z_j \in \{0, ..., 7\}$ to denote its possible value symmetries in Σ_{lee}. We perform a row-wise scan of the matrix and, for every Y_{ij}, we check if there exists a value symmetry in Z_j yielding a smaller value than X_{ij}. If such a symmetry is found, α is not canonical solution. Otherwise, we remove all value symmetries in Z_j that would yield a larger value than X_{ij} (the remaining values in Z_j preserves the values of $Y_{1j}, \ldots, Y_{ij}$). The whole matrix needs to be scanned only once and the check in each cell takes $O(1)$ time (we only need to index each symmetry pattern with the value of Y_{ij}). The total runtime is therefore $O(n!m!nm)$. ☺

6 Practical Considerations

The earlier section illustrated a key aspect of the approach: one can choose which symmetries to enumerate to obtain the best performance. For instance, on some problems, it may be more appropriate to enumerate the value symmetries first, while it may be unpractical to do so in others.

In practice, it may not be cost-effective to break all symmetries systematically. For instance, one can restrict the feasibility checker to the first k rows of the model, reducing the running time to $O(k!k \max(m, v))$. On some problems, this may significantly improve the performance of the approach. Once again, it is useful to note that the earlier LexLeader feasibility checks cost less and may prune large portions of the search tree. This is a nice property of the approach.

Finally, it is always possible to use several feasibility checkers simultaneously capturing different combinations of symmetries, provided that they use the same variable ordering. For instance, in a fully-interchangeable matrix model with value symmetries, one may use one checker for row and value symmetries, another for column and value symmetries, and a third for row and column symmetries.

7 Experimental Evaluation

This section evaluates the performance of LexLeader feasibility checkers empirically. The primary goal is to assess the effectiveness of the approach, i.e., whether the reduction in search space outweighs the time spent in the feasibility checkers. Five benchmark problems were used, most of which can be found in the CSPLib. They are concerned with finding either all solutions or the optimal solution. All models and feasibility checkers are implemented in the COMET system. Our experiments are run on a Core2Duo 2.4GHz with 4GB of memory. The symbol $\cdot$ indicates a timeout of 1800 seconds. Unless otherwise specified, variables are labeled in the symmetry-breaking order and values are tried in increasing order.

Equidistant Frequency Permutation Array problem (EFPA) The task is to find a set of v codewords drawn from q symbols and each symbol appears for exactly λ times such that the Hamming distance between every pair of codeword is exactly d. The non-boolean model in [Huczynska *et al.*, 2009] is used. The model is a $v \times q\lambda$ matrix of variables with domain $\{1, ..., d\}$. There are three main classes of symmetry: row interchangeability, column interchangeability, and value interchangeability. In [Katsirelos *et al.*, 2010], it was shown that completely breaking row and column interchangeability significantly reduces the number of solutions found. Our evaluation confirms this and pushes it further: completely breaking row, column, and value interchangeability achieves the best runtime performance.

The experiments compare several approaches. DOUBLELEX and SNAKELEX post static symmetry-breaking constraints, while ROWCOL-ROWWISE, ROWCOL-SNAKE, VALROWCOL-ROWWISE, and VALROWCOL-SNAKE add a feasibility checker on the top of the static model. ROWCOL only considers row and column symmetries: all row symmetries are first enumerated and columns are then sorted as in Theorem 3. VALROWCOL considers all symmetries: all value and row symmetries are enumerated and the columns are sorted (as in Theorem 6). The sorting uses a specific variable ordering (either ROWWISE or SNAKE), producing four feasibility checkers.

Table 1 presents the results. The feasibility checkers VALROWCOL break all symmetries, find the fewest solutions and are the fastest. The improvements are more than 3 orders of magnitude when compared with the static methods and many times faster than the feasibility checkers breaking only row and column symmetries. The difference between the two variable orderings ROWWISE and SNAKE is small compared to the overall improvement, with a slight avantage to SNAKE. Note that both orderings achieve the same number of non-symmetric solutions since the choice of variable ordering has no effect on the number of solutions when the symmetry-breaking method is complete.

Balanced Incomplete Block Design (BIBD) The experiments use the boolean fully-interchangeable matrix model from [Flener *et al.*, 2002] and compare static methods and the LexLeader feasibility checkers. We label large value

first. In some large instances, the number of rows becomes very large ($v = 10$ is a matrix with 10 rows and requires $10! = 3,628,800$ permutations). Hence, the experiments also evaluate the performance of the feasibility checker in which only the first k rows in the matrix are enumerated, for some specified value of k. Table 2 presents the result. The complete feasibility checkers return the fewest solutions and are generally faster than the static method. Checkers with a row limitation achieves the most robust results: they are up to 8 times faster on large instances (ROWWISE on (22,7,2)) and only slightly slower on others.

Cover Array problem (CA) The experiments use the integrated model in [Hnich *et al.*, 2006] which has row, column, and value symmetries. The traditional comparisons are performed. However, since in earlier benchmarks, the impact of the variable ordering was negligible among complete checkers, only the ROWWISE ordering was considered for this, and subsequent, problems. Table 3 gives the results and the complete method is generally the fastest.

Error Correcting Code, Lee Distance This is CSPLib 036, an optimization problem whose the goal is to find the maximum number m of codeword of length n drawn from 4 symbols such that the Lee distance between every pair of codewords is exactly c. The decision problem is to find m codewords satisfying the constraints. The search starts with $m = 1$ and increases m by 1 each time. Optimality is proven when no solution is found. The model has both row and column symmetries and value symmetries discussed earlier in the paper. It is forbiddingly expensive to enumerate all possible value symmetries for each column. Instead the experiments use a number of feasibility checkers dealing with different combinations of symmetries. The results show that this approach dramatically reduces the search space.

More precisely, LEEROWCOL enumerates all the value symmetries globally (like in value interchangeability), meaning that it ignores that each column can have its own symmetry. ROWLEE implements the algorithm from Theorem 7, except that the column symmetries are not enumerated. Both checkers have row limitation as well. Table 4 depicts the results. Due to its huge symmetry size, static symmetry-breaking approaches only solve a few small instances. ROWLEE produces significant improvements in performance: On instance $(5, 6)$, it reduces the time from 600 seconds to a fraction of a second. Overall, the combination of LEEROWCOL and ROWLEE produces the best results as the rightmost column indicates. The performance improvements on this benchmark are spectacular.

Error Correcting Code, Hamming Distance This is a common benchmark problem for set variables and the experiments use the model from [Sadler and Gervet, 2008] and length-lex constraints from [Yip and Van Hentenryck, 2009] enhanced with a few redundant constraints. The model has both row and column symmetry and the experiments compare the static method, the feasibiity checkers, and the results in [Gange *et al.*, 2010]. Table 5 gives the results, indicating that the feasibility checker achieves the best overall results.

8 Conclusion

This paper proposed the idea of LexLeader feasibility checkers that verify, during search, whether the current partial assignment can be extended into a canonical solution. The feasibility checkers are based on a result by [Katsirelos *et al.*, 2010] on how to check efficiently whether a solution is canonical. This paper showed how to generalize this result to partial assignments, various variable orderings, and value symmetries. Several checkers combining different types of symmetries can be used simultaneously, instead of tackling all symmetries together which may be prohibitive. Empirical results on 5 standard benchmarks showed that feasibility checkers may bring significant, sometimes spectacular, performance gains.

References

[Bessière *et al.*, 2004] C. Bessière, E. Hebrard, B. Hnich, and T. Walsh. The complexity of global constraints. In *AAAI*, 2004.

[Crawford *et al.*, 1996] J. Crawford, M. Ginsberg, E. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In *KR*, 1996.

[Flener *et al.*, 2002] P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. In *CP*, 2002.

[Gange *et al.*, 2010] G. Gange, P.J. Stuckey, and V. Lagoon. Fast set bounds propagation using a bdd-sat hybrid. In *JAIR*, 2010.

[Grayland *et al.*, 2009] A. Grayland, I. Miguel, and C.M. Roney-Dougal. Snake lex: An alternative to double lex. In *CP*, 2009.

[Hnich *et al.*, 2006] B. Hnich, S. Prestwich, E. Selensky, and B. Smith. Constraint models for the covering test problem. *Constraints*, 11(2-3), 2006.

[Huczynska *et al.*, 2009] S. Huczynska, P. McKay, I. Miguel, and P. Nightingale. Modelling equidistant frequency permutation arrays: An application of constraints to mathematics. In *CP*, 2009.

[Katsirelos *et al.*, 2010] G. Katsirelos, N. Narodytska, and T. Walsh. On the complexity and completeness of static constraints for breaking row and column symmetry. In *CP*, 2010.

[Yip and Van Hentenryck, 2009] J. Yip and P. Van Hentenryck. Evaluation of Length-Lex Set Variables. In *CP*, 2009.

[Sadler and Gervet, 2008] A. Sadler and C. Gervet. Enhancing set constraint solvers with lexicographic bounds. *J. Heuristics*, 14(1), 2008.

	DoubleLex		SnakeLex		RowCol-RowWise		RowCol-Snake		ValRowCol-RowWise		ValRowCol-Snake	
(q, λ, d, v)	#s	Time	#s	Time	#s	Time	#s	Time	#s	Time	#s	Time
(3, 3, 2, 3)	6	0.01	6	0.01	6	0.01	6	0.01	**1**	**0.01**	**1**	**0.01**
(4, 3, 3, 3)	16	0.07	16	0.06	8	0.05	8	0.05	**2**	**0.03**	**2**	**0.03**
(4, 4, 2, 3)	12	0.02	12	0.02	12	0.03	12	0.03	**1**	**0.02**	**1**	**0.02**
(3, 4, 6, 4)	11215	27.49	10760	24.05	1427	13.38	1427	14.79	**263**	**3.79**	**263**	3.93
(4, 3, 5, 4)	61267	329.97	58582	221.43	8600	117.3	8600	96.88	**371**	8.62	**371**	**8.11**
(4, 4, 5, 4)	72309	682.05	66977	422.14	9696	252.15	9696	187.46	**419**	15.83	**419**	**15.68**
(5, 3, 3, 4)	21	1.56	20	0.76	5	1.04	5	0.55	**1**	0.19	**1**	**0.15**
(3, 3, 4, 5)	71	0.69	71	0.55	18	0.39	14	0.38	**4**	**0.15**	**4**	**0.15**
(3, 4, 6, 5)	77535	662.7	71186	512.21	4978	130.33	4876	128.22	**864**	29.88	**864**	**27.26**
(4, 3, 4, 5)	2708	77.52	2754	45.06	441	27.42	447	20.08	**27**	2.98	**27**	**2.45**
(4, 4, 2, 5)	12	0.07	14	0.05	12	0.24	12	0.24	**1**	**0.06**	**1**	**0.06**
(4, 4, 4, 5)	4752	137.03	5354	83.34	717	54.55	822	42.43	**45**	5.29	**45**	**4.85**
(4, 6, 4, 5)	7662	253.85	21782	181.09	819	96.57	3017	117.8	**51**	8.98	**51**	8.27
(5, 3, 4, 5)	24619	1731.65	28214	818.31	3067	573.38	3523	337.19	**43**	15.59	**43**	**11.18**
(6, 3, 4, 5)	·	·	·	·	·	·	·	·	**58**	69.91	**58**	**45.36**

Table 1: Equidistant Frequency Permutation Array problem

	DoubleLex		SnakeLex		RowCol-RowWise		RowCol-Snake		RowCol-RowWise (k=8)		RowCol-Snake (k=8)	
(v, k, λ)	#s	Time	#s	Time	#s	Time	#s	Time	#s	Time	#s	Time
(5, 2, 7)	**1**	**0.01**	**1**	0.03	**1**	0.05	**1**	0.1	**1**	0.05	**1**	0.1
(5, 3, 6)	**1**	**0.01**	**1**	**0.01**	**1**	0.02	**1**	0.02	**1**	0.02	**1**	0.02
(6, 3, 4)	21	**0.02**	25	0.03	**4**	0.09	**4**	0.1	**4**	0.1	**4**	0.14
(6, 3, 6)	134	**0.14**	146	0.22	**6**	0.18	**6**	0.25	**6**	0.21	**6**	0.27
(7, 3, 5)	33304	17.95	85242	51.78	**109**	**4.49**	**109**	8.05	**109**	5.03	**109**	8.98
(7, 3, 6)	250878	177.29	566230	452.1	**418**	**19.08**	**418**	38.45	**418**	21.54	**418**	40.88
(7, 3, 7)	1460332	1315.66	·	·	**1508**	**83.29**	**1508**	182.92	**1508**	92.95	**1508**	193.42
(8, 4, 6)	2058523	1341.93	·	·	**2310**	**73.35**	**2310**	150.29	**2310**	82	**2310**	153.23
(10, 3, 2)	724662	281.83	·	·	**960**	74.83	**960**	341.98	12563	**43.84**	14420	203.8
(10, 5, 4)	8031	18.69	13069	78.33	**21**	2.14	**21**	7.61	68	**1.91**	89	8.0
(22, 7, 2)	**0**	11.12	**0**	85.1	**0**	23.21	**0**	14.23	**0**	**2.86**	**0**	14.0

Table 2: Balanced Incomplete Block Design Problem

	DoubleLex		RowCol-RowWise		ValRowCol-RowWise	
(t, k, g, b)	#s	Time	#s	Time	#s	Time
(2, 3, 2, 4)	2	**0.01**	2	**0.01**	**1**	**0.01**
(2, 3, 2, 5)	15	**0.01**	8	**0.01**	**4**	**0.01**
(2, 3, 3, 9)	12	**0.01**	6	0.01	**3**	**0.01**
(2, 3, 3, 10)	368	0.14	104	0.09	**21**	**0.05**
(2, 3, 3, 11)	6824	2.33	1499	1.26	**271**	**0.46**
(2, 3, 4, 16)	576	0.33	150	0.25	**15**	**0.32**
(2, 3, 4, 17)	43368	23.52	8236	11.93	**391**	**3.13**
(2, 3, 5, 25)	161280	134.91	27280	**77.71**	**283**	92.91
(2, 4, 2, 5)	10	0.01	5	0.01	**3**	**0.01**
(2, 4, 2, 7)	2285	1.06	333	0.32	**175**	**0.19**
(2, 4, 3, 9)	36	0.03	5	**0.02**	**2**	**0.02**

Table 3: Cover Array Problem

		DoubleLex		LeeRowCol (k=8)		RowLee (k=5)		LeeRowCol (k=8) + RowLee (k=5)	
(n, c)	Opt	Time	Fails	Time	Fails	Time	Fails	Time	Fails
(4, 2)	8	1.44	21327	0.37	2882	**0.11**	253	0.12	**240**
(4, 4)	8	101.17	1834887	9.38	93085	1.29	4280	**0.88**	**2761**
(4, 6)	2	0.05	1211	0.03	337	**0.01**	**77**	**0.01**	**77**
(5, 2)	10	13.98	159808	1.75	11826	0.27	516	**0.26**	**435**
(5, 4)	8	·	·	425.09	4615063	13.87	63425	**6.05**	**27492**
(5, 6)	6	649.75	13466477	37.58	469869	0.4	2597	**0.32**	**1861**
(5, 8)	2	0.08	2152	0.04	509	**0.01**	**115**	0.02	**115**
(6, 2)	12	300.86	3114351	8.49	47859	0.69	1133	**0.53**	**752**
(6, 4)	8	·	·	·	·	39.39	246749	**12.79**	**79602**
(6, 8)	4	92.8	2187585	8.82	129252	**0.05**	584	0.07	**553**
(7, 2)	14	·	·	35.73	166890	1.85	2842	**1.09**	**1343**
(7, 4)	8	·	·	·	·	73.78	522444	**20.17**	**132809**
(8, 2)	16	·	·	156.45	599460	5.5	8011	**2.64**	**2701**
(8, 4)	8	·	·	·	·	154.18	972759	**30.55**	**183120**

Table 4: Error Correcting Code, Lee Distance

		BDD-SAT		Length-Lex + redundant		RowCol (k=7) + redundant	
(l, d, w)	Opt	Time	Fails	Time	Fails	Time	Fails
(8, 4, 4)	14	0.03	61	**0.03**	**0**	0.05	**0**
(9, 4, 3)	12	0.06	300	**0.03**	**1**	0.05	**1**
(9, 4, 6)	12	0.06	256	**0.05**	**3**	0.08	**3**
(10, 6, 5)	6	0.03	145	**0.03**	**14**	0.04	**14**
(9, 4, 4)	18	1.04	4466	**0.11**	**23**	**26**	
(10, 4, 3)	13	2.37	16755	**0.08**	42	0.11	**42**
(10, 4, 4)	30	14.66	34503	**0.19**	31	0.27	**31**
(10, 4, 5)	36	104.39	184051	4.24	6425	**1.34**	**1127**
(10, 4, 6)	30	48.96	131379	5.52	8067	**1.48**	**1163**
(10, 4, 7)	13	1.96	13533	**0.11**	25	0.17	**15**

Table 5: Error Correcting Code, Hamming Distance