On the complexity and completeness of static constraints for breaking row and column symmetry. CP 2010

George Katsirelos, Nina Narodytska, and Toby Walsh.

December 11, 2010

Presentation of the paper and extension to:
Row and Column Symmetry in Generalized Sudoku
Jenny Lam

1 Introduction

The paper [KNW10] presents several results on breaking variable symmetry in constraint problems possessing row and column symmetry. In this paper, we will highlight the main results of the paper [KNW10] and apply them to investigate two methods of breaking symmetry in Sudoku problems.

A (variable) *symmetry breaking constraint* is a constraint such that, when added to a constraint problem, preserves the symmetry classes, but reduces the number of solutions in each class. Ideally, we would like a symmetry-breaking constraint to be *complete*. That is, we would like it to preserve a single solution in each symmetry class.

LexLeader is such a constraint [RBW06]. By imposing a linear ordering on the domain of the variables and a linear ordering on the variables themselves, LexLeader can choose as the representative of each symmetry class the lexicographically least solution, thereby achieving complete symmetry breaking.

The paper [KNW10] focuses on the class of problems possessing row and column symmetry.

Definition 1 (Row and column symmetry)**.** A constraint satisfaction problem (X, D, C) in which each variable has the same domain satisfies row and column symmetry if the variables can be arranged in a matrix:

$$\begin{matrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \dots & x_{mn} \end{matrix}$$

and all row and column permutations preserve solutions.

To break these symmetries, [KNW10] proposes three schemes:

1. RowWiseLexLeader: from the matrix configuration, we rearrange the variables into a linear ordering by lining the rows end-to-end and applying LexLeader. As was shown in the paper, checking satisfaction of this constraint takes $O(m!mn \log n)$ for an $m \times n$ matrix.

2. DoubleLex: we require that the rows be lexicographically ordered, and the columns be likewise lexicographically ordered. Unfortunately, as the paper shows, this constraint is not complete and can, in one particular instance, leave $n!$ solutions in a symmetry class for a problem with a $2n \times 2n$ matrix.

3. SnakeLex: the first column should be lexicographically less than the second and third columns. The reverse of the second column should be lexicographically less than the reverse of the third and fourth columns, and so on. Consecutive rows must satisfy an entwined lexicographical ordering. These sets of

constraints are chosen to approximate the SnakeLexLeader constraint [GMRD09], which linearizes the matrix in a snakelike fashion before applying LexLeader. [KNW10] again shows that, in one worse-case scenario, SnakeLex leaves in $O(4^n/\sqrt{n})$ symmetric solutions to a $2n \times (2n+1)$ matrix.

Despite their incompleteness, DoubleLex and SnakeLex were shown to perform reasonably well at reducing the size of symmetry classes. We show that the main advantage of posting these constraints over the complete RowWiseLexLeader lies in the complexity of checking satisfaction:

	RowWiseLexLeader	DoubleLex	SnakeLex
Completeness	Yes	No	No
Check satisfaction	$O(m!mn \log n)$	$O(mn)$	$O(mn)$

To show that a solution of an $m \times n$ matrix satisfies DoubleLex, we first check that its rows are lexicographically ordered. In other words, we check that the first row is less than the second, the second row is less than the third, and so forth. Each check takes $O(n)$ and there are $m-1$ such constraints to check. So it takes $O(mn)$ to check that the rows are lexicographically ordered. Then we check that the columns are ordered, also in $O(mn)$. This shows that DoubleLex can be checked in $O(mn)$. Since SnakeLex is also a decomposition into row and column constraints, a similar analysis shows that SnakeLex can likewise be checked in $O(mn)$.

2 Definitions

Definition 2 (Generalized Sudoku)**.** n-Sudoku is a constraint satisfaction problem with n^4 or N^2 variables, where $N = n^2$. The domains of the variables are the numbers 1 through N. The variables are arranged in an $N \times N$ matrix. They satisfy the RowColAllDifferent constraint which requires that the variables of any given row or column must have different values.

The matrix of variables is also subdivided into blocks of n rows called bands and blocks of n columns called stacks.[1] The bands and stacks divides the matrix into $n \times n$ blocks. n-Sudoku also has the BlockAllDifferent constraint which requires that the variables of each block have different values.

For example, standard Sudoku is n-Sudoku where $n = 3$ and $N = 9$.

Definition 3 (Band and stack symmetry)**.** Consider a constraint satisfaction problem with matrix variables divided into row blocks (or bands) of equal height and column blocks (or stacks) of equal width. This problem has *band symmetry* if, within each band, any row permutation preserves solutions, and if any permutation of the bands preserves solutions. The problem has *stack symmetry* if the same condition holds when bands are replaced with stacks.

3 RowWiseLexLeader

Following its proof that checking satisfiability of RowWiseLexLeader takes $O(m!mn \log n)$ for a $m \times n$ matrix, [KNW10] states that

> "This result easily generalizes to when rows and columns are partially interchangeable."

This is exactly the situation described by n-Sudoku's band and stack symmetries. So we propose posting RowWiseLexLeader as a constraint that breaks Sudoku's band and stack symmetries completely.

Theorem 4. Checking satisfaction of RowWiseLexLeader in n-Sudoku is $O((n!n)^4)$.

[1] This terminology is borrowed from the Wikipedia article *Mathematics of Sudoku* (`http://en.wikipedia.org/wiki/Mathematics_of_Sudoku`).

Proof. Let $\vec{a}$ be a complete solution to n-Sudoku, thus satisfying RowColAllDifferent. To check that it satisfies RowWiseLexLeader, this solution must be compared to all the solutions obtained by a row, column, band or stack permutation, or combination thereof. In fact, each symmetric solution can be obtained from $\vec{a}$ precisely by a composition of a single permutation of each kind just described. There are $(n!)^4$ compositions of a band permutation and a row permutation.

Once the band and row permutations are fixed, the columns and stacks do not need to be all permuted. This is because we want to check that $\vec{a}$ is the least representative when ordered rowwise. So to do so, it only needs to be checked against those permutations that satisfy LexLeader for fixed band and row permutations. There is only one such permutation (for fixed band and row permutations) and it is the one obtained by ordering the columns within each stack and by ordering the stacks themselves by treating them as a long column vector formed by putting the columns within the stack end-to-end and ordering those. But because the entries of the first row are all different, this ordering is equivalent to ordering just the top entry in each column and ignoring the rest of it. Within each stack, the entries of the first row are ordered in $n \log n$ time, and these entries, treated as words of length n, are also ordered in $n \log n$. Finally we check that our solution $\vec{a}$ is lexicographically less than this newly obtained solution in $O(N^2)$ by treating them as rowwise vectors. For each fixed band and row permutations, checking that $\vec{a}$ is less than all column and stack permutations takes $O((n \log n)^2 + N^2) = O(N^2) = O(n^4)$. □

It is interesting to note that in the case of complete row and column symmetry as discussed in [KNW10], checking that a solution is the lexicographically least for fixed row permutations is dominated by the step of ordering the columns lexicographically. Using the fact that the first row contains all different entries in the case of n-Sudoku, checking that a solution is the lexicographically least for fixed row and band permutations is dominated by the step of comparing the solution with the column-ordered solution.

Because RowWiseLexLeader is a form of LexLeader, we know that RowWiseLexLeader completely breaks the band and stack symmetries of n-Sudoku. It is also fair to ask whether there are more than one such symmetry class. The answer is yes. Consider the following solution:

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
2	3	1	5	6	4	8	9	7
5	6	4	8	9	7	2	3	1
8	9	7	2	3	1	3	6	4
3	1	2	6	4	5	9	7	8
6	4	5	9	7	8	3	1	2
9	7	8	3	1	2	6	4	5

The transpose is also a solution. However it cannot be obtained by row, column, band and stack permutations only. We can see this by noticing that in this solution, within each block, the digits 1, 2, and 3 are always lined up in a row. So in the transpose, these digits will always be lined up in a column within the same block. But row, column, band and stack permutations preserve this property, and so the two solutions are not band or stack symmetric.

4 DoubleBlockLex

Given that it takes $O((n!n)^4)$ to check satisfaction of RowWiseLexLeader, we would like to attempt to find a more efficient symmetry breaking constraint by perhaps giving up on completeness of the symmetry breaking.

We would like to consider DoubleLex which was proposed in [KNW10] as one possible approach. Unfortunately, such a constraint only makes sense when the matrix of variables enjoys full row and column symmetry. So we adapt the constraint to Sudoku as follows.

Definition 5 (DoubleBlockLex)**.** A solution to n-Sudoku satisfies DoubleBlockLex if

1. the rows within a band are lexicographically ordered; and

2. the columns within a stack are lexicographically ordered; and
3. the bands are lexicographically ordered: compare the bands as vectors of rows assembled end-to-end; and
4. the stacks are lexicographically ordered: compare the stacks as vectors of columns assembled end-to-end.

Just as DoubleLex is not complete [KNW10], neither is DoubleBlockLex.

1	2	4	3
4	3	2	1
2	1	3	4
3	4	1	2

swap rows in bands

4	3	2	1
1	2	4	3
3	4	1	2
2	1	3	4

swap stacks

2	1	4	3
4	3	1	2
1	2	3	4
3	4	2	1

swap bands

1	2	3	4
3	4	2	1
2	1	4	3
4	3	1	2

In this example, symmetric solutions of 2-Sudoku are obtained by row, stack and band permutations. The first and the last solutions are therefore symmetric. They also satisfy DoubleBlockLex. Therefore DoubleBlockLex does not break all band and stack symmetries. On the upside, checking satisfaction for and enforcing DoubleBlockLex can be done efficiently as we now show.

Theorem 6. Checking that a solution of n-Sudoku satisfies DoubleBlockLex is $O(N)$.

Proof. DoubleBlockLex with RowColAllDifferent is equivalent to the following conditions:
1. in the leftmost column, the entries within each band are ordered;
2. in the top row, the entries within each stack are ordered;
3. within the leftmost column, the top entries of the bands are ordered;
4. within the top row, the leftmost entries of the stacks are ordered.

To check that the first condition holds, we check that the top entry of the leftmost column is less than the second entry, that the second entry is less than the third, and so on within each band. This requires $O(N)$ checks. The same reasoning holds for the second condition.

To check that the third condition holds, we check that, in the first column, the top entry of the top band is less than the top entry in the second band, that the top entry in the second band is less that the top entry in the second band, and so on. And similarly for the fourth condition. This requires $O(n)$ checks.

Therefore, checking for satisfaction is dominated by the checks of the first and second conditions. □

Definition 7 (Domain consistency)**.** A constraint that is not necessarily binary is *domain consistent* or DC with respect to a variable x within its scope, if every value a in the domain of x can be extended to an assignment on all other variables within the constraint's scope in a way that is consistent with the constraint. Domain consistency is also known in the literature as *generalized arc consistency* [Dec03].

We first prove a result that clarifies the proof of Theorem 4 of [KNW10], that DC of Order1stRowCol can be enforced in polynomial time and will also be used to prove the complexity of enforcing DC on DoubleBlockLex.

Lemma 8. Consider a constraint graph that is a rooted tree such that all constraints are inequalities directed towards the root. That is, the constraint between a parent variable x and child variable y is $x < y$. Then enforcing DC of all the constraints on this tree is $O(e)$ where e is the number of constraints.

Proof. Given a sequence of inequality constraints between variables with domains bounded by the given values:

$$\begin{array}{c} x_1 < x_2 < x_3 \\ [a,b] \quad [c,d] \quad [e,f] \end{array}$$

the lower bound c depends on a for x_2 to be domain-consistent with respect to x_1 and e depends c for x_3 to be domain-consistent. However, since x_1 does not have a parent variable, its lower bound a does not depend on anything and remains fixed even through enforcement of DC. Similarly, bound f does not change. Upper bound d depends on the value of f and b depends on d.

Procedure `tree-bound-DC`
Input: a constraint satisfaction problem whose constraint graph has the structure of a rooted tree and whose constraints are all inequalities directed towards the root.
Output: an equivalent domain consistent constraint satisfaction problem or notification that the problem is inconsistent
queue ⟵ *root*
stack ⟵ ∅
while the *queue* is non-empty
    node ⟵ next element removed from *queue*
    for each *child* of *node*
        domain(*child*) ⟵ `revise-down`(*node*, *child*)
        if the domain(*child*) is empty, **return** inconsistent
        queue ⟵ *child*
    end for each
    stack ⟵ *node*
end while
while the *stack* is non-empty
    node ⟵ next element removed from *stack*
    if *node* is the *root*, **break**
    parent ⟵ parent(*node*)
    domain(*parent*) ⟵ `revise-up`(*parent*, *node*)
    if the domain(*parent*) is empty, **return** inconsistent
end while
return the constraint problem with modified domains

Subprocedure `revise-down`
Input: *parent* node, *child* node
Output: revised domain of the *child* node
a ⟵ minimum of domain(*parent*)
c ⟵ minimum of domain(*child*)
d ⟵ maximum of domain(*child*)
if $c < a < d$,
    let the minimum bound of domain(*child*) be a
else if $d < a$
    domain(*child*) ⟵ ∅
end if
return the domain(*child*)

Subprocedure `revise-up`
Input: *parent* node, *child* node
Output: revised domain of the *parent* node
d ⟵ maximum of domain(*child*)
a ⟵ minimum of domain(*parent*)
b ⟵ maximum of domain(*parent*)
if $a < d < b$,
    let the maximum bound of domain(*parent*) be d
else if $d < a$
    domain(*parent*) ⟵ ∅
end if
return the domain(*parent*)

These observations justify the algorithm `tree-bound-DC` (see figure) that enforces DC on a constraint problem satisfying the hypotheses of this lemma. `tree-bound-DC` visits every node of the tree at most twice, each time in constant time. So it has complexity linear in the number of nodes or, equivalently for a tree, linear in the number of edges. □

Theorem 9. Enforcing domain-consistency of DoubleBlockLex on n-Sudoku is $O(N)$.

Proof. Assuming that RowColAllDifferent holds, decomposing the global constraint DoubleBlockLex as in the proof of the previous theorem results is a graph with a tree structure rooted at the top left entry of the matrix. The edges of this graph are inequalities directed towards this root. Moreover, there are $n \times (n-1)$ such edges satisfying condition 1, another $n \times (n-1)$ edges satisfying condition 2, n edges for condition 3 and n edges for condition 4. This gives a total of $2n^2$ edges. So by the lemma, domain-consistency of DoubleBlockLex can be enforced in $O(N)$. □

The last two theorems and their proofs suggest that whether a solution to n-Sudoku satisfies DoubleBlockLex is entirely determined by the ordering of the first row and column. A question that naturally arises from this observation is thus:

> Are any two band and stack symmetric solutions that satisfy DoubleBlockLex and have the same first row and first column necessarily equal?

If this is the case, we could devise a symmetry breaking constraint that is complete and efficient: among all symmetric solutions that satisfy DoubleBlockLex, choose the one with the lexicographically least first column. And if there are more than one solution still, choose the one among the remaining solutions with the lexicographically least first row. If the answer to the question above is yes, this representative would be unique among all band and stack symmetric solutions. Also, this representative could be found efficiently because there are at most N^2 symmetric solutions that could satisfy DoubleBlockLex, since each $n \times n$ block can be moved to the top left corner and potentially form a new symmetric DoubleBlockLex compliant solution.

Through exhaustive search, we found that the answer to the question above was yes for 2-Sudoku. However, we were not able to answer the question in the general case.

5 Conclusion

We reviewed the paper [KNW10] and extended its results to the problem of n-Sudoku. In particular, we showed that the RowWiseLexLeader was still complete, though not efficient. We generalized the DoubleLex constraint to a DoubleBlockLex constraint which, while not complete, can be used more efficiently than RowWiseLexLeader. Specifically, checking satisfaction and enforcing DC on DoubleBlockLex can be done in linear time in the width of the Sudoku grid.

6 Acknowledgements

This paper is part of a project for the course CS 275, Constraint Networks, taught by Professor Rina Dechter in Fall 2010 at the University of California, Irvine.

References

[Dec03] Rina Dechter. *Constraint Processing.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[GMRD09] Andrew Grayland, Ian Miguel, and Colva M. Roney-Dougal. Snake lex: an alternative to double lex. In *Proceedings of the 15th international conference on Principles and practice of constraint programming*, CP'09, pages 391–399, Berlin, Heidelberg, 2009. Springer-Verlag.

[KNW10] George Katsirelos, Nina Narodytska, and Toby Walsh. On the complexity and completeness of static constraints for breaking row and column symmetry. *CoRR*, abs/1007.0602, 2010.

[RBW06] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence).* Elsevier Science Inc., New York, NY, USA, 2006.