TLP **14** (3): 363–377, 2014. © Cambridge University Press 2012
doi:10.1017/S1471068412000361 First published online 30 October 2012

Technical Notes
Redundant Sudoku rules

BART DEMOEN
Department of Computer Science, KU Leuven, Belgium
(*e-mail:* `bart.demoen@cs.kuleuven.be`)

MARIA GARCIA DE LA BANDA
Faculty of Information Technology, Monash University, Australia
(*e-mail:* `Maria.GarciaDeLaBanda@monash.edu`)

submitted 15 February 2012; revised 17 July 2012; accepted 17 July 2012

Abstract

The rules of Sudoku are often specified using 27 `all_different` constraints, referred to as the *big* constraints. Using graphical proofs and exploratory logic programming, the following main and new result is obtained: Many subsets of six of these big constraints are redundant (i.e., they are entailed by the remaining 21 constraints), and six is maximal (i.e., removing more than six constraints is not possible while maintaining equivalence). The corresponding result for binary inequality constraints, referred to as the *small* constraints, is stated as a conjecture.

KEYWORDS: Sudoku, *all_different* constraints, inequalities, maximal redundancy

1 Introduction

On the 18th of May 2008, the following question was posted on `rec.puzzles`: "What's the minimum amount of checking that needs to be done to show that a completed 9×9 grid is valid?". We prove that the short answer is: "21 *all_different* constraints." The complete answer shown here is the result of a set of theorems whose proofs are presented in an intuitive graphical representation, together with a set of Prolog programs[1] whose help was welcomed for guiding our intuition, and for dealing with some of the combinatorial explosion resulting from the symmetries of the Sudoku puzzle.

A very common formulation of the Sudoku (Jussien 2007; Wikipedia n.d.) puzzle is as follows: Each 3×3 box, as well as each row and each column, must contain all the numbers from 1 to 9. As a constraint satisfaction problem (CSP), the Sudoku

[1] The relevant programs are available at http://people.cs.kuleuven.be/bart.demoen/sudokutplp. We have used different Prolog systems, including SICStus Prolog, B-Prolog, and hProlog. These programs run in other systems with little change.

puzzle can be modeled using a set of 81 variables x_{ij}, one per row $i \in [1..9]$ and column $j \in [1..9]$, 81 *domain* constraints indicating that the domain of each x_{ij} is $[1..9]$, and 27 *all_different* constraints with 9 variables each (9 constraints for the variables in each of the rows, another 9 for each of the columns, and a final 9 for each of the boxes). We refer to these 27 constraints as *the big constraints* and use the word *Sudoku* in italics to denote the associated CSP model, i.e., the one containing all 27 big constraints together with the 81 domain constraints. An *all_different* constraint can also be formulated as the pairwise binary inequality constraints of its input variables. For example, *all_different*$(\{y_1, y_2, y_3\})$ is logically equivalent to the conjunction of the constraints $y_1 \neq y_2$, $y_1 \neq y_3$, and $y_2 \neq y_3$. We refer to these binary $\neq$–constraints as the *small constraints*. When Sudoku is modeled using small constraints, it is easy to see that each cell is involved in 20 small constraints: 8 in the same box, 6 in the same row, and 6 in the same column. Since there are 81 cells, and each constraint is posted twice, there are in total 810 different small constraints (as opposed to 27 big ones). Whenever a CSP model M specified using (big or small) constraints, together with the 81 domain constraints, is equivalent to *Sudoku* (i.e., it has the same set of solutions), we say M is *Sudoku*.

It was always intuitively clear to us that some of the small constraints must be *redundant*, i.e., entailed by the others. However, the questions "which and what is the size of the largest redundant set of small constraints?" remained to be answered. The situation was even worse for big constraints: when we started this research, it was not even clear to us whether any single big constraint is redundant. Both issues are attacked here: we give a complete answer for the big constraints, and a partial answer for the small constraints.

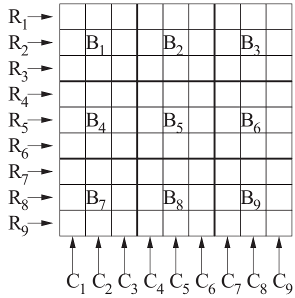
We begin by recalling some common Sudoku-related terminology in Section 2. Section 3 introduces our graphical representation of *Sudoku* modeled with big constraints. This representation significantly simplifies the reasoning required for showing that *some* sets of big constraints with six or less elements are redundant (Section 4). We then describe a Prolog program that systematically applies these two positive lemmas to find *all* sets of redundant big constraints with six or less elements (Section 5). While doing this, we discover seven *negative* lemmas (Section 5.1). The combination of positive and negative lemmas results in a complete classification of all sets of 21 (27 – 6) big constraints (Section 5.2). We then turn to the study of sets of seven big constraints and show that none of them are redundant (Section 5.3). As before, our Prolog program discovers a new negative lemma, whose proof is also presented graphically. We then show that at least 20% of the small constraints can be redundant (Section 6), and conjecture that no more is possible. Finally, in Section 7 we conclude and discuss related work and possible extensions.

2 Terminology

The 27 big constraints in Sudoku correspond to the 27 regions in which its board is usually divided (see the attached picture): the 9 rows, 9 columns, and 9 boxes, whose big constraints will be denoted as $R_1, \ldots, R_9$, as $C_1, \ldots, C_9$, and as $B_1, \ldots, B_9$ respectively. We use the word horizontal (vertical) *chute* to refer to three horizontal

https://doi.org/10.1017/S1471068412000361 Published online by Cambridge University Press

(vertical) boxes. For instance, the boxes associated to constraints B_2, B_5, and B_8 denote a vertical chute.



As mentioned above, we are interested in exploring Sudoku models where some big constraints are missing. In the following, we will use *Missing*(n) to denote the set of Sudoku models that have $27 - n$ big constraints. For example, every model in *Missing*(5) has 22 big constraints (plus, of course, the usual 81 domain constraints).

3 A graphical representation of sets of big constraints

The standard set notation is not visually clear once the number of elements in the set is high. Since we will be dealing mostly with sets of more than 20 big constraints, we have developed a graphical representation of the Sudoku model that we find more useful. This graphical representation always shows the borders of the boxes of a Sudoku board and assumes that all 81 domain constraints are specified in the model. Further, all 27 big constraints are also specified unless they are explicitly represented as missing in the figure. A column, row, or box constraint is represented as missing if it is shaded. Figure 1 shows an example.

The pictures provide a quick and intuitive view into which big constraints are present and not present in the model. Note that the absence of a big constraint does not mean it is violated, simply that it has not been specified in the associated model.

Using the same idea, we can represent a set of big constraints applicable only to a chute (any chute): this is illustrated in Figure 2.

Fig. 1. The left-hand side of the figure shows *Sudoku*, i.e., a CSP model with all domain constraints and all big constraints; the right-hand side shows a model with all domain constraints as usual, but with only 22 out of the 27 big constraints, since C_2, R_5, B_2, B_5, and B_7 are marked as missing.

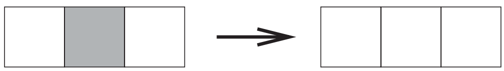
https://doi.org/10.1017/S1471068412000361 Published online by Cambridge University Press

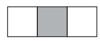
Fig. 2. The left-hand side represents a chute where all row constraints (R_1, R_2, and R_3) and box constraints (B_1, B_2, and B_3) are present, while the middle one is missing B_2 and the right one is missing both B_2 and R_2. Note that, as before, the 27 domain constraints associated to the cells in the chute are assumed to be present in all three pictures.

4 Two constructive lemmas

Let us now prove two positive lemmas, i.e., how a subset of big constraints can be shown to entail another big constraint.

Lemma 4.1
The conjunction of big constraints in $\{R_1, R_2, R_3, B_1, B_3\}$ entails B_2. This is represented graphically by means of the following picture:

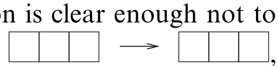


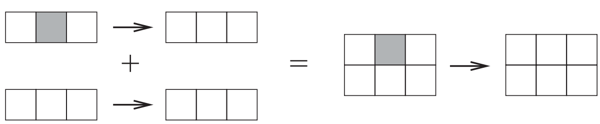
Proof
Let us fill the chute with 27 numbers so that the constraints in  are satisfied. To do so, let us try to place any value $N \in 1..9$ in the chute. Since R_1, R_2, and R_3 are present, there must be exactly one N in each row, which means there must be three Ns in the chute. Since B_1 and B_3 are also present, exactly one of these three Ns must be in box 1 and exactly another one in box 3. This leaves exactly one (the third) N in box 2. Since this holds for any $N \in 1..9, B_2$ also holds. □

The dual of Lemma 4.1 is Lemma 4.2.

Lemma 4.2

Proof
Let us fill the chute with 27 numbers so that the constraints in  are satisfied. To do so, let us again try to place any value $N \in 1..9$ in the chute. Since B_1, B_2, and B_3 are present, there must be exactly one N in each box, which means there must be three Ns in the chute. Since R_1 and R_3 are also present, exactly one of these three Ns must be in row 1 and exactly another one in row 3. This leaves exactly one (the third) N in row 2. As before, this means R_2 also holds. □

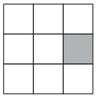
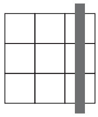
From now on we assume that the graphical representation is clear enough not to require accompanying text. Together with the trivial lemma , the above two lemmas form the building blocks of a corollary and a whole set of theorems: We simply glue several applications of these lemmas to form a new one, as exemplified in the following picture:



https://doi.org/10.1017/S1471068412000361 Published online by Cambridge University Press

We are now ready for our corollary.

Corollary 4.3

 and  are both *Sudoku*.

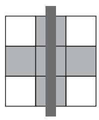
Proof

Glue together twice the trivial lemma with Lemmas 4.1 and 4.2 respectively, and obtain the result immediately. □

Taking into account the symmetries of the puzzle, it follows that every single big constraint is (by itself) redundant, i.e., every model in *Missing*(1) is *Sudoku*! We will see later that this is not true for any other *Missing*(n) with $n > 1$.

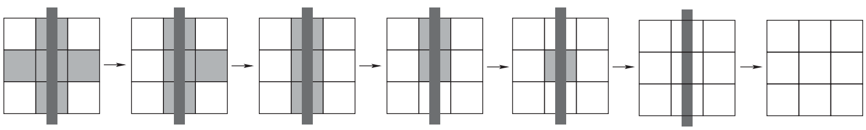
Note that the two lemmas really are *constructive*, i.e., they show how to infer one new big constraint from a set of big constraints. The following two theorems exploit that constructive power to reason further about redundancy.

Theorem 4.4

 is *Sudoku*.

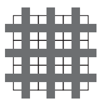
Proof

We prove this by repeatedly using Lemmas 4.1 and 4.2 as follows:



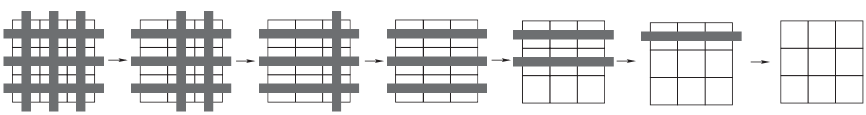
where the first five rewrites use Lemma 4.1, and the last step uses Lemma 4.2. □

Theorem 4.5

 is *Sudoku*.

Proof

We prove this by repeatedly using Lemma 4.2 as follows:



□

https://doi.org/10.1017/S1471068412000361 Published online by Cambridge University Press

```
classify_each([], [], []).
classify_each([Model|MissingN], Stuck, Reducibles) :-
    exhaustively_apply_lemmas(Model, NewModel),
    ( has_all_bigs(NewModel) ->
         Reducibles = [Model|Tail],
         classify_each(MissingN, Stuck, Tail)
    ;
         Stuck = [NewModel|Tail],
         classify_each(MissingN, Tail, Reducibles)
    ).

exhaustively_apply_lemmas(Model, NewModel) :-
    ( apply_lemmaI(Model, ModelI) ->
         exhaustively_apply_lemmas(ModelI, NewModel)
    ; apply_lemmaII(Model, ModelII) ->
         exhaustively_apply_lemmas(ModelII, NewModel)
    ;
         NewModel = Model
    ).
```

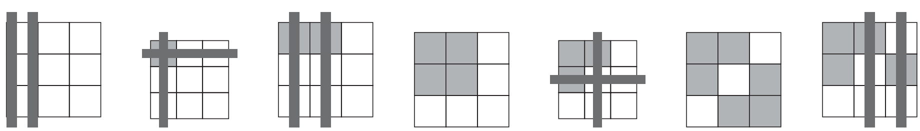
Fig. 3. Program I.

Each of the above two theorems shows a model in *Missing*(6) that is *Sudoku*. While there are many symmetric versions of these theorems, we have chosen those that are visually most pleasing to us. The next section fully classifies *Missing*(6).

5 A full classification of *Missing*(6)

Lemmas 4.1 and 4.2 allow us to add a new big constraint to a set of big constraints while retaining equivalence, as shown in the proof of Theorem 4.4. We use this to implement a Prolog program that attempts to classify all models in *Missing*(6) as either *Sudoku* or not, and whose simplified form is shown Figure 3. Intuitively, the program receives as input in `MissingN` a list with all models in *Missing*(n), for some particular n. Then for each model `Model` of `MissingN`, it exhaustively applies Lemmas 4.1 and 4.2 computing the (possibly reduced) model in `NewModel`. If `NewModel` contains the 27 big constraints (and, thus, it is *Sudoku*) it adds `Model` to the `Reducible` list, and otherwise it adds `NewModel` to the list `Stuck` of models with less than 27 big constraints at which it got stuck. These latter models need special attention.

While the number of models in *Missing*(6) is relatively small (296,010), we can further reduce it by eliminating the spatially symmetric models. We have run[2] complete the Program I over the (reduced) set of *Missing*(n) for $n = 6$ (from which we can also derive the results for $n \in 2..5$). Surprisingly, the program only failed to prove equivalence to *Sudoku* for the following models:



[2] See file `classify.pl` at the already mentioned website; the actual Prolog code has an extra argument collecting the models shown in Appendix B.

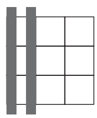
https://doi.org/10.1017/S1471068412000361 Published online by Cambridge University Press

Note that while the last two are models in *Missing*(6), the others (from right to left) are models in *Missing*(5), *Missing*(4), *Missing*(3), and *Missing*(2), which were obtained during the proving process by applying Lemma 4.1 or 4.2 to some model in *Missing*(6). As we prove in the next section, none of these seven models is *Sudoku*, and thus none of the models in *Missing*(6) whose proof got stuck is *Sudoku* either. This is because if a model M in *Missing*(n) is not *Sudoku*, then any model M' in *Missing*(n') where $n' > n$ and the constraints in M' are a subset of those in M, cannot be *Sudoku* either.

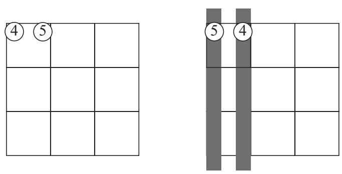
5.1 *Seven* negative *lemmas*

We proceed by proving seven *negative* lemmas, stating that each of the seven models shown above is not *Sudoku*. The proof to each lemma consists of two pictures: the left picture represents a solution to the Sudoku puzzle where the circled cells have the specified value of 4 or 5 (note that there might be many solutions that satisfy this). For example, in the first lemma, the left picture represents any solution where cell x_{11} has value 4 and cell x_{13} has value 5. The right picture in any proof represents the result of changing every circled 4 in the left picture by a circled 5, and *vice versa*. In all cases the result is a non-solution (to *Sudoku*) with the violated big constraints depicted as shaded. These violated constraints are exactly those that, if removed, the lemma claims cannot yield *Sudoku*. Since the picture proves that if the big constraints in question are removed then the non-solution is accepted as a solution, lemma is proved.

Lemma 5.1


is not *Sudoku*.

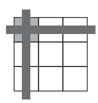
Proof



□

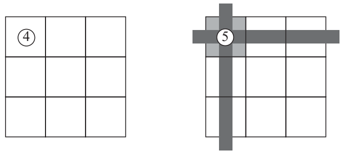
The other six negative lemmas follow the same schema. We expect readers to work out the details for them after convincing themselves that such an initial solution exists for each proof (some such solutions are provided in Appendix C).

Lemma 5.2


is not *Sudoku*.

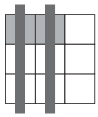
https://doi.org/10.1017/S1471068412000361 Published online by Cambridge University Press

Proof



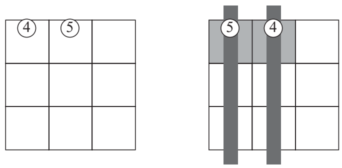
□

Lemma 5.3



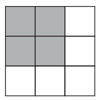
is not *Sudoku*.

Proof



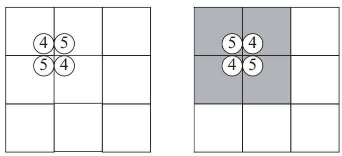
□

Lemma 5.4



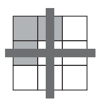
is not *Sudoku*.

Proof



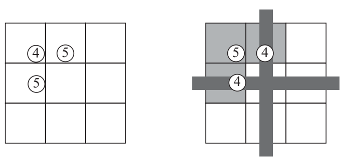
□

Lemma 5.5



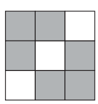
is not *Sudoku*.

Proof



□

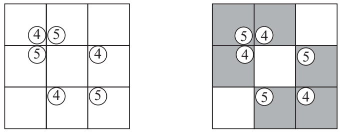
Lemma 5.6



is not *Sudoku*.

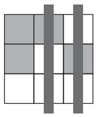
https://doi.org/10.1017/S1471068412000361 Published online by Cambridge University Press

Proof



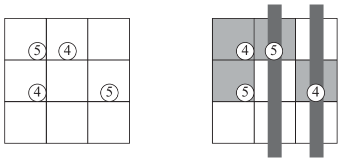
□

Lemma 5.7



is not *Sudoku*.

Proof



□

5.2 Making use of negative lemmas

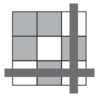
The above positive and negative lemmas give us a complete method for determining whether any model in *Missing*(6) is *Sudoku* or not: If the application of the constructive lemmas results in *Sudoku*, then the model is *Sudoku*, otherwise it will get stuck in one of the seven negative models, and thus is known not to be *Sudoku*. In this sense, the two constructive lemmas are complete (and also confluent). This can be used to render our first program more useful by changing the `classify_each/3` predicate to also check whether the models that do not have all 27 big constraints are one of the seven negative lemmas. If so, it ignores them, otherwise, as before, it adds them to `Stuck`. Note that, for the case of *Missing*(6), `Stuck` is then empty. We refer to the modified version of Program I by Program II, and we have further modified it to generate the pictures[3] that can be found in Appendices A and B: We run this modified program with $n = 6$, and for each model in (the reduced) *Missing*(6) a picture is output. Interestingly, there are 39 different models in (the symmetry reduced) *Missing*(6) that are *Sudoku*, and 70 that are not.

5.3 No model in Missing(7) *is* **Sudoku**

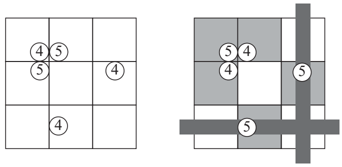
When we run Program II with $n = 7$, every model gets stuck either in one of the previous seven lemmas, or in a model with a new set of big constraints. This model results in one more negative lemma, which is not implied by any of the previous negative lemmas.

[3] See file `genfigs.pl` at the website.

Lemma 5.8

 is not *Sudoku*.

Proof



□

Readers can easily check that none of the models in Lemma 5.1 up to Lemma 5.7 is contained in the above model. As a result, no model in *Missing*(7) is *Sudoku*. Or put otherwise, no redundant set of big constraints has more than six elements.

5.4 *Generalizing to puzzles of size N*

Our techniques can be readily applied to the investigation of Sudoku puzzles of different sizes. Up to now, we have dealt with puzzles of size 3, i.e., there are 3^4 cells, in a 3^2 by 3^2 board, with 3^2 rows, columns, and boxes. Clearly, Lemmas 4.1 and 4.2 easily generalize to other sizes. For example, for size 4, one just needs to add one non-shaded block constraint to the pictures to ensure that the lemmas remain true.

This suggests that for size n, no model in *Missing*$(2\times n + 1)$ is *Sudoku*. Proving this, however, is outside the scope of the current paper.

6 Redundancy for small constraints

For each of the models in *Missing*(6) one can easily count the number of different small constraints it represents: for the ones that are *Sudoku*, the highest count is 690, and the lowest count is 648. This lowest count occurs only for the set of Theorem 4.5, and we denote the model with this set of small constraints by $Small_{4.5}$.

It seems worth trying to remove small constraints from $Small_{4.5}$ and check whether the resulting model is still *Sudoku*. To achieve this, we have implemented a Prolog program[4] that selects every small constraint $x \neq y$ in $Small_{4.5}$, creates a new set $Rest = Small_{4.5} \setminus \{x \neq y\}$, and then tries to prove *Rest* is not *Sudoku* by posting all constraints in *Rest* plus constraint $x = y$ to a constraint solver and running the solver on a set of Sudoku puzzles. If a solution is found, then *Rest* cannot be *Sudoku*, since x cannot be equal to y in it. Note that this is similar to our manual treatment of the set of models classified as stuck by Program I, where each model is proved not to be *Sudoku* by finding a solution to the model that is not a solution of *Sudoku*. The (simplified) Prolog program is provided in Figure 4. The set of Sudoku puzzles that we have used comes from Royle's (2006) website and consists of more than

[4] See file `sudoku648.pl` at the website.

https://doi.org/10.1017/S1471068412000361 Published online by Cambridge University Press

```
try_each_inequality(Model):-
    remove(X#\=Y,Model,Rest),
    (gordonRoyle(Givens), solve([X#=Y|Rest],Givens) ->
        writeln(is_not_Sudoku(Rest))
    ;
        writeln(maybe_Sudoku(Rest))
    ).
```

Fig. 4. Program III.

50,000 *minimal* Sudoku puzzles, each containing 17 given entries: their minimality was proven recently in McGuire *et al.* (2012). We refer to this set as *GR*.

Interestingly, the above program determines that every strict subset *Rest* of $Small_{4.5}$ is not *Sudoku*: for each *Rest*, there is indeed a puzzle in *GR* which has a solution that makes the two variables in the removed inequality equal. This proves that the set $Small_{4.5}$ forms a locally minimal set of small constraints for *Sudoku*. This was independently verified by Michael Codish (Private communication, 2012) by running a CNF-encoding of that statement using the BEE-compiler described in Metodi and Codish (2012). Moreover, using the same technology, we were jointly able to prove that each of the 39 models *M* of *Missing*(6) that are *Sudoku* (see Appendix A) has the following property:

> *M* has a subset of inequalities of size 648 that is *Sudoku* and is also a locally minimal set of small constraints.

We were not able to reduce those *M*s any further, i.e., beyond 648. Although these results do not allow us to conclude that Sudoku models with a smaller set of small constraints are not possible, we dare to conjecture the following:

Conjecture. No model with less than 648 small constraints is *Sudoku*.

7 Discussion and conclusion

The message in rec.puzzles mentioned in the Introduction also refers essentially to our Corollary 4.3, i.e., that in every chute, one row (or column) constraint needs no checking if the other constraints in that chute are validated.[5] Clearly, other people have wondered about redundant big constraints in Sudoku, and our main result – many sets of six big constraints are redundant – often surprises people. It is all the more interesting that the popular (Ist *et al.* 2006) refers to the "minimal encoding" as one containing *all* big rules: our results clearly indicate that such encoding is not minimal at all. Further, while redundant rules can strengthen propagation and, thus, reduce the search space, it has already been noted (Kwon and Jain 2006) that the classical Conjunctive Normal Form encodings for Sudoku in SAT generate too many redundant clauses, and compact encodings (which eliminate redundant clauses) are more efficient. Our work can be used to inform such encodings.

[5] At the time of that post, we had already completed our classification of big constraints.

Our conjecture that no model with less than 648 small constraints is *Sudoku* remains to be proven. While the combinatorial challenge is great, we are currently investigating the use of *unavoidable sets* as in McGuire *et al.* (2012). We have also obtained a full classification of models that use small constraints for the more restricted problem of Latin Squares (Demoen and Garcia de la Banda 2012).

Apart from our novel results themselves, and the use of exploratory (Constraint) Logic Programming, this paper also introduces a powerful graphical representation of sets of constraints that renders the proofs easy to understand, and can be reused for larger Sudoku puzzles.

Exploratory programming was essential in this research: it helped us discover potential theorems and lemmas which we subsequently turned into hard general proofs. Further, the use of Prolog has been critical: as can be seen from the website, the programs are small, fast, and easy to read and modify. This would have been very difficult without the combined power of backtracking (for almost everything, particularly finding all solutions satisfying a set of conditions), constraint solving (to easily define Sudoku and test the satisfiability of many of its subsets), and logic variables (to easily identify and access the variables in the model).

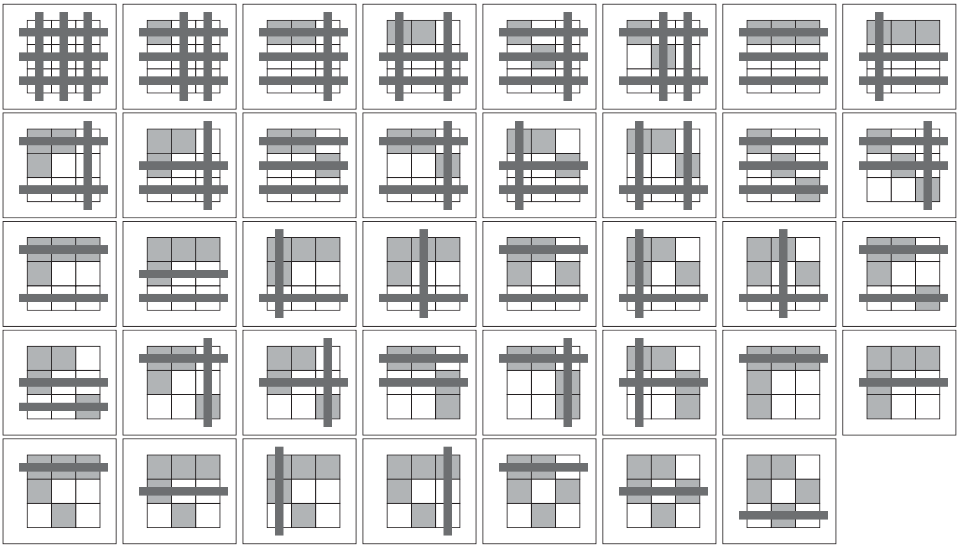
Redundant constraints are very often good for the performance of constraint solving systems, and indeed all solvers that we checked perform much slower (about a factor 2000) with a minimal set of big constraints. So it might seem counterproductive to try to find redundant constraints if the aim is to remove them. However, our work gives some insight into the *construction* of new (redundant) inequality constraints: while deriving new equalities from a set of equalities is easy because equality is transitive, this does not hold for inequalities. The difficulty and possibility of deriving new inequalities depends crucially on the domains of variables. For instance, from a chain of inequalities $x_1 \neq x_2 \neq \ldots \neq x_n$ between *boolean* variables, one may conclude that $x_1 \neq x_4$ (among others), but if the domains have a larger cardinality, this is no longer true. Since our work provides a complete set of rewrite rules on sets of *all_different* constraints (together with the domain constraints) for a particular CSP, it forms the first step in the development of a more general inequality inference framework.

Finally, note that our result on big constraints completes in some sense the result in McGuire *et al.* (2012): 17 clues are necessary, and so are 21 big constraints. It would be interesting to have the corresponding result for small constraints.

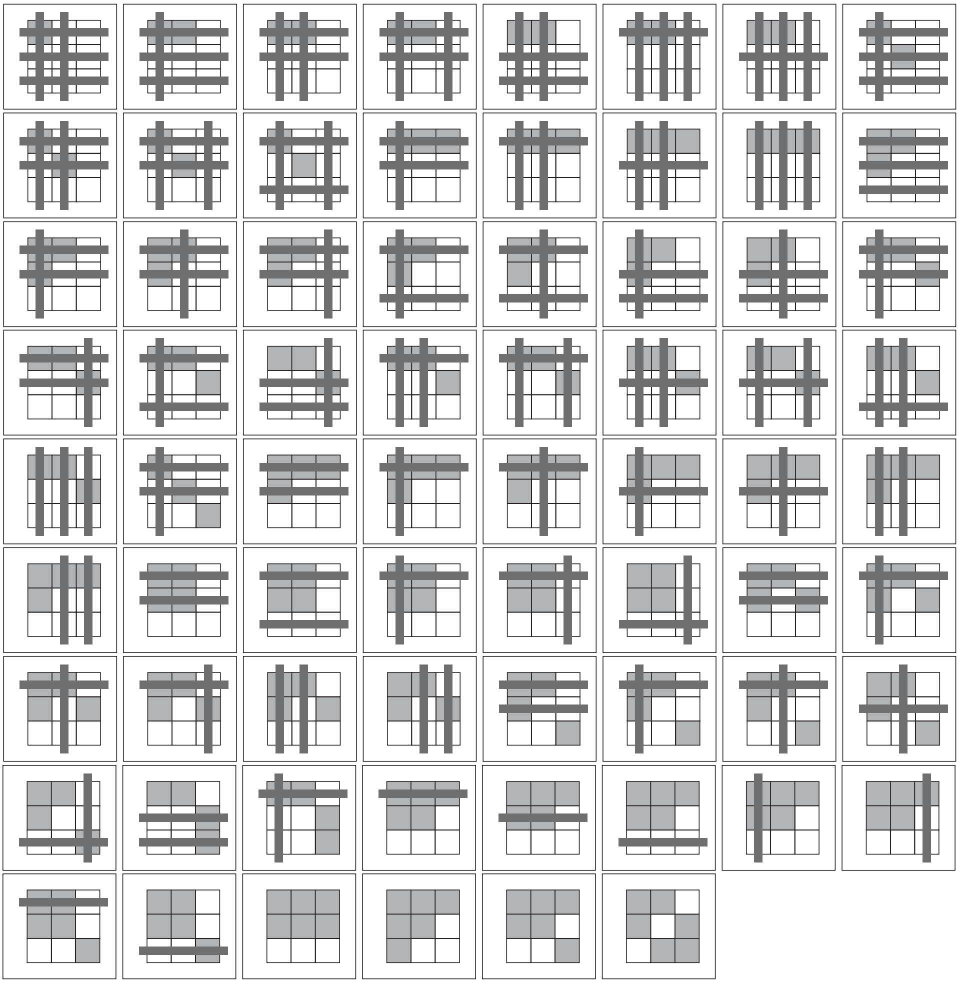
Acknowledgements

The main results reported here were obtained while the first author was on a research visit at Monash University in April 2008, and enjoying the Stuckey hospitality in Apollo Bay and Elwood, Australia. Many thanks for the most enjoyable stay. We are grateful to Michael Codish for his help for obtaining some of the results related to the conjecture. This research was partly sponsored by the Australian Research Council grant DP110102258, by the Brussels-Capital Region through project ParAps, and by the Research Foundation Flanders (FWO) through projects *WOG: Declarative Methods in Computer Science* and *G.0221.07*.

Appendix A: All *Sudoku* models in *Missing*(6) up to symmetry



Appendix B: All *non-Sudoku* models in *Missing*(6) up to symmetry



https://doi.org/10.1017/S1471068412000361 Published online by Cambridge University Press

Appendix C: Initial solved puzzles for Lemma 5.1 up to Lemma 5.8

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

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

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

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

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

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

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

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

References

DEMOEN, B. AND GARCIA DE LA BANDA, M. 2012. *Maximal Sets of Redundant Constraints in Latin Square*. Monash University, no. 2012/269. Technical Report.

IST, I. L., LYNCE, I. AND OUAKNINE, J. 2006. Sudoku as a SAT problem. *Proceedings of the 9th International Symposium on Artificial Intelligence and Mathematics (AIMATH 2006)*, Fort Lauderdale, FL, USA. Springer, New York, USA.

JUSSIEN, N. 2007. *A to Z of SUDOKU*. ISTE, London.

KWON, G. AND JAIN, H. 2006. Optimized CNF encoding for Sudoku puzzles. In *Short paper presentation at the 13th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR 2006)*.

MCGUIRE, G., TUGEMANN, B. AND CIVARIO, G. 2012. There is no 16-clue Sudoku: Solving the Sudoku minimum number of clues problem. *CoRR abs/1201.0749*.

METODI, A. AND CODISH, M. 2012. Compiling finite domain constraints to SAT with BEE. *Theory and Practice of Logic Programming 12*, 4–5, 445–464.

ROYLE, G. Minimum Sudoku. Accessed September 2012. URL: http://school.maths.uwa.edu.au/~gordon/sudokumin.php.

WIKIPEDIA. n.d. Sudoku. Accessed September 2012. URL: http://en.wikipedia.org/wiki/Sudoku.

https://doi.org/10.1017/S1471068412000361 Published online by Cambridge University Press