Mathematics of NRC-Sudoku

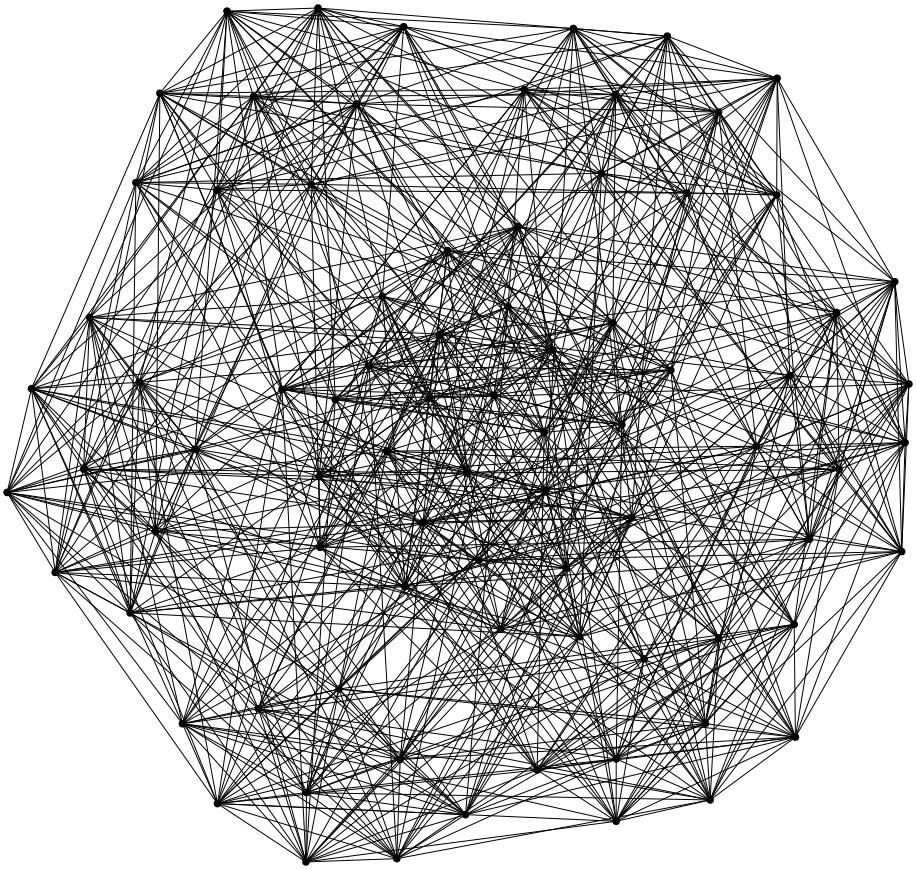
Michel, B.

Publication date
2007

Mathematics of NRC-Sudoku

Bastian Michel

December 5, 2007



Abstract

In this article we give an overview of mathematical techniques used to count the number of validly completed 9×9 sudokus and the number of essentially different such, with respect to some symmetries. We answer the same questions for NRC-sudokus. Our main result is that there are 68239994 essentially different NRC-sudokus, a result that was unknown up to this day.

In dit artikel geven we een overzicht van wiskundige technieken om het aantal geldig ingevulde 9×9 sudokus en het aantal van essentieel verschillende zulke sudokus, onder een klasse van symmetrieën, te tellen. Wij geven antwoorden voor dezelfde vragen met betrekking tot NRC-sudoku's. Ons hoofdresultaat is dat er 68239994 essentieel verschillende NRC-sudoku's zijn, een resultaat dat tot op heden onbekend was.

Dit artikel is ontstaan als Kleine Scriptie in het kader van de studie Wiskunde en Statistiek aan de Universiteit Utrecht. De begeleidende docent was dr. W. van der Kallen.

Contents

1 Introduction

Sudoku, originally invented by Howard Garns in 1979 and published in Dell Magazine as 'Number Place', has been popular in Japan from 1986 on. Only in 2005 it became an international success under its current name 'Sudoku', meaning something like 'single number'.
Sudoku is a puzzle played on a grid of 9×9 cells, divided in nine 3×3 boxes. At the initial state of the puzzle some cells are filled with one of the digits $1, \ldots, 9$. Its objective is to fill the remaining cells of the grid with the digits such that the following simple rule, also called 'the one rule', is met: each digit must occur exactly once in each of the rows, columns and boxes. The solution to a given sudoku puzzle is always required to be unique.

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

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

The sudokus published by the Dutch newspaper NRC Handelsblad differ from normal sudokus only in satisfying one more rule: each of the digits $1, \ldots, 9$ must occur exactly once in each of the four grey underlaid boxes.

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

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

For lack of a better name, these special sudoku puzzles are commonly refered to as 'NRC-sudokus'.

1.1 Mathematics of sudoku

Currently there is no theory that would enable a concise, profound and complete analysis of the mathematically interesting structures underlying sudoku, nor is it likely that such a theory could be developed. However, there are a few theories that relate to sudoku in some way and many others that provide different useful techniques. Amongst the former, the theory of magic squares, dating back to the middle ages, and the theory of graph colourings should be mentioned. The most important of the latter is, without doubt, combinatorics. In fact, sudoku being popular

quite a short period only, the mathematicians interested in sudoku are still assembling different techniques from various fields for its analysis.
The mathematically interesting challenges of sudoku largely fall apart in two classes: analysing the properties of completed sudoku grids and analysing the features of initial states of sudoku puzzles.

- How many different sudokus are there?
- Which transformations of a completed grid preserve its validity?
- How many essentially different sudokus, i.e. not being equivalent through those transformations, are there?
- How many clues are required in a puzzle to render its solution unique?
- What makes a puzzle harder or easier to solve?
- What is exactly the importance of place and value of the clues in that question?
- Which solving techniques are the best, for man and machine?
- Can our techniques be generalised to arbitrary rectangular grids with arbitrary divisions in boxes?

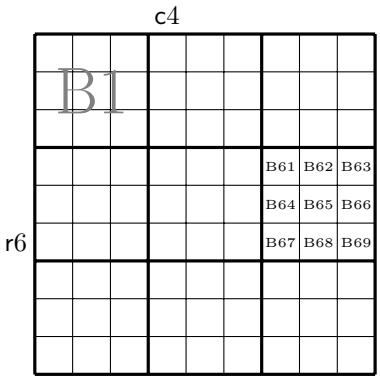
1.2 Aim of this paper

In this article we will concentrate on the first three of these questions, dealing with properties of completed sudoku grids.
Our main aim is to answer these questions for the case of NRC-sudokus. The total number of NRC-sudokus is known by a result ascribed to Andries E. Brouwer, however there is no documentation of his approach to be found anywhere. We confirm his result in section 3.3. The transformations that preserve validity of an NRC-sudoku are fairly easily recognisable and probably well-know within the community interested in this complex. However, this does by no means weaken the necessity of a thorough analysis, which we will give in section 3.2. The number of essentially different NRC-sudokus is, to our best knowledge, unknown up to this day. We will give it in section 3.4.
A secondary aim of this article is to provide the reader with a range of techniques that are used to analyse sudoku. To this end, we give an overview of the results for ordinary sudoku corresponding to our three questions in the second part of the paper. Also, this part serves as a lead-up to our investigation of the NRC-case. Most techniques for ordinary sudoku cannot be used for NRC-sudokus and it will be interesting to see why. The only technique, in fact, that can be used for the NRC-case is the one that Russel and Jarvis apply to ordinary sudokus in their article [2], in order to find the number of essentially different sudokus. It is thanks to this technique that we are able to give our main result in 3.4.

1.3 Terminology

To refer to rows and columns we enumerate them top-to-bottom and left-to-right and write $\mathsf{r1}$, $\mathsf{r2}$, ... and $\mathsf{c1}$, $\mathsf{c2}$, ... We will be careful not to confuse single cells with boxes or 'blocks' consisting of 3×3 cells. The boxes will be enumerated as $\mathsf{B1}$, $\mathsf{B2}$, ... top-to-bottom and left-to-right. Rows, columns, boxes and NRC-boxes are the four sorts of 'regions' of a grid. The cells will be referred

to by their block number and, within the block, by enumerating the cells top-to-bottom and left-to-right, such that, for instance, the cell in the very middle will be called B55. The initial state of a sudoku, as published in newspapers, is called a puzzle. Often, we will refer to a correctly completed grid simply by calling it a sudoku.



1.4 Sudoku as a graph colouring problem

For some detail of our analysis graph theory provides useful techniques. The 9×9 sudoku grid is represented as the following graph: two of 81 points, intended to represent the cells, are joined by an edge whenever the two respective cells occur in the same row, the same column or the same block (or, in the case of NRC-sudokus, in the same NRC-block), i.e. sharing at least one region. We discard the possibility of double edges: two points are either linked by one edge or not linked. Also, we stress that no point is linked to itself. So, e.g. the point representing cell B61 is linked to the eight points representing the other cells in its row r4, the eight points representing the other cells in its column c7 and the four points representing the remaining four cells in its block B6. The situation is similar for all other points: each point is joined to exactly 20 others. In fact, all points in the graph are similar: there is none that has any special graph theoretic property, or, put differently, looking at the graph from one point gives the same view as looking from any other point. This is not true for the graph representing the NRC-sudoku grid. We will discuss this later on.
The graph representing the ordinary sudoku grid is shown on the title page.
The problem of solving a sudoku can now be thought of as a colouring problem on the graph: give each point one out of nine colours in such a way that the following rule, the reflection of the one rule, is fulfilled: two points that are joined by an edge must never have the same colour.
As said before, this representation of sudoku will provide good grounds for some analysis. It should, however, be quite clear that for the normal problem of solving a given sudoku puzzle, this representation would be a very bad choice: no human being could possibly solve a puzzle on its graph as effectively as it could solve it on its grid.

1.5 Computerised solving by backtracking

Before concentrating on the main questions of this article, we should briefly describe the principle of a backtracking algorithm that can be used to solve sudokus. This algorithm is incredibly stupid, but, put in practice, works astonishingly fast. A puzzle should be fed to the program. The algorithm searches the first free cell and fills it with the least digit that does not immediately violate the one rule at this place. Then it continues to the next free cell. When at any state it is impossible to fill in any of the nine digits, i.e. having reached a contradiction, the algorithm

goes back to the cell before and tries a greater digit here. Put abstractly, the algorithm proceeds to fill in anything as far as possible and, having reached a contradiction, tracks back as little as possible. A backtracking algorithm that solves NRC-sudokus is given in appendix A.
Solving a sudoku with a computer program based on a backtracking algorithm is a matter of fractions of a second. Such a program is easily adapted to count completed sudokus rather than solving puzzles.

2 Ordinary sudoku

2.1 Symmetries

The easiest way to make a new sudoku puzzle is probably to take an old one, which is known to lead to a unique solution, and change it in some trivial ways. We can, for instance, turn it a quarter turn clockwise and exchange all 1s with the 2s. There are several of those transformations that preserve validity of a completed sudoku. We will call them symmetries.

1)	All permutations of the nine digits.	$9!$
2)	All permutations of the three bands.	$3!$
3)	All permutations of the three stacks.	$3!$
4)	All permutations of the three rows in one of the three bands.	$3!^3$
5)	All permutations of the three columns in one of the three stacks.	$3!^3$
6)	Transposition	2

Note that some other obvious symmetries are nothing but compositions of symmetries mentioned here. For instance, reflection in the horizontal axis is the same as permutation of the upper and lower band followed the row permutations (r1 r3)(r4 r6)(r7 r9). The analogous transformation for stacks and columns coincides with reflection through the vertical axis. Rotation by a quarter turn clockwise coincides with transposition followed by the permutations that constitute a reflection in the vertical axis, anticlockwise rotation by a quarter turn is nothing but transposition followed by the permutations that constitute a reflection in the horizontal axis, a half turn is a vertical and a horizontal reflection composed.
Note also that we did not include any transformation that would be valid only on some sudokus and on others not. Consider for instance a sudoku containing the following:

	2				6			
	6				2			

In such a sudoku we could permute the entries 2 and 6 on these places and keep them fixed on other places. This manipulation, however, is not valid on most sudokus. So, with the purpose of taking in account only those transformations being valid on all completed sudokus, we claim to have given them all in our list above.

For the transformations 2) – 6) it is also clear that they are all independent of each other and therefore, they form a group of order

$$3! \times 3! \times 3!^3 \times 3!^3 \times 2 = 3!^8 \times 2 = 3359232,$$

in other words: each sudoku is related to 3359231 other ones through these transformations.
However, including the permutation of digits, 1), in our considerations, changes the situation dramatically. There are lots of completed sudokus for which a relabelling coincides with a composition of transformations 2) – 6). Consider, for instance, the following completed sudoku, provided by Russel and Jarvis [2], for which the permutation of digits (1 3 9 7)(2 6 8 4) has the same result as a quarter turn anticlockwise:

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

This non-independentness will become important when we want to count the number of essentially different sudokus.
All but the first symmetry are in fact symmetries of the grid, and will therefore be called 'grid symmetries'.

2.2 How many different sudokus are there?

Our mathematical apparatus provides some techniques to give upper bounds to the number of sudokus. However, the exact answer, until now, is essentially found by computerised brute-force counting, in which mathematical techniques are only used to achieve a considerable reduction of the cases to be counted.
By basic combinatorics, there are 9! ways to insert the nine digits into one row, one column or one box. Therefore there are $9!^9 \approx 1.0911 \times 10^{50}$ completed grids that fulfil the one rule only with respect to one of rows, columns or boxes.
Sudokus that fulfil the one rule with respect to rows and columns, but not necessarily with respect to boxes, coincide with 9×9 magic squares. The theory of magic squares does not provide any general formula to calculate the number of such squares. The number of 9×9 magic squares was calculated by Bammel and Rothstein in 1973: $5524751496156892842531225600 \approx 5.525 \times 10^{27}$. The counting techniques they used are essentially the same as those used for sudokus today.

2.3 Ad hoc counting by Felgenhauer and Jarvis

Apparently, the first to find the number of sudokus were Felgenhauer and Jarvis, see [1]. Their brute-force counting is implemented as a backtracking algorithm. First, they count the number of consistent fillings of the first band. Then, using the symmetries, they deduce that these fillings can be split into just 44 classes such that two members of one class have the same number of

completions. This latter number they count by means of a brute-force algorithm for each of the 44 cases. Each result has to be multiplied by the respective number of class members. Adding these products gives the result $6670903752021072936960 \approx 6.671 \times 10^{21}$.

2.4 Counting by band generators

Even before official publication of Felgenhauer and Jarvis's approach, a different and more effective method seems to have appeared on different forums, referred to as 'the method of band generators'. The following is largely based on the current content of [5].

The underlying idea of this approach is to initially ignore some of the requirements for a valid sudoku. We begin by considering one band. For each column in this band, also called a mini-column, we choose an unordered set of three entries such that the one rule with respect to the boxes is fulfilled, i.e. we choose $A_1, A_2, \ldots, A_9 \subset \{1, \ldots, 9\}$ with $\#A_i = 3$ such that $A_1 \cup A_2 \cup A_3 = A_4 \cup A_5 \cup A_6 = A_7 \cup A_8 \cup A_9 = \{1, \ldots, 9\}$. Such a sequence of sets $(A_1, A_2, \ldots, A_9)$ we call a band generator g. Note that there are $\binom{9}{3} = 84$ choices for A_1 and another $\binom{6}{3} = 20$ for A_2, which together determine A_3. So we have $84 \times 20 = 1640$ choices for A_1, A_2, A_3. The situation for the other A_i is alike, giving us $1640^3 = 4741632000$ band generators.

We say that a band generator $g = (A_1, \ldots, A_9)$ generates a band if each entry of the ith mini-column of the band is an element of A_i, i.e. a band is obtained from a band generator by fixing places in a minicolumn for all elements of the respective set A_i. We stress that any band generated by a band generator fulfils the one rule with respect to boxes, also each of them weakly fulfils the one rule with respect to the columns, i.e. mini-columns contain no repeated value, as those all appear in the same box. But only quite few respect the row rule. Let $B(g)$ be the number of valid sudoku bands that g generates, i.e. bands that also fulfil the one rule with respect to rows. Typically B has values around 200.

To fill all three bands of a grid and take into account the one rule with respect to columns, we introduce a ternary relation on band generators: we call three band generators g_1, g_2, g_3 *compatible* if, when placed in a grid, each column contains each digit exactly once, and we will write $C(g_1, g_2, g_3)$. More formally: $C((A_1, \ldots, A_9), (B_1, \ldots, B_9), (C_1, \ldots, C_9))$ holds if $A_i \cup B_i \cup C_i = \{1, \ldots, 9\}$ for each $i \in \{1, \ldots, 9\}$. It is this very notion that is essential to our approach: compatibility can be defined, as we did, on band generators, as the position of a digit in one mini-column does not matter for the column rule to hold or not. And, tragically, it is this very point in the approach that does not work out for NRC sudokus: we cannot define such a compatibility notion in order to force the one rule to hold with respect to the grey NRC-boxes while keeping that notion on the level of generators.

Now, the number of valid sudokus is

$$\sum_{g_1} \left(B(g_1) \times \sum_{(g_2, g_3) \mid C(g_1, g_2, g_3)} B(g_2) \times B(g_3) \right)$$

This neat formula would be quite useless for an efficient computation if we really had to loop over all 4741632000 band generators. But fortunately, we can use symmetries to reduce the number of band generators g_1 has to loop over. We take into account the following symmetries:

- relabelling the nine digits,
- permuting the three boxes,
- permuting the three mini-columns within one of the three boxes.

It turns out that under these symmetries the 4741632000 band generators are partitioned into just 44 classes. Let $K(g) = 1, \ldots, 44$ be the number of the class g falls in, under some fixed enumeration of the classes. As we used validity preserving symmetries only, we know that two generators of the same class generate the same number of valid bands, i.e. B is constant on each class and can therefore be lifted, to be understood as a function on the class numbers $\{1, \ldots, 44\}$ from now on. Let $\#(i)$ be the number of class members of the ith class and $G(i)$ any of its representatives. Then the number of valid sudokus is

$$\sum_{i=1}^{44} \left(\#(i) \times B(i) \times \sum_{(g_2,g_3) \mid C(G(i),g_2,g_3)} B(K(g_2)) \times B(K(g_3)) \right).$$

One can now leave the calculation to a computer program. The program, for each $i = 1, \ldots, 44$, will have to find all g_2, g_3 that are compatible, i.e. fulfilling $C(G(i), g_2, g_3)$. The computation of the two times 44 values of $\#$ and B will be easy. The computation of G and K should be made efficient by choosing neat representatives for each class and a neat enumeration of the classes. Efficient implementations are reported to compute the result within fractions of a second. The number of valid sudokus calculated by Felgenhauer and Jarvis has been confirmed by this approach several times.

2.5 Essentially different sudokus

In their article [2] Russel and Jarvis provide an approach to calculate the number of essentially different sudokus that we will later use for NRC-sudokus. It is based on using Burnside's lemma, stating that

> If G is a group acting on a finite set A then the number of orbits $\#A/G$ is equal to the average number of fixed points of A under the elements $g \in G$, i.e.
>
> $$\#A/G = \frac{1}{\#G} \times \sum_{g \in G} \#A^g,$$
>
> where for each $g \in G$, A^g denotes the fixed points of A under g.

The number of orbits calculated thus, will be the number of essentially different sudokus. They take the $3!^8 \times 2$-elements group of grid symmetries as G. In order to take into account the 9! relabellings, they define an equivalence relation on all sudokus, two of them being equivalent if they are equal through some relabelling. So A is the set of equivalence classes of this relation. Now, for each $g \in G$, they must count sudokus fixed under g up to relabelling, i.e. equivalence classes fixed under g. They denote the symmetries $g \in G$ as permutations of the 81 cells, although they could have denoted them as permutations of the 18 rows and columns, as we will do later. Now they point out that it is not necessary to count the number of fixed classes for each of the $3!^8 \times 2 = 3359232$ elements of G. Each two members of a conjugacy class of G will give the same number: suppose $g, g' \in G$ and $g = h^{-1}g'h$ for some $h \in G$ and suppose g fixes exactly n sudokus $X_1, \ldots, X_n$ up to relabelling, i.e. $X_i = p_i g X_i$ for all $1 \leq i \leq n$, where the $p_i \in S_9$ are relabellings. Then $X_i = p_i h^{-1} g' h X_i$, hence $h p_i X_i = g' h X_i$, and as all relabellings commute with all grid symmetries, we have $p_i h X_i = g' h X_i$, which means that g' fixes hX_i up to relabelling. As all those hX_i are distinct, we know that there are at least n sudokus, namely $hX_1, \ldots, hX_n$, that are fixed by g' up to relabelling, which means that $\#A^g \leq \#A^{g'}$. By symmetry also $\#A^{g'} \leq \#A^g$, so $\#A^g = \#A^{g'}$ whenever g and g' are conjugate, as we wanted to prove.

So, instead of counting the number $\#A^g$ of sudokus fixed by a symmetry g up to relabelling for each symmetry $g \in G$, they count $\#A^g$ just for one representative per conjugacy class and multiply by the size of the respective class. Let $\mathcal{C}$ be the set of conjugacy classes and k some fixed function $\mathcal{C} \to G$ that chooses a representative from each class. Then the formula above simplifies to

$$\#A/G = \frac{1}{\#G} \times \sum_{C \in \mathcal{C}} \#A^{k(C)} \times \#C.$$

Most symmetries, however, do not fix any sudoku up to relabelling, so the factor $A^{k(C)}$ will be zero in most cases. The non-zero factors are calculated by a computer program. Russel and Jarvis have documented the respective results on their website. The final result is that there are

$$5\,472\,730\,538$$

or

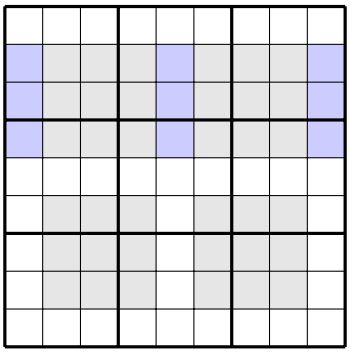
> five billion four hundred seventy-two million seven hundred thirty thousand five hundred thirty-eight

essentially different sudokus.
We will use the same techniques to calculate the number of essentially different NRC-sudokus.

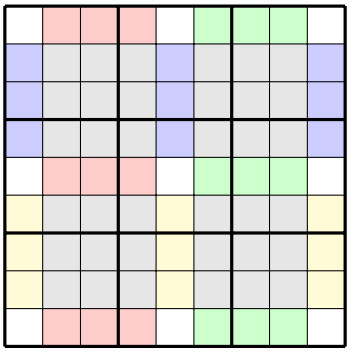
3 NRC-sudokus

3.1 An initial observation concerning NRC-sudokus

As remarked in many forums on NRC-sudokus, the four new boxes induce five more boxes to which the rule applies. As those are disconnected sets of cells, they might not be immediately seen. But consider the rows r2, r3, r4. In this scope each digit occurs exactly three times, two times of which are consumed by the two upper grey boxes. This leaves exactly one occurrence of each digit for the other nine cells, highlighted blue in the picture below.



Obviously, the same reasoning also applies to the rows r6, r7, r8 and to the columns c2, c3, c4 and c6, c7, c8, leading to three more disconnected boxes:



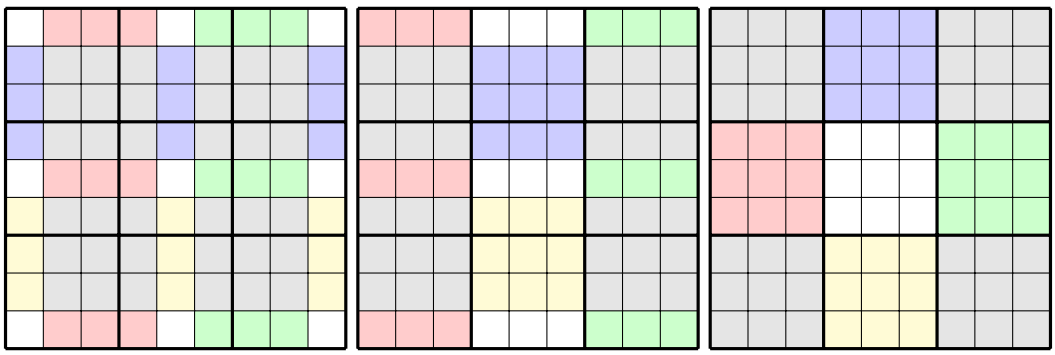
Now this picture suggests our final observation: in the entire sudoku each digit occurs exactly nine times, eight times of which are consumed by the four grey and four coloured boxes. This leaves exactly one occurrence of each digit for the other nine cells, the ones that are not yet highlighted. Therefore, we will consider these cells to form a new box too.
Having revealed these hidden structures induced by the four grey boxes, we end up with a fourth complete covering of our sudoku grid: not only the rows, the columns and the boxes form an exact cover of the grid, also the four grey plus five new boxes do. We will call them NRC-boxes.

3.2 Valid transformations of NRC-sudokus

A brief check on our list of symmetries of ordinary sudokus shows that most of them do not preserve the validity with respect to the NRC-boxes and are therefore not symmetries of NRC-sudokus. The remaining ones are:

1) All permutations of the nine digits. $9!$
2) The row permutations (r2r3) and (r7r8). $2 \times 2 = 4$
3) The column permutations (c2c3) and (c7c8). $2 \times 2 = 4$
4) Reflections in the horizontal and vertical axes, transposition and rotation. $2 \times 2 \times 2 = 8$

Now, the role of transposition can be a hint to discovering another symmetry. Transposition of a sudoku grid exchanges the rows and columns and leaves the boxes as such. Exchanging rows and boxes while leaving the columns as such is impossible, as is exchanging columns and boxes while leaving the rows as such. But now we have a fourth complete covering of the grid, and in fact it turns out that it is possible to exchange boxes and NRC-boxes while leaving the rows and columns as what they are. From this feature it is obvious that this transformation will preserve validity of the sudoku. The transformation is a composition of permutations that taken separately would not preserve validity: permute (r1r4)(r6r9)(c1c4)(c6c9) as in the picture below:



Observe that not only do the NRC-boxes shift in the place of the normal boxes, also do the normal boxes shift in the place of the NRC ones. For lack of better, we will call this composition of permutations

5) The NRC-transformation. 2

In fact, we can assure ourselves that there are no more grid symmetries than the above mentioned. We employ the following definition: a permutation of the 81 cells of the NRC-sudoku grid is called a symmetry if it sends regions to regions. As under such a symmetry the structure of the grid is unchanged, the uncoloured graph is preserved and any correctly completed grid will remain such. In particular, all true statements about the uncoloured graph will remain so. Now, this intensionally broad concept will in extension narrow to just the symmetries mentioned above.
A symmetry preserves incidence of two regions: let S be any symmetry and R_1, R_2 two regions. If some cell a is a member of the overlap $R_1 \cap R_2$, then its image $S(a)$ will be member of the overlap of regions $S[R_1 \cap R_2] = S[R_1] \cap S[R_2]$. On the other hand, for any member b of the overlap $S[R_1] \cap S[R_2]$, we know that $S^{-1}(b)$ is a member of $R_1 \cap R_2$. So

$$R_1 \cap R_2 \neq \emptyset \iff S[R_1] \cap S[R_2] \neq \emptyset.$$

Moreover, as a symmetry is a bijection on the cells and regions are entirely determined by the cells they contain, a symmetry is also a bijection of regions. Therefore, if a region R is disjoint from n other regions $R_1, \ldots, R_n$, then and only then its image $S[R]$ under a symmetry S will be disjoint from exactly the n regions $S[R_1], \ldots, S[R_n]$.
We observe that

- A row is disjoint from
  - 8 rows
  - 0 columns
  - 6 blocks
  - 6 NRC-blocks

  in total 20 regions.
- Similarly, a column is disjoint from 20 regions.
- A block is disjoint from
  - 6 rows
  - 6 columns
  - 8 blocks
  - 0, 3 or 5 NRC-blocks

  a total of 20, 23 or 25 regions.
- Similarly, an NRC-block is disjoint from 20, 23 or 25 regions.

From this it can be seen how few symmetries there are. Suppose that a symmetry S would map a row r to a block. Then this can only be a block that is also disjoint from 20 regions, the only such being the middle block B5. This also means that from now on no other row can possible be mapped to any other block. But what happens to the other 8 rows, 8 regions that are not only disjoint from our row r but also mutually disjoint. We have to find regions for them with this same property. As B5 is not disjoint from any of the NRC-blocks, those cannot be used.

We cannot use a row and a column at the same time, as those are never mutually disjoint. As we can use at most 6 rows (or 6 columns) disjoint from B5, at least two rows (or two columns) would have to be mapped to blocks, which is impossible. So a row can never be mapped to a block under any symmetry.
Similarly we can prove that no row can be mapped to any NRC-block: the white NRC-block is the only that is disjoint from 20 regions, but it intersects with all blocks, so at least two of the other eight rows must be mapped to an NRC-block, which is impossible. The reasoning for columns is the same. Therefore we know the following:

> No symmetry maps any row or column to any block or NRC-block.

Suppose any row r were mapped to a column. Then any of the other 8 rows, being disjoint from r, must be mapped to a region disjoint from the column $S[r]$. According to the above mentioned result, it can only be mapped to a column. So we see that:

> Under a symmetry either all rows remain rows and all columns remain columns, or rows and columns are completely interchanged to columns and rows.

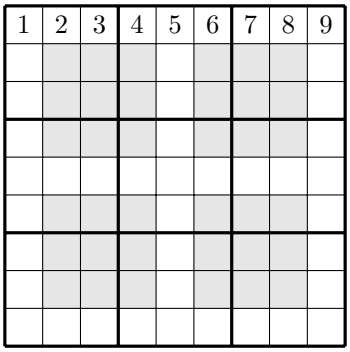
Now, this result has a beautiful consequence. Let us consider any cell a. We can identify a with the pair $(\mathsf{r}x, \mathsf{c}y)$ of its row and column. Now for any symmetry S, one of $S[\mathsf{r}x]$ and $S[\mathsf{c}y]$ will be a row and the other a column. As $S(a)$ will be member of both of them, it must be their only common cell. This shows that if we know the action of a symmetry S on the rows and columns, that we know its action on each of the cells. Therefore, a symmetry is entirely determined by its action on the rows and columns.
This means that in order to find all symmetries, we only have to look for valid permutations of the rows and columns. Let us collect some more facts about our grid.
We noticed in our arguments that the middle block B5 and the white NRC-block have a unique property. Their only common cell is the middle-most, which is uniquely determined by the following property: all four regions in which it falls are disjoint from 20 other regions. As this is an uncoloured-graph-theoretic property, it must be preserved by any symmetry. As it is unique, the middle-most cell must be a fixed point of any symmetry. In particular, r5 and c5 can only be fixed or interchanged.
Also, the sixteen grey cells in the corner blocks have a property distinguishing them from all the others: in the graph of the NRC-sudoku grid each of them is connected to just 23 other cells, as opposed to the 24 cells all others are connected to: 8 through its row, 8 through its column, 8 trough its block of which 4 are already counted, and 8 through its NRC block of which not only 4 (like in the case of all other cells) are already counted but 5. Therefore, rows and columns 2, 3, 7 and 8 can only be permuted with each other and not with the other rows and columns.
It is now easy to check that the only valid permutations of rows and columns that respect all these properties are the ones we found before.

3.3 An ad hoc approach to counting NRC-sudokus

We want to count all possible NRC-sudokus, using as many symmetries as possible to reduce the computation time to an acceptable amount.
Our first step is to just count one of the 9! NRC-sudokus that are equivalent up to relabelling the entries, i.e. up to permutation of the digits. Let S_9 be the set of all these permutations. Let $\mathcal{X}$ be the set of all valid NRC-sudokus. We define an equivalence relation $\sim$ on $\mathcal{X}$ by

$$X \sim Y \quad :\Longleftrightarrow \quad X = p(Y) \text{ for some } p \in S_9$$

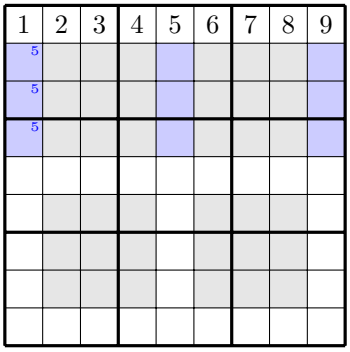
Let $\overline{\mathcal{X}}$ be the set of all equivalence classes. Then $\#\mathcal{X} = 9! \times \#\overline{\mathcal{X}}$. Our preferred representative of each class will be the sudoku whose first row is in standard form. It is this representative that will actually be counted by our brute-force algorithm.



Next, it is sufficient to count only one of each two NRC-sudokus that are equal up to reflection in the vertical axis (r). First, note that the reflection r is none of the relabellings: r keeps all entries in the middle column `c5` unchanged, and those are exactly the nine different digits. On the other hand, all entries outside `c5` are changed. So two sudokus being equal up to the reflection r can never fall into the same $\sim$-equivalence class. Moreover, it is clear that the reflection r and any permutation $p \in S_9$ commute. Therefore r lifts from $\mathcal{X}$ to $\overline{\mathcal{X}}$ in a natural way: suppose $X_1 = r(X_2)$, $Y_1 = r(Y_2)$ and $X_1 \sim Y_1$, say $X_1 = p(Y_1)$; then $X_2 = r(X_1) = r(p(Y_1)) = r(p(r(Y_2))) = r(r(p(Y_2))) = p(Y_2)$, so $X_2 \sim Y_2$. In other words: each $\sim$-equivalence class is linked through r to some other such class and we will need to count only one of those two. Formally, we update our equivalence relation $\sim$ to the following:

$$X \sim Y :\Longleftrightarrow X = p(\xi(Y)) \text{ for some } p \in S_9 \text{ and some } \xi \in \{\mathrm{id}, r\}.$$

For the updated set of equivalence classes $\overline{\mathcal{X}}$ we have $\#\mathcal{X} = 9! \times 2 \times \#\overline{\mathcal{X}}$. The choice of a preferred representative to be counted becomes a little more subtle here. Still, we want to count a representative whose first row `r1` is in standard form. But now there are two of those in each $\sim$-equivalence class, say X_1, X_2, related to each other through the reflection r and the necessary relabelling: $X_1 = (19)(28)(37)(46)(r(X_2))$. We can profit from the fact that the 5s are not relabelled. Consider our blue NRC-block. Here, in any valid NRC-sudoku whose first row is in standard form, the 5 occurs outside the middle column `c5`. In one of X_1, X_2 it occurs left from the middle, in the other right from the middle. We decide to count the one where it occurs left from the middle. So our preferred representative to be counted will be in the following form:



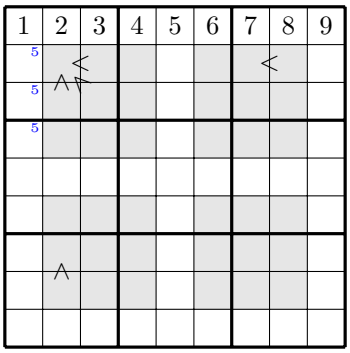
We will achieve the next reduction with a factor 2 by considering the valid transformation (r2 r3), i.e. permuting the second and third row. Similar arguments as above apply and we update our equivalence relation accordingly. Now we do the same for the column permutation (c2 c3), again giving us a factor 2. Our updated equivalence relation now will be

$$X \sim Y \quad :\Longleftrightarrow \quad X = p(\xi(Y)) \ \text{ for some } p \in S_9 \text{ and some } \xi \in \big\langle r, (\mathsf{r2\ r3}), (\mathsf{c2\ c3}) \big\rangle,$$

where $\big\langle r, (\mathsf{r2\ r3}), (\mathsf{c2\ c3}) \big\rangle$ is the 2^3-elements group generated by these symmetries.
For the updated set of equivalence classes $\overline{\mathcal{X}}$ we have $\#\mathcal{X} = 9! \times 8 \times \#\overline{\mathcal{X}}$.
Again, we have to be careful choosing our preferred representative to be counted. Now, each equivalence class contains four different sudokus in the form we preferred until now: say X_1, X_2, X_3 and X_4 with $X_1 = (\mathsf{r2\ r3})X_2 = (23)(\mathsf{c2\ c3})X_3 = (23)(\mathsf{c2\ c3})(\mathsf{r2\ r3})X_4$ We will take the one for which the entry in B15 is less then the entries in B16, B18 and B19. Note that this criterion does not collide with the relabelling (23), as 2 and 3 do not occur a second time in the first block B1.
Now we will use the permutations (c7 c8) and (r7 r8) in a similar way, updating

$$X \sim Y \quad :\Longleftrightarrow \quad X = p(\xi(Y)) \ \text{ for some } p \in S_9 \text{ and some } \xi \in \big\langle r, (\mathsf{r2\ r3}), (\mathsf{c2\ c3}), (\mathsf{c7\ c8}), (\mathsf{r7\ r8}) \big\rangle.$$

For the updated set of equivalence classes $\overline{\mathcal{X}}$ we have $\#\mathcal{X} = 9! \times 32 \times \#\overline{\mathcal{X}}$.
First, concentrating on (c7 c8), we have two candidates X_1 and X_2 with $X_1 = (78)(\mathsf{c7\ c8})X_2$. We will count the one for which B34 is greater than B35, stressing that the relabelling (78) does not interfere with our previous choices. In particular, B15 being less than three other entries of B1, it can neither be 7 nor 8.
Finally, the choice whether to count a sudoku X_1 in this form, or $X_2 = (\mathsf{r7\ r8})X_1$, which is also in this form, can be decided again by taking the one in which B62 is less than B65.
So, summarising, we will count only the sudokus of the form



A backtracking program performing this counting is given in appendix B. On a modern machine it takes approximately thirteen hours of running time. The result of this counting is $\#\overline{\mathcal{X}} = 545736055$. Consequently, there are $\#\mathcal{X} = 9! \times 32 \times 545736055 =$

$$6\,337\,174\,388\,428\,800$$

different NRC-sudokus, in words

> six quadrillion three hundred thirty-seven trillion one hundred seventy-four billion three hundred eighty-eight million four hundred twenty-eight thousand eight hundred.

This confirms the result published on [4] and ascribed to Andries E. Brouwer of the TU Eindhoven.

3.4 Essentially different NRC-sudokus

We will find the number of essentially different sudokus by using the techniques that Russel and Jarvis have used for ordinary sudokus as described in 2.5. We will also follow their approach in using GAP, see [6], for some of the more tedious algebraic work.
Let G be the group of grid symmetries as described above. We will denote the symmetries as permutations of the rows and columns, denoting the rows as $1, \ldots, 9$ and the columns as $11, \ldots, 19$.
The group G is generated by

· reflection in the vertical axis	$(11,19)(12,18)(13,17)(14,16)$
· transposition	$(1,11)(2,12)(3,13)(4,14)(5,15)(6,16)(7,17)(8,18)(9,19)$
· (r2 r3)	$(2,3)$
· the NRC-transformation	$(1,4)(6,9)(11,14)(16,19)$.

It has 256 elements that are split into 40 conjugacy classes.
Let A be the set of all NRC-sudokus, and A^g the set of all NRC-sudokus fixed by $g \in G$ up to relabelling. We will use the formula of Burnside's lemma:

$$\#A/G = \frac{1}{\#G} \times \sum_{C \in \mathcal{C}} \#A^{k(C)} \times \#C,$$

where we will leave the choice of k, i.e. the choice of representatives of each conjugacy class, to GAP. The so chosen representatives can be found in the table below.
Like Russel and Jarvis pointed out, most symmetries $g \in G$ do not fix any sudoku up to relabelling. The same holds in our case of NRC-sudokus. So let us first state some criteria for symmetries that rule out the possibility of fixing any sudoku up to enumeration.
Rather than looking for symmetries $g \in G$ for which there is some relabelling $p \in S_9$ such that there are NRC-sudokus X with $pgX = X$, and therefore $gX = p^{-1}X$, we will directly look for g for which such p^{-1} exists, i.e.: given a symmetry g we will look for relabellings p that coincide with g on some sudokus X, so p for which there are X with $gX = pX$. For each such symmetry g we will have to count how many such NRC-sudokus X there are.
Note that any relabelling p is entirely determined by its actions on the nine digits. Assume we have a symmetry g and an NRC-sudoku X. As in each region of X all nine digits occur, the effect of g on just one region entirely determines what p must be in order to have the desired equality $gX = pX$. We can therefore conclude:

> A symmetry $g \in G$ that fixes some NRC-sudoku X up to relabelling, must behave similarly on each of the regions, more concretely: let $R_1, \ldots, R_{36}$ be the regions of the NRC-grid and for each of them let $g|_{R_i}$ be the permutation of cells of R_i that g induces together with X; then $g|_{R_i}, \ldots, g|_{R_i}$ all must have the same cycle structure – the one that the corresponding p has too.

Also, without looking at the entries of X, which we will not know a priori, we can use the following fact:

> For any such symmetry g that maps a region R to itself, we know that the permutation of digits p it induces through that region is similar to the permutation of cells of R that g induces. So on any couple of regions R_1, R_2 mapped to themselves under g we know that the cell permutations within them must be of equal cycle structure.

Whenever transposition is not involved, we know that in order to induce permutations of the same cycle structure on r5 and c5 the symmetry's cycles on rows and the ones on columns must

have the same form. We will call this criterion 'Razor 1' and we will cut away quite a lot of cases already.
Also we know that:

> A g as desired must change the same number of entries in each region – the one that the corresponding p changes.

Furthermore, note that $gX = pX$ implies that $g^2X = p^2X$, simply because the relabelling p commutes with the symmetry g and everything is associative, therefore $g^2X = ggX = gpX = pgX = ppX = p^2X$. The contraposition of this implication gives us 'Razor 2': if g^2 is already cut out, then consequently g can be cut too. Here we use that for any relabelling p, p^2 is a relabelling too.
In the table beneath we give all data required for the calculation. The representatives are shown in the third column as ordered by GAP. The forth column shows a reason to know that there are no NRC-sudokus that are equal to themselves up to relabelling under the corresponding representative. If we do not have such a reason we indicate that the case has been counted by one of our brute-force algorithms. The last column shows the number of equivalence classes (through relabelling) that are fixed by the corresponding symmetry. If it is non-zero, we also need the size of the respective conjugacy class, which in this case is put in column 2.

1.	1	id		17463553760
2.		$(17,18)$	id on c5	0
3.		$(12,13)(17,18)$	id on c5	0
4.		$(11,19)(12,17)(13,18)(14,16)$	id on c5	0
5.		$(11,19)(12,17,13,18)(14,16)$	id on c5	0
6.		$(7,8)(17,18)$	id on B1	0
7.		$(7,8)(12,13)(17,18)$	id on B2	0
8.		$(7,8)(11,19)(12,17)(13,18)(14,16)$	Razor 1	0
9.		$(7,8)(11,19)(12,17,13,18)(14,16)$	Razor 1	0
10.		$(2,3)(7,8)(12,13)(17,18)$	id on B5	0
11.		$(2,3)(7,8)(11,19)(12,17)(13,18)(14,16)$	Razor 1	0
12.		$(2,3)(7,8)(11,19)(12,17,13,18)(14,16)$	Razor 1	0
13.		$(1,4)(6,9)(11,14)(16,19)$	1)	0
14.		$(1,4)(6,9)(11,14)(16,19)(17,18)$	Razor 1	0
15.		$(1,4)(6,9)(11,14)(12,13)(16,19)(17,18)$	Razor 1	0
16.		$(1,4)(6,9)(11,16)(12,17)(13,18)(14,19)$	Razor 1	0
17.		$(1,4)(6,9)(11,16)(12,17,13,18)(14,19)$	Razor 1	0
18.		$(1,4)(6,9)(7,8)(11,14)(16,19)(17,18)$	2)	0
19.		$(1,4)(6,9)(7,8)(11,14)(12,13)(16,19)(17,18)$	Razor 1	0
20.		$(1,4)(6,9)(7,8)(11,16)(12,17)(13,18)(14,19)$	Razor 1	0
21.		$(1,4)(6,9)(7,8)(11,16)(12,17,13,18)(14,19)$	Razor 1	0
22.	1	$(1,4)(2,3)(6,9)(7,8)(11,14)(12,13)(16,19)(17,18)$	counted	16384
23.	4	$(1,4)(2,3)(6,9)(7,8)(11,16)(12,17)(13,18)(14,19)$	counted	116336
24.		$(1,4)(2,3)(6,9)(7,8)(11,16)(12,17,13,18)(14,19)$	Razor 1	0
25.	4	$(1,6)(2,7)(3,8)(4,9)(11,16)(12,17)(13,18)(14,19)$	counted	135568

26.		$(1,6)(2,7)(3,8)(4,9)(11,16)(12,17,13,18)(14,19)$	Razor 1	0
27.		$(1,6)(2,7,3,8)(4,9)(11,16)(12,17,13,18)(14,19)$	Razor 2 with 10.	0
28.	4	$(1,9)(2,7)(3,8)(4,6)(11,19)(12,17)(13,18)(14,16)$	counted	155816
29.		$(1,9)(2,7)(3,8)(4,6)(11,19)(12,17,13,18)(14,16)$	Razor 1	0
30.		$(1,9)(2,7,3,8)(4,6)(11,19)(12,17,13,18)(14,16)$	Razor 2 with 10.	0
31.	8	$(1,11)(2,12)(3,13)(4,14)(5,15)(6,16)(7,17)(8,18)(9,19)$	counted	7784
32.		$(1,11)(2,12)(3,13)(4,14)(5,15)(6,16)(7,17,8,18)(9,19)$	Razor 2 with 6.	0
33.		$(1,11)(2,12,3,13)(4,14)(5,15)(6,16)(7,17,8,18)(9,19)$	Razor 2 with 10.	0
34.	16	$(1,11,9,19)(2,12,7,17)(3,13,8,18)(4,14,6,16)(5,15)$	counted	372
35.		$(1,11,9,19)(2,12,7,17,3,13,8,18)(4,14,6,16)(5,15)$	Razor 2 with 30.	0
36.	8	$(1,14)(2,12)(3,13)(4,11)(5,15)(6,19)(7,17)(8,18)(9,16)$	counted	520624
37.		$(1,14)(2,12)(3,13)(4,11)(5,15)(6,19)(7,17,8,18)(9,16)$	Razor 2 with 6.	0
38.		$(1,14)(2,12,3,13)(4,11)(5,15)(6,19)(7,17,8,18)(9,16)$	Razor 2 with 10.	0
39.	16	$(1,14,9,16)(2,12,7,17)(3,13,8,18)(4,11,6,19)(5,15)$	counted	264
40.		$(1,14,9,16)(2,12,7,17,3,13,8,18)(4,11,6,19)(5,15)$	Razor 2 with 30.	0

1) This symmetry keeps five entries of c5 fixed but changes five entries in B2.

2) This symmetry has only three fixed points in c5 but at least four in B1.

The programs used to count the non-zero cases can be found in Appendix C.
The result of our calculation is that there are

$$68\,239\,994$$

or

sixty-eight million two hundred thirty-nine thousand nine hundred ninety-four

essentially different NRC-sudokus.
It may also be interesting to consider less symmetries, as in fact Russel and Jarvis have done too. The NRC-transformation is surely the most ingenious and unnatural symmetry we have considered until now, so let us drop this one first.
Our group G will now be generated by

- reflection in the vertical axis $(11,19)(12,18)(13,17)(14,16)$
- transposition $(1,11)(2,12)(3,13)(4,14)(5,15)(6,16)(7,17)(8,18)(9,19)$
- (r2 r3) $(2,3)$

It has 128 elements that are split into 20 conjugacy classes.
The corresponding table of results consists of rows 1. to 12. and 28. to 35. of our former table.
The number of essentially different NRC-sudokus not considering the NRC-transformation is

$$136\,439\,416$$

or

one hundred thirty-six million four hundred thirty-nine thousand four hundred sixteen.

Also excluding transposition, G can be generated by

- reflection in the vertical axis $(11,19)(12,18)(13,17)(14,16)$
- reflection in the horizontal axis $(1,9)(2,8)(3,7)(4,6)$
- (r2 r3) $(2,3)$
- (c2 c3) $(12,13)$

Then G has 64 elements split into 25 conjugacy classes.
In the corresponding table, only the identity symmetry and 28. in our first table contribute.
The number of essentially different NRC-sudokus, considering neither the NRC-transformation nor transposition, is

$$272\,877\,766$$

or

> two hundred seventy-two million eight hundred seventy-seven thousand seven hundred sixty-six.

Bibliography

[1] B. Felgenhauer and A. F. Jarvis, Mathematics of Sudoku I, *Mathematical Spectrum* **39** (2006), 15–22; consulted on `www.afjarvis.staff.shef.ac.uk/sudoku/felgenhauer_jarvis_spec1.pdf`

[2] E. Russel and A. F. Jarvis, Mathematics of Sudoku II, *Mathematical Spectrum* **39**, 54–58; consulted on `www.afjarvis.staff.shef.ac.uk/sudoku/russell_jarvis_spec2.pdf`

[3] "Sudoku", *Wikipedia*, as of November 5th, 2007; permanent URL: `http://en.wikipedia.org/w/index.php?title=Sudoku&oldid=169295558`

[4] "Mathematics of Sudoku", *Wikipedia*, as of November 5th, 2007; permanent URL: `http://en.wikipedia.org/w/index.php?title=Mathematics_of_Sudoku&oldid=168941284`

[5] Discussion page on"Mathematics of Sudoku", *Wikipedia*, as of November 5th, 2007; permanent URL: `http://en.wikipedia.org/w/index.php?title=Talk:Mathematics_of_Sudoku&oldid=137768055`

[6] The GAP Group, *GAP*, version 4.4.10, URL: `http://www.gap-system.org`

Appendix A

The following program solves NRC-sudokus by backtracking. The NRC-specific commands are easily recognisable. Erasing them gives a solving program for ordinary sudokus. This program is based on a version proposed by Ulf Rehmann.

```
/* NRC-sudokusolver.c */

#include <stdio.h>

/*
002006090
100009705
000030000
037000050
800000000
000067008
080005000
015400600
000090004
*/

char  z,r;

char matrix[9][9];

char* pnrcmatrix[9][9];

char nrcnr[9] = {3,1,2,0,4,8,6,7,5};

void getmatrix() {
  char i,j,c;
  printf("Insert 9 sudoku rows top-to-bottom\n taking 0 for blanks: \n");
  for (i = 0; i < 9; i++) {
    for (j = 0; j < 9; j++) {
      while ( (c = getchar()) != EOF ) {
        if ( !isdigit(c) ) continue;
        else {
          matrix[i][j] = c - '0';
          break;
        }
      }
    }
  }
  return;
}

void output () {
  int i,j;
  for ( i = 0; i <= 8; i++ ) {
    for ( j = 0; j <= 8; j++ ) {
      printf("%d ",matrix[i][j]);
      if ( j % 3 == 2) printf(" ");
    }
```

```
    printf("\n");
    if ( i%3 == 2 ) printf("\n");
  }
  printf("-----------\n");
}

char check(char a, char b, char u) {
  int i,j;
  for (i = 0; i < 9; i++) {
    if (u == matrix[a][i]) return 1;
  }
  for (i = 0; i < 9; i++) {
    if (u == matrix[i][b]) return 2;
  }
  for ( i = a - a%3 ; i < a - a%3 + 3; i++) {
    for (j = b - b%3 ; j < b - b%3 + 3; j++)
    if (u == matrix[i][j]) return 3;
  }
  for ( i = nrcnr[a] - nrcnr[a]%3 ; i < nrcnr[a] - nrcnr[a]%3 +3 ; i++) {
    for ( j = nrcnr[b] - nrcnr[b]%3 ; j < nrcnr[b] - nrcnr[b]%3 +3 ; j++) {
      if ( u == *pnrcmatrix[i][j]) return 4;
    }
  }
  return 0;
}

char initcheck(char k) {
  char u;
  while ( matrix[k/9][k%9] == 0 && k <= 80 ) k++;
  if ( k == 81 ) return r;
  u = matrix[k/9][k%9];
  matrix[k/9][k%9] = 0;
  if ( check(k/9,k%9,u) == 1 ) {
    printf("Your puzzle contains a contradiction in row %d.\n", k/9+1);
    r = 1;
    }
  if ( check(k/9,k%9,u) == 2 ) {
    printf("Your puzzle contains a contradiction in column %d.\n", k%9+1);
    r = 2;
    }
  if ( check(k/9,k%9,u) == 3 ) {
    printf("Your puzzle contains a contradiction in block %d.\n",
    k/9-k/9%3+(k%9-k%9)%3/3+1);
    r = 3;
    }
  if ( check(k/9,k%9,u) == 4 ) {
    printf("Your puzzle contains a contradiction in one of the NRC-blocks\n");
    r = 4;
    }
  matrix[k/9][k%9] = u;
  initcheck(k+1);
}
```

```
void search (char k) {
  char i;
  while ( k <= 80 && matrix[k/9][k%9] ) k++;
  if (k == 81 ) {
    z++;
    if (z == 1) {
      printf("First solution found:\n");
      output();
    }
  }
  for ( i = 1; i <= 9 && z <= 1; i++) {
    if ( check(k/9, k%9, i) == 0) {
      matrix[k/9][k%9] = i;
      if ( k == 80) {
        z++;
        if (z == 1) {
          printf("First solution found:\n");
          output();
        }
        matrix[8][8] = 0;
        break;
      }
      else {
        search(k+1);
        matrix[k/9][k%9] = 0;
      }
    }
  }
}

int main() {
  char i, j;
  for ( i = 0; i <= 8; i++ ) {
    for ( j = 0; j <= 8; j++ ) {
      pnrcmatrix[i][j] = &matrix[nrcnr[i]][nrcnr[j]];
    }
  }
  getmatrix();
  output();
  if ( initcheck(0) == 0 ) {
    search(0);
    if (z == 0) printf("There is no solution to your puzzle.\n");
    if (z == 1) printf("This solution is unique!\n");
    if (z > 1) printf("This solution is not unique!\n");
  }
}
```

Appendix B

```
/* NRCcount.c */

#include <stdio.h>

int    z;

char matrix[9][9] = {
  {1,2,3,4,5,6,7,8,9},
  {0,0,0,0,0,0,0,0,0},
  {0,0,0,0,0,0,0,0,0},
  {0,0,0,0,0,0,0,0,0},
  {0,0,0,0,0,0,0,0,0},
  {0,0,0,0,0,0,0,0,0},
  {0,0,0,0,0,0,0,0,0},
  {0,0,0,0,0,0,0,0,0},
  {0,0,0,0,0,0,0,0,0},
};

char* pnrcmatrix[9][9];

char nrcnr[9] = {3,1,2,0,4,8,6,7,5};

char check(char a, char b, char u) {
  int i,j;
  for (i = 0; i < b; i++) {
    if (u == matrix[a][i]) return 1;
  }
  for (i = 0; i < a; i++) {
    if (u == matrix[i][b]) return 2;
  }
  for ( i = a - a%3 ; i < a - a%3 + 3; i++) {
    for (j = b - b%3 ; j < b - b%3 + 3; j++)
    if (u == matrix[i][j]) return 3;
  }
  for ( i = nrcnr[a] - nrcnr[a]%3 ; i < nrcnr[a] - nrcnr[a]%3 +3 ; i++) {
    for ( j = nrcnr[b] - nrcnr[b]%3 ; j < nrcnr[b] - nrcnr[b]%3 +3 ; j++)
    if ( u == *pnrcmatrix[i][j]) return 4;
  }
  return 0;
}

void search (char k) {
  char i;
  for ( i = 1; i <= 9; i++) {
    if ( k == 10 && i >= 7 ) break;
    if ( k == 11 && i <= matrix[1][1]) i = matrix[1][1]+1;
    if ( k == 15 && i == 9) break;
    if ( k == 16 && i <= matrix[1][6]) i = matrix[1][6]+1;
    if ( k == 19 && i <= matrix[1][1]) i = matrix[1][1]+1;
    if ( k == 20 && i <= matrix[1][1]) i = matrix[1][1]+1;
    if ( k == 28 && matrix[1][0] != 5 && matrix[2][0] != 5 && matrix[3][0] != 5 ) break;
```

```
    if ( k == 55 && i == 9 ) break;
    if ( k == 64 && i <= matrix[6][1] ) i = matrix[6][1]+1;
    if ( check(k/9, k%9, i) == 0) {
      matrix[k/9][k%9] = i;
      if ( k == 80) {
        z++;
        if (z%10000 == 0) {
          printf("%d\n", z);
          }
        matrix[8][8] = 0;
        break;
      }
      else {
        search(k+1);
        matrix[k/9][k%9] = 0;
      }
    }
  }
  if ( k == 9 ) printf("Calculation terminated: %d\n", z);
}

void main() {
  char i, j;
  for ( i = 0; i <= 8; i++ ) {
    for ( j = 0; j <= 8; j++ ) {
      pnrcmatrix[i][j] = &matrix[nrcnr[i]][nrcnr[j]];
    }
  }
  search(9);
}
```

Appendix C

With this program, contributed by W. van der Kallen, we counted the non-zero terms of the Burnside formula.

```
/* burnsidecount.c */

#include <stdio.h>

/* g is the symmetry (a,b) |--> (transrow(a,b), transcol(a,b)) */
/* We will count the number of sudokus X for which there is a  */
/* relabelling pi with g X = pi X                              */
/* This version works for g of order 2 and also for g of       */
/* order 4 whose square fixes only one cell.                   */
/* example: to input case nr. 23 type
   444444444 333333333 222222222 111111111 555555555 999999999
   888888888 777777777 666666666
   678951234 678951234 678951234 678951234 678951234 678951234
   678951234 678951234 678951234                              */

int z;

char nrcnr[9]= {3,1,2,0,4,8,6,7,5};

char pi[10];

char piinv[10];

char matrix[9][9];

char transrow[9][9];

char transcol[9][9];

char* pnrcmatrix[9][9];

char* ptrans[9][9];

char* ptransinv[9][9];


char check(char a, char b, char u) {
  int i,j,k,l;
  for (i = 0; i < 9; i++)
    if (u == matrix[a][i]) return 1;
  for (i = 0; i < 9; i++)
    if (u == matrix[i][b]) return 2;
  for ( i = a - a%3 ; i < a - a%3 + 3; i++)
    for (j = b - b%3 ; j < b - b%3 + 3; j++)
          if (u == matrix[i][j]) return 3;
  k=nrcnr[a] - nrcnr[a]%3 ;
  l=nrcnr[b] - nrcnr[b]%3 ;
  for ( i = k ; i < k +3 ; i++)
    for ( j = l ; j < l +3 ; j++)
```

```
      if ( u == *pnrcmatrix[i][j]) return 4;
  return 0;
}


char checkpi () {
  char i;
  for ( i = 1; i <= 9; i++) pi[i] = matrix[0][i-1];
  /* we aim for pi[ *ptrans[i][j] ] == matrix[i][j] */
  for ( i = 0; i <= 8; i++)
    if ( (*ptransinv[0][i] != 0) && ( *ptransinv[0][i] != pi[matrix[0][i]] )) return 1;
  for ( i = 1; i <= 9; i++)
    if ( pi[pi[pi[pi[i]]]] != i ) return 2;
  for ( i = 1; i <= 9; i++) piinv[pi[i]]= i ;
  return 0;
}


void init () {
  char i,j,c;
  for ( j = 0; j <= 8; j++ )
    for ( i = 0; i <= 8; i++ )
      while ( (c=getchar()) != EOF )
        if ( !isdigit(c) ) continue;
        else {
          transrow[j][i] = c - '1';
          break;
        }
  for ( j = 0; j <= 8; j++ )
    for ( i = 0; i <= 8; i++ )
         while ( (c=getchar()) != EOF )
             if ( !isdigit(c) ) continue;
       else {
          transcol[j][i] = c - '1';
          break;
       }
  for ( j = 0; j <= 8; j++ )
    for ( i = 0; i <= 8; i++ ) {
         pnrcmatrix[i][j]= &matrix[nrcnr[i]][nrcnr[j]];
         ptrans[i][j]= &matrix[transrow[i][j]][transcol[i][j]];
         ptransinv[transrow[i][j]][transcol[i][j]]= &matrix[i][j];
    }
  for ( j = 0; j <= 8; j++ ) *ptrans[0][j] = j+1;
}
/* init fills in the image of r1 under g. Thus r1 will determine pi */


void search2 (char k) {
  char i;
  while ( k <= 80 && matrix[k/9][k%9] ) k++;
  if ( k == 81 ) z++;
  else
    for ( i = 1; i <= 9; i++) {
```

```
      if ( check(k/9,k%9,i) == 0 ) {
        matrix[k/9][k%9] = i; /* try this digit */
  if (( (*ptransinv[k/9][k%9] != 0) && ( piinv[*ptransinv[k/9][k%9]] != i ))||
        ((*ptrans[k/9][k%9] != 0) && ( pi[*ptrans[k/9][k%9]] != i ))) {
              matrix[k/9][k%9] = 0; continue;
        }
        if ( *ptrans[k/9][k%9] == 0) { /* try to fill this cell also */
          if ( check(transrow[k/9][k%9],transcol[k/9][k%9],piinv[i]) == 0 ) {
            *ptrans[k/9][k%9] = piinv[i]; /* try this digit */
      search2(k+1);
            *ptrans[k/9][k%9] = 0;
      /* the instance of search2 that tries a digit must clean it up */
          }
        } else  search2(k+1);
  matrix[k/9][k%9] = 0; /* clean up */
      }
    }
}

void search1 (char k) {
  char i;
  while ( k <= 8 && matrix[k/9][k%9] ) k++;
  if ( k>=9 ) {
        if ( checkpi()==0 )  search2(9);
  } else
    for ( i = 1; i <= 9; i++ )
    if ( check(k/9, k%9, i) == 0 ) {
      matrix[k/9][k%9] = i;
      search1(k+1);
      matrix[k/9][k%9] = 0; /* clean up */
    }
}
/* search1 fills r1 arbitrarily but validly;
this induces the relabelling pi: i |-~-> matrix[0][i-1] */

int main() {
init();
search1(0);
printf("Result: %d\n", z);
return 0;
}
```