Mathematics of Sudoku

Ambra Lisi

Level 6, 20 credit points
Andrew Booker
12 March 2025

Acknowledgement of Sources

Signed [illegible]

Date 12/03/2025

Contents

1 Introduction

Sudoku is a popular number puzzle that needs only logic to solve. Though it does not require mathematics itself, there are still mathematical questions that we can consider about it. We will motivate this study by exploring the origins of Sudoku and the research already done on it. Then move onto answering two important questions: how many Sudoku grids are there and, of these grids, which do we consider 'essentially different'? We investigate this on a smaller scale, working with 4×4 Sudoku grids to formulate mathematical methods of answering these questions, which were similarly used by researchers working on 9×9 Sudokus before.

2 Motivation

We begin by introducing what a Sudoku is and what its historical origins are. This will lead to exploring mathematical questions surrounding it and allow us to choose specific questions to focus on.

2.1 History of Sudoku

Most people are at least familiar with the popular number puzzle known as Sudoku. It features in all kinds of newspapers around the world and has become a regular pass-time for many puzzle enthusiasts.

A classic Sudoku is a 9×9 grid sectioned into 9 smaller squares. The objective is to fill each row, column, and square with the numbers 1 to 9 only once. The grid is given with a few numbers already in place, allowing the puzzler to use logic clues to fill out the rest of the numbers.

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

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

Puzzles involving filling out square grids have existed for centuries, but it is probably the Latin Square that resembles our modern Sudoku most closely. Published works with Latin Squares can be found as far back as the 17th century, by Choi Seok-jeong, a Korean politician and mathematician, but there are remarks that suggest they may have been used even earlier. Leonard Euler is most often attributed for the existence of Latin Squares, as he presented papers that made use of them in 1776 [1].

Latin squares themselves are just $n \times n$ grids where the rows and columns are filled with n different numbers or symbols only once. For example, a 5×5 Latin Square may look something like the following.

1	2	3	4	5
2	3	4	5	1
3	4	5	1	2
4	5	1	2	3
5	1	2	3	4

A Sudoku, is then simply a 9×9 Latin Square with further restrictions imposed by the smaller squares.

The creation of Sudoku as we know it is believed to have been invented by American architect, Howard Garns. In the late 1970s, *Dell Magazines* published his puzzles under the name Number Place [2]. Though it did not gain immediate popularity, it did lead to a Japanese company publishing these puzzles in 1984 and coining the term *Su Doku*, a shortening of the phrase 'Suuji wa dokushin ni kagiru', which means 'that number is limited only single (unmarried)' [3]. It gained widespread popularity in Japan, which led to it being picked up by UK newspapers in 2004 and it has been popular globally ever since.

Due to Sudoku's relatively recent invention, the mathematical research done into it has also only been explored in the last 25 years or so and, in a field such as mathematics, where most principles were conceived hundreds of years ago, it is exciting when we can explore something in a new context such as this.

2.2 What mathematical questions can we answer?

As discussed earlier, Sudoku only requires logical thinking to solve; we notice the restrictions caused by current numbers already in the grid and fill out the remaining numbers. So if it does not require mathematical knowledge to solve, what can mathematicians study about it?

For example, some PhD researchers have studied using Latin Squares and multiplication tables, known as Cayley tables, to find how many solved Sudoku grids are mutually orthogonal [4]. Others have asked questions about the minimum number of clues necessary to ensure a unique Sudoku grid solution (this turns out to be 17, which was calculated using hitting set enumeration by Gary McGuire, Bastian Tugemann, and Gilles Civario in 2014 [5]). There are also many computer programs written to generate playable Sudoku puzzles for newspapers and apps, and similarly there are many programs written to solve Sudoku grids [6]. Often they use something called backtracking algorithms to one-by-one insert a number into the grid and backtrack only if it clashes and does not generate a valid solution.

The research in which we are most interested is that done by Bertram Felgenhauer and Frazer Jarvis and published as a report under the title *Mathematics of Sudoku I* in 2006 [7]. It detailed their method of calculating the total number of filled Sudoku grids that exist. They set the top left 3×3 square in the Sudoku as numbers 1 to 9 in order, and from there logically calculated the other possible ways of filling the grids. This led to them finding there are $6670903752021072936960 \approx 6.671 \times 10^{21}$ total Sudoku grids, which has since been confirmed by several other researchers. Along with this, Ed Russell and Frazer Jarvis published the report *Mathematics of Sudoku II* [8] which, using the total number of Sudoku grids, found the number of those considered 'essentially different'. They used Burnside's lemma and other Group Theory results (along with a computer program called GAP) to calculate that there are 5472730538 different grids.

We want to explore the same questions that Bertram Felgenhauer, Ed Russell, and Frazer Jarvis did, but on a smaller scale. In the next section we will introduce a 4×4 Sudoku grid and begin by answering how many total grids there are.

3 How many 4×4 Sudoku grids are there?

To answer the question of how many 9×9 Sudoku grids there are, requires a computer to do calculations, which is what Felgenhauer and Jarvis did previously. We want to imitate their work, but on a much more manageable scale. Therefore we can use a smaller version of the regular Sudoku.

3.1 Shidoku

A 4×4 Sudoku is sometimes referred to as a *mini-Sudoku* or a *Shidoku* where the term 'shi' means 'four' in Japanese. Just like a regular Sudoku, a Shidoku is a 4×4 Latin Square, which has smaller 2×2 squares in each corner. Each row, column, and square must be filled with the numbers 1 to 4 only once.

Throughout this project report we will be working with Shidoku, and if we refer to a Shidoku grid, we mean one that is filled out. We have no need to consider incomplete grids with clues, still to be solved.

3.2 Counting Shidoku grids

The total number of Shidoku grids can most easily be calculated by brute-force. The simplest method for this is by using the same idea Felgenhauer and Jarvis had, and setting the top left square as so:

1	2		
3	4		

.

Then we want to consider all possible ways of filling out the rest of the grid. We can start by filling in the first column in the only two possible ways:

either

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

or

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

.

If we focus on the first option, we can choose where to place 1 and 3 in the second column, choose where to place 3 and 4 in the first row, and finally where to place 1 and 2 in the second row. These will be enough filled in numbers to create unique Shidoku grids, along with some non-valid ones we will eliminate.

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

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

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

1	**2**	4	3
3	**4**	1	2
2	3	X	1
4	1	2	X

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

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

1	**2**	3	4
3	**4**	2	1
2	3	1	X
4	1	X	2

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

This leaves us with six valid Shidoku grids for our first batch. Similarly, the second batch produces the following grids.

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

1	**2**	4	3
3	**4**	1	2
4	1	3	X
2	3	X	4

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

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

1	**2**	3	4
3	**4**	2	1
4	1	X	2
2	3	1	X

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

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

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

Therefore we have the following 12 valid Shidoku grids, which we'll label $\mathbb{G}_1$ to $\mathbb{G}_{12}$ for future convenience.

$\mathbb{G}_1 =$

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

$\mathbb{G}_2 =$

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

$\mathbb{G}_3 =$

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

$\mathbb{G}_4 =$

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

$\mathbb{G}_5 =$

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

$\mathbb{G}_6 =$

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

$\mathbb{G}_7 =$

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

$\mathbb{G}_8 =$

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

$$\mathbb{G}_9 = \begin{array}{|c|c||c|c|}\hline \mathbf{1} & \mathbf{2} & 4 & 3 \\ \hline \mathbf{3} & \mathbf{4} & 1 & 2 \\ \hline\hline 4 & 3 & 2 & 1 \\ \hline 2 & 1 & 3 & 4 \\ \hline \end{array} \quad \mathbb{G}_{10} = \begin{array}{|c|c||c|c|}\hline \mathbf{1} & \mathbf{2} & 4 & 3 \\ \hline \mathbf{3} & \mathbf{4} & 2 & 1 \\ \hline\hline 4 & 1 & 3 & 2 \\ \hline 2 & 3 & 1 & 4 \\ \hline \end{array} \quad \mathbb{G}_{11} = \begin{array}{|c|c||c|c|}\hline \mathbf{1} & \mathbf{2} & 3 & 4 \\ \hline \mathbf{3} & \mathbf{4} & 2 & 1 \\ \hline\hline 4 & 3 & 1 & 2 \\ \hline 2 & 1 & 4 & 3 \\ \hline \end{array} \quad \mathbb{G}_{12} = \begin{array}{|c|c||c|c|}\hline \mathbf{1} & \mathbf{2} & 4 & 3 \\ \hline \mathbf{3} & \mathbf{4} & 2 & 1 \\ \hline\hline 4 & 3 & 1 & 2 \\ \hline 2 & 1 & 3 & 4 \\ \hline \end{array}$$

We found there are 12 different Shidoku grids given the top left square is set. Therefore, if we set the top left square as any other combination of the numbers 1 to 4, we would again find that there are 12 total Shidoku grids with this restriction. Then, because there are $4! = 24$ possible ways of relabelling these 4 numbers in the top left square, we actually get that there are

$$4! \cdot 12 = 24 \cdot 12 = 288 \text{ total Shidoku grids.}$$

Compared to the 6.671×10^{21} or so total Sudoku grids, 288 Shidoku grids is a much smaller and more manageable number. We can next move onto defining 'essentially different' grids, and doing mathematics along the same line of logic as *Mathematics of Sudoku II* [8].

4 How many essentially different 4×4 Sudoku grids are there?

In this section we will define what we mean by 'essentially different' and go over the brute-force method of finding the essentially different Shidoku grids. Then, in line with what Russell and Jarvis did before, we will introduce Burnside's lemma and form the group of operations so that we can count which Shidoku grids remain fixed. This will let us calculate the essentially different grids using Burnside's lemma, which is what was used to calculate the essentially different 9×9 Sudoku grids before.

4.1 Essentially different

We have found there are 288 total different Shidoku grids, but there is another definition we may want to consider for the number of grids.

Definition 4.1 (Essentially different)**.** We consider grids *essentially the same* if there is a sequence of the operations (1)–(3) which transforms one of the grids into another. These operations are:

1. row and column permutations,
2. reflections and rotations, and
3. relabelling.

If grids are not essentially the same, we call them *essentially different.*

What are the number of *essentially different* Shidoku grids? We can find the answer by brute-force doing some calculations. If we consider just the 12 Shidoku grids listed earlier, we can perform operations on each one and observe if it becomes equal to other grids. We can use any of the operations from earlier, granted that it generates another valid Shidoku grid.

For example, if we use a row permutation on the first Shidoku grid, $\mathbb{G}_1$ and then relabel it so that the top left square matches our previous grids, we find that this grid is essentially the same as one of our other ones, $\mathbb{G}_8$.

$$\mathbb{G}_1 = \begin{array}{|cc|cc|}\hline 1&2&3&4\\ 3&4&1&2\\ \hline 2&1&4&3\\ 4&3&2&1\\ \hline\end{array} \quad \xrightarrow{\substack{\text{swap first and}\\ \text{second row}}} \quad \begin{array}{|cc|cc|}\hline 3&4&1&2\\ 1&2&3&4\\ \hline 2&1&4&3\\ 4&3&2&1\\ \hline\end{array} \quad \xrightarrow{\text{relabel}} \quad \begin{array}{|cc|cc|}\hline 1&2&3&4\\ 3&4&1&2\\ \hline 4&3&2&1\\ 2&1&4&3\\ \hline\end{array} = \mathbb{G}_8$$

Similarly, we could reflect $\mathbb{G}_8$ along its diagonal, from top left to bottom right, and find it is essentially the same as $\mathbb{G}_5$.

$$\mathbb{G}_8 = \begin{array}{|cc|cc|}\hline 1&2&3&4\\ 3&4&1&2\\ \hline 4&3&2&1\\ 2&1&4&3\\ \hline\end{array} \quad \xrightarrow{\substack{\text{reflect along}\\ \text{diagonal}}} \quad \begin{array}{|cc|cc|}\hline 1&3&4&2\\ 2&4&3&1\\ \hline 3&1&2&4\\ 4&2&1&3\\ \hline\end{array} \quad \xrightarrow{\text{relabel}} \quad \begin{array}{|cc|cc|}\hline 1&2&4&3\\ 3&4&2&1\\ \hline 2&1&3&4\\ 4&3&1&2\\ \hline\end{array} = \mathbb{G}_5$$

At the very least, we have now found that $\mathbb{G}_1$, $\mathbb{G}_5$ and $\mathbb{G}_8$ are all essentially the same as, by definition, there is a sequence of operations which transforms one of the grids into another.

Continuing to perform these calculations would allow us to observe the following. The grids $\{\mathbb{G}_1, \mathbb{G}_5, \mathbb{G}_8, \mathbb{G}_{12}\}$ are all essentially the same as each other. Similarly, the grids $\{\mathbb{G}_2, \mathbb{G}_3, \mathbb{G}_4, \mathbb{G}_6, \mathbb{G}_7, \mathbb{G}_9, \mathbb{G}_{10}, \mathbb{G}_{11}\}$ are also all essentially the same grid. However, we would not be able to find any operations that would transform a grid from $\{\mathbb{G}_1, \mathbb{G}_5, \mathbb{G}_8, \mathbb{G}_{12}\}$ into a grid from $\{\mathbb{G}_2, \mathbb{G}_3, \mathbb{G}_4, \mathbb{G}_6, \mathbb{G}_7, \mathbb{G}_9, \mathbb{G}_{10}, \mathbb{G}_{11}\}$ and vice versa. Therefore, these represent the only 2 essentially different Shidoku grids with the top left square set.

Any Shidoku grid not already included in our 12, is simply a relabelling of one of those 12 grids. Since relabelling is an operation included in our definition of essentially the same, then any other grid is essentially the same as one of our 12 grids, $\mathbb{G}_1$ to $\mathbb{G}_{12}$. Due to there only being 2 essentially different grids among our 12, we have also found that out of 288 total Shidoku grids, there are only 2 essentially different grids.

Using some brute-force calculations, we have found the answer to our question already. However, this is only possible because of the small number of total Shidoku grids. Sudoku, because it is a 9×9 grid, has a much larger number of possible filled grids. Trying to do the same brute-force calculations with all Sudoku grids would not be feasible. We want a method that can be replicated on a larger scale. What Russell and Jarvis did instead, and what we can do as well, is use Group Theory and something called Burnside's lemma to find the number of essentially different Shidoku grids. To introduce Burnside's lemma, we should first introduce some other definitions.

4.2 Burnside's lemma

In order to understand Burnside's lemma, we must first understand some other concepts about groups. You should be familiar with the definitions of binary operations and sets already. You should also know what a group is, though we will define it anyway [9].

Definition 4.2 (Group)**.** A *group* $(G, *)$ is a non-empty set G with a binary operation $*$ such that

- $*$ is associative;
- G contains an identity element e for $*$; and
- for every element $g \in G$ there exists an element $g^{-1} \in G$ such that $g^{-1} * g = e$.

We will be using groups that contain the operations mentioned in the definition of *essentially different.* There will be a non-infinite number of operations, so we are only working with finite groups, where a finite group is a group that has a finite number of elements. The way that groups interact with other mathematical elements is also important to define, especially because the operations in our group are those that act on Shidoku grids. The general framework for such interactions are called *group actions.*

Definition 4.3 (Group action)**.** Let G be a group and X a non-empty set. Suppose that for each $g \in G$ we assign a map $g : X \to X$. We say that this assignment defines an *action of* G *on* X, or G*-action on* X, if

1. $e(x) = x$ for all $x \in X$;
2. $(gh)(x) = g(h(x))$ for all $g, h \in G$ and $x \in X$.

A group action is essentially just a group of elements that act on the elements in a set. The group action we will be looking at is the group of all operations listed earlier. These operations act on the set of Shidoku grids, and so the group containing all operations is our group action. There are other important definitions relating to group actions.

Definition 4.4 (Orbit; stabiliser)**.** Let G be a group acting on a set X. Then for every $x \in X$, the set

$$\mathrm{Orb}_G(x) = \{g(x) : g \in G\}$$

is called the *orbit* of x. The set

$$\mathrm{Stab}_G(x) = \{g \in G : g(x) = x\}$$

is called the *stabiliser* of x.

The orbits for our set of Shidoku grids represent the classes of equivalent Shidoku grids, where equivalent means they are essentially the same. The number of orbits would be the number of essentially different grids.

Definition 4.5 (Fixed point)**.** Let G be a group acting on a set X. Then an element $x \in X$ is called a *fixed point* for this action if $\mathrm{Orb}_G(x) = \{x\}$.

Our fixed elements will be Shidoku grids that remain the same after being acted on by elements in our group action. We have now defined all the definitions necessary in order to understand Burnside's lemma. Though you may want to note that Burnside's lemma is known by several different names, including Burnside's counting theorem, the Cauchy-Frobenius lemma, and the orbit-counting theorem.

Theorem 4.6 (Burnside's lemma). *Let G be a finite group that acts on a set X. For each $g \in G$, let X^g denote the set of elements in X that are fixed by g: that is, $X^g = \{x \in X | g \cdot x = x\}$. The number of orbits, denoted $|X/G|$ is:*

$$|X/G| = \frac{1}{|G|}\sum_{g\in G}|X^g| \tag{1}$$

There is a simple proof [10] for Burnside's lemma that we can understand if we know the *orbit-stabiliser theorem.*

Theorem 4.7 (Orbit-stabiliser theorem). *Suppose G is a group acting on a set X and let $x \in X$. Then*

$$|\operatorname{Orb}_G(x)| = [G : \operatorname{Stab}_G(x)].$$

In particular, if G is finite, then $|G| = |\operatorname{Orb}_G(x)||\operatorname{Stab}_G(x)|$ by Lagrange's theorem.

Proof of Burnside's lemma.

$$\begin{aligned}
\sum_{g\in G}|X^g| &= \sum_{g\in G}|\{x \in X : g \cdot x = x\} \\
&= \sum_{x\in X}|\{g \in G : g \cdot x = x\}| \\
&= \sum_{x\in X}|\operatorname{Stab}_G(x)| && \text{by definition of a stabiliser} \\
&= \sum_{x\in X}\frac{|G|}{|\operatorname{Orb}_G(x)|} && \text{by the orbit-stabiliser theorem} \\
&= |G|\left(\sum_{x\in X}\frac{1}{|\operatorname{Orb}_G(x)|}\right) \\
&= |G|\sum_{\operatorname{Orb}_G(x)\in X/G}\left(\sum_{x\in\operatorname{Orb}_G(x)}\frac{1}{|\operatorname{Orb}_G(x)|}\right) \\
&= |G|\sum_{\operatorname{Orb}_G(x)\in X/G} 1 \\
&= |G||X/G|
\end{aligned}$$

□

Because Burnside's lemma allows us to find the number of orbits, we would be finding the number of classes of essentially the same grids. That is, we find the number of essentially different Shidoku grids.

In order to use the formula in Burnside's lemma, we must count the number of Shidoku grids that are fixed for each operation. To do so, we need to construct the group that contains all operations. We start this by forming the group that represents row and column permutations, then reflections and rotations, and finally relabellings. Once we have constructed our group of operations, we can count the fixed grids and implement Burnside's lemma [11].

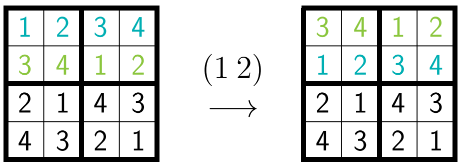
4.3 Group of operations

In this section we aim to construct the group containing all operations that act on the Shidoku grids. We begin by working on row and column operations and move onto reflections, rotations and relabellings.

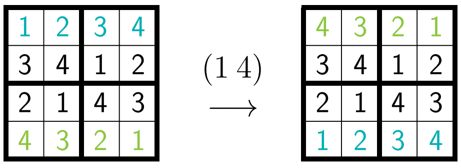
4.3.1 Row and column permutations

We start by finding the group of all row permutations. To do so we need to find the elements that represent rows swapping with other rows, but still creating valid Shidoku grids. We can see this best by labelling all rows as numbers 1 to 4. We will represent row permutations as disjoint cycles. If you are unfamiliar with the notation, consider, for example, $(1\,2)(1\,3)(2\,4)$. If we follow this from right to left, we see that $1 \mapsto 3$, $3 \mapsto 1 \mapsto 2$, $2 \mapsto 4$, $4 \mapsto 2 \mapsto 1$, and so $(1\,2)(1\,3)(2\,4)$ can more simply be written as $(1\,3\,2\,4)$.

An easy row permutation to find is the swapping of rows 1 and 2 of a Shidoku grid, represented as $(1\,2)$. This produces another valid Shidoku grid regardless of which grid we are applying it to, because it permutes rows without disturbing the smaller 2×2 squares.



However, we find that not all swaps produce valid grids, take the following as an example.



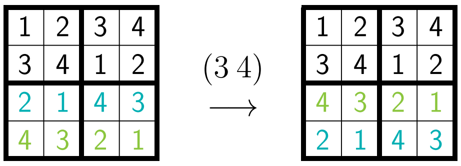
This transformation, represented by $(1\,4)$ does not produce a valid grid for all Shidoku; we see each square has repeated numbers and therefore is not a valid grid. So, how do we find all permutations that produce valid grids? We could again implement brute-force methods to go through all possible transformations and find which are valid, but we can also find a set of a few basic ones and then generate the whole group using this set as so [9].

Definition 4.8 (Subgroup generated by a set)**.** Let G be a group, and let $X \subseteq G$. We call

$$\langle X\rangle = \{x_1^{\epsilon_1}x_2^{\epsilon_2}\dots x_m^{\epsilon_m} : x_i \in X, \epsilon_i \in \{1,-1\}, m \in \mathbb{N}_0\},$$

the subgroup generated by X. If $\langle X\rangle = G$ then we say that X is a *generating set* for G.

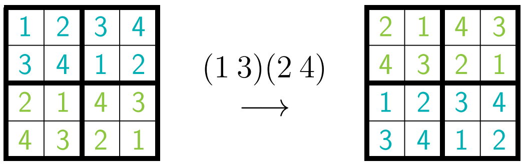
Hence, we aim to find enough row permutations to form a set that generates our group. For example, we can easily notice that (3 4) and (1 3)(2 4) are also valid transformations, as they also keep the smaller 2×2 squares intact while permuting rows.



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

$\xrightarrow{(3\,4)}$

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



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

$\xrightarrow{(1\,3)(2\,4)}$

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

There are three valid row transformations found so far and so our group is $\langle(1\ 2),(3\ 4),(1\ 3)(2\ 4)\rangle$. The process to generate the group using these elements is by combining them as follows:

- $(1\,2)(1\,2) = e$;
- $(1\,3)(2\,4) = (1\,3)(2\,4)$;
- $(1\,2)(1\,3)(2\,4) = (1\,3\,2\,4)$;
- $(3\,4)(1\,3)(2\,4) = (1\,4\,2\,3)$; and
- $(1\,2)(3\,4)(1\,3)(2\,4) = (1\,4)(2\,3)$.

Here, e is the identity element that represents no row swaps. We could also visualise it as $e = (1)(2)(3)(4)$. In total we now have 8 elements, and any other combination of these elements will continue to give us elements we have already found. Hence, we let

$$R = \{e,(1\,2),(3\,4),(1\,3)(2\,4),(1\,2)(3\,4),(1\,4\,2\,3),(1\,3\,2\,4),(1\,4)(2\,3)\}$$

be the set of all row permutations. We can prove that this set of row permutations is a group.

Proof. We know R is non-empty, and the operation we consider is just the action of applying two elements of R together, which is associative. Then R contains an identity element $e = (1\ 2\ 3\ 4)$ since for any $r \in R$, $er = r$ and $re = r$. It is trivial to check that this is true for all elements in R. Now we need to show that for every element

$r \in R$, there exists an element $r^{-1} \in R$ such that $r^{-1}r = e$. It is trivial to show that $e, (1\,2), (3\,4), (1\,3)(2\,4), (1\,2)(3\,4)$, and $(1\,4)(2\,3)$ are all self-inverse. For example:

$$(1\,2)(1\,2) = (1)(2)(3)(4) = e.$$

We also find that $(1\,3\,2\,4)$ and $(1\,4\,2\,3)$ are inverses of each other:

$$(1\,4\,2\,3)(1\,3\,2\,4) = (1)(2)(3)(4) = e,$$

$$(1\,3\,2\,4)(1\,4\,2\,3) = (1)(2)(3)(4) = e.$$

Therefore R is a group. $\square$

Though we have now found R, there is still an interesting relation we can show. To introduce this you should refamiliarise yourself with what an isomorphism is.

Definition 4.9 (Isomorphism)**.** Let $(G, *)$ and $(H, \cdot)$ be groups. Then an *isomorphism* from $(G, *)$ to $(H, \cdot)$ is a bijection $\varphi : G \to H$ such that $\varphi(x * y) = \varphi(x) \cdot \varphi(y)$ for all $x, y \in G$. If such a map exists, we say that G and H are isomorphic, denoted $G \cong H$.

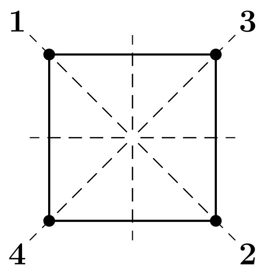
We find that R is isomorphic to the group of all symmetries of a square, the dihedral group D_4, where

$$D_4 = \{e, r, r^2, r^3, s, sr, sr^2, sr^3\}.$$

Note that:

- $e = (1)(2)(3)(4) =$ the identity element,
- $r =$ a 90° clockwise rotation, and
- $s =$ a reflection down the vertical axis.

The easiest way to see the relation between R and D_4 is by labelling each axis of a square with the numbers 1 to 4 as done in the diagram below. These numbers then correspond to the labelled rows in the Shidoku grid. It is important that the numbers 1 and 2 are on opposite axis of the square, and same with 3 and 4. This layout causes the elements in D_4 to correspond to valid row transformations.



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

Then performing any of the transformations in D_4 on the square, represents an element in R. For example, r sends $1 \mapsto 3$, $3 \mapsto 2$, $2 \mapsto 4$, and $4 \mapsto 1$, which represents the same tranformation as $(1\,3\,2\,4)$. We find the following.

- $e \mapsto e$
- $r \mapsto (1\,3\,2\,4)$
- $r^2 \mapsto (1\,2)(3\,4)$
- $r^3 \mapsto (1\,4\,2\,3)$
- $s \mapsto (1\,3)(2\,4)$
- $sr \mapsto (3\,4)$
- $sr^2 \mapsto (1\,4)(2\,3)$
- $sr^3 \mapsto (1\,2)$

Therefore there is a clear isomorphism between D_4 and R. We can prove this mathematically.

Proof. We know that we can represent R and D_4 by their generators as so

$$R = \langle (1\,2), (3\,4), (1\,3)(2\,4) \rangle,$$

$$D_4 = \langle r, s \mid r^4 = s^2 = e, srs = r^{-1} \rangle.$$

Let $\varphi : D_4 \to R$ be the mapping defined above. Then $\varphi(r) = (1324)$ and $\varphi(s) = (13)(24)$, such that:

- $\varphi(sr) = (3\,4) = (1\,3)(2\,4)(1\,3\,2\,4) = \varphi(s)\varphi(r)$,
- $\varphi(r^4) = \varphi(e) = e = (1\,3\,2\,4)(1\,3\,2\,4)(1\,3\,2\,4)(1\,3\,2\,4) = \varphi(r)^4$,
- $\varphi(s^2) = \varphi(e) = e = (3\,4)(3\,4) = \varphi(s)^2$,
- $\varphi(srs) = \varphi(r^{-1}) = (1\,4\,2\,3) = (1\,3)(2\,4)(1\,3\,2\,4)(1\,3)(2\,4) = \varphi(s)\varphi(r)\varphi(s)$.

Then φ is at least a homomorphism between D_4 and R. The kernel of φ is the set $\ker(\varphi) = \{g \in D_4 : \varphi(g) = e\} = \{e\}$. Because the kernel is just the identity element, φ must be injective, and because $|D_4| = 8 = |R|$, then this also implies that φ is surjective. Therefore φ is a bijective homomorphism, and so an isomorphism between D_4 and R.

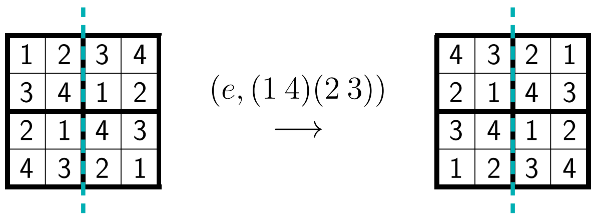
$\square$

The group of all row permutations can therefore be represented by just D_4. If we label the columns of a Shidoku grid as the numbers 1 to 4, then we can follow the exact same line of logic and conclude that the group of column permutations is also D_4. The group representing both row and column permutations would be the direct product $D_4 \times D_4$. We will represent row and column operations as elements $(r, c) \in D_4 \times D_4$ where $r, c \in D_4$. Then applying (r, c) to a Shidoku grid would apply the row operation r, and the column operation c, in no particular order, because row and column operations themselves do not affect each other.

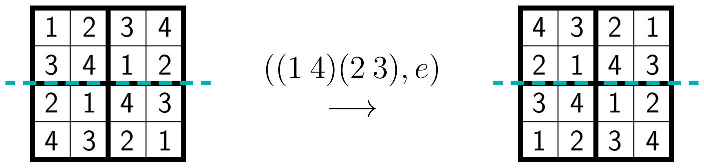
4.3.2 Reflections and rotations

Now that we have represented our row and column operations as the group $D_4 \times D_4$, we can move onto representing the possible reflections and rotations. Since a Shidoku grid is itself a square, then any reflection and rotation can also transform the grid. We can note that any reflection along any axis, or rotation by any 90° increment will still produce a valid Shidoku grid, so we need to find a way to represent all of them in one group with our row and column permutations.

Reflection along the horizontal or vertical axis can already be represented by a row or column operation. For example, consider the reflection through the vertical axis.

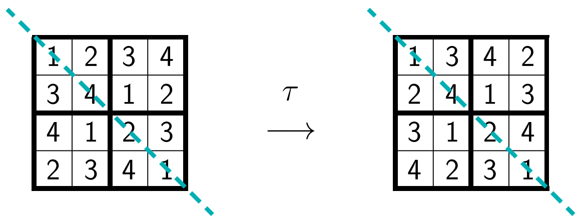


This operation sends columns $1 \mapsto 4$, $2 \mapsto 3$, $3 \mapsto 2$, and $4 \mapsto 1$. This is the same as the column operation represented by $(1\,4)(2\,3)$ or sr^2, and so is already represented in the group $D_4 \times D_4$ as the element $(e, (1\,4)(2\,3))$. Similarly, the reflection along the horizontal axis is the row operation represented by $((1\,4)(2\,3), e) \in D_4 \times D_4$.

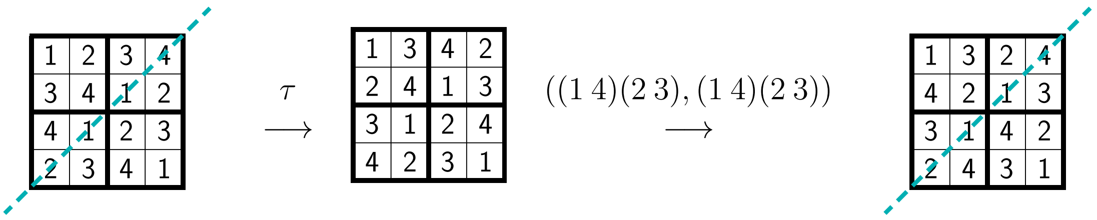


Specifically on the grid $\mathbb{G}_1$ these transformations even generate the same Shidoku grid, though this would not be the case on all grids. As for reflections through the diagonal axis, there is no element $x \in D_4 \times D_4$ such that x reflects all Shidoku grids through a diagonal [12].

Let τ be an element that, when applied to a Shidoku grid, reflects it along the diagonal from top left to bottom right. Then applying τ to $\mathbb{G}_7$ does the following.

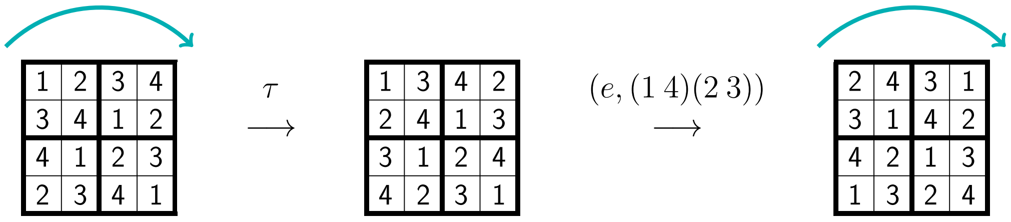


The last reflection possible is the one through the opposite diagonal. We can successfully apply this operation by applying τ and then the row and column operation $(1\,4)(2\,3)$. Since this is possible with τ and an element already in $D_4 \times D_4$, we have no need to introduce another element.

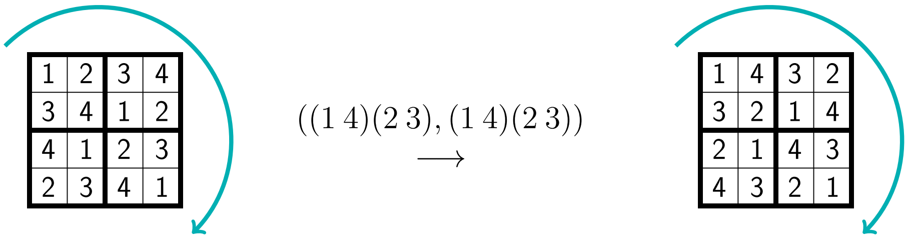


We have applied all possible reflections, so we next need to consider rotations, including the 90° , 180° and 270° clockwise rotations. Interestingly, we can represent these rotations also using τ and elements in $D_4 \times D_4$.

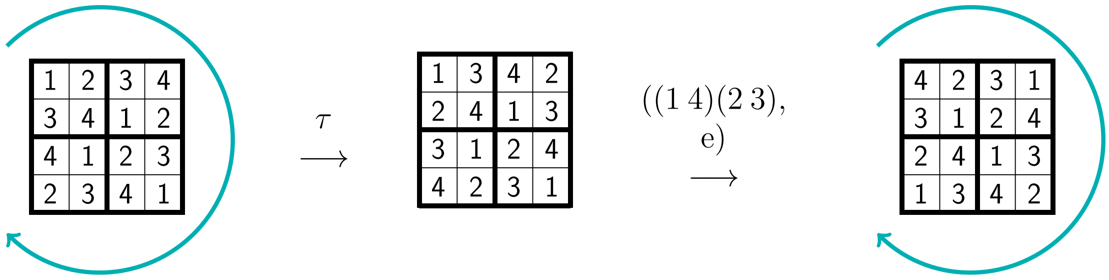
For example, a 90° clockwise rotation can be represented by the element $(e, (14)(23))\tau$, where this notation applies τ and then $(e, (1\,4)(2\,3))$ to a grid.



A 180° clockwise rotation does not require τ, it is simply $((1\,4)(2\,3), (1\,4)(2\,3))$.



For a 270° clockwise rotation, we find it is represented by $((1\,4)(2\,3), e)\tau$.



I have displayed these calculations as an example on $\mathbb{G}_7$, but they are accurate for any other Shidoku grid. Hence, by introducing just one element, τ, we are now able to perform all reflections and rotations. So we need a way to include τ in our group of operations.

Remark. *The set containing τ and the identity element is a group.*

Proof. Let $C_2 = \{e, \tau\}$. We know C_2 is non-empty and contains an identity element e. The operation is associative because for any $a, b, c \in C_2$, $a(bc) = (ab)c$. Every element in C_2 also has an inverse, since $e \cdot e = e$ and $\tau \cdot \tau = e$. Therefore, by definition, C_2 is a group. $\square$

However, forming a group with C_2 and $D_4 \times D_4$ is more complicated than simply forming a direct product between them. We can tell that τ is not commutative with all elements of $D_4 \times D_4$, because τ interchanges the rows and columns of a Shidoku grid. For any $g, h \in D_4$, $(g, h)\tau = \tau(h, g)$. Which is to say, the row permutation, g, becomes the column permutation, and the column permutation, h, becomes the row permutation. Thus the groups do not commute, and so we cannot form a direct product. We can instead use the semidirect product, a generalisation of a direct product that allows us to use two groups to form a new group. Let C_2 and $D_4 \times D_4$ form a semidirect product as so:

$$(D_4 \times D_4) \rtimes C_2.$$

This is actually something called a wreath product, and shows up in other areas of Group Theory. Finally we have formed a group that represents row permutations, column permutations, reflections, and rotations on Shidoku grids. The only operations left to consider are relabellings.

4.3.3 Relabelling

The operations that relabel Shidoku grids are not already included in $(D_4 \times D_4) \rtimes C_2$. However, we can easily represent all relabellings as the symmetric group S_4 where S_4 is the group of all permutations of the numbers 1 to 4. The $4! = 24$ different permutations of $\{1, 2, 3, 4\}$ represent the 24 different ways of relabelling Shidoku grids. Since S_4 is a well-established group, we know that:

$$((D_4 \times D_4) \rtimes C_2) \times S_4$$

contains all of the operations in our previous group combined with all possible ways of relabelling grids. Therefore, this is our group of all possible operations that produce valid Shidoku grids.

Now that we have formed our group of operations, we can continue to the next steps necessary to apply Burnside's lemma, which is to count the Shidoku grids fixed by our operations.

4.4 Counting fixed points

To use Burnside's lemma, we need to count the number of fixed Shidoku grids for each operation in $((D_4 \times D_4) \rtimes C_2) \times S_4$. We know D_4 contains 8 elements, as we have listed them, C_2 only contains 2 elements, and S_4 contains $4! = 24$ total elements. Then the total number of elements in our group is

$$\begin{aligned}|((D_4 \times D_4) \rtimes C_2) \times S_4| &= ((|D_4| \cdot |D_4|) \cdot |C_2|) \cdot |S_4| \\ &= ((8 \cdot 8) \cdot 2) \cdot 24 = (64 \cdot 2) \cdot 24 \\ &= 128 \cdot 24 = 3072.\end{aligned}$$

However, applying 3072 operations to every 288 Shidoku grids in order to count how many grids are fixed, is computationally feasible but unnecessary. There are ways we can narrow down our calculations into much more manageable numbers, such as using conjugacy classes and mostly ignoring relabelling.

4.4.1 Conjugacy classes

Definition 4.10 (Conjugacy class)**.** Let G be a group and let $x \in G$. Then the *conjugacy class* of x, denoted x^G, is defined by

$$x^G = \{gxg^{-1} : g \in G\}.$$

There is a relation between conjugacy classes and fixed points that becomes very useful for us.

Corollary 4.11. *Let G be a group and let $g, h \in G$. If g and h are conjugate, then $|fix(g)| = |fix(h)|$ where fix(x) is the set of elements fixed by x.*

Proof. Let $g, h \in G$ be conjugate, then $g = khk^{-1}$ by definition. For any $a \in A$ where A is a group that G acts on,

$$g \cdot a = a \iff khk^{-1} \cdot a = a \iff h \cdot (k^{-1} \cdot a) = (k^{-1} \cdot a) \implies |\text{fix}(g)| = |\text{fix}(h)|.$$

That is, for every $a \in A$ that $g \in G$ fixes, where $g = khk^{-1}$, then $h \in G$ fixes $k^{-1} \cdot a$, and vice versa, so g and h have the same number of fixed points in A. $\square$

In simple terms, if elements are conjugate, then they fix the same number of points, where we recall that fixed means that the element remains the same after being acted on. So if two elements in $((D_4 \times D_4) \rtimes C_2) \times S_4$ are in the same conjugacy class then they fix the same number of Shidoku grids.

Since Burnside's lemma requires us to count fixed grids, finding conjugacy classes narrows down the number of calculations we have to do, as we know that the other elements in that same conjugacy class will also fix the same number of grids. We will start by finding the conjugacy classes in D_n, where $n \geq 3$. Then we can apply this to D_4.

Theorem 4.12. *For $n \geq 3$, the dihedral group D_n is as follows*

$$D_n = \{e, r, \ldots, r^{n-1}, s, rs, \ldots, r^{n-1}s\},$$

where $r^n = s^2 = e$ and $sr = r^{-1}s$. Then the conjugacy classes in D_n are the following.

1. *If n is odd,*
    - $\{e\}$
    - $\{r^i, r^{-i}\}$ *for each* $i < \frac{n}{2}$
    - $\{s, rs, \ldots, r^{n-1}s\}$.
2. *If n is even,*
    - $\{e\}$
    - $\{r^i, r^{-i}\}$ *for each* $i = 1, \ldots, \frac{1}{2}n - 1$
    - $\{r^{\frac{n}{2}}\}$
    - $\{r^i s : i \text{ odd}\}$
    - $\{r^i s : i \text{ even}\}$.

Proof. We represent rotations in D_n as elements of the form r^i and we represent reflections as elements of the form $r^j s$ for any $i, j \in \mathbb{N}$. For each $i \in \mathbb{N}$ we want to show that the conjugacy class of r^i in D_n is $\{r^i, r^{-i}\}$. The subgroup of rotations is abelian, so we only need to consider conjugation by reflections. Hence, let $r^j s \in D_n$ be any reflection in D_n. We can show that

$$\begin{aligned}(r^j s)r^i(r^j s)^{-1} &= r^j s r^i(s^{-1}r^{-j}) = r^j s r^i(s r^{-j}) = r^j s r^i(r^j s)\\ &= r^j r^{-i} s s r^{-j} = r^j r^{-i} s^2 r^{-j} = r^j r^{-i} r^{-j}\\ &= r^{-j}.\end{aligned}$$

By definition of conjugacy classes, this shows that $\{r^i, r^{-i}\}$ is a conjugacy class.

Next we want to show that the conjugacy class of $r^i s$ in D_n is $\{r^j : j \in \mathbb{Z}, j \equiv i \pmod 2\}$. First let $k \in \mathbb{N}$ so that $r^k \in D_n$. When we conjugate by rotation we find

$$(r^k)r^i s(r^k)^{-1} = r^k r^i s r^- k = r^{i+2k}s.$$

Letting k vary over $\mathbb{Z}$, we see that the set of conjugates by rotation is $\{r^j : j \in \mathbb{Z}, j \equiv i \pmod 2\}$. Now to conjugate by reflection

$$\begin{aligned}(r^k s)r^i s(r^k s)^{-1} &= r^k s r^i s s r^{-k} = r^k r^{-i} s s^2 r^{-k}\\ &= r^k r^{-i} r^k s = r^{2k-i}s\\ &= r^{i+2(k-i)}s \in \{r^j : j \in \mathbb{Z}, j \equiv i \pmod 2\}.\end{aligned}$$

Therefore, for odd n, D_n has conjugacy classes $\{e\}$, $\{r^i, r^{-i}\}$ for each $i < \frac{n}{2}$, and $\{s, rs, \ldots, r^{n-1}s\}$. For even n, D_n has conjugacy classes $\{e\}$, $\{r^i, r^{-i}\}$ for each $i = 1, \ldots, \frac{1}{2}n - 1$, $\{r^{\frac{n}{2}}\}$, $\{r^i s : i \text{ odd}\}$, and $\{r^i s : i \text{ even}\}$.

□

Hence, we know that D_4 has the following conjugacy classes, which we can write in any of the following ways:

- $\{e\}$
- $\{r, r^{-1}\} = \{r, r^3\} = \{(1\,3\,2\,4), (1\,4\,2\,3)\}$
- $\{r^2\} = \{(1\,2)(3\,4)\}$
- $\{rs, r^3s\} = \{sr^3, sr\} = \{(1\,2), (3\,4)\}$
- $\{s, r^2s\} = \{s, sr^2\} = \{(1\,3)(2\,4), (1\,4)(2\,3)\}$.

Instead of doing calculations for all 8 elements of D_4, we now only need to do calculations for the 5 conjugacy classes of D_4. Furthermore, this narrows down the $8 \cdot 8 = 64$ calculations for $D_4 \times D_4$, to $5 \cdot 5 = 25$ total calculations necessary for $D_4 \times D_4$.

Finding the number of conjugacy classes in $(D_4 \times D_4) \rtimes C_2$ is slightly more complex. There are several methods we can choose to employ, and I chose to use Python to brute-force perform calculations that would show which elements are conjugate. We find there are no conjugacy classes that contain elements in both $(D_4 \times D_4) \rtimes \{e\}$ and $(D_4 \times D_4) \rtimes \{\tau\}$, so we can look at these two cases separately.

The conjugacy classes in $(D_4 \times D_4) \rtimes \{e\}$ are similar, but not the same, as the conjugacy classes in $D_4 \times D_4$. This is because when calculating the conjugacy classes for an element $x \in D_4 \times D_4$, we are considering $\{gxg^{-1} : g \in D_4 \times D_4\}$ by definition, but now we need to consider, for $x \in (D_4 \times D_4) \rtimes \{e\}$, the conjugacy class $\{gxg^{-1} : g \in (D_4 \times D_4) \rtimes C_2\}$. We already know the number of conjugacy classes will be 25 or less, as we have found some elements conjugate in $D_4 \times D_4$ before. However, there may be some elements that are now also conjugate due to τ. The operation τ in $C_2 = \{e, \tau\}$ swaps the row and column operations applied, as explained earlier. So it should not be surprising that we find the elements $(r, c) \in (D_4 \times D_4) \rtimes \{e\}$ are now conjugate to the elements (c, r). My Python code, which is included at the end of this document, performs the calculations and observes this fact. Hence, the 25 conjugacy classes in $D_4 \times D_4$, becomes 15 conjugacy classes in $(D_4 \times D_4) \rtimes \{e\}$ for $(D_4 \times D_4) \rtimes C_2$.

Now, if we consider just $(D_4 \times D_4) \rtimes \{\tau\}$, we can again use Python code to brute-force calculate $\{gxg^{-1} : g \in (D_4 \times D_4) \rtimes C_2\}$ for any $x \in (D_4 \times D_4) \rtimes \{\tau\}$. My code for this is included in the last section of this project, which you can look at to observe the mathematics done. Using this code we find there are 5 conjugacy classes in all of $(D_4 \times D_4) \rtimes \{\tau\}$. They are listed below.

The first conjugacy class contains the elements:

- $(e, e)\tau$,
- $(x, y)\tau$ for $x, y \in \{(1\,3\,2\,4), (1\,4\,2\,3)\}$,
- $((1\,2)(3\,4), (1\,2)(3\,4))\tau$,

- $(x, y)\tau$ for $x, y \in \{(1\,3)(2\,4), (1\,4)(2\,3)\}$,
- $(x, y)\tau$ for $x, y \in \{(1\,2), (3\,4)\}$.

The second conjugacy class contains the elements:

- $(x, y)\tau$ for $x, y \in \{(1\,3\,2\,4), (1\,4\,2\,3)\}$,
- $(e, (1\,2)(3\,4))\tau$,
- $((1\,2)(3\,4), e)\tau$.

The third conjugacy class contains the elements:

- $(e, x)\tau$ for $x \in \{(1\,3\,2\,4), (1\,4\,2\,3)\}$,
- $(x, e)\tau$ for $x \in \{(1\,3\,2\,4), (1\,4\,2\,3)\}$,
- $(x, (1\,2)(3\,4))\tau$ for $x \in \{(1\,3\,2\,4), (1\,4\,2\,3)\}$,
- $((1\,2)(3\,4), x)\tau$ for $x \in \{(1\,3\,2\,4), (1\,4\,2\,3)\}$,
- $(x, y)\tau$ for $x \in \{(1\,3)(2\,4), (1\,4)(2\,3)\}$ and $y \in \{(1\,2), (3\,4)\}$,
- $(x, y)\tau$ for $x \in \{(1\,2), (3\,4)\}$ and $y \in \{(1\,3)(2\,4), (1\,4)(2\,3)\}$.

The fourth conjugacy class contains the elements:

- $(e, x)\tau$ for $x \in \{(1\,3)(2\,4), (1\,4)(2\,3)\}$,
- $(x, e)\tau$ for $x \in \{(1\,3)(2\,4), (1\,4)(2\,3)\}$,
- $((1\,2)(3\,4), x)\tau$ for $x \in \{(1\,3)(2\,4), (1\,4)(2\,3)\}$,
- $(x, (1\,2)(3\,4))\tau$ for $x \in \{(1\,3)(2\,4), (1\,4)(2\,3)\}$,
- $(x, y)\tau$ for $x \in \{(1\,3\,2\,4), (1\,4\,2\,3)\}$ and $y \in \{(1\,2), (3\,4)\}$,
- $(x, y)\tau$ for $x \in \{(1\,2), (3\,4)\}$ and $y \in \{(1\,3\,2\,4), (1\,4\,2\,3)\}$.

The fifth conjugacy class contains the elements:

- $(e, x)\tau$ for $x \in \{(1\,2), (3\,4)\}$,
- $(x, e)\tau$ for $x \in \{(1\,2), (3\,4)\}$,
- $(x, y)\tau$ for $x \in \{(1\,3\,2\,4), (1\,4\,2\,3)\}$ and $y \in \{(1\,3)(2\,4), (1\,4)(2\,3)\}$,
- $(x, y)\tau$ for $x \in \{(1\,3)(2\,4), (1\,4)(2\,3)\}$ and $y \in \{(1\,3\,2\,4), (1\,4\,2\,3)\}$,
- $((1\,2)(3\,4), x)\tau$ for $x \in \{(1\,2), (3\,4)\}$,
- $(x, (1\,2)(3\,4))\tau$ for $x \in \{(1\,2), (3\,4)\}$.

Therefore, because there are 15 conjugacy classes in $(D_4 \times D_4) \rtimes \{e\}$ and 5 conjugacy classes in $(D_4 \times D_4) \rtimes \{\tau\}$, there are a total of $15 + 5 = 20$ conjugacy classes in all of $(D_4 \times D_4) \rtimes C_2$. If we continue to ignore relabelling, which we look at later, we will only need to calculate the fixed points for 20 operations.

Using Python to brute-force calculate the conjugacy classes is a valid method that worked to let us find the answer, but Russell and Jarvis implemented a different method. They also used a computer to perform calculations, but they used a programming language called GAP that, with only a few lines of code, allowed them to construct the group of operations and find the conjugacy classes. We can also implement GAP to construct the group of Shidoku operations. We imported SageMath in Python to work with matrices, but SageMath also allows us to call on GAP functions. For the interest of the reader, I have implemented these GAP functions in Python and also included them in the Python code section at the end of this report. It finds that there are 20 conjugacy classes in $(D_4 \times D_4) \rtimes C_2$, just like our previous method did. Next we move onto how exactly we count fixed grids.

4.4.2 Fixed grids

We know all 288 total Shidoku grids can be relabelled into being one of the 12 original grids, the ones labelled $\mathbb{G}_1$ to $\mathbb{G}_{12}$. Therefore, the easiest way to narrow down the calculations from relabelling, is to use the 12 Shidoku grids, and calculate which of those grids remains fixed for our operations (once we have relabelled it back to being one of the grids in $\mathbb{G}_1$ to $\mathbb{G}_{12}$).

For example, we can apply the operation $((1\,3\,2\,4),(1\,2)(3\,4))\tau$ to $\mathbb{G}_6$ and relabel it to find it is equal to $\mathbb{G}_{10}$.

$$\mathbb{G}_6 = \begin{array}{|c|c|c|c|}\hline 1&2&4&3\\ \hline 3&4&1&2\\ \hline 2&1&3&4\\ \hline 4&3&2&1\\ \hline\end{array} \quad \xrightarrow{\substack{((1\,3\,2\,4),(1\,2)(3\,4))\\ \tau}} \quad \begin{array}{|c|c|c|c|}\hline 2&3&4&1\\ \hline 1&4&3&2\\ \hline 3&1&2&4\\ \hline 4&2&1&3\\ \hline\end{array} \quad \xrightarrow{\text{relabel}} \quad \begin{array}{|c|c|c|c|}\hline 1&2&4&3\\ \hline 3&4&2&1\\ \hline 2&3&1&4\\ \hline 4&1&3&2\\ \hline\end{array} = \mathbb{G}_{10}$$

Since it did not once again equal $\mathbb{G}_6$, we can say that $\mathbb{G}_6$ is not fixed for the operation $((1\,3\,2\,4),(1\,2)(3\,4))\tau$.

Therefore, we need to do 20 calculations on 12 grids and count which of those grids remains fixed. This can be done by hand, though it would be a labour-intensive process, so I elected to write some Python code to do the brute-force calculations for me. Again, you can find this code in the section at the end of the document.

The number of fixed grids in the 15 conjugacy classes in $(D_4 \times D_4) \rtimes \{e\}$ can most easily be observed as a table, showing the conjugacy classes in D_4 as the row and column operations. This table shows the number of fixed points for 25 elements, and we see the results are symmetric along the diagonal because these elements are conjugate, as mentioned earlier. The results my code produced show that, for $D_4 \times D_4$, we find the following number of fixed grids.

(row $\times$ column)	$\{e\}$	$\{(1\,3\,2\,4), (1\,4\,2\,3)\}$	$\{(1\,2)(3\,4)\}$	$\{(1\,3)(2\,4), (1\,4)(2\,3)\}$	$\{(3\,4), (1\,2)\}$
$\{e\}$	12	4	8	6	0
$\{(1\,3\,2\,4), (1\,4\,2\,3)\}$	4	0	0	2	0
$\{(1\,2)(3\,4)\}$	8	0	4	10	4
$\{(1\,3)(2\,4), (1\,4)(2\,3)\}$	6	2	10	8	2
$\{(3\,4), (1\,2)\}$	0	0	4	2	0

To count the number of total fixed grids this produces, we need to keep in mind that the 5 columns and rows are representing conjugacy classes, not single elements. So, for example, the number of fixed grids for $\{e\} \times \{(1\,3\,2\,4), (1\,4\,2\,3)\}$ says 4 on the table, but this represents 4 fixed points for $(e, (1\,3\,2\,4))$, and 4 fixed points for $(e, (1\,4\,2\,3))$, so is actually $(4) \cdot 2 = 8$ total fixed points. Thus the total number of fixed grids is

$$\begin{aligned}
\text{total fixed} &= 12 + (4) \cdot 2 + 8 + (6) \cdot 2 + 0 \\
&\quad + (4) \cdot 2 + 0 + 0 + (2) \cdot 4 + 0 \\
&\quad + 8 + 0 + 4 + (10) \cdot 2 + (4) \cdot 2 \\
&\quad + (6) \cdot 2 + (2) \cdot 4 + (10) \cdot 2 + (8) \cdot 4 + (2) \cdot 4 \\
&\quad + 0 + 0 + (4) \cdot 2 + (2) \cdot 4 + 0 \\
&= 12 + (10) \cdot 4 + (8) \cdot 6 + (6) \cdot 4 + (4) \cdot 9 + (2) \cdot 16 \\
&= 192.
\end{aligned}$$

For $(D_4 \times D_4) \rtimes \{\tau\}$ we know the first conjugacy class contains 10 elements, the second contains 6 elements, and the third, fourth and fifth conjugacy classes all contain 16 elements. The results produced by the code therefore showed the following.

1. The first conjugacy class contains 10 total elements. We found that each of these produces 2 fixed Shidoku grids, so there are $(2) \cdot 10 = 20$ fixed grids.
2. The second conjugacy class produces 2 fixed grids, then with 6 elements, this gives $(2) \cdot 6 = 12$.
3. The third conjugacy class produces 0 fixed Shidoku grids.
4. The fourth conjugacy class also produces 2 fixed grids, therefore there are $(2) \cdot 16 = 32$ total.
5. The fifth and final conjugacy class produces 0 fixed grids.

Then total fixed $= 20 + 12 + 32 = 64$.

Combining these results shows there are $192 + 64 = 256$ total fixed Shidoku grids for $(D_4 \times D_4) \rtimes C_2$ applied to $\mathbb{G}_1$ to $\mathbb{G}_{12}$.

4.5 Essentially different grids

We now have all the results required to apply Burnside's lemma and find the number of essentially different Shidoku grids. As a reminder, the equation from Burnside's lemma is:

$$|X/G| = \frac{1}{|G|}\sum_{g\in G}|X^g|,$$

where X is the set of Shidoku grids, G is $((D_4 \times D_4) \rtimes C_2) \times S_4$, $|X/G|$ is the number of essentially different grids, and $|X^g|$ is the set of fixed grids.

From earlier, we know that

$$|G| = |((D_4 \times D_4) \rtimes C_2) \times S_4| = 128 \cdot 24 = 3072$$

and

$$\sum_{g\in G}|X^g| = 256 \cdot 24 = 6144$$

since we found 256 fixed grids when only considering the 12 grids, and these grids can be relabelled into the $12 \cdot 24 = 288$ total grids. Hence there are $256 \cdot 24 = 6144$ total fixed Shidoku grids. Inserting these numbers into the equation gives us:

$$\text{essentially different grids} = \frac{256 \cdot 24}{128 \cdot 24} = \frac{6144}{3072} = 2.$$

Thus, there are 2 essentially different Shidoku grids. This is the same result we found when brute-force calculating essentially the same grids, though now we have done so using Group Theory. Despite this method taking longer for Shidoku grids, it is much more efficient for classic 9×9 Sudoku grids. Constructing the group of operations and counting the fixed Sudoku grids is exactly what Jarvis and Russell did when calculating the number of essentially different Sudoku. At the end of their report, they went on to consider different ways of defining essentially different. In the next section we will also consider an alternate definition.

5 What similar questions can we answer?

We have found that there are 288 total Shidoku grids, and only 2 of which are essentially different. However, we could decide to define *essentially different* in a different way. For example, if we were to consider only row and column permutations and relabelling, while excluding reflections and rotations, we'd find a different number of essentially different Shidoku grids.

Definition 5.1 (Essentially different, version 2)**.** We could consider grids *essentially the same* if there is a sequence of the operations (1) and (3) which transforms one of the grids into another. These operations are:

1. row and column permutations,
2. ~~reflections and rotations, and~~
3. relabelling.

If grids are not essentially the same, we call them *essentially different.*

By our previous calculations, we found there are 192 fixed Shidoku grids in $D_4 \times D_4$ when not including the 64 from τ. So we can apply Burnside's lemma as so,

$$|X/G| = \frac{1}{|G|}\sum_{g\in G}|X^g| = \frac{\sum_{g\in G}|X^g|}{|(D_4 \times D_4) \times S_4|} = \frac{192\cdot 24}{64\cdot 24} = \frac{4608}{1536} = 3$$

and find there are 3 essentially different Shidoku grids when we do not include reflections and rotations. We would come to the same conclusion if we brute-force worked out essentially the same grids until we narrowed it down to 3 essentially different grids.

6 Conclusion

We have briefly looked at the history of the popular number puzzle, Sudoku, and the mathematical research done into it in recent years. There is far more history to Latin Squares and related puzzles that we could choose to explore. The research done into Sudoku specifically also goes into many different areas of mathematics that we did not focus on in this report.

What we did choose to focus on was Group Theory and two specific questions: how many 4×4 Sudoku (also known as Shidoku) grids are there, and of those grids, which are considered essentially different? Using some brute-force calculations we found there are 288 total Shidoku grids, and only 2 essentially different ones. Though there could be more efficient and meticulous ways of brute-force calculating these grids, we played around with calculations until we were confident in our answers.

In an effort to answer our question along the same lines of mathematics as Russell and Jarvis did before, we introduced Burnside's lemma, and formed the group of operations $((D_4 \times D_4) \rtimes C_2) \times S_4$. By finding conjugacy classes, we simplified our calculations and counted fixed grids to again find there are only 2 essentially different Shidoku. The Python and SageMath code used matrices to represent Shidoku grids and brute-force transformed and calculated fixed grids. We briefly used GAP to count conjugacy classes, but for someone more literate in it, it may be more efficient to utilise GAP for counting fixed points as well.

Finally, we briefly looked at using different operations in our definition of essentially different. We found that using only row and column permutations along with relabelling created 3 essentially different Shidoku. Something we did not consider, that would also be interesting, is what the number of essentially different grids are when using only reflections, rotations and relabelling, excluding the other row and column permutations.

By working with Shidoku, we were able to use Group Theory to understand what other researchers did previously, but in future, it may be interesting to try the same mathematics with classic 9×9 Sudoku as well.

7 References

[1] Colbourn, C.J. and Dinitz, J.H. (2006). *Handbook of Combinatorial Designs.* 2nd ed. CRC Press, p.12. ISBN: 9781420010541. Available at: https://books.google.com/books?id=Q9jLBQAAQBAJ&pg=PA12 [Accessed 26 Feb. 2025].

[2] Cook, S., Fujimoto, C., Mingle, L. and Sawyer, C. (2007). Sudoku: Just for Fun or Is It Mathematics? *Math Horizons*, [online] 14(3), pp.13–15. Available at: https://www.jstor.org/stable/25678670 [Accessed 26 Feb. 2025].

[3] Sharp, J. (2006). International Perspectives: Beyond Su Doku. *Mathematics Teaching in the Middle School*, [online] 12(3), pp.165–169. Available at: https://www.jstor.org/stable/41182906 [Accessed 26 Feb. 2025].

[4] Pedersen, R.M. and Vis, T.L. (2009). Sets of Mutually Orthogonal Sudoku Latin Squares. *The College Mathematics Journal*, [online] 40(3), pp.174–181. doi:https://doi.org/10.1080/07468342.2009.11922356.

[5] McGuire, G., Tugemann, B. and Civario, G. (2014). There Is No 16-Clue Sudoku: Solving the Sudoku Minimum Number of Clues Problem via Hitting Set Enumeration. *Experimental Mathematics*, 23(2), pp.190–217. doi:https://doi.org/10.1080/10586458.2013.870056.

[6] Sasse, D. (2021). *Generating & Solving Sudoku Puzzles.* [online] Medium. Available at: https://dsasse07.medium.com/generating-solving-sudoku-puzzles-9ee1305ced01 [Accessed 11 Mar. 2025].

[7] Felgenhauer, B. and Jarvis, F. (2006). Mathematics of Sudoku I.

[8] Russell, E. and Jarvis, F. (2006). Mathematics of Sudoku II.

[9] Tointon, M. (2024). Group Theory (MATH33300) University of Bristol.

[10] Koswara, I., Corn, P., Khan, S., Khim, J. and Ross, E. (n.d.). *Burnside's Lemma.* [online] Brilliant.org. Available at: https://brilliant.org/wiki/burnsides-lemma/ [Accessed 1 Mar. 2025].

[11] Rosenhouse, J. and Taalman, L. (2011). *Taking Sudoku Seriously : The Math Behind the World's Most Popular Pencil Puzzle.* Oxford ; New York: Oxford University Press.

[12] Arcos, C., Brookfield, G. and Krebs, M. (2010). Mini-Sudokus and Groups. *Mathematics Magazine*, 83(2), pp.111–122. doi:https://doi.org/10.4169/002557010x482871.

8 Python code

8.1 Finding conjugacy classes

8.1.1 Using brute-force

Implementing Python and SageMath functions to brute-force calculate the conjugacy classes with matrices.

```python
from sage.all import *

# List only non-conjugate operations
ops = [
    (0, 1, 2, 3),       # e (identity)
    (3, 2, 0, 1),       # (1324)
    (1, 0, 3, 2),       # (12)(34)
    (2, 3, 0, 1),       # (13)(24)
    (0, 1, 3, 2),       # (34)
]

# List the inverses of the operations
rev_ops = [
    (0, 1, 2, 3),       # e is self-inverse
    (2, 3, 1, 0),       # (1423) is the inverse of (1324)
    (1, 0, 3, 2),       # (12)(34) is self-inverse
    (2, 3, 0, 1),       # (13)(24) is self-inverse
    (0, 1, 3, 2),       # (34) is self-inverse
]

# Defining a function to perform row operations
def row_operations(M, num):
    """
    Given a 4x4 matrix and a row operation,
    performs the row operation and returns the
    transformed matrix.
    """
    mat = matrix([M[num[0]],M[num[1]],M[num[2]],M[num[3]]]) # Swap rows
    return mat # Return transformed grid

# Defining a function to perform column operations
def column_operations(M, num):
    """
    Given a 4x4 matrix and a column operation,
    performs the column operation and returns the
    transformed matrix.
    """
    col1 = M.column(num[0]) # Find the current columns
    col2 = M.column(num[1])
    col3 = M.column(num[2])
    col4 = M.column(num[3])
    row1 = [col1[0], col2[0], col3[0], col4[0]] # Construct new rows
    row2 = [col1[1], col2[1], col3[1], col4[1]]
    row3 = [col1[2], col2[2], col3[2], col4[2]]
    row4 = [col1[3], col2[3], col3[3], col4[3]]
    mat = matrix([row1, row2, row3, row4]) # Construct the new matrix
    return mat # Return transformed grid
```

```python

# Write a function to reflect a grid by its diagonal, i.e. tau
def diagonalise(M):
    """
    Given a 4x4 matrix, reflects the grid along the
    diagonal from top left to bottom right.
    """
    mat = matrix(4) # Generate an empty 4x4 matrix
    for r in range(0,4): # Iterate through rows and columns
        for c in range(0,4):
            mat[r,c] = M[c,r] # Fill each number in the matrix
    return mat # Return transformed grid

# Create a 4x4 matrix with a different number in each box
# Then we can tell which row and column operations transform a grid the same way
test_mat = matrix([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]])

# Create a function to find conjugate operations
# Remember if there exists an g such that x = gxg^{-1}
# Then x is conjugate to that gxg^{-1}
def find_conjugates(row_op, col_op, tau = False):
    """
    Given a row and column operation and whether to use Tau or not,
    returns a list of all other row and column operations that
    the operation is conjugate to
    """
    x = []
    for row in ops: # Iterate through row and col ops
        for col in ops:
            conjugate = False # Set conjugate to false
            Mtest = test_mat
            if tau == True:
                Mtest = diagonalise(Mtest) # Apply tau if set to true
            Mtest = row_operations(Mtest, row) # Calculate operation on test matrix
            Mtest = column_operations(Mtest, col)
            for r in range (0, 5): # Iterate through row and col ops
                for c in range(0, 5):
                    for d in range(0, 2): # Iterate through using tau to calculate g
                        M = test_mat
                        if d == 1:
                            M = diagonalise(M) # Calculate g
                        M = row_operations(M, ops[r])
                        M = column_operations(M, ops[c])
                        if tau == True:
                            M = diagonalise(M) # Calculate x (use tau if set to True)
                        M = row_operations(M, row_op)
                        M = column_operations(M, col_op)
                        M = row_operations(M, rev_ops[r]) # Calc g^{-1}
                        M = column_operations(M, rev_ops[c])
                        if d == 1:
                            M = diagonalise(M)
                        if M == Mtest: # Check if gxg^{-1} is equal to the operation being checked
                            conjugate = True # If there exists a g such that they are equal, set conjugate to True
            if conjugate == True and tau == True:
                x.append([row, col, "tau"]) # If they are conjugate, then add it to list (and show we used tau)
            elif conjugate == True:
                x.append([row, col]) # If they are conjugate, then add it to list
    return x

# Observe that the (r, c) is conjugate to (c, r)
# By applying the find_conjugates function to a few examples
print("((0, 1, 2, 3), (0, 1, 2, 3)) is conjugate to", find_conjugates((0, 1, 2, 3),(0, 1, 2, 3)), "\n")
print("((0, 1, 2, 3), (3, 2, 0, 1)) is conjugate to", find_conjugates((0, 1, 2, 3), (3, 2, 0, 1)), "\n")
print("((0, 1, 2, 3), (1, 0, 3, 2)) is conjugate to", find_conjugates((0, 1, 2, 3), (1, 0, 3, 2)), "\n")
print("((0, 1, 2, 3), (2, 3, 0, 1)) is conjugate to", find_conjugates((0, 1, 2, 3), (2, 3, 0, 1)), "\n")
print("((0, 1, 2, 3), (0, 1, 3, 2)) is conjugate to", find_conjugates((0, 1, 2, 3), (0, 1, 3, 2)), "\n")

# Find all 5 conjugacy classes when we apply tau
# By using the find_conjugates function and observing all elements are within one of those conjugacy classes
print("((0, 1, 2, 3), (0, 1, 2, 3), tau) is conjugate to", find_conjugates((0, 1, 2, 3), (0, 1, 2, 3), True), "\n")
print("((3, 2, 0, 1), (3, 2, 0, 1), tau) is conjugate to", find_conjugates((3, 2, 0, 1), (3, 2, 0, 1), True), "\n")
print("((2, 3, 0, 1), (0, 1, 3, 2), tau) is conjugate to", find_conjugates((2, 3, 0, 1), (0, 1, 3, 2), True), "\n")
print("((3, 2, 0, 1), (0, 1, 3, 2), tau) is conjugate to", find_conjugates((3, 2, 0, 1), (0, 1, 3, 2), True), "\n")
print("((0, 1, 3, 2), (2, 3, 0, 1), tau) is conjugate to", find_conjugates((0, 1, 3, 2), (2, 3, 0, 1), True), "\n")
```

8.1.2 Using GAP

Implementing GAP functions to construct the group and find the conjugacy classes. We could use further GAP functions to print the elements in each conjugacy class and then use this to count fixed grids.

```
from sage.all import *

C2 = gap.SymmetricGroup(2) # Construct the C_2 group
D4 = PermutationGroup([[(1,2)],[(3,4)],[(1,3),(2,4)]]) # Construct the D_4 group

# Construct (D_4 x D_4) semidirectproduct C_2
# This is something called a wreath product, so we can construct it as so
G = gap.StandardWreathProduct(D4,C2)
```

```

# The set of all conjugacy classes is then
cc = gap.ConjugacyClasses(G)

# Print the number of conjugacy classes

print("There are", len(cc), "conjugacy classes in (D_4 x D_4)
    semidirectproduct C_2")
```

8.2 Counting fixed grids

Implementing Python and SageMath to construct grids as matrices. Then using functions to perform row and column operations, τ, and relabelling on those matrices, in order to count fixed grids.

```
from sage.all import *

# Represent all 12 Shidoku grids as 4x4 matrices
G1 = matrix([[1,2,3,4],[3,4,1,2],[2,1,4,3],[4,3,2,1]])
G2 = matrix([[1,2,4,3],[3,4,2,1],[2,1,3,4],[4,3,1,2]])
G3 = matrix([[1,2,4,3],[3,4,2,1],[4,3,1,2],[2,1,3,4]])
G4 = matrix([[1,2,3,4],[3,4,1,2],[4,3,2,1],[2,1,4,3]])
G5 = matrix([[1,2,3,4],[3,4,2,1],[2,1,4,3],[4,3,1,2]])
G6 = matrix([[1,2,4,3],[3,4,1,2],[2,1,3,4],[4,3,2,1]])
G7 = matrix([[1,2,4,3],[3,4,1,2],[4,3,2,1],[2,1,3,4]])
G8 = matrix([[1,2,3,4],[3,4,2,1],[4,3,1,2],[2,1,4,3]])
G9 = matrix([[1,2,3,4],[3,4,1,2],[2,3,4,1],[4,1,2,3]])
G10 = matrix([[1,2,4,3],[3,4,2,1],[2,3,1,4],[4,1,3,2]])
G11 = matrix([[1,2,4,3],[3,4,2,1],[4,1,3,2],[2,3,1,4]])
G12 = matrix([[1,2,3,4],[3,4,1,2],[4,1,2,3],[2,3,4,1]])

# Put the grids in a list so we can iterate through them later
matrices_list = [G1, G2, G3, G4, G5, G6, G7, G8, G9, G10, G11, G12]

# Write list of all row and column operations
all_ops = [
    (0, 1, 2, 3),       # e (identity)
    (3, 2, 0, 1),       # (1324)
    (1, 0, 3, 2),       # (12)(34)
    (2, 3, 1, 0),       # (1423)
    (2, 3, 0, 1),       # (13)(24)
    (0, 1, 3, 2),       # (34)
    (3, 2, 1, 0),       # (14)(23)
    (1, 0, 2, 3)        # (12)
]

# Narrow list down into only non-conjugate operations
ops = [
    (0, 1, 2, 3),       # e (identity)
    (3, 2, 0, 1),       # (1324)
    (1, 0, 3, 2),       # (12)(34)
    (2, 3, 0, 1),       # (13)(24)
    (0, 1, 3, 2),       # (34)
]

```

```python
# Defining a function to perform row operations
def row_operations(M, num):
    """
    Given a 4x4 matrix and a row operation,
    performs the row operation and returns the
    transformed matrix.
    """
    mat = matrix([M[num[0]],M[num[1]],M[num[2]],M[num[3]]]) # Swap rows
    return mat # Return transformed grid

# Defining a function to perform column operations
def column_operations(M, num):
    """
    Given a 4x4 matrix and a column operation,
    performs the column operation and returns the
    transformed matrix.
    """
    col1 = M.column(num[0]) # Find the current columns
    col2 = M.column(num[1])
    col3 = M.column(num[2])
    col4 = M.column(num[3])
    row1 = [col1[0], col2[0], col3[0], col4[0]] # Construct new rows
    row2 = [col1[1], col2[1], col3[1], col4[1]]
    row3 = [col1[2], col2[2], col3[2], col4[2]]
    row4 = [col1[3], col2[3], col3[3], col4[3]]
    mat = matrix([row1, row2, row3, row4]) # Construct the new matrix
    return mat # Return transformed grid

# Write a function to reflect a grid by its diagonal, i.e. tau
def diagonalise(M):
    """
    Given a 4x4 matrix, reflects the grid along the
    diagonal from top left to bottom right.
    """
    mat = matrix(4) # Generate an empty 4x4 matrix
    for r in range(0,4): # Iterate through rows and columns
        for c in range(0,4):
            mat[r,c] = M[c,r] # Fill each number in the matrix
    return mat # Return transformed grid

# Defining a function to relabel a Shidoku grid
def relabel(M):
    """
    Given a 4x4 matrix, formatted as a Shidoku grid,
    relabels the matrix ao the top left square
    is the numbers 1 to 4 in order.
    """
    mat = matrix(4) # Generate an empty 4x4 matrix
    for a in range(0,2): # Iterate through the placement
        for b in range(0,2):
            for j in range(1,5): # Iterate through numbers 1-4
                if M[a,b] == j:
                    for r in range(0,4): # Go through each item
                        for c in range(0,4): # and replace the number
                            if M[r,c] == j and a == 0 and b == 0:
                                mat[r,c] = 1
                            elif M[r,c] == j and a == 0 and b == 1:
                                mat[r,c] = 2
                            elif M[r,c] == j and a == 1 and b == 0:
                                mat[r,c] = 3
                            elif M[r,c] == j and a == 1 and b == 1:
                                mat[r,c] = 4
    return mat # Return transformed grid

# Write a function to find the number of fixed grids
def fixed_points(row_op, col_op, x = False):
    """
    Given a row and column operation, counts the number of
    grids fixed by those operations. If x is set to true it
    also applies tau and reflects the grids along diagonal.
    """
    num_of_fixed = 0 # Set the original number of fixed grids as 0
    for M in matrices_list: # Iterate through all 12 matrices
        mat = M
        if x == True: # Apply tau if x is set to True
            mat = diagonalise(mat)
        mat = row_operations(mat, row_op) # Apply row operation
        mat = column_operations(mat, col_op) # Apply column operation
        mat = relabel(mat) # Relabel grid
        if mat == M:
            num_of_fixed += 1 # Add one count to number of fixed grids
    return num_of_fixed # Return the number of fixed grids

# In order to count fixed grids for conjugacy classes in D_4 x D_4:

fixed_table_D4xD4 = matrix(5) # Set an empty 5 by 5 matrix to hold results

for n in range(0, 5): # Iterate through each place in the matrix
    for m in range(0, 5):
        # Call on the fixed_points function and count the number of
        # fixed matrices for each row and column operation
        fixed_table_D4xD4[n,m] = fixed_points(ops[n], ops[m])

# Print the table with results
print("Table of fixed grids in D_4 x D_4:\n" + str(fixed_table_D4xD4))

# Also find fixed grids for each conjugacy class with tau
# Print the results
print("The first conjugacy class with tau has", fixed_points(ops[0], ops[0], True), "fixed grids")
print("The second conjugacy class with tau has", fixed_points(ops[1], ops[1], True), "fixed grids")
print("The third conjugacy class with tau has", fixed_points(ops[0], ops[1], True), "fixed grids")
print("The fourth conjugacy class with tau has", fixed_points(ops[0], ops[3], True), "fixed grids")
print("The fifth conjugacy class with tau has", fixed_points(ops[0], ops[4], True), "fixed grids")
```