Triplets

A dialectological study into the Geometry of Sudoku

Work in progress – comments & suggestions welcome!

Alex Pfaff
nonintersective@gmail.com

Draft: September 12, 2026

Contents

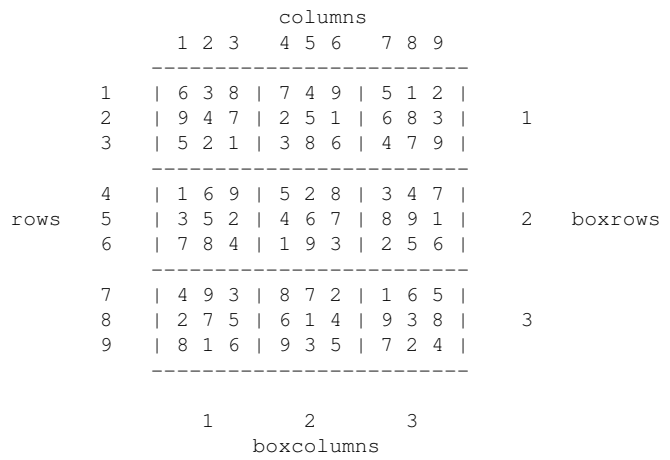
Abstract

This study is part of my ongoing project *The Syntax of Sudoku* . . . .

0 Prologue

This is a study into certain geometric properties of Sudoku grids involving permutations, construction and enumeration of possible grids, as well as identifying impossible grids. A **grid** is a 9×9 matrix filled with the digits $1-9$ according to the rules of Sudoku: no digit may repeat in any *row*, *column* or 3×3 *box*. By convention, numbering starts in the top left corner; we will additionally make use of the substructures *boxcolumn* and *boxrow*:[1]

(1)



The notion *empirical/experimental mathematics* may not ring a bell nor sound appealing to some readers, but it is an apt description of things to come. For a large part, we will not offer rigid proofs, but instead motivate procedures, computations, and algorithms that lead to certain numbers or objects, occasionally resorting to random data and brute force strategies (see Sect. 0.1). Conversely, where we actually do offer proofs, those will contain more prose than many a mathematician can master (sorry). The underlying logic will be just as rigid nonetheless.

0.1 Empirical vs. np-rical data

Every (empirical) science runs on data — data obtained from experiments, observations, measurements, field work, and the like. Sudokology is a special case in this respect, because its domain of study is finite and its objects are 100% subject to deterministic rules: there is no fuzziness, no measurement imprecision. The objects of interest are, of course, **grids**, **classes/families of grids**, as well as sub-structures of grids such as **boxes, rows, boxcolumns** . . . and as we shall see: **triplets**.

Mathematical data is ideally obtained by following rigid mathematical/algorithmic procedures. Sudoku, however, being an NP-complete problem poses a number of, well, problems, and many well-behaved algorithms and neat solutions to those problems are

[1] A terminological note: there are other terms on the market such as *stack, band* and whatnot, but we will use the more transparent terms *boxcolumn, boxrow* when referring to the vertical or horizontal sequence of three boxes; e.g. boxcolumn 2 $\Leftrightarrow$ boxes 2, 5, 8 $\Leftrightarrow$ columns 4, 5, 6.

still taxpayers in that undiscovered country from whose bounds no traveller returns with no intention of migrating to a more tangible realm. Sudoku grids themselves are not usually observed in nature, and it's fair to say that launching questionnaires about grids or blasting grids through the particle accelerator at CERN will be of little help, so we need to get our data elsewhere.

empirical data. One approach is to create a (large) sample of randomly generated *valid* Sudoku grids, to analyse and classify them, and to hope that some generalisations emerge. This approach is closely related to the notion of *empirical probability* in the field of stochastics. A textbook example is flipping a coin: since a (fair) coin has two sides — head and tail — both with equal chances of showing after a toss, the hardwired probability for either outcome is 50%. In a real-life experiment, however, we may find that after n tosses, heads appears in only 42% of the cases. But the expectation is that, as n increases, the observed probabilities will converge towards the hardwired ones.[2]

For Sudoku, matters are more complicated. In many cases, we do not even know the hardwired probabilities in advance — the combinatorics of Sudoku is rather more sophisticated than that of coin tossing — but the hope is that sufficiently large samples will nevertheless yield empirically meaningful patterns and hints. We will use this strategy in order to present "the big picture" highlighting high-level generalizations, rough proportions and orders of magnitude, without aiming at 100% accuracy or completeness or even a balanced dataset. For our purposes, this is good enough because the data are not meant to be fully representative, and they are not used for statistical analysis proper.

`np`-rical data. As an alternative, we can generate datasets by fully deterministic algorithms that do not rely on randomness and can be expected to produce exhaustive results. They may, however, rely on semi–brute-force strategies and deliberately overgenerate, followed by the filtering of redundant data.[3] This description applies especially to the operation **abc-reduction**, which potentially filters out redundant grids (see Sect. 0.2).

The relevant tools stem from the Python library `PseudoQ`, which, in turn, relies heavily on the library `numpy`; the latter has the conventional import alias **`np`** – hence the designation *`np`-rical*. In Sect. 0.3, we will briefly illustrate some functionalities of `PseudoQ` that will be relevant for the discussion to come.

0.2 Labeling Permutations: abc-Encoding and abc-Reduction

At what level are Sudoku grids identical or distinct? We may have a collection of grids, compare them cell by cell (top-down, left-right, row-wise), and find that they represent distinct sequences, for instance:

[2] This idea is also known as *law of large numbers*. Notice that it is more a plausibility, rather than a safe bet; it is certainly not a law in the strict sense and we are not talking about asymptotic behaviour.

[3] An elegant algorithm would ideally not produce undesirable data in the first place, but this is also an exploratory article and we will thus tolerate this sort of redundancy for the moment.

(5, 7, 8, 4, 2, 9, 3, 1, 6, 9, 6, 2, 3, 5, 1, 7, 4, 8, 3, 4, 1, 7, 6 . . . .)

(2, 7, 8, 4, 5, 9, 6, 1, 3, 9, 3, 5, 6, 2, 1, 7, 4, 8, 6, 4, 1, 7, 3 . . . .)

Yet, this kind of procedure runs the risk of missing relevant generalizations. Consider the following four grids:

(2)

(a)

```
-------------------------
| 1 2 3 | 4 5 6 | 7 8 9 |
| 4 5 6 | 7 8 9 | 2 3 1 |
| 7 8 9 | 2 3 1 | 4 5 6 |
-------------------------
| 2 3 1 | 5 6 4 | 8 9 7 |
| 5 6 4 | 9 7 8 | 3 1 2 |
| 8 9 7 | 3 1 2 | 5 6 4 |
-------------------------
| 3 1 2 | 6 4 5 | 9 7 8 |
| 6 4 5 | 8 9 7 | 1 2 3 |
| 9 7 8 | 1 2 3 | 6 4 5 |
-------------------------
```

(b)

```
-------------------------
| 3 2 1 | 5 6 4 | 8 7 9 |
| 5 6 4 | 8 7 9 | 2 1 3 |
| 8 7 9 | 2 1 3 | 5 6 4 |
-------------------------
| 2 1 3 | 6 4 5 | 7 9 8 |
| 6 4 5 | 9 8 7 | 1 3 2 |
| 7 9 8 | 1 3 2 | 6 4 5 |
-------------------------
| 1 3 2 | 4 5 6 | 9 8 7 |
| 4 5 6 | 7 9 8 | 3 2 1 |
| 9 8 7 | 3 2 1 | 4 5 6 |
-------------------------
```

(c)

```
-------------------------
| 9 8 7 | 6 5 4 | 3 2 1 |
| 6 5 4 | 3 2 1 | 8 7 9 |
| 3 2 1 | 8 7 9 | 6 5 4 |
-------------------------
| 8 7 9 | 5 4 6 | 2 1 3 |
| 5 4 6 | 1 3 2 | 7 9 8 |
| 2 1 3 | 7 9 8 | 5 4 6 |
-------------------------
| 7 9 8 | 4 6 5 | 1 3 2 |
| 4 6 5 | 2 1 3 | 9 8 7 |
| 1 3 2 | 9 8 7 | 4 6 5 |
-------------------------
```

(d)

```
-------------------------
| 7 4 1 | 5 2 8 | 9 6 3 |
| 5 2 8 | 9 6 3 | 4 1 7 |
| 9 6 3 | 4 1 7 | 5 2 8 |
-------------------------
| 4 1 7 | 2 8 5 | 6 3 9 |
| 2 8 5 | 3 9 6 | 1 7 4 |
| 6 3 9 | 1 7 4 | 2 8 5 |
-------------------------
| 1 7 4 | 8 5 2 | 3 9 6 |
| 8 5 2 | 6 3 9 | 7 4 1 |
| 3 9 6 | 7 4 1 | 8 5 2 |
-------------------------
```

Anticipating some terminology, we observe that the first three of those grids involve the *triplets* $\langle 1, 2, 3\rangle$, $\langle 4, 5, 6\rangle$, $\langle 7, 8, 9\rangle$, whereas the fourth comprises completely different triplets, $\langle 1, 4, 7\rangle$, $\langle 2, 5, 8\rangle$, $\langle 3, 6, 9\rangle$. Cellwise comparison reveals that these are four distinct grids.

However, all these grids (and many more) are identical at a deeper (= structural or distributional) level; they all translate to the same **abc-grid**:

(3)

```
-----------------------
| A B C | D E F | G H I |
| D E F | G H I | B C A |
| G H I | B C A | D E F |
-----------------------
| B C A | E F D | H I G |
| E F D | I G H | C A B |
| H I G | C A B | E F D |
-----------------------
| C A B | F D E | I G H |
| F D E | H I G | A B C |
| I G H | A B C | F D E |
-----------------------
```

Abc-grids are a special kind of encoding, where an *abc-grid* encodes the structural distribution independently of symbol choice. Encoding proceeds in two steps: first, a key is extracted by mapping the top row of a given grid, e.g. (2a) to the fixed alphabetic sequence $ABCDEFGHI$:

encoder key:

from	1	2	3	4	5	6	7	8	9	top row of a given grid
to	A	B	C	D	E	F	G	H	I	fixed sequence

and secondly, the digits in the entire grid are substituted by letters according to this key. The resulting abc-grid encodes a deep identity in terms of structure/distribution.

Recoding, conversely, creates a key by mapping the alphabetic sequence to some ordering of the digits $1-9$; take as example the top row of (2d):

recoder key:

from	A	B	C	D	E	F	G	H	I	fixed sequence
to	7	4	1	5	2	8	9	6	3	some ordering of digits $1-9$

and then substitutes the letters accordingly. Since there are $9! = 362{,}880$ such orderings ($\Leftrightarrow$ valid Sudoku rows), each abc-grid can be recoded in 362,880 ways.

More specifically, this means **one abc-grid translates to 362,880 distinct digit grids**; these digit grids are still distributionally identical, and essentially *are* the same grid; the distinctness is superficial. So when examining structural properties of such a collection of grids, it is actually sufficient to look at one representative. In principle, this representative could be any one out of 362,880 digit grids, but in order to unambiguosly make the intention clear we will usually talk about the corresponding abc-grid as representative. An abc-grid can be regarded as the ID of the grid family. We will keep talking about ***abc*** *-grids, -encoding, -reduction* etc., but in practice we will recode grids by the standardized key 123456789, simply because numeric values can be processed more efficiently than strings.

Other than the neat geometric insights – if we have large collections of grids that may or may not contain grids from distinct families, this notion is quite powerful because it allows us to only consider distinct abc-grids. In practice, this means that we

recode the entire grid collection by the same key, and then dismiss duplicates; this **potentially reduces the search space / runtime by a factor of up to 362,880**. Crucially, our "virtual" abc encoding serves as a standardization.[4][5]

0.3 PseudoQ – Methodology

This subsection briefly introduces some functionalities of the Python library `PseudoQ`[6]; it is not intended as a thorough tutorial, but rather to make (aspects of) the methodology more transparent. On several occasions we will employ phrases like: *operation X leads to result Y* or *after procedure Z, there are k grids left*, without showing the actual calculation, i.e. seemingly without motivation. In those cases, the actual computations are performed behind the scene; we are merely reporting the results, and, for the sake of space, we refrain from reproducing the entire code each time.

Class `Grid`. This class handles individual grids: initialize grids from a given input or generate a (valid) grid at random, check for validity, manipulate or transform grid structure etc. In the following example, we generate a grid, retrieve the unique triplets it contains and count them:

(4)
```
>>> from PsQ_Grid import Grid
>>> g = Grid()
>>> g.generate_rndGrid(seed=42)
>>> triplets = g.get_uniqueTriplets()
>>> n = len(triplets)
>>> print(n)
    17
```

One typical example of a brute force strategy to obtain raw data would be the following where we test how many grids with triplet count n are generated in X iterations:

(5)
```
>>> triplet_count = {n : 0 for n in range(3,28)}
>>> rnd_seed = 42
>>> for iteration in range(X):
...     g.generate_rndGrid(seed=rnd_seed)
...     n = len(g.get_uniqueTriplets())
...     triplet_count[n] += 1
...     rnd_seed += (n % 7) + 1
```

[4]This isomorphism is a well-known fact about Sudoku geometry, see for instance Felgenhauer and Jarvis (2005, Sect. 1), who use an ordered box 1 as canonical form, rather than the top row.

[5]Fun fact: it is relatively easy to train a model on a complete set of recodings and achieve 100% accuracy (with loss < 0.0000001); what the model actually learns, however, is not the rules of Sudoku, but the abc-grid; see `https://github.com/A-Lex-McLee/PseudoQ-2.1/blob/main/psq_binary_clf.ipynb`

[6]Ongoing project from 2020-2026, created by the author of this paper; it is accessible here: `https://github.com/A-Lex-McLee/PseudoQ-2.1`; for a detailed overview and introduction, consult the manuals
`https://github.com/A-Lex-McLee/PseudoQ-2.1/blob/main/Grid_class_manual.pdf`
`https://github.com/A-Lex-McLee/PseudoQ-2.1/blob/main/GridCollection_class_manual.pdf`.

All *empirical* data are generated by this type of procedure. Even though methods to generate random seeds have been varied throughout in order to produce more diversity, the data thus obtained are by no means guaranteed to be either exhaustive nor balanced, as has already been pointed out.

Class `GridCollection`. This class handles large collections of grids: manipulate grid collections, create datasets for ML learning pipelines, store collections of grids to sql databases etc. A `GridCollection` object can be initialized directly from an existing collection of grids (numpy array), or generated via various procedures; the class `Grid`, for instance, has a method that performs grid-level permutations on the currently initialized grid (the *vertical permutation series*), and returns the resulting collection of 1,679,616 grids as a `GridCollection` (optionally, as a numpy array):

(6)
```
>>> from PsQ_Grid import GridCollection as GC
>>> gc = GC(myGridCollection)

>>> gc = g.permuteGrids(toCollection=False) # -> numpy array
>>> gc = g.permuteGrids(toCollection=True)
>>> print(gc.shape)
    (1679616, 81)
```

Among other things, `GridCollection` can recode an entire collection of grids to the same recoder key. We will always use the recoder key 123456789 (which is virtual abc-encoding):

(7)
```
>>> gc.activate_recodedSeries(recoder="123456789")
>>> gc.activeSeries[42].reshape(9,9)
    array([[1, 2, 3, 4, 5, 6, 7, 8, 9],
           [8, 9, 6, 2, 7, 3, 1, 5, 4],
           [5, 7, 4, 1, 8, 9, 2, 3, 6],
           [9, 5, 8, 7, 1, 2, 4, 6, 3],
           [7, 3, 1, 9, 6, 4, 5, 2, 8],
           [6, 4, 2, 8, 3, 5, 9, 1, 7],
           [2, 8, 7, 6, 4, 1, 3, 9, 5],
           [4, 1, 5, 3, 9, 8, 6, 7, 2],
           [3, 6, 9, 5, 2, 7, 8, 4, 1]])
```

Several methods are also available as class methods, to be performed on external grid collections (numpy arrays). We illustrate this for abc-reduction: first generate a collection of grids with $n = 3$ (see Sect. 2.3),[7] then perform some operation that produces new grids. At this stage, the resulting collection comprises $2,239,488$ unique grids, which is expected since grid-level permutations increase the grid count by a factor of $3! = 6$, cf. Sect 1.2. The class method `recode_collection_CLS` recodes the entire collection, and now the collection merely contains $746,496$ unique grids. Under a common recoder key, the operation has obviously produced a host of duplicates, which get removed by the function `get_uniques`:

[7] `generate_3tripletCollection`, `permute_boxcols`, and `get_uniques` are part of the Triplet-API, which is currently under construction, but a working version is available at `https://github.com/A-Lex-McLee/PseudoQ-2.1`. These functions and other return numpy arrays.

(8)
```
>>> ag = generate_3tripletCollection()
>>> ag.shape
    (373248, 9, 9)
>>> ag = permute_boxcols(ag)
>>> ag.shape
    (2239488, 9, 9)  #  = 373248 x 6; expected factor 6
>>> ag.get_uniques(ag)
>>> ag.shape
    (2239488, 9, 9)
>>> ag = GC.recode_collection_CLS(recoder="123456789",grids=ag)
>>> ag.get_uniques(ag)
>>> ag.shape
    (746496, 9, 9)  #  = 373248 x 2; effective factor: 2
```

As soon as grid collections exceed 20,000,000 grids, in order to avoid kernel crash, the grids will be stored in a database where sql sorts out duplicates. Subsequent operations will then be performed batch-wise from one db to the next.

(9)
```
>>> # ... some operations on the grid collection
>>> gc.activate_recodedSeries(recoder="123456789")
>>> gc.active_toSQL("triplets_rowPerms1")
```

This sort of procedure falls under the heading *np-rical*, and data thus obtained are reliable because the underlying algorithms are deterministic and no randomness is involved. In this paper, we will not, however, attempt to explain exactly which grids will get duplicated nor try to derive the effective growth factor. We will tolerate overgeneration (with subsequent reduction), and merely observe the effective result.

1 Introduction

1.1 What is a triplet?

Assume we are given a (valid) Sudoku grid g (*valid* meaning: the digits 1–9 occur exactly once in each row, column, and box):

(10)

```
-------------------------
| 5 2 1 | 9 7 6 | 8 3 4 |
| 8 3 6 | 1 4 2 | 5 9 7 |
| 4 9 7 | 5 8 3 | 1 6 2 |
-------------------------
| 2 1 5 | 7 3 9 | 6 4 8 |
| 3 6 8 | 4 2 5 | 9 7 1 |
| 9 7 4 | 8 6 1 | 3 2 5 |
-------------------------
| 1 5 2 | 3 9 7 | 4 8 6 |
| 6 8 3 | 2 5 4 | 7 1 9 |
| 7 4 9 | 6 1 8 | 2 5 3 |
-------------------------
```

Proceeding row-wise, we may decompose g into 27 contiguous segments of length 3 (the width of a box). Thus g can be represented as a sequence of segments:

(5, 2, 1), (9, 7, 6), (8, 3, 4), (8, 3, 6), (1, 4, 2), . . . , (7, 4, 9) (6, 1, 8), (2, 5, 3).

Segments are *concrete occurrences*: they are ordered triples tied to specific positions in a specific grid. By contrast, we define a **triplet** as an abstraction over segments, whose identity is determined solely by **membership**, not by order or location. A triplet could hence be represented either as a set or as a canonically ordered tuple, but to distinguish triplets from concrete segments, we will notate them using angle brackets:[8]

Segment	Set	Triplet
(5, 2, 1)	$\{2, 5, 1\}$	$\langle 1, 2, 5\rangle$

This definition immediately allows for *cross-segment identity*: distinct segments may instantiate the same triplet. For instance, consider box-column 1. It contains $3 \times 3 = 9$ segments, but these reduce to only three distinct triplets:

(5, 2, 1), (2, 1, 5), (1, 5, 2) $\Leftrightarrow$ $\langle 1, 2, 5\rangle$,

(8, 3, 6), (3, 6, 8), (6, 8, 3) $\Leftrightarrow$ $\langle 3, 6, 8\rangle$,

(4, 9, 7), (9, 7, 4), (7, 4, 9) $\Leftrightarrow$ $\langle 4, 7, 9\rangle$.

Thus, while every valid grid contains exactly 27 segments, the number of *distinct* triplets may vary. The grid g shown above, for example, contains 15 unique triplets.

[8] Under the hood, however, triplets *are* sets because they are defined by membership, the members being three distinct digits, and, in contrast to segments, they are not ordered. Therefore it is legitimate to use set-theoretic operations and relations, e.g $t \in T_1$ or $T_1 \cap T_2 = \emptyset$.

Since a classical 9×9 Sudoku grid contains 27 segments, the **maximum** possible number of distinct triplets is 27. At the other extreme, a **minimum of 3 triplets** is required: their union must cover all 9 digits in order for the grid to be valid at all.

It is plausible that not all values of n (the number of unique triplets) occur with equal frequency. To explore this, a sample of **111,230,620** randomly generated valid Sudoku grids was partitioned into classes indexed by n, and the cardinality $|cls_n|$ of each class was recorded.

unique **triplet** count class n	attestations of **grids** class size $\lvert cls_n \rvert$	order of magnitude
25	22,883,329	$\sim 10^7$
24	18,517,142	
23	15,075,995	
26	13,247,241	
20	8,647,525	$\sim 10^6$
21	7,785,875	
22	7,150,343	
19	6,238,520	
27	5,054,871	
18	2,635,381	
17	1,725,354	
15	939,704	$\sim 10^5$
14	638,052	
16	346,101	
13	196,148	
12	67,571	$\sim 10^4$
11	37,500	
9	29,114	
8	9,347	$\sim 10^3$
7	4,508	
5	621	$\sim 10^2$
3	206	
6	172	
10	0	
4	0	

Table 1: Number of unique triplets per grid (n) versus observed grid counts in a large random sample. Classes are ranked by decreasing frequency.

Given the size of the full Sudoku universe ($\approx 6.7 \times 10^{21}$ valid grids), this sample is necessarily tiny, and strict representativity cannot be claimed ($\rightarrow$ *empirical data*; cf. Sect. 0.1). Nevertheless, as a heuristic first probe, we may try to make some first careful assessment. At first glance, there is a coarse positive correlation between n and $|cls_n|$: grids with many triplets are more frequent than those with few. This is unsurprising, since the number of available triplets grows combinatorially: there are $\binom{9}{3} = 84$ possible triplets, $\binom{84}{n}$ combinations of n triplets, and the number of ways to

assemble larger collections grows rapidly as well.

However, the ranking is far from monotonic. For example, $|cls_{25}|$ is the most frequent class, while $|cls_{27}|$ appears only in ninth position; $|cls_3|$ greatly exceeds $|cls_6|$; and $|cls_{23}|$ is significantly larger than $|cls_{26}|$. One might suspect that these irregularities are artefacts of sampling, and that a truly representative sample (or the full universe) would enforce a strict ordering

$$|cls_{27}| \gg |cls_{26}| \gg |cls_{25}| \gg \cdots \gg |cls_3|.$$

Yet this need not be the case. Increasing n also interacts with the constraints imposed by Sudoku rules in a non-linear manner. Certain values of n may restrict grid construction, while others may inadvertently facilitate it, leading to local increases or decreases in $|cls_n|$. On this interpretation, Table 1 may already reflect genuine structural effects to the effect that we should not expect a strict linear correlation between n and $|cls_n|$. In this article, we will not, however, address this particular question.

Given that there are various well-known symmetries and permutation groups, see Sect. 1.2, it may appear that triplet count n is a mere emergent or ephemeral property that changes on different perspectives. Thus for the purpose of this article, we will assume that grid orientation is a primitive and the only way in which two grids can be identical is when they translate to the same abc-grid (i.e. when they are isomorph). Once this is established, triplet-hood is actually a constitutive property of the Sudoku geometry. Notably, it divides the universe of all grids into 25 discrete partitions (two of which we will show to be empty):

$$\forall g \in \mathbb{G} : \exists! n \in \{3, ..., 27\} : \texttt{uniqueTriplets}(g) = n$$

For all valid grids, there is one and only one count of unique triplets n.

In other words, n is a global invariant for every grid. Several numerical properties of n suggest themselves as potentially relevant, in particular:

(i) $n = 3k$ or $n = 9k$ (multiples of 3 or 9),

(ii) $n \equiv m \pmod 3$ or $n \equiv m \pmod 9$ (residue classes),

The numbers 3 and 9 are hard-wired into the very architecture of Sudoku, and it would be surprising if divisibility and congruence with respect to these numbers played no role. We will return to this point in due course.

The grouping of classes by powers of ten in Table 1 is merely an optical convenience. In what follows, our attention will focus on four particularly revealing meta-classes:

(i) $n = 3$ $\rightarrow$ straightforwardly constructible (and enumerable),

(ii) $n = 4$ $\rightarrow$ impossible,

(iii) $n = 10$ $\rightarrow$ impossible,

(iv) all remaining cases $\rightarrow$ an insane challenge.

1.2 Permutations, Transformations, Properties, Triplet Counts

Structure preserving vs. structure modifying permutation. In the framework of `PseudoQ`, the operation of recoding (= label permutation) alluded to in the context of abc-reduction (Sect. 0.2) is referred to as the *horizontal (permutation) series*. This contrasts with the *vertical (permutation) series*, which is the set of grid-level permutations (swapping rows, columns, boxrows, boxcolumns):[9] Given a valid grid, swapping any two columns within a boxcolumn yields another valid grid. There are three columns in one boxcolumn, hence there are $3! = 6$ such swappings; there are three boxcolumns, hence three sets of column swappings, and finally, the boxcolumns themselves can be swapped leading to $(3!)^4 = 1296$ column permutations. The same reasoning gives us 1296 row permutations, which in turn leads to $\mathbf{(3!)^8 = 1296 \times 1296 = 1,679,616}$ valid grids.

Horizontal permutation **preserves structure,** while vertical permutation is **invasive** and potentially **destroys structural properties**. To illustrate, suppose we have some valid grid g and colour the cells with nine distinct colours such that every colour corresponds to one of the digits $1-9$. Horizontal permutation (relabeling) may change the colour-digit correspondence (it will still be a uniform assignment), but crucially, each colour remains in the same position (thus you can think of it as the visualization of the underlying abc-grid), cf. (11-a). On the other hand, if we swap rows or columns, the distribution of colours will be changed: some cells and thus colours will occur in a different position, cf. (11-b) where boxrows 1 and 2 are swapped.

(11) a.

b.

Rotation and diaflection. Assume we have performed the full vertical permutation on some grid and obtain a grid collection gc of size $1,679,616$. If we now apply the

[9] Imagine all permutations of a given grid enumerated in a table where grid-level permutations occur as row labels, arranged vertically, whereas recoder keys occur as column labels, arranged horizontally. This hypothetical arrangement is what motivated the the terminology.

symmetry operations diaflection (= diagonal reflection) and rotation to gc, we obtain $2 \times 1,679,616 = 3,359,232$ distinct grids – not $8 \times 1,679,616 = 13,436,928$! The reason is that vertical permutation already produces all grids that correspond to rotations by (0° and) 180° of gc. The additional set of grids gc' correspond to rotations of gc by 90° and/ 270°, and/or gc^T (these are all permutations of each other). For this reason, we effectively only observe a factor of 2.

Other properties. In addition, Sudoku grids may have specific (geometric) properties that are often utilized in Sudoku puzzles, for instance:

(i) grid satisfies **anti knight's move constraint**: two cells a chess knight's move apart (on a bounded grid) may not contain the same digit;

(ii) **modular anti knight's move constraint**: as the former, but the grid is construed as a toroidal wrap-around structure, thus the knight can "see" across grid borders;

(iii) (main / anti / both) **diagonal(s)** contain(s) 9 distinct digits;

(iv) the digits in certain regions must **sum to certain values** (Killer Sudoku);

Now the question is: which property is preserved under which permutations/transformations? Relabeling is bound to change digit sums, cf. (iv), and row/column swapping brings all of (i)-(iv) into disarray. But **both horizontal and vertical permutation preserve the triplet count**. Horizontal permutation is bound to change triplet identities, and vertical permutation the triplet positions, but if a given grid contains n triplets, it will still contain n triplets after horizontal/vertical permutation.

On the other hand, rotation by 90°/270° and diaflection change the triplet count. The reason is that we defined triplets as abstractions over row-wise segments, and applying either transformation will transform the original row-wise segments into column-wise segments, while we now abstract over the original column-wise segments, which usually leads to a different triplet count:

```
original:                      rotated by 90° (clockwise)
-------------------------      -------------------------
| 2 3 1 | 5 4 6 | 9 7 8 |      | 4 8 3 | 6 7 1 | 5 9 2 |
| 9 8 7 | 1 2 3 | 4 5 6 |      | 6 7 1 | 5 9 2 | 4 8 3 |
| 5 4 6 | 7 8 9 | 3 1 2 |      | 5 9 2 | 4 8 3 | 6 7 1 |
-------------------------      -------------------------
| 1 2 3 | 6 5 4 | 8 9 7 |      | 9 2 4 | 8 3 6 | 7 1 5 |
| 7 9 8 | 3 1 2 | 5 6 4 |      | 7 3 6 | 9 1 5 | 8 2 4 |
| 6 5 4 | 8 9 7 | 1 2 3 |      | 8 1 5 | 7 2 4 | 9 3 6 |
-------------------------      -------------------------
| 3 1 2 | 4 6 5 | 7 8 9 |      | 2 6 7 | 1 5 8 | 3 4 9 |
| 8 7 9 | 2 3 1 | 6 4 5 |      | 3 4 8 | 2 6 9 | 1 5 7 |
| 4 6 5 | 9 7 8 | 2 3 1 |      | 1 5 9 | 3 4 7 | 2 6 8 |
-------------------------      -------------------------

    unique triplets:               unique triplets:
     <1, 2, 3>                      <1, 5, 7>,
```

```
<4, 5, 6>
<7, 8, 9>
```

```
<1, 5, 8>,
<1, 5, 9>,
<1, 6, 7>,
<2, 4, 7>,
<2, 4, 8>,
<2, 4, 9>,
<2, 5, 9>,
<2, 6, 7>,
<2, 6, 8>,
<2, 6, 9>,
<3, 4, 7>,
<3, 4, 8>,
<3, 4, 9>,
<3, 6, 7>,
<3, 6, 8>,
<3, 6, 9>
```

With that in mind, we may now precisify our first task as follows: for a given triplet count n, can we construct a core set of abc-distinct grids that are not related by grid permutation? If the answer is affirmative, we can then extend the core set by applying triplet-count-preserving operations. This staged approach is computationally far more tractable than attempting to construct all grids of triplet count n in one monolithic enumeration.

The deeper methodological question, however, is one of completeness. We must ask whether we risk missing genuine grids of triplet count n. Put differently: are there additional transformations—beyond horizontal and vertical permutation—that preserve triplet count and produce grids not already contained in our extended core set? We will come back to that question in due course.

Appendix: Abc-reduced vertical series

Recall that the vertical series comprises a collection of $1,679,616$ grids and is obtained by the set of grid-level – column, boxcolumn, row, boxrow – permutations. Ideally, this means we generate $1,679,616$ abc-distinct grids on the basis of one (valid) grid, but in several cases, we find that, after applying abc-reduction to some vertical series, the set contains a smaller number of grids. Evidently, some grids possess internal symmetries or other properties such that, in the course of permutation, structures get replicated.

To get a first impression, we conducted a little test by creating a set of grids, broadly balanced regarding triplet count (roughly 300-400 grids for each possible value of n), and for every grid in that set, we (i) generated the vertical series, (ii) abc-reduced the resulting grid collection, and (iii) counted the remaining distinct grids. Some grid counts are listed below according to n:

3: { 5184, 10368, 31104, 93312, 139968, 186624, 279936, 559872 }

5: { **1679616** }

12: { 559872, 839808, **1679616** }

14: { **1679616** }

18: { 279936, 559872, 839808, **1679616** }

27: { **1679616** }

As always with our empirical data, there is no guarantee that this is an exhaustive overview, but at the same time, this small sample highlights some points of interest:

(i) grids with $n = 3$ are highly volatile (or multiply symmetrical, if you will), in that they display by far the largest number of numbers (of remaining grids) to which a corresponding vertical series can be reduced.

(ii) for every n – except $n = 3$(?) – there are grids yielding a vertical series that does not get reduced (i.e. $1,679,616$ unique grids after abc-reduction).

(iii) all numbers (of remaining unique grids) reported above divide $1,679,616$, which suggests that it is not some random subset that is missing.

We will come back to some of the implications of this overview in Sect 4, but for now, we will note a practical issue: vertical permutation is not "reliable", as it were. Even though the procedure is fully deterministic, in conjunction with abc-reduction, we cannot predict the outcome, that is we do not know the number of unique grids the procedure yields eventually. This is especially true for $n = 3$, which we will address in the next section.

1.3 The macro ingredients

⇒ **Set of triplets**:

$\mathbb{T}^*$ = {⟨1, 2, 3⟩, ⟨1, 2, 4⟩ . . . ⟨5, 6, 7⟩, ⟨5, 6, 8⟩ . . . ⟨6, 8, 9⟩, ⟨7, 8, 9⟩}

$|\mathbb{T}^*| = \binom{9}{3} = 84$

⇒ **Set of combinations of n triplets**:

$\mathbb{T}^n = \ldots$

$|\mathbb{T}^n| = \binom{84}{n}$

$\mathbb{T}^3$ = { {⟨1, 2, 3⟩, ⟨1, 3, 7⟩, ⟨4, 6, 8⟩}, {⟨2, 3, 4⟩, ⟨5, 7, 8⟩, ⟨6, 7, 8⟩} . . . }

$|\mathbb{T}^3| = \binom{84}{3} = 95,284$

⇒ **Set of combinations of n triplets that lead to valid grids**:

$\mathbb{T}^n_g = ?$

$\mathbb{T}^n_g \subset \mathbb{T}^n$ (proper subset because not every combination of triplets leads to valid grids)

$|\mathbb{T}_g^n| = ?$

$\mathbb{T}_g^3 \quad = \quad \{\ \{\langle 1,2,3\rangle, \langle 4,5,6\rangle, \langle 7,8,9\rangle\}, \{\langle 1,4,7\rangle, \langle 2,5,8\rangle, \langle 3,6,9\rangle\} \ldots .\ \}$

$\Rightarrow$ **Set of valid grids / the language of Sudoku grids**:

$\mathbb{G}$

$\Rightarrow$ **Set of valid grids with n triplets**:

$\mathbb{G}_T^n = ?$

$|\mathbb{G}_T^n| = ?$

$|\mathbb{G}_T^1| = |\mathbb{G}_T^2| = 0$ (minimum of 3 triplets required to cover all digits 1–9)

$|\mathbb{G}_T^3| = 624,127,031,377,920$ or $1,719,926,784$ (abc-reduced) (see Sect. 2)

$|\mathbb{G}_T^4| = |\mathbb{G}_T^{10}| = 0$ (see Sects. 3.1; 3.3)

$\Rightarrow$ **Finally**:

$$\sum_{n=3}^{27} |\mathbb{G}_T^n| \quad = \quad |\mathbb{G}| \quad = \quad 6,670,903,752,021,072,936,960^{10} \quad \approx \quad 6,7 \times 10^{21}$$

2 $n = 3$: Minimal Triplet Grids – Construction

2.1 Preliminaries

Assume we have exactly three distinct triplets $T_1, T_2, T_3 \in \mathbb{T}^3$, with the property that together they exhaust the digit set:

$$T_1 \cup T_2 \cup T_3 = \{1,2,3,4,5,6,7,8,9\}, \qquad T_i \cap T_j = \varnothing \text{ for } i \neq j.$$

For concreteness, and without loss of generality, let

$T_1 = \langle 1,2,3\rangle,$

$T_2 = \langle 4,5,6\rangle,$

$T_3 = \langle 7,8,9\rangle.$

[10] This is the number of possible (valid) 9×9 Sudoku grids, disregarding symmetries and reductions, cf. Felgenhauer and Jarvis (2005).

Triplets as row-wise building blocks. By design, any concatenation of the three triplets yields a valid Sudoku row, regardless of order or internal permutation. For instance:

$$[(1,2,3),(4,5,6),(7,8,9)] \qquad [T_1,T_2,T_3]$$

$$[(4,5,6),(7,8,9),(1,2,3)] \qquad [T_2,T_3,T_1]$$

$$[(9,7,8),(2,3,1),(6,5,4)] \qquad [T_3,T_1,T_2]$$

Vertical stacking and box structure. The same observation applies vertically. Stacking (permutations of) the three triplets produces a valid 3×3 box:

$$\begin{bmatrix}(1,2,3)\\(4,5,6)\\(7,8,9)\end{bmatrix}, \quad \begin{bmatrix}(4,5,6)\\(7,8,9)\\(1,2,3)\end{bmatrix}, \quad \begin{bmatrix}(9,7,8)\\(2,3,1)\\(6,5,4)\end{bmatrix}$$

$$\begin{bmatrix}T_1\\T_2\\T_3\end{bmatrix}, \qquad \begin{bmatrix}T_2\\T_3\\T_1\end{bmatrix}, \qquad \begin{bmatrix}T_3\\T_1\\T_2\end{bmatrix}$$

Triplet grid templates. Once one triplet is given, there are only two possible continuations within the respective row/box, and as long as triplets are not repeated, validity is guaranteed at the (local) row/box level. Since a *boxcolumn* is essentially three boxes stacked, it follows that each boxcolumn contains each triplet three times. Combining these observations straightforwardly leads to the construction of a *grid template*, in which each row and each box contains all three triplets exactly once:

(12)
$$\begin{array}{ccc} T_1 & T_2 & T_3 \\ T_2 & T_3 & T_1 \\ T_3 & T_1 & T_2 \end{array}$$
$$\begin{array}{ccc} T_1 & T_2 & T_3 \\ T_2 & T_3 & T_1 \\ T_3 & T_1 & T_2 \end{array}$$
$$\begin{array}{ccc} T_1 & T_2 & T_3 \\ T_2 & T_3 & T_1 \\ T_3 & T_1 & T_2 \end{array}$$

This template already satisfies all Sudoku constraints at the triplet level. What remains is to choose *how* each triplet is internally permuted.

2.2 Column constraints and compatibility classes

At this point, an important subtlety enters. While rows and boxes are guaranteed to be valid by construction, column validity depends on how the triplets are permuted; recall we have 3×3 triplets per boxcolumn. Each triplet has $3! = 6$ internal permutations ($\rightarrow$ possible segments). These permutations fall naturally into two compatibility classes ($\rightarrow$ Latin square compatibility). For $T_1 = \langle 1, 2, 3 \rangle$, for example:

(Class 0) $(1, 2, 3),\ (2, 3, 1),\ (3, 1, 2)$ vs. $(1, 3, 2),\ (2, 1, 3),\ (3, 2, 1)$ (Class 1)

Permutations within the same class can be stacked vertically without causing column repetitions; mixing classes generally leads to conflicts. In the above grid, all permutations of T_1 belong to class 0; substituting any of those with a permutation from class 1 leads to a violation of the column constrain in the respective boxcolumn:

(1, 2, 3)	(1, 2, 3)	(1, 3, 2)	(1, 3, 2)
(2, 3, 1)	(2, 1, **3**)	(2, **3**, 1)	(2, 1, 3)
(3, 1, 2)			(3, 2, 1)

Thus, for each triplet and boxcolumn, one compatibility class must be chosen. Once chosen, the three segments, in turn, can be ordered (vertically!) in $3! = 6$ ways; we will refer to this set of permutations as the ***distribution*** *of a triplet* (= distribution over boxes within one boxcolumn). The distributions of class 0 are illustrated below:

(1, 2, 3)	(1, 2, 3)	(2, 3, 1)	(2, 3, 1)	(3, 1, 2)	(3, 1, 2)
(2, 3, 1)	(3, 1, 2)	(1, 2, 3)	(3, 1, 2)	(1, 2, 3)	(2, 3, 1)
(3, 1, 2)	(2, 3, 1)	(3, 1, 2)	(1, 2, 3)	(2, 3, 1)	(1, 2, 3)

Notice that, locally, the two compatibility classes differ only by a relabeling of digits (e.g. swapping 2 and 3):

(1, 2, 3)	[swap values: $2 \mapsto 3$; $3 \mapsto 2$]	$\Rightarrow$	(1, 3, 2)
(2, 3, 1)	[swap values: $2 \mapsto 3$; $3 \mapsto 2$]	$\Rightarrow$	(3, 2, 1)
(3, 1, 2)	[swap values: $2 \mapsto 3$; $3 \mapsto 2$]	$\Rightarrow$	(2, 1, 3)

This entails that class 1 can be reduced to class 0 (or vice versa) without losing deep structural distinctions. Effectively, we need to consider only one compatibility class per triplet – per boxcolumn, not at the grid level. So for now, we will ignore compatibility, but will return to the issue once we have constructed a core set.

2.3 Constructing and enumerating grids – the core set

So the economic question we need to ask: **How many distinct abc-grids with $n = 3$ triplets can we construct?** (and how?). This task is of some complexity, and we will therefore proceed step-wise: Let us first (re-)state our assumptions and then construct a core set:

(i) $T_1 = \langle 1, 2, 3 \rangle$; $T_2 = \langle 4, 5, 6 \rangle$; $T_3 = \langle 7, 8, 9 \rangle$;

(ii) At this first stage, we will use an ordered top row as an invariant (the digit pendant to an abc-grid):

top segment of boxcolumn 1: (1, 2, 3)
top segment of boxcolumn 2: (4, 5, 6)
top segment of boxcolumn 3: (7, 8, 9)

(iii) The two compatibility classes can be reduced to one (via labeling permutations); at this construction stage we will make use of only one compatibility class per triplet – grid-wide.

(iv) Since the top row segments are fixed in every boxcolum, every boxcolum contains one triplet with only **two admissible distributions**.

boxcol 1: **(1, 2, 3)**	boxcol 2: **(4, 5, 6)**	boxcol 3: **(7, 8, 9)**	
(2, 3, 1)	(5, 6, 4)	(8, 9, 7)	distribution I
(3, 1, 2)	(6, 4, 5)	(9, 7, 8)	

boxcol 1: **(1, 2, 3)**	boxcol 2: **(4, 5, 6)**	boxcol 3: **(7, 8, 9)**	
(3, 1, 2)	(6, 4, 5)	(9, 7, 8)	distribution II
(2, 3, 1)	(5, 6, 4)	(8, 9, 7)	

(v) The remaining two triplets per boxcolumn, respectively, each have the full range of 3! = **6 distributions**.

(vi) On this count, we actually have $2 \times 6 \times 6 = \mathbf{72}$ possible valid boxcolumns.

(vii) The prospective grid already determines the top row as [T_1 , T_2 , T_3] and thus reduces the range of possible continuations, box-wise. We stipulate the same ordering – ... T_3 , T_1 , T_2 , T_3 , T_1 , T_2 ... – vertically, effectively determining a template for each boxcolumn, which yields the grid template already illustrated in (12) above:

$$\begin{array}{ccc} T_1 & T_2 & T_3 \\ T_2 & T_3 & T_1 \\ T_3 & T_1 & T_2 \end{array}$$

$$\begin{array}{ccc} T_1 & T_2 & T_3 \\ T_2 & T_3 & T_1 \\ T_3 & T_1 & T_2 \end{array}$$

$$\begin{array}{ccc} T_1 & T_2 & T_3 \\ T_2 & T_3 & T_1 \\ T_3 & T_1 & T_2 \end{array}$$

This way we ensure that every triplet occurs exactly once per row and per box (and three times per boxcolumn), and they are compatible Sudoku-wise.

(viii) Since the boxcolumns are therefore independent of each other, valid grids are now ensured by the Cartesian product $boxcol_1 \times boxcol_2 \times boxcol_3$, which amounts to $72 \times 72 \times 72 = 72^3 = \mathbf{373,248}$ valid abc-grids.

The grids so generated are already valid and genuinely distinct, i.e. they cannot be further abc-reduced. Validity follows from the design: 1. three disjoint triplets, 2. valid arrangements of triplets in rows and boxes/ boxcolumns, 3. identical compatibility classes for each triplet. Abc-distinctness also follows from design: all grids are generated with an identical top row, cf. point (ii), and de facto *are* abc-grids. Henceforth, we will refer to this procedure as `generate_3tripletCollection`, which is the name of the corresponding function in the triplet API.

So our core set does indeed contain 373,248 grids. Still it is valid to ask: Did we miss anything? Notably, didn't we loose possible grids by only considering two distributions per boxcolumn for the triplet that occurs in the respective top row? Let's check: Omitting points (ii/iv) has two consequences:

- Our set includes grids with non-canonically ordered top rows:

  [(2, 1, 3), (4, 5, 6), (7, 8, 9)]

  [(1, 2, 3), (6, 5, 4), (7, 8, 9)]

  [(1, 2, 3), (4, 5, 6), (9, 7, 8)] . . . etc.

- We raise the count of possible valid boxcolumns to $6 \times 6 \times 6 = \mathbf{216}$, and, consequently, generate $216 \times 216 \times 216 = \mathbf{10,077,696}$ grids.

Checking behind the scene shows: The collection generated does indeed contain $10,077,696$ valid grids. However, if we abc-reduce this collection, we once again arrive at **373,248** grids. This shows that the additionally generated grids are mere notational variants (label permutations / recodings) of the ones we generate by `generate_3tripletCollection` as described above. We were right to restrict the construction the way we did, i.e. fixing the top row and thus exempting the triplets involved from permutation.[11]

2.4 Grid-level permutations: Extending the core set

It seems we have exhausted the range of low-level permutations, let's now attend to applying grid-level permutations to the core set. In Sect. 1.2, it was shown that vertical permutation, very especially of grids with $n = 3$, creates redundancy, i.e. heavily overgenerates grids that get pruned by abc-reduction. Thus we cannot simply assume that we will generate $373,248 \times 1,679,616$ distinct grids on the basis of our core set. For this reason and for the sake of economy, we will not permute each individual grid in the core set, but instead apply the individual components of vertical permutation

[11] As a matter of fact, `generate_3tripletCollection` has a parameter `full_perm` with the default setting `False`, which suppresses these additional permutations, but it can be set to `True`.

$$\begin{bmatrix} permuteColumns(1,2,3) \\ permuteColumns(4,5,6) \\ permuteColumns(7,8,9) \\ permuteBoxColumns() \\ permuteRows(1,2,3) \\ permuteRows(4,5,6) \\ permuteRows(7,8,9) \\ permuteBoxRows() \end{bmatrix}$$

successively directly to the core set and abc-reduce at intermediate stages.

Each such component potentially contributes a factor of $3! = 6$, but in many cases, we will actually observe a smaller factor.

Column permutations. If we apply some column permutation to the entire core set, we get a collection of $2,239,488 = 6 \times 373,248$ distinct(!) grids ($\rightarrow$ the expected factor of 6). However, after we abc-reduction, only $746,496 = 2 \times 373,248$ grids remain. Effectively, we only have a factor of 2. However, if we apply all three column permutations to the core set with subsequent abc-reduction, we only get $1,492,992 = 2 \times 2 \times 373,248$ unique grids. In other words, instead of another factor of 2, it seems as though the third set of column permutations is inert in that it does not produce any new grids. Notice that it is not any particular set of column swappings that is, as it where, spinning its wheels (because we tested them individually); rather, it's the combination of the three permutations.

Finally, if we apply the full range of the column permutation series: three sets of column permutations $\times$ boxcolumn permutation, and perform abc-reduction, we obtain $\mathbf{2,985,984} = 2 \times 2 \times 1 \times 2 \times 373,248$ unique grids. Thus, boxcolumn permutation contributes another factor of 2. In short, instead of an expected factor of $(3!)^4 = 1,296$, we only observe an effective factor of $2^3 = \mathbf{8}$.

Row permutations. We repeat the same procedure for the row permutation series, and after abc-reduction we get $\mathbf{26,873,856} = 6 \times 6 \times 2 \times 1 \times 373,248 =$ grids. Row permutations display the full factor of 6, but again, we see a reduction for all three sets of row permutations together (2 instead of 6). Finally, boxrow permutation does not generate any new grids and thus contributes a factor of 1.[12] At any rate, row permutations contribute an effective factor of $6 \times 6 \times 2 \times 1 = \mathbf{72}$.

Combining all permutations. The expectation is that the full set of grid permutations should yield a grid collection of size $8 \times 72 \times 373,248 = 214,990,848$. But performing the procedure shows that we effectively get **107, 495, 424 unique = abc-distinct grids**; combining all columns and row permutations has eliminated one factor of 2. In other words, $107,495,424$ is the final result of our current computations; let us refer to the total grid collection just created as the **E**xtended **C**ore **S**et zero $= \mathrm{ECS}_\mathbf{0}$.

[12] This is a consequence of our design: at the triplet level, all boxrows are identical, cf. (12), and since any row permutation does not change the (internal) ordering of triplets, crucially, does not change compatibility class, every change brought about by boxrow swapping is effectively undone by abc-reduction.

The subscript zero is meant to indicate that we still have not reached the end; there is one more thing to consider.

2.5 Compatibility classes once more: The Compatibility Index

At the construction stage, we decided to build our grids from one (and the same) compatibility class 0 for each triplet, cf. Sect. 2.3; that was economic and elegant. In a next step, we performed grid-level permutations to extend the core set; this was bound to stir things up a bit and introduce new structures. Notably, column permutations will create the respective other compatibility class 1. In (13), we show three of the six grids created by the column permutations of a grid from the core set, cf. (13-a). Focus on boxcolumn 1.

(13) **column permutations within boxcolumn 1** (partial)

```
(a) original: class 0          (b) permutation: class 1       (c) permutation: class 0
-----------------------        -----------------------        -----------------------
| 1 2 3 | 4 5 6 | 7 8 9 |      | 1 3 2 | 4 5 6 | 7 8 9 |      | 2 3 1 | 4 5 6 | 7 8 9 |
| 4 5 6 | 7 8 9 | 1 2 3 |      | 4 6 5 | 7 8 9 | 1 2 3 |      | 5 6 4 | 7 8 9 | 1 2 3 |
| 7 8 9 | 1 2 3 | 4 5 6 |      | 7 9 8 | 1 2 3 | 4 5 6 |      | 8 9 7 | 1 2 3 | 4 5 6 |
-----------------------        -----------------------        -----------------------
| 2 3 1 | 5 6 4 | 8 9 7 |      | 2 1 3 | 5 6 4 | 8 9 7 |      | 3 1 2 | 5 6 4 | 8 9 7 |
| 5 6 4 | 8 9 7 | 2 3 1 |      | 5 4 6 | 8 9 7 | 2 3 1 |      | 6 4 5 | 8 9 7 | 2 3 1 |
| 8 9 7 | 2 3 1 | 5 6 4 |      | 8 7 9 | 2 3 1 | 5 6 4 |      | 9 7 8 | 2 3 1 | 5 6 4 |
-----------------------        -----------------------        -----------------------
| 3 1 2 | 6 4 5 | 9 7 8 |      | 3 2 1 | 6 4 5 | 9 7 8 |      | 1 2 3 | 6 4 5 | 9 7 8 |
| 6 4 5 | 9 7 8 | 3 1 2 |      | 6 5 4 | 9 7 8 | 3 1 2 |      | 4 5 6 | 9 7 8 | 3 1 2 |
| 9 7 8 | 3 1 2 | 6 4 5 |      | 9 8 7 | 3 1 2 | 6 4 5 |      | 7 8 9 | 3 1 2 | 6 4 5 |
-----------------------        -----------------------        -----------------------
```

The permutation shown in (13-c) belongs to the same compatibility class as the original, but the one in (13-b) displays compatibility class 1 (in boxcolumn 1). Notice three things:

(i) column permutation affects all triplets in the respective boxcolumn simultaneously; in (13-b), all $\langle 1,2,3\rangle$, $\langle 4,5,6\rangle$, $\langle 7,8,9\rangle$ occur in compatibility class 1 in boxcolumn 1

(ii) abc-encoding adds yet another effect as illustrated in (14): in the (b)-permutation, the top triplet $\langle 1,2,3\rangle$ now occurs in class 0 again, whereas the other two triplets remain in class 1. Moreover, in boxcolumns 2 and 3, which were not affected originally, triplet $\langle 1,2,3\rangle$ now occurs in class 1:

(iii) The (c)-permutation has remained in class 0 all along, only, after abc-encoding, see (14-c), the top row is ordered once again. As a matter of fact, (14-c) is a grid that must have been generated as a member of the original core set and will be pruned eventually. This explains the observed factor of 2 observed in Sect. 2.4: only compatibility class survives abc-reduction (after column permutation).

(14) **column permutations re-abc-encoded**

```
(b)  011    100     100          (c)  000    000     000
-------------------------        -------------------------
| 1 2 3 | 4 5 6 | 7 8 9 |        | 1 2 3 | 4 5 6 | 7 8 9 |
| 4 6 5 | 7 8 9 | 1 3 2 |        | 5 6 4 | 7 8 9 | 3 1 2 |
| 7 9 8 | 1 3 2 | 4 5 6 |        | 8 9 7 | 3 1 2 | 4 5 6 |
-------------------------        -------------------------
| 3 1 2 | 5 6 4 | 8 9 7 |        | 2 3 1 | 5 6 4 | 8 9 7 |
| 5 4 6 | 8 9 7 | 3 2 1 |        | 6 4 5 | 8 9 7 | 1 2 3 |
| 8 7 9 | 3 2 1 | 5 6 4 |        | 9 7 8 | 1 2 3 | 5 6 4 |
-------------------------        -------------------------
| 2 3 1 | 6 4 5 | 9 7 8 |        | 3 1 2 | 6 4 5 | 9 7 8 |
| 6 5 4 | 9 7 8 | 2 1 3 |        | 4 5 6 | 9 7 8 | 2 3 1 |
| 9 8 7 | 2 1 3 | 6 4 5 |        | 7 8 9 | 2 3 1 | 6 4 5 |
-------------------------        -------------------------
```

Thus the application of grid-level permutations plus subsequent abc-encoding obviously introduces novel distributions of compatibility classes. The question is whether the procedure is exhaustive and creates all possible distributions. In other words, are there (abc-encoded) grids with some distribution of compatibility classes that are not already contained in ECS_0?

Before addressing the question let's operationalize and define the notion more precisely, and establish some conventions:

1.) from the $3! = 6$ permutations of some triplet T, consider the two with the lowest digit in leftmost position:

   - ◇ the one that is ordered in ascending order, e.g. $\langle \mathbf{4}, 5, 6 \rangle$, as well as the two permutations that are Latin square compatible with it, belong to class **0**;
   - ◇ the one that is not ordered, e.g. $\langle \mathbf{4}, 6, 5 \rangle$, along with the two that are Latin square compatible with that one, belong to class **1**.

2.) order the triplets themselves in ascending order and assign them row indices from $1 - 3$ in that order, e.g. $\langle 1, 2, 3 \rangle \rightarrow 1$, $\langle 4, 5, 6 \rangle \rightarrow 2$, $\langle 7, 8, 9 \rangle \rightarrow 3$.

3.) construct a 3×3 matrix – the **Compatibility Index** – e.g.

$$\begin{bmatrix} 0, 0, 0 \\ 0, 0, 0 \\ 0, 0, 0 \end{bmatrix}$$

such that columns represent boxcolumns and rows represent triplets according to the index assigned in the previous step.

4.) The values of the compatibility index represent the compatibility class of a given triplet in a given boxcolumn.

Obviously, the one illustrated, with only zeros, is the one we find in our core set. Now it so happens that our function `generate_3tripletCollection` has a parameter with the default setting `idx_arr = np.zeros((3, 3))`, which gives us precisely our core set. But we can pass any other binary-valued 3×3 matrix as an argument, for instance

(15) a.

$$\begin{bmatrix} 1, 0, 0 \\ 0, 1, 0 \\ 0, 0, 1 \end{bmatrix}$$

b.
```
>>> comp_idx = np.array([[1,0,0],[0,1,0],[0,0,1]])
>>> generate_3tripletCollection(idx_arr=comp_idx) )
```

which generates a core set based on this distribution of compatibility classes, and gives us for instance the following grid:

(16)
```
-------------------------
| 1 3 2 | 4 6 5 | 7 9 8 |
| 4 5 6 | 7 8 9 | 1 2 3 |
| 7 8 9 | 1 2 3 | 4 5 6 |
-------------------------
| 3 2 1 | 6 5 4 | 9 8 7 |
| 5 6 4 | 8 9 7 | 2 3 1 |
| 8 9 7 | 2 3 1 | 5 6 4 |
-------------------------
| 2 1 3 | 5 4 6 | 8 7 9 |
| 6 4 5 | 9 7 8 | 3 1 2 |
| 9 7 8 | 3 1 2 | 6 4 5 |
-------------------------
```

Be sure to understand what this means exactly: in Boxcolum 1, T_1 belongs to class 1, T_2 and T_3 to class 0; in Boxcolum 2, T_2 belongs to class 1, T_1 and T_3 to class 0; in Boxcolum 3, T_3 belongs to class 1, T_1 and T_3 to class 0.

Since there are nine positions in a compatibility index, and compatibility class is binary-valued, there are $2^9 = 512$ possible compatibility indices. If we flatten the matrix, construe the output as a binary number, we can use the corresponding decimal number as an index as well: what we have so far considered our core set, is actually core_set$_0$, and the grid displayed above belongs to core_set$_{273}$ ($\leftarrow 100010001_2$).

Back to our question: will we, with this new tool, generate grids that are not already contained in ECS$_0$? The short answer is: yes.

2.6 Final step: Maximally extending the core sets

The good news is that we do not have to generate 512 *extended* core sets. Out of the 512 minimal core sets (core_set$_0$−core_set$_{511}$), 32 are fully contained in ECS$_0$; by extension, all their grid-level permutations will also be included. Let us refer to such a group as a **M**aximally **E**xtended **C**ore **S**et: **MECS**. It turns out that, however, that the MECSs are not evenly distributed and comprise different numbers of core sets, and that the respective core sets themselves expand differently. Altogether, we can identify four non-overlapping MECSs (see Sect. 7.2 for the full listing):

MECS$_0$: after column permutation: 2,985,984; comprises 32 core sets; total: $\mathbf{107,495,424}$

MECS$_1$: after column permutation: 2,985,984; comprises 96 core sets; total: $\mathbf{322,486,272}$

MECS$_2$: after column permutation: 8,957,952; comprises 96 core sets; total: $\mathbf{322,486,272}$

MECS$_3$: after column permutation: 8,957,952; comprises 288 core sets; total: $\mathbf{967,458,816}$

Since these four groups are non-overlapping, we arrive at a count of

$$107,495,424 + 322,486,272 + 322,486,272 + 967,458,816 = \mathbf{1,719,926,784}$$

grids, and since they are standardized $\Leftrightarrow$ abc-distinct (i.e. not recodings of each other), we could now apply horizontal permutation, to obtain a final count of

$$1,719,926,784 \times 362,880 = \mathbf{624,127,031,377,920 \text{ grids with } n = 3}.$$

This will be the size of the maximally extended set, i.e. the number of (surface-distinct) grids, for $n = 3$.

Towards the end of Sect. 1.2, we asked whether there are operations, beside horizontal and vertical permutation, that preserve triplet count, but produce genuinely new grids. Given our methodology – starting out from simple core sets – the inclusion of the compatibility index certainly qualifies. But is that it? We cannot offer a proof for exhaustiveness for now, but likewise, we cannot think of any legal transformation leading to novel grids with $n = 3$ that is is not captured by any of the permutations that we have applied; in short: we don't think there are any grids with $n = 3$ that are not comprised in the union of the four MECS (plus recoding).

In spite of some twists, constructing our maximally extended set has been relatively straightforward. For grids with $n \neq 3$, this will be considerably more complex. For one thing, independence of boxcolumns cannot be guaranteed by construction because different triplets will occur in different boxcolumns and there will be triplets with non-empty intersections. As a consequence, compatibility classes will lose their force because it is no longer clear that the same triplet occurs three times in one boxcolumn. So there are quite some challenges ahead.

3 Impossible Grids and Boxrow Triplet Count

In Table 1, the entries for $n = 4$ and $n = 10$ are 0, respectively, i.e. no valid grid with exactly four or ten distinct triplets is attested. Are those systemic gaps or do such grids exist, but are simply too rare to appear in the sample? If they do not exist, do the numbers 4 and 10 have any special property in common? We notice that $4 \equiv 10 \pmod 3$, but the same is true for 7, 13 ... 25, and grids with these triplet counts do exist (in fact, as noted earlier, grids for $n = 25$ appear to be the most frequent class).

Anyway, in this section, we will show that grids with $n = 4$ or $n = 10$ cannot exist, and that the "rest 1" reasoning is not entirely off (perhaps off by one). A fundamental element in the arguments to follow is simply the rules of Sudoku: we will assume the validity of a grid (component) and show that, in conjunction with a second assumption (e.g. $n = 4$), this leads to a contradiction.

3.1 $n = 4$: Impossible Grids

Proof by contradiction. Assume that there exists a valid Sudoku grid g built from **four distinct triplets** T_1, T_2, T_3, T_4.

Minimally, it must be the case that three of them constitute a valid row in g:

$$[T_1, T_2, T_3] \sqsubset g \qquad \Leftrightarrow \qquad T_1 \cup T_2 \cup T_3 = \{1,2,3,4,5,6,7,8,9\}$$

By assumption, a fourth triplet T_4 occurs somewhere in g; wherever it occurs, it must be accompanied by two triplets $T_x, T_y \in \{T_1, T_2, T_3\}$ to complete the respective row. If it doesn't lead to a valid row because $T_4 \cup T_x \cup T_y \neq \{1,2,3,4,5,6,7,8,9\}$, grid construction already fails here. Suppose, then, that T_4 *can* be combined to yield a valid row. But in that case, T_4 *must* be identical to the triplet not selected:

$$T_4 \cup T_x \cup T_y = \{1,2,3,4,5,6,7,8,9\} = T_1 \cup T_2 \cup T_3$$

$$\text{(i)}\ T_4 \cup T_2 \cup T_3 = \{1,2,3,4,5,6,7,8,9\} \quad \Rightarrow \quad T_4 = T_1,$$

$$\text{(ii)}\ T_4 \cup T_1 \cup T_2 = \{1,2,3,4,5,6,7,8,9\} \quad \Rightarrow \quad T_4 = T_3,$$

$$\text{(iii)}\ T_4 \cup T_1 \cup T_3 = \{1,2,3,4,5,6,7,8,9\} \quad \Rightarrow \quad T_4 = T_2.$$

This, in turn, contradicts the original assumption that there are **four distinct triplets**. Once T_4 coincides with any of the others, the grid in fact contains only **three** distinct triplets. We conclude that a valid Sudoku grid with exactly four distinct triplets cannot exist: either the grid violates Sudoku constraints, or it collapses into the already admissible case $n = 3$.

q.e.d.

Since grid construction already fails at the row-level, we can summarize the finding of this subsection at two levels:

(α-1) For $n = 4$, **no valid grid** can be constructed.

(β-1) For $n = 4$, **no valid boxrow** can be constructed.

3.2 Boxrow triplet count can only be 3 or 9

Grids with $n = 5, 7, 8...$ escape the above problem ("new" triplet is identical to an already existing one) via *pairwise union identity*: The union of two "new" triplets is identical to the union of two existing ones without repeating a triplet. Assume we have five distinct triplets: $T_1 \neq T_2 \neq T_3 \neq T_4 \neq T_5$ such that

- $T_1 \cup T_2 \cup T_3 = \{1,2,3,4,5,6,7,8,9\}$
- $T_2 \cup T_3 = T_4 \cup T_5$ (pairwise union identity)

$\Rightarrow$ $[\boldsymbol{T_1}, T_2, T_3]$ is a valid row $\Leftrightarrow$ $[\boldsymbol{T_1}, T_4, T_5]$ is a valid row

This allows us to construct three valid boxrows each containing three triplets, but together hosting five, seven or eight distinct triplets while avoiding the problem we observed for $n = 4$. In grids with $n = 6$, pairwise union identity is not required because two distinct triplet sets each of count three are used for two distinct boxrows. Grids with $n \geq 9$, introduce the possibility that one or more boxrows have a triplet count of nine. Some valid templates for $n = 5 - 9$ are given below (double brackets signify "boxrow"):

5:

$$\begin{matrix}[[T_1 & T_2 & T_3]] \\ [[T_1 & T_4 & T_5]] \\ [[T_1 & T_2 & T_3]]\end{matrix}$$

6:

$$\begin{matrix}[[T_1 & T_2 & T_3]] \\ [[T_4 & T_5 & T_6]] \\ [[T_1 & T_2 & T_3]]\end{matrix}$$

7:

$$\begin{matrix}[[T_1 & T_2 & T_3]] \\ [[T_4 & T_5 & T_3]] \\ [[T_6 & T_7 & T_3]]\end{matrix}$$

8:

$$\begin{matrix}[[T_1 & T_2 & T_3]] \\ [[T_4 & T_5 & T_6]] \\ [[T_7 & T_1 & T_8]]\end{matrix}$$

9:

$$\begin{bmatrix}T_1 & T_2 & T_3 \\ T_4 & T_5 & T_6 \\ T_7 & T_8 & T_9 \\ T_3 & T_2 & T_1 \\ T_1 & T_3 & T_2 \\ T_2 & T_1 & T_3 \\ T_1 & T_2 & T_3 \\ T_3 & T_2 & T_1 \\ T_2 & T_3 & T_1\end{bmatrix} \quad \begin{bmatrix}T_1 & T_2 & T_3 \\ T_2 & T_3 & T_1 \\ T_3 & T_2 & T_1 \\ T_4 & T_5 & T_6 \\ T_7 & T_8 & T_9 \\ T_2 & T_3 & T_1 \\ T_3 & T_8 & T_5 \\ T_1 & T_9 & T_6 \\ T_2 & T_7 & T_4\end{bmatrix} \quad \begin{bmatrix}T_1 & T_2 & T_3 \\ T_4 & T_5 & T_6 \\ T_7 & T_8 & T_9 \\ T_4 & T_5 & T_6 \\ T_7 & T_8 & T_9 \\ T_1 & T_2 & T_3 \\ T_8 & T_2 & T_5 \\ T_9 & T_3 & T_6 \\ T_7 & T_1 & T_4\end{bmatrix}$$

But crucially, the number of triplets per boxrow is either 3 or 9:

(17)

n	triplet count per boxcolumn
$3, 5 - 8$	3
$9, 11 - 21$	3, 9
$22 - 27$	9

For what follows, we will us the shorthands br for boxrow and n_{br} for triplet count in a boxrow; also keep in mind the following twist of the pigeonhole principle imposed by the rules of Sudoku:

(γ) Given some valid grid g, then

$$\forall\, T_x, T_y, T_z,\ [T_x, T_y, T_z] \sqsubset g\colon\ \{1,2,3,4,5,6,7,8,9\} \setminus T_y \;=\; T_x \cup T_z \;=\; R$$
$$\wedge \quad \left|\begin{matrix} T_y \\ R \end{matrix}\right| \quad \sqsubset \quad g$$

For any three triplets constituting a valid row, the union of any two of them comprise the digits that the remaining one requires to complete a valid box.

Since a boxrow contains minimally 3 triplets and cannot be valid if it does not even comprise a valid row, let that be our point of departure: we are given three triplets $T_1 \neq T_2 \neq T_3$ that complete one valid row.

- $T_1 \cup T_2 \cup T_3 = \{1,2,3,4,5,6,7,8,9\}$
- $T_1 \cap T_2 \cap T_3 = \emptyset$

A boxrow comprises nine segments, and thus nine potential triplet positions, and thus there remain six positions ($?_4$–$?_9$) whose contents need to be identified:

(18)

$\boldsymbol{T_1}$	T_2	T_3
$?_4$	$?_5$	$?_6$
$?_7$	$?_8$	$?_9$

Let us assume the perspective of T_1: by architecture, it can "see" the positions marked in red because they are in the same row / box. From (γ), it follows that

(γ-1) $T_1 \cup T_2 = ?_6 \cup ?_9$

(γ-2) $T_1 \cup T_3 = ?_5 \cup ?_8$

(γ-3) $T_2 \cup T_3 = ?_4 \cup ?_7$

Let us then consider the positions "invisible" to T_1 in boxes 2 and 3, here marked in light blue ($?_5, ?_8, ?_6, ?_9$). There are two options to consider:

(A) some of $\{?_5, ?_8, ?_6, ?_9\}$ is identical to T_1,

(B) or it is not.

Ad (A): For the sake of illustration, assume that $?_8 = T_1$:

(19)

$\boldsymbol{T_1}$	T_2	T_3
$?_4$	$?_5$	$?_6$
$?_7$	T_1	$?_9$

Now the identity of $?_5$ is forced by (γ-2) / because it sees both T_1 and T_2; so in order to validly complete the box it must be identical to T_3:

(20)

$\boldsymbol{T_1}$	T_2	T_3
$?_4$	T_3	$?_6$
$?_7$	T_1	$?_9$

This in turn forces the identity of box 3: the digits of T_1 must occur there, but none of them can occur in $?_9$ nor in T_3, so T_1 must be $?_6$, which in turn forces $?_9$ to be T_2, (γ-1). This finally forces $?_4$ to be T_2 and $?_7$ to be T_3:

(21)

$\boldsymbol{T_1}$	T_2	T_3
T_2	T_3	T_1
T_3	T_1	T_2

In short, we observe a ripple effect: the moment one triplet repeats in a boxrow br_i, three triplets repeat exactly three times and $n_{br_i} = 3$.

Ad (B): Let's say that T_1 is **not** repeated. Then by (γ), each of the triplets "invisible" to T_1 will have a non-empty intersection with T_1, which has another ripple effect: no triplet can be repeated. Consider the template as a point of departure once more.

(22)

$\boldsymbol{T_1}$	T_2	T_3
$?_4$	$?_5$	$?_6$
$?_7$	$?_8$	$?_9$

i. $(T_1 \cap ?_5 \neq \emptyset \wedge ?_5 \cap T_3 \neq \emptyset) \wedge (T_1 \cap ?_8 \neq \emptyset \wedge ?_8 \cap T_3 \neq \emptyset)$ by (γ-2)

$?_5 \subset (T_1 \cup T_3) \wedge ?_8 \subset (T_1 \cup T_3)$

ii. $(T_1 \cap ?_6 \neq \emptyset \wedge ?_6 \cap T_2 \neq \emptyset) \wedge (T_1 \cap ?_9 \neq \emptyset \wedge ?_9 \cap T_2 \neq \emptyset)$ by (γ-1)

$?_6 \subset (T_1 \cup T_2) \wedge ?_9 \subset (T_1 \cup T_2)$

The triplets in light blue contain digits from both T_1 and T_3 (box 2), or from both T_1 and T_2 (box 3). Therefore, T_2 cannot be repeated in box 3 nor T_3 in box 2 because their digits are distributed across two triplets, respectively, because any combination would lead to repeated digits. But because of this digit distribution, $?_5$ cannot be identical to $?_9$ and $?_8$ not to $?_6$, either. Finally, by the same reasoning, $?_4$ and $?_7$ cannot be identical to any of T_2, T_3 or $?_5, ?_6, ?_8, ?_9$.

In other words, once there is one non-empty intersection, i.e. the digits of some triplet are distributed across two others in some boxrow br_i, no triplet can repeat in br_i, thus we have a triplet count triplet count $n_{br_i} = 9$.

(β) Regardless of the overall triplet count $\boldsymbol{n}$, in a valid grid,
the triplet count per boxrow is either 3 or 9.

3.3 $n = 10$: Impossible Grids

Preliminaries. Given (β), since there are three boxrows, we could maximally construct a grid for $n = 9$ only using boxrows with $n_{br} = 3$. Thus in order to achieve $n = 10$, we must assume that there exists at least one boxrow with $n_{br} = 9$. This will be our point of departure.

Assume then that we have one valid boxrow br_i with $n_{br_i} = 9$. T_{10} must be introduced in a separate boxrow br_j, and in whichever position it occurs in br_j, it will have to be accompanied by two triplets $T_x, T_y \in \{T_1, T_2 \ ... \ T_9\}$ in order to complete the respective row. In short:

i. $br_i = [[T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9]]$ — one valid boxrow must be given

ii. $T_x, T_y \in \{T_1, T_2 ... T_9\}$ — $T_x \neq T_y$ are already given

iii. $T_{10} \notin \{T_1, T_2 ... T_9\}$ — T_{10} is a "new" triplet

iv. $T_{10} \cup T_x \cup T_y = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ — every digit must occur once

v. $T_x \cap T_y = \emptyset$ — no digit may be repeated

Finding companions for T_{10}. We have established that T_x, T_y must be chosen from br_i. There are two options, both of which lead to the violation of an assumption:

(A) T_x and T_y are in the same row (or box) in br_i $\Rightarrow$ $T_{10} \in \{T_1, T_2 ... T_9\}$

(B) T_x and T_y are not in the same row (or box) in br_i $\Rightarrow$ $T_x \cap T_y \neq \emptyset$

Case (A) is straightforward: if T_x, T_y are chosen from the same row (or box) in br_i, T_{10} must be identical to the remaining triplet in that same row (or box), which is in br_i and thus an element of $\{T_1, T_2 ... T_9\}$. This is precisely the same problem we observed for $t = 4$ above. If the resulting row (or box) is to be valid, the identity of a triplet is forced if two are given that already occur together elsewhere. In this case, assumption iii. above is violated; effectively, we end up with $n = 9$.

Case (B) makes use of what we established in the previous subsection. Since the prerequisite is that $n_{br_i} = 9$, no triplet may be repeated. If T_x and T_y are in positions invisible to each other (and they cannot be identical), they will always have a nonempty intersection. As a consequence, T_x, T_y such that $[\dots \ T_x \ \dots \ T_y \ \dots]$ constitutes a valid row cannot exist – regardless of the identity of T_{10} because one or two digits will be repeated before we even introduce T_{10}.

For the sake of illustration, consider the concrete boxrow below with $n_{br_i} = 9$; we want to pick suitable T_x, T_y as companions for T_{10} to complete a row in br_j.

(23) a. br_i =

$$\left|\left| \begin{array}{ccc|ccc|ccc} \mathbf{9} & \mathbf{3} & \mathbf{6} & 1 & 7 & 8 & 5 & 2 & 4 \\ 5 & 4 & 8 & 3 & 6 & 2 & 1 & 9 & 7 \\ 2 & 7 & 1 & 4 & 5 & 9 & 6 & 8 & 3 \end{array} \right|\right|$$

b. br_j = $\left|\left| \dots \ [\, T_{10} \ \ T_x \ \ T_y \,] \ \ \dots \right|\right|$

Assume we have picked the dark blue segment (9, 3, 6) as our T_x. We cannot pick T_y from a position in the same row or box as T_x (the ones marked in red) because then T_{10} would be identical to one triplet already occurring. But among the positions "invisible" to T_x (here marked in light blue), we know by (γ), that every segment occurring there shares one or two digits with T_x. Thus, we cannot pick T_y from there either.

The upshot is that we cannot construct a (valid) grid with $n = 10$; either the construction effectively reduces to $n = 9$, case (A), or it is not possible to find two row mates for T_{10} in order to validly complete the row in a new boxrow, case (B).

(α)-2 For $n = 10$, **no valid grid can be constructed**.

3.4 Summary

In this section, we have shown that there is a systematic reason for the absence of (valid) grids with $n = 4$ and $n = 10$ from table 1. We can summarize our findings at different levels of abstractions:

(α) **For $n = k$, $k \in \{4, 10\}$, no valid grid can be constructed**!

(α') **For $n = k + 1$, $k \in \{3, 9\}$, no valid grid can be constructed**!
where k is the number of possible triplet counts per boxrow, cf. (β).

(α'') **For $n = 3^k + 1$ ($k \in \mathbb{N}^+$), no valid grid can be constructed!**

(α) merely states the facts, whereas (α') adds an explanatory component in that it relates the impossible triplet counts at grid level to the legal counts of triplets at the boxrow level. (α'') is even stronger, but also largely vacuous because there are only relevant values for k to consider: $3^1 + 1 = 4$ and $3^2 + 1 = 10$; on the other hand, it emphasizes the importance of the basenumber 3 for the geometry of classical 9×9 Sudoku grids.

4 n in General – Futility of random generation

Let's take a detour and consider some practical issues. As shown in (6), the class `Grid` offers a function to generate (valid) Sudoku grids at random. In addition, the random generator can be constrained: (i) anti knight's move constraint; (ii) modular anti knight's move constraint; (iii) one or both diagonal must contain distinct digits; all constraints can be imposed simultaneously:

```
>>> g.setGenerator(anti_knight=True,
                   modular_knight=True,
                   diagonal_choice="both")
>>> g.generate_rndGrid(seed=42)
>>> g.showGrid()
    -------------------------
    | 4 9 8 | 3 2 1 | 6 5 7 |
    | 1 5 3 | 7 9 6 | 8 2 4 |
    | 6 2 7 | 4 5 8 | 3 9 1 |
    -------------------------
    | 8 6 2 | 1 4 5 | 7 3 9 |
    | 5 3 1 | 9 6 7 | 2 4 8 |
    | 7 4 9 | 8 3 2 | 1 6 5 |
    -------------------------
    | 2 7 4 | 5 8 3 | 9 1 6 |
    | 3 1 5 | 6 7 9 | 4 8 2 |
    | 9 8 6 | 2 1 4 | 5 7 3 |
    -------------------------
```

This setting imposes some severe restrictions on a possible output making the computation quite complex; nonetheless, in maximally 3-5 seconds the desired output is delivered. However, as soon as we attempt to construct a corresponding function

```
>>> def generate_randomTripletGrid(n: int) -> np.ndarray: . . .
```

that receives an input value n in order to output a valid Sudoku grid with n distinct triplets ($\rightarrow$ grid $\in \mathbb{G}_T^n$), it is highly unlikely that it will (systematically) produce the desired output in an acceptable time. Concrete implementation specifics aside, this is not accidental, and there is a reason why the general random grid generator (even with some constraints) works so well, whereas the random triplet grid generator does not. Although both problems concern the same combinatorial object, their constraint geometry differs fundamentally.

(A) Unconstrained Sudoku generation. In this problem, the algorithm searches for any grid satisfying the local Sudoku constraints: each row, column, and 3×3 box must contain the digits $1, \ldots, 9$ exactly once. These constraints are local, homogeneous, and symmetric across the grid. Crucially, the algorithm is free to accept *any* configuration that satisfies them.

From a combinatorial perspective, this problem benefits from the enormous size of the solution space (on the order of 10^{21} grids, modulo symmetries and reductions).

Randomized construction or backtracking with constraint propagation is highly effective: partial assignments typically admit many valid extensions, and dead ends are pruned early. The algorithm is therefore performing an existence search in a large, well-connected feasible space, which explains why valid grids can be generated reliably and quickly in practice.

(B) Sudoku generation with prescribed triplet structure. This problem introduces a qualitatively different constraint. The grid must realize *exactly* n distinct digit triplets, drawn from the set of all

$$\binom{9}{3} = 84$$

possible triplets. This requirement is global rather than local: it constrains the multiset of patterns used across all 27 Sudoku dimensions (rows, columns, and boxes), not merely their individual validity.

For a given n, the algorithm must implicitly solve several coupled combinatorial subproblems:

1. **Triplet selection:** choose a suitable(!) subset of n triplets from 84 possibilities, a space of size $\binom{84}{n}$.
2. **Multiplicity constraints:** assign frequencies to these triplets so that they cover exactly 27 dimensions while maintaining balanced digit usage.
3. **Compatibility constraints:** ensure that selected triplets can be combined in triples whose union is $\{1, \ldots, 9\}$, corresponding to valid rows, columns, or boxes.
4. **Geometric realizability:** embed these abstract combinations consistently into the fixed 9×9 Sudoku architecture, in particular satisfying column constraints that couple otherwise independent row and box constructions.

Each layer introduces combinatorial growth, but the principal difficulty arises from their *non-linear interaction*. A triplet collection may satisfy balance and compatibility conditions in isolation, yet still be unrealizable as a concrete Sudoku grid due to positional conflicts or column inconsistencies. As n varies, the density of feasible configurations changes sharply, producing irregular feasibility regions and forbidden gaps.

Fundamental source of the complexity gap. The essential distinction is that problem (A) is a pure existence problem in a vast solution space, whereas problem (B) is a conditional existence problem constrained by a high-dimensional global invariant. In problem (A), randomness and backtracking exploit the abundance of solutions; in problem (B), they contend with scarcity. The latter problem effectively requires solving a constrained design problem over a compatibility hypergraph, followed by a constrained embedding into a rigid geometric structure.

This explains why algorithms that perform extremely well for unconstrained Sudoku generation may fail or stall entirely when an exact triplet count is imposed. The resulting combinatorial explosions are not artifacts of implementation, but intrinsic to the global structure of the constrained problem.

4.1 Divide and conquer(?)

Let's split the problem into the two components

(i) Selecting a triplet collection $tc^n \in \mathbb{T}^n$, i.e. a collection of n triplets

(ii) backtracking algorithm that attempts to arrange (some permutation/s of) the triplets in tc^n to a valid grid $\in \mathbb{G}_T^n$

where we will only focus on (i) here. In most cases, it is not clear a priori whether a given tc^n can actually be assembled properly, i.e. whether $tc^n \in \mathbb{T}_g^n$, but we can develop methods to determine whether it definitely won't lead to a valid grid. The idea is that such grids can be sorted out right at the start so step (ii) wouldn't run in circles without finding a solution. For instance, *Digit Completeness*

$$\bigcup_{t \in tc^n} = T_1 \cup T_2 \cdots \cup T_n = \{1, 2, 3, 4, 5, 6, 7, 8, 9\},$$

is a necessary condition for a valid grid, but also a low-level criterion easily checked. There are several more criteria, some of which we will introduce in the following.

The properties discussed in Sects. 4.2–4.4 focus on set-theoretic relations between triplets, whereas 4.5 focuses more directly on triplet-digit interactions. The numbers (counts) for given n are empirical data and as such, completeness is not guaranteed. A full overview for all counts is given in Appendix B.

4.2 Empty Intersections given n Triplets

How many instances of $T_i \cap T_j = \emptyset$? Wherever some triplet occurs, it will have to be accompanied by two further triplets to complete the row (box); these cannot have any overlap. Notice that the reverse does not hold, if for a given grid with a given n, $T_i \cap T_j = \emptyset$, it does not follow that T_i, T_j occur in the same row. For $n = 3$, there are always exactly three empty intersections, but as n grows, not only does the number of empty intersections itself grow, but also the number of possible numbers. The counts for some n are given below, see Sect. 7.3 for the full overview:

5: {6}

6: {8, 9}

9: {9, 11, 12, 13, 14, 15, 16, 17, 18}

17: {41, 43, 44, 45, 46, 47, 48, 49, 50, 51}

25: {74, 78, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99}

4.3 Compatible Trios given n Triplets

How many instances of $T_i \cup T_j \cup T_k = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$? Wherever some triplet occurs, together with two others, they must comprise the digits $1-9$. The counts for some n are given below, see Sect. 7.4 for the full overview:

3 : {1}

9 : {3, 4, 6}

17 : {12, 13, 14}

23 : {17, 18, 19, 20, 21, 22, 24}

27 : {18, 19, 20, 21, 22, 23, 24, 25, 27, 28, 36}

4.4 Pairwise Union Identities given n Triplets

How many instances of $T_i \cup T_j = T_k \cup T_l$ (for $T_i \neq T_j \neq T_k \neq T_l$)? We already discussed this property in Sect. 3.2. The counts for some n are given below, see Sect. 7.5 for the full overview:

3 : {0}

5 : {1}

6 : {0}

9 : {0, 3, 9}

24 : {24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 36, 39}

4.5 Balanced Digit Occurrences

Possible digit distributions? For a fixed triplet count n, the digit count sums to $3n$ – across the triplet collection, not grid. For every grid in the sample, we record the (sorted) 9-tuple of digit frequencies per n: (df_1, df_2, df_3, df_4, df_5, df_6, df_7, df_8, df_9) where df_j means: digit d_j occurs df_j times in the triplet collection.

The motivation for examining this property is this: every digit $1-9$ occurs exactly nine times in a valid grid. If the digit frequencies across the triplet collection are too imbalanced, it puts immense pressure on constructing a valid grid, especially, in connection with the restriction on triplet counts per boxrow and pairwise union identities. For instance, only for $n \in \{3, 5, 7\}$ can there be a digit frequency $df_i = 1$. The digit frequencies for some n are given below:

3 : (1, 1, 1, 1, 1, 1, 1, 1, 1)

7 : (1, 1, 1, 3, 3, 3, 3, 3, 3)
(2, 2, 2, 2, 2, 2, 3, 3, 3)

9 : (3, 3, 3, 3, 3, 3, 3, 3, 3)

22 : (4, 4, 7, 7, 8, 9, 9, 9, 9)
(4, 4, 7, 8, 8, 8, 9, 9, 9)
(4, 7, 7, 8, 8, 8, 8, 8, 8)
(5, 5, 6, 8, 8, 8, 8, 9, 9)
(5, 5, 7, 8, 8, 8, 8, 8, 9)
(6, 6, 6, 7, 8, 8, 8, 8, 9)

(6, 6, 6, 8, 8, 8, 8, 8, 8)
(6, 6, 7, 7, 7, 7, 8, 9, 9)
(6, 6, 7, 7, 7, 8, 8, 8, 9)
(6, 6, 7, 7, 8, 8, 8, 8, 8)
(6, 7, 7, 7, 7, 8, 8, 8, 8)
(7, 7, 7, 7, 7, 7, 7, 8, 9)
(7, 7, 7, 7, 7, 7, 8, 8, 8)

27 : (9, 9, 9, 9, 9, 9, 9, 9, 9)

4.6 Summary

The counts discussed above can be used as reference data for a preliminary filtering stage in the construction algorithm. Given a triplet collection tc^n, the Triplet API provides functions that test whether the count for a certain property falls within the empirically observed possibilities for n listed above and in Appendix B. Two variants are currently implemented: either the count itself must belong to the observed set of values, or it must satisfy a weaker interval condition

$$min \leq count \leq max.$$

The underlying idea is simple: before attempting an actual grid construction, we first eliminate triplet collections that already violate basic structural constraints. This does not solve the combinatorial problem itself, but it prevents the backtracking stage from exploring regions of the search space that are provably hopeless from the outset.

Naturally, an important goal for future work is to replace these empirical reference sets by exact derivations. In particular, we would like to determine whether the "gaps" observed in the data merely reflect the incompleteness of the sample or whether they correspond to genuinely impossible configurations. This is not only a practical question related to optimization; it also concerns the deeper combinatorial geometry of triplets and their interaction within the Sudoku architecture.

A particularly useful criterion is the following feasibility test. Instead of merely probing for digit balance heuristically (for instance via deviation from the median), we explicitly test whether the triplet collection can realize the correct global digit frequencies. The procedure recursively selects 27 triplets from tc^n (with repetition), while ensuring that every triplet is selected at least once, and checks whether each digit $1-9$ occurs exactly nine times overall. In some cases, several valid multiplicity distributions exist; in others, none at all:

```
>>> triplets = [(2,4,8), (3,4,7), (4,6,8), (1,5,6), (6,8,9), (1,5,9), (2,3,7))]
>>> has_feasibleDigitCount(triplets)
[{(2, 4, 8): 4,
  (3, 4, 7): 4,
  (4, 6, 8): 1,
  (1, 5, 6): 4,
  (6, 8, 9): 4,
  (1, 5, 9): 5,
  (2, 3, 7): 5},
```

```
 {(2, 4, 8): 1,
  (3, 4, 7): 1,
  (4, 6, 8): 7,
  (1, 5, 6): 1,
  (6, 8, 9): 1,
  (1, 5, 9): 8,
  (2, 3, 7): 8},   # . . . .  + two more solutions
]

>>> triplets = [(3, 5,8), (1,7,8), (2,5,9), (3,4,7), (6,7,9), (3,7,9), (1,4,7)]
>>> has_feasibleDigitSum(triplets)
[]
```

This criterion is computationally inexpensive and already excludes many impossible triplet collections. However, it remains only a necessary condition. A collection may satisfy all digit-frequency constraints and still fail to admit a valid embedding into the fixed geometric structure of Sudoku. In particular, the procedure does not yet incorporate the positional dependencies induced by columns, box interactions, or permutation compatibility.

The overall picture is therefore somewhat paradoxical. Splitting the problem into a filtering stage and a combinatorial construction stage — steps (i) and (ii) above — is clearly beneficial and substantially reduces the search space. Yet the remaining space is still extraordinarily complex. Randomly generated triplet collections may require a very long time before they pass all currently known tests, and even then there is no guarantee that they can actually be assembled into a valid grid in $\mathbb{G}_T^n$. Feasibility and realizability are closely related, but they are not the same problem.

The alternative approach is more constructive in spirit: instead of randomly probing the search space, one attempts to generate valid core sets systematically, as demonstrated for $n = 3$ in Sect. 2, and then derives grids from these structures directly. Such an approach would likely be far more efficient and mathematically illuminating. At present, however, a general theory of core-set generation for arbitrary n remains out of (our) reach.

5 Summary

[to be summarized . . . .]

6 APPENDIX A: Triplet Puzzles

Triplet construal of a grid can, of course, also be used to construct Sudoku puzzles. The overall task will be to (i) identify the triplet contents, (ii) figure out the individual realizations of the triplets for every position (i.e. the segments). Since the mere triplet template is heavily underspecified, some further clues will have to be given in order for the puzzle to have a unique solution.

- Given a triplet grid:

$$\begin{bmatrix} T_1 & T_2 & T_3 \\ T_4 & T_5 & T_6 \\ T_7 & T_8 & T_9 \\ T_4 & T_5 & T_6 \\ T_7 & T_8 & T_9 \\ T_1 & T_2 & T_3 \\ T_8 & T_2 & T_5 \\ T_9 & T_3 & T_6 \\ T_7 & T_1 & T_4 \end{bmatrix}$$

- and some further clues, e.g.
  - $T_5 = \langle \dots \rangle$ or $\text{segment}_{23} = (\dots)$
  - T_1, T_2, T_3 are residue classes (mod 3)
  - some given digit(s)
  - constraint X
  - the digits in some region sum to n
  - some further clue(s)
  - …etc.

$\Rightarrow$ Normal Sudoku rules apply – gerðu svo vel!

(i.e. convert the triplet grid into a valid 9×9 digit grid)

7 APPENDIX B: Triplet Grid Properties – Counts

7.1 Vertical series abc-reduced

3: { 5184, 10368, 31104, 93312, 139968, 186624, 279936, 559872 }

4: {}

5: { **1679616** }

6: { **1679616** }

7: { 279936, 559872, **1679616** }

8: { 839808, **1679616** }

9: { 279936, 839808, **1679616** }

10: {}

11: { **1679616** }

12: { 559872, 839808, **1679616** }

13: { **1679616** }

14: { **1679616** }

15: { 559872, 839808, **1679616** }

16: { **1679616** }

17: { 839808, **1679616** }

18: { 279936, 559872, 839808, **1679616** }

19: { 839808, **1679616** }

20: { **1679616** }

21: { 279936, 559872, 839808, **1679616** }

22: { 839808, **1679616** }

23: { 839808, **1679616** }

24: { **1679616** }

25: { 279936, 559872, 839808, **1679616** }

26: { **1679616** }

27 { **1679616** }

7.2 Collections of core sets – MECSs

Below are the indices of the individual core sets according to the MECSs that contain them. Recall that a core set's (subscript) index is simply the decimal representation of its compatibility index, which can easily be reconstructed, for instance for core_set$_{424}$:

(24)
```
>>> subscript = 424  # = binary: 110101000
>>> cIdx = np.array([
                     int(bit)
                     for bit in format(subscript, '09b')
                     ]).reshape(3,3)
>>> print(cIdx)
    [[1 1 0]
     [1 0 1]
     [0 0 0]]
```

MECS$_0$: [0, 7, 56, 63, 73, 78, 113, 118, 146, 149, 170, 173, 219, 220, 227, 228, 283, 284, 291, 292, 338, 341, 362, 365, 393, 398, 433, 438, 448, 455, 504, 511]

MECS$_1$: [10, 13, 19, 20, 25, 30, 33, 38, 43, 44, 50, 53, 67, 68, 80, 87, 90, 93, 98, 101, 104, 111, 123, 124, 129, 134, 139, 140, 152, 159, 160, 167, 179, 180, 185, 190, 194, 197, 200, 207, 209, 214, 233, 238, 240, 247, 250, 253, 258, 261, 264, 271, 273, 278, 297, 302, 304, 311, 314, 317, 321, 326, 331, 332, 344, 351, 352, 359, 371, 372, 377, 382, 387, 388, 400, 407, 410, 413, 418, 421, 424, 431, 443, 444, 458, 461, 467, 468, 473, 478, 481, 486, 491, 492, 498, 501]

MECS$_2$: [11, 12, 17, 22, 26, 29, 34, 37, 41, 46, 51, 52, 66, 69, 83, 84, 88, 95, 96, 103, 107, 108, 122, 125, 131, 132, 136, 143, 153, 158, 161, 166, 176, 183, 187, 188, 193, 198, 202, 205, 208, 215, 232, 239, 242, 245, 249, 254, 257, 262, 266, 269, 272, 279, 296, 303, 306, 309, 313, 318, 323, 324, 328, 335, 345, 350, 353, 358, 368, 375, 379, 380, 386, 389, 403, 404, 408, 415, 416, 423, 427, 428, 442, 445, 459, 460, 465, 470, 474, 477, 482, 485, 489, 494, 499, 500]

MECS$_3$: [1, 2, 3, 4, 5, 6, 8, 9, 14, 15, 16, 18, 21, 23, 24, 27, 28, 31, 32, 35, 36, 39, 40, 42, 45, 47, 48, 49, 54, 55, 57, 58, 59, 60, 61, 62, 64, 65, 70, 71, 72, 74, 75, 76, 77, 79, 81, 82, 85, 86, 89, 91, 92, 94, 97, 99, 100, 102, 105, 106, 109, 110, 112, 114, 115, 116, 117, 119, 120, 121, 126, 127, 128, 130, 133, 135, 137, 138, 141, 142, 144, 145, 147, 148, 150, 151, 154, 155, 156, 157, 162, 163, 164, 165, 168, 169, 171, 172, 174, 175, 177, 178, 181, 182, 184, 186, 189, 191, 192, 195, 196, 199, 201, 203, 204, 206, 210, 211, 212, 213, 216, 217, 218, 221, 222, 223, 224, 225, 226, 229, 230, 231, 234, 235, 236, 237, 241, 243, 244, 246, 248, 251, 252, 255, 256, 259, 260, 263, 265, 267, 268, 270, 274, 275, 276, 277, 280, 281, 282, 285, 286, 287, 288, 289, 290, 293, 294, 295, 298, 299, 300, 301, 305, 307, 308, 310, 312, 315, 316, 319, 320, 322, 325, 327, 329, 330, 333, 334, 336, 337, 339, 340, 342, 343, 346, 347, 348, 349, 354, 355, 356, 357, 360, 361, 363, 364, 366, 367, 369, 370, 373, 374, 376, 378, 381, 383, 384, 385, 390, 391, 392, 394, 395, 396, 397, 399, 401, 402, 405, 406, 409, 411, 412, 414, 417, 419, 420, 422, 425, 426, 429, 430, 432, 434, 435, 436, 437, 439, 440, 441, 446, 447, 449, 450, 451, 452, 453, 454, 456, 457, 462, 463, 464, 466, 469, 471, 472, 475, 476, 479, 480, 483, 484, 487, 488, 490, 493, 495, 496, 497, 502, 503, 505, 506, 507, 508, 509, 510]

7.3 Empty Intersections given n Triplets

3 : {3}

4 : { }

5 : {6}

6 : {8, 9}

7 : {9, 10, 11}

8 : {9, 11, 12, 13, 14}

9 : {9, 11, 12, 13, 14, 15, 16, 17, 18}

10 : { }

11 : {23, 24}

12 : {21, 25, 26, 27, 28}

13 : {28, 29, 30, 31, 32}

14 : {24, 26, 28, 29, 30, 31, 32, 33, 34, 35, 36}

15 : {30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41}

16 : {39, 40, 41, 43, 44, 45, 46}

17 : {41, 43, 44, 45, 46, 47, 48, 49, 50, 51}

18 : {39, 42, 44, 45, 46, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57}

19 : {49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63}

20 : {49, 51, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68}

21 : {54, 55, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74}

22 : {66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79}

23 : {66, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86}

24 : {66, 72, 73, 74, 75, 76, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92}

25 : {74, 78, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99}

26 : {78, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 107}

27 : {93, 94, 96, 97, 98, 99, 100, 101, 102, 103, 105, 106, 108, 109, 111, 114}

7.4 Compatible Trios given n Triplets

3 : {1}

4 : { }

5 : {2}

6 : {2}

7 : {3}

8 : {3}

9 : {3, 4, 6}

10 : { }

11 : {7}

12 : {7, 8}

13 : {8, 9, 10}

14 : {8, 9, 12}

15 : {8, 9, 10, 11, 12}

16 : {11, 12, 13}

17 : {12, 13, 14}

18 : {12, 13, 14, 15, 16}

19 : {13, 14, 15, 16, 17}

20 : {13, 14, 15, 16, 17, 18}

21 : {13, 14, 15, 16, 17, 18, 19, 21}

22 : {16, 17, 18, 19, 20}

23 : {17, 18, 19, 20, 21, 22, 24}

24 : {17, 18, 19, 20, 21, 22, 23}

25 : {18, 19, 20, 21, 22, 23, 24}

26 : {18, 19, 20, 21, 22, 23, 25}

27 : {18, 19, 20, 21, 22, 23, 24, 25, 27, 28, 36}

7.5 Pairwise Union Identities given n Triplets

3 : {0}

4 : { }

5 : {1}

6 : {0}

7 : {1, 2}

8 : {1}

9 : {0, 3, 9}

10 : { }

11 : {9}

12 : {9, 10, 11, 12}

13 : {9, 10, 11, 14}

14 : {9, 10, 11, 12, 13, 14, 16}

15 : {9, 10, 11, 12, 13, 14, 15}

16 : {14, 15, 16, 17, 20}

17 : {15, 17, 18, 19, 20, 21}

18 : {15, 16, 17, 18, 19, 20, 21, 22, 27}

19 : {17, 18, 19, 20, 21, 22, 23, 27}

20 : {17, 18, 19, 20, 21, 22, 23, 24, 25, 27}

21 : {18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28}

22 : {22, 23, 24, 25, 26, 27, 28, 29, 30, 32, 34}

23 : {23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34}

24 : {24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 36, 39}

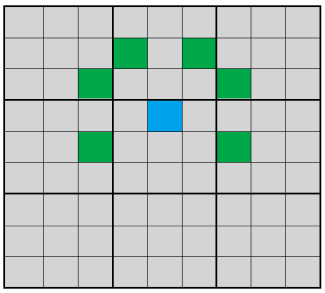
25 : {25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40}

26 : {26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 43}

27 : {27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 45}

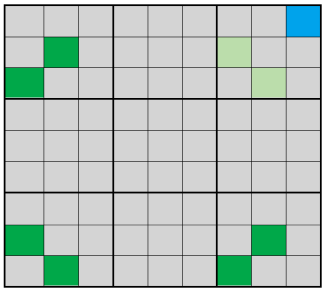
8 APPENDIX C: Enter the Modulo Knight
(a non triplet pertinent excursion)

The classical *Knight's Move constraint* is modeled on the legal movements of a knight in chess, namely two steps in one of the cardinal directions followed by one step orthogonally. Formally, two cells are said to be related by a knight's move if their coordinates differ by $(\pm 2, \pm 1)$ or $(\pm 1, \pm 2)$. The Knight's Move constraint then requires that no two such cells may contain the same digit. The grid below illustrates this situation: the blue cell is related by a knight's move to each of the surrounding green cells.



In the standard setting, however, the effective range of knight moves is curtailed by the boundaries of the grid: some of its potential knight's-move neighbours simply lie outside the 9×9 structure; also, corner cells are not actually affected by the constraint because the affected cells lie inside the same box and couldn't contain the same digit anyway.

To extend this idea, we introduce the *Modulo Knight*. This is a knight operating in a wrap-around, or *overflow*, Sudoku geometry. In this toroidal environment, grid boundaries no longer block movement. Instead, moving beyond an edge simply entails re-entering the grid from the opposite side. Accordingly, a modular knight enjoys the full complement of eight (six!) possible moves, even when starting from a boundary or corner position.



In the above grid, we see that the blue cell retains its classical knight-move neighbors (shown in light green), but now also acquires additional neighbors across the boundary (shown in dark green). The modular knight's constraint thus enlarges the web of disallowed equalities across the grid, making the puzzle structure both richer and more challenging.

8.1 Intermezzo: The travels of a modular knight

The *Knight's Tour* is a classical combinatorial puzzle: can a chess knight move across a board in such a way that every square is visited exactly once, without repetition? This question has fascinated mathematicians for centuries, and countless variations have been studied. Among these variations, one can imagine placing the knight not on an 8×8 chessboard, but on a 9×9 Sudoku grid.

For the sake of exploration and to annoy you, let us propose a new constraint for this journey:

> Traverse a Sudoku grid using knight's moves such that
>
> - every cell is visited exactly once, *and*
> - the start and landing positions of every move belong to **different** 3×3 **boxes**.

At first glance, the task looks forbidding. In the classical setting, a knight starting from one of the four corner positions $(1,1), (1,9), (9,1), (9,9)$ cannot even make the first move without violating the box condition. The geometry of the Sudoku grid constrains the knight so tightly that many starting points appear hopeless. (We have not tested whether / how many other journeys are possible.)

Enter the *modular knight*. By adopting the wrap-around geometry described earlier—where rows and columns cycle back on themselves—the knight is liberated from the edges of the grid. Moving "off the board" simply means reappearing on the opposite side, still within the same row or column. Under these modular rules, the knight's horizon expands dramatically: from *every* starting position on the Sudoku grid, the modular knight can, in principle, attempt the full traversal while respecting both the uniqueness condition and the box constraint. Two journeys are illustrated below, from the center and a corner position (start position marked blue, end position marked green; numbers indicate the steps).

15	66	57	28	9	26	41	22	68
60	46	30	39	51	4	13	77	44
72	53	48	58	31	42	50	61	16
47	59	38	52	49	12	5	43	76
17	71	54	34	1	32	62	19	73
75	35	80	37	63	6	11	24	70
79	55	64	7	33	2	20	74	18
67	81	36	56	27	10	25	69	23
45	29	8	65	40	21	3	14	78

50	13	43	6	18	38	20	36	68
2	63	4	22	44	24	46	51	48
73	30	7	76	9	52	39	56	26
57	3	74	31	23	77	25	47	40
27	72	29	8	75	10	53	78	55
41	60	32	71	16	81	66	34	58
33	28	15	61	11	70	79	54	58
59	42	12	17	80	19	37	67	35
64	5	62	14	21	45	69	49	1

The contrast is striking: what is literally impossible in the classical setting suddenly becomes feasible once the grid is reimagined as a toroidal world. The modular knight is no longer trapped by corners or edges—his travels become truly unbounded. Moreover, this tour is possible from all 81 positions; for illustration, run `PsQ_ModuloKnight.py`, which displays an iteration over all coordinates and the respective tour across the grid.

References

Felgenhauer, Bertram, and Frazer Jarvis. 2005. Enumerating possible sudoku grids. `https://api.semanticscholar.org/CorpusID:16859520`.

Actually a section: trip-rot-trip

3: 3, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 21, 27

5: 8, 15, 16, 18, 19

6: 15, 19

7: 9, 17, 18, 19, 21, 22, 23, 24

8: 15, 17, 18, 19, 20, 21, 22, 23, 24, 25

9: 9, 12, 13, 15, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26

11: 9, 11, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27

12: 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27

13: 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27

14: 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27

15: 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27

16: 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27

17: 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27

18: 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27

19: 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27

20: 8, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27

21: 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27

22: 8, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27

23: 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27

24: 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27

25: 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27

26: 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27

27: 9, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27

core sets: 0, 7, 56, 63, 73, 78, 113, 118, 146, 149, 170, 173, 219, 220, 227, 228, 283, 284, 291, 292, 338, 341, 362, 365, 393, 398, 433, 438, 448, 455, 504, 511
MECS_0

3: 72

6: 1728

9: 17640

12: 38880

15: 112752

18: 104976

21: 93312

27: 3888

core sets: 10, 11, 12, 13, 17, 19, 20, 22, 25, 26, 29, 30, 33, 34, 37, 38, 41, 43, 44, 46, 50, 51, 52, 53, 66, 67, 68, 69, 80, 83, 84, 87, 88, 90, 93, 95, 96, 98, 101, 103, 104, 107, 108, 111, 122, 123, 124, 125, 129, 131, 132, 134, 136, 139, 140, 143, 152, 153, 158, 159, 160, 161, 166, 167, 176, 179, 180, 183, 185, 187, 188, 190, 193, 194, 197, 198, 200, 202, 205, 207, 208, 209, 214, 215, 232, 233, 238, 239, 240, 242, 245, 247, 249, 250, 253, 254, 257, 258, 261, 262, 264, 266, 269, 271, 272, 273, 278, 279, 296, 297, 302, 303, 304, 306, 309, 311, 313, 314, 317, 318, 321, 323, 324, 326, 328, 331, 332, 335, 344, 345, 350, 351, 352, 353, 358, 359, 368, 371, 372, 375, 377, 379, 380, 382, 386, 387, 388, 389, 400, 403, 404, 407, 408, 410, 413, 415, 416, 418, 421, 423, 424, 427, 428, 431, 442, 443, 444, 445, 458, 459, 460, 461, 465, 467, 468, 470, 473, 474, 477, 478, 481, 482, 485, 486, 489, 491, 492, 494, 498, 499, 500, 501 $\text{MECS}_1 + \text{MECS}_2$

7: 1512

8: 2592

9: 1728

11: 17496

12: 3888

13: 21384

14: 15552

15: 50544

16: 34992

17: 64152

18: 23328

19: 93312

21: 42768

core sets: 1, 2, 3, 4, 5, 6, 8, 9, 14, 15, 16, 18, 21, 23, 24, 27, 28, 31, 32, 35, 36, 39, 40, 42, 45, 47, 48, 49, 54, 55, 57, 58, 59, 60, 61, 62, 64, 65, 70, 71, 72, 74, 75, 76, 77, 79, 81, 82, 85, 86, 89, 91, 92, 94, 97, 99, 100, 102, 105, 106, 109, 110, 112, 114, 115, 116, 117, 119, 120, 121, 126, 127, 128, 130, 133, 135, 137, 138, 141, 142, 144, 145, 147, 148, 150, 151, 154, 155, 156, 157, 162, 163, 164, 165, 168, 169, 171, 172, 174, 175, 177, 178, 181, 182, 184, 186, 189, 191, 192, 195, 196, 199, 201, 203, 204, 206, 210, 211, 212, 213, 216, 217, 218, 221, 222, 223, 224, 225, 226, 229, 230, 231, 234, 235, 236, 237, 241, 243, 244, 246, 248, 251, 252, 255, 256, 259, 260, 263, 265, 267, 268, 270, 274, 275, 276, 277, 280, 281, 282, 285, 286, 287, 288, 289, 290, 293, 294, 295, 298, 299, 300, 301, 305, 307, 308, 310, 312, 315, 316, 319, 320, 322, 325, 327, 329, 330, 333, 334, 336, 337, 339, 340, 342, 343, 346, 347, 348, 349, 354, 355, 356, 357, 360, 361, 363, 364, 366, 367, 369, 370, 373, 374, 376, 378, 381, 383, 384, 385, 390, 391, 392, 394, 395, 396, 397, 399, 401, 402, 405, 406, 409, 411, 412, 414, 417, 419, 420, 422, 425, 426, 429, 430, 432, 434, 435, 436, 437, 439, 440, 441, 446, 447, 449, 450, 451, 452, 453, 454, 456, 457, 462, 463, 464, 466, 469, 471, 472, 475, 476, 479, 480, 483, 484, 487, 488, 490, 493, 495, 496, 497, 502, 503, 505, 506, 507, 508, 509, 510 MECS_3

5: 216

6: 432

7: 432

8: 2592

9: 6696

11: 12960

12: 12096

13: 15552

14: 12960

15: 72144

16: 7776

17: 66096

18: 29808

19: 77760

21: 54432

27: 1296

References

Felgenhauer, Bertram, and Frazer Jarvis. 2005. Enumerating possible sudoku grids. https://api.semanticscholar.org/CorpusID:16859520.