In search of the (4, 3, 2, 1)-frequency square Fillomino grid

Ian Tullis, September 2022, informal CS399 projectlet

Introduction

Both the name Sudoku (which is a contraction of a phrase meaning something like "the digits must be alone") and the puzzle form's original English name, Number Place, emphasize *numbers*. Many folks – perhaps especially those who don't like math – would call Sudoku a kind of number or math puzzle. But is it really? The digits 1-9 used in a Sudoku are only meaningful insofar as they are different from each other and easy for humans to keep track of as a set. I have always found it a little unsatisfying that they could be replaced with – for example – different kinds of fruit.[1]

How might we create a Sudoku-like puzzle in which the meanings of the digits *do* matter? I started thinking about matrices in which each row and column have one 1, two 2s, and so on up to some number. For example, here is a 6×6 grid of this form that uses the numbers 1 through 3:

$$\begin{bmatrix} 3 & 1 & 2 & 3 & 3 & 2 \\ 2 & 3 & 3 & 1 & 2 & 3 \\ 3 & 2 & 2 & 3 & 1 & 3 \\ 1 & 2 & 3 & 2 & 3 & 3 \\ 2 & 3 & 1 & 3 & 3 & 2 \\ 3 & 3 & 3 & 2 & 2 & 1 \end{bmatrix}$$

It turns out to be pretty easy to just write down such a matrix by hand, row by row, making sure each new row obeys the constraints imposed by the previous ones. This kind of flexibility is perhaps a bad sign when one ultimately wants to create a puzzle that will be constrained enough to guide or force the solver in some direction.

I've experimented in the past with Sudoku minus their 3×3 boxes – so, just Latin squares with entries missing – and such puzzles either require a lot more "given" entries in each row and column, or are not much fun, or both. A matrix like the above can be viewed as a Latin square in which one of the original values has been mapped to "1", two of the other original values have been mapped to "2", and so on – that is, one can derive many such matrices from a single Latin square. Would basing a puzzle on these even less constrained matrices just magnify the aforementioned problems?

I first tried to check whether someone else had already made this idea work. I asked my

[1] Sometimes this flexibility is welcome, though. See `https://motris.livejournal.com/11096.html` for an example of a beautiful Sudoku by Thomas Snyder that uses letters instead of numbers, and irregular bounded regions instead of 3×3 boxes. I won't spoil the aesthetic surprise here.

friend Wei-Hwa Huang, who has competed in many international puzzle events, if he had seen anything like this, and he (impressively!) recalled and shared the following puzzle from the 2015 World Sudoku Championship:

1234 SUDOKU **95 points**

Place one 1, two 2s, three 3s and four 4s in every row, column, and outlined 10-cell region. Adjacent cells must contain different digits.

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

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

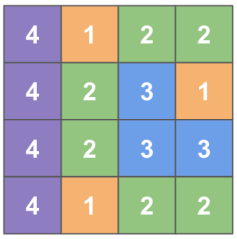
This was a sample puzzle given to contestants before the competition, to demonstrate the format.[2] I attempted to solve it for a while, only succeeding in placing a few digits, before realizing that I had made the classic Ian blunder of not reading part of the directions: "Adjacent cells must contain different digits." This is an additional constraint on top of what I had in mind, and I'm actually glad I tried to solve without it, because it reinforced my worry that there's just not a lot to work with here *without* an extra constraint. The 4s were all but useless, and I had to focus on 1s and 2s (and on the rectangular region that helpfully already contains three 3s) to get anywhere at all.

So it seemed that my original proposal was amounting to (re)designing a less fun Sudoku which still wouldn't be much of a number puzzle per se. Therefore I decided to introduce an additional constraint, much as the authors of the above puzzle did... but since I'm not actually a big Sudoku fan, I instead drew from another logic puzzle type that seemed to fit the "one 1, two 2s... per row/column" constraint well: Fillomino.

[2]Wei-Hwa says that the actual puzzle of this form that appeared in the contest used non-rectangular 10-cell regions, which the sample specifications sneakily did not rule out – a common trick in these competitions.

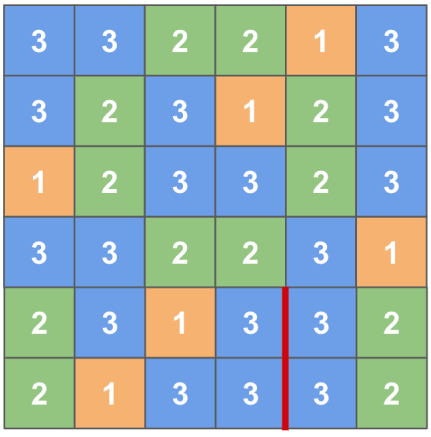
Maybe it'll get easier if we make it harder!

What is a Fillomino puzzle? Let's only worry about completed grids for now, and not about how to turn them into puzzles.[3] The idea is that the grid is divided into regions defined by orthogonal adjacencies of cells bearing the same number, and these numbers also give the *sizes* of those regions. This is much easier to understand visually from a small example like the one below.



Notice that it *is* legal for different regions of the same size to touch diagonally. However, we could not, for example, change the 1 in the top row to a 2, because then we would have a region of 2s (hereafter a "2-region") of size 5.

Naturally, I wondered whether I could impose Fillomino constraints on my earlier "one-two-three" matrix idea. This *almost* works, as demonstrated by this grid that I found by hand. It does have one 1, two 2s, and three 3s in each row and column, and except for the two blue regions that touch at the borders indicated in red, it is a valid Fillomino grid.



Unfortunately, I confirmed (via a program) that there is no solution for these 6×6 "one-two-three" matrices. Intuitively, and handwavily, the problem with adding Fillomino constraints

[3]For the standard rules of the puzzle, see: `https://www.nikoli.co.jp/en/puzzles/fillomino/`

to "one-two-three" matrices is that the 1- and 2-regions are just not large enough to keep the 3-regions from touching each other orthogonally.

What about 10×10 "one-two-three-four" matrices? Then the combined might of the 1-, 2-, and 3-regions could be enough to force the 4s apart... but we may somehow also be able to use the 4-regions to keep the 2-regions apart and the 3-regions apart. It's a nice idea, but it certainly does not seem tractable to explore by hand; my own attempts broke down after about five rows, at which point the constraints within each column hadn't really even started to kick in yet.

At this point I realized that I would need to lean heavily on a computer, or on past research, or both. Had these one-two-three-etc. matrices been studied before?

Time for some 21st century mathematics

What if we *generalize* the one 1, two 2s, etc. per row/column constraint? I can imagine a real-world application here: an experimenter – who is, say, trying out some treatments on different plants, and growing them all together in a greenhouse – might plausibly need a grid-based design in which there are exactly k_i instances of the i-th treatment in each row and column.

I tried and tried to search for research on this, but I couldn't seem to come up with the right terms. I ultimately struck gold with the following method, which I somewhat sarcastically call 21st century mathematics.[4]

1. Manually determine the number of matrices with one 1 and two 2s in each row and column (6).
2. Write a program to determine the numbers of matrices with one 1, two 2s, and three 3s in each row and column; this turns out to be 5450400.
3. Type 1, 6, 5450400 into the Online Encyclopedia of Integer Sequences. Get a hit: `https://oeis.org/A269516`
4. Follow breadcrumbs from the citations on that page to learn that there are in fact papers about these matrices, and they are called *frequency squares*.[5]

So it turned out that I'd been thinking about (4, 3, 2, 1)-frequency squares, and then trying to impose Fillomino constraints on those.

[4] Another form of this is: see if Terry Tao has blogged about it. I was able to help one of Greg's other students out a lot at one point by finding the right post!

[5] In the late 19th century, P.A. McMahon called these *quasi-Latin squares*, but this seems to now mean something different.

The literature on frequency squares is surprisingly sparse, and – as far as I can tell – unfortunately not particularly pertinent for my purposes here. Dénes and Mullen [DM93] explored the aforementioned one-to-many relationship between frequency squares and Latin squares, and derived a formula for counting Latin squares of a given size (which quickly becomes intractable in practice). Most other research seems to focus on frequency squares in which the subparts are of the *same* size – understandable, since experiements are more likely to be designed this way.

More recently, Cavenagh et al. [CMW21] focused on finding constructions for mutually orthogonal sets of such squares – i.e., sets such that when the squares are superimposed, each combination of symbols occurs the same number of times. These sets are foundational to (and extend the idea of) *Greco-Latin* squares in which each cell has a different ordered pair of symbols, and the sets of first/second entries, when taken in isolation, behave as Latin squares. There are connections here with Hadamard matrices and potentially with coding theory, but nothing that I could distill without getting too sidetracked from my specific quest.[6]

Do (4,3,2,1)-frequency Fillomino squares exist?

When searching for (4,3,2,1)-frequency squares with the Fillomino property, it is possible to use mathematical reasoning to impose *some* constraints. For instance, if there is a 2-region that crosses two columns, there must be another 2-region crossing the same two columns. The alternative would be to use *part* of another 2-region to provide the remaining 2 that, say, the leftmost of those two columns needs. But then the other column of that 2-region would require *another* partial 2-region, and so on, until we hit a grid boundary and run out of options.

But that kind of thinking only took me so far, and this is really a job (albeit a very grungy one) for a backtracking program. I wrote a program in Python (with PyPy to make it fast enough) that first generates all $\binom{10}{4,3,2,1} = 12600$ possible rows, then tries to use each of these in turn as the first row of a grid, and investigate from there.

To explore a partially filled grid, the code tries placing each of the possible rows as the next row, and checks each possibility for violations of the rules (too many of one number in a column, or a k-region with more than k cells, or a completed k-region with fewer than k cells). For 1-, 2-, and 3-regions, these checks are implemented as a bunch of careful `if`-

[6] One of the authors of that paper – and indeed, a person to whom all modern Latin square research roads seem to lead – is Ian Wanless. He was the Ph.D. supervisor of Darcy Best, my Google Code Jam colleague with whom I wrote a Code Jam problem about constructing Latin squares with specified traces – `https://codingcompetitions.withgoogle.com/codejam/round/000000000019fd27/0000000000209aa0`. I emailed him to share a couple results of this projectlet and to ask about connections with coding theory. I'll let you know if that goes anywhere!

statements. For 4-regions, which can sprawl in ways that are tedious (and error-prone) to check with casework, the code uses BFS to find the regions' sizes and check whether a region has been completed prematurely. If a newly placed row obeys all the rules, then the code tries to place the next row below that, and so on; otherwise, it backtracks. (I handled the backtracking implicitly via recursive calls, but it would have been more efficient to do so explicitly.)

The code allows for partial regions (e.g., a set of two orthogonally adjacent 3s), presuming that they could be completed later on. This poses a problem when we get down to the last row or two – there might still be an unfinished region even in a seemingly complete grid – so there is one final BFS check of each region to verify that a potential solution is legit.

At first it seemed that this might not be tractable (or that there were no solutions), because the code spent a seemingly arbitrarily long time exploring a particular starting row. To encourage more exploration, I had the code iterate through all possible starting rows, but only spend up to 7 seconds exploring each. This yielded a solution before long, to my great excitement, and more dribbled out thereafter.

Here are some examples; the patterns at the bottom show the locations of the 4-regions.

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

* . * . . . * * . .    * * . . * * . . . .    . . . * . . * * * .
* . * . . . * * . .    * . . . * * . . . *    * . . * . * . * . .
* . * . * . . . . *    * . . * . . . . * *    * * . * . * . . . .
* . * . * . . . . *    . . * * . . * . * .    * . . * . * . . . *
. . . . * * . . * *    . * . * . . * * . .    . . . . . * . * * *
. * . * . . * * . .    * * . . * . . * . .    . * * . * . * . . .
. * . * . . * * . .    . * . . * * . . * .    * * . . * . * . . .
. * . * . * . . * .    . . * . . * . . * *    . . * . * . * * . .
. * . * . * . . * .    . . * . . . * * . *    . . * . * . . . * *
. . . . * * . . * *    . . * * . . * * . .    . * * . . . . . * *
```

I personally find solutions less interesting (and think they would make less satisfying puzzle fodder) when they have:

- many linear 4-regions (and 3-regions, for that matter)
- fewer regions that cross columns
- obvious internal symmetry

The leftmost solution has all of these shortcomings. The middle solution is perhaps better, with no linear 4-regions and with less clear symmetry, but notice that no regions cross between the second and third columns, or the fourth and fifth, and so on. That is, the solution (somewhat disappointingly?) consists of five 10×2 "stripes"... although if we turn our heads 90 degrees, this problem goes away. The rightmost solution is a bit better on this front despite the linear 4-regions, with vertical "stripes" of sizes 3 and 4.

I wondered whether the stripiness might be a side effect of the way I was searching for solutions by spending only 7 seconds for each possible starting row. Maybe these "columnar" solutions are just easier to find quickly? So I changed the timeout to 600 seconds to allow for more exhaustive exploration of a smaller number of starting states (and then left the code running while I went to the grocery store), but this didn't yield any meaningfully different solutions. I think it's likely that there is a bug in my code that somehow rules out the 90-degree-tilted versions of these solutions.[7]

A nice surprise, and a puzzle to boot!

While watching solutions gradually appear over the course of a few hours, I spotted one with such an interesting and unexpected property that I had to make a puzzle out of it. It appears on the next page, if you'd like to try solving it! It uses standard Fillomino rules, plus the additional constraints.

Potential further work

Although (4,3,2,1)-frequency squares with the Fillomino property are rare, I would estimate that there are still hundreds or thousands (or even tens of thousands) of them in all.[8] This might be enough "meat" for a new puzzle form. If I had connections with the Nikoli puzzle magazine, I would reach out!

I have not thought terribly deeply about the combinatorics of these squares, but if I were to pitch this as a novel puzzle type, it would be good to know approximately or exactly how many solutions there are. I am reminded of another Code Jam problem[9] that I wrote, which

[7] The code is fairly rough, but I'm happy to share it upon request!

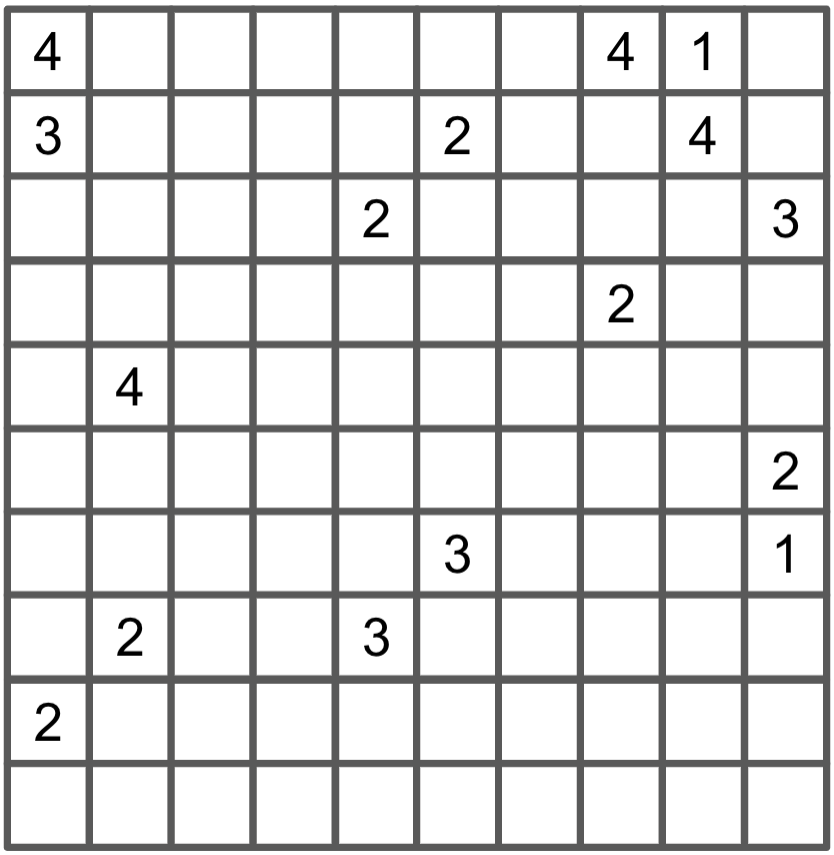
[8] Enumerating them would have taken a bit too much time for this paper.

[9] `https://codingcompetitions.withgoogle.com/codejam/round/0000000000433651/000000000043373a`

has some superficially similar properties and also boils down to stacking different "stripes" in ways that do not conflict. There, Burnside's Lemma can be used to count the number of solutions. In our present case, though, there is an additional global constraint on the numbers of 1s, 2s, 3s, and 4s in each column, so I do not think the same method would work; the amount of information to keep track of seems to be too much even for dynamic programming.

I see no reason why (5,4,3,2,1)-frequency squares (and beyond) should not also exist, and these could also be fun to explore and turn into puzzles. I note, though, that even keeping track of four 4s is a lot, and the 5s might not end up being as useful or fun to work with while solving (try the puzzle below to get an idea of what I mean).

Enjoy the puzzle![10]



Place a 1, 2, 3, or 4 in each cell such that:

Each row and column contains one 1, two 2s, three 3s, and four 4s.

Any region of cells (defined by orthogonal borders) that share a number is of size exactly equal to that number.

All regions of size 4 are an "L" shape (possibly reflected and/or rotated).

[10]When designing this kind of logic puzzle, it is considered clever / aesthetically pleasing to make the given numbers have rotational symmetry, but I either am not at that level of logic puzzle construction yet or have never seen the appeal of that convention. This is just a sample of how to turn this particular grid into a (hopefully satisfying) puzzle, anyway!

Acknowledgments

I thank Darcy Best and another anonymous math friend for being great and helpful sounding boards about this project. Wei-Hwa Huang kindly provided the 2015 WSC puzzle example.

Also, thank you again, Mary, for agreeing to supervise a unit of CS399 – I can't emphasize enough how much of a big help it was to be able to get a parking pass during summer teaching (and how much I enjoyed starting to learn about design theory!)

References

[CMW21] Nicholas J. Cavenagh, Adam Mammoliti, and Ian M. Wanless. Maximal sets of mutually orthogonal frequency squares. 89(3), 2021.

[DM93] József Dénes and Gary L. Mullen. Enumeration formulas for latin and frequency squares. *Discrete Mathematics*, 111(1-3):157–163, February 1993. Funding Information: Correspondence to: Jozsef Des, Csaba u. e 10 V 42, 1122 Budapest, *This author thanks the National Security Agency for partial MDA904-87-H-2023.