GEORGE MASON UNIVERSITY
COMPUTATIONAL & DATA SCIENCES

Project # 5

Sudoku

December 8, 2020

1 HYPOTHESIS

There are three parts to our destruction of the Sudoku industry:

1. Build a program to solve any puzzle.
2. Build a program to create new puzzles.
3. Prove that there is actually only one Sudoko puzzle.

We have accomplished the first two in class, and this project will begin the journey into the third. Here's the hypothesis:

> If there is only one Sudoku puzzle, then every puzzle is derived from the trivial puzzle.

If that is true, then we should be able to start with *any* puzzle and then work our way back to the trivial puzzle.

In the lecture, we started with the trivial puzzle and used process such as cell swap, flip, rotation, etc. to create new puzzles. There is a path from the trivial puzzle to the final puzzle.

In this project, we are to start with the final puzzle and work our way back to the trivial puzzle.

2 OLD MAN'S THEORY

I am not going to tell you how to solve this project. There are numerous ways to approach this. I am going to tell you about how I solved it, but I am not going to give away the big secrets. Feel free to follow this method or go in a completely different direction.

I used an optimization process which is basically this: Try something, and if it gets you closer to a solution, then keep this step.

Let's go into some detail. The trivial puzzle is named P_0 and the final puzzle is named P_F. We have steps that we can take to change the puzzle (swap, rotate, etc.) and we will call these S_i where i is just the iteration.

A process to create a new puzzle can be described as

$$P_0 \rightarrow S_1 \rightarrow P_1 \rightarrow S_2 \rightarrow P_2 \rightarrow S_3 \rightarrow P_F$$

The first step S_1 converts the trivial puzzle P_0 into the intermediate puzzle P_1.

In this project, we will work in reverse. Start with P_F and try to find the steps that get back to P_0. However, we don't know what the steps are or what the intermediate puzzles are. All we have at the beginning is P_0 and P_F.

In the first step of solving this process, consider $P_2 \rightarrow S_3 \rightarrow P_F$. We don't know, P_2 or S_3. What do we know?

The optimization process starts with the idea (or hope) that P_2 is more similar to P_0 than P_F is to P_0. Basically, each step backwards makes a puzzle that is more similar to the trivial one.

However, there is a complication. What does "more similar" mean? We need a way of measuring the similarity of two puzzles. At first, this may seem easy - just count the number of cells that are the same in each puzzle.

Like most things in life - easy doesn't work. Consider the case of $P_0 \rightarrow S_1 \rightarrow P_1$. This is just one step. If that step were to swap cells (exchange the 4's an 9's) then most of the cells are unchanged. P_0 is similar to P_1. BUT, if S_1 was to rotate the puzzle, then almost all of the cells will be mismatched. (There might be a cell or two that coincidentally kept the same value.)

Thus, just measuring the number of cells that match between two puzzles is wholly insufficient. In my solution, I measured two types of similarity, of which one was this mismatch. Remember, I did say, that I wasn't going to reveal all secrets of my approach.

The optimization approach uses this measure of similarity. Start with P_F and try several changes such as different cell swaps or rotations. Compute the similarity for each change. Pick the one that makes P_2 more similar to P_0.

Example. Start with P_F. Try several changes:

- $P_a \rightarrow$ Swap Cells (4,9) $\rightarrow P_F$
- $P_b \rightarrow$ Swap Cells (1,3) $\rightarrow P_F$
- $P_c \rightarrow$ Rotate 3 times $\rightarrow P_F$
- $P_d \rightarrow$ Flip up and down $\rightarrow P_F$

Compute the similarities of P_a, P_b, P_c and P_d to P_F. Pick the one that is most similar. That defines what step S_3 and P_2 are. For example, if the best similarity was from the swap of (1,3), then S_3 is swap cells (1,3) and P_2 is the puzzle that we get by swapping those cells from P_F.

Repeat the process for $P_1 \rightarrow S_2 \rightarrow P_2$. Repeat the process until you get back to P_0.

The example used just four possible changes, but in reality there are many. There are 36 different possible cell swaps, 3 different rotations, 2 types of flips, and so on. In the program, all of these were tried.

I did run into the problem of being stuck in a *local energy minimum*. This means that S_2 and S_3 were opposing functions. For example S_3 was determined to be the swap of values (4,9) and S_2 was determined to be the swap of values (9,4). The two cancel each other out the project could not be solved. So, I had to come with a plan to not get stuck in this cycle. Remember, I did say that I wasn't going to reveal all my secrets.

Finally, you should know that the project was solvable. I did get a set of steps to go from P_F to P_0. However, it was not the original set of steps used to create P_F. I created P_F using 6 steps, but when I tried to reverse engineer it, the process took 25 steps. So, I found a path from P_F to P_0, but I didn't find the optimal path. Then again, it doesn't matter, because all I wanted to do was to find any path.

3 THE PROJECT

Start with this puzzle.

```
[[9 5 3 4 7 1 6 8 2]
 [2 6 8 9 5 3 4 7 1]
 [1 4 7 2 6 8 9 5 3]
 [4 7 1 6 8 2 5 3 9]
 [6 8 2 5 3 9 7 1 4]
 [5 3 9 7 1 4 8 2 6]
 [7 1 4 8 2 6 3 9 5]
 [8 2 6 3 9 5 1 4 7]
 [3 9 5 1 4 7 2 6 8]]
```

This puzzle was created from the trivial puzzle using only cell swaps, flips, rotates, and swapping of rows. I did not use swapping of columns or the swapping of blocks of rows or columns. I did use some functions more than once.

Thus, you don't need to use the swapping of columns, blocks of rows, or blocks of columns. You can if you want.

Create a process that determines S_1, S_2, ... , S_k, where k is the final step). In my case, $k = 25$, but it will probably be different for you.

You are free to use any method that you deem best.

4 Reporting

In your 5 minute video, please include an overview of how your process works. In my case, I would say that I used the optimization process, and I would describe how I measured the similarity of two puzzles and how I circumvented the local energy minimum problem.

Put in your a code a function that runs all of the S_i steps. For example,

```
triv = sudoku.TrivialSolution()
b1 = np.rot90(triv,3)
b1 = np.fliplr(b1)
b1 = sudoku.SwapCells(b1,1,2)
b1 = np.rot90(b1,3)
. . .
b1[[0,2]] = b1[[2,0]]
```

At the end of the code, the `b1` variable is the trivial puzzle. By the way, these are just random steps and have nothing to do with the actual solution.

You do NOT have to fully automate your process of finding out what the steps are. You just need to get to the prescription like that shown in the previous code.

5 Final Thoughts

You can exchange ideas with other groups. Just don't give away your secrets. We can also set up times for groups to have an open discussion on Zoom. Just let me know, and I will be glad to set it up.

This is a step in the process of the destruction of the Sudoku industry. The next step would be to apply our successful process to *any* puzzle to get back to the trivial puzzle. But that is not part of this project. We are just going to create the process for a puzzle that is known to have derived from the trivial puzzle. Someday, this hypothesis will be fully proven.