Independent Study Report

Becker Mathie

April 2021

Puzzle and backtracking solution

My first task was to implement the recursive backtracking solution outlined in this research paper. Some terminology in this research paper will be helpful for talking about the puzzle. mainly the puzzle is made up of 9 pieces, each with a picture on every side. The pictures on the sides of the pieces connect with other pieces to form patterns. Arranging the 9 pieces into a 3 by 3 grid with all pictures matched into their respective patterns solves the puzzle. Here is the example puzzle I modeled my program after.

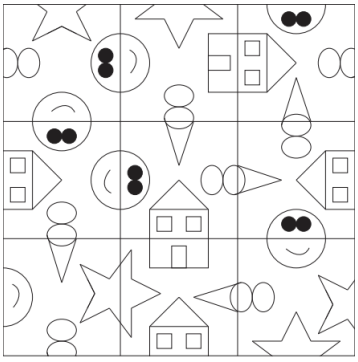

Figure 1: Example Puzzle

I was able to implement the backtracking solution representing the patterns of the puzzle with characters A through D. The capital letter is used for the top picture of the patter, and the lowercase letter is used for the bottom of the picture.

Star = Aa, Cone = Bb, House = Cc, Face = Dd

Implementing the backtracking algorithm with patterns represented with characters, my programs output looks something like this.

```
|-------------------|
|   B   |   C   |   a   |
| D   b | B   D | d   A |
|   C   |   b   |   B   |
|-------------------|
|   c   |   B   |   b   |
| A   B | b   C | c   C |
|   d   |   D   |   A   |
|-------------------|
|   D   |   d   |   a   |
| C   d | D   B | b   B |
|   B   |   a   |   d   |
|-------------------|
```

Figure 2: Puzzle Output

The other task was to represent the solution to the puzzle using the graph theoretical approach explained in depth in this research paper. for this I used matplotlib. The 8 nodes of the graph, labeled with the characters a-d both upper and lowercase, represent the 8 possible pictures. The edges of the graph are the inner corners of the solution, for example, the upper left piece has one corner involved in the solution with pictures b and C. We track the pictures clockwise, so the edge we would see in the solution graph is b to C. It is easy to see which edges correspond to which pieces as they are colored according to the piece.

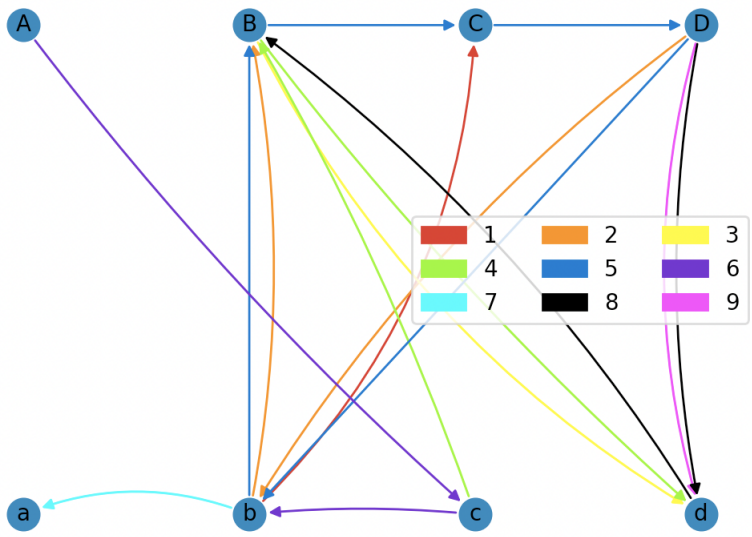

Figure 3: Example Graph

Maximizing the center

The research paper that provided the graph theoretical approach also provides a method of "maximizing the center". For a 3 by 3 puzzle the center piece plays a unique role in that it connects with 4 other pieces rather than 2 or 3. By counting the occurrences of each picture and assigning this as the pictures index, we can then assign a value index to each piece b counting the indices of the pictures appearing on the piece. You can then check for solutions putting the best potential middle pieces in the middle and avoid a lot of backtracking.

```
Pieces sorted in order of potential to be middle piece (paired with value index)
[('Bdba', 21), ('bCDB', 20), ('CDbB', 20), ('DbBC', 20), ('dBaD', 18), ('BdAc', 18),
 ('CAcb', 18), ('BCDd', 17), ('ABda', 16)]
```

Figure 4: Pieces sorted by value index

The question was whether identifying potential middle pieces to avoid backtracking would result in a significant speed up. I added some code to measure the execution time of the backtracking algorithm and recorded some results both using and not using the method of maximizing the center.

Run	Normal	MTC
1	0.05226	0.03665
2	0.00350	0.03701
3	0.05193	0.03738
4	0.05546	0.03754
Average	0.04219	0.04123

Looking at the results, the average was about the same, but the method of maximizing the center is more consistent. In fact run 2 was the only reason the averages were similar. Normal had a very fast run because it got lucky and randomly selected the correct middle piece first while the method of maximizing the center checks the correct middle piece after a couple of other ones that technically had higher potential, it is not perfect.

Larger puzzles

As some additional exploration of this puzzle, I constructed a 4 by 4 puzzle in order to test how the efficiency of the backtracking algorithm scales for larger puzzles. Here is the 4 by 4 puzzle I constructed. The formatting becomes very tedious, but here are the pieces as strings in there correct positions.

```
cBdA baBd CDbB BbaC
bBCD DdBa CAcb DBbC
CaBc ACdB cbcd ADdb
adcb ABab dBCD BdaA
```

Figure 5: 4 by 4 puzzle

An interesting thing to note about the 4 by 4 puzzle is that it really has 4 middle pieces rather than 1. In fact counting for the value indices and sorting the 16 pieces by potential to be a middle piece almost always lists the 4 middle pieces first. Anyway, here are some test results measuring execution time solving the 3 by 3 and the 4 by 4

Run	3 by 3	4 by 4
1	0.01832	3.19725
2	0.00872	1.96208
3	0.05721	1.23899
4	0.03720	2.04518
5	0.02396	0.12017
Average	0.029082	1.712734

As you can see from the results the larger puzzle is quite a bit slower, sometimes taking up to 3 seconds. So while the algorithm does work on larger puzzles, I think increasing the dimension of the puzzle would quickly cause the algorithm to take too long to be very convenient.

Two sided puzzles

As some more exploration I built a 2 sided puzzle and represented it using the graph theoretical approach like above to see if any interesting patterns emerged. For this I constructed a 2 by 2 puzzle to avoid some headaches and more easily be able to interpret the solution graphs. Here is an example two-sided puzzle solution.

```
dbaB cbCA
BCDb dBCD


Abaa aACA
BCaC dcAD
```

Figure 6: Example two-sided puzzle

Once the program solves the 2 by 2 puzzle one method of generating a two sided puzzle is to just to add the same pictures to the other side, or some have some rule for changing the images for the other side so its a little more interesting. Another method is to generate a random picture for each link in the solution and put it on the other side. The puzzle above was generated with the random method. Here is what the solution graph of the two-sided puzzle above looks like.

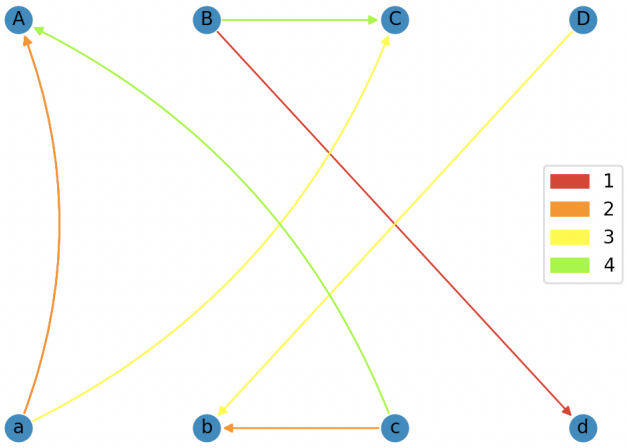

Figure 7: Example two-sided puzzle solution graph

The research paper outlining the graph theoretical approach provides us with 3 key rules when determining if you have a valid solution based on the solution graph.

1. Each edge is a different color.
2. The in-degree of each vertex is equal to the out-degree of its complement.
3. If $X \to A \to Y$ is a directed path in Gs(P), then Y must be the complement of X.

a two sided puzzle is just two different solutions graphs each following the rules layered on top of each other.

Conclusions and Future Work

The last bit of exploration I was planning to look into was regarding adding conditions to the algorithm in order to handle solving puzzles with repetitions. This problem is very difficult and currently unsolved. The research paper outlining the graph theoretical approach talks briefly about this issue on page 14.

From implementing the algorithm to solve the puzzle and then doing some additional exploration with larger and two-sided puzzles I have a few takeaways. Having first attempted to solve this puzzle with no computer assistance and failing for hours as there are some $4^9 \cdot 9!$ different ways to arrange the 9 pieces, I have some more appreciation for the power of a backtracking algorithms ability to solve complex problems and the implications it has for Artificial Intelligence. I also learned a lot working through understanding the graph theoretical approach of representing the puzzle and its solutions. The patterns and rules used to determine the valid solution graphs in order to prove the correctness of the algorithm is very interesting. It is also important to acknowledge that while the algorithm is good, the brute force nature of it causes the execution time to grow immensely. This image from thisresearch paper shows the number of nodes at each level of the search, up to the 9 pieces.

Level	Number of nodes
Root	1
1	9
2	288
3	8,064
4	193,536
5	3,890,720
6	61,931,520
7	743,178,240
8	5,945,425,920
9	23,781,703,680

Figure 8: number of nodes per search depth

We can imagine the numbers becoming to big for the algorithm to be of use, and began to see this with the 4x4 puzzle sometimes taking several seconds to finish. This sort of situation is the case for a lot of puzzle solving algorithms.

I had a lot of fun working on implementing this algorithm to better understand backtracking and graph theory.