RPM Milestone 2

David Adamashvili
dadamashvili3@gatech.edu

***Abstract*— In this journal entry, I will be discussing my approach to solving 2x2 Raven's Progressive Matrices (henceforth abbreviated as **RPM**). The solution will be mostly based in heuristic algorithms, such as geometric transformations.**

1 AGENT DESCRIPTION

The implementation of the agent in based on a few observations. First of all, 2x2 problems contain row-wise and column-wise analogies. Meaning, "A is to B as C is to D" (row-wise) and "A is to C as B is to D". Second of all, almost all problems in the first problem set can be solved using **geometric transformations** and **pixel counting**.

So, what my agent does is, it looks for all the ways figure A can be transformed into the figure B and stores these transformations, then it substitutes D with each of the answers and sees if C becomes D after applying any of the abovementioned transformations. We call this the **row-analogy**. It also does the same for **column-analogies**, namely transforming A into C, and then seeing if we can get D by transforming B. All of these are done using the python image library **Pillow** and **numpy**. Note that, doing nothing is also considered a transformation, albeit an empty one.

Now I will list some of the transformations my agent checks.

1.1 Equality

Image equality is established very simply by first checking if the dimensions of the two images are the same, and then counting all of the pixels where both of them are equal (basically finding their intersection), accounting for a 3% error, because sometimes extra pixels can be added (or lost) after rotations.

1.2 Horizontal and vertical reflections

Thankfully, Pillow has built in support for horizontal and vertical image reflections using **ImageOps.mirror** and **ImageOps.flip** functions respectively. After applying these, image equality is checked as mentioned before.

1.3 Rotations

Pillow image objects have methods for rotating them. The rotate method is used in the following way: **image.rotate(deg,fillcolor="white")**, where **deg** is the degrees by which the image needs to be rotated by. Fill color is used because after rotating, some pixels turn into empty space. We need to fill these with whitespace.

In our case, checking rotations of 45, 90, 135, 180, 225, 270 and 315 degrees should be enough.

1.4 Additional transformations (implemented, but not yet in use)

In addition to the abovementioned simple geometric transformations, I also implemented other heuristics, such as **LOGICAL_AND**, **LOGICAL_OR** and **LOGICAL_XOR** of two images. They are exactly what they sound like – pixel-wise logical operations. A black pixel corresponds to a true value, whereas a white pixel represents a false value. Counting black pixels is another heuristic I implemented. This will come in handy when looking for arithmetic and geometric progressions of black pixel values.

2 AGENT PERFORMANCE

My agent received a 9/12 score on the basic problem set B. It also solved 6 out of the 12 problem set B challenge problems. The agent received a 100/100 score on gradescope, meaning it also solved at least 5 of the 12 test problems.

2.1 Agent success

My agent is currently equipped to solve problems where row-wise and column-wise analogies are geometric transformations from the list of the ones mentioned above. These are: image equality, horizontal and vertical reflections and rotations (multiples of 45 degrees).

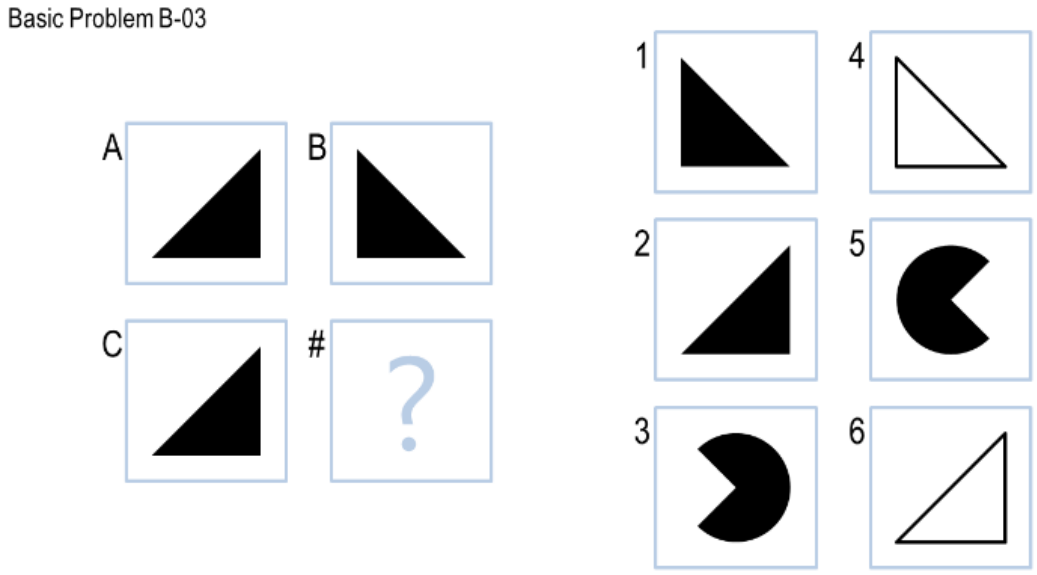

Figure 1 - **Horizontal reflection**

For example, my agent returns the correct answer for the problem on figure 1 by using **ImageOps.mirror**. In much the same way, if the horizontal analogy (geometric transformation from A to B) and the vertical analogy (geometric transformation from A to C is one of those that I mentioned before, then the agent solves it correctly).

2.2 Agent struggles

The agent struggles in quite a few cases. One case is when the analogy is to fill a given shape.

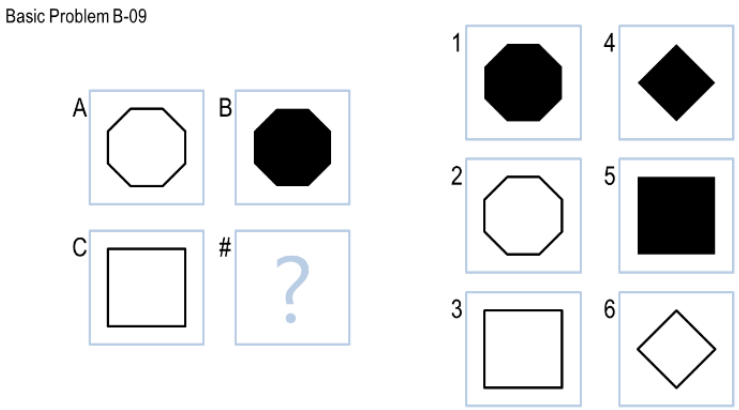

Figure 2 - **Fill analogy**

For example, in figure 2, the correct answer is 5, because we go from A to B by filling in the insides of the octagon. However, my agent is not yet equipped to handle these kinds of problems.

The other types of problems my agent is not able to solve currently are ones where a certain amount of pixels are being added when going row-wise and column-wise.

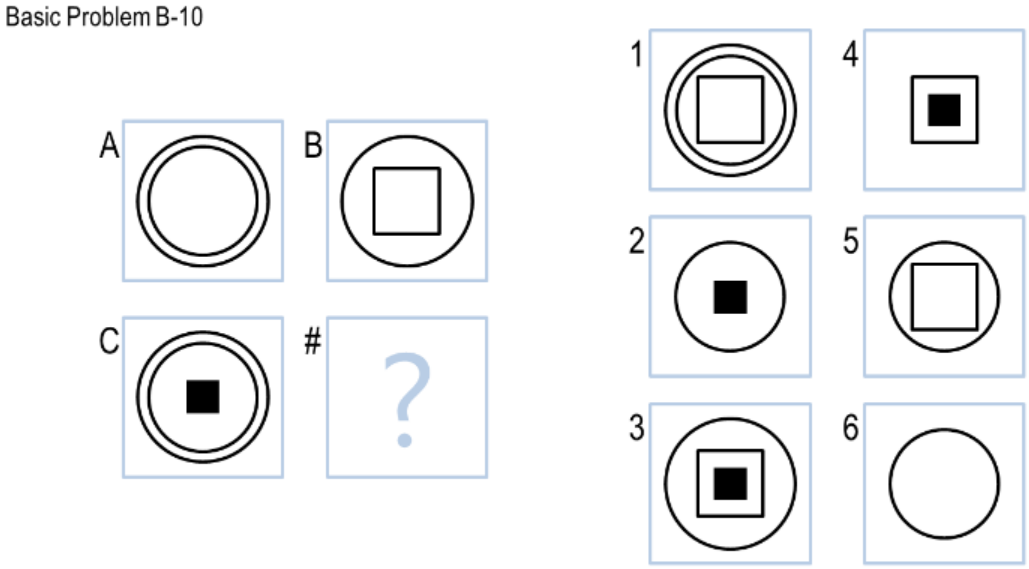

Figure 3 - **Pixel addition**

One might think that we will need verbal representations of semantic networks to solve such problems as the one in figure 3, but in fact, they can be approached in a very simple way.

Going from A to C, a black rectangle is added to the center of the image. Following this thought, going from B to D, 3 must be the correct answer. We can actually look at this problem in a different way, by counting pixels added to the picture. This will be very easy to add to the agent, but since my agent is already getting a full score, I will do this at a later time (probably when working on the next milestone).

2.3 Agent efficiency

The agent turned out to be a bit slow, on the order of tens of seconds (10-20 seconds on my machine) for the entire problem set. Loading these images from the disk into RAM also takes time. The problem type does not determine how long the algorithm runs, as it follows the same steps regardless of the problem. It only depends on the image resolution.

Since there are a fixed number of transformations that I'm testing, and also a fixed number of potential answers to be tested (6 answers in the case of 2x2 RPM problems)

In fact, the asymptotic complexity of the agent (per problem, not the whole problem set!) is O(N*M), where N and M are the dimensions of each image, with a really big constant factor.

3 FUTURE PLANS

3.1 Agent improvement

The problems on which my agent struggles on are going to need a new approach utilizing semantic networks to solve. These problems deal with the relative positioning of shapes inside the figures. These figures need to be identified using **OpenCV** and a semantic network using verbal representations needs to be created. Additionally, we need to be able to distinguish hollow and filled figures, scaled out and shrunk shapes, shapes that were rotated and/or translated relative to other shapes.

I think that at some point in the future I will need to implement this approach regardless of whether I want to improve my score on 2x2 problems or not, so I might as well start practicing on these simpler types of problems.

Alternatively, if I just want to get more points on the basic problem set, I can just implement the pixel addition heuristic mentioned previously in this journal entry.

3.2 Agent generalization

This approach to RPM problems does not generalize well to 3x3 problems. This is not to say that my current work will not be useful. On the contrary, there are

some 3x3 problems where all the images in a row are permutations of other rows for example. Here, my image equality function will come in handy. A lot of problems can be boiled down to adding a certain amount of pixels when moving from one figure to another. Here I can use my black pixel counting function.

But, regardless of the heuristics I use, I think that verbal representations of the shapes in the 3x3 figures will be a vital part of my final project solution. There are lots of problems where just comparing two figures is not enough – we will also need to look at the relative positions of the shapes inside these figures.