RPM Milestone 2 Journal

Luke Nam
lnam8@gatech.edu

***Abstract*—** This is the second RPM Milestone Journal that documents the early strategies and approaches in creating an AI agent that can solve Raven's Progressive Matrices (RPM) problems.

1 RPM MILESTONE 2

Raven's Progressive Matrices (RPM) are a set of 2x2 and 3x3 visual matrix problems that are used to measure human intelligence. RPM Milestone 2 revolved around 2x2 problems that had simpler transformations such as exact shapes and reflections.

1.1 Agent Function

The general approach to my agent's design was to use a generate-and-test problem solving strategy combined with a very simple production system framework. The main workflow is to load the figures from the problem, which include images of A, B, C, and solution options 1, 2, 3, 4, 5, 6. I used the cv2 package to read each image and store it in a dictionary where the key represented the image name, and the value represented a numpy array that represented the pixels of the image. This image dictionary was run through four different methods to come up with a solution to each problem in Problem Set B.

The four different methods represent four different types of transformations that my AI agent can detect. The four transformations represent a very simple production system that have different production rules to detect exact shapes, horizontal reflection, vertical reflection, and rotation. The agent goes through each of these production rules and returns a solution (action) when a production rule

is met. The production rules are ordered from most obvious to least obvious based on the transformation type seen in Figure 1.

- 1) If A, B, and C are the exact same shape (Exact Shape)
  - then return the same shape key
- 2) If A to B is a horizontal reflection (Horizontal Reflection)
  - then horizontally reflect C and test to see if solution is present in 1-6
- 3) If A to C is a vertical reflection (Vertical Reflection)
  - then vertically reflect C and test to see if solution is present in 1-6
- 4) If A to B is a 90-degree rotation (Rotation)
  - then rotate C clockwise or counter-clockwise and see if solution is present in 1-6

Figure 1 — Production rules for agent

The image dictionary is run through each of these transformation checks, and a solution is returned if a transformation is detected otherwise -1 is returned. -1 was chosen to distinguish if my agent was coming up with a solution rather than randomly guessing correctly with a return value of 1.

The generation of a solution is based on the generate-and-test approach. Each transformation detection method compares problem image A with problem B or C. A transformed shape is generated based on the transformation it is checking for. If the transformed shape matches image B or C, a potential solution is generated and tested against images 1-6 to see if the solution is present. If the solution is present, the image key is returned which represents the answer to the current problem set.

The first and most obvious transformation is a check to see if A, B, and C are the exact same shape. I used the TemplateMatch method in the cv2 library that slides a source image across a destination image to see how many times it can locate a match to the source image. In this case, all the shape images from 1 – 6 were run across the problem image to how many matches there were. A match was limited to the pixel locations of A, B, and C in the problem set image to prevent a match with itself in the solution section. If a shape had a match count of three the agent knew this shape had to be the solution and the shape's key was immediately returned.

The next two transformations dealt with reflections across the X or Y-axis. My personal approach was to always check for horizontal reflections first, so I designed my agent to check for this transformation before a vertical transformation. The DetectFlipHorizontal method flipped image A across the Y axis using the cv2 flip method. The horizontally flipped image was compared with image B to determine if the two images were the same. I created a helper function called ExactCheck that used the subtract method in cv2 to come up with the differences of the two images. Images that were closer in similarity had more black pixels present in the subtracted image. Black pixels have a pixel value of 0, so I took a sum of the array of the subtracted image and set a threshold of 25000 pixels to determine if the two pictures were the same. Once I was able to detect a match between horizontally reflected A and B, I horizontally reflected image C and did another ExactMatch to see if the solution was present in 1-6. If the solution was detected, the image key was returned as the solution.

Figure 2 — Exact match after subtraction between two images

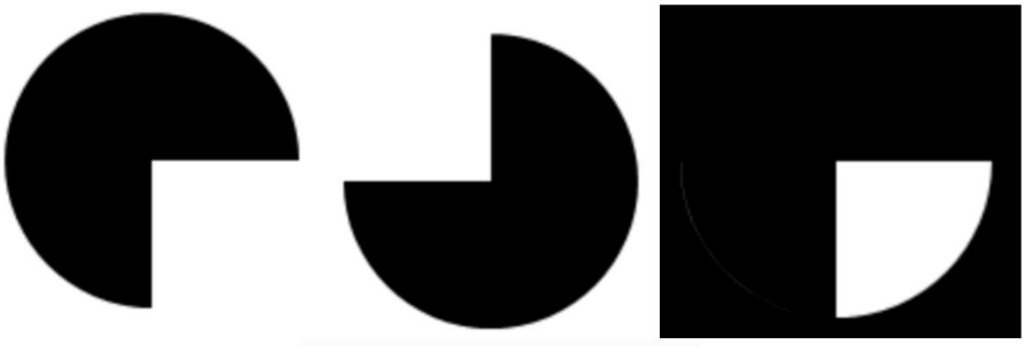

Figure 3 — Not an exact match after subtraction between two images

Reflections across the X-axis functioned very similarly to reflections across the Y-axis. Image A was reflected across the X-axis using the same cv2 flip method, but this time was compared against image C. The same ExactMatch was used to determine if the two images were the same. An exact match triggered the agent to generate a solution that vertically flipped image B and checked it against images 1-6 to see if the solution was present.

The last transformation that my AI agent can detect are rotations. Again, the cv2 library was used to generate a 90-degree clockwise rotated version of image A. This rotated image was compared against image B and if an exact match was detected, a clockwise rotated version of image C was generated and checked against images 1-6. Also, the counterclockwise transformation was checked between A and B. If a match was detected, image C was rotated in a counterclockwise manner and checked against images 1 -6 to see if the solution was present.

1.2 Agent Performance

The evaluation of my agent's performance is defined by how many problems it was able to guess correctly. My agent was designed to only return a solution between 1 – 6 if it was able to explicitly detect a transformation that are present in my production cases otherwise -1 was returned. I have a print statement that displays the transformation that was detected so I can see what transformation my agent detected. My agent performs well if the transformations mentioned above are present in the problem set. The four transformations: exact shapes, horizontal reflection, vertical reflection, and rotation were enough to get the first eight questions correct in the Basic Set B. The following stats display my agent's performance.

Basic Set B: 8/12

Test Set B: 6/12

Challenge Set B: 1/12

As you can see, my agent performs very poorly on Challenge Set B getting only one question right out of twelve. The reason for this is due to the complexity of the challenge problems. These problem sets have complex transformations that my agent currently can't handle such as the deletion/additions of shapes, multiple shapes in the same frame, different types of shading, and different number

of shape sides. These are production rules that I will need to add to my production system to be able to detect these more complex transformations.

1.3 Agent Problems

My agent performs well on reflections and exact shape matches. This was visually observed by the pixel difference between a reflected image subtracted against B or C that was a perfect match for the reflected image. In an ideal scenario, the pixel difference between a reflected match should be 0 pixels, however it seems that the images have very minor differences in pixel density. A pixel threshold was set at 50,000 pixels, which represented a sum of all the white pixels that were seen in the resulting subtracted image. This threshold allowed me to accurately detect horizontal and vertical transformations without generating false positives. My agent is also able to detect exact shape matches relatively quickly. This was determined by printing out the hit count for an exact shape match and checking against the problem set to see if the hit count was accurate. This took some fine-tuning by tweaking the threshold ultimately setting it at 0.99 to get the best performance in detecting shape matches.

My agent seems to have a harder time with rotations. This seems to be due to a larger difference in the transformed images. I had to set the pixel threshold much higher at 125,000 pixels. The initial threshold was set at 50,000 pixels and failed to detect a rotation in problem B-06. There is a rotation problem in Challenge Problem B-01 (Figure 4) that my agent fails to detect correctly. This could be due to the multiple shapes that are present in image C that leads to a larger pixel difference that ends up being greater than the pixel threshold causing it to not detect properly. I can either increase the threshold to a higher value based on the number of shapes present in the image or find a way to normalize the image to lead to smaller pixel difference between exact images.

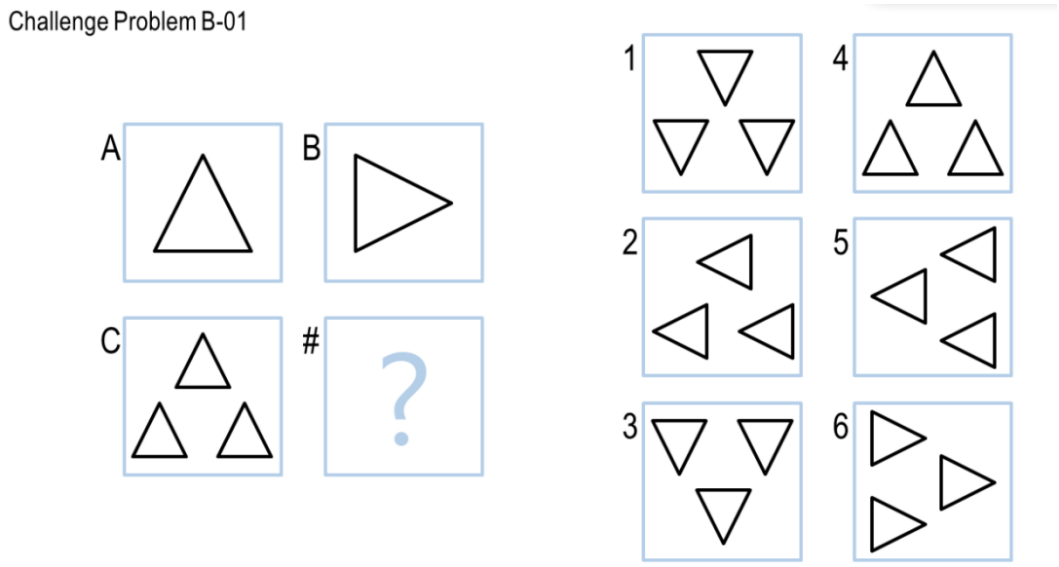

Figure 4 — Exact shape detection challenge

1.4 Agent Efficiency

My definition of efficiency is based on the length of time it takes my agent to complete the Gradescope tests. My most recent three submissions have been averaging around 120 seconds to get through all the tests. This is relatively slow because my agent isn't performing anything computationally intensive. Most of the transformation methods are 2D array manipulations, which are optimized for performance using numpy and should be quick. I noticed that I was loading in the image dictionary as a parameter into each of the transformation detection methods and reading the image using the cv2 library each time a transformation method was executed. This was incredibly inefficient, so I removed the image dictionary as a parameter and loaded the entire image list using cv2 at the very beginning of the solve method. I stored the processed images as a dictionary with its image ID as a key and passed that into every transformation instead. This ended up saving ~10 seconds in the subsequent Gradescope submissions.

However, I was still wondering what was causing my agent to take over 100 seconds to complete the Gradescope tests. Upon further investigation, I realized that my agent was spending most of its time stuck on Challenge Problem B-05 (Figure 5). I noticed that one of the answers (image 6) was a blank image. The TemplateMatch method from cv2 slides a source image against a destination image and records every time there is a match. This blank image was generating a ton of

matches and significantly slowing down the agent because of the large white space in the problem set image. This was something I didn't account for when implementing the detection of the exact same shapes in image A, image B, and image C. My plan is to replace the problem set image and directly use image A, image B, and image C as the destination image for the TemplateMatch method. This should make my agent much more efficient and reduce the runtime to under 10 seconds.

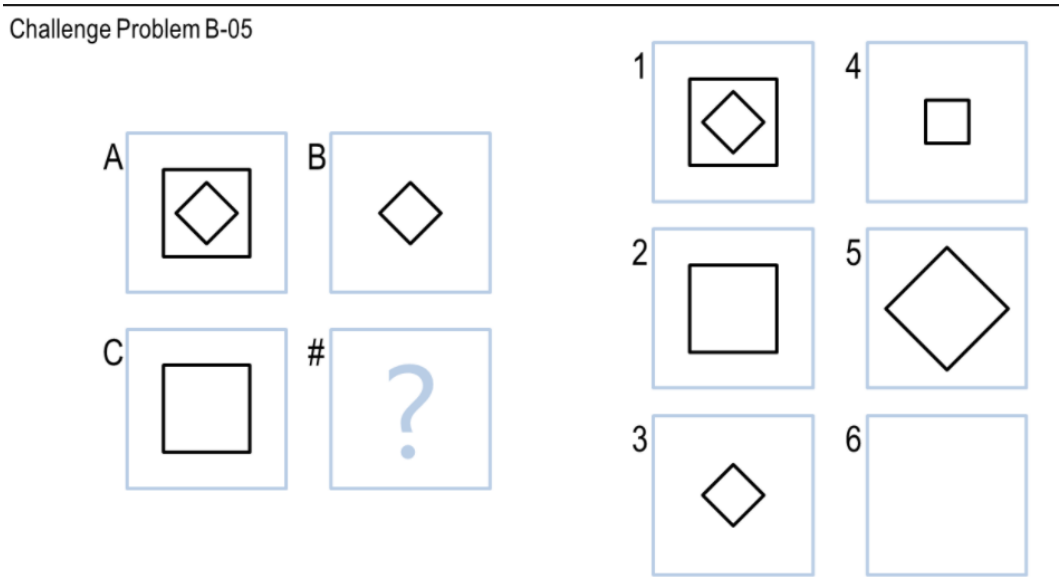

Figure 5 — Empty solution problem with TemplateMatch

1.5 Improving Agent Performance

My plan is to add to the production cases in my production system. The remaining four problems in Basic Set B that my agent needs to be able to solve can be broken into the following production cases: shade detection, shape addition, and shape addition. The immediate plan would be to develop the methods for the three different transformations and add them after the rotation section in the solve method in Agent.py. The new transformations are more complex than just transforming an image and comparing its exactness to another image. The production cases will need to detect the addition or deletion of a shape. My initial plan is to compare the pixel count between image A and image B or image A and image B. If the pixel count goes from high to low, a deletion is detected. If the pixel count goes from low to high, an addition is detected. The resulting difference of pixels can be used to apply the same subtraction or addition to find a

solution with a similar number of pixels. Another option would be to use the findContours method in cv2 for shape detection and measure the differences in contours between two images.

1.6 Generalizing Agent's Design for 3x3 Problems

My current agent employs a very simple production system and maintains the different types of transformations as production cases. Only two images are compared before coming up with a solution, so I didn't feel the need to add a more complex knowledge representation. However, the 3x3 problems will require the storage of relationship information between multiple images and coming up with the best solution based on the transformations it recognizes. My plan is to create a generalized semantic network to represent the relationship information between images or nodes. This will be done by creating an image class that will store information about the nodes along with its relationship with another image. The relationship information will be a single or list of transformations that the agent will detect. It will be important to create a weighting scale to pick the best solution for these more complex problems. I plan on starting with the weighting scale shown in Figure 6 as a starting point and building off that foundation.

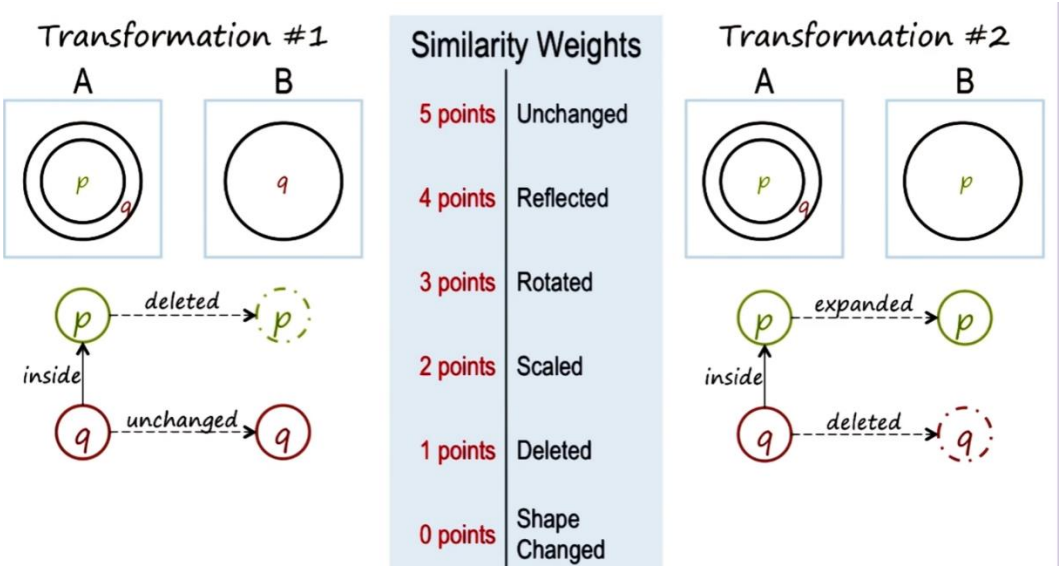

Figure 6—Weighting scale from Lecture 3

1.7 Classmate Feedback and Benefits

The feedback that I would like to get from my classmates is regarding a better way compare the exactness of two images. I am having some trouble with certain transformations like rotations. I will rotate an image 90 degrees and will compare it with an image that should be the exact same shape as the rotated image. It seems like there is a larger discrepancy between the transformed image when you do a subtraction function with a comparison image. I've solved this by increasing the threshold pixel difference between the two images. This works with the current test set that I am given, but I don't know how it would work for more test cases since the threshold seems high.

I would also like some feedback about my plan to handle shape deletion and addition. My current plan is to just compare pixel differences between two images and find a solution image that has a similar number of pixels after the addition or removal of the calculated pixel difference. I'll have to figure out the ideal threshold based on trial and error until my agent is able to accurately produce the right solution. There seems to be a more complex option using the findContours method from cv2 but the approach I just mentioned seems a lot easier to implement and good enough to solve these problems.

The last bit of feedback I would like to receive is regarding my design for a 3x3 matrix. My initial thoughts are to add a semantic network to capture transformation relationships between multiple images with the combination of a weighting scale to produce the best scoring solution. I am wondering if this is a sufficient approach to these 3x3 problems and if I am missing any important aspects to consider for these larger, more complex problems.

The challenges that I think I could benefit are to understand if there is a better path to take for more complex transformations along with improving the performance of my agent. Since we can't see the test cases in Gradescope, I am left to wonder if my agent isn't getting some of these test cases correct because my threshold is too high. I think I would benefit from someone else's feedback regarding the overall design of my agent. There will need to be some refactoring that needs to happen to generalize my transformation detection methods along with adding a few classes that will represent my semantic network. Critical feedback about my choice of using a production system would be helpful because I can pivot to another knowledge representation if needed.