CS 7637: RPM - Final Project Report

Stephen Yang
syang639@gatech.edu

1 FINAL PROJECT

The Raven's Progressive Matrices test, or RPM, is a method that is used to test general intelligence. Given a grid of images and possible answers, one is expected to be able to read the grid, determine a possible pattern, and solve by selecting the correct answer. While this may be trivial for a human, how would we allow an agent to solve these questions? In the report below, I will be exploring and walking through how I designed my agent for this project.

2 DESIGN AND FUNCTIONALITY: HOW DOES IT WORK?

2.1 - Design: How does it work?

When the agent first receives a question, there is a lot of image processing that the agent will perform before attempting to solve. For one, the agent will first load all of the figures into a dictionary with its visualFilename, as this name will be randomized, so selecting the correct name is vital in ensuring the agent will return the correct answer.

After loading all of the base figures in a dictionary, the agent will then preprocess the image three different ways: Inverting the image, slightly blurring the image, and then blurring and inverting the image. Most of these were created with the cv2 functions such as threshold, GaussianBlur, and cv2Color, as these helped with quicker processing of the images.

During this phase, the agent also loads some information, such as lists of trios and keys, which are later used by the agent. Trios lists include trios like [ABC, DEF, GH] and [ADG, BEH, CF], and will be used to test different strategies.

After the information is processed, the agent will then decide what method to use in solving. As shown in Figure-1 on the next page, the agent will first look at the type of the question, possibly look at the name if necessary, to decide on the path to take. All paths taken and objects checked by the agent are shown in the figure below.

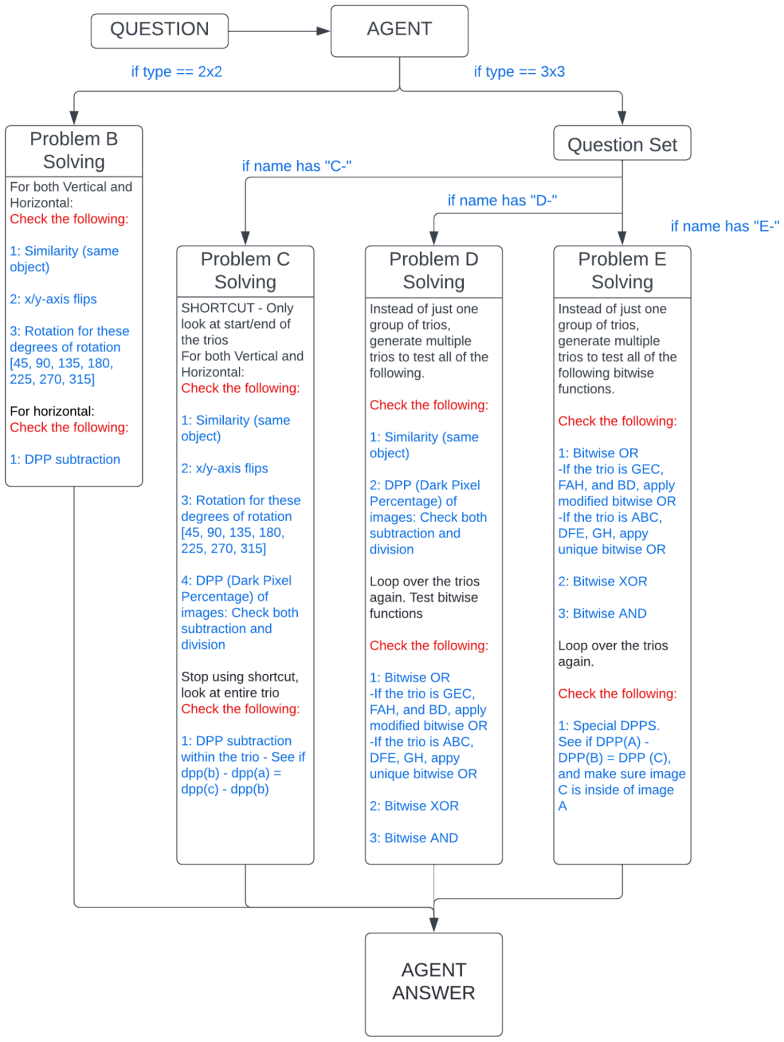

Figure 1—High level overview of the Agent's reasoning process. Red text implies a loop or major conditional check, blue text is the topics/methods used in solving. Higher quality located in appendix 4.1

Similar to the preprocessing phase, many of the methods used in solving the questions provided were done by using many Python Libraries, such as numpy, Pillow (PIL), and OpenCV (cv2). Table 1 below lists out some of the more important functions that were used for each method from the library.

Table 1—Major functions and libraries used for method creation

METHOD	**FUNCTIONS/LIBRARY USED**
Similarity	cv2.matchTemplate

X/Y-Axis Flips	PIL.ImageOps.mirror PIL.ImageOps.flip cv2.matchTemplate
Rotation	PIL.Image.rotate cv2.matchTemplate
DPP (Dark Pixel Percentage)	Numpy
Bitwise OR	cv2.bitwise_or cv2.matchTemplate
Bitwise XOR	cv2.bitwise_xor cv2.matchTemplate
Bitwise AND	cv2.bitwise_and cv2.matchTemplate
Modified Bitwise OR	cv2.bitwise_or cv2.matchTemplate DPP

While going through each method, the agent will be comparing the result of the method either through the correlation obtained from the matchTemplate function, or by the closeness of the DPP operation, whether it is subtraction or division. Depending on the type of question, there may be more than of these values appearing, such as a question from parts D and E having two correlations, one for the top row and one for the middle row. However, as long as one of these correlations is relatively high, usually above 95%, or the DPP is only several percent off, the agent will then try and solve the final trio using the method prescribed.

For most questions, the methodology is pretty simple: Given the possible relationship, generate the expected image from the remaining images, and then perform matching to see if the expected image is in the answers. For this, the agent adopts a more generate and test approach, as the agent will generate its expected state, which is the image and test to see if the expected state is present. Methods that follow this image generation include Similarity, X/Y-Axis Flips, Rotation, and Bitwise OR/AND/XOR.

However, there are some methods in which the agent is unable to generate the expected image, such as any method involving DPP. For these, the agent adopts a slightly different approach, as it would determine which value it is searching

for, and compare the DPP results from the answers to see which one was the closest. In this case, instead of generating new images and states, the agent generates a number to look for, and then will test the number with each answer to see which one is the closest. Methods that follow this number generation include DPP subtraction, DPP division, and modified bitwise OR.

If the agent goes through all the methods and does not find a method that would best suit the problem, the agent will then "skip" the question. However, since there is no harm in guessing, I changed the agent to guess a possible answer instead of skipping, allowing it to possibly gain some more points for the project.

2.1.1 - Design: DPP in depth

One heuristic that was used heavily that hasn't been explained, other than the correlation obtained from cv2's matchTemplate function, was DPP, or Dark Pixel Percentage. Given any image, the Dark Pixel Percentage is simply the number of dark/black pixels divided by the total amount of pixels in the image. This function is used as dark pixel count is too varying, and trying to account for possible errors with just dark pixel count is impossible.

While the other methods used by the agent are more straightforward, such as x/y-axis flip and similarity checks, DPP is used in many different ways depending on the question. To check for objects growth or duplication, such as Basic C-06, the agent looked for a consistent ratio of DPP between AC and DF. However, using the ratio for Basic C-05 would not work, as the numbers would vary too much to lead the agent to have a definitive answer. For this question, DPP subtraction would be more fitting, as the growth in each group would result in the same percentage of pixels growing from A to C as D to F. Because of these potential problems, my agent would need to incorporate both the division side and subtraction side in order to catch all possible cases.

2.2 - Design: Approach

When it came to the overall design and approach when creating the agent, I originally thought that the final result would be a group of general heuristics, such as rotations and flips, which would then be used to solve all the questions given. However, as I worked more on the project, my agent slowly ended up becoming a large list of possible heuristics, which would easily lead to overfitting if not monitored carefully enough.

In the end, the agent used a lot of Generate and Test methods in order to solve as many questions as possible. In questions like C-07, the agent was able to generate the reflection of image G to compare against all of the answers, eventually resulting in choosing 2 as the correct answer. In question B-11, the agent first generated the expected difference of dark pixels from image C to the answer. Afterwards, it then calculated the difference between C and all the possible answers to see which pixel difference was the closest to the expected answer.

3 PERFORMANCE & ANALYSIS

3.1 - Overall Performance

As of the time writing this report, the agent seems to be performing relatively well, scoring a total of 81/96 when submitting to Gradescope. Given the preliminary results from the Milestones, I was expecting my agent to score at around 60 to 70 points, so I am pretty pleased with the agent's current performance. While it may not be performing perfectly, the agent itself is performing quite well considering the amount of time put into it.

Table 2—Performance overall on the project from Gradescope.

MAIN SET	SUB SET	CORRECT	INCORRECT
B	Basic	11	1
B	Test	11	1
B	Challenge	8	4
C	Basic	11	1
C	Test	10	2
C	Challenge	6	6
D	Basic	9	3
D	Test	8	4
D	Challenge	1	11
E	Basic	11	1
E	Test	10	2
E	Challenge	2	10

Overall, my agent seems to be doing pretty well, managing to score above 10 on most of the Basic and Test cases, but falling short on many of the challenge

questions. However, since the challenges were not graded, it worked out better for my agent in the end.

3.2 - Problem Analysis

This section will be for in-depth question analysis, where I explain the rationality and reasoning behind my Agent on questions that it solved successfully and questions it did not solve successfully.

3.2.1 - Successful Solves

As seen with the statistics above, my agent is able to solve a large amount of questions properly. To explain this, I will be looking at and explaining the reasoning behind these three questions: Basic B-11, Basic C–08, and Basic D-01.

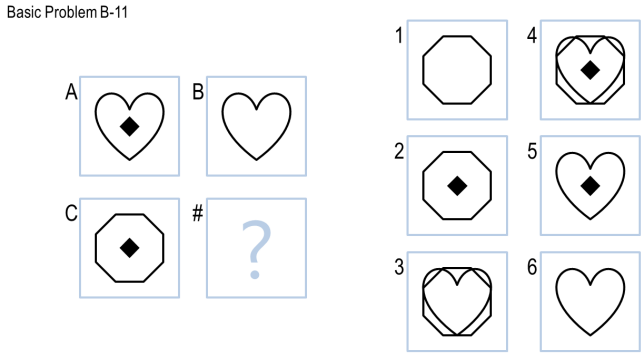

Figure 2—Basic Problem B-11. Answer should be 1

For Basic Question B-11, the agent starts looking at the problem with the standard 2x2 method implemented, checking for similarity, reflections, rotations, etc. However, none of these basic transformation methods work, as this case is actually one of removal, where the diamond is being removed from the middle. In this case, my agent then calculates the DPP, or Dark Pixel Percentage, of the images A and B, and finds their difference. The agent then will compare the DPP of image C with all the DPP's of the answers, calculating each difference and storing them in an array. Given the original difference between A and B, the agent finds the difference that is closest to the original, which results in the answer 1 being chosen.

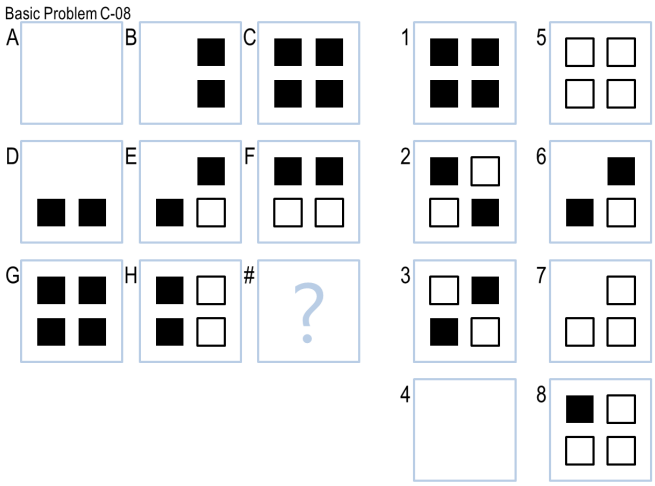

Figure 3—Basic Problem C-08. Answer should be 5

For all of the Problem Set C's, the agent tried to take some shortcuts to result in the answer, only evaluating the beginning and end parts of the trio, such as only analyzing the relationship between A and C, and D and F. For C-08, the agent was initially unable to find a method that worked, returning low correlation values for similarity, axis flips, and all the possible rotations. Even while checking the DPP values, the subtraction and division results were not close enough to result in those methods being chosen.

However, I noticed that this pattern dealt with similarity in the DPP within a trio instead of between trios, where the dark pixels of B subtracted by the dark pixels of A would be the same as the dark pixels of C being subtracted by the dark pixels of B. Thus, the agent had a new check, which was to see if the differences between consecutive DPPs in a trio were the same. If they were the same for the first and second trio, such as ABC and DEF, the agent would then solve for the answer by finding the image that had a DPP most similar to the DPP difference between image G and H.

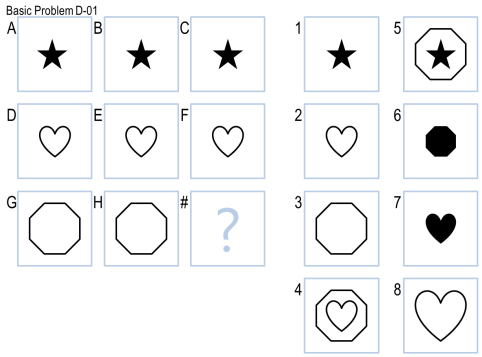

Figure 4—Basic Problem D-01. Answer should be 3

The main concept behind D-01 is similarity, as the same figure that is present in image A is present in image B and C, similarly with the second trio of DEF. When solving this question, the agent first checks to see if the images of A, B, and C are the same, using the similarity check developed for B. If the images are similar, the agent will then perform the same similarity check with D, E and F. If the images are determined to be similar enough, the agent will then take the last two images, G and H, and find the answer that is the most similar to the images, which would result in 3 being chosen.

3.2.2 - Unsuccessful Solves

As seen in the table above, there were many questions that the agent was unable to solve. While a majority of these questions were skips or guesses, there were still a few in which the agent did incorrectly choose without guessing. In this section, I will be looking at a few of these questions, specifically Basic C-09 and D-10.

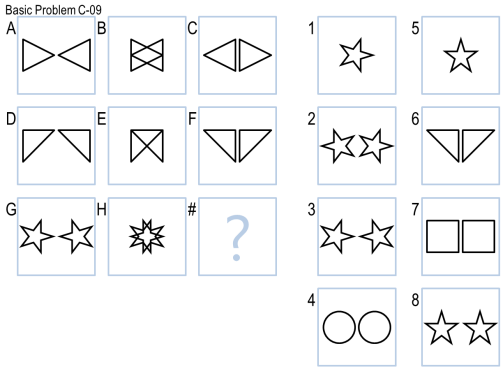

Figure 5—Basic Problem C-09. Answer should be 2

For Basic C-09, the agent tried all the methods for C-type questions, but was unable to find a method that worked, such as basic rotation, flip, or similarity. Since it exhausts all those options, the agent will then use the secondary check that I implemented, which is the method of DPP comparison talked about when solving for Basic C-08.

One way that an agent can look at this question is by splitting the image into half, performing a y-axis flip on both images, and then combining them back together. However, since my agent does not have this logic, it ends up checking the DPP comparison between the figures. Since there is little change in the DPP in A, B, and C, the agent mistakenly assumes that this method is subtraction, when in reality it isn't. By doing this, it ends up with the answer of 6, even though the answer should be 2.

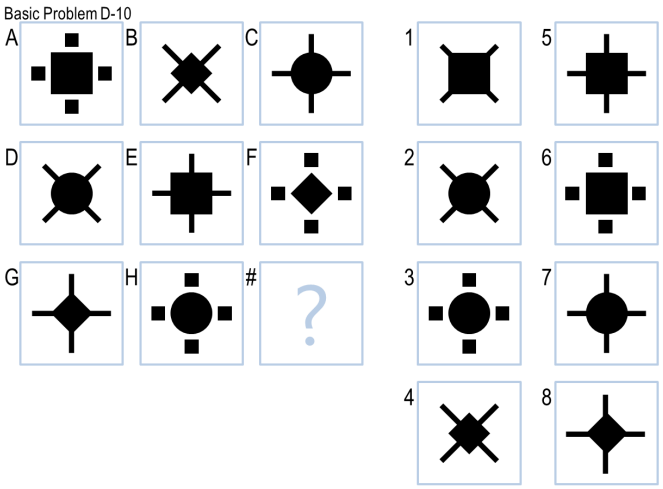

Figure 6—Basic Problem D-10. Answer should be 1

For D-10, one way that a human could reason and solve this question is to look at this diagonally, with the diagonals consisting of AE#, FGB, and HCD, as the outer shapes will change in that pattern. I had reasoned that between these figures, there should be a constant change in the DPP which the agent would be able to detect, possibly through the division of their DPP values. However, due to the shape and inconsistency of the images, the agent concluded that the DPP values were not valid enough to proceed with the test. In the end, the agent reasoned that a different trio of GEC, FAH, and BD would be able to solve the question, using the modified_or method to select the incorrect answer of 4.

4 HUMAN VS. AGENT

Given the methods used to construct and perform these tests, I would say that the agent itself is not very similar to how I would approach this test.

As a human, there is an extremely large variety of ways that I would approach this test. For example, I am able to tell if an object has been shifted, added, or even shaded in by simply looking at the picture. However, while these things can be done with little thought by a human, an agent is unable to perform these things as efficiently, and must rely on techniques such as template matching or the difference between the pixel ratios in order to determine which category of solving the question will fall under.

Another thing that my final agent lacks is the ability to learn by cases by itself. For example, if a human encounters a new problem and is unable to solve it, they might take a look at the answer, look back at the question, and find the relationship from there. Afterwards, a human can then commit this type of question into their knowledge base, allowing them to learn from previous cases. However, by itself, the final agent is unable to do so, as it is only programmed to handle certain types of tests. Because of this, if an agent encounters a new question that it gets incorrect, it is unable to reason and save the case in memory, allowing it to get it correct in future attempts. This has to be done by the programmer, so I would say that the agent is different from humans in this way.

Another way where the agent differs from a human would be in terms of solving by frames. For a question like D-12, the pattern is rooted in two parts: Object shape and number of objects. If a human observes this, they may be able to put this information in a frame, labeling image A as “Square, 3”, image B as ”Triangle, 4”, and image C as “Stars, 5”. This type of labeling and putting information into frames might lead the human to see the pattern, which is that each row must have one of each of the shapes, as well as a 3, 4, and 5 count. However, my current agent is unable to do this, as it is currently only coded to be able to perform DPP and small template matching, not any frame representation.

4 APPENDIX

4.1 - Full image of Figure 1