Li (Lily) Xu (lxu321@gatech.edu)

Project 2 Reflection

1. Introduction

Raven's progressive matrices was first published in 1938 and designed to measure one's mental ability to reason by analogy and form perceptual relations. RPMs are composed of 60 multiple choice questions, 5 sets with different difficulty levels, each contains 12 questions. The problems usually contain several panels of pictures with one missing. The test taker is supposed to pick up the best match of this missing piece from the provided candidates of answers. Since it is not verbal, RPMs can be used with persons from child to adult in any language. In this class, we will assess the algorithms and reasoning of our artificial intelligent agents by examining their performance against the RPM problems.

There are many potential difficulties about how to solve the RPM problems with an AI agent. For example, first of all, how does the agent start to understand a question? How can it recognize a circle, a triangle or any other element? Secondly, how does an agent build the concept of "transformation" or in another word, how does the agent decide on the changes and make correlations between elements before and after these changes? Moreover, how does an agent compare and rank different types of transformations and how does it find the best choice, if there is no perfect match? Here, I implemented an AI agent to solve RPM problems with knowledge learnt from this KBAI class.

2. A brief summary of the agent

- **Data structure**

The agent starts from the main method in RavensProject.java and reads in all the RPM problems. These problems are then stored in ProblemSet.java, which is composed of a list of Raven problems. Each Raven Problem is represented by RavensProblem.java, which consists of its problem name, problem type (such as “2x2” or “3x3”) and a HashMap of RavenFigures. Each RavenFigure is composed of a figure name (“A”-“H” or “1”-“8”) and the path of the visual file. The class RavenFigure also contains a HashMap describing attributes of Raven objects, but I won’t be using this information as I intend to use purely visual method for this project.

With the path of a .pgn file, the program reads in the visual presentation of a RavenFigure:

```
RavensFigure A = problem.getFigures().get("A");
BufferedImage figureAImage = ImageIO.read(new File(A.getVisual()));
```

Then I implemented a method called `imageToArrayRGB` () to covert data stored in the BufferedImage into an int 2D array with 0 or 1:

```
int[][] A2D = imageToArrayRGB(figureAImage);
```

Every pixel in each row and column of the BufferedImage is iterated through, and converted to 0 (white) or 1 (black) based on its RGB value. For a grey pixel, a threshold was set so that it is considered as a black pixel if its value is above the threshold or otherwise a white pixel.

- **Methods implemented**

to find the transformation between figures:

```
private String getTransformation(int[][] a2d, int[][] b2d, int[][] c2d)
```

to compare two transformations by scoring differences:

```
private int calculateDiff(int[][] a, int[][] b)
```

to find the choices closest to expectation

```
private int getAnswer(int[][] expect, int[][][] choices)
```

- **Converting 3x3 to 2x2**

3x3 Raven problems are generally harder than 2x2, the difficulties come from:

1) More transformations to generate:

The transformation between A and B tends to be more ambiguous and arbitrary in 3x3 problems. Thus an agent needs to examine and compare more transformations between different panels to narrow down possible changes.

2) More transformations to evaluation:

With a large pool of transformations, one need to come up with more sophisticated strategies to rank all the possible transformations based on information collected from different rows and different columns. Additionally, besides the horizontal and vertical, the agent often needs to take diagonal relations into consideration.

Sometimes, hard 3x3 problems can be easily solved when considered as a 2x2 problem. For example, in the following problem (Figure 1), it is hard to define the appropriate transformations that represent the whole problem. However, if we only look at the A-C-G-# subset, then the transformation is a simple "disappearing of circles" transformation.

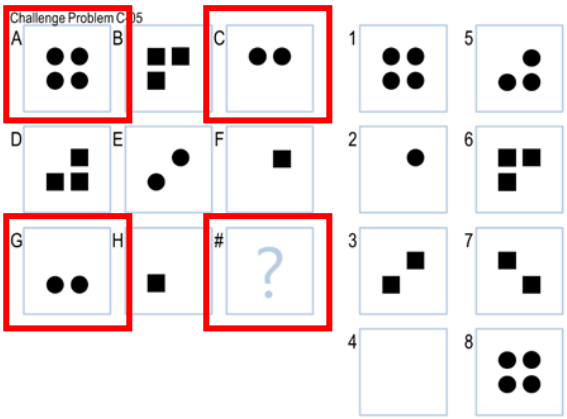

Figure 1

More ways to convert 3x3 problems into 2x2 problems need to be considered and evaluated:

1) subsets of panels

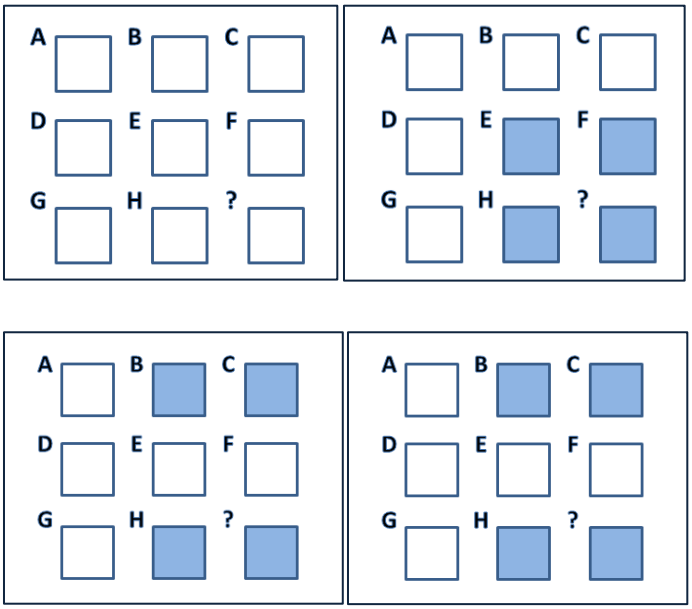

Figure 2

2) a single row/column

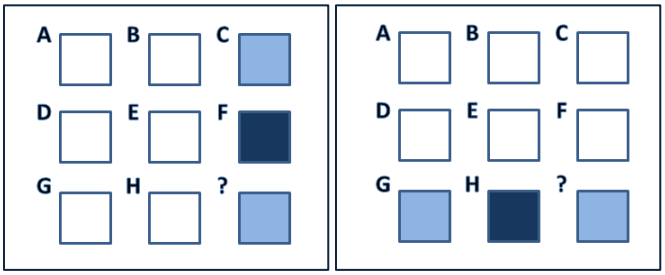

Figure 3

3) diagonal

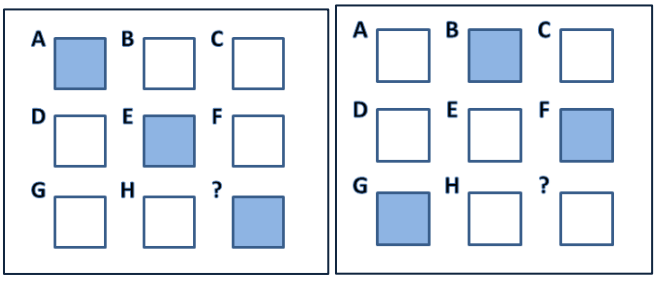

Figure 4

3. Result and analysis

Problem	Agent's Answer	Correct?	Correct Answer
Basic Problem C-01	3	Correct	3
Basic Problem C-02	4	Correct	4
Basic Problem C-03	4	Correct	4
Basic Problem C-04	8	Correct	8
Basic Problem C-05	3	Correct	3
Basic Problem C-06	7	Correct	7
Basic Problem C-07	2	Correct	2
Basic Problem C-08	5	Correct	5
Basic Problem C-09	2	Correct	2
Basic Problem C-10	7	Correct	7
Basic Problem C-11	4	Correct	4
Basic Problem C-12	8	Correct	8

Overall, I was able to solve all RPM problems in Basic Problem set C (12/12) within a few seconds. A few examples were chosen to illustrate how my agent solved these problems:

- **Identical figures**

For some simple RPMs, the figures in different panels are identical, as shown in Figure 5. By calling the method calculateDiff (A, B) and calculateDiff (B, C), my agent can compare the pixel-by-pixel the differences between pairs of Raven Figures. Ideally, calculateDiff (A, B) should return 0. However, sometimes these figures may have slight shifts that are easily ignored by naked eyes. Thus I set a small number as a threshold (for example, 100 pixels), and anything pairs of images with a smaller difference should be considered as identical.

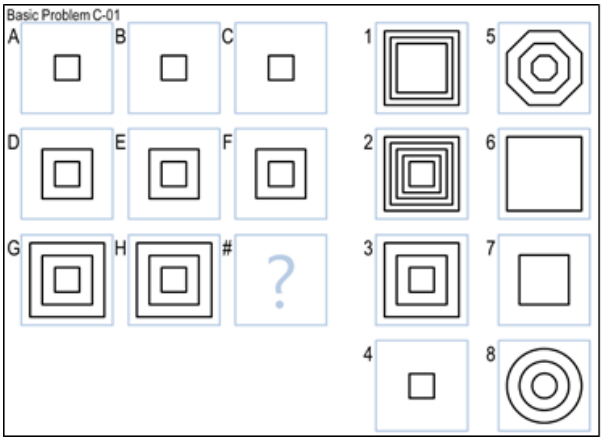

Figure 5

Thus, if a certain problem satisfies the criteria (A is identical to B and B is identical to C), the transformation is labeled as "**identical ABC**" (Figure 1) or similarly "**identical ADG**" (vertical) or

"**identical BFG**" (diagonal). Then my agent will generate an answer based on an existing panel and pick up the best matching answers from choices 1-8.

- **Symmetry**

Some problems can be solved by symmetric transformations: either vertical symmetry or horizontal symmetry. My agent defined this transformation to be "**mirror_leftRight**" (Figure 6, panel A-C-G-?). So the agent will generate an answer based on panel **G**, and compare it to choices 1-8. Similarly, my agent can detect whether horizontal symmetry exists and define similar transformation such as "**mirror_upDown**". Moreover, 3x3 problems can be converted into different 2x2 subsets by changing combinations of panels chosen, to find out the best transformation by comparison in every possible direction.

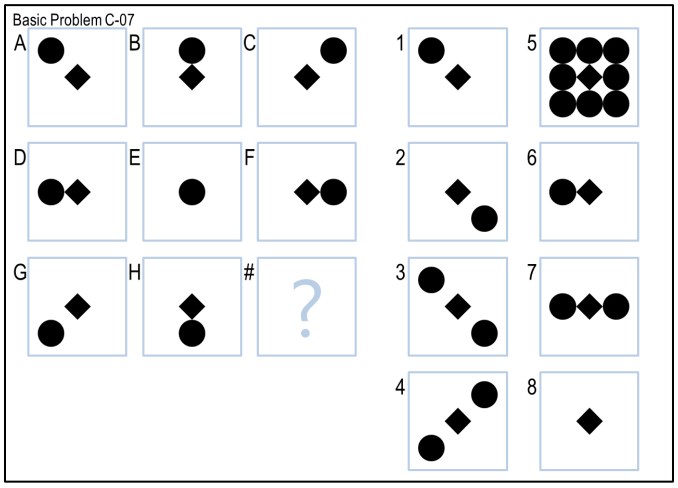

Figure 6 (vertical symmetry example)

- **Counting Pixel**

Although it is not quite intuitive in human cognition, an AI agent can solve quite a few RPM problems by just counting the number of pixels in each figure. An example is shown in figure 7-left panel, and the number of small solid diamond in a figure is proportional to the black pixel counts. Then the expected pixel numbers in "?"can be calculated by pix(G) * (pix(C)*1.0/pix(A)). In another example shown in the right panel, the appearance of the small solid circle corresponds to an increase of black pixel counts from panel A to B and B to C. The pixel numbers of the answer can be calculated by pix(G)+ (pix(C)-pix(A)). Each choice answer will be examined to find the one with the closest number of pixels. Considering slight shifts of some figures, small differences should also be tolerated.

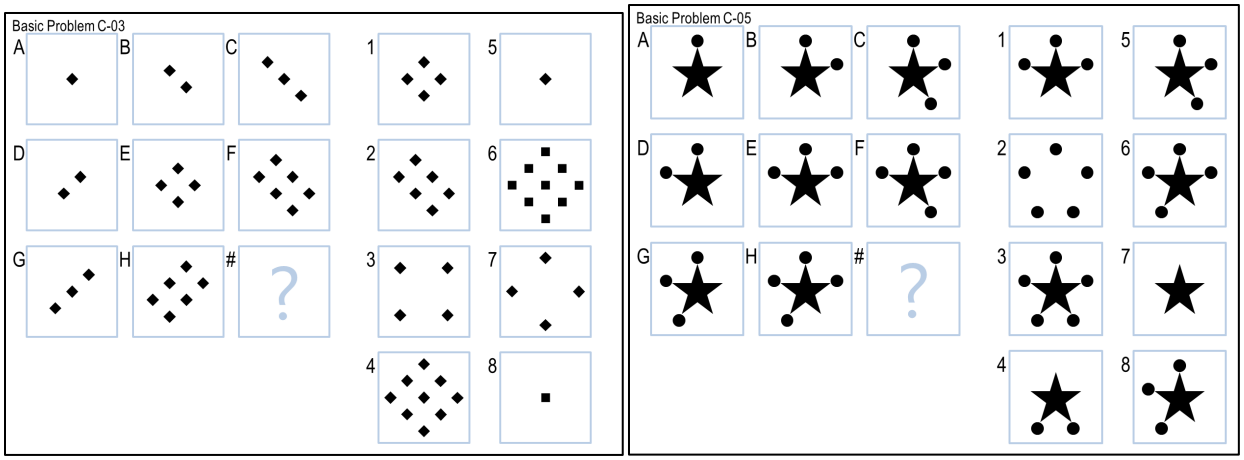

Figure 7

❖ **Problem remained**

This AI agent seems to be quite successful as it solved all RPM problems in Basic C problem set. However, there are still many problems, which could potentially help to broaden and improve the current version of my agent in Project 3. For example, problems like shown in figure 8, cannot be solved successfully at the moment, as my rather simple agent is not able to make correlations between open and filled objects of the same shape. Although it is not implemented yet, I think a combination of visual and verbal methods will be the way to solve this kind of problems.

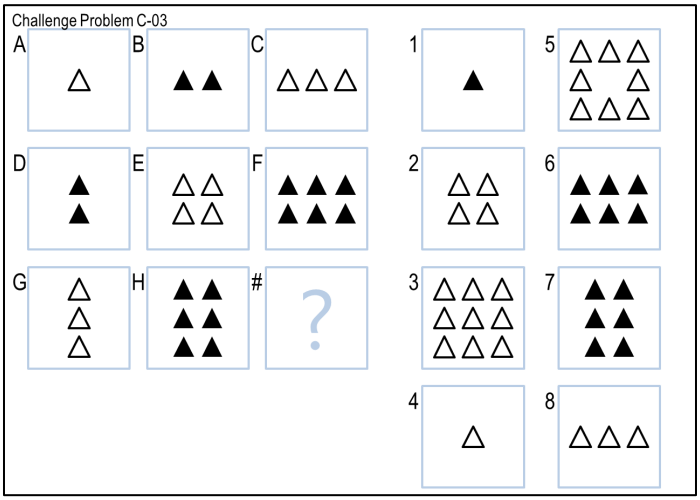

Figure 8

4. Discussion

Sometimes, a person can solve RPM problems with hidden rules that one does not even realize. For example, when we look at the answers, we pay attention to the most likely answer first. The computers can only iterate choices in order. Additionally, we constantly make logic deductions, generate hypotheses about correlations, then find violation to reject them/ evidence to confirm them. We gained

these abilities most likely by receiving stimuli from the environment all the time. Our memory reinforces the experiences that we previously found stereotypes and helps us to better adapt to the nature. Thus the responses can be both helpful and illusive, but will not affect the reasoning of our computers.

For example, when we say two figures to be identical, for example panel A, B and C or D, E and F in figure 9, we use definitions such as shape, size, fill, color, position. These are features that we use to examine the world. However, when AI read in these pictures as 2D array, it strictly compared data at each pixel, and will not be able to compromise unless intentionally designed to do so, and these panels will be considered different even though they looks the same.

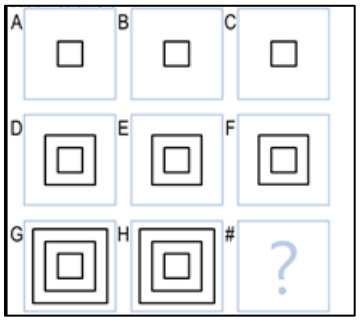

Figure 9

In another example shown in figure 10, real intelligence often thinks that the small black dot moves towards the center and then away from the center. So if one has this hypothesis in mind, when looking at panel E, he/she will see the small black dots overlaps with the small diamonds in the center. However, an AI agent can only read panel E as a single black dot, thus reason the transformation to be diamond appeared/disappeared. For this particular problem, the agent still chooses #2 as its answer due to many other hints. However, one can appreciate the huge difference in terms of understating between artificial and real intelligence.

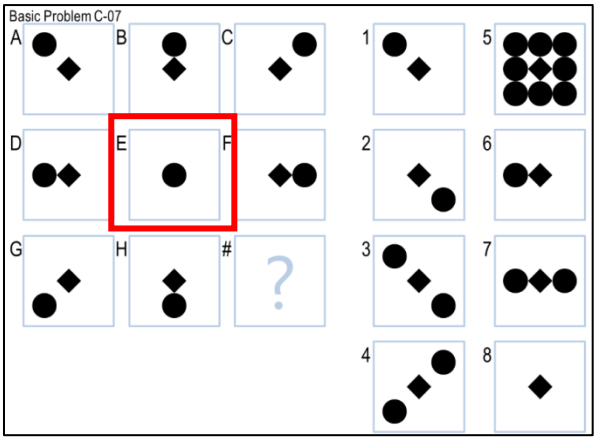

Figure 10