

CS7637 Project 1 Reflection

Chenxi Gao
cgao71@gatech.edu





INTRODUCTION:







Before I start this project, I planned to build an agent that mimic the way human would tackle the problem, designing an agent that could not only solve the 2x2 RPM, but also solve the RPM by any dimension. However, after I closely examine multiple problems in the dataset, I quickly realized that this is not an easy task because there are overwhelmingly many cases that my agent has to consider and be able to solve. Additionally, while the dimension of the RPM goes up, the number of cases that my agent needs to consider grow exponentially. For example, when dealing with 2x2 RPM, my agent needs to find the relationships between A and B or A and C, then applying the relationship to choose a best figure among the 6 choices. However, when dealing with 3x3 RPM, my agent needs to find the relationship of every row, every column, and the diagonals. Each consists of 2 operations: comparing the first to second, and second to third. After closely observe the nature of RPM, I realize that being able to solve the 2x2 RPM is the building block of the ability to solve general RPMs in higher dimensions. Therefore, I decided to first focus on the 2x2 RPM instead.

After revisiting the lecture semantic networks and observing how the professor construct a semantic network to solve the RPMs, I decide to use the same approach.

My first attempt is to only use a verbal approach to solve the problem. For example, in the following problem, I expect my agent to recognize A, B, and C to

Basic Problem B-01

A		B	
C		#	

1		4	
2		5	
3		6	


```
A.size = "LARGE"  
A.shape = "RECTANGLE"  
A.color = "BLACK"  
B.size = "LARGE"  
B.shape = "RECTANGLE"  
B.color = "BLACK"  
C.size = "LARGE"  
C.shape = "RECTANGLE"  
C.color = "BLACK"
```

be large, black rectangles. However, unlike ways of humans detecting shapes or objects in images, computers view images as pixels, where each pixel consisting of a red, green, and yellow component. Although computers are smarter to tell the color in the image in a more precise manner, they are unable to detect shapes like humans because images are stored as vectors of images. Computers cannot actually see the image.

I did some research about image shape detection, and it turns out that it is under the category of computer vision. Given that the only libraries we are allowed to use are Numpy and Pillow, detecting shapes, especially for some of the figures in the advanced problems, would not be an easy task. Therefore, I started the project using a different approach described below.

Instead of using verbal approach, I used a combination of visual and verbal approach. I use the visual approach to transform and compare images, then I use the verbal approach to capture the kind of relationship between images, then using the visual approach again to find a best match among the answers. I will include my implementation in the journals below.

JOURNAL 1:

This is my first submission. Below is a summary of how I implemented it from scratch to a working agent that may solve a small portion of the problem.

The first question that came into my mind is how the agent can tell that the two images are the same. This is the most important feature that my agent should be able to handle because no matter what transformation I made to the figure, the result is comparing the two figures and tell if they are the same. Since images are stored as pixels, if all the pixels in an image is the same, so does the image itself. Therefore, my first attempt is sum up the absolute difference between the two images using *ImageChops.difference*. I will later describe that this is not the best approach.

After closely examine the *Basic Problems B*, I noticed that most transformations are either *UNCHANGED*, *ROTATION*, or *REFLECTION*. More precisely, rotation includes rotate left, right, or reversed (180 degrees), and reflection includes reflection along vertical axis or horizontal axis. Therefore, I implemented rotation by using *image.rotate(degree)*, where degree is multiples of 90. I

implemented reflection by *Image.transpose(image.type)*, where *type* is *FLIP_LEFT_RIGHT* or *FLIP_TOP_BOTTOM*.

I divided the process of solving the RPM into two steps:

1. To capture the type of transformation from A to B and from A to C and record it accordingly.
2. Apply the same transformation I obtained from step 1 and apply it to find a best match within the answer set.

If a relationship is found between A and B, A and C, my agent will apply the same transformation from C, D to all the problems in the answer set, respectively. I realized that a single transformation maybe not enough to find a solution. For example, there may exists cases where A and B involve both rotation and reflection, but only reflection to C is in the answer set. Therefore, I manage to have my agent find all the possible relationships between AB and AC first before moving to step 2.

Comparing my agent to humans, I think my agent works similarly to what a human would do to find such transformations. I think my agent would do even better than I do because it captures all the simple transformations from A to B or A to C at the same time while I usually come up with 1 or 2 simple transformations, which is enough to solve the problem.

My agent has a different way to compares images. As human, we recognize shapes in the images while my agent only compares pixels. Therefore, my agent can tell the tiny differences between the images, but it cannot distinguish or compare different sizes of shapes.

The result is my agent only passed 3 out of 12 basic problems. It does well in comparing identical images such as problem 1 and problem 2, but surprisingly it fails most problems that involve rotation and reflection. I will explain the reasons and how I tackle it in the following journal.

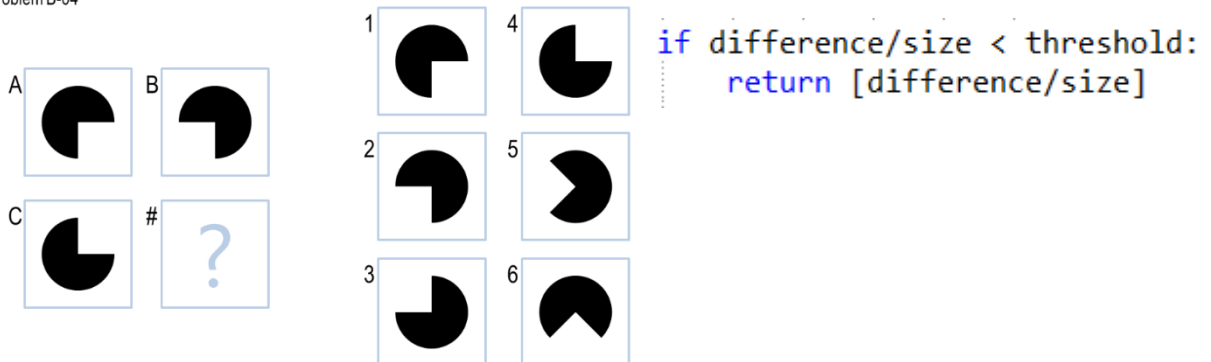
JOURNAL 2:

This is my second submission. Below is a summary of how I found and resolve a problem that cause most problems in *Basic Problems B* that involve rotation and reflection to fail.

I traced back my code to where my agent compares the transformed image in the problem set to another one in the problem set. Although the two images look the same, and thus should be a match, the sum of the absolute differences of the two images is not zero. I then realized that the transformed image does not perfectly align with the one in the problem set. In the below example, the transformed image of A with respect to the vertical axis is not the same with B. As humans, we cannot tell the difference between the two images, but there is subtle difference because the sum of the difference is small but not zero. Therefore, I added a variable called *THRESHOLD* to tolerate subtle changes between similar images. Each time during the comparison process, my agent compares the difference of the two images over the size of the image. The size equals the total number of pixels in the image, which is the number of pixels in the x-axis multiplies the number of pixels in the y-axis. If the result is smaller than the *THRESHOLD*, the agent finds a match and returns the value of *THRESHOLD*, which also serves as a confidence of how closely the images is. The closer the value to 0; the higher chance that the image is similar to the other one.

Additionally, I managed my agent sort the relationships based on the *THRESHOLD* value, so it will first find the answer with the corresponding relationship by the greatest confidence score.

Basic Problem B-04



At this moment, my agent is able to pass all the simple transformations that only involve *UNCHANGED*, *ROTATION*, or *REFLECTION* relationships, but it still fails some of those in the challenge problems set.

JOURNAL 3:

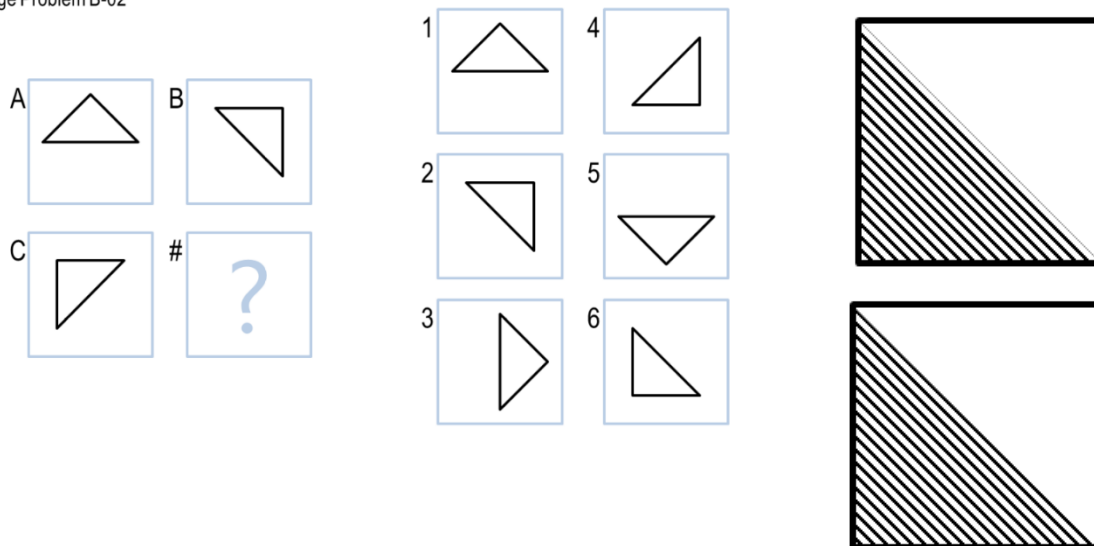
This is my third submission. Below is a summary of how I tried to improve my agent's performance in simple transformations (*UNCHANGED*, *ROTATION*, *REFLECTION*) in both *Basic* and *Challenge Problem Set*.

Rotations of degrees that is multiples of 90 is not enough to solve all the questions that involve rotation. For example, in *Challenge Problem 2*, the rotation is 45 degrees clockwise from A to B. Therefore, I added rotations in multiples of 45 degrees to my agent's library. Because a rotation of 45 degrees would turn a square picture to a rhombic one, I made the agent to maintain the original size of the image and fill the background color to white.

```
self.unchanged(img1, img2.rotate(45, fillcolor="white"))
```

In addition, I realize that my agent is unable to tell if the two images are the same in the following circumstances. The two pictures appears on the bottom right of this page seems the same. Therefore, a match should be reported. However, the two images in my agent's view is different. If you closely observe the two pictures, you may notice that the distance from the bottom left corner to the first diagonal line is different in the two pictures, and all the other diagonal lines have equal distance between each other. Therefore, sum of the difference is huge in this case. Therefore, I increased the value of *THRESHOLD*, attempting to solve

Challenge Problem B-02



questions like this.

Comparing my agent's performance to humans', I realize that a small shift or mismatch between the two images may cause my agent to make the wrong decision while humans typically ignore such shift or mismatch.

Although my agent gets more answers correctly in the *Challenge Problems* set, its performance worsen in the *Basic Problem* set. I conclude the value of *THRESHOLD* should be kept as small so that dissimilar images would be excluded.

JOURNAL 4:

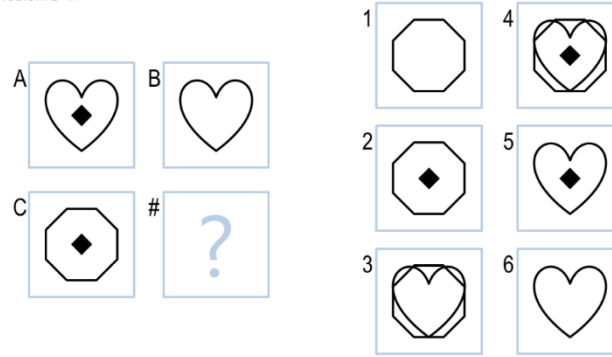
This is my fourth submission. Below is a summary of my final improvement of my agent's performance in simple transformations (*UNCHANGED*, *ROTATION*, *REFLECTION*) in both *Basic* and *Challenge Problem Set*. Also, I implemented a method to find relationship for addition or subtraction.

In *Basic Problem B-04* (on page 4), there are two possible relationship between A to B or A to C, reflection and rotation. In case of reflection, B is obtained by vertically reflecting A by the y-axis, and C is obtained by horizontally reflecting A by the x-axis. Therefore the answer is 3, which is the correct answer. However, in case of rotation, B is obtained by rotating A 90 degrees clockwise, and C is obtained by rotating A 90 degrees clockwise. Therefore, 1 is another "correct" candid. Why is the correct answer 3 but not 1? As a human, answer 3 prioritize over answer 1 because 3 "completes" the image set so that it forms a square in the center. Therefore, I ranked the relationships in the order of *UNCHANGED*, *ROTATION*, *REFLECTION* given that the value of *THRESHOLD* is small.

Another feature I implemented in this version is to find addition and deletion among the two images. In the example below, B is obtained by deleting the small black small rhombus in the center. Similarly, 1 is obtained by deleting the same rhombus, and therefore it is the correct answer.

If my agent did not find a match in the simple transformation, I instruct my agent to do the following:

Basic Problem B-11



1. Get the absolute difference (XOR) between A and B, A and C
2. Compare it to the absolute difference between C and each figure in the answer set, B and each figure in the answer set, respectively.
3. If a match is found, return the answer.

At this time, my agent can answer 11 out of 12 problems in the *Basic Problem* set. The feature, addition/deletion, is similar to how I would approach the problem. Also, the overall runtime is less than 2 seconds.

JOURNAL 5:

This is my final submission. Below is a summary of adding flood fill into my agent library.

To address Basic Problem B-09, B is obtained by flood filling inside the octagon. Therefore, I instruct compare the flood filled images A, B to B, A, respectively. If a match is found, my agent applies the same transformation to find an answer.

At this time, my agent is able to solve all the problems in the *Basic Problem* set, and got 10/12 in the hidden tests on Gradescope.

CONCLUSION:

Below is a result summary for my last submission:

Results:

```
Basic passed: 12, failed: 0
Test passed: 10, failed: 2
Challenge passed: 4, failed: 8
Ravens passed: 6, failed: 6
Runtime: 2.1591696739196777 seconds
```

I believe the result is fair, but not the best. My agent can tackle simple transformations, including *UNCHANGED*, *ROTATION*, *REFLECTION*, and more complex problems like addition/deletion and flood fill.

I would characterize the overall process of designing my agent to be targeting one type of problem at a time. Since simple transformation may solve most of the problems in the *Basic Problem* set, I started working on that first. Then I worked on addition/deletion and flood fill to solve the remaining problems.

Comparing my agent's approach to my approach to solve the RPMs, I think there are both similarities and differences. The similarity is my agent mimics human's knowledge and reasoning by using a library to transform and find similarities between images. The differences are:

1. My agent compares images by pixels in greater precision, but humans compare them by pattern/shape recognition in less precision.
2. My agent is not able to learn or create knowledge like a human would do. If a human encounter an RPM that he/she have never seen before, he/she may try to use a different approach to solve the problem. If successful, the new approach would be added to his/her memory. Unlike human, my agent would does not invent or learn by itself. If it sees a problem that is not in its library, it simply gives up. The only chance for my agent to learn is when I add more code into my agent's library.

There are some limitations that impact my agent's overall performance:

1. Since my agent is unable to detect shapes, it cannot tell what is inside/outside/added/deleted.
2. My agent is unable to count the number of shapes in a figure.
3. My agent is unable to crop a portion of the figure or move a shape to a different location. It would fail to address the inside/outside relationship. (E.g. Figure A is a circle inside a square, and Figure B is a circle outside a square.)
4. If there is a slight shift in the image, my agent would be unable to tell if the two images are the same

If I have more time or computational resources, I will implement the 1st to the 3rd features above and add them to my agent's library. I believe it would dramatically improve my agent's performance.