

## CS 7637 Project 3

Denise Kutnick

[denise@gatech.edu](mailto:denise@gatech.edu)

### Introduction

The Raven's Progressive Matrices are a set of problems aimed to test pattern recognition: each problem provides a square-shaped matrix of patterned images with the bottom right image omitted. Given a variety of choices, one must choose the image that best fits the existing pattern demonstrated in the matrix.

This project introduced two new sets of 3x3 Raven's Problems. As I observed the basic problems within these problem sets, I identified a general pattern for each set:

- **Basic Problems D:** Most images tend to contain the same few shapes in a horizontal/vertical/diagonal line. If there are multiple shapes per image, this pattern could be different for each shape within the image.
- **Basic Problems E:** Most answers are the result of a bitwise AND/OR/XOR between two horizontally/vertically/diagonally adjacent images.

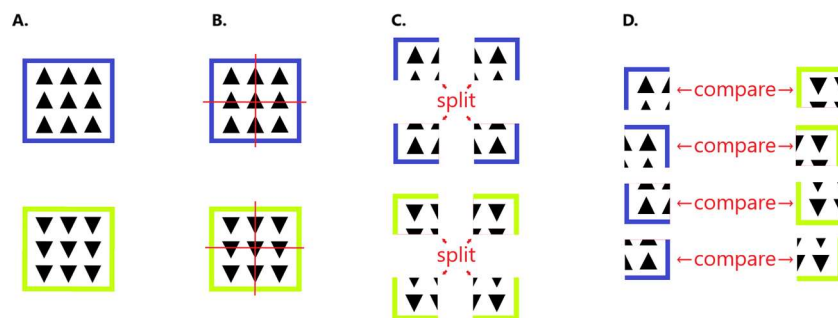
### Approach

I approached this project with a few major goals in mind: leverage my learnings from my existing visual agents, generalize the agent to use the same exact methodologies for both problem sets, and improve performance metrics such as runtime.

In my previous projects, I implemented several different types of visual techniques that can be split into two categories: techniques used to generate potential solutions, and techniques used to test the similarity of the proposed solution to all answer choices. I utilized the same generate-and-test methodology for my current project, because it provides fairly generalized solutions for many different types of problems. Some of the generation techniques I explored when building my agent were the fractal method and affine method, and some of the testing techniques I explored while building my agent were an XOR-based similarity method and the dark pixel method.

## Fractal Method

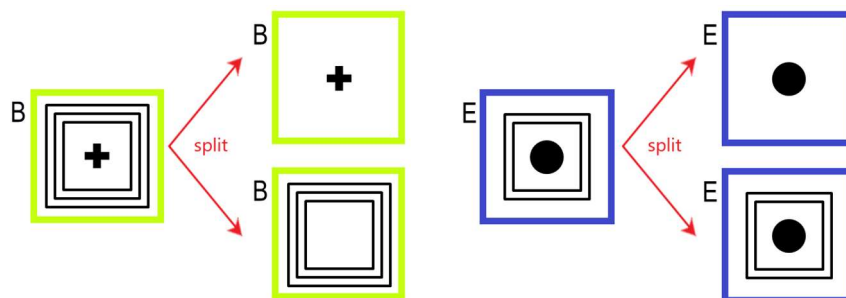
In this project, the first technique that I explored gives my agent the ability to split each image into multiple smaller sub-images. Each sub-image can then be processed separately by my agent. This process is also known as the fractal analogy approach to similarity (McGreggor, Kunda, & Goel, 2010).



**Figure 1.** The fractal analogy approach on two images (Georgia Institute of Technology [GT], 2018).

## Learnings from Previous Projects

The fractal analogy approach was the *least* effective methodology I had in my previous agent, so I was determined to make it the *most* effective methodology in the current project. One of the reasons why my implementation of the fractal method failed in Project 2 was because the fractals I created split the image too symmetrically, causing some sub-images to appear similar when they were not actually similar at all. To fix this, I started splitting the images by shape rather than in a symmetrical manner, and eventually I created a methodology to dynamically split each image by shape. Figure 2 shows the shape-based fractal splitting technique.























**Figure 2.** Shape-based fractal splitting on images B & E from Basic Problem D-06 (GT, 2018).

## Dynamic Fractal Generation

I created the dynamic fractal generation method to generalize the shape-based fractal splitting method introduced earlier. Edge detection has is often used to detect 2D shapes in images (Kumar, Pandey, Pal, & Sharma, 2016), and I believed it could be applied to help create fractals too, so I created a 4-step fractal generation process: apply edge detection on the image, sample the middle row/column of the image, detect the white pixels in those rows/columns, and use the locations of white pixels to create fractal boxes. Table 1 shows dynamic fractal generation on four different types of images: image A has multiple shapes, image B has a single non-filled shape, image C has one shape encasing another, and image D has a single filled shape. Dynamic fractal generation works on many use cases, because it is versatile and generalized.

**Table 1.** Dynamic Fractal Generation steps shown on several different types of Raven's Progressive Matrices images from Basic Problems D (GT, 2018).

	Original	Edges	Splices	Splice Points	Fractals
A.					
B.					
C.					
D.					

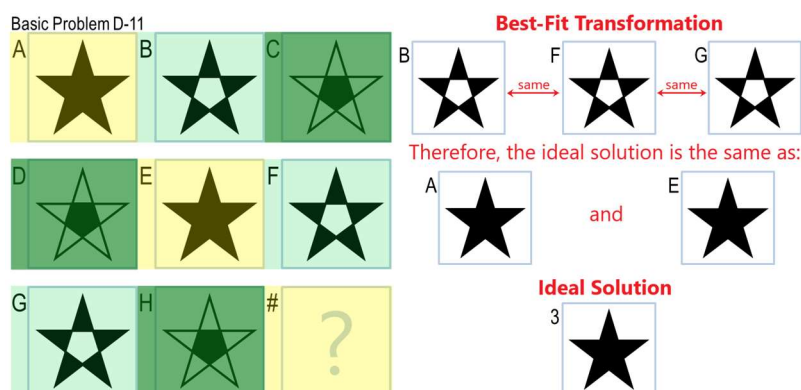
## Affine Method

One of the most effective methods that I explored for detecting relationships between images was the Affine Method. This method identifies a set of transformations that have occurred between multiple images in the matrix, and analogizes these transformations into a possible solution for the Raven's Problem (Kunda, McGregor, & Goel, 2010).

Though the affine method can be used in many ways, some components of background knowledge stay the same across implementations. Since the background knowledge needed for the affine method is mostly relational, I used a semantic network to map these components (see the section on my final agent):

- the initial **known relationship(s)** between two or more images in the matrix (i.e. horizontal, vertical, diagonal).
- the **analogous relationship** that the agent is solving for.
- a formula to describe the **best-fit transformation** between multiple images in a known relationship.
- an **ideal solution**, derived from applying the best-fit transformation to the known images in the analogous relationship.

An example showing each of these components is illustrated in Figure 3 below.



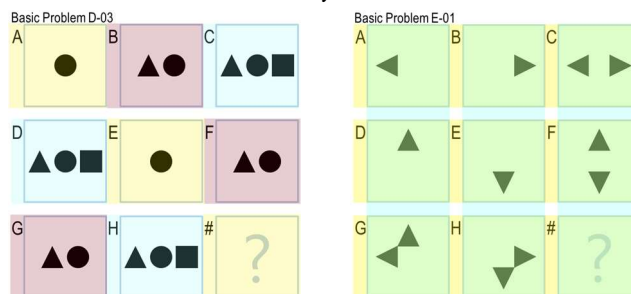
**Figure 3.** Affine Approach on Basic Problem D-11. (GT, 2018). Left: Known relationships in light/dark green, analogous relationship in yellow. Top Right: Best-fit transformation shown on one of the known relationships. Bottom Right: Ideal solution and the derivation process.

## Learnings from Previous Projects

By far, the most important lesson that I learned from using the affine method in my previous projects was that my methodology needed to be lightweight: my original implementation of the generate-and-test format of the affine method generated way too many combinations of transitions/relationships, and attempted to test them all before gaining enough confidence to reach a solution. These tactics led to inefficient use of runtime and space. In this project, I improved my approach in three ways.

### Performing the Affine Approach on 3-Image Relationships

My previous affine approach created a relationship between any two images in the matrix, which is 7 known relationships for each image. By changing my affine approach to consider 3 images in each relationship, my agent obtains the same amount of knowledge by considering just 3 known relationships for each image. This is demonstrated by Basic Problem D-03 in Figure 4. Additionally, for every single problem in Problem Set E, relationships could *only* be detected if the agent considered 3 images at a time. This is demonstrated by Basic Problem E-01 in Figure 4.

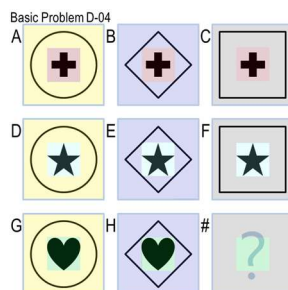


**Figure 4.** Affine Approach using 3-Image relationships on Basic Problems D-03 & E-01 (GT, 2018).

### Performing the Affine Approach on Fractals

Another way that I improved my agent was to apply the affine method to each fractal of the image generated using the dynamic fractal generation method. This allowed me to detect different affine relationships for different shapes in the image. For example, Figure 5 shows Basic Problem D-04, which has a vertical affine relationship on the

outer fractal of the image (circles, diamonds, and squares), and a horizontal affine relationship on the inner fractal of the image (plus signs, stars, and hearts).



**Figure 5.** Affine Approach on the fractals of Basic Problem D-04 (GT, 2018).

### Answer Choice Elimination

My previous agent used the affine method to pick the correct answer for each problem. The agent had to have high confidence to choose an answer, and when confidence levels were not reached, my agent spent some extra iterations trying to gain more confidence. In my current approach, I instead used the affine approach to eliminate obviously-wrong answers from consideration, and used other similarity methods in conjunction with the affine method to choose the correct answer. This made my agent quicker, and also helped with overfitting issues that I had previously.

## XOR Similarity Method

One measure of similarity I relied heavily on was a bitwise XOR between the agent's generated "ideal solution" and each of the answer choices.

The bitwise XOR of two images shows a pixel-by-pixel representation of which pixels are the same color in both images (represented by black in the XOR) and which pixels are different color in each image (represented by white in the XOR). To begin, each image is represented as an array where a black pixel is represented by 0 (in binary: 0'b00000000) and a white pixel is represented by 255 (in binary: 0'b11111111). When two arrays are passed into the bitwise XOR, each corresponding *element* of the matrix is compared by *bit*. Table 2 (on the next page) shows a truth table for this bitwise comparison.

**Table 2.** The truth table of an 8-bit bitwise XOR.

<b>Input Color</b>	<b>Input Bitwise</b>	<b>Output Bitwise</b>	<b>Output Color</b>
white, white	11111111, 11111111	00000000	black
white, black	11111111, 00000000	11111111	white
black, white	00000000, 11111111	11111111	white
black, black	00000000, 00000000	00000000	black

The output of a bitwise XOR can be used in several ways. When comparing a generated solution to the solution choices, an XOR with all (or almost all) 0s generally indicates a close match to the solution. An output XOR can also be used as an input to other statistical methods that calculate similarities between more than two images. This way, the XOR can be used to compare multiple relations with each other. My agent in particular uses the output of a bitwise XOR in the pixel ratio approach.

## Pixel Ratio Similarity Method

My agent's pixel ratio method was created to normalize the XOR similarity method in cases where images are very visually similar, but offset by a few pixels. The method expands upon the concept of a dark pixel ratio (DPR), which measures the number of black pixels in adjacent images and creates a ratio between the corresponding pixel counts (Joyner et. al., 2015).

For each XOR that my agent does, a ratio of similarity is created by calculating the ratio of differences that the XOR sees to the average amount of pixels in the Raven's Problem. This provides a standardized measure of similarity which can be thresholded by the agent as a confidence measure.

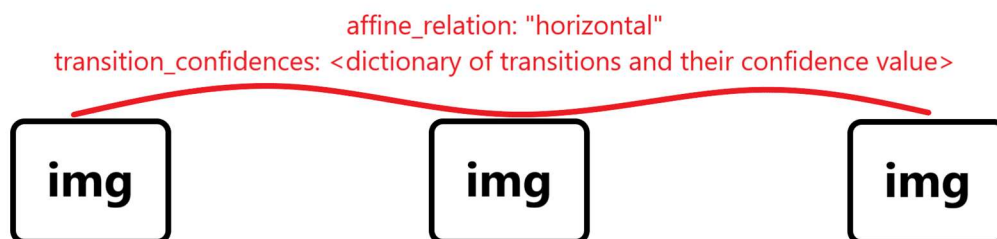
## Final Agent

My final agent is best described as a variation of a technique known as hybrid reasoning, where an agent tries to find the simplest solution to a problem and works its way towards more complex solutions based on the agent's initial confidence of its first solution (Joyner, Bedwell, Graham, Lemmon, Martinez, & Goel, 2015). The agent uses a four-phase process to find its way to a solution: it begins by splitting the problem into smaller sub-problems, then builds a knowledge representation for each

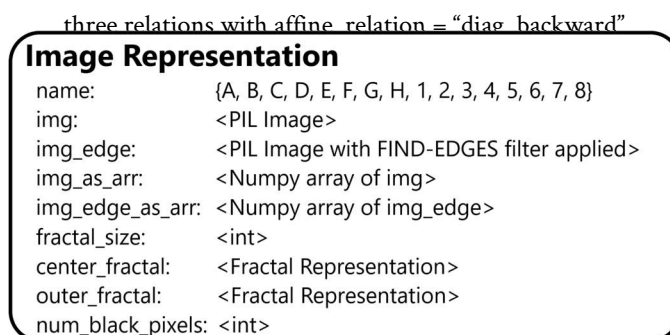
subproblem, then detects potential patterns that occur in each subproblem, then merges the optimal subproblems together to obtain a complete solution.

## Methodology

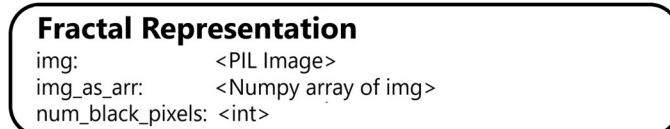
My agent represents knowledge through several hierarchies. A single Raven's problem is represented through a semantic network (see Figure 6), which is made up of image frames (see Figure 7), which is made up of fractal frames (see Figure 8).



**Figure 6.** Sample semantic network representation of three horizontally-related images. The full semantic network for each problem contains three relations with `affine_relation = "horizontal"`, three relations with `affine_relation = "vertical"`, three relations with `affine_relation = "diag_forward"`, and



**Figure 7.** Frame representation of each image in the problem.



**Figure 8.** Frame representation of each fractal in the problem.

## Performance

My agent performed relatively well on the Basic and Test problems. The best performance for each problem set is listed in Table 3 below. It is important to note



that my best results occurred over multiple submissions: although I tried to optimize a single agent to perform well on both D and E problems, my performance on each problem set was 1-2 problems better when the agent was specifically optimized for that problem set.

My agent struggled with overfitting, but it did not struggle with overfitting between the Basic and Test problems like my previous agents had. Instead, there were significant differences in performance between Basic and Challenge problems.

**Table 3.** A summary of my agent’s best performance on each problem set.

<b>Problem Set Name</b>	<b># Correct</b>	<b># Incorrect</b>
Basic Problems D	9	3
Test Problems D	8	4
Raven’s Problems D	7	5
Challenge Problems D	5	7
Basic Problems E	9	3
Test Problems E	9	3
Raven’s Problems E	4	8
Challenge Problems E	2	10

By far, the largest performance difference between my previous project’s agent and this project’s agent is the improved runtime. My previous agent took between 1-2 seconds to solve each problem, whereas my current agent solves each problem in under 0.05 seconds. Individual runtimes are located in the appendices.

## Limitations

My agent was limited in its ability to use the same methodology across both problem sets. Problem Set D generally obtained the most knowledge from my use of the affine approach, while Problem Set E generally obtained the most knowledge from my use of dynamic fractals. When I combined those two knowledge representations, the agent had lower accuracy across each problem set. However, on the questions that the agent got correct with the combined fractal + affine knowledge representation, it was more confident in its answer than when it obtained the answer using just a single

method. This indicates that my agent needed a better fallback plan when it was not confident enough about its solution to match it to an answer.

## Cognitive-Based Design

As I built my initial agent with the affine approach, I paid special attention to the cognitive areas that Raven's Progressive Matrices had intended to test on humans. John C. Raven described his two main indicators of intelligence as "the capacity to think clearly and make sense of complex data (eductive ability); and the capacity to store and reproduce information (reproductive ability)" (1938). Eductive ability is similar to a knowledge representation within an AI agent, and reproductive ability is similar to the learning aspect of an AI agent.

My agent's knowledge representation emulates eductive ability by looking at the image, putting its knowledge about the image into semantic networks and frames, and drawing conclusions about the relationships between each shape in the image. My agent is similar to a human in this way because there is a clearly structured thought process involved in reasoning towards a solution to the problem.

My agent does not use the reproductive ability to solve Raven's Problems. To gain confidence in its solution, my agent uses a dark pixel ratio technique, which examines each of the  $184 \times 184$  pixels in each image and mathematically calculates its confidence in a solution. Humans, on the other hand, use would likely *never* count the  $184 \times 184$  pixels in an image. They would instead use background knowledge and reproductive ability extensively, relying on their past knowledge to influence their future decisions. This is one of the major differences between my agent's approach and a human approach.

## References

1. Georgia Institute of Technology. (2018). Project Overview: Basic Problems. In *CS 7637: Knowledge Based AI: Fall 2018*. Retrieved from <https://drive.google.com/file/d/1ReB1hABAUGPYgS-TtquqqnjZr7dw65t2/view>
2. Goel, A. & Joyner, D. *CS7637: Knowledge-Based AI*, Lesson 12, "Rules of Inference" [Course Video].
3. Joyner, D., Bedwell, D., Graham, C., Lemmon, W., Martinez, O., & Goel, A. (2015). Using Human Computation to Acquire Novel Methods for Addressing Visual Analogy Problems on Intelligence Tests. In *Proceedings of the Sixth International Conference on Computational Creativity*. Provo, Utah.
4. Kumar, V., Pandey, S., Pal, A., & Sharma, S. (2016). Edge Detection Based Shape Identification. In *Cornell University Library*. Retrieved from <https://arxiv.org/abs/1604.02030>
5. Kunda, M., McGreggor, K., & Goel, A. K. (2011). Two Visual Strategies for Solving the Raven's Progressive Matrices Intelligence Test. In *Proceedings of the Twenty Fifth National Conference on AI (AAAI-2011)*, San Francisco, CA.
6. McGreggor, K., Kunda, M., & Goel, A. K. (2010). A Fractal Analogy Approach to Raven's Test of Intelligence. In *Proceedings of the AAAI-2010 Workshop on Visual Representations and Reasoning*, Atlanta, GA.
7. Raven, J. C. (1938). *Raven's Progressive Matrices*. Los Angeles, CA: Western Psychological Services.

## Appendices

### Final Agent Runtime Details

**Table 4.** A summary of my agent’s runtime on Problem Sets D & E.

Problem Name	Runtime	Problem Name	Runtime
Basic Problem D-01	0.046696	Basic Problem E-01	0.038628
Basic Problem D-02	0.045114	Basic Problem E-02	0.038695
Basic Problem D-03	0.045353	Basic Problem E-03	0.041831
Basic Problem D-04	0.041298	Basic Problem E-04	0.041911
Basic Problem D-05	0.040316	Basic Problem E-05	0.047179
Basic Problem D-06	0.040955	Basic Problem E-06	0.042248
Basic Problem D-07	0.046000	Basic Problem E-07	0.038636
Basic Problem D-08	0.045436	Basic Problem E-08	0.038352
Basic Problem D-09	0.042315	Basic Problem E-09	0.038400
Basic Problem D-10	0.045318	Basic Problem E-10	0.042541
Basic Problem D-11	0.041448	Basic Problem E-11	0.040011
Basic Problem D-12	0.045612	Basic Problem E-12	0.047075