Jonathan Satria
March 22, 2015
KBAI / CS 7637
jonathan.satria@gmail.com

Project 3 Design Report

1.0 Introduction

The agent I have designed for project 3 applies a fractal technique to solving Raven's Progressive Matrices test and is heavily based on the approach explored by Dr. McGreggor and his group of researchers. The current implementation of my agent is able to solve 6 2x1 and 10 2x2 RPM problems. Although its success as defined by correct problems solved is poor, I am confident that with more time and careful study the approach can be refined and improved greatly. I believe the fractal technique has immense potential in visual analogy and I am quite satisfied with and proud of the agent I have designed.

Interestingly, I originally attempted to use the propositional approach for solving the RPM test. I used OpenCV (Java) to translate the problem images into a similar form as the problem inputs given in Project 1 and 2. After two weeks of implementation, nearly 1000 lines of code, and mediocre success, I decided to abandon my approach and look for alternatives. The fractal approach seemed most reasonable to me because it seems to represent human cognition at a fundamental level. This human cognition connection will be discussed in more detail later.

In this report I will first discuss the fundamental idea behind fractal encoding which will lay the groundwork for the discussion on the basic design and implementation of my agent. I will then describe the primary observations and results that indicate the current weak points of the agent. Based on these weaknesses, I will discuss some of my attempts to add features and steps to the fractal technique that are meant to address the problems. Lastly, I will discuss how the fractal technique is a superior approach towards visual analogy both in terms of technical implementation and its relationship to human cognition.

2.0 Fractal Fundamentals

The ability to apply the fractal approach in visual analogy problems relies on the fundamental idea that an image in 2-dimensional space is comprised of a limited set of repeating patterns or blocks that are translated and transformed to constitute the overall image. In my background research, I discovered the technique of fractal image compression, a type of lossy compression for 2D images. This process is fascinating and helped me to understand the use of fractals in image processing and so I discuss it in detail here.

In fractal image compression, the memory footprint of an image *S* can be reduced by identifying the repeating units of patterns and encoding them into a fractal encoding. The idea

is that this encoding will reduce the amount of space needed to store the image by re-representing the image as a set of transformations on the unique patterns. The basic process is:

Given an image *S*:

1. Divide S into *n* x *n* "parent blocks". → Store in a set P
2. Further subdivide S into *n/2* x *n/2* "child" blocks → Store in a set D
3. For each element *di* in D
    a. For every element *pi* in P, select the *pi* that is most similar to *di*
        i. Compress the subimage *pi* to be the same pixel size as *di*
        ii. Transform and calculate the similarity of *pi* and *di* .
            - Eligible transformations are Identity, Horizontal Flip, Vertical Flip, Rotation 90, Rotation 180 and Rotation 270 degrees.
            - Similarity is calculated as the lowest Mean Squared Error between *di* and *si*.

The image below illustrates the partitioning method of the above process.

This image is divided into 16 parent blocks, each of which are further subdivided into 4 child blocks. The parent blocks are identified as the "domain" or "source" image and the child blocks are considered the "range" subimages.

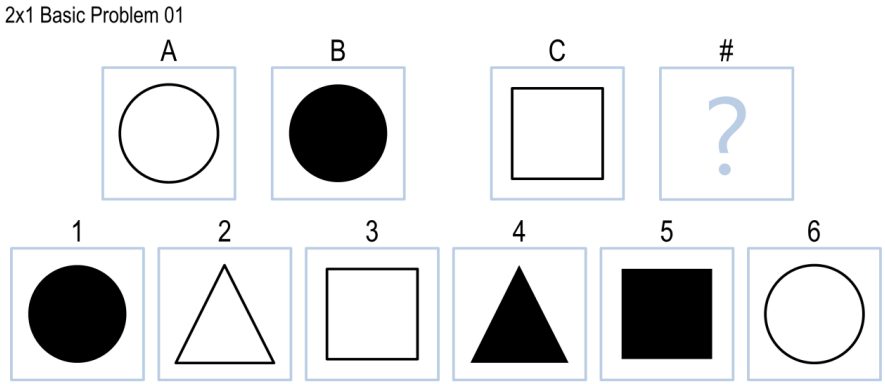
It is important to mention that the reason domain images are taken to be twice the size of range subimages is because of the need for the fractal encoding of an image to converge upon itself given a compressed image. While this is important for fractal image compression, it is not so for visual analogy problems. I will discuss this topic shortly.

After partitioning the image into blocks, each smaller child block *di* is compared to every parent blocks for similarity. In order to do this, the parent block must first be rescaled to match the dimensions of the child block through compression or some other related technique. The parent block undergoes a set of individual affine transformations. Each generated transformation is compared to *di* for similarity by measuring the mean squared error of corresponding pixel grayscale values.

Once a parent block with the highest similarity is selected, the location of the parent block, the child block, the affine transformation, and the grayscale colorshift is stored as a fractal code. The collection of fractal codes for all images in D forms the fractal encoding of an image.

Fractal Approach Towards Visual Analogy

Equipped with this understanding of fractal image compression, it is not difficult to imagine how it can be extended and applied to solve visual analogies. Consider the following 2x1 Ravens Progressive Matrix Problem:



There exists an unknown transformation *T* that transforms A → B and similarly, a transformation *T'* that will transform C into the correct answer. These transformations can be represented by the fractal encoding of a source image into a destination image (e.g. A → B). The actual representation of these fractal codes are stored as fractal *features* which will be discussed next. Nonetheless, the crux of the fractal technique for solving visual analogies is that the transformation from *T* is analogous to *T'*. We can define the most analogous relationship between *T'* and *T'* as the one that has the highest intersection of fractal features.

Fractal features are combinations of the elements of a fractal code. A fractal code might be represented in the following form:

$$\langle\langle s_x, s_y\rangle, \langle d_x, d_y\rangle, k, c\rangle$$

Where $\langle s_x, s_y\rangle$ represents the top left coordinate position of the source subimage and $\langle d_x, d_y\rangle$ represents the top left coordinate position of the destination subimage. *k* represents the affine transformation where $k \in \{ I, HF, VF, R90, R180, R270\}$ (identity, horizontal-flip, vertical-flip, rotation 90 degrees, rotation 180 degrees, and rotation 270 degrees}. *c* represents the overall colorshift of the source and destination subimages.

Although a fractal code represents a succinct wealth of information, it is more effective to include combinations of the element codes into the transformations as described above. Fractal features might include:

<<sx, sy>, <dx, dy>, k, c >: a specific feature;
<< dx - sx, dy- sy>, k, c >: a position agnostic feature;
<<sx, sy>, <dx, dy>, c >: an affine transform agnostic feature;
<<sx, sy>, <dx, dy>, k >: a color agnostic feature;
<k, c>: an affine specific feature;
<c>: a color shift specific feature.
<dx-sx, dy-sy>: offset
<abs.(dx-sx), abs.(dy-sy)>: displacement

In the context of a 2x1 visual analogy problem, the transformation T is calculated for A→ B. Then, each transformation T_i is calculated for each answer candidate *i*. The similarity of T_i to T is calculated using Tversky's formula:

$$\textbf{Similarity}(T, T_i) = \textbf{f}(T \cap T_i) / \textbf{f}(T)$$

where f(T) is defined as the number of fractal features in T

Let us now define the steps required to apply the fractal technique for visual analogies. Note that the process is very similar to fractal compression, although here we include steps necessary to compare answer choices and identify the most similar transformation.

Process 1

Fractal Technique for Geometric Analogy (in 2x1 RPM)

Partitioning

1. Partition the *B* frame into grid blocks b_i of some pre-defined grid size *n*.
2. Partition the *A* frame into grid blocks a_i into grid blocks of size *2n*.

A→ B Fractal Feature Generation

3. For each image b_i

Source Image Selection

a. Scan the source image and for each a_i
   i. Apply affine transformations to a_i and compare for similarity
   ii. Compare similarity by measuring the mean squared error (MSE). A lower MSE indicates higher similarity.
b. Once the source image has been, chosen identify the fractal code that specifies the fractal encoding. Generate defined fractal features from the code and store in transformation T

Candidate Answer Selection

4. For each answer selection Ans_i
   a. Repeat steps 1-3 (replacing C and Ans_i for A and B, respectively) and store fractal features in transformation T'
   b. Generate the similarity value S_i of T and T'
5. Select the answer that produces the highest similarity value.

Although the above process outline addresses only 2x1 RPM problems, it is not difficult to extend this process to 2x2 problems as well. The following outline describes how the fractal technique can be applied to 2x2 geometric analogies.

Process 2

Fractal Technique for Geometric Analogy (in 2x2 RPM)

Partitioning

1. Partition the *B* frame into grid blocks b_i of some pre-defined grid size *n*.
2. Partition the *C* frame into grid blocks c_i of some pre-defined grid size *n*.
3. Partition the *A* frame into grid blocks a_i into grid blocks of size *2n*.

A→ B Fractal Feature Generation

4. For each image b_i

Source Image Selection

a. Scan the source image and for each a_i
   i. Apply affine transformations to a_i and compare for similarity
   ii. Compare similarity by measuring the mean squared error (MSE). A lower MSE indicates higher similarity.

b. Once the source image has been, chosen identify the fractal code that specifies the fractal encoding. Generate defined fractal features from the code and store in transformation T_b

A→ C Fractal Feature Generation

5. For each image c_i

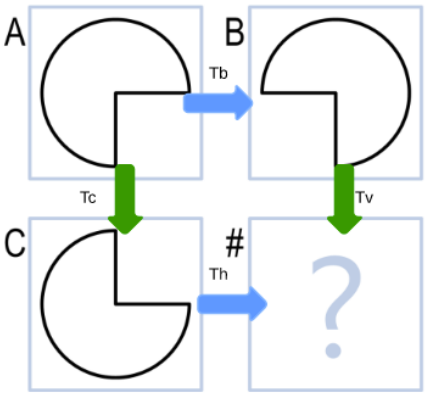
Source Image Selection

a. Scan the source image and for each a_i
   i. Apply affine transformations to a_i and compare for similarity
   ii. Compare similarity by measuring the mean squared error (MSE). A lower MSE indicates higher similarity.

b. Once the source image has been, chosen identify the fractal code that specifies the fractal encoding. Generate defined fractal features from the code and store in transformation T_c

Candidate Answer Selection

6. For each answer selection Ans_i
   a. Repeat step 4 (replacing C and Ans_i for A and B, respectively) and store fractal features in transformation T_h
   b. Repeat step 4 (replacing B and Ans_i for A and C, respectively) and store fractal features in transformation T_v
   c. Generate the similarity value S_h of T and T_h
   d. Generate the similarity value S_v of T and T_v
   e. Calculate overall similarity by taking the average of T_h and T_v
7. Select the answer that produces the highest similarity value.



The figure on the left illustrates the 2x2 concept where transformations in the horizontal and vertical directions are compared with each other. The process is very similar to 2x1s, except that an additional dimension is accounted for.

3.0 Design and Implementation

The agent I have designed follows the fractal technique for geometric analogies described above (in Process 1 and Process 2). The pseudocode below describes my agent at a more technical level, specifically highlighting the pre-feature generating step of converting and partitioning the images.

Agent Pseudocode

1. For each “frame” image A, B, C, and each of the answer choices
    a. Convert the image to binary by taking the grayscale value and applying a threshold filter
    b. Partition the images into blocks, also referred to as chunks. Pad the image as necessary in order to obtain equal sized chunks.
2. Generate the fractal features as described in the processes above. The following fractal features are generated:
    - <<sx, sy>, <dx, dy>, k, c >: a specific feature;
    - << dx - sx, dy- sy>, k, c >: a position agnostic feature;
    - <<sx, sy>, <dx, dy>, c >: an affine transform agnostic feature;
    - <<sx, sy>, <dx, dy>, k >: a color agnostic feature;
    - <k, c>: an affine specific feature;
    - <c>: a color shift specific feature.
    - <dx-sx, dy-sy>: offset
    - <abs.(dx-sx), abs.(dy-sy)>: displacement
3. Calculate similarity of each answer choice and return the highest answer choice.

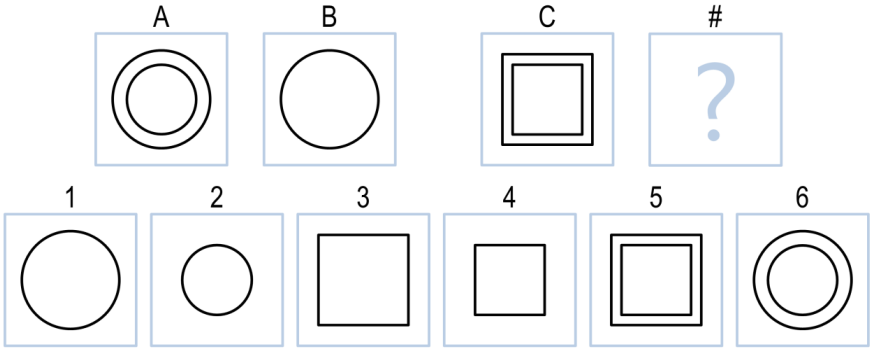
As part of my experimentation efforts, I introduced several parameters to toggle features of the agent. I will describe the motivation for these in the next section, but I mention them briefly here as they are a feature set of the agent. The following features are configurable (within source) in `Configuration.java`.

- `CHUNK_SIZE`: The pixel dimension (integer) of the chunks. e.g. 8 for an 8x8 chunk
- `MSE_THRESHOLD`: The maximum mean squared error value a transformation can have to be considered as a fractal code
- `IGNORE_WHITESPACE`: If the destination image d_i only contains whitespace values, ignore it and do not generate a fractal code
- `TRIM`: Crop all frames to remove extra whitespace
- `INCLUDE_SHAPE_SIMILARITY`: Add the similarity of an answer choice and B frame to the total similarity.

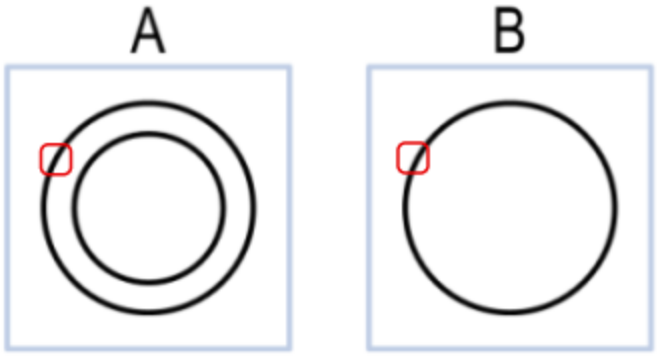
Sample Execution

I will now demonstrate, albeit in condensed form, how my agent works on the 2x1 Ravens Problem 4

2x1 Basic Problem 04



Suppose the image has already been divided into chunks of size 8x8 pixels. The agent is now in its searching and encoding phase. Now suppose the agent is searching for the corresponding s_i for d_i where the coordinates of the chunk d_i are <40, 80>.



The agent has found a chunk s_i at <40,80> that requires the identity transform (stays the same) and generates the following fractal code:

<<40,80>, <40,80>, I, 0>

This fractal code identifies that the corresponding parent chunk can be found at <40,80>, the destination chunk is at <40,80>, an identity transform was applied and there was no colorshift. The agent then generates the following fractal features:

<<sx, sy>, <dx, dy>, k, c >	<<40,80>, <40,80>, I, 0>
<< dx - sx, dy- sy>, k, c >	<0, 0, I, 0>
<<sx, sy>, <dx, dy>, c >	<<40,80>, <40,80>, 0>
<<sx, sy>, <dx, dy>, k >	<<40,80>, <40,80>, I>
<k, c>	<I, 0>
<k>	<I>
<dx-sx, dy-sy>	<0, 0>
<abs.(dx-sx), abs.(dy-sy)>	<0,0>

Note that there are identical and repeated fractal features. Technically speaking, these fractal features are stored in HashSet and so duplicates are not accounted for. It is the fractal feature that is of interest to this technique - not the number a fractal feature appears.

After the step of generating fractal features is completed for A → B, this process is repeated for each of the C → Answer choices. In a certain configuration, my agent will produce the following values:

A->B 89 features found.
Evaluating Answer 1
C-0 100 features found.
Intersection: 9
Fractal Feature Similarity: 0.10112359550561797
Evaluating Answer 2
C-1 100 features found.
Intersection: 8
Fractal Feature Similarity: 0.0898876404494382
Evaluating Answer 3
C-2 93 features found.
Intersection: 11
Fractal Feature Similarity: 0.12359550561797752
Evaluating Answer 4
C-3 88 features found.
Intersection: 11
Fractal Feature Similarity: 0.12359550561797752
Evaluating Answer 5
C-4 89 features found.
Intersection: 10
Fractal Feature Similarity: 0.11235955056179775
Evaluating Answer 6
C-5 106 features found.
Intersection: 6
Fractal Feature Similarity: 0.06741573033707865

The similarity is calculated as a function of the intersection size divided by the number of features in A → B. Based on the similarity values, it is apparent that the candidate answer choice is 3 (although 4 is the same with regards to similarity, 3 is the first iteratively and is set as the highest scoring candidate).

In summary of this example, it is hopefully clear how the agent is able to construct the fractal features set of the geometric analogies and identify similarity.

4.0 Experimentation and Configuration Optimization

Professor Goel described the agent and its parameters perfectly as a radio that has many fine tune control knobs. I designed my agent to be easily configurable so that I could easily

experiment with different parameters and find the optimal configuration. To do this, I created a Sampler (not included with the source code) to iterate through all permutations of the configurations. The experiment lasted nearly 4 hours, but I was able to optimize the configurations for 2x1 and 2x2 RPMs independently. Those parameters are below:

2x1 RPM Configuration

- `CHUNK_SIZE`: 40
- `MSE_THRESHOLD`: 0.4
- `IGNORE_WHITESPACE`: true
- `TRIM`: false
- `INCLUDE_SHAPE_SIMILARITY`: false

2x2 RPM Configuration

- `CHUNK_SIZE`: 48
- `MSE_THRESHOLD`: 0.2
- `IGNORE_WHITESPACE`: false
- `TRIM`: false
- `INCLUDE_SHAPE_SIMILARITY`: false

It is difficult to exactly explain for the specific values in the configurations and the reasons for the difference between 2x1 and 2x2 configurations. However, what is interesting to point out and discuss is the optimal chunk size. Originally I had thought a smaller chunk size (of say 4x4) would yield the best results because it allowed for improved sub-image matching. This was certainly not the case. When reducing to smaller chunk sizes, computational time increases exponentially while success on the problems decreased.

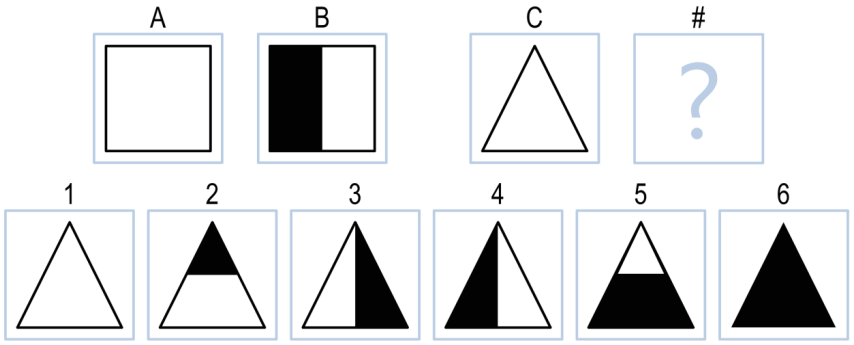
5.0 Discussion

Reasoning to Correct Answers

Throughout my experimentation, it appears that my agent does very well in analogies in which a object deletion occurs and there are partial fills. With regards to object deletion, I believe my agent does quite well because the process by which I generate fractal features ignores the deleted object entirely. In other words, the agent is generating the fractal features from A → B by searching A for an equivalent subimage that would comprised B. Deletion is essentially accounted for by not generating any fractal features. It is interesting to note that Dr. McGreggor also mentions computing a mutual fractal representation set which also contains fractal features generated in the B → A transformation. Had I included this process, my agent would have likely selected an incorrect answer.

With regards to partial fills such as the following problem, it is easy to explain that intersection of the fill is causing a higher similarity value. In this case, answer 4 has the highest

2x1 Basic Problem 10



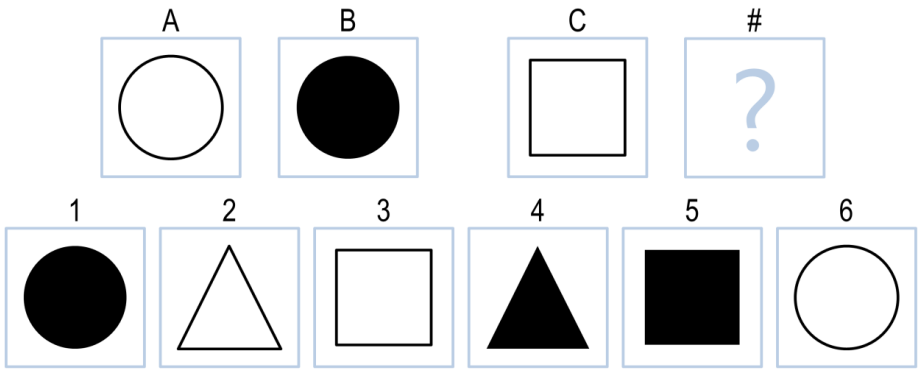
In this case, answer 4 has the highest similarity intersection of filled area and is correctly selected.

Identifying the correct reasoning abilities of the agent is an aspect I would like to investigate further. There is uncertainty surrounding the ability of the agent to solve complex problems such as 2x1 Basic Problem 20, but inability to solve “easy” problems such as 2x1 Basic Problem 01.

Weaknesses and Mistakes

From the first set of problems solved until the very last, it was apparent that the fractal technique seems to fail in seemingly easy problems such as 2x1 Basic Problem 01.

2x1 Basic Problem 01



The agent would continually choose answers 1 or 6. Although incorrect, this reasoning was actually quite interesting to me because both 1 and 6 are circles which reflect the circle in B. This was fascinating to me because it demonstrated to some extent the fractal encoding method causing an image (C) to be transformed to another (B).

My idea for why this was happening was because of the number of white space or background chunks that were being encoded into fractal features. Because a circle has more white space surrounding it than a box that bounds it, answers like 1 and 6 had more similar fractal features than answer choices 3 and 6.

In order to address this discrepancy, I introduced features to trim the white space, ignore whitespace chunks completely (whether they surround or are inside an object), and to compare object similarity by evaluating pixel intersection. Unfortunately, none of these options worked for this particular problem.

This problem with background whitespace noise can also be reconsidered as an issue where a particular problem set contains a knowledge frame (e.g. B) as an answer choice. Because of the nature of how the fractal features are generated from A→ B, it would seem likely that the agent would tend for answers that resemble B.

Improvements

As it stands, the agent I have implemented is a basic fractal technique with a few features added in an attempt to solve individual problems. In projects 1 and 2, it was easier to fix individual problems through exception cases and production rules that handled unique circumstances. In the fractal approach, the relationship between image characteristics and geometric analogy is less about image characteristics and much more granular and mathematical. Therefore, I must take a different approach towards If given more time and focused study, I would have improved on several aspects, described below.

Identifying Relationships between fractal features and transformations

Being able to identify the conditions and types of problems the agent is more successful in will allow me to better address those issues. I suspect that the agent will need to evaluate the problem and determine an optimal configuration to solve it. For example, changing chunk sizes might be helpful depending on the type problem.

Image Comparison using Fractal Decompression

An interesting approach suggested by peers who also attempted the fractal technique is to apply the fractal encoding on the C frame to generate the “ideal” answer. This generated answer can then be compared to each answer choice for similarity, and the most similar answer choice is selected. I think this approach may even improve computational time of the agent since the fractal encoding step does not need to be generated for each answer.

Increased Confidence Intervals and Analysis

Regretfully I was unable to implement any sort of confidence analysis on the chosen answer choice. If a degree of certainty can be measured by the agent, I think that it would provide helpful advice to more accurately select the correct answer. I would like to follow an approach similar to the one outlined in Dr. McGreggors paper “Confident Reasoning on Raven’s Progressive Matrices Tests”.

6.0 Relation to Human Cognition

As mentioned before, I chose the fractal approach after abandoning the propositional approach in which the visual input is converted to a verbal representation. The propositional approach using an image library such as OpenCV required an incredible amount of code to perform simple tasks. If one of the learning goals of this course and project is to identify underlying cognitive processes through the study and design of intelligent agents, then writing 1000 lines of code to perform shape detection does not achieve that goal personally. It seemed unreasonably challenging to code the simplest task of shape detection and size comparison, when such a task is nearly instantaneous in human cognition.

I also noticed that when solving RPM problems, I did not need to perform verbal reasoning to solve the problem. I identified patterns in the image and applied them. Although verbal reasoning contains a wealth of information, it seems like an unnatural way for humans to think about visual problems. Indeed, a verbal representation of the problems captures the essence and depth of an RPM problem. But to me, it is not the fundamental and basic process by which we solve visual problems.

Instead, it is through learning a set of building blocks and applying patterns of these blocks to build the different objects in the world around us that we can identify and reason over different visual inputs, no matter their shape, size, color, or other spatial characteristic. The fractal technique is built on this idea and it seemed most likely in solving the RPM problems.

7.0 References

[1] McGreggor, Goel *2014* **Confident Reasoning on Raven's Progressive Matrice Tests** http://dilab.gatech.edu/test/wp-content/uploads/2014/11/ConfidentReasoningRavensAAAI2014Final.pdf

[2] McGreggor, Kunda, Goel *2010* **A Fractal Analogy Approach to the Raven's Test of Intelligence** http://www.aaai.org/ocs/index.php/WS/AAAIW10/paper/viewFile/2025/2437