Jonathan Satria
April 12, 2015
KBAI / CS 7637
jonathan.satria@gmail.com

Project 4 Design Report

1.0 Introduction

The agent I have submitted for project 4 extends on my project 3 approach of using the fractal technique to solve Raven's Progressive Matrices test. It is based on the approach explored by Dr. McGreggor and his group of researchers. The current implementation of my agent is able to solve 8 2x1, 12 2x2, and 8 3x3 RPM problems. Although its success as defined by correct problems solved is poor compared to the propositional approaches in Project 1 and 2, 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 reasoning and I am quite satisfied with and proud of the agent I have designed.

As mentioned in my project 3 report, 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 these problems. Lastly, I will discuss how the fractal technique is a superior approach towards visual reasoning both in terms of technical implementation and its relationship to human cognition.

2.0 Fractal Fundamentals

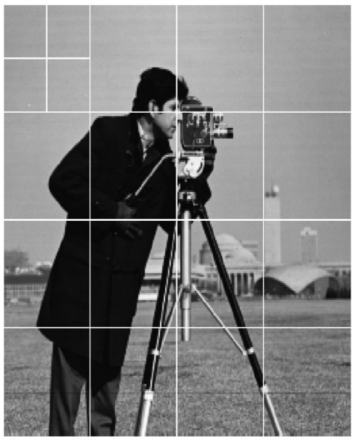
The ability to apply the fractal approach in visual analogy problems relies on the fundamental idea that an image in 2-dimensional space is comprised of a limited set of repeating patterns or blocks that are translated and transformed to constitute the overall image. In my background research, I discovered the technique of fractal image compression, a type of image compression. 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.

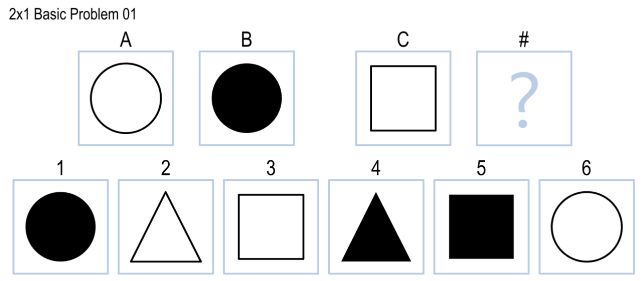
It is important to mention that the reason domain images are taken to be twice the size of range subimages is because of the need for the fractal encoding of an image to converge upon itself given a compressed image. While this is important for fractal image compression, it is not so for visual analogy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

$$<<s_x, s_y>, <d_x, d_y>, k, c>$$

Where $<s_x, s_y>$ represents the top left coordinate position of the source subimage and $<d_x, d_y>$ represents the top left coordinate position of the destination subimage. *k* represents the affine transformation where *k* ∈ *{ 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 simple unit of information, it is more effective to include combinations of the element codes into the transformations as described above into tuples known as fractal features. These fractal features might include the following:

<<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:

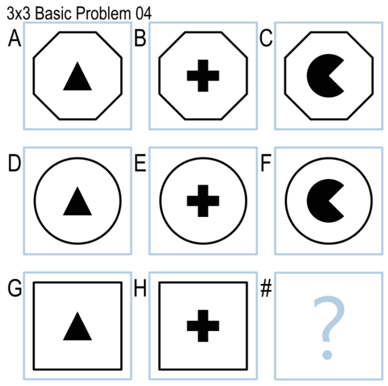
$$\textbf{Similarity}(T, T_i) = \textbf{f}(T \cap T_i) \,/\, \textbf{f}(T)$$

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

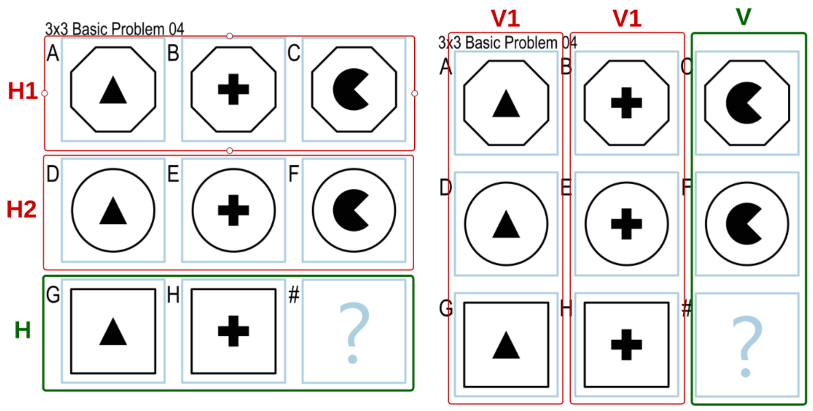
Solving 3x3 RPMs

Solving 3x3 RPMs requires additional fractal feature computation and comparisons because of the additional horizontal and vertical relationships. The general approach for 2x1, 2x2, and 3x3 problems are similar except for additional calculations for the different dimensional relationships. Please refer to my project 3 report for details on how 2x1 and 2x2 problems are solved.

To make the process more clear, suppose the agent is solving the following 3x3 RPM:



As discussed, the idea is that the transformation across each dimension are analogous to one another in some way. By constructing the simultaneous horizontal and vertical relationships of the known knowledge of the problem, the correctness of each answer choice can be understood as a measure of the similarity between the vertical and horizontal relationships. More specifically, A → B → C and D → E → F represents the horizontal relationships H1 and H2 respectively, while A → D → G and B → E → H represent the vertical relationships V1 and V2, respectively. These relationships constrain the horizontal relationship G → H → ? and C → F → ?. These relationships are illustrated below:



Fractal features for each relationship, H1, H2, V1, and V2 are first constructed as sets of fractal features which contain the fractal features from each transformation in a dimension. For example:

$$H1 = \text{Fractal}(A, B) \cup \text{Fractal}(B, C) \cup \text{Fractal}(A, C)$$

Then, for each answer choice A_i, the fractal features for the horizontal relationship H and vertical relationship V are calculated. The similarities between H and the horizontal relationships and V and the vertical relationships form the set Θ.

$$\Theta = \{ S(H1, H(A_i)), S(H2, H(A_i)), S(V1, V(A_i)), S(V2, V(A_i)) \}$$

where $S(T, T_i)$ is defined as before:

$$\mathbf{S}(T, T_i) = \mathbf{f}(T \cap T_i) / \mathbf{f}(T)$$

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

The *overall* similarity O can be calculated as the euclidean distance vector:

$$O = \sqrt{}(S(H1, H(A_i))^2 + S(H2, H(A_i))^2 + S(V1, V(A_i))^2 + S(V2, V(A_i)^2)$$

Below is the pseudocode for solving 3x3 RPMs

```
Fractal Technique for 3x3 Geometric Analogy
    Partitioning
        1. Partition all knowledge frames (A, B, C, D, E, F, G, and H) into grid blocks $g_i$
           of some pre-defined grid size $n$.
    H1 Fractal Feature Generation
        2. Generate the fractal features Fractal(A, B)
            a. For each subimage in B $b_i$ scan the source image A 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 $T_b$
        3. Generate the fractal features Fractal(B, C) as in step 2
        4. Generate the fractal features Fractal(A, C)
    H2 Fractal Feature Generation
        5. Repeat the steps as in H1 for figures D, E, and F
    V1 Fractal Feature Generation
        6. Repeat the steps as in H1 for figures A, D, and G
    V2 Fractal Feature Generation
        7. Repeat the steps as in H1 for figures B, E, and H
    Candidate Answer Selection
        6. For each answer selection $Ans_i$
            a. Determine H by constructing the fractal features of G, H, and $Ans_i$.
            b. Determine V by constructing the fractal features of C, F, and $Ans_i$.
            c. Generate the similarity value $S_{h1}$ of H1 and H
            d. Generate the similarity value $S_{h2}$ of H2 and H
            e. Generate the similarity value $S_{v1}$ of V1 and V
            f. Generate the similarity value $S_{v2}$ of V2 and V
            g. Calculate overall similarity by calculating the Euclidean distance
        7. Select the answer that produces the highest similarity value.
```

3.0 Design and Implementation

The agent I have designed follows the fractal technique for geometric analogies described above. 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

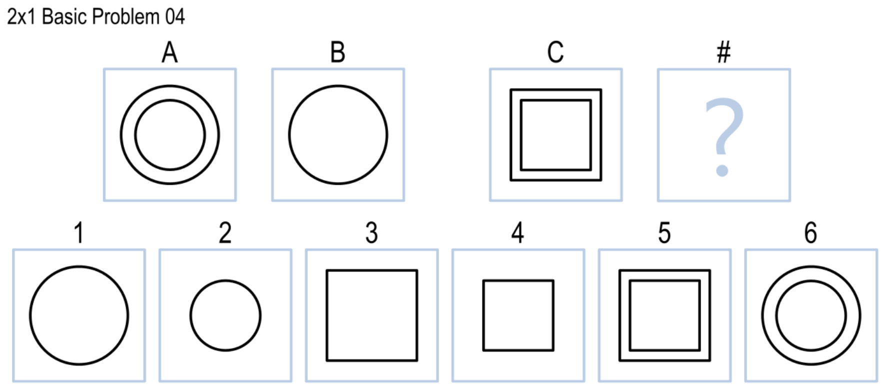
```
1. For each "frame" image A, B, C, and each of the answer choices
    a. Convert the image to binary by taking the grayscale value and applying a
       threshold filter
    b. Partition the images into blocks, also referred to as chunks. Pad the image as
       necessary in order to obtain equal 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.`

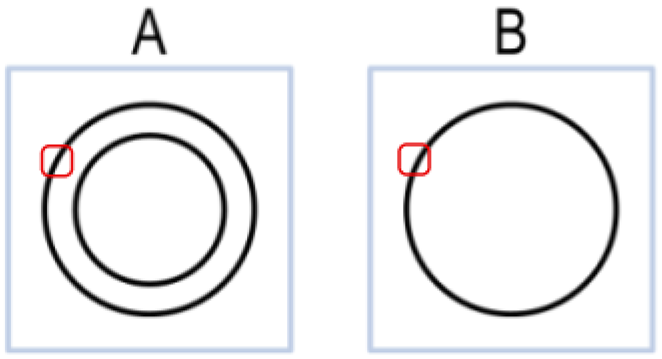
- `CHUNK_SIZE`: The pixel dimension (integer) of the chunks. e.g. 8 for an 8x8 chunk
- `MSE_THRESHOLD`: The maximum mean squared error value a transformation can have to be considered as a fractal code
- `IGNORE_WHITESPACE`: If the destination image d_i only contains whitespace values, ignore it and 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, in condensed form, how my agent works on the 2x1 Ravens Problem 4. Note again that a 3x3 problem would be solved in similar fashion. I have merely excluded it because of the length of discussion it would entail.



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

Resolving Ambiguity

Dr. McGreggors paper “Confident Reasoning on Raven’s Progressive Matrice Tests” establishes an approach for resolving ambiguity and “ties” between two candidate answer choices. It outlines a process of:

- Measuring the similarity of an answer choice at multiple abstraction levels (partition sizes)
- Calculating the mean, standard deviation, and standard error to establish a confidence metric
- Choosing the answer with the highest confidence

Although absent from the code, I implemented this approach but with minimal success. For 2x1 and 2x2 problems, the agent did the same or slightly worse, while the number of 3x3 problems solved dropped to 2/20. This may be due to inherent weaknesses of the current agent and its generation of fractal features. If given more time, I would include this approach because it provides a more complete measure of similarity based on confidence instead of solely relying on the highest similarity value.

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 6 hours, but I was able to optimize the configurations for 2x1, 2x2 and 3x3 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

3x3 RPM Configuration

- `CHUNK_SIZE`: 40
- `MSE_THRESHOLD`: 0.2
- `IGNORE_WHITESPACE`: false
- `TRIM`: false
- `INCLUDE_SHAPE_SIMILARITY`: false

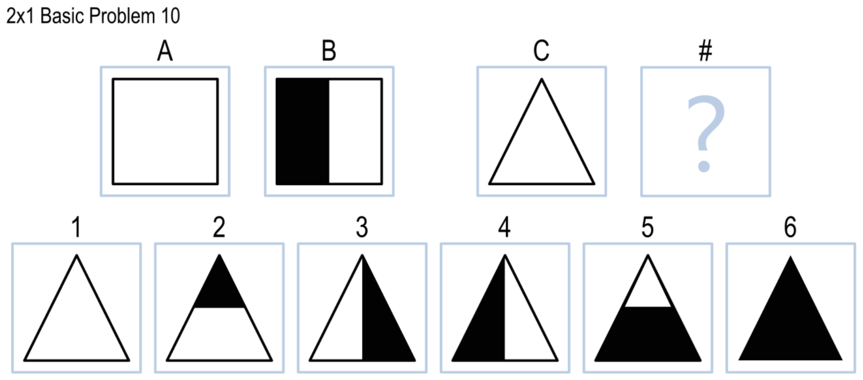
It is difficult to exactly explain for the specific values in the configurations and the reasons for the difference between 2x1 and 2x2 configurations. However, what is interesting to point out and discuss is the optimal chunk size. Originally I had thought a smaller chunk size (of say 4x4) would yield 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

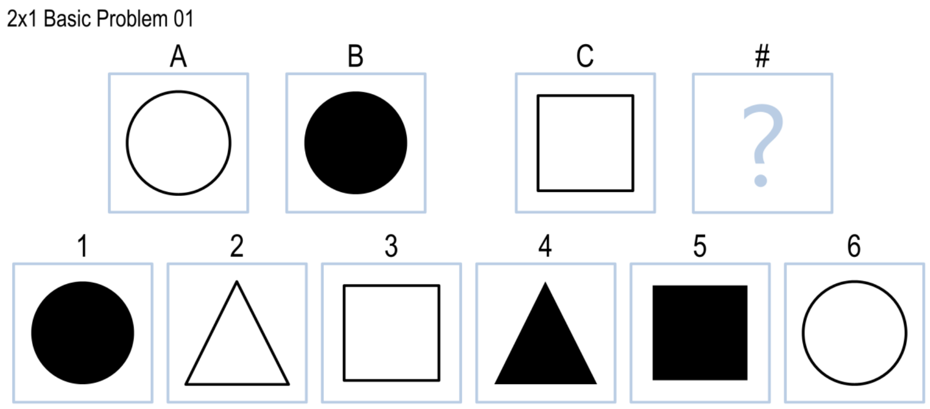


In this case, answer 4 has the highest similarity intersection of filled area and is correctly selected.

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.



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.

Improved Partitioning Scheme

At the moment, the agent applies a feature agnostic and basic gridding scheme based on a defined chunk size. It would be helpful to implement different approaches in which the image in partitioned in a more significant manner. For example, partitioning by colorimetric characteristics or other level of detail. Optimizing the partitioning method has two interesting implications:

- It may reflect the scale at which humans identify and abstract visual problems. A greater chunk size represents a higher level of abstraction of detail.
- By ignoring large contiguous areas of white space, the agent will produce less features for uninteresting blocks. This should also speed up the agent.

Additional Fractal Features

It would be interesting to explore additional fractal features that are meaningful to the agent. For example, a fractal feature that captures the gradient change of a block might be meaningful in distinguishing edge blocks in non-grayscale images.

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.

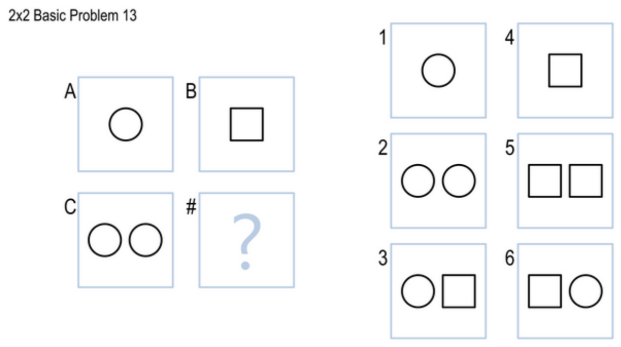
6.0 Visual Reasoning vs Verbal Reasoning

In Projects 1 and 2, the inputs to the RPM problems were provided as sets of characteristics about the different objects and figures. Projects 3 and 4 required the additional step of visual reasoning to either identify these characteristics or attempt an entirely different approach. This provided a unique opportunity to take a step back and rethink how humans would think and solve RPM problems. As mentioned in the introduction, I ultimately chose to pursue the purely visual reasoning approach. However, both verbal and reasoning approaches have unique challenges and opportunities that I will discuss briefly here.

Verbal Reasoning
Verbal reasoning was a straightforward way to get started with designing intelligent agents because it provided the a concise and complete level of abstraction of problem details. Each image input file contained all of the *essential* details needed to reason, such as shape, fill, size, and spatial location. Additionally, verbal reasoning afforded a simple way of representing the objects in object oriented languages such as Java. Though this fact may seem trivial, throughout the design of Project 1/2 agent and my fractal agent for Project 3/4, I found this fact intriguing because it encouraged me to explore how humans might think. I will discuss question more intensively in the last section.

However, the verbal reasoning approach was not without limitations. First, characteristics like rotation/reflection and fill became increasingly difficult to address. For shapes with rotational symmetry, it was important to implement a means to detect that a shape rotated by its rotational angle would be identical to itself. Although represented differently in computer memory, visually speaking the object was identical. Secondly, I also recall how difficult it became "hard-coding" unique production rules in Project 2 for specific problems due to the nature of how objects were described. For example, handling spatial relationships were difficult because evaluating for moving objects required maintaining some sort of internal map of where objects were stored and track their movements.

Visual Reasoning
In visual reasoning, many of the above challenges were forgotten and replaced with new ones as well as new opportunities. Visual reasoning allowed the flexibility of freeing the agent from strict and defined characteristics about the problem given by the input files. One of the unique opportunities of visual reasoning is the ability to choose different levels of abstraction needed for a problem. Consider the following problem:



In this problem, it is not exactly necessary to know that in figure C two circles are side by side to each other. None of the answer choices break this side-by-side relationship, so it makes sense to ignore it.

Perhaps the biggest opportunity, as well as challenge, that visual reasoning creates is the need to explore how humans visually identify and think about the objects we see in the world. In the fractal approach, the underlying idea is that we perceive objects in the world as a repeating patterns that can be identified, transformed and composed to create new objects. Unfortunately, the challenge that I faced with this opportunity is enabling an intelligent agent written in lines of code to identify these patterns and reason over them. It seems that fractal features for one set of problems might not be necessarily helpful, and may actually be detrimental to the success of another set of problems. It is very difficult to discern which attributes would be helpful for each type of problem.

7.0 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 myself, I did not need to perform verbal reasoning to solve the problem. Instead, 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. Therefore, I believe that the agent I have designed does attempt to solve RPM problems in the same a human does.

8.0 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