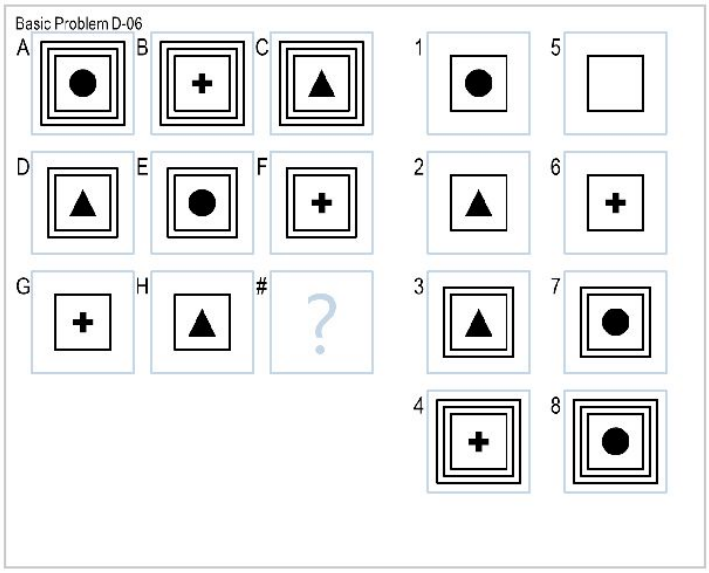
Traci Fairchild
CS7637 KBAI
Project 3
Project Reflection
November 20, 2015

Project Reflection for Solving 3x3 Raven's Progressive Matrices using Only Visual Representations

Raven's Progressive Matrices

Raven's Progressive Matrices, referred to as RPM, is a group of nonverbal tests that are typically used to measure a person's reasoning skills. Reasoning is a skill commonly thought to be an adequate measure of a person's intelligence. The tests are a series of images in which one image relates to another in a certain way. The test taker is expected to analyze the relationship, in our example below, between the images below, looking for similarities or differences. The agent is expected to choose the image (images 1 through 8 below) that best completes the analogy of the first set. For example, a given 3x3 RPM test could look like the image below:



The Project

In this project I had to write an agent that can pass this human intelligence test. Project 1 and Project 2 relied heavily on the verbal representations of the problem set. This project, Project 3 had no such information and in processing the problems the agent had to rely solely on the visual representations of the images. The goal is to design the agent so that it uses any and all information provided to make a logical decision as to the correct answer for an RPM problem. This project is presented in this course as a way to expose the student to real world problems in the field of artificial intelligence, specifically, the cognitive approach. That is, how can artificial intelligence master the same types of thought processes that occur in human reasoning.

The Design

I chose to design my agent in Python (v2). For Project 3 I wrote additional tests (from those of Project2) using simple heuristics to determine the problem's answer without relying on any verbal representations. My logic is quite similar to that of solving RPMs using fractal representations. I had written a few tests like this for Project 2, but I needed more for project 3 to get better results. The logic designed for the project performed very well with Project 2 so I stayed with this idea. The more information I gathered about the image relationships the more I could eliminate answer choices from the answer set provided.

Upon instantiating the ***Agent*** class, and entering it's *solve()* function, the first thing my agent does is to create an instance of the ***AgentAnswer*** class. The ***AgentAnswer*** class, while not new for Project 3, creates a list of all the problem answer choices, which I will refer to as an *answer pool*. I then pass this pool of answers on to my individual solve methods. The intention is to examine each problem and, one by one remove potential candidates from the answer pool as they are determined they could not possibly be answers to our questions. Finally, when we are left with only one image in our answer pool, it is chosen and returned back from the agent.

So to begin, with the ***AgentAnswer*** class instantiated, I passed the problem and the answer pool over to a class that I developed for Project 1, ***AgentBasic*** (it's main procedure is called *solve()*). This class, in both Project 1 and Project 2, performs simple tests on the problem set. For Project 3 I rely solely on this class which contains all my tests. In order to get better results with visual processing only, I needed to add more tests to identify relationships without dependence on verbal data. These simple tests were run one after the other, reducing the answer pool bit by bit, to just one remaining selection.

So, for each successive test within ***AgentBasic.****solve()****,*** I would apply a simple test and based on the results of this test the agent would conclude that some answers in the pool could not possibly be accurate because they do not have a certain characteristic, as determined by the test. With this information, the ***AgentAnswer*** class was used to search through itself, that is, it's pool of answers and to remove from the pool those answers that did not apply.

Finally when only one remaining answer was left in our pool this selection was returned back to the Agent.solve(). If there was more than one answer to chose from, I skipped the problem. My

goal was to not rely on any one method for discovery, but instead to spread out the reasoning using as many tests as I could.

The Tests

Below is a description of each test used to process each problem set. Whenever possible, every problem was run through these tests, each yielding its own unique results.

Equality Test 1

Looks at images A - H looking for identical images, such that A==B==C and D==E==F and G==H. In this situation the answer is determined to be equal to G & H. The answer pool is not necessary, the answer list is searched for the matching image and returned.

Equality Test 2

The image comparisons are the same as in equality test 1, however images may appear equal visually but in fact they are not (off by a few pixels, etc). In this situation I examine the images more deeply with the function images_are_equal(). This function looks at the difference in the image channels and compares the results to a predetermined threshold. If the difference is below the threshold it is determined to be equal.

Equality Test 2b

Looks at images diagonally, A&E, B&F, D&H - looking for identical images, such that A==E and B==F and D==H. In this situation the answer is determined to be equal to A & E. The answer pool is not necessary, the answer list is searched for the matching image and returned.

Blend Test

Determine if the images are just blended versions of each other. For example, the subroutine combines the pixels of A with the pixels of B and takes the resulting image and checks to see if it is equal to C. The same is done with D,E&F. If they pass this test then the answer must be equal to the blended version of G and H.

Subtraction Test

Determine if the images are just “subtracted” versions of each other. For example, the subroutine subtracts the pixels of B from the pixels of A and takes the resulting image and checks to see if it is equal to C. The same is done with D-E=F. If they pass this test then the answer must be equal to the subtracted version of G - H.

Blend and Subtract Test, XOR test

The subroutine performs the following steps to get the unique objects in each image.
First, create an image of the unique objects in A by subtracting B from it.
Next, create an image of the unique objects in B by subtracting A from it.
Then blend these two images together and looks to see if it equals C.
Do the same for D and E to see if the combined image is equal to F. If we pass the first two tests then perform a blend and subtract test for images G and H and look for the answer.

Transpose and Subtract Test

This is like the Subtract Test except before I subtract image B from A, I first transpose image B. And, due to the nature of the images, I had to remove the whitespace from the image.

Image Uniqueness Test

For each image in our problem set, if it is NOT repeated anywhere else in the problem set then it is NOT likely to ever be a correct answer. If it is found in our answer pool then it should be removed. I used a function called elim_image() to look at each image in the problem set, if it is not repeated then remove it from the answer pool if it is present:

```
for x in ('A','B','C','D','E','F','G','H'):
  # if none of the images match image x, then image x is likely not an answer
  if self.elim_image(hasVerbal, problem, x):
    agtAns.removeImage(problem.figures[x].visualFilename)
```

Crop Test

In this test I cropped a portion of each image and compared to the same cropped portion of the next image. If each image had the same cropped portion then the answer must also have the same cropped portion.

```
rc, croppedImg = self.crop_and_test(problem, 'A','B','C', 'bottom-third')
if rc == True:
  rc, croppedImg = self.crop_and_test(problem, 'D','E','F', 'bottom-third')
  if rc == True:
    rc, croppedImg = self.crop_and_test(problem, 'G','H','H', 'bottom-third')
    if rc == True:
      #answer must have same cropped portion, eliminate those that do not
      agtAns.removeCropped_Not(problem, croppedImg, 'bottom-third')
```

I performed the crop test 11 times in all, each test varying in either the cropped section of the image or the comparison of cropped images (for example, A == B vs. A == E). Below is a list of the sections identified to create the crop_box. The crop box format is (left, top, right, bottom):

```
if section == 'top':
  crop_box = (0, 0, w, h/3)
elif section == 'top-small':
  crop_box = (0, 0, w, h/4)
elif section == 'middle':
  crop_box = (0, h/3, w, h/3*2)
elif section == 'bottom':
  crop_box = (0,h/3*2, w, h)
elif section == 'inner':
  crop_box = (h/4,w/3,w-w/3,h-h/4)
elif section == 'r-2/3': #right 2/3 of the image
  crop_box = (w-(w/3*2)+25, 0, w-25, h)  # try to eliminate white space with the +/- 25 pixels
elif section == 'l-2/3': #left 2/3 of the image
  crop_box = (0, 0, w-w/3, h)
else: #section == 'bottom-third' default to bottom third, this seems to work well...
  crop_box = (0,h/3,w,h)
```

Number of Answer Left Test

At this stage I have performed a sufficient number of tests and hope to have only 1 answer left in our answer pool. If so, then this must be our answer:

```
if agtAns.get_num_answers() == 1:
   answer = agtAns.get_fig_num(0)
```

The Classes
Below is a summary of the classes which my agent uses:

Class Name	Source File	Description
Agent	Agent.py	The main agent class
AgentBasic	AgentBasic.py	The class that performs all the tests described in this document
AgentAnswer	AgentAnswer.py	loads all problem answers into a list and defines functions related to this list, such as show_prob_answer_list() or eliminate_answer()
Attribute	AgentAttribute.py	class for a single attribute, defining variables and functions for testing and processing an attribute
AgentImage	AgentImage.py	helper class used for performing common functions on input images
AgentSemanticNet	AgentSemanticNet.py	the class used to perform the semantic network logic. Not effective in Project 2 or Project 3.

Project Reflection

The following questions were presented as part of the project to enable the student to more closely reflect upon the design choices made and the outcomes of the project.

- How does your agent reason over the problems it receives? What is its overall problem-solving process? Did you take any risks in the design of your agent, and did those risks pay off?

In Project 3, I still used the process of elimination, performing a series of tests one after the other, hoping to reduce the set of answers down to just 1 possible solution - our answer.
I was very apprehensive at the start of the project, indeed the entire semester! I did not have any idea how I would manage to identify relationships in images without any background in computer vision whatsoever. The method I used is risky because even though they solve the problems during development, they may not necessarily solve the Test problems that we are graded on. I do feel that it paid off though, as I was getting 22 out of 24 problems correct. Each problem set had some reliable characteristics that I could identify, such as the top 3rd of the images showed relationships, or perhaps the center of the images revealed a relationship, etc. I developed tests for the possible combinations of these characteristics and the results were quite good.

- How does your agent actually select an answer to a given problem? What metrics, if any, does it use to evaluate potential answers? Does it select only the exact correct answer, or does it rate answers on a more continuous scale?

The agent performs test after test in hopes of reducing the number of allowable answers. The answer is selected when only 1 remaining possibility is left.

For Project 3 I had to discard what I thought was the most important metrics used in my tests for Project 2 - which was the number of objects that a figure has and the number of fills/non-fills that a figure has. With Project 3, this information was unavailable to me so I had to abandon reliance on these tests and examine the images in different ways. Another metric used periodically in the tests was a variable threshold value in image comparisons. But overall, the logic of eliminating answers from information collected on the relationships was a good design.

It is an exact correct answer. There is no rating, nor scaling to choose from. If the number of allowable answers is not reduced to 1 then the tests continue. If I reach the end of my problem solving and there is more than 1 answer to choose from then I skip it.

- What mistakes does your agent make? Why does it make these mistakes? Could these mistakes be resolved within your agent's current approach, or are they fundamental problems with the way your agent approaches these problems?

One thing I had to discovered was when looping through the possible answers, looking for a particular image with a given characteristic, it would take the first image it found that was under the threshold. This was wrong! It should process all images in the answer list first, and then select the image that had the lowest difference value. After discovering this flaw and correcting my agent was more accurate.

My agent does not have logic in place on what to do if more than one answer remains, however. I did not want to make a guess because we get penalized for guessing so I just skip the problem when this occurs. Luckily I only had to skip one problem.

I'm not sure how I could logically assign weights using my design. To me this doesn't make sense so I did not implement it. This could be a fundamental problem with my design, but since I had to rely solely on visual methods, I am now more confident that assigning weights and measures in this way would not be necessary. I thought given enough time and examples I could solve this limitation but I no longer think it is a limitation.

- What improvements could you make to your agent given unlimited time and resources? How would you implement those improvements? Would those improvements improve your agent's accuracy, efficiency, generality, or something else?

I did set out to find a way to identify the metrics I use in my pre-existing tests - # of objects and the # of fills, etc. I was going to accomplish this using Connected-Component Labeling and then, with that same data I was going to attempt to find the number of objects that were filled. I started to do this, but got distracted by what I thought were the more obvious tests that I could build. After a while, I found I had invested a lot of time and analysis in the tests I was building and I was able to get a lot of the problems answered correctly and so I realized that connected-component labeling was not necessary and I abandoned the idea. Besides, the way I was coding it was a lot of fun. It was like solving puzzle after puzzle, finally getting a problem correct after spending hours on it.

If I had unlimited time I would complete the connected component labeling and build a routine to count the number of filled objects. I would also like to try the Affine methods that I have heard mentioned many times in this class but that remains a mystery for now.

While reviewing my Project 2 comments, I stated that “Throwing out the reliance on the verbal representation would be a big step in designing an agent to think like a human. I am not sure I will be able to overcome this, because I don’t have much experience in manipulating and analyzing images graphically but I will try. I suspect I will still need to use some verbal representation if it is available to me.” -- haha. I did it after all without building any kind of verbal representations :D

I was convinced that if I were able to accomplish this then I would maintain my agents accuracy, yet falter on its efficiency as it would take longer to run. Well I was wrong there too because all 8 problems sets took only 61 seconds to run. I am very proud of that. Generally speaking, I believe my improvements made my agent closer to thinking like a human because it had to rely on visual representations, much like humans do 99% of the time.

- How well does your agent perform across multiple metrics? Accuracy is important, but what about efficiency? What about generality? Are there other metrics or scenarios under which you think your agent’s performance would improve or suffer?

I feel my agent performs efficiently, though not instantaneously. For Project 1 it took almost a minute to perform 24 problems. Project 2 took approximately 30 seconds to process 48 problems. Project 3 only takes 61 seconds to process all 8 data sets.

Project 1 (24 problems): Approximately 50 seconds
Project 2 (48 problems): Approximately 30 seconds
Project 3 (96 problems): 61 seconds

My accuracy for Project 3 is good. I have not made big strides with the challenge problems but the basic problem results were very good overall (listing all problem set results) :

Set	Correct	Incorrect	Skipped
Basic Problems B	12	0	0
Challenge Probs B	3	3	6
Basic Problems C	11	0	1
Challenge Probs C	2	7	3
Basic Problems D	11	1	0
Challenge Probs D	3	7	2
Basic Problems E	11	0	1
Challenge Probs E	5	5	2

Some other metrics I could mention are process metrics such as human effort used, time

spent on the project, conformity to the project requirements, etc. I used a lot of time designing tests.

- Which reasoning method did you choose? Are you relying on verbal representations or visual? If you're using visual input, is your agent processing it into verbal representations for subsequent reasoning, or is it reasoning over the images themselves?

All of my Project 3 tests relied 100% on visual representations. Luckily, for project 2 I did not rely on all of the characteristics of the verbal representation. For example, I had no need for the following attributes: size, width, length, angle, inside, left-of, above, and overlaps. The only attributes I needed were the shape and fill. I didn't think about it at the time, of course, but this gave me a good starting place for continuing to build sensible tests for Project 3.

- Finally, what does the design and performance of your agent tell us about human cognition? Does your agent solve these problems like a human does? How is it similar, and how is it different? Has your agent's performance given you any insights into the way people solve these problems?

For Project 1, I was feeling rather dubious about my process and that it never even came close to thinking like a human. To me it was just another coding problem that I had to account for all the scenarios I could. I was feeling unimpressed with the cognitive approach to AI if this was all it was. By Project 2 I was beginning to feel like I managed to move away from this mindset, if just a little bit. I felt my results approached the realm of AI, if only within a fraction. By Project 3 I had realized that my agent was indeed performing like a human would, seeking out patterns visually and that's all. There was no "interference" from other forms of data gathering. It was pure visual.

By performing test after test, eliminating answers, I felt at ease with the coding process. Breaking it up into smaller chunks is, again, what humans do naturally and I felt like this is definitely the way a human would look at it - one by one deciding what should not belong.

I wouldn't admit it before now but I had a lot of fun coding each problem, solving little by little. I only wish I had not stressed out so much over the course of the semester but ever since I learned that project 3 was going to be challenging in this way I was quite apprehensive about it. I wished that went a little differently for me but I am glad the project is done. I am curious how I would score on the tests were I to take it, but for now I think I will settle for a KBAI T-shirt!