Traci Fairchild
CS7637 KBAI
Project 1
Project Reflection
September 26, 2015

Project Reflection for Solving 2x2 Raven's Progressive Matrices using Verbal and/or 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 image A and image B, looking for similarities or differences. And then when presented with another set of images, image C in our example, the test taker is expected to choose the image (images 1 through 6) that best completes the analogy of the first set. For example, a given RPM test could look like figure 1 below:

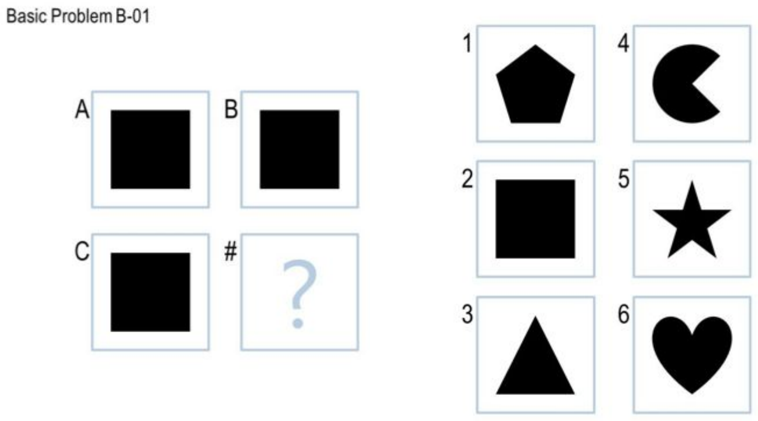

Figure 1

The Project
Project 1 is the first phase of writing an agent that can pass a human intelligence test called the Raven's Progressive Matrices Intelligence test. 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 to the student in this course as a way to expose us 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
Project 1 has provided the basic code necessary to run the program. I chose to design my agent in Python (v2). Here, I will summarize the basic logic used to discover an accurate answer.

Upon instantiating the ***Agent*** class, and entering it's solve() function, I entered the first stage of analysis. I tried to solve the problems using simple heuristics. For example, if figure A is equal to figure B then the answer must be the image that is equal to figure C. It is scenarios like this that I explored first, before all other processing when encountering a new RPM. The class used to process and examine the problems in this way is called ***AgentBasic a***nd it's main procedure is called *solve()*. So, for each successive test within ***AgentBasic.****solve()****,*** I would apply a simple test, search for an answer if conditions were met and if no answer was found I would return from the test, only to perform another. If an answer was found, it would return all the way back to the Agent.solve() where, again, the answer would be inspected before moving on to the next stage of processing.

The second stage of processing was to design a class which used Semantic Network methodology based on the problem information given. This class is called ***AgentSemanticNet*** and accessed via a *solve()* function. The test in this class only performs semantic network problem solving. There are no other extraneous tests being performed at this stage.

My goal was to not rely on any one method for discovering answers, but instead to spread out the reasoning using as many tests as I could. Indeed, the first stage has more testing done then the second and this seems to have paid off since most of my answers were solved in this way. It is also important to note that some of the tests in the first phase and the entire logic of the second phase rely on the verbal representation.

There are also a number of other supporting classes that I developed which my agent uses. I will briefly list them here:

Class Name	Source File	Description
AgentAnswer	AgentAnswer.py	loads all problem answers into a list and defines functions related to this list
Attribute	Attribute.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

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?

As I mentioned above, the agent first tries to solve the problem with the simplest heuristics it can use. The goal here was to knock out the answers quickly without adding too much to the processing time of the overall project run. The problem solving process was to test a scenario, and then if unsuccessful, move on to another test. In the end, if no answer was found then the problem was simply skipped. If an answer was not found, I could have just randomly chose an answer in hopes that it would be accurate. Although a human would do this often, I chose not to do this because it is illogical, the chances of getting it wrong are high and I would be penalized if it was a wrong answer (as opposed to a neutral effect if it was simply skipped).

- 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 search of an answer. The only metric it uses is in the basic tests, and that is the definition of a threshold. This threshold is used for some of the tests which compare images after they have been manipulated in a certain way.
Besides this, the basic tests in the agent selects answers based on exact correctness, such as the hash values or direct image matching.

In the semantic network logic, the tests also use exact correctness, except for the handling of the attribute values ‘left-of’, ‘inside’ and ‘above’. I did not use these values when comparing attributes because I could not figure out a good design in the time that I had. I had intended on mapping the alphabetic figure identifies that these attributes us to corresponding numeric identifiers but my logic was not supporting it at the time and so I decided to abandon it and exclude those attributes from comparison.

- 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?

I think the mistakes my agent makes can be avoided if given enough time. First I feel I can give more attention to the threshold value. It must be able to fluctuate according to some image metric that I have not discovered yet. Luckily, my agent has been relatively successful with the threshold value that I have set.

Another mistake my agent makes is that the Attribute class is not entirely dictionary based. Some of it is, and some of it is list based. This is causing some confusing with my processing and a lot of time was wasted trying to get some simple comparisons to work.

Another mistake my agent is bound to get caught on is the fact that in my semantic network processing I simply compare figureA-1 to figureB-1, figureA-2 to figureB-2, etc. This is definitely not going to work itself out, and it will only cause more of a problem as the projects increase in complexity.

- 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?

As far as improvements go, I need to find a way to process the images without the use of the verbal representation. While I managed to land on some ideas that work, I still have a long way to go with this. Based on piazza readings I know performing edge-detection is probably something I should do, but I frankly do not know how to approach this problem yet. I find working with the verbal representations frustrating and it feels like a cheat. So throwing out the reliance on the verbal representation would be a big step in designing and agent to think like a human, but it will surely add overhead to it's efficiency and possibly its accuracy. I am not sure I will be able to overcome this, but I will try.

- 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. It takes less than a minute to process 24 problems. The more I rely on the verbal representation, the longer it takes. Accuracy for project 1 is fair. I have not made any big strides with the challenge problems but the basic problems were pretty accurate at 11/0/1 correct/wrong/skipped (the challenge problems are at 3/2/7).

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 researching and designing, but most of all my time was spent on detecting and fixing defects. I think if my project were measured with this metric it would do well.

Another metric which could be measured is the design and organization of the project code. I feel like it doesn't have a real strong design and in places it feels haphazard or messy. If my project were measured against this metric it would probably not do too well.

- 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?

I've mostly covered this above, but for the sake of summary I will say that my reasoning methods first tried to attempt an answer without the use of the verbal representation and then falling back on it for lack of a better heuristic. My agent does not process visual input into verbal representations although this is an idea I have not considered and could be useful with future projects.

- 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?

The design and performance of my agent tells me that humans are the most powerful computer of all. My agent can only do what I have first carefully considered it should do, and then carefully considered how to make it do it. This is a difficult task and in the end I do not think my agent solves these problems at all like a human does. The outcome, if correct, was not attained using the same methods. It is pretty hard for me to see computer processing and human cognition on the same level. It is different because humans process things intuitively with information they do not know they have. Computers cannot access information they do not know they have, if they even have it in the first place and so it can only do whatever scenarios it has been prepared for. My final comment with this project is that the human mind is incredibly vast and to think that computers can compete with the mind is a concept that is hard for my mind to understand - see what I did there? :)