Jacob Kilver
Knowledge Based AI
Project 1 Reflection
12 June 2016

Introduction

This project set out to solve 2x2 Raven's Progressive Matrix (RPM) problems. A production system was used as a tool to solve these matrices. While verbal descriptions were provided, this project attempted to solve the problems visually. Although this method proved trivial for solving some RPMs, it proved quite difficult for others. Some of this difficulty was attributed to the deficiencies of the solution method, but much of the challenge was due to the complexity in designing a near human-level cognitive system.

Theory of Operation

A production system architecture blended with generate and test was used as the basis for project. While the production system in class contained several types of knowledge, the procedural knowledge subsystem was the focus here. Based on different "percepts" of the world (provided through the RPM data structure) various "actions" were taken (i.e., answers were generated) and then tested against the available answers using a pixel-by-pixel comparison.

The actions in this case were image transformations. A few operations were trivial: no transformation, rotation 90° clockwise, rotation 90° counter-clockwise, reflection across the vertical axis, and reflection across the horizontal axis. Image transformations beyond this became less trivial mostly due to the complexity of some of the transforms. If none of the simple transforms above resulted in a suitable match, the absolute differences between pairs of images (A and B, A and C) were found. These differences were then applied to the remaining image (e.g., C + (A-B), B + (A-C)) and then compared to the possible answers. The assumption here was that the most important aspect of the relationship between pairs of images dealt with the differences between them.

Implementation

Python was the language of choice for this project. Since visual solutions were generated here, the Python package `pillow` (http://pillow.readthedocs.io/en/3.2.x/index.html) was used to manipulate images. This package contained built in methods for rotation and reflection which were used for the simple transformations discussed above. For transformations beyond these, combinations of operators were used. For instance, image difference was found using `ImageChops.difference` and images were added using `ImageChops.add`.

Implementing the simple transformations was rather straightforward: manipulate image A and see if this manipulated image matches B, and similarly for A to C. Matches were determined by finding the percentage of pixels that differed between two images. If the percentage of similarity was above a certain threshold than a match was found. If multiple answers were found, then the one with the highest percentage of similarity was chosen and the transformation corresponding to that answer was saved for later use. If multiple transformations resulted in the same maximum confidence value, the order of precedence in favoring a transformation was as follows: no transformation, 90° counter-clockwise rotation, 90° clockwise rotation, vertical axis reflection, horizontal axis reflection.

The one problematic aspect here was in applying the selected transforms. Since there were two transformations (one from A to B and one from A to C), they could be combined in several different ways. For this project only single transformations were used, i.e., only A to B applied to C or only A to C applied to B. From here it was expected that the answer with the greatest confidence (highest percentage of matching pixels) would result in the best answer. However this was not the case due to slight imperfections between the transformed images and the answers.

As an example, see Figure 1. In this case the transformation between A and B was a reflection along the vertical axis. A human can tell that the transformation from A to C is a reflection along the horizontal axis. Unfortunately getting the agent to distinguish between the filled and unfilled shape was not possible in this project, so the A-C transform resulted in a "no match." However, due to the implementation details the guess that was returned was simply image A. When the guesses were compared to the answers, image A exactly matched answer 1. The second guess very closely matched answer 6 (the correct answer), but not 100%. To avoid selecting answer 1, "fuzzy logic" was used to equate these two answers as both having the same confidence. The guess of image A could then be eliminated because the transformation associated with that guess was not valid.

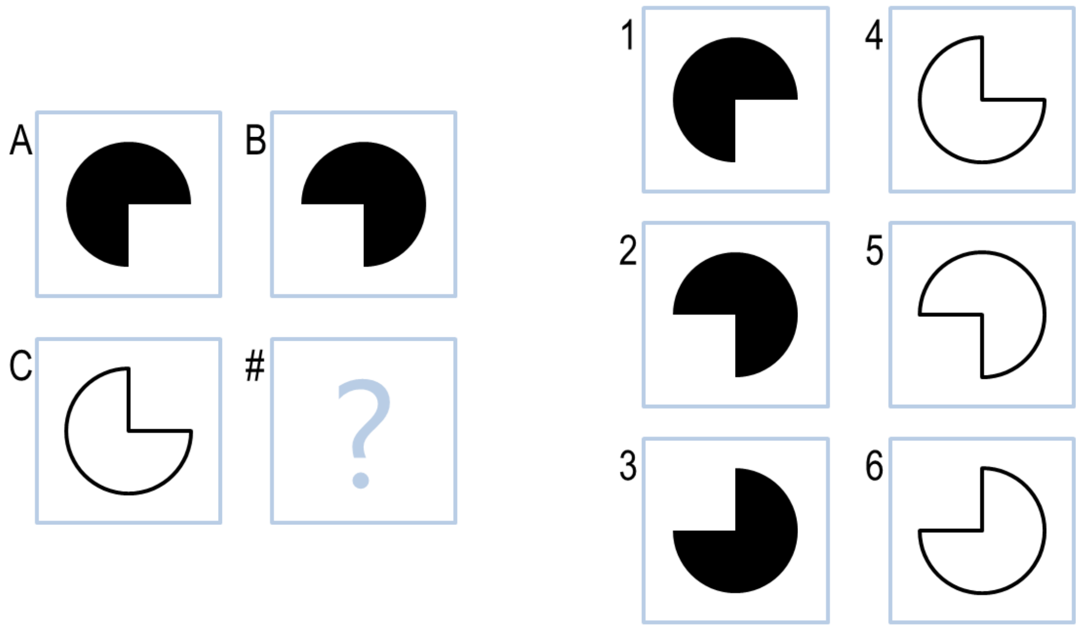

Figure 1: Raven's Progressive Matrix where "fuzzy logic" was used

The more difficult task was finding answers when simple transformations resulted in "no matches." Take Figure 2 below. For a human this is a trivial task – the answer is clearly 1, a hexagon without the black diamond in the center. For the agent, no simple transformation existed between any of these images. Since no match could be found, the absolute differences between pairs of images were compared. (Note that black had a value of 0 and white a value of 255. Thus differences were denoted by white pixels.) For A to B, this resulted in a white diamond in the center of the image. This difference was then applied in two ways to image C. The first was to add the difference to C, the second was to subtract the difference from C. Adding the diamond resulted in near match to answer 1. Fuzzy logic again had to be used here because the match was not exact – there was a border around the diamond area.

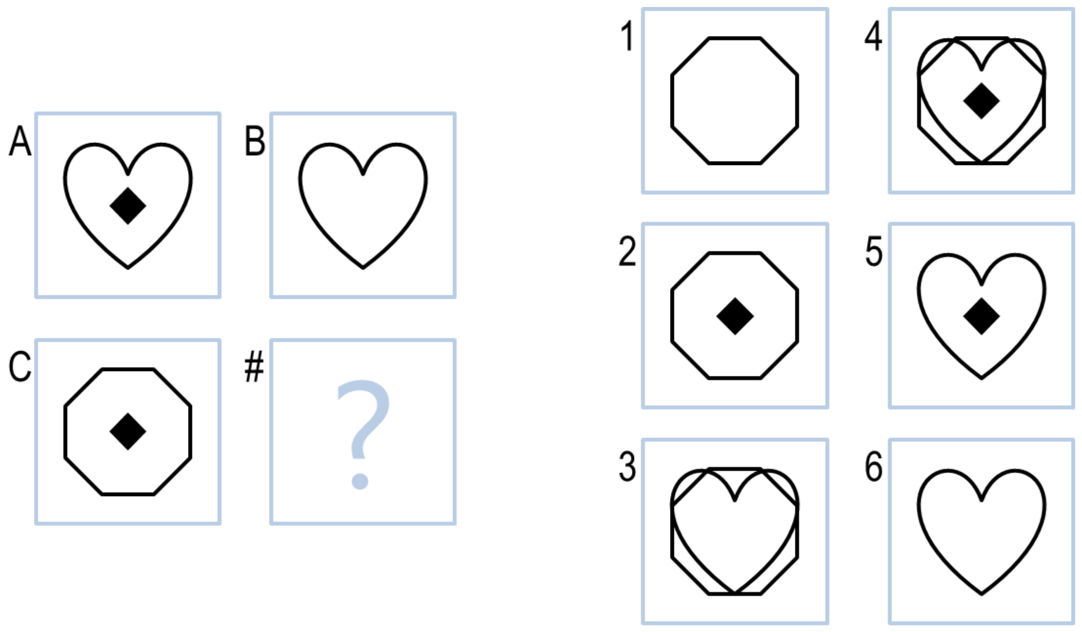

Figure 2: Raven's Progressive Matrix where simple transformations result in no useful information

As could be imagined, making special cases for all these different RPMs was not only time consuming but also increased in complexity dramatically as the number of cases grew. Determinations had to be made between answers that had the same (or about the same) confidence. Encapsulating what exactly the transformation between one image and another became difficult as the difficulty increased. Abstracting enough to handle new cases while being concrete enough to answer the known problems was no simple task. In fact, large portions of the in-sample problems were simply ignored in order to focus on solving a few problems correctly. This underscores one of the inherent difficulties in creating an agent with human level performance. Humans are able to determine relationships between vastly different inputs even in cases where similar cases have not been previously encountered. Also, we are able to easily distinguish the caliber of similarity without affixing a specific value and can thus equate things that would otherwise be considered dissimilar. Both these tasks are difficult to bestow on a computer because they depend so much on nuance and intuition.

Results and Discussion

The implementation was tested several different ways. The first test was a set of in-sample RPMs provided by the instructor. These results can be seen in the first two bars in Figure 3. This shows that heavy emphasis was placed on getting the Basic Problems correct while little attention was given to the Challenge Problem Set. Since the Basic Problems could be observed the solution methods were tailored to answer these correctly. The same could have been done for the Challenge Problem Set, but as noted in the previous section this would have been extremely time consuming and complex.

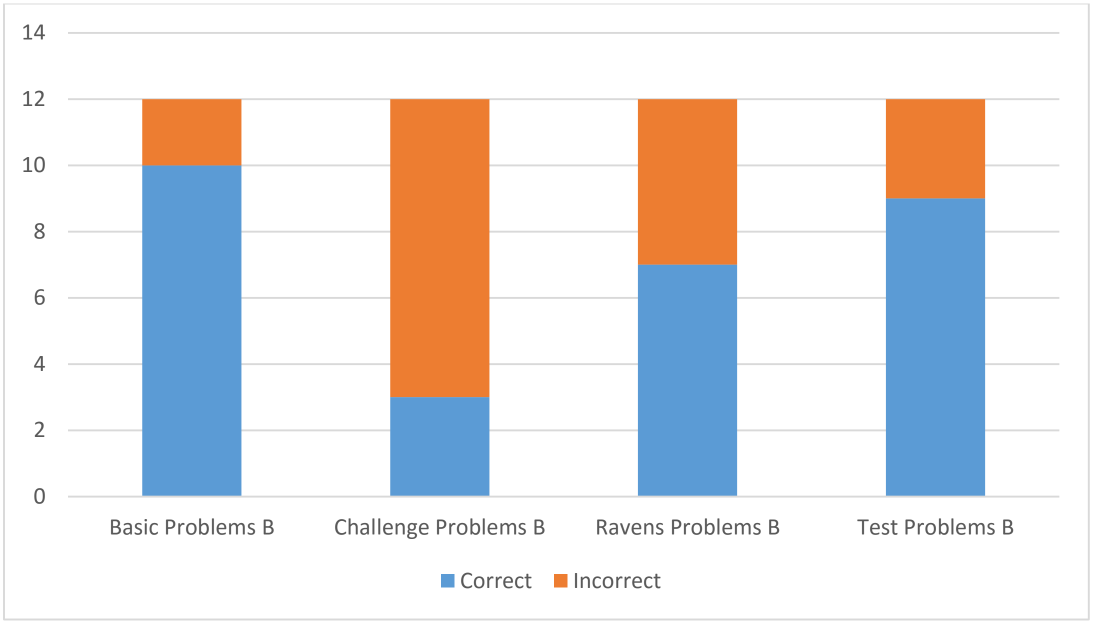

Figure 3: Correct vs. Incorrect Answers by Problem Set

The second test was using out-of-sample RPMs that were not provided by the instructor but that could be tested a limited number of times. These results can be seen in the right two columns in Figure 3. It was expected that these results would be similar to the in-sample results. One data set would be mostly correct while the other was mostly wrong. Fortunately this was not the case. The agent performed about as well on the Test Problems as on the Basic Problems. Additionally the performance on the Ravens Problems far exceeded that of the Challenge Problems.

Figure 4 summarizes the agent's performance across all the tests. Forty-eight problems were given to the agent, with 29 answered correctly and 19 answered incorrectly. Thus the agent had an accuracy rate of approximate 60%.

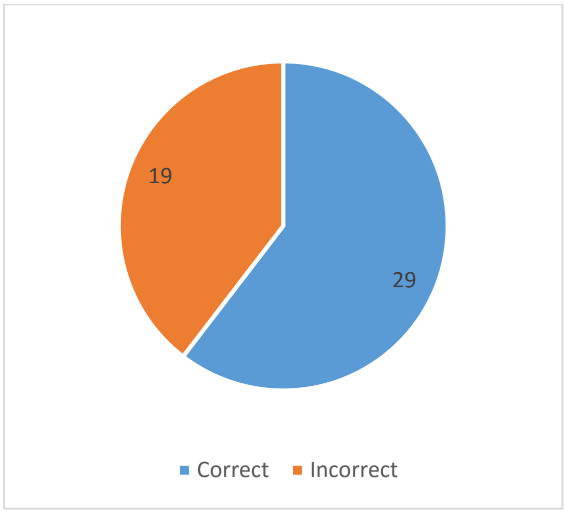

Figure 4: Correct vs. Incorrect across all problem sets

Conclusions and Future Work

Crafting a near human-level intelligence for something as simple as a series of multiple choice questions of pattern matching is difficult. This project explored techniques to create such an intelligence using production systems and generate and test. Decent performance was observed, but it was noted that to achieve better results would require a great deal more complexity. It was postulated that this is in part due to the difficulty in creating human level intelligence. However, the author also believes it to be in part due to the deficiency in the selected methods. Production systems and generate and test are useful methods, but they are very general. More tailored methods need to be used, and more research needs to be done exploring how similar problems have been approached by the larger AI community.