Project Three Visual Agent

Troy Wiipongwii
twiipongwii3@gatech.edu

Approach, Construction, Adaptation:

Initial Approach:

For project 3 I began again by going through the problems in D and E and looking for general patterns in problem solving. In D, I noticed that the vast majority of problems involved analogies with cyclic patterns, while in E the vast majority of the problems involved analogies with combinations or reductions of objects across rows. By default, I went back to generate and test as a problem solving method, since it performed reasonably well on the first 2 projects with verbal reasoning. My initial project required comparing images based on a few parameters to determine answers. These parameters included a few of the following:

1. If ImageA == ImageB:
    a. Then ImageC == ImageD
2. If ImageA == ImageC:
    a. Then ImageE == ImageI (This represents a cyclic pattern)
3. If ImageC == Combo(ImageA, ImageB):
    a. Then ImageI == Combo(ImageG, ImageH) (Combination pattern)

This logic works well if images are in fact =, however, I noticed quickly by playing with this simple logic in basic problem be that similar images were not always = and that in fact they may deviate slightly by pixel count. I then adjusted my agent and decided for focus on image arrays instead of images themselves.

Determining Similarity:
As a matter of determining the similarity of Objects I used 2 methods. Both were useful, but consistency was elusive at times. These methods were comparing black pixel count across figures as well as using the MSE of Image Arrays.

$$MSE = \frac{1}{m\,n}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2$$

Equation 1: Mean Squared Error

MSE was far superior in Basic Problems B, however failed to identify equality with transformations at times. Using the MSE == 0 or PixelCount == 0 was helpful in the 3x3 problems, although it occasionally gave inconsistent results.

Updated Approach:

Under the new problem solving method, the idea was that I could compare image arrays by use of pixel counts and mean squared errors. This idea allowed me to put ranges of acceptability on the differences of images that are similar to each other. With images that are supposed to be equal, the idea was to take the minimum pixel difference or the minimum MSE as the image of equality. This looked like the following:

1. If pixeldifference(ImageA, ImageB) <= 1:
    a. Then pixeldifference(ImageC,ImageD) <=: (equal images)
2. If pixeldifference(ImageC, Combo(ImageA,ImageB) <=1:
    a. Then pixeldifference(Combo(ImageG, ImageH), ImageI) <=1

In addition to the change from image comparison to comparison around arrays, I found a way to center the images so as to reduce any differences among the image that may have distorted equality by means of slight position differences. This type of logic improved problem solving substantially. In addition to logic above, under iteration two, I also took into account the reality that some of the cyclic patterns in D were not a matter of 3 distinct images in row 1 having an identical counterpart in row 2 positioned in a different column. These cyclic patterns involved a number of interesting transformations where some object was equal across the entire row, but another object in these images were transforming, while that same object was equal down columns but another object would be present among the next row. These combos required not just minimum differences among images, they required a transformation from one image to the next to be the nearest neighbor to other images and their transformations. For example:

1. If transdiff(ImageA,ImageB) == transdiff(ImageD,ImageE):
    a. Then transdiff(imageE,F) == transdiff(H, I)

This method also worked well for some cases, but performed incorrectly in several cases and even overrode answers that were previously answered correctly by having a transformation difference smaller than other methods, although those methods were correct.

Pivoting to a new approach for final agent:

This led me to realize several things which include some of the following:

1. Some images have equal black pixel count even if the images are not equal creating false equality among images.
2. Some transformations should not be designed to look for the minimum value difference between two images, but should be designed to look for the minimum value difference between two transformations.
3. Some transformations should not be set on the minimum (if the minimum is zero) the image chosen as the answer may be identical to another image in another row where it should not be. In fact the 2nd minimum should be selected.

Ultimately the second iteration of my agent correctly answered 7/12 basic D problems correctly consistently and 6/12 of the basic E problems correctly. Unsatisfied with this agent, I chose to adjust my problem solving method once more.

Final Agent:

This time I have chosen to use a case-based reasoning. My model has proven to be successful in some instances where rules and transformations were well defined in the code and performed poorly where the agent did not clearly know where to reduce its range of potential answers. What I mean by this is that when the agent was confronted with cyclic pattern, versus combo, versus some other transformation, it sometimes choose the “wrong” one. We saw this issue in many lessons in the course, it stuck out most in production systems, when we used chunking to break impasse. It also was showed in case-base, explanation-based, scripts, analogical and learning by mistakes. A few of these methods, I feel wouldn’t work for our current model, because we aren’t using correct answers as a feedback for adjustment in our model.

My belief here was that proper storage, retrieval and rules for adaptation would drastically improve the entire model. This means that in situations where I perceived the agent as choosing the wrong problem solving calculation, the agent will be designed with criteria to determine the best transformation for that problem even if the agent can identify multiple.

My final model retrieves cases through a discrimination tree. My entire retrieval model moves through one tree and when reaching the final leaf and still has no answer it moves onto to another tree. All cases are stored in either dictionary and lists and are indexed. The rules by which the model operate are similar to the generate and test method, with a few adjustments.

As a rule of thumb, the first rules tested are for equality of figures among the row, followed by equality down columns and equality diagonally. From there the testing of transformations across rows, down columns, diagonally are performed where equality does not exist.

Determining Similarity:

I continued using MSE and PixelCount in my updated model, however I also added a structural similarity model. To create a more robust comparison method. The following model was used:

$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

Equation 2: Structural Similarity Index Model

This model was retrieved from Image Quality Assessment: From Error Visibility to Structural Similarity. This model is available in Scikit-learn, however I hard coded the algorithm myself using just numpy and pillow. This was not helpful in increasing the accuracy of the model however.

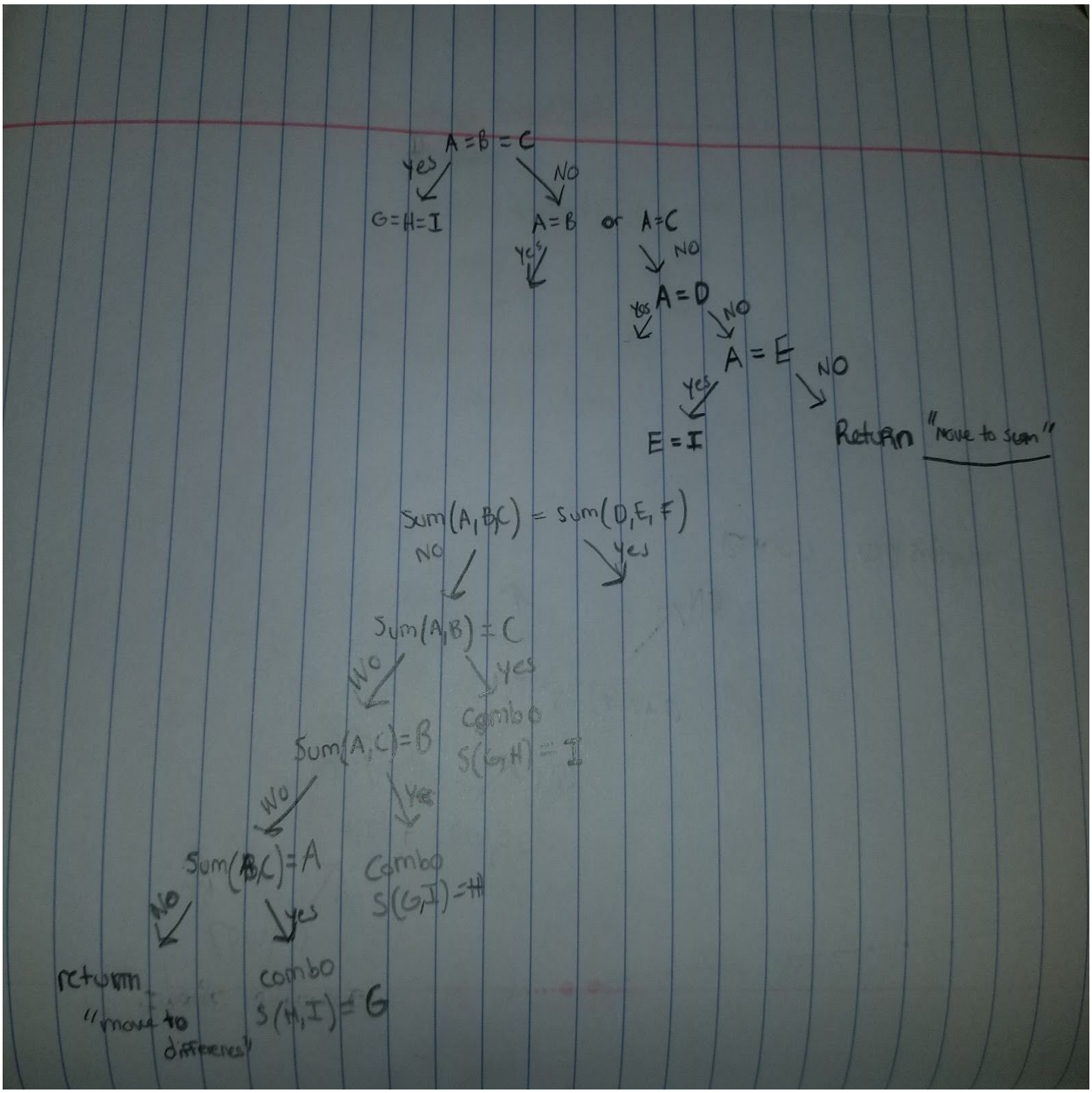

Figure 1: How My agent searches for a list of potential answers to a related analogy

```
ansListH = [self.pixels_comparison(prob_array['H'],ans_array[str(x)]) for x in range(1,9)]
ansListF = [self.pixels_comparison(prob_array['F'],ans_array[str(x)]) for x in range(1,9)]
ansListG = [self.pixels_comparison(prob_array['G'],ans_array[str(x)]) for x in range(1,9)]
ansListE = [self.pixels_comparison(prob_array['E'],ans_array[str(x)]) for x in range(1,9)]
```
Figure 2: how the agent stores lists of answers for later comparison

My final agent only improved slightly from my previous agents performing 8/12 on Basic D problems and 7/12 on Basic E problems. My agent also solves 10/12 Basic B problems correctly and only 4/12 C problems correctly.

Although the model is technically passing the required 7/12 for D and E the time trade off is very poor. Using both profile, time and manual time calculated the improvement from my generate and test method to the case-based reasoning method in terms of accuracy drastically reduced the efficiency. When time was calculated from the start to finish of the code, the Agent in the Case-Based reasoning model was approximately 45 seconds slower. I do believe however this is in large part due to the new comparison method of figures. I initially used just a pixel_count algorithm, but now using a method that takes into account pixel count, color and position.

In terms of performance, the model does really well with combinations, image equalities and recognizing cyclic patterns. In the B basic problems, it did a great job recognizing transformations such as rotation and combination. The agent's failure in basic problem C was due in large part to the fact that, I had isolated some problem solving techniques in 2x2 and 3x3 problems that would have been helpful if they had been combined.

The agent also does not perform well on analogies that have small deviations from one figure to the next or with analogies where the picture object count changes, but with different images.

One Improvement that I should have finished by the end of the day is being able to stop the agent from choosing a figure that has already been shown in the analogy, but the analogy transformation itself is not one of equality. For example in a few problems, when we are looking for an answer that has more than two possible answers because of structural similarity, the agent often picks the previously seen figure. This happened more often in the D problems, like D-10. This also happened with combinations problems in E. Where the difference between two images returned one of the two images, because one of the images was not a full picture and fell behind the dark pixels.

Although, I ran out of time it was my home to find away to detect edges, create examples of shapes of all sizes in memory and use that as a base for problem solving. There were many figures that had a base shape with a few extra features, but I believe an agent with a memory of examples would have done a better job with some of the problems mentioned above. Especially the ones where size changed, but shape did not.

The one problem in particular that I believe would have benefited the most is D- 12 where there is a cycle of 3/4/5 objects in the figure, but of different shapes. Also E-9 and E- 12 would have benefited as well.

Meta-Reasoning:

In these projects throughout the class, we as the students have been operating as the meta-reasoners of our agent, since our agent is not using feedback from the correct answers internally, we are doing so for the agent. We are also being meta-reasoners on our own reasoning since we are essentially asking ourselves, how would I solve a problem if I had limited understanding of many components of the problem. We discussed this in goal-based autonomy where we explained how an agent is essentially looping into thinking about what it thinks about instead of having a n-layered meta. As meta-reasoners of our agent, we had to ask ourselves, how do we determine to images are = if we visually see two images and heuristically determine they are equal even if the pixels are off by 2. We were confronted with determining still if we will problem solve propositional or analogical representation. Although our final project was to use images, our encoding itself was a choice on whether we would extract pixel data and represent it verbally or visually in reasoning. In my case at least, there was a combination of both.

So where does my agent I think similarly? And differently? For me I solved 100% of the ravens problems correctly in a short amount of time. I also was able to explain the patterns. How did I know? Maybe an undergraduate degree in mathematics and masters in statistics has given me a strong logical brain? So if I can answer them correctly and reason why I did, why couldn't I train my agent to do so initially with perfection? My agent has different tools available for reasoning. Also I have a knowledge gap myself. That knowledge gap is I don't know exactly what tools it has that a different than mine that cause a difference in problem solving. So I had to create a learning goal to figure out what that meant. This required me exploring pillow in much more depth as well as numpy. So my knowledge gap resulted in my agents mistake in learning and reasoning. For instance, I only need to look at AtoC in many situations to know I. my agent may miss something by just looking at AtoC. so I must provide even more information to my agent than I would use consciously to determine I. This is likely the result of the fact that I have already built heuristics to know that AtoC mimics DtoF where the agent doesn't know that unless i specifically tell them. Also there are some cases where that relationship fails. As a result my agent does have complete

information so the agent may be I based on AtoC when it in fact also needs DtoF. again unless I encode that, the agent won't know.

I myself also learned by correcting mistakes. The process of adjusting my code for better performance allowed me to learn where I was making shortcuts in code that lead to incorrect answers, where my command of certain data structures was not where it should be etc. This itself overlapped with diagnosis as well. My agent for instance was mistaking a few problems for combos when they should have been cycles. This is the data. The hypothesis using abduction is that I have not properly designed code for cycles or combos resulting in improper classification. The treatment then is to tweak on or the other try again until problem is solved. It may be a stretch to say that diagnosis or learning by correcting mistakes is taking place at the answer selection method for my agent, but I will attempt. Once a criteria has been set for what a transformation will look like for the final answer based on previous images, when we put the answer process through a loop of trial and error essentially, the agent is determining that an answer is wrong and moves on until it finds its golden goose.

My agent certainly makes use of and is indicative of cognitive reasoning discussed in KBAI. Dictionaries in python were used as frames (representation) images as arrays potential answers as lists there is certainly a representation component. In terms of reasoning, my agent is specifically designed around a set of rules to reason through to determine an answers, these rules are also built in with constraints. I think the agent learned, but I am not sure to which extent I would say it continued learning. Certainly it learned when it created transformations and stored them to compare to others in the same problem, but each problem in this project was addressed independently. My agent did not learn from one and apply that to two. The rules were hard coded already. If my agent used training and testing data across problems, I would certainly be more comfortable saying that it was a learning agent. Of course this itself comes down to the magnitude and state space that we define learning.

References:

Zhou Wang et el, Image Quality Assessment: From Error Visibility to Structural Similarity, IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 13, NO. 4, APRIL 2004 retrieved from: http://www.cns.nyu.edu/pub/eero/wang03-reprint.pdf

The Python Profilers, Python Module, retrieved from: https://docs.python.org/3/library/profile.html

Ardit, How to measure the execution time of a Python script, Python How, 2017, Retrieved from: https://pythonhow.com/measure-execution-time-python-code/