A Recipe for Learning Machine Learning:

5 Supervised Learning Methods + 2 Datasets

CS 7641: Machine Learning

Spring 2016

*Instructors: Drs. Charles Isbell and Michael Littman*

*Submitted by:*

*Miriam Heller*

*February 7, 2016*

*23:55:00 EST*

GT ID # 903136890

Mheller8@gatech.edu

Table of Contents

1.0 Introduction 3

2.0 Designs and Implementations 4

Project 1: Losing Sight of the Big Picture While Sifting through Objects 4

Project 2: A Figure by Any Other Name 5

Project 3: Visual Mashup and Pray 7

3.0 Promising Results But No Way to Plug & Play Improvements 8

Learning by Correcting Mistakes or At Least Identifying How to Correct Them 8

4.0 Solving RPMs to Learn about AI and Human Cognition 10

5.0 Conclusions 11

6.0 References 12

7.0 Appendices 13

# 1.0 Introduction

This reflection integrates my experience and thinking on 1) solving the Ravens Progressive Matrices (RPMs) problems using Knowledge-based Artificial Intelligence (KBAI) concepts; 2) as a project-based mechanism for *learning* about solving KBAI; and 3) understanding, comparing and contrasting KBAI solutions to the RPMs versus my perception of human cognition methods.

While this reflection is supposed to focus on Project 3, it will focus on what I learned from all three projects from reading and listening more than what I was able to accomplish in the Projects as I was hardly successful in generating working code. This inability will nonetheless enrich my discussions on 2) the value of project-based learning and 3) cognition and RPM solutions.

To keep this reflection compact, a brief overview of RPMs is offered here. RPMs consist of five groups of visual, geometric analogy problems developed by Raven (1936) and Raven and Raven et al. (2003). Each group consists of a square matrix of cells with figures except one empty cell to be filled in. Depending on the size of the matrix, there a number of figures to choose to fill in the empty cell. The projects described here include 2x2 matrices with 6 multiple choices for the answer and 3x3 matrices with 8 multiple choices for the answer (see Figures 1 and 2). RPM’s are viewed as one of the best measures of general human intelligence, yet there is evidence that difference exist in how the test is approached. Research on how we process RPM information has evolved from looking only at verbal strategies and rules of logic (e.g., Carpenter et al., 1990) to wholly visuospatial reasoning (e.g., Kunda et al., 2010).

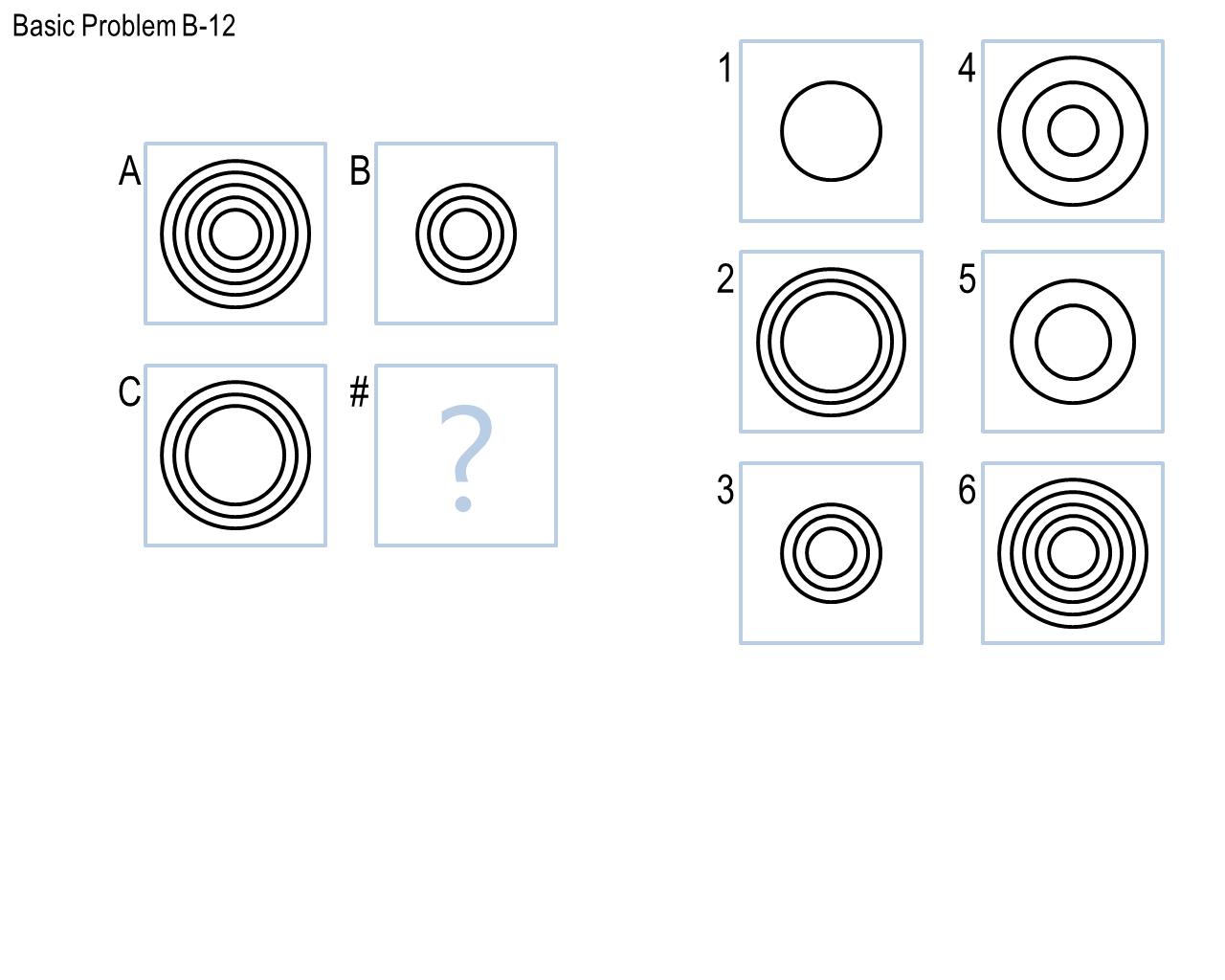


Figure 1. 2x2 RPM with 6 answer choices

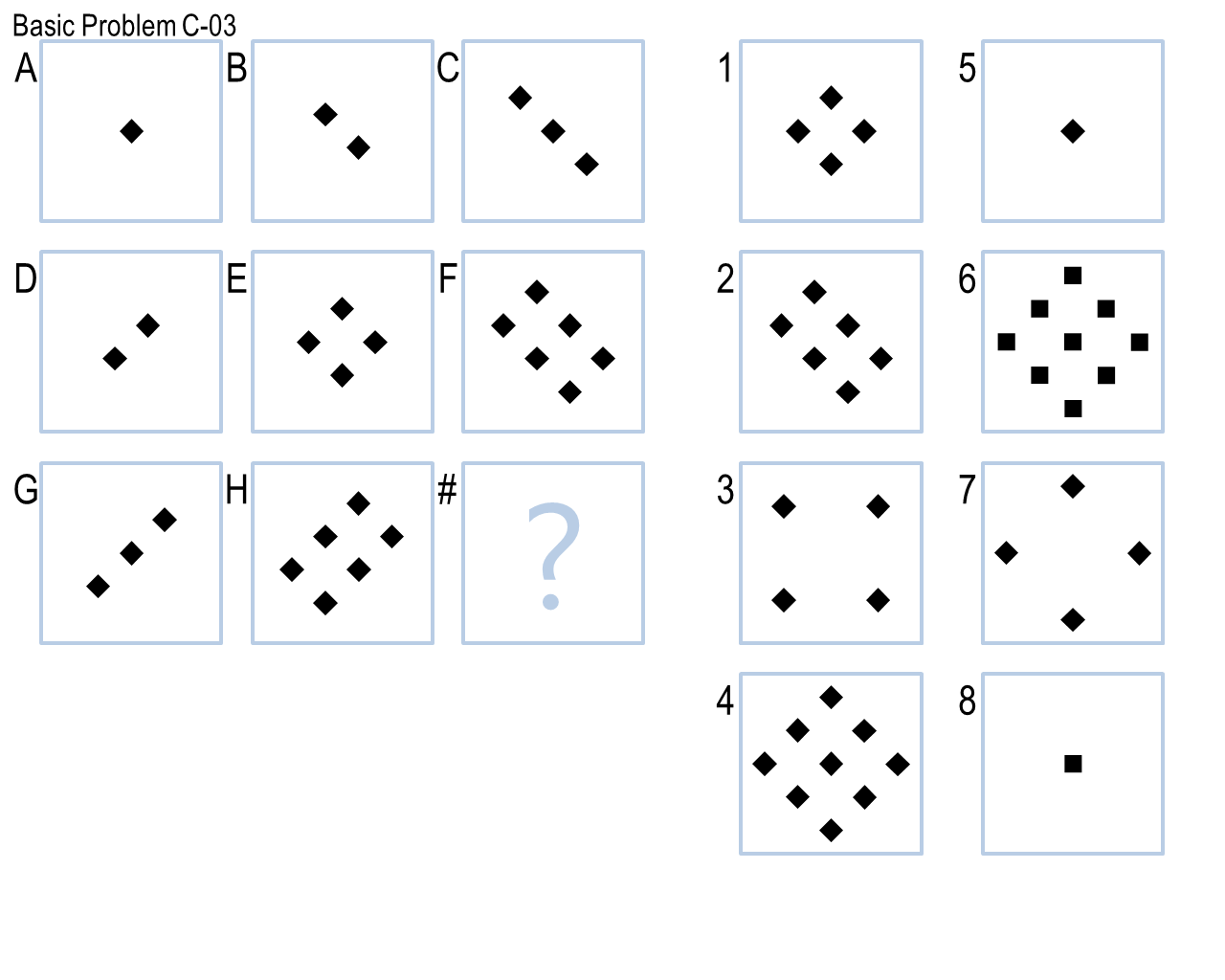


Figure 2. 3x3 RPM with 8 answer choices

Finding the missing entry for an RPM entails finding the pattern or transformation across a one row and one column (2x2) or n-1 rows and n columns in a (nxn) RPM that when applied to row or column m yields the missing entry. Multiple forms of reasoning, learning and memory appear define how we solve for the missing entry in RPM’s. RPMs can challenge our general intelligence in terms of how we process information. From a verbal perspective, our ability to reason verbally tests our capabilities of logical, i.e., induction, deduction and reduction, our ability to reason verbally and/or visually through analogy and correlation, ability to learn from previous examples either verbally or visually, and thus our ability to remember from similar cases. Additionally, either form of reasoning may be challenged by our ability remember the complexity of solving even one transformation.

Thus at a minimum what makes solving RPMs difficult is the need for a certain level of general intelligence, ability to reason logically, ability to reason visuospatially, ability to learn by example, ability to recall examples, and ability to remember and keep track of intermediate results, e.g., if one were trying transformations on an object by object basis and trying to remember the series of transformations to apply to another figures. Carpenter et al. (1990) also alludes to remembering the numerous transformations that may apply to a particular problem.

# 2.0 Designs and Implementations

KBAI instructors 5 modules of code in Python, the language I felt most proficient in: RavensProject.py, ProblemSet.py, RavensProblem.py, RavensFigure, RavensObject, Agent.py. Embedded within these modules were data structures that could be used directly to represent the problems verbally. They consisted of classes for RavensProblem objects (incl. name, prolemType, correctAnswer, has Visual and hasVerbal attributes; RavensFigure objects (incl. name and an object dictionary) and RavensObject objects (incl. name and an attribute dictionary. I coded in Python 2.7 using spyder. Of note was my lack of knowledge of version control. Additionally, I encountered many problems with spyder not refreshing its memory even when I’d close the console, quite out of the application, exit the terminal and start all over again.

## k-Nearest Neighbor : First Lesson in Learning

The initial task was focused solely on implementing the Intelligent AI Agent module, which calls a Solve method to solve RPM problems. Project 1 entailed two problem sets of 2x2 RPMs, each with 12 problems including visual descriptions (.png files): the Basic and Challenge problems. The 2x2 Challenge Problems have no verbal descriptions. My initial aim was to have running code to perform reasonably well on the Basic Problems, enough so that my Intelligent Agent might even get some of the Test Problems correct.

Having not coded an end-to-end problem in decades, finding the notion of dictionaries new, and taking the advice to start with one problem, I wrote code for Basic Problem 01. Since Figure A = B = C, (using e.g., figures[“A”].object[“a”].attributes = figures[“B”].object[“b”].attributes), this seemed trivial and foolishly I did not design the high-level program structure that would taken the step of matching B or C to one of the Figures. While it seemed like a trivial extension, this was likely the single worst decision I made, the reason why I drifted so far behind in the class and handicapped me throughout the rest of the course. Regardless of the structure, I also needed to have taken the step of writing code that would have scored the similarity of each answer with B or C to yield an answer. Then I should have written code to link to the output routines.

I felt that this *identity\_transform* I wrote was trivial, couldn’t solve identical figures with more than one object and didn’t provide me with enough insight into the data structure I would need to store at least the current best transform and score and the best transform and score. So I opted to move on to solve Basic Problem 02. At this point, I was **fixated on matching objects** **by their attributes**. I embarked on a solution that would enumerate all possible mappings of objects from one Figure A to Figure B (A->B) or Figure A to Figure C (A->C). These transforms would then be applied to Figure C or Figure B, respectively to generate (and verify) my estimate for the missing entry, say D for the 2x2. I would then estimate the similarity of D to each of the 6 answer choices according to some measure of similarity I had not yet devised.

I was pleased that I was able to get code running (see Appendix 6.1). I verified that all possible mappings were afforded. I noted that my coding did make explicit which objects would be deleted or created and again this oversight had to do with my lack of a scoring routine. Simultaneously I realized that Basic Problem B-12 had 64 mapping combination. With **this brute force generate and test strategy**, there would have been 64 \* n transformations (if I ever wrote them) multiplied by 6 tests against the solution for one problem. I had simultaneously conceptualized some of the transforms. Since notions like horizontal and vertical flip didn’t appear to be part of the verbal description I generated two rules to test identity of an object after a horizontal or vertical flip:

if initial\_angle < 270 :

horizontal\_flip\_angle = 270 - initial\_angle

else:

horizontal\_flip\_angle = 630 - initial\_angle

and

if initial\_angle <= 90:

vertical\_flip\_angle = 90 - initial\_angle

else:

vertical\_flip\_angle = 450- initial\_angle

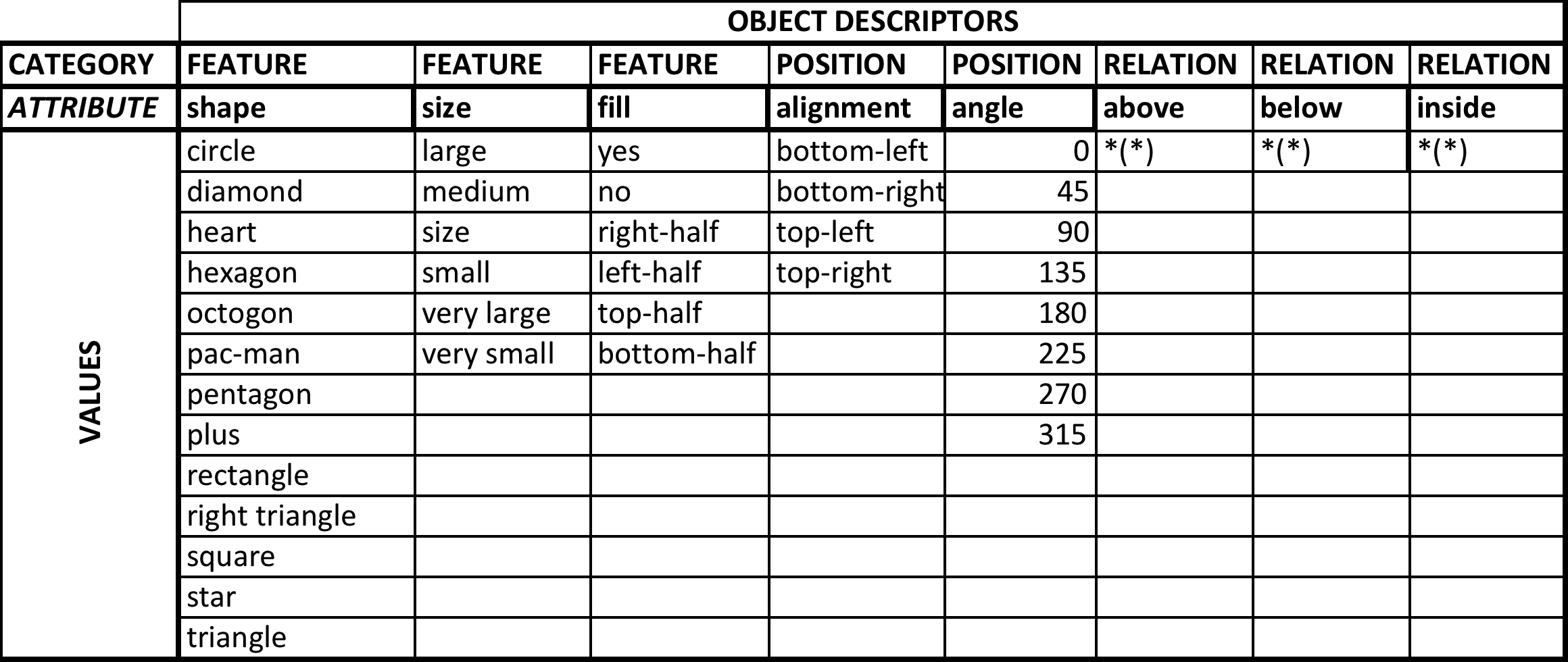
While this is definitely indicative of only rudimentary coding skills, it is worth mentioning that every new line of code I attempted to introduce resulted in anywhere from 5 minutes to a day of debugging. **This level of penalty made me highly risk averse in terms of testing anything.** I felt that working to make this inelegant exhaustive enumeration solution work would not be an efficient use of my time. I also recognized that n, the number of transformations, would grow very large as I moved on to solve the 3x3 problems. I moved on to another strategy.

## Project 2: A Figure by Any Other Name

I decided to **think about my solution techniques for solving RPM,** especially since all but 2 problems from Problem Sets B through E posed no difficulty to me in solving. It was obvious that I compared two figures to identify changes in shapes between the figures. Additionally, I would use Figure level patterns to deduce the missing entry, e.g., Basic Problem B-04, and many 3x3 problems where there was Figure level diagonal symmetry. I anticipated needing a way to capture the patterns or the transformations at the Figure level.

So I embarked on **a new figure level approach** whereby I identified the changes between one **figure’s attributes aggregated into a set** and another figure’s attributes aggregated into a set. That this is the logical way to identify change was corroborated first in the visual isolation method for **Learning by Correcting Mistakes.**  In this algorithm, the we isolate features that are exclusive to the positive example and features exclusive to the negative example. The transformation from the positive to the negative example is what we want to isolate in both learning by Correcting Mistakes and just how one Figure transforms to another. **Tversky’s similarity metric** (1977) captures both these transformations generally. I had already missed the Project 2 deadline by this point and had not yet implemented code that was solving the 2x2 end to end. A chunk of what I implemented is given in Appendix 6.2.

At this point, **I analyzed my attributes because it would be the changes in attributes at the Figure level that defined my verbal transformations.** Since attributes were now the focus, I generated Table 1. in which I distinguished between 1) attributes that defined the object in terms of its features; 2) attributes that defined the object in terms of its position; and 3) defined the objects in some way relative to another object. Although I abandoned this way of characterizing objects,I was reminded of the approach during **the Advanced Topics lecture when visuospatial reasoning was defined to have a “what” and “where” component.** I view the relational position as being somewhat different though, than absolute location, since it will always be relative.

Table 1. Characterization of Object Attributes

In the partial code I was able to get running, the shapes, sizes, and fill were parameterized even though they are enumerated here. Thus I was not restricted to predefined features. My sense was that I would be able to deal with the relational attributes recursively, testing to see if the object of the predicate was indeed an object in the Figure.

Using a new object class call CompareFigures (my first recent foray into defining an class), I was thrilled that the set functions enabled me to transcend the need to reason over objects, since theoretically attributes, if countable with some level of uniqueness, could be sufficient to define each Figure (frame). I was able to a figure and generate a list of tuples of the keys and attributes for all objects within that figure. This worked fine on some of the Basic Problems B. By virtue of the shape attribute, the deletion or addition of an object/shape could be determined. Unfortunately, this **deductive logic** became more tedious as multiple objects had the same shape. A set will not identify two separate shapes that are not distinguished. It will only show one tuple if there are two key attribute tuples (shape, circle). However, if they are different sizes, then the addition or disappearance of one can be deduced. This is tedious. For instance, in Basic Problem B-12, I would see the change in the “inside” attributes and be able to deduce that objects (shapes) were deleted by looking at the original key-attribute pairs. This really didn’t seem very efficient. I was successful in actually coding a second rule that detected reflections, but I hadn’t integrated it in to decision-based system that had a data structure to hold this information and assign a score to it based on Table 2 so that I could determine whether I needed to test another rule. As it were, Project 2 deadline came and went and I was advised to try my hand at the visual approach.

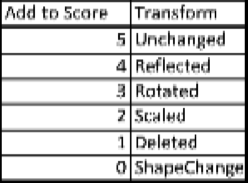


Table 2. Sample scoring system for detected transformations

## Project 3: Visual Mashup and Pray

My previous lack of success in coding the RPM solution and limited time remaining, demanded an immediate solution upon me – to do the simplest thing possible. I created some initialization calls in the Agent code: one invoking a *VisualSolver* class and a *solve* method. The *VisualSolver* loads the images and preprocesses them to a list of 184 x 184 black and white elements. While I had been trying to learn a bit about Pillow, I also hung out at several Office hours, read forum entry and came to appreciate some difficulties that other students had run into, namely anti-aliasing. Having only dealt with image processing in MatLab in a Coursera ML course, where everything just ran if you fed it 1-6 correct equations, this was all new to me.

I created a method to the class called *percent\_difference,* that stupidly took the sum of the bits of the image if they were different, e.g., (1,0) or (0,1). I created a single rule for the 2x2 problems. This was a logical transition from my verbal attempt since the visual solution at this point corresponded to the whole Figure. It solve for the missing entry by calculating the *sum over the comparison sets* (1 – (difference of a comparison set) – (singleton – answer choice). Since this was summed over the comparison sets, one could consider it a **voting mechanism**. For instance, a comparison set might have been A, B and the singleton would be C. There were two comparison sets because the relationships should also be derivable from the comparison set A,C and the singleton B. I normalized the score by the number of comparison sets to be a number between [0,1].

The code was simple to alter to accommodate a 3x3 and apply this logic as a generalized *fourcorner\_solver*, which would be easily extensible to 4x4, 5x5 or 6x6 RPMs.

# 3.0 Promising Results But No Way to Plug & Play Improvements

The simplistic solver yielded surprisingly good results when applied to the Problem Sets, (see Table 3). These results seemed fantastic compared to those I had generated up to this point — none. On average, this simple pixel differencing performed 223% better than random guessing with 6 and 8 answer choices for the 2x2 and 3x3 matrices.



Table 3. Results of FourCorner\_Solver Method in VisualSolver

## Learning by Correcting Mistakes or At Least Identifying How to Correct Them

The logical next step was to identify common causes among the incorrectly solved problems. One **cause of incorrect solutions is that the *fourcorner\_solver* alone is guaranteed optimal only for the identity transformation between two figures**. To understand this precisely, assume we’re operating on a 2x2 RPM, then given Raven’s Figures A, B and C, we seek , such that:

where

G =

F =

S = {RPM answer choices} and

(Note: that this notation could represent RPM verbal or visual representations, where verbal distance is defined as binary or otherwise. The notation can be modified for 3x3’s, different four corners and other types of transformations.)

The mathematical notation makes obvious the *fourcorner\_solver* uses an identity transformation, i.e., and the distance metric, *d,* used for this solver is pixel differencing, which ignores *where* the differences occur.

Issues with using pixel differencing as the similarity metric when it is location independent are elucidated by Basic Problem B-04, in which there is ~19% difference between all pairs (A,B), (A,C), (B,C), (C,X) for X=1,2,3,5,6. (Note C = 4 and the difference = 0.) So the agent’s scoring function cannot credibly distinguish between these five solutions.

Defining **similarity as pixel differencing** while general and efficient in calculation has **accuracy issues**. I would have liked to try **Tversky’s similarity metric**. I have since discovered that it is part of Python’s GraphSim Toolkit 2.2.1, which contains many other graphical metrics. Other image similarity metrics I hope to explore include the **sum of squared errors**, **x and y axes projections** and **diagonal projections**. I hypothesize that there may be a mapping of one or a set of metrics to a transformation.

Whatever metric of similarity I would have chosen, **I would have liked to have compared the performance of many transformation rules**. Above, I indicated mathematically that there were many transformations of possible interest, e.g., reflection, rotation, dilation, contraction, deletion and addition, translations, etc. For 3x3 matrices, I would have liked to test rules to indicate sequences of object addition or other progressions of change along the rows and columns of the RPMs. The Python Imaging Library, PIL, is very user-friendly. It took about 1 minutes to test horizontal im.transpose(Image.FLIP\_LEFT\_RIGHT) and vertical reflection im.transpose(Image.FLIP\_TOP\_BOTTOM) functions and the rotation im.transpose(Image.ROTATE\_#degrees) functions with the Python interpreter. Reflection capabilities alone would have improved my agent dramatically. For instance, had I been able to implement these reflection functions as a transformation rule prior to testing the pixel difference, my agent would have corrected 3 out of the 4 incorrect solutions in Basic Problem Set B, 1 out of 9 incorrect Basic Problems C.

These examples and the definition of the *G* makes obvious the **potential computational complexity and intractability of RPMs problems** in that it affords not just some unlimited number of transformation functions, but also an unlimited number of ordered compositions of these transformation functions. It would appear, though, that the intractability is worse in the verbal problems.

I would very much liked to have experimented with **edge detection** and **object definition**. I would have liked to apply the “**fractal method**” and compared them to see if there were any advantages in defining objects. There are some RPMs in the Problem Sets D and E that have row and column progressions of star points. I find it difficult to believe that the fractal method catches these. I would like to implement these methods and test them for accuracy and performance.

Two additional issues I had in implementing new transformations was not being able to find thresholds to tests whether to accept a current solution, versus trying another rule. I also had issues defining appropriate data structure, e.g., here I’d need to maintain information on the transformation so that I might apply the transformation it as needed and also keep information on scores, the best transformation rule, and the current transformation.

My coding ability hampered me from experimentation as invariably a small change translated into hours of attempted implementation and in some cases due to my lack of know of how to use tools to control version, days. **I could not take risks because my level of proficiency in coding.** As I use this 3-week break to improve my coding skills, I will most definitely try to implement some of these rules. I realize the course was set up for us to be able to try to implement into our RPM programs the course concepts as we learned them. This would have been facilitated by having a “plug and play” structure to our programs. While I struggled to define object classes and methods, I noted in peer review I did that many students were able to define a holistic program structure for the RPMs with modules for retrieval, transformation identification, scoring, evaluation/solution selection, and storage. **I wish there had been a way to offer such a structure to students after Project 1.** This would have enabled students like me or students who didn’t have good structure but managed to get their program working well, to continue to benefit from the “learning by doing” instructional strategy in the course.

# 4.0 Solving RPMs to Learn about AI and Human Cognition

Using RPMs as a way to learn about KBAI, human cognition and learning was brilliant. Having the projects approach RPMs both verbally and visually was very instructive. I know that I use both approaches when reasoning and, although, I didn’t get them all to work, I learned lessons from my efforts. I use analogous reasoning to identify similar shapes in two Figures that I’m comparing. Then I note some differences in the frames and my verbal/logic kicks in.

I would have liked to succeed in all the approaches I started or took. I was very envious listening to students during office hours or on the forum who were keeping up with or who had gotten ahead of the course and had the opportunity test new methods they learned or their intuition. Although mundane, I wish I had succeeded in implementing the exhaustive enumeration mapping of the objects from one figure to another. This represents the **object level, reductionist approach that relies on objects and logic only in a generate and test mode**. It would have provided a baseline in whatever I ran subsequently on straight computational performance, based solely on verbal attribute matching and production rules based on logic – *modus ponens*. I could have modified the strategy to see what gains I might have had using **means-ends analysis** perhaps using a heuristic that reduces the size of the set of answer choices by only including plausible solutions based on total number of objects expected in the solution to the missing entry.

Then I would have pursued my **Figure level verbal attribute approach** where I focus on the set of attributes only in the base and only in the target Figures to derive a transformation (or I would have accommodated compositions of transformations with time). I would have compared the computational performance and accuracy of this approach with the enumeration approach to define the benefit of viewing the problem at the Figure level. I would have embellished

Next, I would have used my **VisualSolver** and recorded its accuracy and computational performance. I would have seen whether there were a way to compare it to the Figure level verbal attribute approach. Then I would have tried all the items I listed above such as **testing different scoring metrics**. I might have tried to use edge detection to increase my agent’s discernment of different objects.

Finally, I would have liked to apply the projection metrics to define **Problem level symmetries** that correspond to what any of us have observed when solving the RPMs. We immediately see the missing element through symmetry. All in all it would appear that we have the capability of visually approaching analogous reasoning and work done in this course provides more information to the research being conducted at GaTech. We have truly engaged in “authentic” learning. The work should be used.

I would have liked to have compared all these approaches and partial approaches. Research shows that individuals with Asperger Syndrome perform better using visual approaches as do even less performing individuals with autism. Are there additional groups who perform better using visual modes or verbal modes? What other characteristics of intelligence if any do individuals who perform better with visual reasoning or verbal reasoning possess? Are there integrated human and computer RPMs with which we could see how the KBAI tools could be run by human cognition to solve RPM tests faster?

I also would have liked to have seen how the order of the transformation tests mattered if at all. We were given weights that prioritized reflection, rotation, scaling, etc. Perhaps the agent could be exposed to many, many RPMs and learn the prioritization of these transformations to increase through use. This could be construed as a form of meta-learning.

# 5.0 Conclusions

There is a great deal of benefit in reflecting on what we learn and how we learn. Such activities enable us to improve our learning. So there are many levels to the intelligence we try to implement in this course. The RPM problem lent itself to so many different applications of KBAI. Each topic covered in the lectures was addressed in the way it would fit in with the RPMs. It is imperative that the student have a working agent with “plug and play” structure to allow the student to learn-by-doing the many interesting KBAI methods learned in the course. This reflection unfortunately for the most part could only address how these KBAI methods fit in and enhance the RPM solutions from a theoretically level. I plan on continuing to extend my code to see whether my understanding as presented here is actually correct.

# 6.0 References

Carpenter, P.A., Just, M.A., Shell, P. (1990). What One Intelligence Test Measures: A Theoretical Account of the Processing in the Raven Progressive Matrices Test, Psychology Review, 97(3):404-31.

Kunda, m. McGreggor, K., and Goel, A. (2010). Taking a look (literally!) at the Raven’s Intelligence Test: Two Visual Solution Strategies. In Procs. 32nd Annual Conference of the Cognitive Science Society, pp 1691-1696.

Raven, J. C. (1936). *Mental tests used in genetic studies: The performance of related individuals on tests mainly educative and mainly reproductive.* MSc Thesis, [University of London](https://en.wikipedia.org/wiki/University_of_London).

Raven, J., Raven, J.C. and Court, J.H. (2003). Manual for RPM and Vocabulary scales. San Antonio, TX: Harcourt Assessment.

# 7.0 Appendices

**Appendix 6.1**

def Solve(self,problem):

print ("Current problem = ", problem.name)

# Currently only solves for 2x2 problems. Will generalize later

# Figures A,B,C and 1-6 are structural components in 2x2 problems.

#Potential answers will go in a list

figures = problem.figures

A\_Objs = figures['A'].objects

B\_Objs = figures['B'].objects

C\_Objs = figures['C'].objects

# AnswerChoices for 2x2 to be used later to test the candidate solutions

Ch1\_Objs = figures['1'].objects

Ch2\_Objs = figures['2'].objects

Ch3\_Objs = figures['3'].objects

Ch4\_Objs = figures['4'].objects

Ch5\_Objs = figures['5'].objects

Ch6\_Objs = figures['6'].objects

def Correspond(Self,fig\_O1,fig\_02,fig\_Map)

# Find feasible mappings btwn all objects from one figure to another.

# a priori scoring is done. All mappings will be found using permutations

# These mapping will form basis for finding figure transformations which

# will be scored and used to generate and test solutions to Figure C.

A\_names = A\_Objs.keys()

B\_names = B\_Objs.keys()

n\_A\_Objs = len(A\_names)

n\_B\_Objs = len(B\_names)

# Special case : no deletions or additions

if n\_A\_Objs == n\_B\_Objs:

map\_fr = A\_names

map\_to = ([list(x) for x in itertools.permutations(B\_names,n\_A\_Objs)])

# Map-dependent deletions

elif n\_A\_Objs > n\_B\_Objs:

map\_fr = ([list(x) for x in itertools.permutations(A\_names,n\_B\_Objs)])

pdb.set\_trace()

map\_to = B\_names

# Map-dependent additions

else:

map\_to = ([list(x) for x in itertools.permutations(B\_names,n\_A\_Objs)])

map\_fr = A\_names

map\_frto=[]

if n\_A\_Objs == n\_B\_Objs:

for x in map\_to:

map\_frto.append(dict(zip(map\_fr,x)))

elif n\_A\_Objs > n\_B\_Objs:

for x in map\_fr:

map\_frto.append(dict(zip(x,map\_to)))

else:

for x in map\_to:

map\_frto.append(dict(zip(map\_fr,x)))

print ("Set of possible mappings from Figure A to B = ", map\_frto)

**Appendix 6.2**

"""

Created on Wed Jun 24 22:10:46 2015

CompareFigures defines variance for one base figure with target figures given

as an attribute. The variance is a set of tuples of attribute values

differences that can be used by transformation rules and heuristics to deduce

the transformations leading to the variance.

@author: miriamheller

"""

class CompareFigures:

def \_\_init\_\_(self,bf,tf,bf\_objs,tf\_objs):

# def \_\_init\_\_(self, bf,tf,figures):

self.bf\_name = bf # name of bf = base figure

self.tf\_name = tf # tf = target figure

# self.bf\_objs = figures[bf].objects

# self.tf\_objs = figures[tf].objects

self.bf\_objs = bf\_objs

self.tf\_objs = tf\_objs

self.n\_bf\_objs = len(self.bf\_objs)

self.n\_tf\_objs = len(self.tf\_objs)

self.bo\_tuples = self.calc\_bo\_tuples()

self.to\_tuples = self.calc\_to\_tuples()

# self.bo\_variance = self.calc\_bovariance() # tuples only in base objects

# self.to\_variance = self.calc\_tovariance() # tuples only in target objects

# self.transform = None # Consider making this a class

# self.score = None # Score associated with the transform

def calc\_bo\_tuples(self):

bo\_tuples = []

for bo in self.bf\_objs.keys():

bo\_tuples = bo\_tuples + self.bf\_objs[bo].attributes.items()

return bo\_tuples

def calc\_to\_tuples(self):

to\_tuples = []

for to in self.tf\_objs.keys():

to\_tuples = to\_tuples + self.tf\_objs[to].attributes.items()

return to\_tuples

def identify\_transforms(self):

bo\_not\_to\_tuples = list(set(self.bo\_tuples) - set(self.to\_tuples))

to\_not\_bo\_tuples = list(set(self.to\_tuples) - set(self.bo\_tuples))

for check in [self.is\_identity\_tranform]:

result = check(bo\_not\_to\_tuples,to\_not\_bo\_tuples)

return result

# couldputtransinfoinhere

def is\_identity\_transform(self,bo\_not\_to\_tuples,to\_not\_bo\_tuples):

if bo\_not\_to\_tuples == [] and to\_not\_bo\_tuples == []:

return "identity"

#

# def calc\_to\_tuples(self):

# for to in tf\_name.objects.keys():

# bos\_tups\_only = list(set(obj.attributes) - set(b\_tuples))

#

# def calc\_variance(self):

#

# return bos\_only\_tuples, tos\_only\_tuples

# Do something here

**Appendix 6.3**