CS 4635 - Knowledge Based AI
Brad Ware
Project 4 Design Report

Contents

1. How does it work?

As everyone knows, project 4 has been a totally unique and distinct challenge from the first three, and first of all I want to thank-you for the opportunity. I was exploited to so many new concepts with this project, and even though my agent score is not what I expected or wanted, I can still very much say I have gained a lot from this assignment. That being said, in this chapter I will detail not how my project works but how I envision it to work - implementation does not always match the innovator due to time and knowledge constraints.

A. Strategy

From a very high-level perspective, the strategy for each of the different types of problems is to analyze the differences in the pixels of each given image. Then depending on the type of problem (2x1, 2x2, and 3x3), you would compare different combinations of figures together based on the relationships toward the answer choices. The agent would then store the gathered information about each image and store the relationship differences in wrapper objects that can be compared later. After these comparisons, a scoring system will result to all of the answer choices, and the one with the highest score will ultimately be chosen as the answer for the problem. Here are some more detailed steps below overviewing the process:

First, utility functions are used to gather information on each of the images, specifically how they change pixel for pixel. Semantically, we try to gather what shape is being used, the fill of it, and how many objects are in each image. But on a higher level, also comparing each matrix of the image pixel by pixel to understand if any transformations were necessary, and what occurred. This will be stored in helper wrapper objects that I created in the earlier projects. For all of the given figures, A - H (depending on the type of problem).

Then, depending on the type of problem, these functions are then applied to the answer choices and there relation to the given figures. After this is calculated, we try to see patterns between the given figures and the answer choices. A scoring system is applied to each of the answer choices and the one that returns the highest score is ultimately returned as the answer.

However, because ties can occur frequently, a list is used to keep track of all the answers that tied with the largest score.

If the list is greater than 1, the real elimination heuristics occur that compare many categories trying to find differences that can differentiate answer choices from another.

Finally, the answer choice with the highest score based on its similarity differential rating back to the groups of comparison (based on the problem type) is ultimately returned as the answer.

B. Architecture

These four helper classes were transformed from Project 3 and adapted to hold useful information for this project. This “meta-data” about each image was used to try and gain some propositional knowledge, and combine that with transformations at the pixel level.

Class Pair (VisualRavensFigure, VisualRavensFigure)

The purpose of this class was to track the relationship differences between each of the objects in the different figures for 2x1 and 2x2 problems. This class kept a hash map which stored numerical indexes for each Object and that index hashed to an attribute in a different table. These attributes stored a difference string trying to describe what transformation occurred from figure to figure. Because of the lack of propositional representation this class should have been used but I could not gather enough information about each image.

Variables

- *fig1*: the first RavensFigure to be compared.
- *fig2*: the second RavensFigure to be compared.
- *score*: the score of the relationship between the two figures.
- *scoreMap*: a hash map containing each object index as the key and another hash map storing the associated transformation with the attribute.
- *diffNumObjects:* string value that compares the different number of Objects that each figure contains and returns the resulted transformation.

Key Functions

- *setUpScoreMap: initializes the hash map with the correct object index and builds the helper map that goes into more detail about each attribute change.*
- *getfDiffNumObjects: returns the transformation from the different number of objects.*
- *getfFillDifference: returns the transformation from the fill difference of objects.*
- *getSizeDifference: returns the transformation from the size difference of objects.*

Class ObjectPair (Ravens Object, Ravens Object)

The purpose of this class was to track the relationship differences between each object in a figure. This class kept a hash map of characteristics and a **string transformation** keeping track of the differences. It was hard to populate this class because propositional representations were not used in this project, however as information could be gathered about each image it was definitely used.

Variables

obj1: the first object to be compared.
obj2: the second object to be compared.
attrMap1: a hash map containing all of the attributes of obj1.
attrMap2: a hash map containing all of the attributes of obj2.
diffMap: a hash map containing all of the associated differences in attributes of the two objects.

Key Functions

compare: made an ArrayList displaying all of the attribute categories that were different between the objects.
setUpDiffMap: used the difference list to build a HashMap storing the category and transformation.
angleDiff: used to track the numerical differences between angles.
sameShape: determined if the object contained the same shape and stored that boolean value in the hash map.
sameSize/Fill: determined if the object contained the same size/fill and stored that boolean value in the hash map.
getSame… : getters for each of the boolean attributes comparisons
getValues: getter for the hash map containing all of the attributes and their boolean values

Class Group (VisualRavensFigure, VisualRavensFigure, VisualRavens Figure)

The purpose of this class was to track the difference between each VisualRavensFigure at the level of 3 at a time. This was used for the 3x3 problems. The Pair class was used for 2x1 and 2x2 problems. This class was populated less than the other three projects because of the lack of propositional representations.

Variables

- *fig1*: the first RavensFigure to be compared.
- *fig2*: the second RavensFigure to be compared.
- fig3: the third RavensFigure to be compared.
- *score*: the score of the relationship between the two figures.
- *scoreMap*: a hash map containing each object index as the key and another hash map storing the associated transformation with the attribute.
- *diffNumObjects:* string value that compares the different number of Objects that each figure contains and returns the resulted transformation.

Key Functions

- *setUpScoreMap: initializes the hash map with the correct object index and builds the helper map that goes into more detail about each attribute change.*
- *getfDiffNumObjects: returns the transformation from the different number of objects.*
- *getfFillDifference: returns the transformation from the fill difference of objects.*
- *getSizeDifference: returns the transformation from the size difference of objects.*

Class ObjectGroup (Ravens Object, Ravens Object, Ravens Object)

The purpose of this class was to track the relationship differences between each of the three objects in a figure. This was only used for the 3x3 problems because ObjectPair was used for the other types of problems. This class kept a hash map of characteristics and a **string transformation** keeping track of the differences. Any different characteristic was stored in the hash map, and it was taken into consideration the number of characteristics and the differing possible values. This class was not used much due to the nature of this project.

Variables

- *obj1:* the first object to be compared.
- obj2: the second object to be compared.
- obj3: the third object to be compared.
- *attrMap1*: a hash map containing all of the attributes of obj1.
- *attrMap2*: a hash map containing all of the attributes of obj2.
- attrMap3: a hash map containing all of the attributes of obj3.
- *diffMap*: a hash map containing all of the associated differences in attributes of the three objects.
- diffList: a list of the attribute names that are different from each other.

Key Functions

- *compare:* made an ArrayList displaying all of the attribute categories that were different between the objects.
- *setUpDiffMap*: used the difference list to build a HashMap storing the category and transformation.
- angleDiff: used to track the numerical differences between angles.
- *sameShape*: determined if the object contained the same shape and stored that boolean value in the hash map.
- *sameSize/Fill*: determined if the object contained the same size/fill and stored that boolean value in the hash map.
- *getSame…* : getters for each of the boolean attributes comparisons
- *getValues*: getter for the hash map containing all of the attributes and their boolean values

Class AttributeGroup (Ravens Attribute, Ravens Attribute, Ravens Attribute)

The purpose of this class was to track the relationship differences between each Ravens Attribute in a figure. Simply stored the transformation difference between the two attributes that were of the same category.

Variables

attr1: the first attribute to be compared.
attr2: the second attribute to be compared.
index: the attribute's name that was used to store the category.
diffValue: a string that stored the difference between each attribute.

Key Functions

getDiffValue: returned the transformation between each attribute's value.
getIndex: returned the index/name that was categorizing this attribute.

Class Problem (Ravens Problem, Ravens Figure)

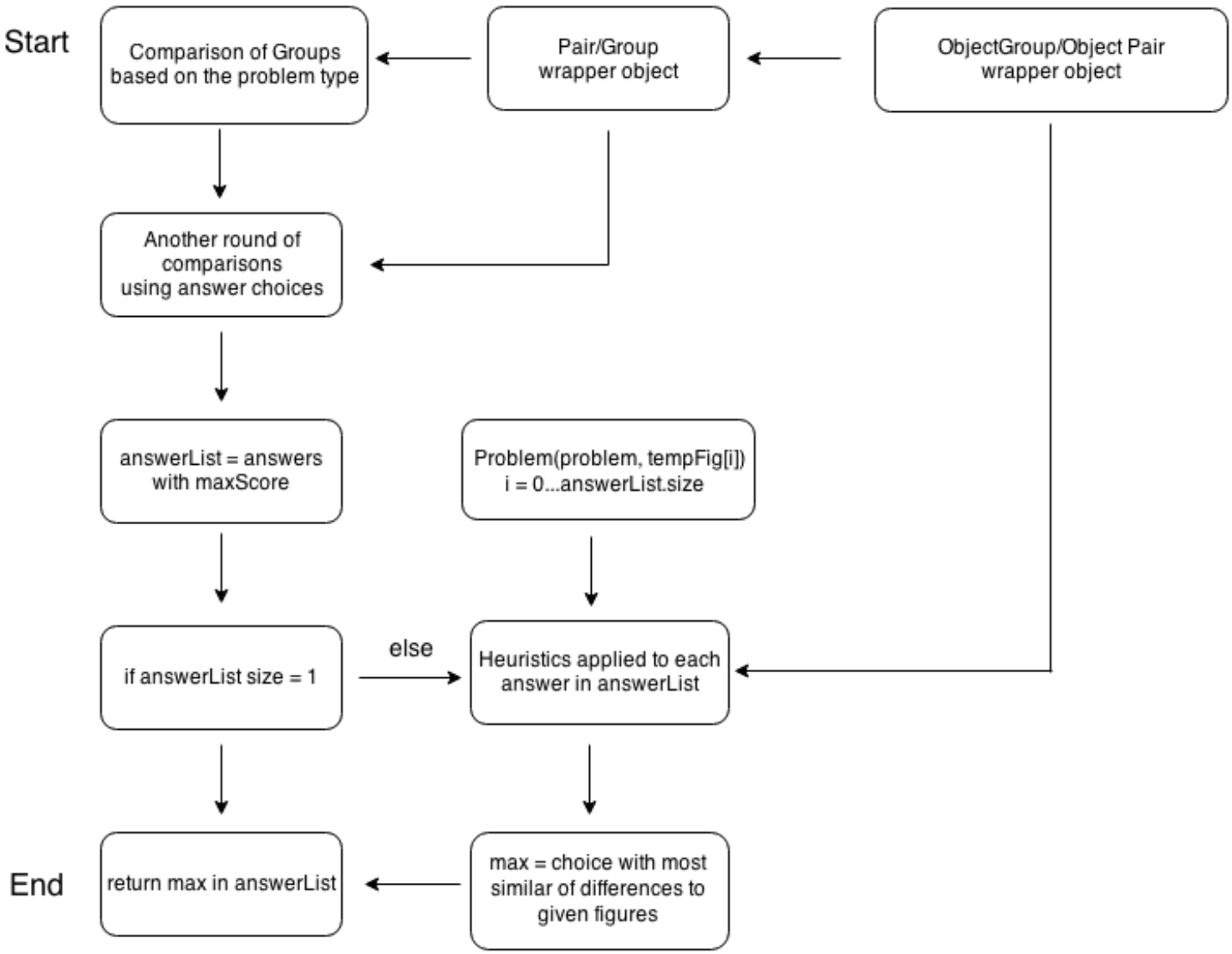
The purpose of this class was to run heuristics on the given problem due to the horizontal and vertical relationships across figures. Then, this was used to compare to the relationship formed from the temporary Raven's Figure passed in as a parameter. This class was a container for library functions that solve leveraged. This class did not end up playing a major factor due to the lack of information gathered about each image.

Variables

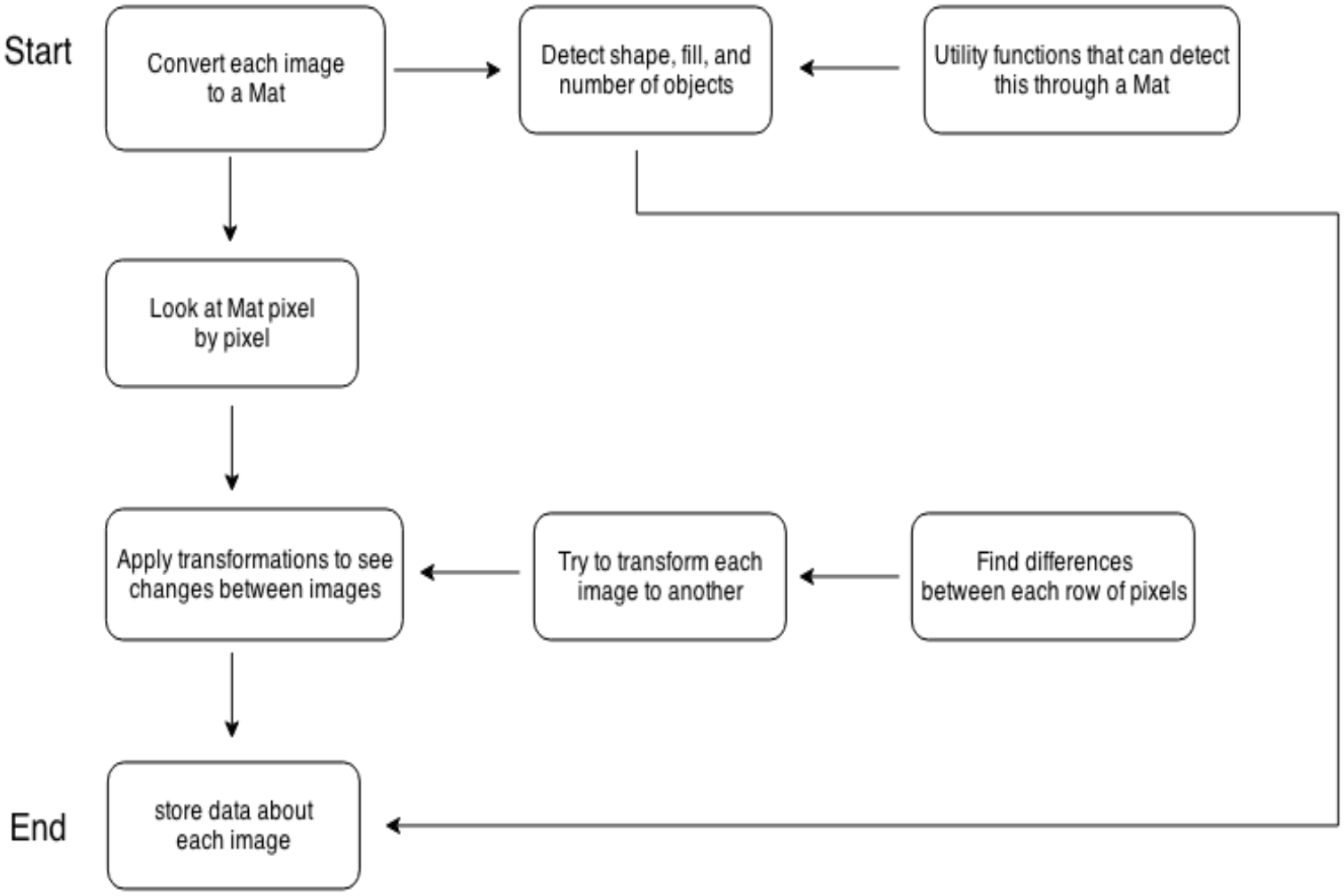
- *problem:* the Ravens Problem passed in as a parameter to the constructor.
- *tempFig*: the Ravens Figure passed in as a parameter to the constructor.
- *figA:* Figure A from the problem passed as the parameter.
- *figB:* Figure B from the problem passed as the parameter.
- *figC:* Figure C from the problem passed as the parameter.
- *figD:* Figure D from the problem passed as the parameter.
- *figE:* Figure E from the problem passed as the parameter.
- *figF:* Figure F from the problem passed as the parameter.
- *figG:* Figure G from the problem passed as the parameter.
- *figH:* Figure H from the problem passed as the parameter.
- *figureList*: array list that contained all of the figures attached to each problem.

Key Functions

- *shapeLeast:* returned true if a shape occurred the least amount of times throughout the figures given and was the same shape as the tempFig.
- *sizeLeast:* returned true if a size occurred the least amount of times throughout the figures given and was the same shape as the tempFig.
- angle*Least:* returned true if an angle occurred the least amount of times throughout the figures given and was the same shape as the tempFig.
- *numObjectsIncrease*: returned true if the number of objects increased horizontally or vertically as you went down the row/column.
- sizeIncrease/sizeDecreased: returned true if the size of each object increased/ decreased horizontally or vertically as you went down the row/column.
- sameShape: returned true if each row and column contained the same shape.
- directionAlt: returned true if the angle/direction of each shape alternated as you went down the row or column.



Flow diagram of how Solve finds the answer with propositional knowledge about the problem



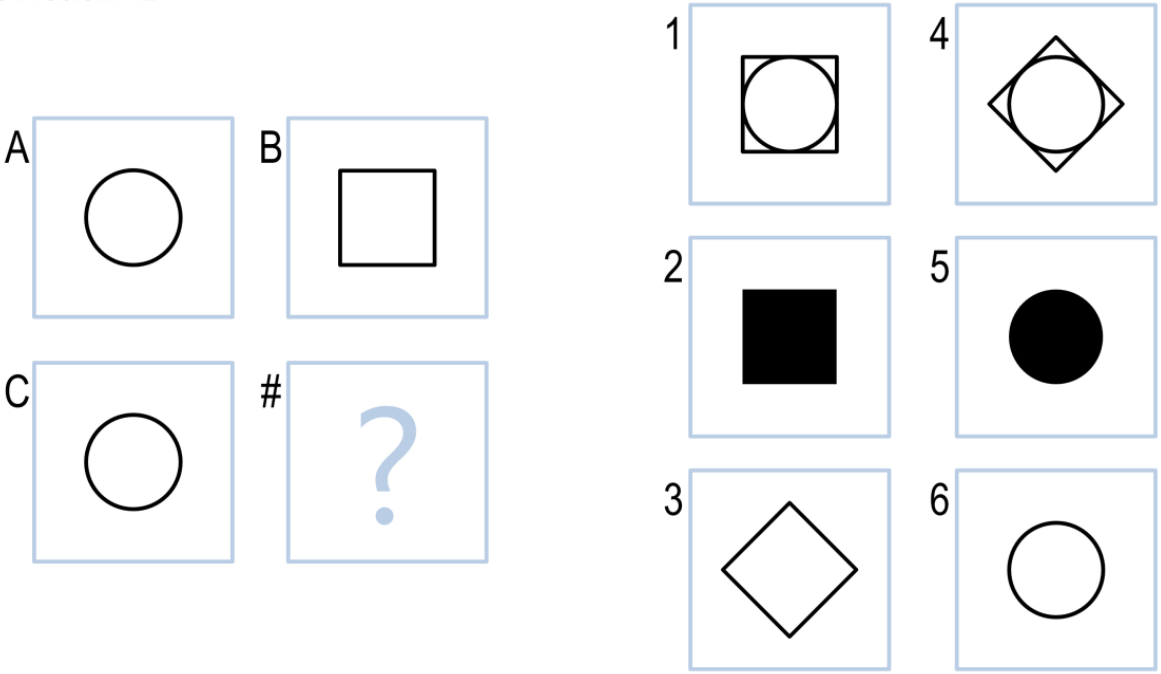
Flow diagram of the strategy to gain inference on each visual image

2. Application

We will walk through two example problems using both of the strategies that I wanted to incorporate. The first is collecting propositional knowledge about the problem and then using that information in a strategy similar to the last three projects. The second example is a different way at analyzing each problem at the pixel level and observing the changes there. The second approach was much more challenging and I was not able to implement it successfully, but hopefully with more experience I will be able to in the future.

A. Example Problem Walk-Through

2x2 Basic Problem 12



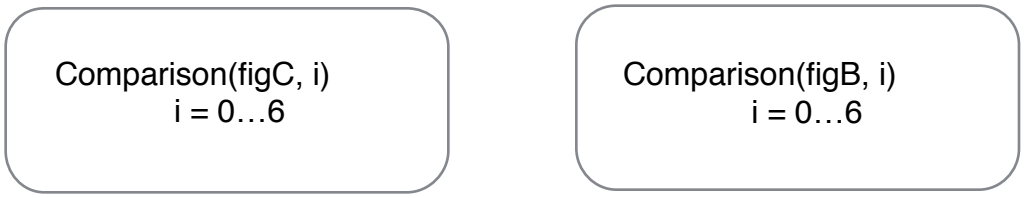
In this problem, in my first strategy we will try to tackle it by identifying the types of shapes in each figure, and then store these to gain inference on the problem.

Comparison of Pairs
AB & AC

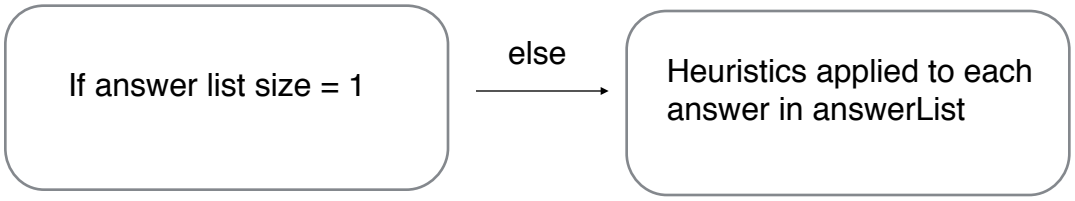
From these comparisons we will notice that each Pair has a different shape of object horizontally and possibly a different angle vertically. This can be found through our HoughCircle and HoughSquare functions. The angle cannot be detected through my image processing, but elimination will take care of this issue.

Hash Map of Groups ABC, DEF, ADG, and BEH	
Shape	VARIES
Fill	SAME
Angle	DON'T KNOW
NumShapes	SAME

From this, we can pass this propositional analysis to our wrapper classes and apply heuristic logic to the problem



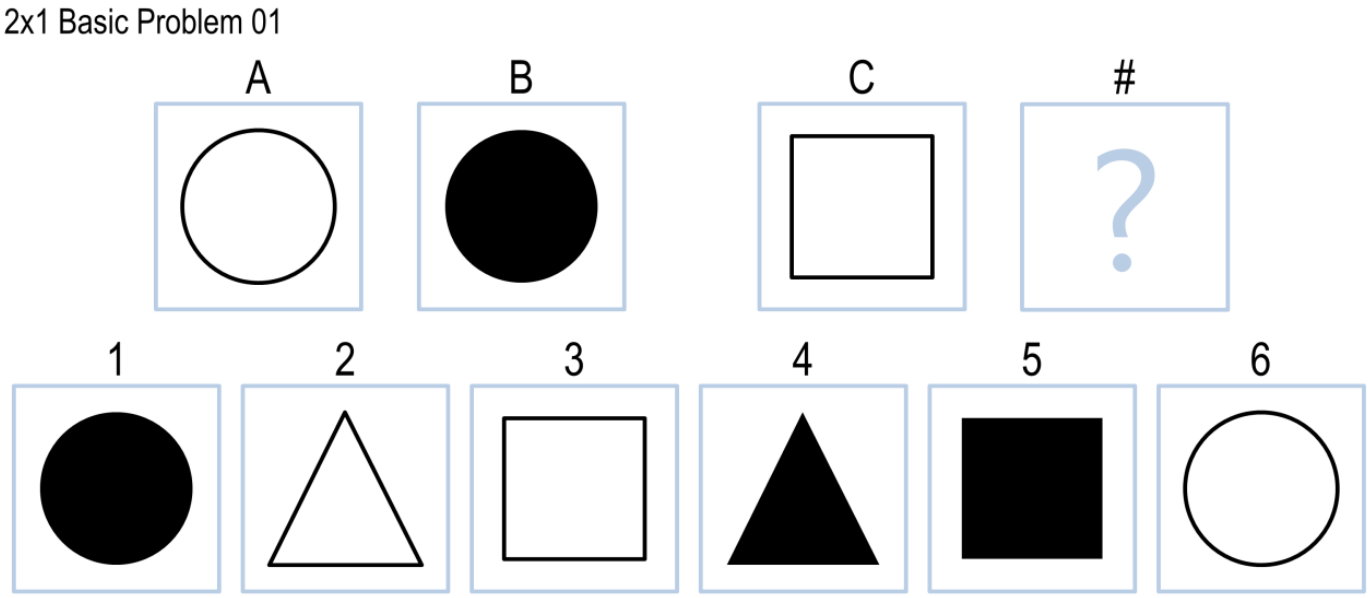
Therefore, when we run the comparison of all the answer choices, we need to look for shape variations horizontally and something else vertically. Therefore, our heuristics will eliminate figures 1, 4, 5, and 6 from the number of objects and also the shape. We know it must be a square.



So know we are left with just figures 2 and 3 as possible answers. Because none of the other objects are filled, our findFilled function heuristic will be applied further to eliminate answer choice 2. Finally, my agent will return:

answer = "3"

The strategy above was the original way of solving these types of problems - taking the visual representations and trying to convert them to propositional facts. From there, we would pass along the data for the images and apply the same strategies used in the previous three projects. However, I failed in pursuing this strategy, but I would like to explain it further because I think it's neat and really plays to the strengths of visual representations. This strategy will include the transformations of images by analyzing them at the pixel level. Let's look at an example problem and walk through how I wanted my agent to solve it:



In this problem we will break down the figures of A, B, and C first to understand anything about the images, and then from there try to solve the problem. The first step will include to turn each of the figures into Matrices representing the image:

MatA, MatB, and MatC stored

After we have the matrices, we can then analyze each image pixel by pixel. In this example, we can find through HoughCircle that figures A and B both contain a circle, but how would you transform figure A to B? That is the ultimate question, and obviously the answer is to fill the circle. So when analyzing the Matrices of both A and B, we can detect there is a circle, but what's different about the circle? How does the circle in A transform to the circle in B? After the fill value is stored as the transformation, which can include many especially for the complex problems, we will then map this same transformation from C to the answer choice.

```
MatC ——> Mati
for i = 1…6
```

After we apply the same transformation to figure C shape, we will find the only figure that matches this is answer choice 5. Therefore, our agent will return

answer = "5"

3. Shortcomings

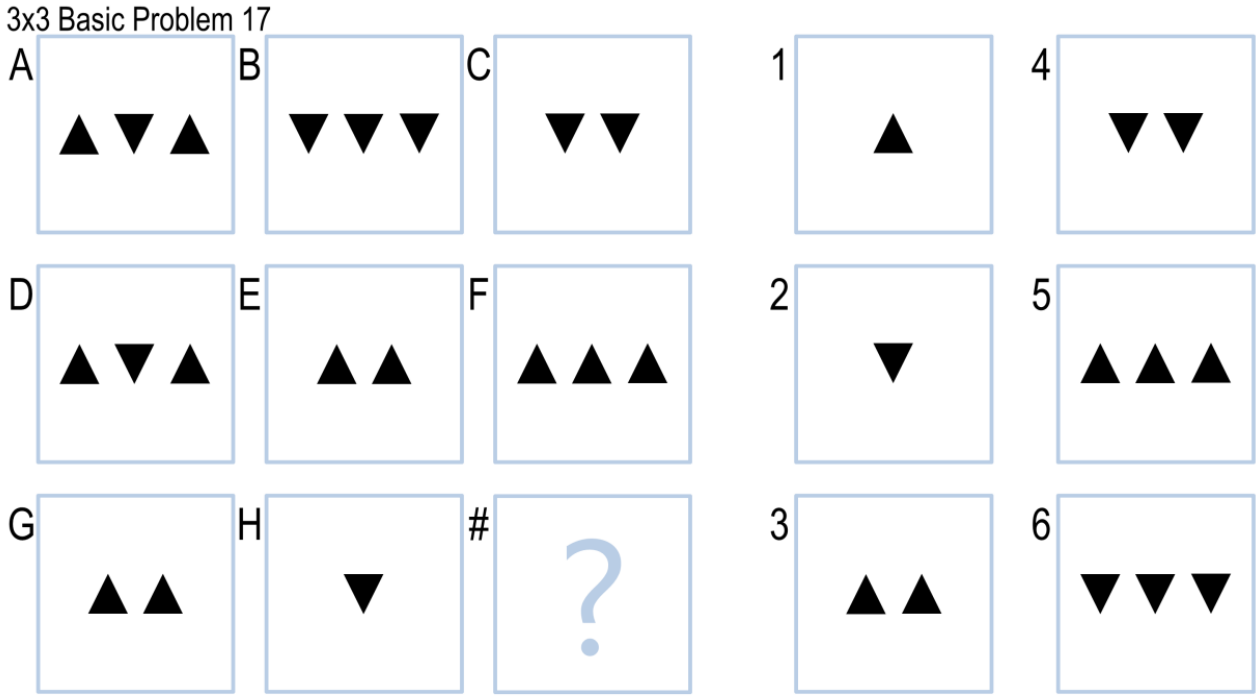
My agent falls short…a lot. More than I would like for it too. One of the biggest issues is dealing with visual representations. Because its hard to program an agent to represent transformations of pixels, I fell back to implementing by finding propositional data about an image, and passing that information along to my heuristic functions. The only problem is that my agent has a hard time detecting basic shapes and also the number of shapes. This makes it hard because it cannot eliminate some of the easy answer choices.

A. When does it fall short?

One of the simplest ways my agent gets tripped up is through multiple figures that contain multiple object. This poses a challenge for my agent because it has a hard time reading the total amount of objects because it does not always accurately find them. It also struggles to find any of the objects that are shapes beyond the normal square, circle, or triangle. Because of this a lot of the later problems in each section pose a significant challenge to my ever blind agent. Lastly, sometime it fails to recognize even the fill of objects, again making it hard to pass even some of the simplest problems.

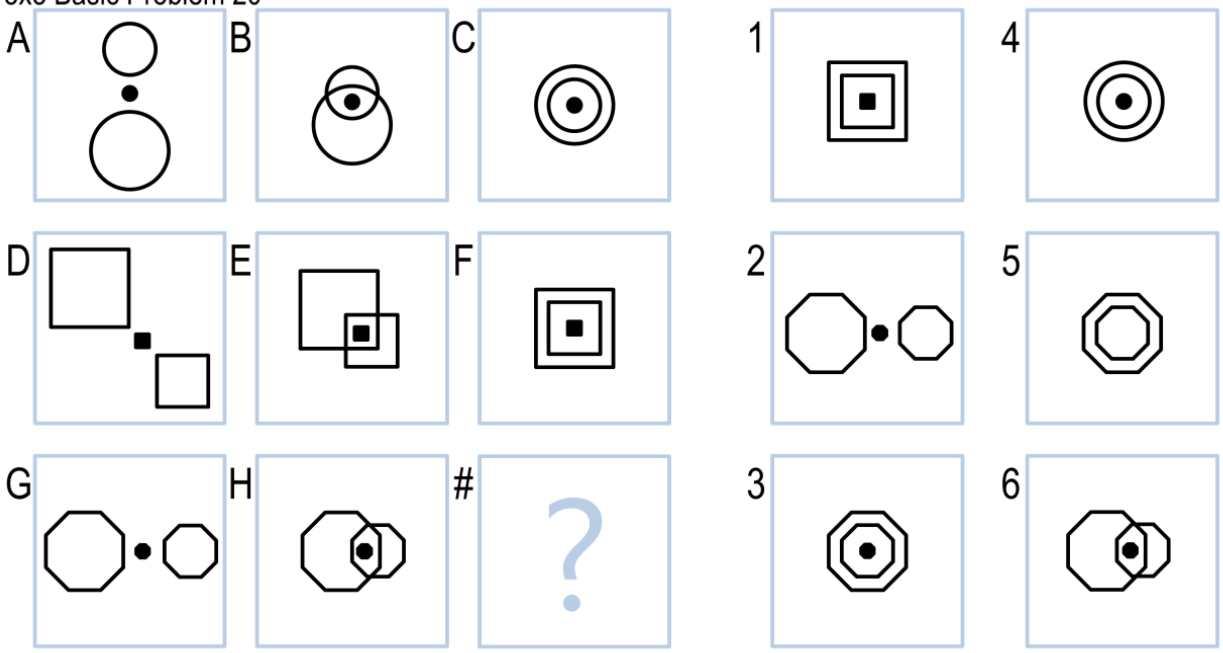
B. Why?

I thought it would be easiest to demonstrate this through a short example.



Right now, my agent will be able to detect the triangles, but has no standard of how many appear. It will detect in each unique figure that there are more triangles than others, even though they have the same amount. This will mess up the relationships between each row and column, such as ABC and BEH. From here, my agent will have no way of identifying what could possibly even be the correct answer. My agent cannot even decipher a better possible answer from any of the six answer choices, as it sees them as all of the same. Simply lacking a stable way to detect the amount of objects really hinders the agent.

3x3 Basic Problem 20



This is another problem that my agent has no chance of solving. This problem introduces octagons, which is a shape that my agent cannot detect. It finds an octagon to be equivalent to a circle. It also really struggles because this problem focuses on the particular location of each object within an image. A problem like this would be excellent for transformations, because you simply could calculate the differences in positions of each object, and record them. However, there is no easy way for our agent to identify the locations of the same objects throughout different figures, as it incorrectly labels them. My agent thinks answer choices 1, 2, and 6 are possibly the right answer through blind judgment of the type of shapes. However, answer choice 3 is ultimately the correct answer due to positioning of the objects. Relational positioning of different objects inside a figure is another challenge for my agent.

4. Potential for improvements

A. What could have been done better?

There are so many improvements for my agent, I honestly could have spent all semester trying to solve project 4 alone. However, I did identify key attributes that would have really helped if I had them.

First, I would have loved a stable way to classify a type of object. It doesn't matter if I used a Hough method, or even just identifying lines and then corners, but a sophisticated system to understand the shape is essential for these types of problems. I would say the number one piece of information an intelligent agent has to have to solve Raven's progressive matrices is identifying different shapes. Unfortunately my agent could not do that. That being said, if I could have found a way to compare matrices of images to the pixel level then it would have been a lot easier. I then could have transformed different sets of pixels to another and recorded the differences.

Secondly, I should have invested more time in understand the computer vision side of this project. I wrong assumed that there would be a simple API for making calls to identifying shapes, fill, color, angle, etc. However, I was very wrong. There are many functions in the javacv jar file, however it still takes work and a decent understanding to apply them in the correct manner. Just having a simple background in the strategies in basic computer vision for image processing would have made my google searches more efficient and the API I had access to a lot easier to navigate.

Lastly, besides the computer vision aspect, I wish I had dedicated more time to the transformations of pixels strategy to identify images. This would have used the visual representations positively instead of working against them in just finding the propositionally representations. Even though I didn't have the knowledge to accomplish this, if I had started earlier I would have known the issues I was going to run into and could have found help quicker.

This project was a lot of fun to attempt, even though my results were not as expected. These are just three simple ways that I would do differently next time if given the opportunity to solve Raven's progressive matrices though visual representations again.

5. Run-Time Analysis

A. How can it be solved more efficiently?

I have yet to see any real efficiency gaps because of my implementation, but that's mainly because the problem space is so small. For instance when AI agents are trying to play chess, the branching factor can become huge, making efficiency one of the most important things. To analyze how my agent runs the solve function, I am going to break down each of the steps it runs through, and try to quantify the run-time there:

For this, we will assume each image is the same size, 184 x 184 pixels.

Converting each image to a matrix of pixels: *O(k)*

Iterate through each row of pixels in each figure: *O(n)*

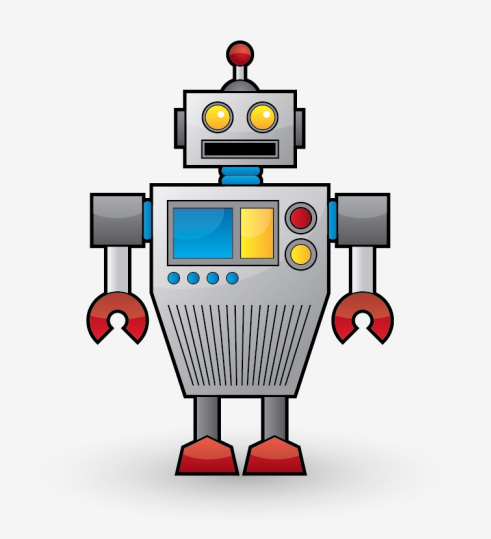
Iterate through each column of pixels in each figure: *O(n)*

Obviously, these steps are run multiple-times throughout solve, but as these numbers approach infinity, addition in big-O goes to 0. So, if I had to estimate the worst-case run-time of my solve algorithm at one point, I would say it's *O(knn)*, where *n^2* is the number of pixels in each image, and *k* is the number of computations it takes to convert an image to a pixel matrix. The run-time could become much worse if each figure started to contain many more objects, where each object had many attributes. This would cause my computer vision utility methods to be run much more each iteration trying to detect more propositions. Right now there also are only 6 possible answer choices, but obviously this could be increased as well, hindering the run-time.

This run-time could also be improved by simplifying some of the computer vision object detection methods. I run them all separately, but if I understood it better I could combine the results from both so that duplication is not rampant each time. I'm sure the results of the initial matrix could be used to detect later if the object is filled or if it's a square/triangle. This duplication is wasted computation and could be reduced by having all of the image processing methods build on each other. I store the propositional data from each image in hash tables so the lookup is just O(1) access time. However, majority of the time complexity comes from analyzing the image.

6. Relation to Human Cognition

A. How “intelligent” is my agent?



I thought after project 3 that I had developed an intelligent agent. However, after having to design an agent that reasoned in representations other than propositional, I realized that intelligence is a high standard to accomplish. In the first three projects, I built a gigantic semantic network where I added in many factors apart of Raven’s progressive matrices such as shape type, fill, angle, size, and position. However, this data was given to my agent in text! This is completely unrealistic to the actual world, where we as agents perceive everything though visual representations. This is true for robotic agents as well as they have to perceive objects, depth, and distance through visual perceptions. So when trying to accomplish this though my agent of project 4, I couldn't even fully get it to process images given that were a standard size. This is not realistic at all to an actual agent as myself where the world has no standard, and I have to manually adjust everything into terms that I can handle. So I would say no my agent is not intelligent at all as it can barely recognize different shapes given as input on a standard image file.

However I will say that if somehow the computer vision aspect of my agent were to be drastically improved, it definitely has qualities of intelligence. It can reason and eliminate problems mimicking many strategies that humans use on multiple choice exams, such as the SAT. One could even argue that vision has no effect on whether a being is intelligent, because one would never claim a blind person is unintelligent just because they are blind. Likewise, I would not call my agent unintelligent just because it lacks image processing and basic computer vision. If given the right representations, it can definitely solve Raven’s progressive matrices very effectively. So my answer is very conflicted. If you are comparing my agent to a person where they are both evaluated in the world, then by no means is my agent intelligent. However, if it is understood that my agent is “blind” in some terms, then it is definitely intelligent enough to solve problems such as Raven’s progressive matrices.

7. Visual vs. Propositional Representations

A. How did it compare to the first 3 projects?

As I have stated earlier, this project did pose significant challenges that I didn't have to face in the previous projects. Working with visual representations did have positives as well, opening new outlooks to solving this problem. However, even though I did not get as many problems correct as I would have liked, this project did allow me to think of solving Raven's progressive matrices in a completely new way.

First, I will start off with the challenges. I had no prior experience with computer vision before this assignment, so I honestly had no idea of even the basics to processing the content of an image. After finding the javacv jar file, it still took me a while to understand how to identify different shapes such as circles and squares. Even after that, I couldn't get my agent to find different objects fill and size consistently, giving my agent incorrect data on a lot of problems. So one of the biggest challenges was just getting correct data. After that, I had no way of perceiving nearly as much data for each problem as when I had propositional text files. When given the information about each figure propositionally, I immediately could store meta-data such as the number of objects, there fill, the name of the shape, and etc. This guaranteed that the information was correct, and gave my agent a wide range of categories to build a semantic network. However, with limited computer vision functions to break down the image propositionally, I really struggled determining the difference between many of the images. This made it extremely hard to eliminate any of the answer choices.

Yes, when trying to solve these problems by interpreting the images into propositions, it was much more difficult than the other projects. However, when looking at each problem as a matrix of pixels instead of true/false values, it becomes much more clear. If I could have successfully designed my agent to find transformations between different figures, then using visual representations would have been easier. However, because I have such little experience with the fundamentals of image detection or computer vision, I had no background in even understanding what to look for in a matrix of binary pixels. But with more experience and practice, using visual representations with a problem like this could be much more powerful in terms of the amount of code you would have to write, and the immediate results. Because Raven's progressive matrices revolves around transformations of images, then finding a way to read an understand the differences between images could have immediately given the solution. Instead I developed a scoring system based off of similarities and differences that were developed over a semantic network over time. Visual representations open the door to a completely different way of thinking about solving Raven's progressive matrices.