Angelina Kelly
ankelly@gatech.edu
CS 7637 - Project 1

Design Report: Project 1

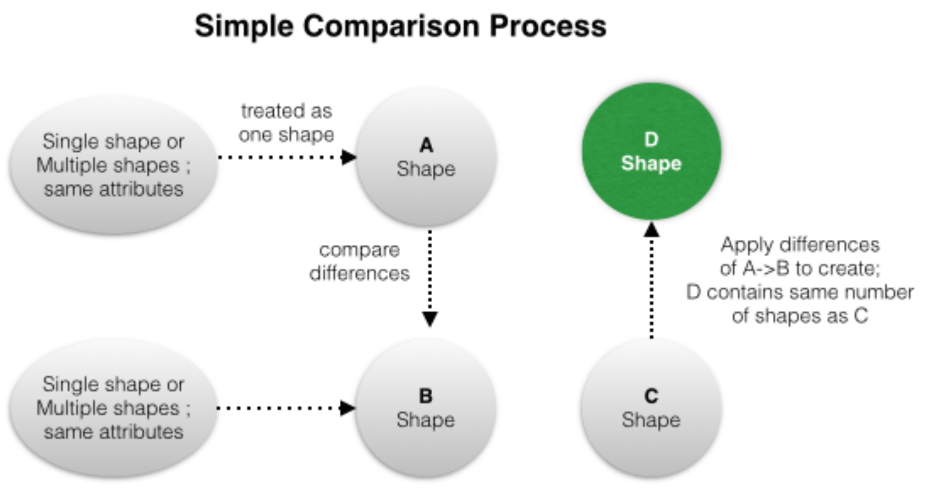
The following report covers details on the agent I built to solve 2x1 Raven's Progressive Matrices. The agent uses a few techniques to solve the various problems it encounters. The techniques utilized were a mix of means-ends analysis and generate and test. In designing the agent, semantic networks had an influence but did not make it into the agent as an algorithm. The agent tries to determine the correct answer by examining each of the figures provided and each of the shapes within the figures. The agent determines the changes between figures A and B and applies those changes to figure C in order to solve for the answer, which I label D. The agent also does not look at the possible solutions as a means of finding a solution, rather it creates a possible solution and then compares to the solutions provided.

The basic idea that the algorithm implements is if there is a change between figures A and B then the same change will apply from C to D. The changes were broken down into the common elements that occurred which were shape, position and modifiers. A ShapeItem class was created that had the attributes name, shape, position and modifiers. The modifiers were size, fill and angle or any unknown modifiers that might appear. Although I didn't break the modifiers down within the class, they were referenced frequently and in future implementations, they will be added as attributes individually.

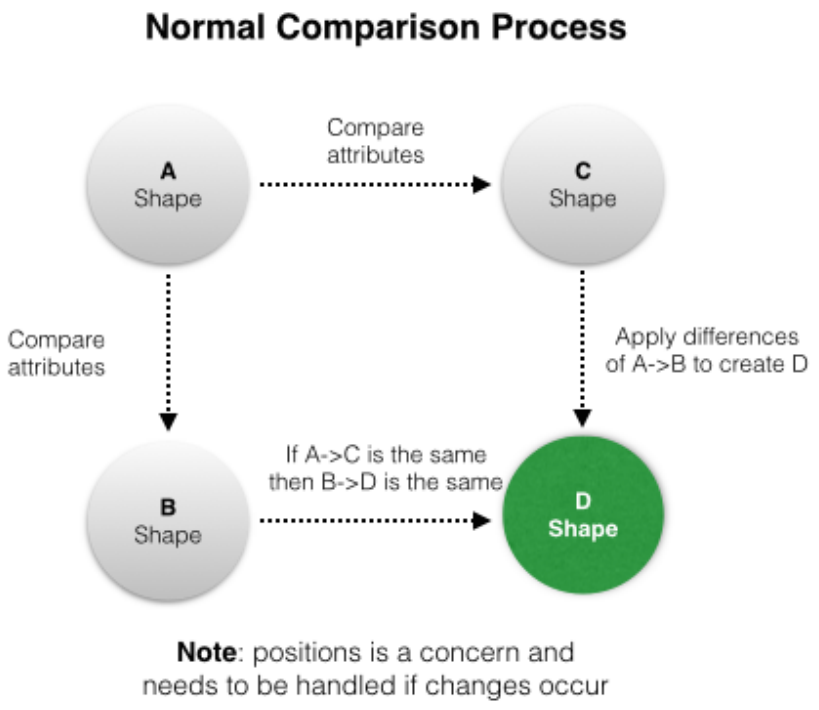
To figure out D, each of the different components of the figures had to be compared. For each of the figures, if some part of A and B were similar then the same part would be similar between C and D. Alternatively, if components of A and C are similar, then B and D would also be similar. Each of the attributes for D was created in a step by step process after reviewing A, B and C. One of the drawbacks of my agent is that it utilizes the given names (X, Y, Z, etc.) extensively. These names were used to compare figures in A, B and C and then create shapes in D with the same names.

In order to compensate for instances where the names given name of the shapes didn't match up, I did create a function to create new names for the shapes within the figure and then compare the shapes between A, B and C to create D. I also had to create a function to fix the positions of each figure so that they would align with the new names. The function to rename shapes examined Figures A, B and C to find the common shapes. Each of the common shapes would receive the same name. Then the function finds the common shapes of A and B, and A and C. Any shape that is similar between A and C will have the same label and for B, the labels begin with any shapes that are common between A and B. There are some limitations in this label mapping in that if there are multiple of the same shape, it won't work. That can easily be fixed but I was looking at solving some specific problems that had unique shapes in them. In the next project, the alternate mapping will need to be more robust as I will rely less on the given labels.

Something I noticed about some of the puzzles is that they had multiple shapes that were exactly the same in each figure. To take advantage of this, I created a simple comparison function which basically treated multiple shapes of the same attributes as a single shape. Then D was created based on the changes that occurred between A and B applied to C. Many of the problems were able to be solved by just using this simple comparison including a puzzle where C had many more shapes than both A and B. The simple comparison didn't worry about positions because positions didn't really matter and D would have the same positions that were in C. The diagram of the process is below.


Simple Comparison Process
Single shape or Multiple shapes ; same attributes
treated as one shape
A Shape
compare differences
Single shape or Multiple shapes ; same attributes
B Shape
D Shape
Apply differences of A->B to create; D contains same number of shapes as C
C Shape

For puzzles that had multiple shapes that had different attributes, that is when shapes fell under the normal comparison algorithm. The normal comparison algorithm had to take into account that there would be different shapes and positions could move as well as needing to compare each shape in each figure in order to come up with a shape D. This is also when labels became important and may need to be remapped or compared in a different way. Another factor that I noticed is that if there was an attribute or even shape changes that was shared in A and C but not in between A and B, then B and D would share the same attribute or shape changes. The diagram of the process is below.


Normal Comparison Process
A Shape
Compare attributes
C Shape
Compare attributes
Apply differences of A->B to create D
If A->C is the same then B->D is the same
B Shape
D Shape
Note: positions is a concern and needs to be handled if changes occur

Once a D shape is built, it is ready to be tested against the possible solutions. If no matching solution is found, there are a few options that are provided. If the figures have position attributes, then the function to match a solution to the solutions given will also keep track of an alternate solution. The alternate solution checks to see if a position attribute exists but will not check the value. If the regular solution doesn't match, but the alternate solution does match, then the value that is returned as an answer is the alternate solution.

Other methods I use if no solution is found initially is to go through the algorithm again with an 'alternate' flag and make a few changes. Since my agent relies heavily on the given mapping, I created a function to rename all the shapes within the figures and then run the figures with their new names through the agent again.

There are also a few very specific to the problem set options that are included to find alternate solutions such as if a shape in B is deleted, delete the first shape instead of the last shape, which is what would've been done through the first run. I also created a special function for what I called the inside-out method in the instance where you have 3 shapes of size small, medium, large where the small is inside of the medium and the medium is inside of the large and the positions change, thus changing the sizes of the shapes. Note, if sizes were expanded here, the method could be modified to include additional sizes but since a small will only fit inside a medium or large and a medium will only fit inside a large, I assumed that there would only be the 3 sizes to worry about. If after testing against all alternate methods, then there is still no solution match, my agent does what any human would do in such a case and pick one of the solutions at random.

There are quite a few limitations in the agent overall. I mentioned previously that the given object names are used as a default, even though the agent does have some ability to remap to new labels. The agent was also not built to deal with comparisons that come with morphing of figures such as merging shapes to create a different shape or deleting/adding sides to a shape. And the agent is not able to add a shape that exists in B but not A. The agent is able to solve the 20 basic problems and 1 challenge problem but was not able to solve the rest of the challenge problems or the student problems.

There are a few improvements I'd add if I had more time which include a more robust object naming, the ability to count sides in shapes, break out some of the known modifiers into attributes in my ShapeItem class and also do an "educated guess" algorithm in the event that no solution matches the solution the agent creates. It seems feasible that an agent wouldn't be able to solve all problems but an agent may be able to rule out solutions and narrow down the random guess to just a few, which would result in a higher chance of guessing correctly.

Although I would describe my agent as fairly basic, I think it does a decent job. There are some improvements that would help it perform better and it will need to be tweaked for 2x2 matrices. I expect the agent will continue to morph and grow as the matrices and challenges grow.