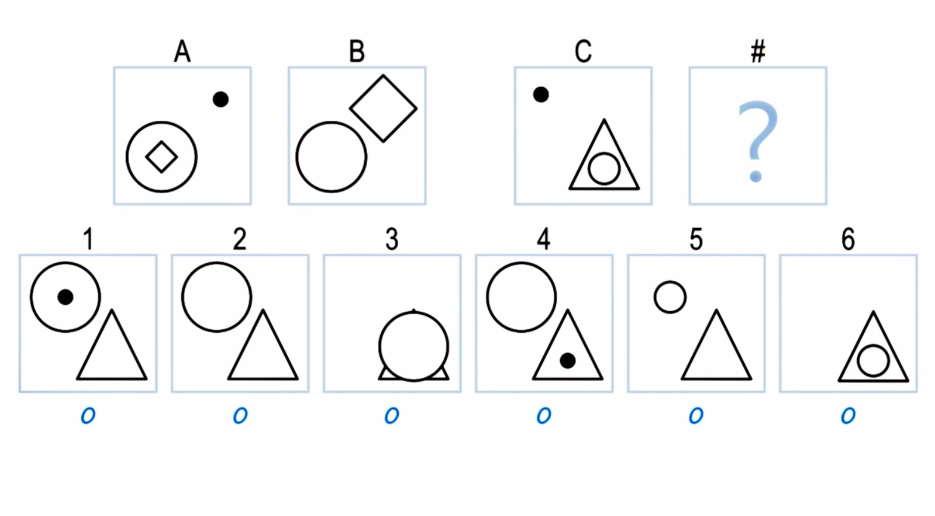
CS 7637: Project 1

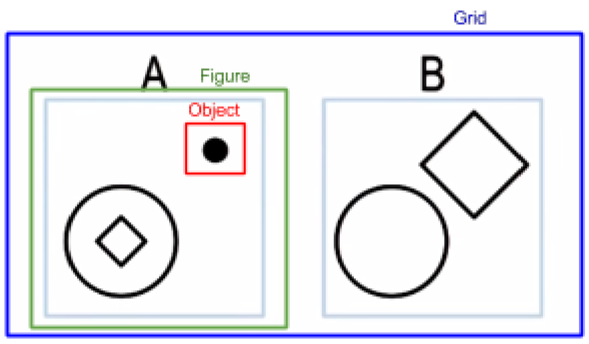
Author: Swapnil Ralhan(sralhan3@gatech.edu)
January 31, 2015

Introduction

This project is an attempt at solving basic Raven's Progressive Matrices problems of size 2x1.



For the purposes of the following discussion, we will refer to the following vocabulary. In any given problem, we will have a ***Grid***, which consists of a certain number of ***Figures***. Within a given figure, we have a number of ***Objects***, each of which has a certain number of ***Attributes.***



There are 3 main kinds of attributes:

1. **Structural Attribute -** Such as shape, size and fill. These are specific to one object, and the attributes and values are similar across different shapes
2. **Orientation -** The angle at which the object is facing, or it's vertical-flip. The attribute values are same across shapes, but the resultant transformation will be specific depending on the shape. For example a circle will be the same no matter what the angle is, while a triangle would change dramatically.
3. **Location -** This is the attribute that specifies the location of an object with relation to another object, such as above, inside, left-of and overlaps.

For a 2x1 problem, we have 3 figures provided A, B and C. We are supposed to guess the figure # from the options such that the ***Transformations*** between A and B are the same as the transformations between C and #.

General Idea

We represent this problem using a semantic network representation, where the transformation between an object to another can be thought of as representing the structure, the objects being the nodes, and the semantics are the specific list of attributes and how they are transformed. More details on these semantics are described in the implementation section.

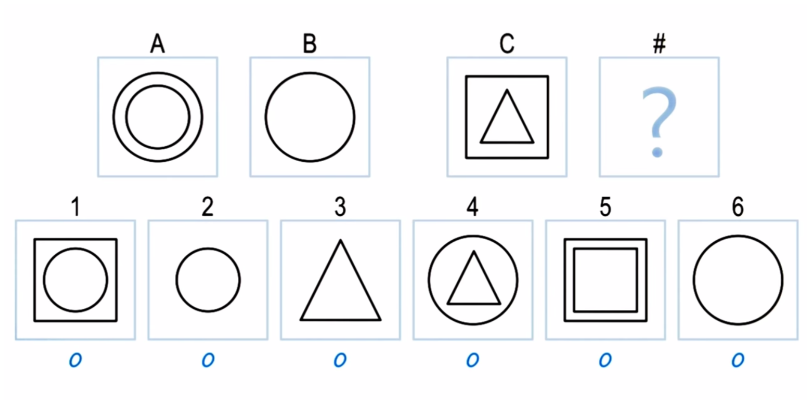
We use 2 nested versions of Generate and Test to sum the overall execution of the agent. It works as follows:

1. Generate all possible transformations from A to B
    a. Generate all possible transformations from C to each potential solution.
    b. Test to see if any of those transformations are identical. If so, the solution is valid.
2. Test to see if any of the transformations worked. If not, guess from one of the options.

All possible transformations from a figure to another, is simply the mapping from all possible objects in the first figure, to all possible objects in the second figure. For instance, if figure A has objects [square, circle] and figure B has objects [triangle, arrow], we generate the following possible transformations: [(square -> triangle, circle -> arrow), (square -> arrow, circle -> triangle). It gets a little complicated when the number of objects changes, and that is described further in the implementation section.

Next we use problem reduction, focussing only the individual object transformations. An object is transformed to another using means-ends analysis. We change each of the attributes, and attributes can be added or removed.

Lastly, we need to do the test. Now we have one possible transformation between A and B, and another possible transformation between C and a guess for #. We test if these 2 transformations are equivalent. We do this by comparing the individual attribute transformations for each of the object maps. For this, we need to try out every possible combination of mapping an object from A to an object from C. This is illustrated best by looking at the example from the lectures:



Here we can see that we need to check if we're mapping the bigger circle in A to the square in C or the triangle in C. These would affect the final answer.

For every possible combination of these objects, we try to see if we get a successful match, that is if the transformation(A,B) is equivalent to the transformation(C, guess). If yes, we declare that as the answer(more finetuning could be done, as described in the "Future Work" section)

Implementation

Here we give a brief summary of some of the implementation specifics, that shed light on how the problem solving actually occurs. Some of the examples of this performance are described in the Performance section.

One possible transformation between 2 figures is represented as a FigureTransformation object, which is a collection of ObjectTransformations. Each ObjectTransformation is a collection of AttributeTransformations.

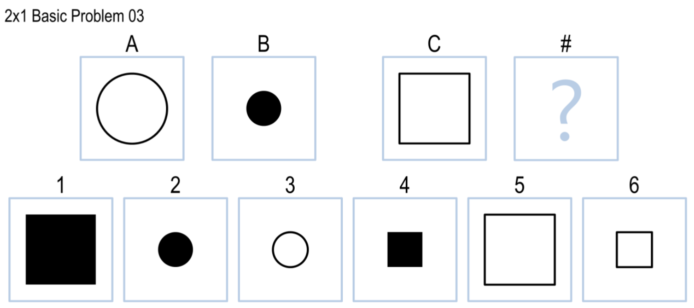
AttributeTransformation is where some of the meat of the implementation lies. A generic AttributeTransformation is simply the key with an initial and final value. Equivalent attribute transformations have equivalent initial and final values. We have more specific transformation objects for different kind of attributes:

1. FillTransformation - Fill can be either "yes", "no", or a list of areas that are filled. These are divided into 4 quadrants. We represent the initial and final values of the fill as a 4-bit binary number, that indicates which areas are filled.
2. OrientationTransformation - Orientations are specified using angle and vertical-flip. For orientation however, we need to check not only if the angles are exactly the same(taking into account full rotations), but also if the object can be vertically or horizontally flipped. We do this using individual Shape objects, that hold the properties for the object, which can be used to determine if the object is flipped.
3. LocationTransformation - Location is specified with relation to another object. On the surface, this is a generic AttributeTransformation, but the one thing we need to correct for is naming. When comparing 2 location transformations, we need to make sure that

they refer to the same object. Thus when we take a guess #, and look at it's "above" attribute, we are comparing to the same attribute value in B for the same object. This mapping is specific to the A-B, C-# and the A-C combination that we're currently trying.

Performance

Our agent successfully solves all 20 basic problems, and the 2 classmates problems. We will show it's performance on 3 problems it successfully solves, and then talk about some of the problems it is unable to solve.



The figures are represented as follows:

A

Z

shape:circle
size:large
fill:no

B

Z

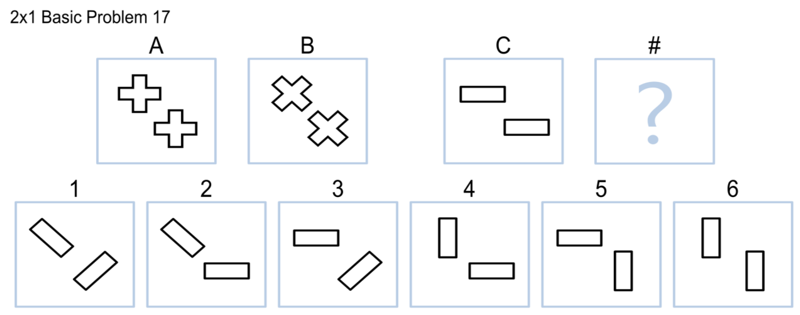
shape:circle
size:small
fill:yes

C

Z

shape:square
size:large
fill:no

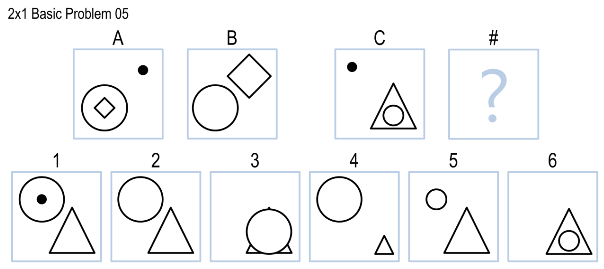
Here for A-B it has a single object to object map {'Z': 'Z'}, with the transformations { size: (large, small), fill: (0, 15) }. Here fill 15 means the entire object is filled.
For C, image 4 gives the exact same transformations, and thus it answers 4.

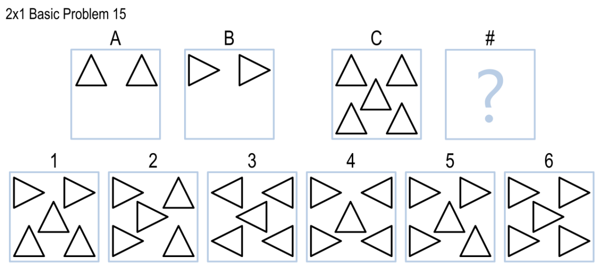


- A
  - Z
    - shape:plus
    - angle:0
  - Y
    - shape:plus
    - angle:0
    - above:Z
- B
  - Z
    - shape:plus
    - angle:45
  - Y
    - shape:plus
    - angle:45
    - above:Z
- C
  - Z
    - shape:rectangle
    - angle:0
  - Y
    - shape:rectangle
    - angle:0
    - above:Z
- 1
  - Z
    - shape:rectangle
    - angle:315
  - Y
    - shape:rectangle
    - angle:45
    - above:Z

Here, we can see that the key to solving the problem lies in realizing that one of the plusses is rotated clockwise, while the other is rotated anti-clockwise.
The agent solves this correctly, my mapping {'Y': 'Y', 'Z': 'Z'} in A-B and in C-#. For the orientation changes, the plus shape has an implicit knowledge that rotating 90 degrees leads it to right where it started. Thus our check converts the 315 degrees into an equivalent angle for the plus, which is 45 degrees.

Lastly, we look at 2 problems that highlight an important extension, that is handling different number of objects:



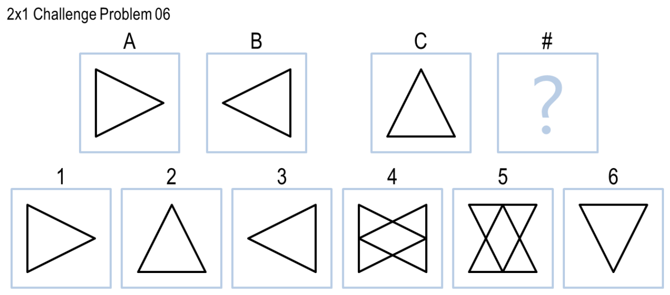


For the first case, we handle deletion by simply mapping any extra objects to None, representing deletion. This is simply represented as each attribute being deleted from that object.
For the second case, we do something a little more complicated. For the additional 3 objects, we try all possibilities of matching the existing 2 objects with them. This leads to a larger number of possible combinations, but leads to us successfully exploring all of them, and thus we realize that 6 is the correct answer.

Failure Cases

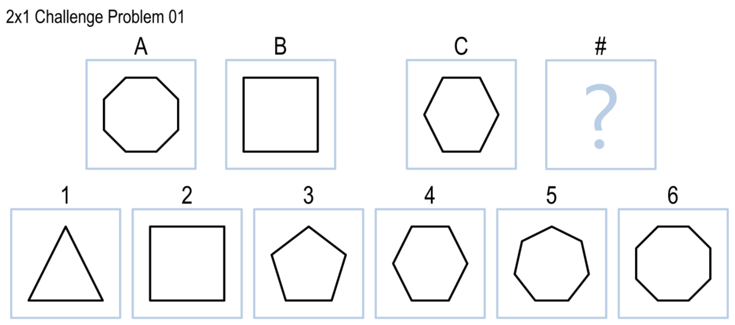
Multiple possible correct answers

One of the biggest cases where this agent would fail, is if there are multiple possible correct answers. Here correct is defined as transformations that could go from A to B and C to the guess that are equivalent. Here we intend to mimic human like behavior, and we would want to choose the answer that is more intuitive, done by weighting the possible responses. This is discussed in the future work section. An example of this is:

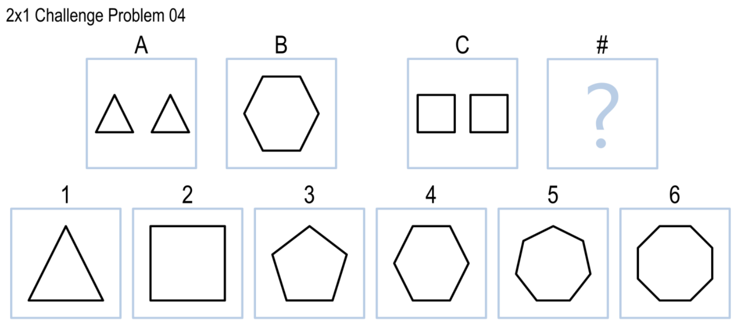


In this case, the agent can guess either 2 or 6 as the correct answer, 2 as A and B are reflected off a vertical axis, and 6 as A and B are horizontally reflected. Since humans are more likely to guess 2, we should weigh vertical reflection more than horizontal reflection, and this has not been accounted yet.

Challenge questions

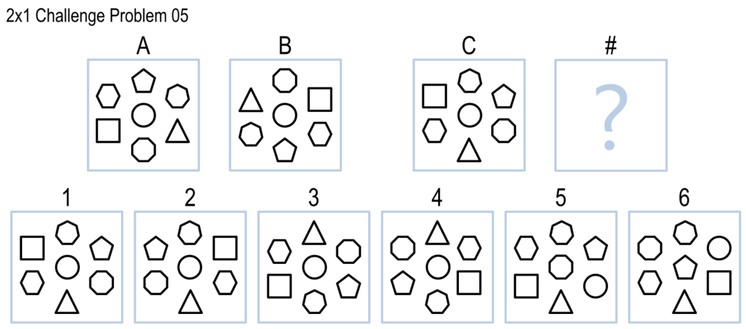


The agent is unable as of now to take into account cases such as the following, where the mapping is not exact, but the transformation is mathematical, that is the number of sides decreased by 1.



The agent also does not work for a case such as this, where the resultant object is an aggregation of the objects in A(in this case by number of sides), and in the case of another challenge problems, for that of angle.

Too Many objects



Cases such as these, where there are a large number of objects, result in an exponential blowup of the number of cases(7! number of possible mappings from A to B, that is 5040). This kind of blowup needs to be resolved with better weighing of possibilities, as explored in the next section.

Future Work

Given more time/resources, the following features would increase the agent's performance greatly.

Weighted Exploration of Options

The first feature I'd like to add is weighted exploration of options. After listing all possible transformations from A to B, we should have weights based on the individual object transformations(such as the object changing shape is a worse transformation that one changing fill). Thus we explore the least weight transformation first. This way we can find an answer for cases such as those with many objects.

Comparison of all Possible Answers

Following from above, once we have the least weight transformation for a single guess, we can compare that to the least weight transformations for any other guess. Picking the least weight transformation overall, we can resolve the issue where we have multiple possible correct answers.

Intelligent Guess

In the end, if we do not have a single "correct" transformation, we can guess from the multiple "almost correct" transformations. For this, we can have a diff function for the transformation that shows the value of the difference, in terms of the same weights as defined above. Thus we can use the lowest diff transformation as a guess for the correct answer.

Better Understanding of Many:1 Relations

Lastly, it would be nice to incorporate better understanding of many:1 relations. For now, deletion is the only case that is considered. Later on we can add more complicated cases of aggregation of some attribute.