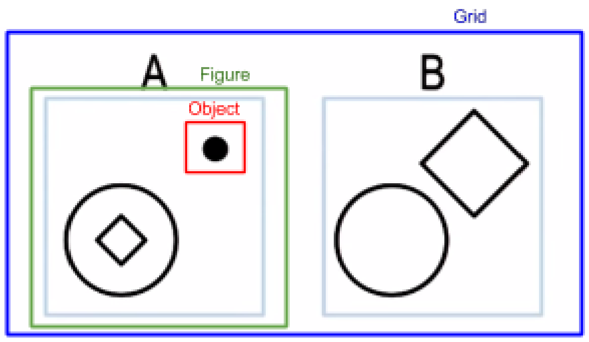
CS 7637: Project 2

Author: Swapnil Ralhan(sralhan3@gatech.edu)
February 22, 2015

Introduction

The goal of this project is to solve Raven's Progressive Matrices of size 2x1 and 2x2.
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 #.

For a 2x2 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 #, and the transformations between A and C are the same as the transformations between B and #.

General Idea

For the purposes of this project, I tried 2 different approaches.

Brute-force Approach

The first, is the approach used in Project 1. The general idea, is we map every possibility from Figure A to Figure B and represent that as a Transformation, then trying every possible mapping from objects in A to objects in B, we consider all transformations from C to every candidate transformation, and if the A-B transformation matches C-# transformation for any particular mapping of A-C, we consider the answer to be correct. The main issue faced with this approach, was the large blowup in complexity, which is further aggravated when we add addition and deletion of objects. Also, it does not take into account weights of transformations at all, so for 2 given answers that map correctly, it will just pick the first one. In an attempt to combat this, I tried a second "weight-first" approach, explained below. However, it should be noted that a misstep in the algorithm's correctness caused major problems, which were not detected till it was too late, and thus the algorithm does not perform too well(solving only about 50% of the problems). Details are shown below, and we talk more about combining the weight based and brute force approach in later sections.

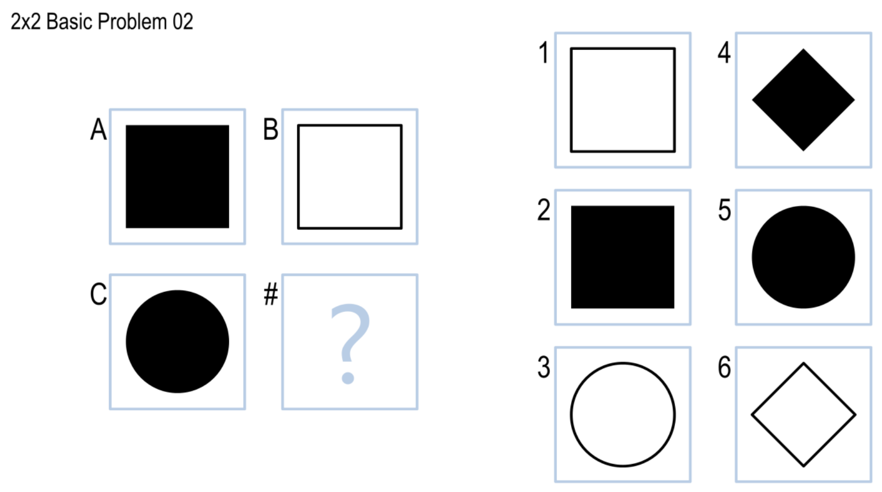
Weight-first approach

The general idea of this solution is as follows:

1. Generate ObjectTransformations. An object transformation is the series of transformations of individual attributes(AttributeTransformations), that are necessary to convert one object to another:
    a. For every object in A, generate an ObjectTransformation to every object in B.
    b. For every object in A, generate an ObjectTransformation to every object in C.
    c. For every object in B, generate an ObjectTransformation to every object in figures 1 through 6.
    d. For every object in C, generate an ObjectTransformation to every object in figures 1 through 6.
2. Save ObjectTransformations by weight. Each ObjectTransformation has a certain calculated weight, and equivalent ObjectTransformations should have the same weight(**this turned out to be the fatal flaw in the algorithm**)
3. For A to B, we store the ObjectTransformations by source object in a hashmap. We combine ObjectTransformations into a FigureTransformation(where every source object is mapped to something). These FigureTransformations are stored in a PriorityQueue.
4. Look at the FigureTransformation with the lightest weight. For each candidate solution, for each object transformation, find all matching possible candidate object transformations(this is made faster due to the calculated weight).
5. If we have at least one matching object transformation for each object transformation in the A-B FigureTransformation, we are done(we can add more complicated checks, such as whether they all map to different objects etc).
6. (Optional, planned to be added, but not due to the above flaw) Repeat the same procedure for A-C Figure Transformation. This is to be used for an extra vote, and/or tie breaking, or in case the above approach does not give us a valid answer.

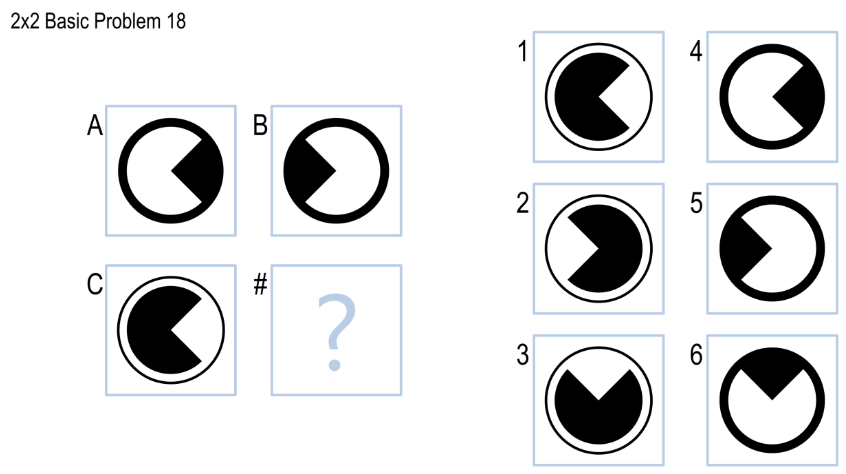
Correct Cases

The original approach works correctly for a number of cases, especially the 2x1s. Those are highlighted in the Project 1 doc. We study the performance of the second approach for the 2x2 problems:



2x2 Basic Problem 02
2x2
3
A
    A
        shape:square
        fill:yes
B
    B
        shape:square
        fill:no
C
    C
        shape:circle
        fill:yes
3
    F
        shape:circle
        fill:no

As we can see, mapping from A to B gives us a transformation of fill, from yes to no. Thus we look for the same transformation, which is found in figure 3.



2x2 Basic Problem 18
2x2
2
A
- A
  - shape:circle
  - fill:yes
- B
  - shape:Pac-Man
  - fill:no
  - angle:0
  - inside:A

B
- C
  - shape:circle
  - fill:yes
- D
  - shape:Pac-Man
  - fill:no
  - angle:180
  - inside:C

C
- E
  - shape:circle
  - fill:no
- F
  - shape:Pac-Man
  - fill:yes
  - angle:0
  - inside:E

2

- I
    - shape:circle
    - fill:no
- J
    - shape:Pac-Man
    - fill:yes
    - angle:180
    - inside:I

In this case, mapping from A to B, we can get the cheapest transformation being the one that maps object A to object C, and the one that maps object B to object D(we defined shape transformations to be a lot more expensive than regular transformations). We map the names of the attribute values for the location attributes(such as inside E), to the object names in A. Thus the answer clearly predicts mapping object E to object I and object F to object J, giving the correct answer.

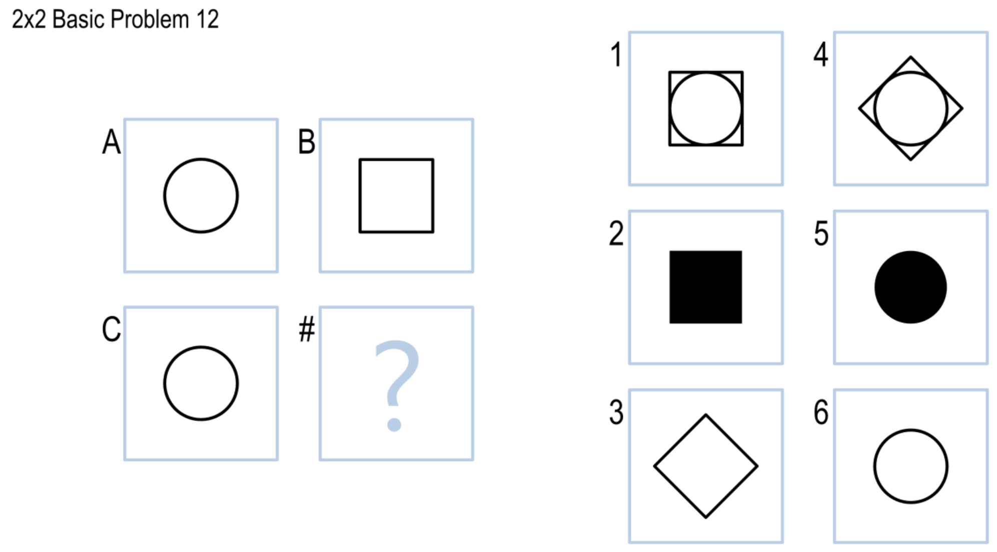
Failure Cases

Flaws in the Approach

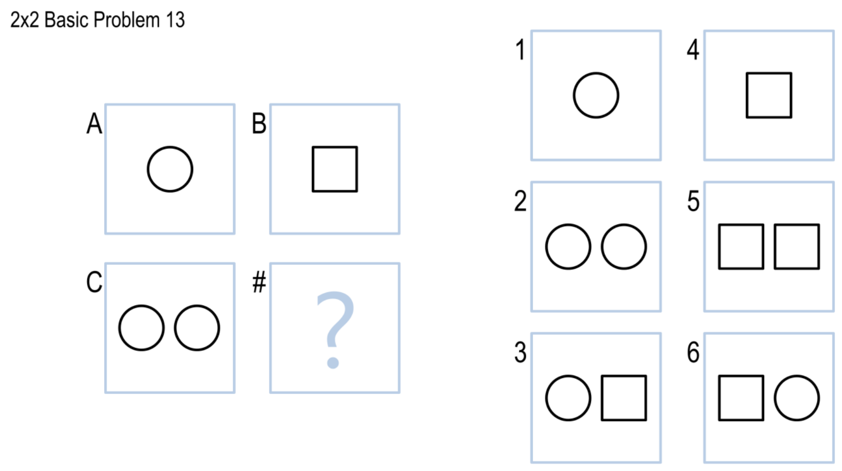
As highlighted above, there are a couple of flaws in the approach, that were discovered a little too late. This was that 2 equivalent transformations, may not have the same weight. Thus, when we explore the possible candidate object transformations, we miss the ones we expect to see.

The second flaw, is that it does not take into account addition of objects(since we look to ensure all objects are mapped from A to B, but not that every object in B has a source in A, even if it maps from None).

False Positives

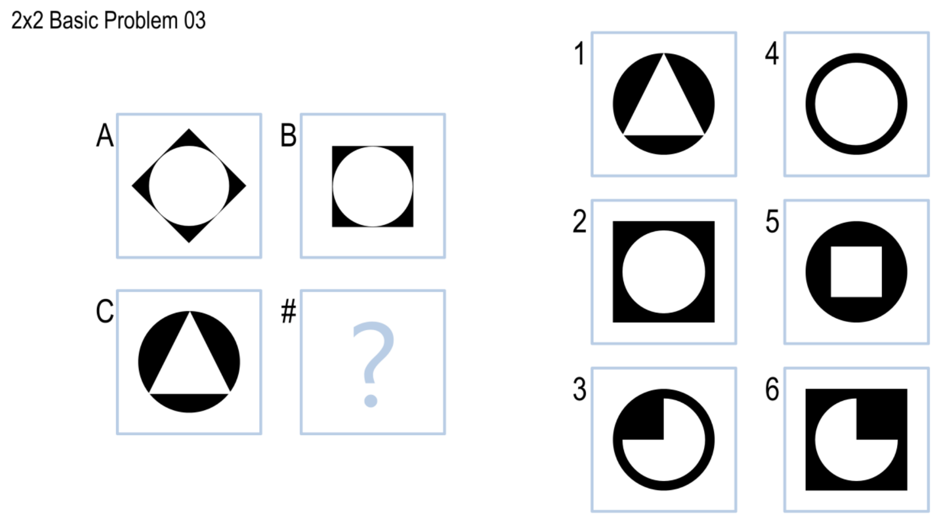


Here our approach gives the answer 1. This is due to the problem mentioned above, where we do not seek to map for every object in the goal figure. Thus in figure 1, the circle is ignored, and the square maps to the circle in C, giving us an incorrect answer.



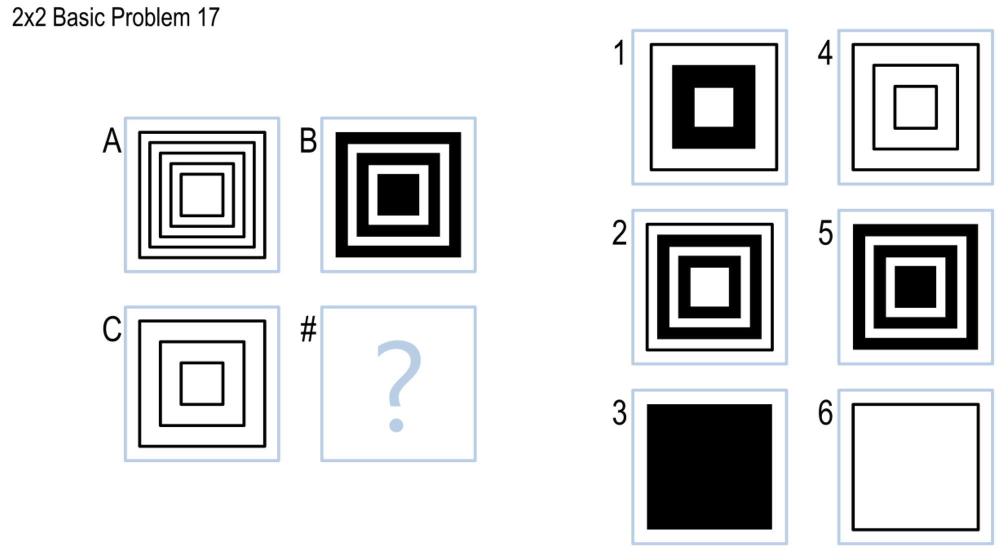
Similarly for the above, we map to option 3, as the mapping from A to C is incomplete, and all we need to do is map one of the circles to a square, giving us the answer 3.

False Negatives



Here, the problem is unable to find a solution. The reason for that, is that the weight of a transformation of the square in A to the square in B = weight(angle transformation). However, the weight of the circle in C to the circle in 1 = 0. Thus we can see, that it would simply never be found, leading to a large source of errors.

Inability to deduce



This is a unique case, where the answer relies on the understanding that a square within another square, where both squares are filled, would appear as one big square. Since your agent has no such understanding, it is completely unable to get the correct answer for this case.

Efficiency

Among the 2 approaches, the 1st approach is notoriously inefficient. It takes about $O((n!)^3)$ time, where n is the maximum number of objects in any figure A/B/C. It does this by trying all permutations of objects in A to objects in B, combining that with all permutations of objects in A to objects in C, and them with all permutations of objects in C to objects in #. This equals to 8 possibilities for 2 objects, 13824 possibilities for 4 objects, and 1.7 million possibilities for 5 objects(the largest practical size). For 6/7 objects, this approach is completely unfeasible.

The 2nd approach, gives us larger scope, working till about 8 objects. The time complexity is $O(n^{n+1}log(n))$ .This is achieved by generating all object transformations and mapping to all possible object transformations, and then putting them in a priority queue($n^n log(n^n)$). For 7 objects, we have a modest 5.7 million possibilities, and for 8 objects we get 134 million possibilities, which are still achievable in about a minute on a computer. It does this, by pre-caching all object transformations, and then mixing and matching all of them before hand, putting them all in a priority queue.

Future Improvements

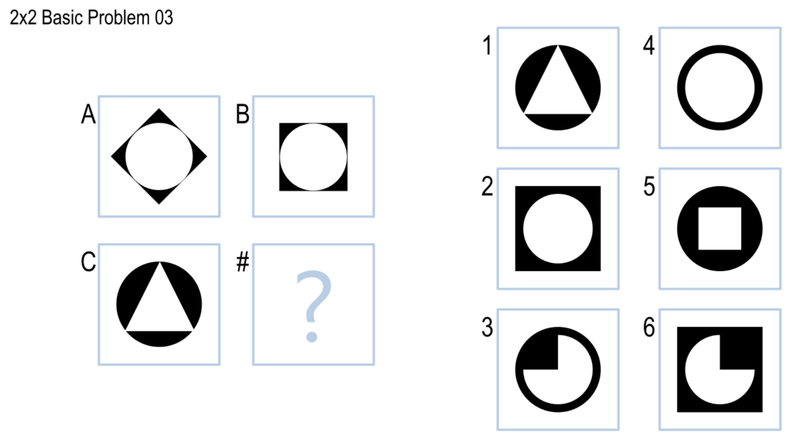
We need to focus on 2 different aspects in future improvements, correctness and efficiency. To get correctness, we take aspects from the first approach, where the performance is a lot better. We take that approach, and make 3 improvements -

Correctness - Better Object Name Mapping

The object name mapping is not entirely correct(names for attributes related to location should match each other). This needs to happen when we're comparing 2 objects to each other, they need to be the same when located next to 2 objects(which need to be mapped to each other in the figure transformation). That is, if we are comparing object B to object E, and B is left-of: A and E is left-of:F, then A needs to map to F in the figure transformation. This map needs to be propagated correctly.

Efficiency - Heuristic Based Elimination

This relates especially to human cognition.



When solving a problem such as the above, most people would immediately eliminate figures 3,4 and 6. This heuristic based elimination(or at least bumping these figures to the back of a mental "queue"), is crucial for efficient problem solving.

Efficiency - Caching

Our existing approach generates a lot of object transformations again and again. We can avoid this by caching object transformations once, and even generating equivalence relations and caching them(it can also be a partial equivalence, pending correctness checking

Correctness - Handling addition/deletion

Right now, the handling of addition and deletion of objects is unreliable and frequently incorrect. I would like to change the way the options are generated and modeled to take account for cases(such as the failure cases highlighted above), in a better manner.

Relation to Human Cognition

The approach modeled above takes inspiration from a number of ideas from human cognition. We highlight some of the similarities and differences to our approach as compared to human cognition.

Similarities

Weights of Transformation

The approach takes into account the fact that some transformations are more “expensive” than others, trying to map an innate sense of cognitive “power” taken to achieve that transformation. For instance, going from σ to ● is easier than going from σ to □. Similarly an “angle transformation” of 90 degrees is more expensive than a reflection. One approach that can be taken in the future is to look at the entropy of the transformation, and transformations that require more bits to represent are more expensive.

Independence of Object Transformations

One of the easy assumptions made by humans, is that transforming from one object to another is independent of another object being transformed(we look at the flip side of this in the differences). The notion of independence is used in problems where the transformations of different objects are different from one another, that is while one has it’s fill changed, the other is made bigger. We use the same notion, where we map each object to another and compute the transformation, more or less independently of other objects(the only exception being transformations to do with location)

Leveraging Knowledge

We leverage knowledge of things such as shapes, such as the angle that it needs to be rotated to look the same, or the number of sides it has. This knowledge is used by humans when coming to the correct answer.

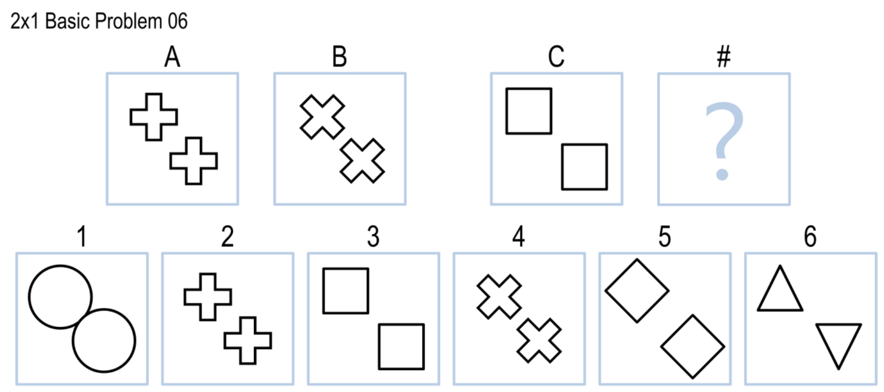
Trying Simple Transformations

Humans try simple transformations before harder ones, and we try to take the same approach in our method of trying less weight transformations before heavier weight ones.

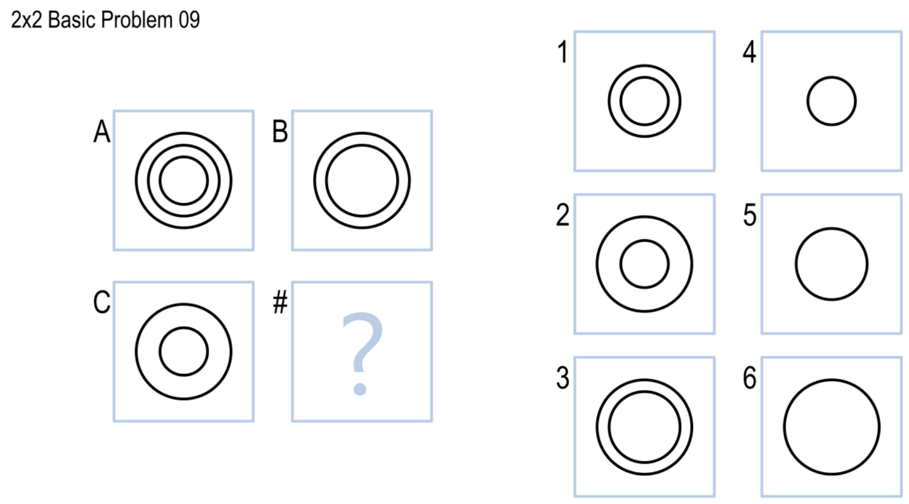
Differences

Inter-dependence of Object Transformations

As a flip side to the case mentioned above, sometimes humans can easily reach an answer by looking at 2 figures being transformed together. Simple examples of that is something like:

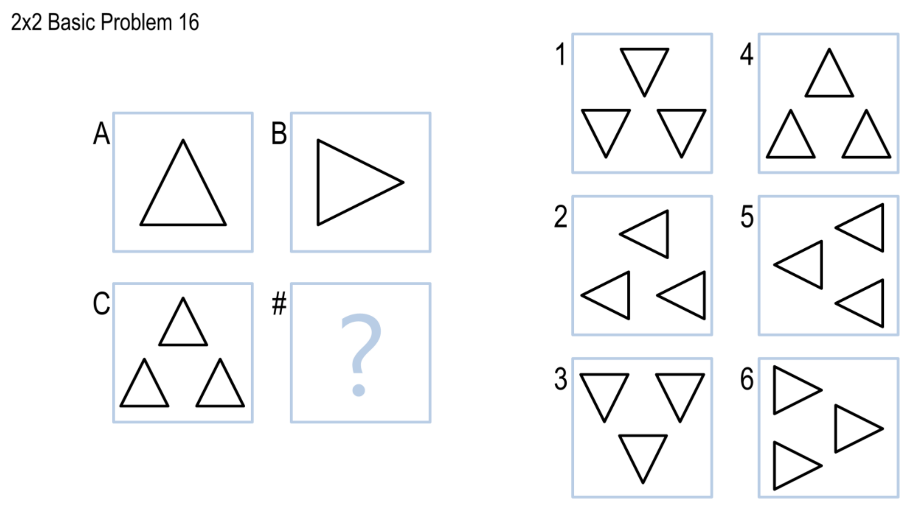


Here we simply see that all objects have the same rotation applied, and can easily deduce the answer. No such concept exists in our agent.
Another side of this is when we have something like deletion, where the deletion of an object modifies the “inside” or “overlaps” attributes of other objects. Viewing the objects independently, it’s easy to miss out on this relation.

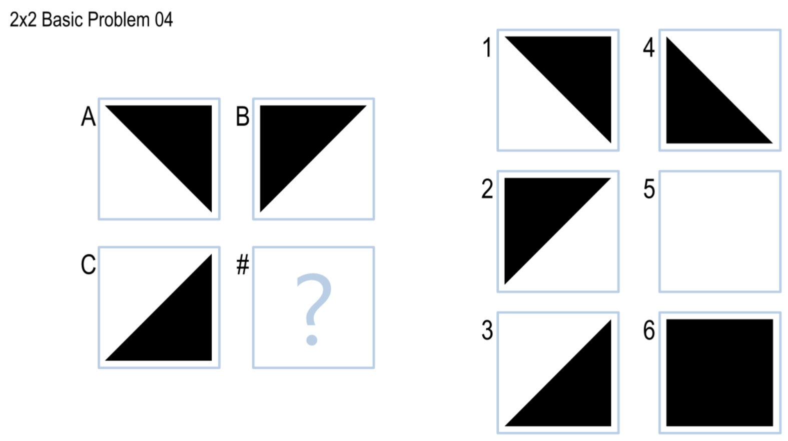


Holistic Approach to Figure Transformations

Humans often reach an answer extremely fast, by thinking of a figure being transformed as a whole. This approach was not viable right now, as we are given the details of each object independently. However if we look at the figure as a whole, we can reach conclusions about problems such as these extremely fast



Another example of this is when humans exploit "symmetry" of the figures when played together to get to a solution, in problems such as:



These options will be explored when we try a visual approach.

Heuristic Based Elimination

As mentioned above, humans employ a pseudo-heuristic based approach, where they eliminate the "obvious" options. This has not been explored by us yet.

Hypothesise and Apply

Another approach that humans use that we don't, is applying hypothesised transformations to C, and seeing if the applied image is present in the options. In some cases it works extremely well, and we don't have any concept of "applying" a transformation. However this approach should be explored, as it can solve the weight map problem that led to issues in our approach.

Special Knowledge

As shown in the earlier section about "Inability to deduce", there is knowledge such as filled objects being inside each other being not visible, which is difficult to represent in our agent, and can lead to problems that are very difficult to solve for agents, but easy for humans.