CS7637: Project 1 Reflection

nkarnati3@gatech.edu

The purpose of the project is to develop a human-level, human-like agent to illustrate the goal of Knowledge-Based AI. Raven's Progressive Matrices are chosen and in particular 2×2 analogy problems are to be solved.

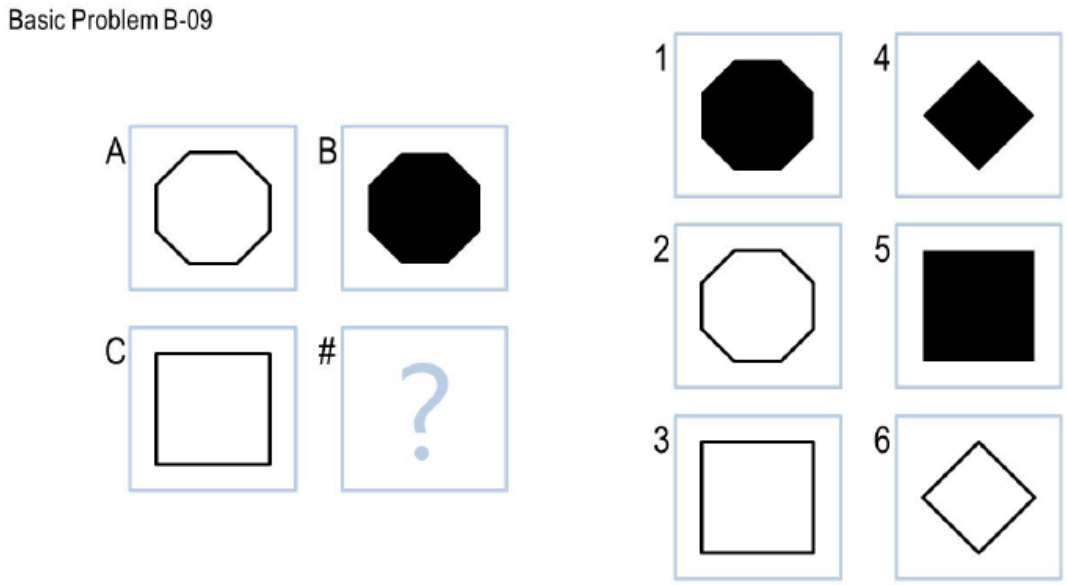


Fig 1 2×2 problem (Ashok et al., 2016)

Given a 2×2 problem(Ex: Fig1) with figures A, B and C and options with figures from 1 to 6 with both visual and verbal representations, need to produce an answer from the options to fill the blank figure #. Or return a negative number to skip.

Initial design was to identify shapes and use semantic net representation and generate and test problem solving strategy. First, verbal approach was used. After solving few cases, considering the shortcomings of the problem structure with verbal attributes when compared with the real world problem, only visual approach was chosen. While trying to implement the same, many questions arose on how to implement detection of shapes. Many approaches using pillow and numpy didn't result in good accuracy.

After careful examination on generic properties of all the problems, found a general property of number of dark pixels relationship between figures which is used to implement the agent using visual approach.

Design of the agent

The agent evaluates measure of each option to be the answer using two mappings.

Mapping 1 (A->B): By comparing the A to B transformation with C to option

Mapping 2 (A->C): By comparing the A to C transformation with B to option

The agent is designed using two main measures.

Measure 1: Percentage change in "the ratio of number of dark pixels of C->Option" to "the ratio of number of dark pixels of A->B"

Measure 2: Percentage change in "the intersection of dark pixels of C and Option, out of total number of dark pixels in both C and Option" to "the intersection of dark pixels of A and B, out of total number of dark pixels in both A and B"

This can be explained by using Fig 2.

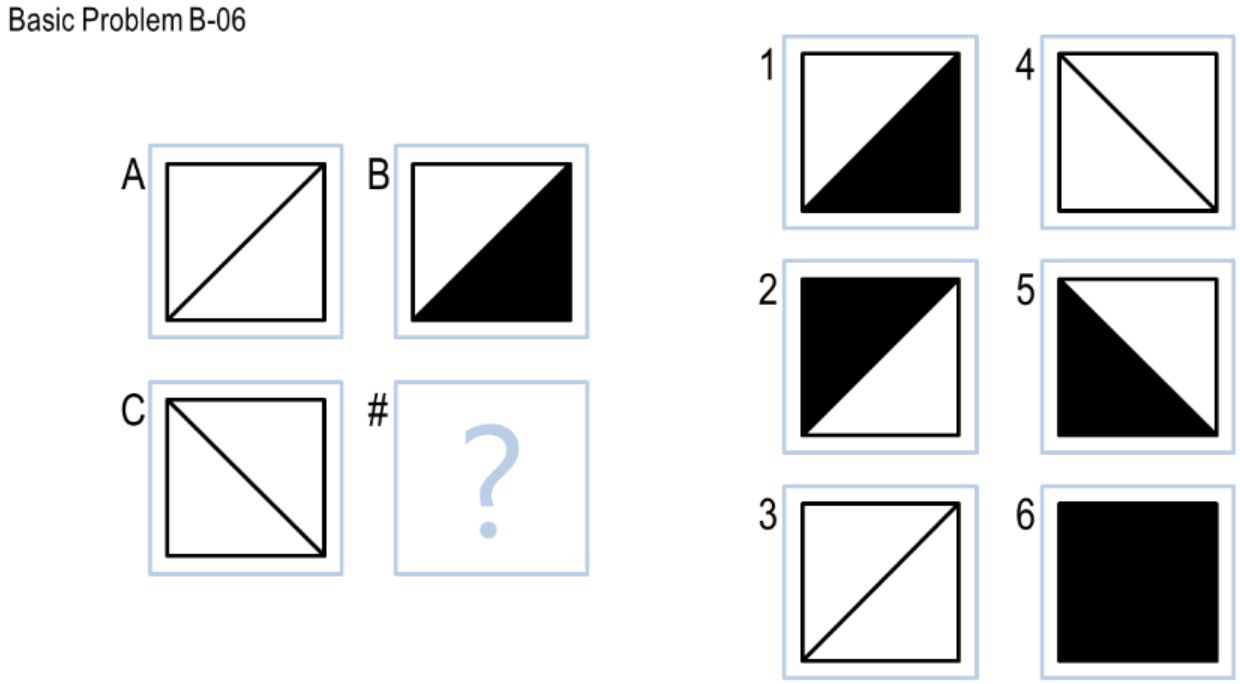
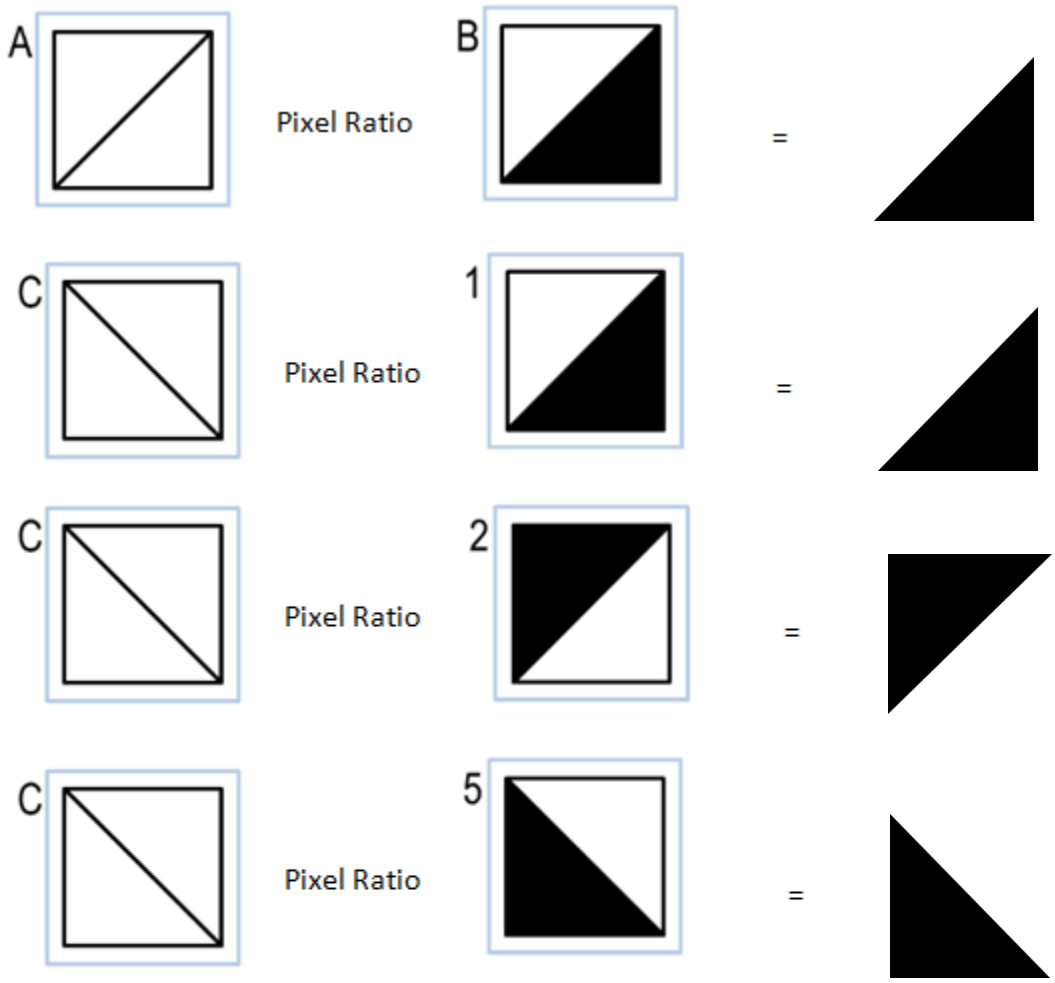


Fig 2 (Ashok et al., 2016)

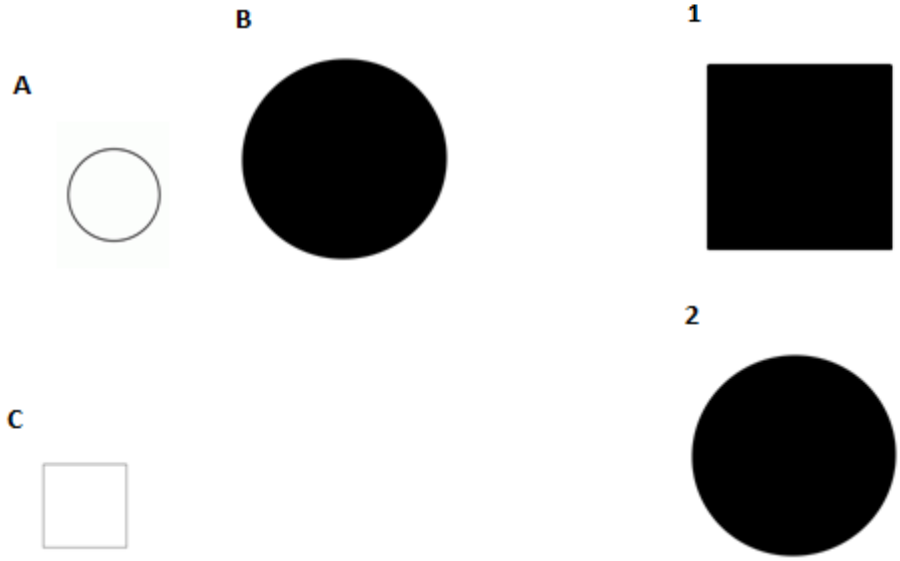
Using human cognition, we can identify that in A->B, half fill has occurred and on applying this to C we get option 5 as answer. This corresponds to agent's view of percentage change in the number of dark pixels in A and B and applying same to the C and options 1,2 and 5 which are half filled.



Options 3, 4 and 6 would give unfilled, unfilled and completely filled images leading them to elimination.

But this property alone cannot determine the answer, as agent cannot differentiate between the diagonal lines. Therefore, we have another measure to match the diagonal and the half fill portion with C.

Initial model was to calculate the number of dark pixels in each figure and to find the percentage change in A->B, A->C and apply it to C and B respectively. But that is evaluated to be wrong after examining few problems. Consider the below example:



A: 9px B: 900px 1: 1000px

C: 10px 2: 900px

C->1: (1000-10)-(900-9)/(900-9)= 0.1111

C->2: (900-10)-(900-9)/(900-9)= |-0.0011|

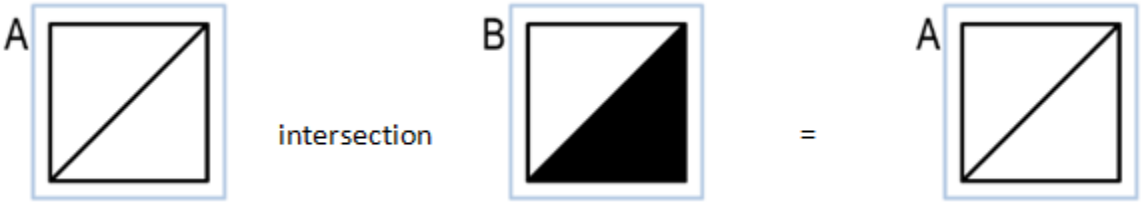
Agent chooses option 2 as the answer. If we change the measure to consider ratio of dark pixels,

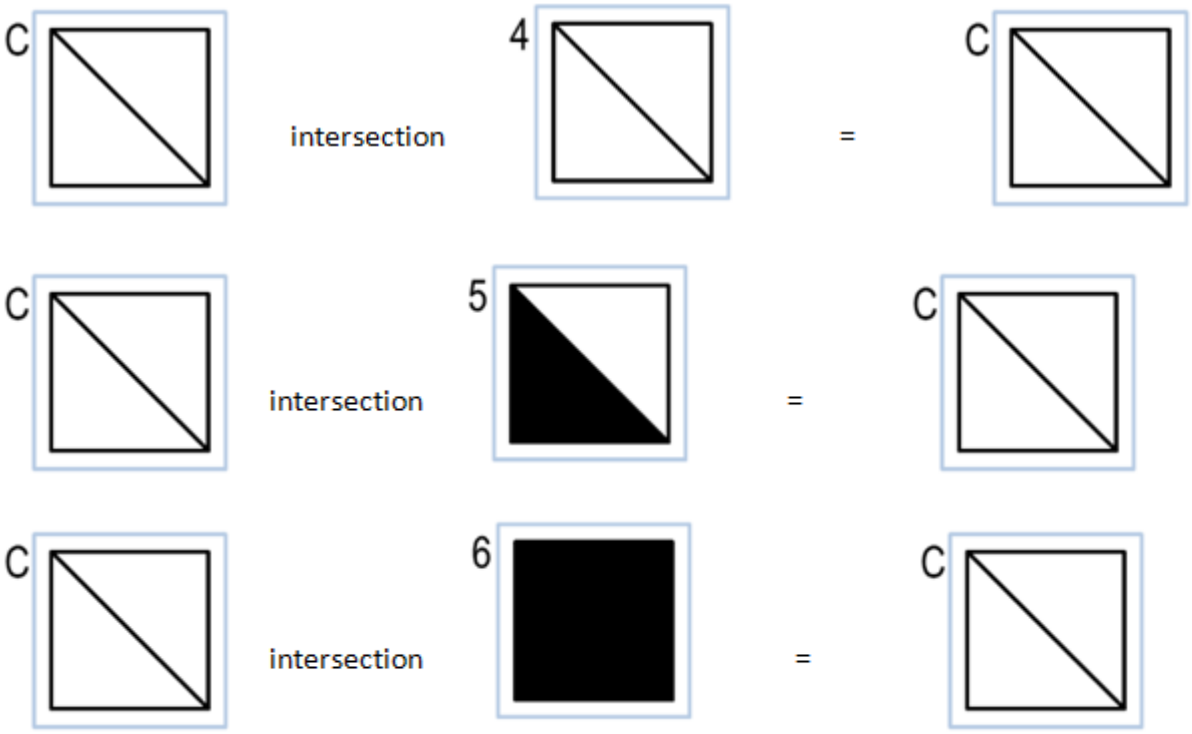
C->1: (1000/10) - (900/9)/(900/9)= 0

C->2: (900/10)-(900/9)/(900/9)= |- 0.1|

Therefore, percentage change in number of times of dark pixels in B to A with number of times of dark pixels in Option to C and also for A->C considers the proportional change in the shape.

Similarly, when we find the intersection of A and B, we get figure A and applying the logic to C and option which can give C as the intersection, we get option 4, 5 and 6.





1, 2 and 3 would not result in C and are eliminated.

By using these two measures, we get the result as option 5 which is correct.

Initially direct percentage change in the intersection measure was used. Discovered it to be an incorrect measure after examining few problems. This is because we need to know how many dark pixels intersect based on the total number of dark pixels in both images. Because, image A can differ from B and C in dark pixels and so the mapping from B->Option, C->Option also differs.

Therefore, we need to compare the intersection pixels in A and B with respect to total number of dark pixels in A and B with intersection pixels in C and Option with respect to total number of dark pixels in C and Option but not the exact change in the number of intersection pixels in A->B with C->Option.

In order to pick the answer using two measures, weights are assigned to every option for each measure based on their alignment with mappings. By using generate and test strategy, each option is evaluated for its plausibility to be the answer by their weights.

As the agent implements the human thought process, it falls under the category of agent that thinks like humans thus falling under Knowledge-Based AI as shown in Fig 3 (Ashok et al., 2016).

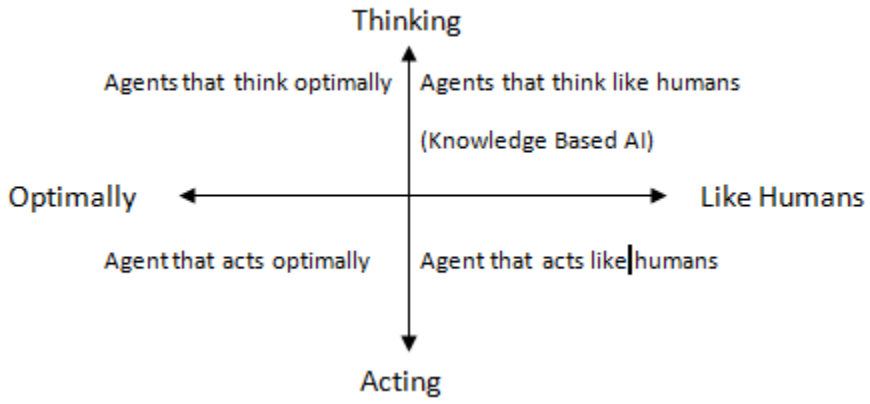


Fig 3 (Ashok et al., 2016)

Knowledge is represented using modification of semantic net as shown in Fig 4.

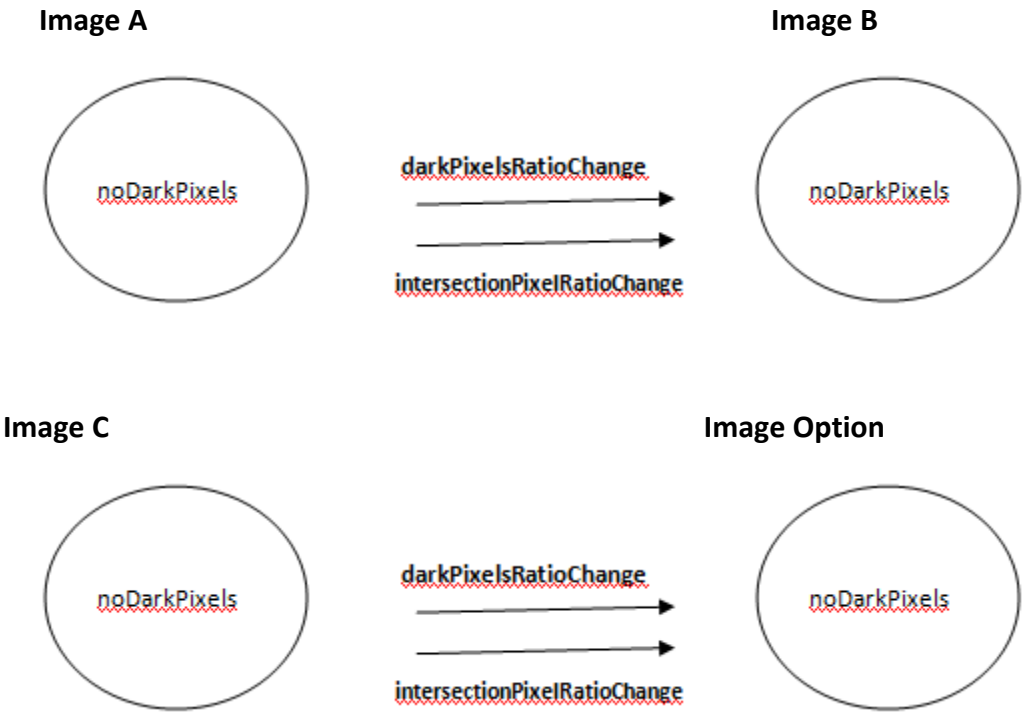


Fig 4

This complies with characteristics of good representation as follows.

1. Made relationship of changes explicit

2. Doesn't store unnecessary details

3. Concise, fast and computable

Generate and Test problem solving strategy is used by picking each option, generating the knowledge and testing the heuristics to check if that is a viable answer.

Overview of the implementation

Step 1: Convert images A, B and C to grayscale using pillow

```
figureA = problem.figures['A']
figureImageA = Image.open(figureA.visualFilename).convert('L')
```

Step 2: Number of dark pixels in the figures are counted as dark if its value<128, using numpy.

noDarkPixelsA = no of dark pixels in figure A

noDarkPixelsB = no of dark pixels in figure B

noDarkPixelsC = no of dark pixels in figure C

Step 3: Calculate the dark pixel ratios of the mappings

darkPixelRatioAB = noDarkPixelsB/noDarkPixelsA

darkPixelRatioAC = noDarkPixelsC/noDarkPixelsA

Step 4: Calculate the intersection of pixels in both mappings. Intersection is nothing but the existence of dark pixel at the exact same position in both the figures.

intersectionAB = no of dark pixels at same location (x, y) in FigureA, B

intersectionAC = no of dark pixels at same location (x, y) in FigureA, C

Step 5: Calculate the intersection ratios

IntersectionRatioAB = intersectionAB/(noDarkPixelsA + noDarkPixelsB)

IntersectionRatioAC = intersectionAC/(noDarkPixelsA + noDarkPixelsC)

Step 6: For each option figure i, do the following until Step 12. Use darkPixelRatioCOption, intersectionPixelRatioCOption lists for A->B and darkPixelRatioBOption, intersectionPixelRatioBOption lists for A->C

Step 7: Convert to gray scale and calculate dark pixels number

```
noDarkPixelsOption = no of dark pixels in the present option figure
```

Step 8: Calculate dark pixel ratio for each mapping

```
darkPixelRatioCOption.append(noDarkPixelsOption/noDarkPixelsC)
darkPixelRatioBOption.append(noDarkPixelsOption/noDarkPixelsB)
```

Step 9: Calculate intersection pixels of C, Option and B, Option for two mappings

```
intersectionCOption = no of dark pixels at same location (x, y) in Figures C, Option
intersectionBOption = no of dark pixels at same location (x, y) in Figures B, Option
```

Step 10: Calculate intersection pixel ratio for each mapping

```
intersectionPixelRatioCOption.append(intersectionCOption/(noDarkPixelsC+noDarkPixelsOption))
intersectionPixelRatioBOption.append(intersectionBOption/(noDarkPixelsB+noDarkPixelsOption))
```

Step 11: Calculate number of dark pixel ratio change

```
darkPixelsChangeAB.append(tuple([abs(darkPixelRatioCOption[i]-darkPixelRatioAB)/darkPixelRatioAB, i]))
darkPixelsChangeAC.append(tuple([abs(darkPixelRatioBOption[i]-darkPixelRatioAC)/darkPixelRatioAC, i]))
```

Step 12: Calculate intersection ratio change

```
intersectionChangeAB.append(tuple([abs(intersectionRatioCOption[i] - intersectionRatioAB)/intersectionRatioAB,i])
intersectionChangeAC.append(tuple([abs(intersectionRatioBOption[i] - intersectionRatioAC)/intersectionRatioAC,i])
```

// Two measures calculation done for each option

Step 13: Sort darkPixelsChangeAB, darkPixelsChangeAC, intersectionChangeAB intersectionChangeAC based on their change values.

```
darkPixelsChangeAB.sort()
darkPixelsChangeAC.sort()
intersectionChangeAB.sort()
intersectionChangeAC.sort()
```

Step 14: Normalize darkPixelChange values in both mappings

```
Find min, max values in darkPixelChangeAB+darkPixelChangeAC change values
Normalize each value x using x-min/max-min formula.
```

Step 15: Normalize intersectionChange values in both mappings

```
Find min, max values in intersectionChangeAB+intersectionChangeAC change values
Normalize each value x using x-min/max-min formula.
```

Step 16: Assign weights to each change value based on change value and position 'i' in sorted list

```
weightsPixelsAB = (1/(1+darkPixelChangeAB) )* 1/(1+i)
weightsIntersectionAB = (1/ (1+intersectionChangeAB))*1(1+i)
weightsPixelsAC = (1/(1+darkPixelChangeAC) )* 1/(1+i)
weightsIntersectionAC = (1/ (1+intersectionChangeAC))*1(1+i)
```

Step 17: Calculate sum of weights

```
sumWeightsAB =  weightsPixelsAB + weighsIntersectionAB
sumWeightsAC =  weightsPixelsAC + weighsIntersectionAC
```

Step 18: Return the option which has max sum among both mappings

```
Return max(sumWeightsAB, sumWeightsAC) Option
```

Agent makes the following mistakes.

Mistake 1:

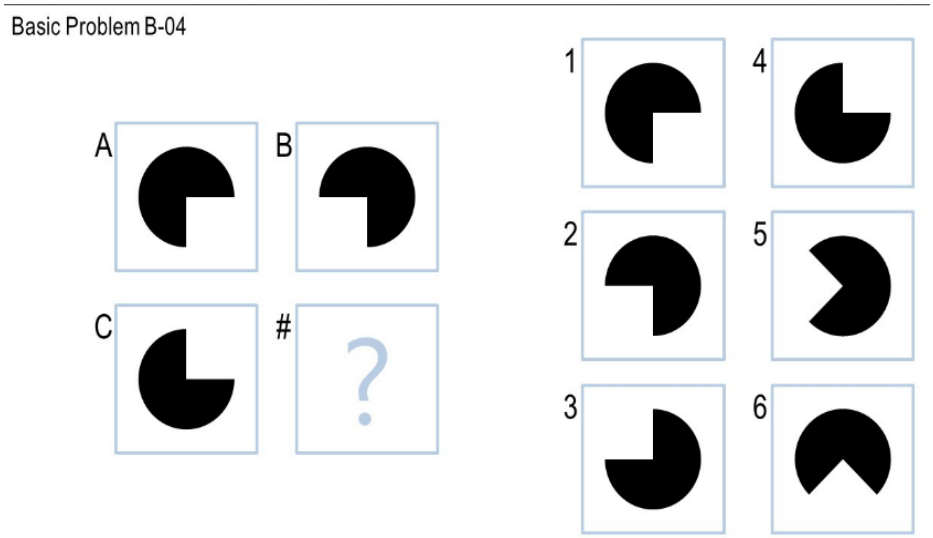


Fig 5 (Ashok et al., 2016)

Here, as we see as humans according to the dark pixel measure, all the options qualify.

According to the intersection measure, A->B would have a semicircle and radius line as the intersection. This can happen with options 1 and 3. But agent returns option 4.

Similar issue happens with B-07

Mistake 2:

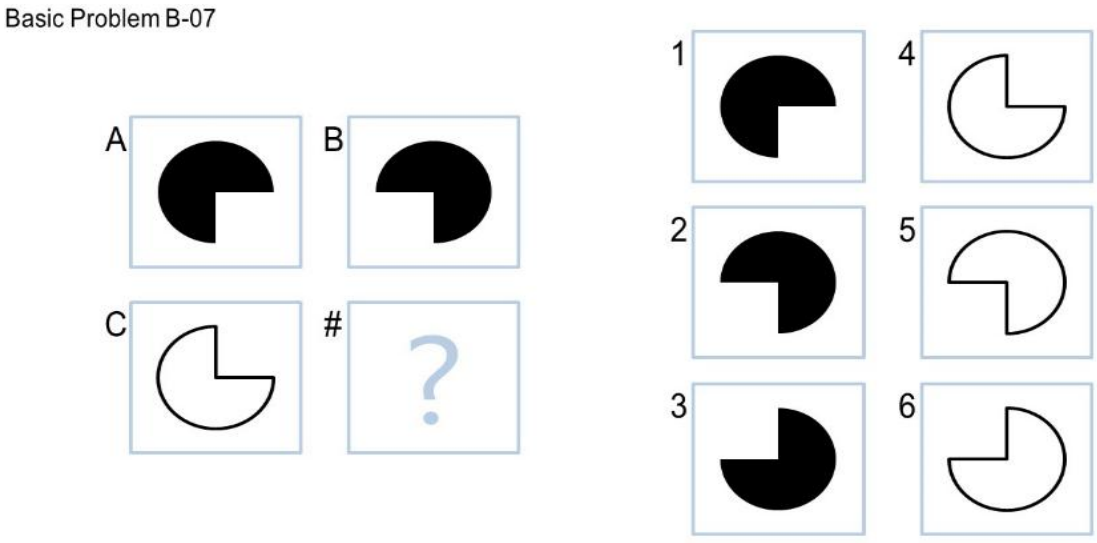


Fig 6 (Ashok et al., 2016)

Using both measures, we see option 5 or 6 as the potential answers. But agent returns 4.

This is due to applying the human solving strategy as the image representation need not be similar to how humans interpret it.

But as agent cannot decide between 1,3 in Fig 5 and 5,6 in Fig 6 as agent is not using the direct intuition of human solving strategy of rotation. This is due to the approach and can be resolved by using the same approach applied to the sub problems of the mapping.

Ex: Divide image A into four quadrants and analyze two measures in each of them and make comparisons. In A->B, top quadrants didn't change, in bottom exchange occurred. When this is applied to C, the bottom quadrants won't change and top quadrants are interchanged thus giving the answer 6.

Thus there is scope to improve the performance of the agent to add extra weights with different transformations like rotation, reflection, expansion, deletion etc.

Mistake 3:

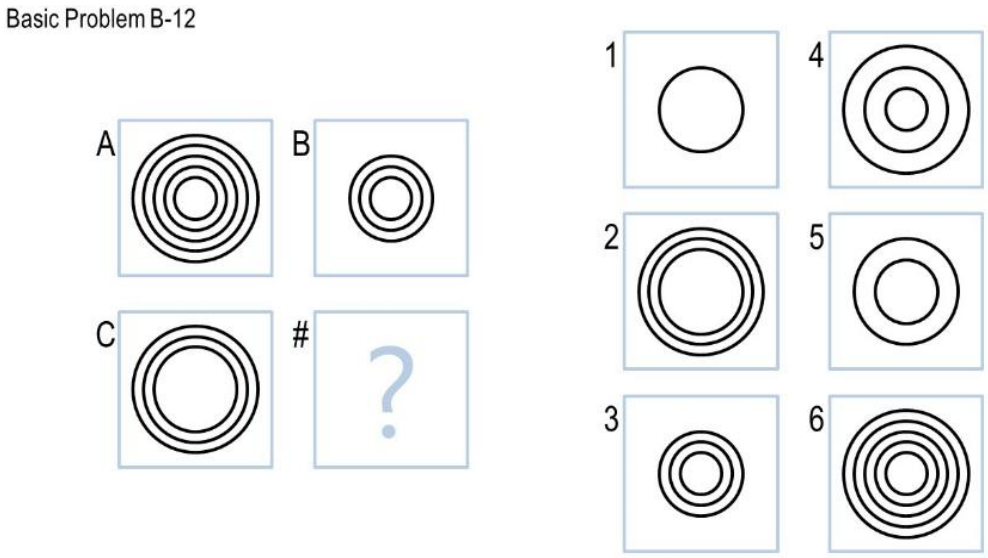


Fig 7

Out of five circles in A, outer two disappear. Whereas out of three circles in C also outer two disappear. This results in difference in pure amount of dark pixels. The agent returns option 3 considering the two measures.

This is due to the approach of agent towards the problem to apply the analogy using the appropriate matching techniques which cannot be pure difference in the values but the proportion. Thus agent lacks the human strategy here to just cut two shapes without considering the proportion of the images.

Agent's accuracy show in Table 1 demonstrates the agent's intelligence to achieve the goal of solving RPMs by 75% and varied problems by 62.5%. This explains generality of the design to solve unseen problems and its approximation to human like solving strategy.

Problems	**Correct - Accuracy**
Basic	9/12 = 75%
Ravens	9/12 = 75%
Test	7/12 = 58.34%
Challenge	5/12 = 41.67%
Overall Accuracy	30/48 = 62.5%

Table 1

The agent is highly efficient as it is not storing any figures or any verbal representation data but just the number of dark pixels thus improving the efficiency. And the calculations are pure mathematical. Use of pillow and numpy made it even faster. Execution time is 39.1605 on Bonnie which shows the efficiency.

Agent lacks the heuristic of confidence to decide on giving the answer or not. Therefore it gives answers for all questions and this deviates its process from that of a human. This gives scope to improve the agent in this aspect.

The agent developed agrees with the characteristics of the AI Agent as to have limited computing power, limited attention, computational logic being deductive, incomplete knowledge relative to the world (Ashok et al., 2016).

Thus, the agent agrees with the principles of CS7637 as follows (Ashok et al., 2016).

1. Represented and organized the knowledge into structures (dark pixels)to guide and support reasoning.

2. Learning was incremental as the design progress evolved with exposure to new problem but for now the agent cannot store the knowledge from one problem and use it on another.

3. It matches the methods of semantic networks, generate and test to the task of solving RPMs

4. It uses heuristics (Measure 1&2) to find solutions that are good enough

5. It uses the recurring patterns of change in number of dark pixels and intersection in the problems

References

Book

Ashok Goel, David Joyner. (2016). KBAI Ebook: Knowledge-based Artificial Intelligence. Retrieved on August 31st, 2017 from https://files.t-square.gatech.edu/access/content/group/gtc-b1b7-becd-51fd-a313-f821fa23709a/KBAI-Ebook/kbai_ebook.pdf